User Guide

Amazon CodeCatalyst

Amazon CodeCatalyst: User Guide

Table of Contents

What is Amazon CodeCatalyst?

Amazon CodeCatalyst is an integrated service for software development teams adopting continuous integration and deployment practices into their software development process. CodeCatalyst puts the tools you need all in one place. You can plan work, collaborate on code, and build, test, and deploy applications with continuous integration/continuous delivery (CI/CD) tools. You can also integrate AWS resources with your projects by connecting your AWS accounts to your CodeCatalyst space. By managing all of the stages and aspects of your application lifecycle in one tool, you can deliver software quickly and confidently.

In CodeCatalyst, you create a space to represent your company, department, or group, and then you create projects that contain the resources needed to support your development teams and tasks. CodeCatalyst resources are structured inside projects that live inside spaces. To help teams get started quickly, CodeCatalyst provides language- or tool-based project blueprints. When you create a project from a project blueprint, the project comes with resources such as a source repository with sample code, build scripts, deployment actions, virtual servers or serverless resources, and more.

What can I do with CodeCatalyst?

You and your development team can use CodeCatalyst to carry out each aspect of software development, from planning your work to deploying your applications. You can use CodeCatalyst to:

- **Iterate and collaborate on code** – Work collaboratively with your team on code with branches, merges, pull requests, and comments in your source code repositories. Create Dev Environments to work on code quickly without having to clone or set up connections to repositories.
- **Build, test, and deploy your application with workflows** – Configure workflows with build, test, and deploy actions to handle the continuous integration and delivery of your applications. You can either start workflows manually or configure them to start automatically based on events such as code pushes or creating or closing pull requests.
- **Prioritize your team's work with issue tracking** – Use issues to create backlogs and monitor the status of in-progress tasks with boards. Creating and maintaining a healthy backlog of items for your team to work on is an important part of developing software.
- **Set up monitoring and notifications** – Monitor team activity and resource status, and configure notifications to stay up to date with important changes.

How do I get started with CodeCatalyst?

If you don't have a space or you want to learn how to set up and manage a space, we recommend that you get started with the Amazon CodeCatalyst Administrator Guide.

If you're new to working in a project or a space, we recommend that you get started by:

- Reviewing the CodeCatalyst concepts
- Creating a space
- Creating your first project by following the steps in Tutorial: Creating a project with the Modern three-tier web application blueprint

Learn more about CodeCatalyst

You can learn more about the functionality of CodeCatalyst in this user guide, as well as the following resources:

- AWS DevOps Blog articles about Amazon CodeCatalyst
- The Amazon CodeCatalyst API Reference Guide
- The Amazon CodeCatalyst Action Development Kit Developer Guide
- CodeCatalyst FAQ
- Testimonials

CodeCatalyst concepts

Get familiar with the key concepts to help speed up your collaboration and application development in Amazon CodeCatalyst. These concepts include terms used in source control, continuous integration and continuous delivery (CI/CD), and modeling and configuring automated release processes.

For additional conceptual information, see the following topics:

- Source repository concepts
- Workflows concepts

Topics

AWS Builder ID spaces in CodeCatalyst

The space administrator invites users to CodeCatalyst by sending individual invitation emails from the members page. Users who are invited or sign up to CodeCatalyst create their own AWS Builder ID. The profile is managed in AWS Builder ID and displays as the user name and profile information in the user settings in CodeCatalyst.

Spaces that support identity federation in CodeCatalyst

Users who have been added to the SSO users and groups for the IAM Identity Center instance and are managed in the identity store and are invited to your space through IAM Identity Center. The **Space administrator** syncs the CodeCatalyst members page for the latest updates. Users sign in using the SSO sign-in portal as set up in the company IAM Identity Center instance. Spaces that support identity federation are connected to the identity store instance through the Identity Center application and its mapping to the identity store ID.

Projects

A *project* represents a collaborative effort in CodeCatalyst that supports development teams and tasks. After you have a project, you can add, update, or remove users and resources, customize your project dashboard, and monitor the progress of your team's work. You can have multiple projects within a space.

For more information about projects, see Organize work with projects in CodeCatalyst.

Blueprints

A *blueprint* is a project synthesizer that generates and extends application support files and dependencies for you, along with creating your CodeCatalyst project in the console. You choose a project type from a selection of blueprints in CodeCatalyst, view the README file, and preview the project repository and resources that will be generated. Your project is generated from the base configuration specified by the blueprint. You synthesize to the project blueprint periodically, which updates your project files, such as software dependencies, and regenerates resources. Projects use a tool called Projen to synthesize projects by syncing the latest project updates and generating support files. These files may include `package.json`, `Makefile`, `eslint`, and more based on your application type and language. Project blueprints can generate files supporting AWS resources

such as CDK constructs, AWS CloudFormation templates, and AWS Serverless Application Model templates.

For more information about project blueprints, see Creating a comprehensive project with CodeCatalyst blueprints.

Account connections

An *account connection* associates a CodeCatalyst space with your AWS account. After your account connection is set up, the AWS account is made available to the space. You can then add IAM roles to CodeCatalyst so that it can access resources in your AWS account. You can also use these roles for your CodeCatalyst workflow actions.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts.

VPC connections

A *VPC connection* is a CodeCatalyst resource which contains all of the configurations needed for your workflow to access a VPC. Space administrators can add their own VPC connections in the Amazon CodeCatalyst console on behalf of space members. By adding a VPC connection, space members can run workflow actions and create Dev Environments that adhere to network rules and can access resources in the associated VPC.

For more information about VPC connections, see Managing Amazon Virtual Private Clouds in the *CodeCatalyst Administrator Guide*.

AWS Builder ID

An AWS Builder ID is a personal identity you can use to sign up and sign in to CodeCatalyst and other participating applications. It is not the same as an AWS account. Your AWS Builder ID manages metadata such as user alias and email address. Your AWS Builder ID is a unique identity that supports users across all spaces in CodeCatalyst. For information about accessing your AWS Builder ID profile, see Updating your profile. To learn more about AWS Builder ID, see AWS Builder ID in the AWS General Reference.

For more information about signing up and signing in, see Set up and sign in to CodeCatalyst.

User profiles in CodeCatalyst

You access your CodeCatalyst user profile by choosing the profile option from the drop-down under your login initials on any page in CodeCatalyst. You can create personal access tokens (PATs) from your profile page, but you can only view or delete PATs using the AWS CLI. Your user name is the alias you chose when you signed up. You cannot change your user name. To view the profile page for another CodeCatalyst user, go to the **Members** tab for your project and choose the appropriate user.

You access your AWS Builder ID by viewing your CodeCatalyst profile and then choosing to go to AWS Builder ID. You will be redirected to your AWS Builder ID profile page. Your profile's full name, email address, and password are managed by your AWS Builder ID, and you can edit that information using the AWS Builder ID page. You entered this information when you signed up. When you are ready to set up MFA to use an authenticator application for signing in, you will use the AWS Builder ID page. For more information about viewing your AWS Builder ID profile, see Updating your profile.

For more information about signing up and signing in, see Set up and sign in to CodeCatalyst.

Source repositories

A *source repository* is where you securely store code and files for your project. It also stores the version history of your files. By default, a source repository is shared with the other users in your CodeCatalyst project. You can have more than one source repository for a project. You can create source repositories for projects in CodeCatalyst, or you can choose to link an existing source repository hosted by another service if that service is supported by an installed extension. For example, you can link a GitHub repository to a project after you install the **GitHub Repositories** extension. For more information, see Storing source code in repositories for a project in CodeCatalyst and Quickstart: Installing extensions, connecting providers, and linking resources in CodeCatalyst.

Source repositories are also where configuration information is stored for your CodeCatalyst project, such as the configuration file that defines the attributes and actions of your CI/CD workflow. If you create your project using a blueprint, a source repository will be created with project configuration information stored inside it. If you create an empty project, you must create a source repository before you can create resources that require configuration information, such as workflows.

For more concepts that can help you work with source repositories and source control, see Source repository concepts.

Commits

A *commit* is a change to a file or set of files. In the Amazon CodeCatalyst console, a commit saves your changes and pushes them to a source repository. The commit includes information about the change, including the identity of the user who made the change, the time and date of the change, the commit title, and any message included about the change. For more information, see Understanding changes in source code with commits in Amazon CodeCatalyst.

In the context of a source repository in CodeCatalyst, commits are snapshots of the changes to the contents of your repository. Every time a user commits and pushes a change, CodeCatalyst saves information that includes who committed the change, the date and time of the commit, and the changes made as part of the commit. You can also add Git tags to commits to help identify specific commits.

For more information about commits, see Understanding changes in source code with commits in Amazon CodeCatalyst.

Dev Environments

A *Dev Environment* is a cloud-based development environment that you can use in CodeCatalyst to quickly work on the code stored in the source repositories of your project. The project tools and application libraries included in your Dev Environment are defined by a devfile in the source repository of your project. If you do not have a devfile in your source repository, a default devfile will be applied automatically. The default devfile includes tools for the most frequently used programming languages and frameworks. By default, a Dev Environment is configured to have a 2-core processor, 4 GB of RAM, and 16 GiB of persistent storage.

Workflows

A *workflow* is an automated procedure that describes how to build, test, and deploy your code as part of a continuous integration and continuous delivery (CI/CD) system. A workflow defines a series of steps, or *actions*, to take during a workflow run. A workflow also defines the events, or *triggers*, that cause the workflow to start. To set up a workflow, you create a *workflow definition file* using the CodeCatalyst console's visual or YAML editor.

> **Tip**
>
> For a quick look at how you might use workflows in a project, create a project with a blueprint. Each blueprint deploys a functioning workflow that you can review, run, and experiment with.

For more information about workflows, see Build, test, and deploy with workflows in CodeCatalyst.

Actions

An *action* is the main building block of a workflow, and defines a logical unit of work, or task, to perform during a workflow run. Typically, a workflow includes multiple actions that run sequentially or in parallel depending on how you've configured them.

For more information about actions, see Configuring the actions that a workflow performs.

Issues

An *issue* is a record that tracks the work related to your project. You can create an issue for a feature, a task, a bug, or any other body of work related to your project. If you're using agile development, an issue can also describe an epic or user story.

For more information about issues, see Track and organize work with issues in CodeCatalyst.

Personal access tokens (PATs)

A *personal access token* (PAT) is similar to a password. It is associated with your user identity for use across all spaces and projects in CodeCatalyst. You use PATs to access CodeCatalyst resources that include integrated development environments (IDEs) and Git-based source repositories. PATs represent you in CodeCatalyst and you can manage them in your user settings. A user can have more than one PAT. Personal access tokens only display once. As a best practice, be sure to store them securely on your local computer. By default, PATs expire after one year.

For more information about PATs, see Grant users repository access with personal access tokens.

Personal connections

A *personal connection* is an authorization between your CodeCatalyst identity and your external source provider, such as GitHub. You use personal connections to allow a CodeCatalyst user to add third-party source repositories. For example, you can connect a GitHub repository to a CodeCatalyst space. An installed connector application is installed in the GitHub account for use with repositories designated by the account owner. You can create one personal connection for one user identity (CodeCatalyst alias) across all spaces for a specific provider type, such as GitHub. Personal connections are either associated with your AWS Builder ID or your SSO user.

For more information, see Accessing GitHub resources with personal connections.

Roles

A *role* defines a user's access to the resources for a project or a space and which actions that user can take. You choose the role for a user when you invite them to a project. There are space-level roles and project-level roles in CodeCatalyst. A user with an administrative role at the correct level can change assigned roles. For example, a user with the **Project administrator** role for a project has full control over that project and can change the roles of users in that project. For information about which roles are available and which permissions each role has, see Granting access with user roles.

For more information about roles, see Granting access with user roles.

Set up and sign in to CodeCatalyst

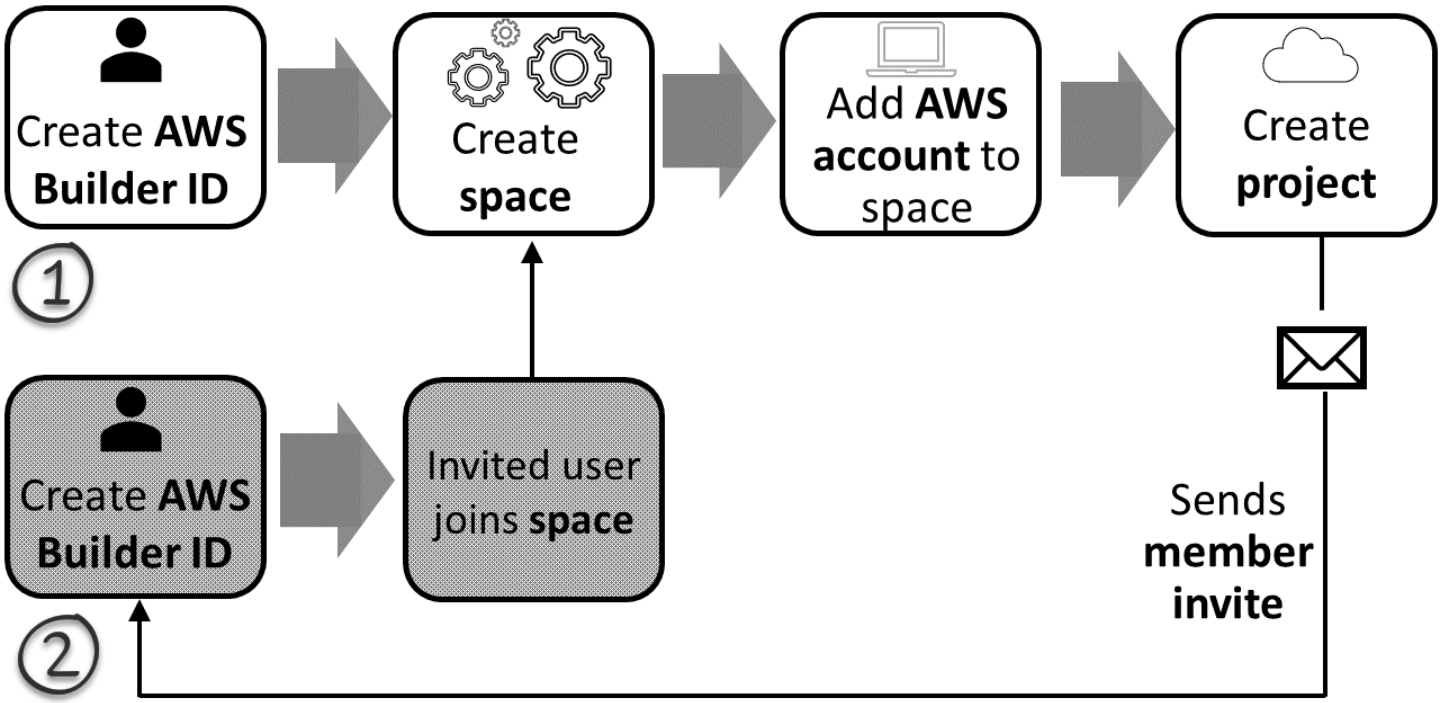
There are two types of space that you can set up in CodeCatalyst: spaces that support AWS Builder ID users, and creating a space that supports identity federation, where SSO users and groups are managed in IAM Identity Center. Users in an AWS Builder ID space sign in to CodeCatalyst with their AWS Builder ID, and users in a space set up for identity federation sign in to CodeCatalyst using the SSO portal for the company associated with the space.

The steps to set up and administer a AWS Builder ID space are provided in this guide. To work with a CodeCatalyst AWS Builder ID space, you will set up CodeCatalyst using the user settings and AWS Builder ID that you use to sign in to CodeCatalyst.

The steps to set up and administer a space that supports identity federation are provided in the *CodeCatalyst Administrator Guide*. To work with spaces that are set up for identity federation, see Setup and administration for CodeCatalyst spaces in the Amazon CodeCatalyst Administrator Guide.

This section provides two common paths for setting up to work in Amazon CodeCatalyst with an AWS Builder ID space: creating a space and a project as the first user, and accepting an invitation to an existing space or project. These setup workflows are necessarily quite different. The following diagram shows both sign-up processes as follows:

1. In the first case, you create and set up a space for your company, team, or group, and create a project before inviting others to these resources. An AWS account must be provided for billing purposes, where you can still default to the Free tier.
2. In the second case, if you join CodeCatalyst by accepting an invitation to a project, someone else has already created a space and project for you. However, you'll still want to configure your profile so that you're ready to start working with others.



Tip

CodeCatalyst uses spaces to group projects and resources. When you first sign up for CodeCatalyst, you'll be prompted to create a space as well as a project.

Whether you sign up to create a space and project or you sign up as part of accepting an invitation, you create an AWS Builder ID that you will use to log in to CodeCatalyst. To create an AWS Builder ID, you provide the full name, password, and email address that you use to sign in to AWS applications. You use the email and password to sign in to CodeCatalyst after this point. You can also use this AWS Builder ID to log in to other applications that use AWS Builder ID credentials.

In CodeCatalyst and in AWS Builder ID, a *profile* is generated based on your login information. Your profile contains your CodeCatalyst preferences for language and notification settings in your CodeCatalyst projects.

> **Tip**
>
> If you encounter any problems while signing up for your Amazon CodeCatalyst profile, follow the steps provided on that page. If you need additional help, see Problems signing up.

Topics

Creating your first space and development role (starting without an invitation)

You can sign up for Amazon CodeCatalyst without an invitation to an existing space or project. When you do, you will create a space and project after creating your AWS Builder ID. As part of creating a space, you will need to add an AWS account for billing purposes.

> **Tip**
>
> If you encounter any problems while signing up for your Amazon CodeCatalyst profile, follow the steps provided on that page. If you need additional help, see Problems signing up.

Here is one possible flow for a user starting out with CodeCatalyst without an invitation to a project or a space.

Mary Major is a developer who is interested in CodeCatalyst and decides to try it out. She navigates to the CodeCatalyst console and chooses the option to sign up and create an AWS Builder ID. Mary

provides an email address and password to create her AWS Builder ID. She will be able to use her AWS Builder ID to sign in to CodeCatalyst and other applications. When asked to choose an alias, she specifies `MaryMajor` as the CodeCatalyst user name that will display in CodeCatalyst and that other project members will use to @mention Mary.

Next, Mary is automatically directed to create a space. As part of this flow, Mary is asked to associate an AWS account with the space she's creating so that she can see the sample code in her first project build and deploy. She adds that information and creates her space, where she chooses the option to create a preview development role that can be used for projects in her new space. Mary chooses to create a project, and then she views a list of blueprints for projects. After reviewing the information for the available blueprints, she decides to try the **Modern three-tier web application** blueprint for her first project. She fills in the required fields and creates the project. As soon as the project is ready, she's taken to a project summary page that includes recent activity as well as links to project code and the workflow that automatically builds and deploys that code. She explores both the code and the workflow, including viewing the deployed sample web application. Liking what she sees, she decides to invite some of her co-workers to the project to start exploring CodeCatalyst.

When she has a moment, Mary configures her AWS Builder ID to sign in to CodeCatalyst with multi-factor authentication (MFA). With MFA configured, Mary can sign in to CodeCatalyst using a combination of her CodeCatalyst password and a passcode or token from an approved third-party authentication app.

Creating your first space and IAM roles

Follow these steps to sign up for your Amazon CodeCatalyst profile, create a space, and add an account, a support role, and a developer role for your space.

The final procedure creates and add the developer role. The developer role is an AWS IAM role that enables your CodeCatalyst workflows to access AWS resources. The developer role is a service role used to manage AWS services and will be created in the account that is signed in. A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role.

> **Note**
>
> As a security best practice, only assign administrative access to administrative users and developers who need to manage access to AWS resources in the space.

Before you begin, you must be ready to provide an AWS account ID for an account where you have administrative privileges. Have your 12-digit AWS account ID ready. For information about finding your AWS account ID, see Your AWS account ID and its alias.

To sign up as a new user

1. Before you start in the CodeCatalyst console, open the AWS Management Console, and then make sure you are signed in with the same AWS account that you want to use to create your space.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. On the welcome page, choose **Sign up**. The **Create your AWS Builder ID** page displays. Your AWS Builder ID is an identity you create to sign in. It is not the same as an AWS account.
4. In **Your email address**, enter the email address you want to associate with CodeCatalyst. Then choose **Next**.
5. In **Your name**, provide the first and last name you want displayed in applications where you use your AWS Builder ID. Spaces are allowed. This will be your AWS Builder ID profile name, such as **Mary Major**. You can change the name later.

   Choose **Next**. The **Email verification** page displays.
6. A verification code will be sent to the email you specified. Enter this code in **Verification code**, and then choose **Verify**. If you don't receive your code after 5 minutes and cannot find it in your spam or junk folders, then choose **Resend code**.
7. Once we verify your code, enter a password that meets the requirements in **Password** and **Confirm password**.

   Select the checkbox confirming your agreement with the AWS Customer Agreement and the AWS Service Terms, and then choose **Create AWS Builder ID**.
8. On the **Create your CodeCatalyst alias** page, enter an alias you want to use for your unique user identifier in CodeCatalyst. Choose a shortened version of your name with no spaces, such as **MaryMajor**. Other CodeCatalyst users will use this to @mention you in comments and pull

requests. Your CodeCatalyst profile will contain both your full name from your AWS Builder ID and your CodeCatalyst alias. You cannot change your CodeCatalyst alias later.

Your full name and your alias will display in different areas in CodeCatalyst. For example, your profile name displays for your listed activity in the activity feed, but project members will use your alias to @mention you.

Choose **Next**. The page updates to show the **Create your CodeCatalyst space** section.

9. In **Name your space**, enter the name of your space. You cannot change this later.

> **Note**
>
> Space names must be unique across CodeCatalyst. You cannot reuse names of deleted spaces.

10. In **AWS Region** dropdown menu, choose the region where you want to store your space and project data. You cannot change this later.
11. Choose **Next**. The page updates to show the page for adding an AWS account. This account will be used as the billing account for the space.
12. In **AWS account ID**, enter the twelve-digit ID for the account you want to connect to your space.

    In **AWS account verification token**, copy the generated token ID. The token is automatically copied for you, but you might want to store it while you approve the AWS connection request.
13. Choose **Go to the AWS console to verify**.
14. The **Verify Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst spaces** page. You might need to sign in to access the page.

    In the AWS Management Console, make sure to choose the same AWS Region where you want to create your space.

    To directly access the page, sign in to the Amazon CodeCatalyst Spaces in the AWS Management Console at https://console.aws.amazon.com/codecatalyst/home/.

    The verification token field in the AWS Management Console is automatically populated with the token generated in CodeCatalyst.
15. (Optional) Under **Authorized paid tiers**, choose **Authorize paid tiers (Standard, Enterprise)** to turn on the paid tiers for your billing account.

> **Note**
>
> This does not upgrade the billing tier to a paid tier. However, this configures the AWS account so that you can change the billing tier for your space at any time in CodeCatalyst. You can turn on the paid tiers at any time. Without making this change, the space is only able to use the Free tier.

16. Choose **Verify space**.

    An **Account verified** success message displays to show that the account has been added to the space.

17. Remain on the **Verify Amazon CodeCatalyst space** page. Choose the following link: **To add IAM roles for this space, view space details.**

    The connections page with **CodeCatalyst space details** opens in the AWS Management Console. This is the **Amazon CodeCatalyst spaces** page. You might need to log in to access the page.

18. Return to the CodeCatalyst page, and then choose **Next**.

19. A status message displays while your space is being created. When the space is created, CodeCatalyst the following message is displayed: **Your space is ready. Your last step is creating a project.**. You can do one of the following:

    - Choose **Skip for now**.
    - Choose **Create your first project** for your space. For a tutorial that shows you how to create a project with a blueprint, see Tutorial: Creating a project with the Modern three-tier web application blueprint

> **Note**
>
> If a permissions error or banner is shown, then refresh the page and try to view the page again.

To create and add the CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*

1. Before you start in the CodeCatalyst console, open the AWS Management Console, and then make sure you are logged in with the same AWS account for your space.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
4. Choose the link for the AWS account where you want to create the role. The **AWS account details** page displays.
5. Choose **Manage roles from AWS Management Console**.

   The **Add IAM role to Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst spaces** page. You might need to log in to access the page.
6. Choose **Create CodeCatalyst development administrator role in IAM**. This option creates a service role that contains the permissions policy and trust policy for the development role. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role.

   > **Note**
   >
   > This role is only recommended for use with developer accounts and uses the `AdministratorAccess` AWS managed policy, giving it full access to create new policies and resources in this AWS account.
7. Choose **Create development role**.
8. On the connections page, under **IAM roles available to CodeCatalyst**, view the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role in the list of IAM roles added to your account.
9. To return to your space, choose **Go to Amazon CodeCatalyst**.

To create and add the CodeCatalyst AWSRoleForCodeCatalystSupport

1. Before you start in the CodeCatalyst console, open the AWS Management Console, and then make sure you are logged in with the same AWS account for your space.
2. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.

3. Choose the link for the AWS account where you want to create the role. The **AWS account details** page displays.
4. Choose **Manage roles from AWS Management Console**.

   The **Add IAM role to Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst Spaces** page. You might need to sign in to access the page.
5. Under **CodeCatalyst space details**, choose **Add CodeCatalyst Support role**. This option creates a service role that contains the permissions policy and trust policy for the preview development role. The role will have a name **AWSRoleForCodeCatalystSupport** with a unique identifier appended. For more information about the role and role policy, see Understanding the **AWSRoleForCodeCatalystSupport** service role.
6. On the **Add role for CodeCatalyst Support** page, leave the default selected, and then choose **Create role**.
7. Under **IAM roles available to CodeCatalyst**, view the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role in the list of IAM roles added to your account.
8. To return to your space, choose **Go to Amazon CodeCatalyst**.

After you create your AWS Builder ID, create your first space, and add an account, you can then create a project. For more information, see Creating a project. If this is your first time using CodeCatalyst, we suggest starting with Tutorial: Creating a project with the Modern three-tier web application blueprint.

Accepting an invitation and creating your AWS Builder ID

You can sign up for Amazon CodeCatalyst as part of accepting an invitation to a project or a space. As part of accepting the invitation, you'll be prompted to create an AWS Builder ID. You'll use your AWS Builder ID to access resources in CodeCatalyst.

> **Tip**
>
> If you need additional help, see Problems signing up.

Here is one possible flow for a user starting out with CodeCatalyst with an invitation to a project or a space.

Saanvi Sarkar is a developer who has received an invitation to join a CodeCatalyst project as a project administrator. Saanvi accepts the invitation, which opens the sign-in page for CodeCatalyst. She chooses to sign up and provides an email address and password to create her AWS Builder ID. Saanvi will be able to use her AWS Builder ID to sign in to CodeCatalyst and other applications. Later, she can edit her profile to change her login email address or password. When asked to choose an alias, Saanvi specifies `SaanviSarkar` as the CodeCatalyst alias that will display in CodeCatalyst and that other project members will use to @mention Saanvi. After she has signed up, Saanvi will also be able to use her sign-in credentials for other applications that use AWS Builder ID credentials.

Upon completing sign up, Saanvi automatically joins the CodeCatalyst project and space specified in the invitation. The invitation also provides predetermined permissions for her roles in the project and space. In the project settings, Saanvi's alias shows in the members list with her assigned project role. To work with source repositories in CodeCatalyst, Saanvi takes a moment to create a personal access token (PAT). The PAT will be used in CodeCatalyst for authentication when making source changes or actions that need an authentication token.

When Saanvi works on a project, her alias will be listed in the work activity log for the project. Issues and comments by Saanvi will show her alias, where other project members are able to @mention her in replies. To @mention another project member, Saanvi looks up their alias on their CodeCatalyst profile.

When she has a moment, Saanvi configures her AWS Builder ID to sign in to CodeCatalyst with multi-factor authentication (MFA). With MFA configured, Saanvi can sign in to CodeCatalyst using a combination of her CodeCatalyst password and a passcode or token from an approved third-party authentication app.

Accepting an invitation and creating an AWS Builder ID

When you're invited to a project or space in Amazon CodeCatalyst, you'll receive an email from notify@codecatalyst.aws asking you to accept the invitation. If you already have a AWS Builder ID and are signed in to CodeCatalyst, choosing **Accept invitation** will automatically open the project or space in a browser tab. If you're not signed in to the console but have a AWS Builder ID, you'll be taken to the sign-in page. For more information, see Sign in with your AWS Builder ID.

If you don't have a AWS Builder ID, choosing **Accept invitation** will take you to the sign-in page, where you should choose the option to create your AWS Builder ID.

To accept an invitation and create a AWS Builder ID

1. In the invitation email, choose **Accept invitation**.
2. On the sign in page, choose **Not signed up? Create your AWS Builder ID**.

   > **Tip**
   >
   > Your AWS Builder ID is an identity you create to sign in. It is not the same as an AWS account.

3. On the **Create your AWS Builder ID** page, in **Email address**, enter the email address you want to use for your AWS Builder ID.

   In **Your name**, provide the first and last name you want displayed in applications where you use your AWS Builder ID. Spaces are allowed. This will be your AWS Builder ID profile name, such as **Mary Major**. You can change the name later.

   Choose **Next**.

   A verification code will be sent to the email you specified. Enter this code in **Verification code**, and then choose **Verify**. If you don't receive your code after 5 minutes and cannot find it in your spam or junk folders, then choose **Resend code**.
4. Once your code is verified, enter a password that meets the requirements in **Password** and **Confirm password**.
5. Choose **Create AWS Builder ID**.
6. On the **Create your alias** page, enter an alias you want to use for your unique user identifier in CodeCatalyst. Choose a shortened version of your name with no spaces, such as **MaryMajor**. Other CodeCatalyst users will use this to @mention you in comments and pull requests. Your CodeCatalyst profile will contain both your full name from your AWS Builder ID and your CodeCatalyst alias. You cannot change your CodeCatalyst alias.

   Your full name and your alias will display in different areas in CodeCatalyst. For example, your profile name displays for your listed activity in the activity feed, but project members will use your alias to @mention you.

Choose **Create alias**. You'll be taken to the project or space you were invited to.

Sign in with your AWS Builder ID

Follow these steps to sign in to your Amazon CodeCatalyst profile.

> **Note**
>
> Have you registered a device for multi-factor authentication (MFA) yet? We strongly recommend that you configure MFA in Amazon CodeCatalyst to increase your security. For more information, see How to register a device for use with multi-factor authentication.

To sign in with your AWS Builder ID

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Enter your **Email address**. Optionally, choose **Save my email address** if you want to save your email address for future sign-ins. Choose **Continue**.
3. Enter your **Password**. Choose **Sign in**. If you don't remember your password, follow the steps in I forgot my password.

Trusted devices

After you choose the option **This is a trusted device** from the sign-in page, Amazon CodeCatalyst considers all future sign-ins from that device as authorized. Amazon CodeCatalyst will not present an option to enter an MFA code as long as you use that trusted device. Some exceptions include signing in from a new browser or when your device has been issued an unknown IP address.

Sign in with SSO

Follow these steps to use SSO to sign in to Amazon CodeCatalyst.

To sign in with your AWS Builder ID instead, see Sign in with your AWS Builder ID.

To sign in with SSO

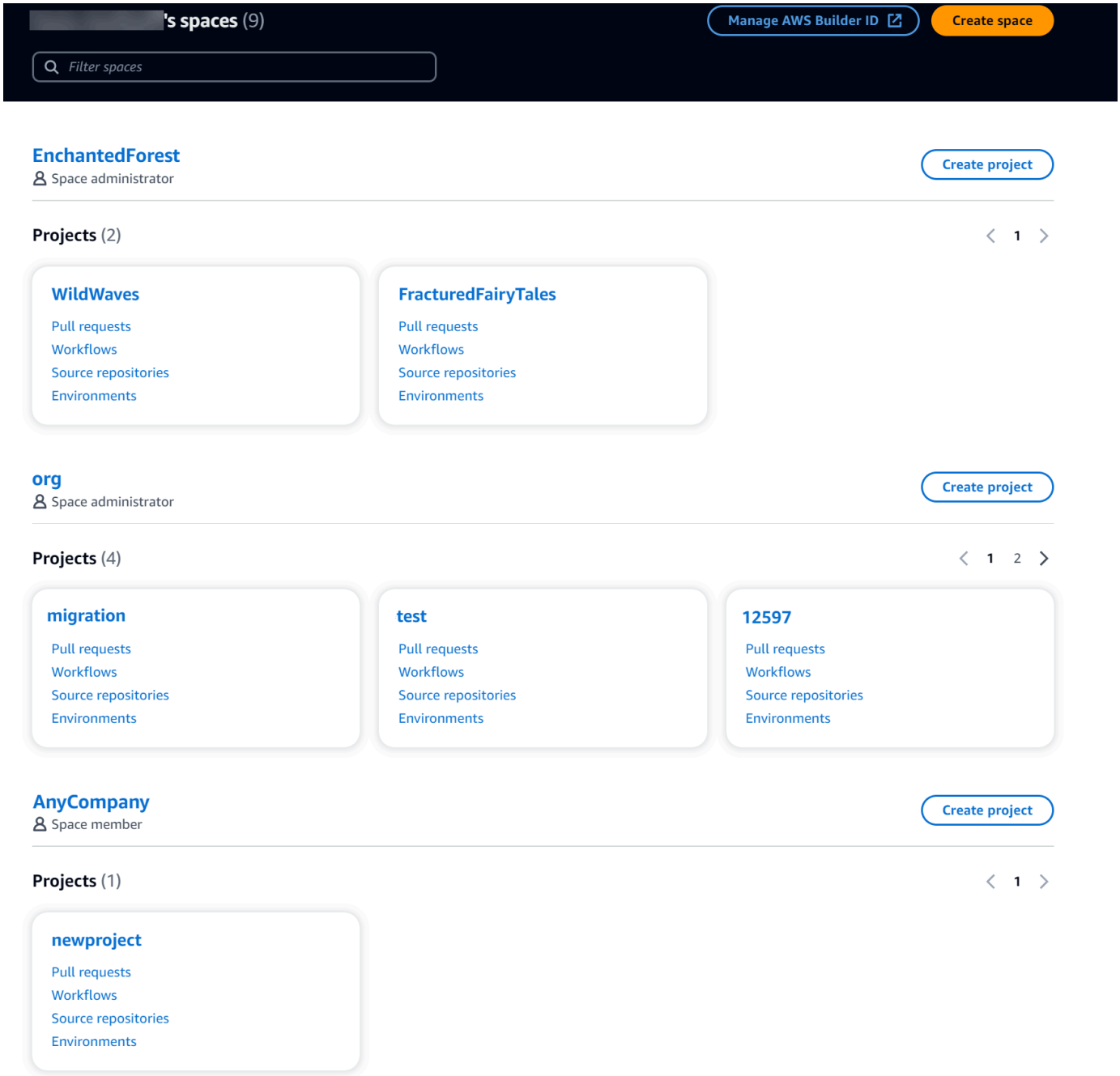
1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. Under **Choose a sign-in option**, choose **Use Single Sign-On (SSO)**.
3. In **AWS Identity Center application name**, enter the application name provided by your identity federation administrator.
4. Choose **Continue to IAM Identity Center**.

View all spaces and projects for a user

You can view a listing of your spaces and projects on the user home page. The user home page shows a listing of each space to which the user belongs, the role for the user in that space, such as **Space administrator**, and the projects in each space where the user has membership.

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the browser, enter the following address: https://codecatalyst.aws/home



3. Choose the space or project you want to open. If you do not see a space or project you expected to see, your might need to sign in as a different user.

Viewing and managing CodeCatalyst profiles

You can view user profiles in Amazon CodeCatalyst to get information such as email addresses and CodeCatalyst aliases. You can also update your profile and your AWS Builder ID. If you forget your password, you can request a password reset.

Viewing your CodeCatalyst profile

You provide information at signup that will be used as your credentials to log in to Amazon CodeCatalyst and that will be managed in your profile. This includes your **Name**, **Nickname**, and the **Email address** you use to sign in to CodeCatalyst.

> **Note**
>
> The AWS Builder ID **Nickname** is not your CodeCatalyst alias. You selected your CodeCatalyst alias at signup.

To view your CodeCatalyst profile

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. At the top right, choose the arrow next to the icon with your first initial, and then choose **My settings**. The CodeCatalyst **My settings** page opens.
3. To update your AWS Builder ID email address or password, or to set up MFA, choose **Manage AWS Builder ID**. The AWS Builder ID page opens.

Viewing another user's CodeCatalyst profile

To view another user's CodeCatalyst profile

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the side navigation, choose **Project settings**. Choose the **Members** tab. View the list of members for your CodeCatalyst project.
3. Choose the member name that you want to look up or @mention. The **My settings** page shows the user's alias, email address, and full name. Use the CodeCatalyst alias to @mention project members.

   > **Note**
   >
   > A user's AWS Builder ID **Nickname** is not their CodeCatalyst alias. They selected their CodeCatalyst alias at signup.

To view another user's profile in your project, choose their name in the list.

Updating your profile

In CodeCatalyst, your profile consists of personal information managed by **AWS Builder ID** and settings managed in CodeCatalyst.

- Your profile's full name, email address, and password are managed by **AWS Builder ID**. You entered this information when you signed up. When you set up MFA to use an authenticator app for application sign-in, CodeCatalyst takes you to the **AWS Builder ID** page.
- CodeCatalyst settings for your personal access token (PAT), CodeCatalyst notifications, and language preferences are managed in the **My settings** page in CodeCatalyst. For more information, see Grant users repository access with personal access tokens.

> **Note**
>
> You can update your AWS Builder ID full name (CodeCatalyst display name) and first name. However, you cannot change your CodeCatalyst alias.

Updating your AWS Builder ID or email address

To update your AWS Builder ID or email address

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. At the top right, choose the arrow next to the icon with your first initial, and then choose **My settings**. The CodeCatalyst **My settings** page opens.
3. On the profile page, choose **Manage AWS Builder ID**. The **AWS Builder ID** page opens.
4. On the left side of the page, choose **My details**.
5. Under **Profile information**, choose **Edit** to update your **Name** or **Nickname**. If you did not specify a nickname, the **Nickname** field reflects the first name in the full name. It is not your CodeCatalyst alias.

> **Note**
>
> This updates the AWS Builder ID full name and first name. This does not update your CodeCatalyst alias.

Under **Contact information**, choose **Edit** to update your **Email address**.

> **Note**
>
> This updates the email address you will use to sign in to CodeCatalyst.

Changing your CodeCatalyst password

To change your CodeCatalyst password

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. At the top right, choose the arrow next to the icon with your first initial, and then choose **User profile**. The CodeCatalyst **My settings** page opens.
3. On the profile page, choose **Manage AWS Builder ID**. The AWS Builder ID page opens.
4. On the left side of the page, choose **Security**.
5. Choose **Change password** and follow the instructions.

Setting up to use the AWS CLI with CodeCatalyst

The Amazon CodeCatalyst console is where you'll work on most of your daily tasks. However, you might want to set up and configure the AWS CLI when you're working with Dev Environments, personal access tokens, or logs of events in CodeCatalyst. You must install the AWS CLI and configure a profile before you can use it with CodeCatalyst.

To set up the AWS CLI for CodeCatalyst

1. Install the latest version of the AWS CLI. If you already have a version of the AWS CLI installed, make sure that it is recent and includes commands for CodeCatalyst, and update it if needed.

To verify that you have a version installed that includes CodeCatalyst commands, open a command prompt and run the following command:

```
aws codecatalyst help
```

If you see a list of CodeCatalyst commands, you have a version that supports CodeCatalyst. If the command is not recognized, update your version of the AWS CLI to the latest version. For more information, see Installing or updating the latest version of the AWS CLI in the AWS Command Line Interface User Guide.

2. Run the **aws configure** command to create a profile if you don't have one or if you want to use a named profile specifically for CodeCatalyst. We recommend creating a named profile to use specifically with CodeCatalyst, but you can also use the default profile. For more information, see Configuration basics.
3. Edit the `config` file for the profile to add a section for connecting to CodeCatalyst as follows. The `config` file is located at `~/.aws/config` on Linux or macOS, or at `C:\Users\`*`USERNAME`*`\.aws\config` on Windows.

```
[profile codecatalyst]
region = us-west-2
sso_session = codecatalyst

[sso-session codecatalyst]
sso_region = us-east-1
sso_start_url = https://view.awsapps.com/start
sso_registration_scopes = codecatalyst:read_write
```

4. Save the file.
5. Before attempting to run any CodeCatalyst commands, open a new terminal or command prompt and run the following command to request and retrieve credentials to run `aws codecatalyst` commands. Replace `codecatalyst` with the name of your profile if needed.

```
aws sso login --profile codecatalyst
```

To view examples of **codecatalyst** commands, see the following topics:

- Grant users repository access with personal access tokens
- Accessing logged events using event logging

Getting started tutorials

Amazon CodeCatalyst provides a number of different templates to help you get started with your projects. You can also choose to start with an empty project and add resources to it. Follow the steps in these tutorials to learn some of the ways you can work in CodeCatalyst.

If this is your first time using CodeCatalyst, we suggest starting with Tutorial: Creating a project with the Modern three-tier web application blueprint.

> **Note**
>
> In order to follow these tutorials, you must first complete setting up. For more information, see Set up and sign in to CodeCatalyst.

Topics

For additional tutorials that focus on specific functional areas in CodeCatalyst, see:

For in-depth tutorials, see:

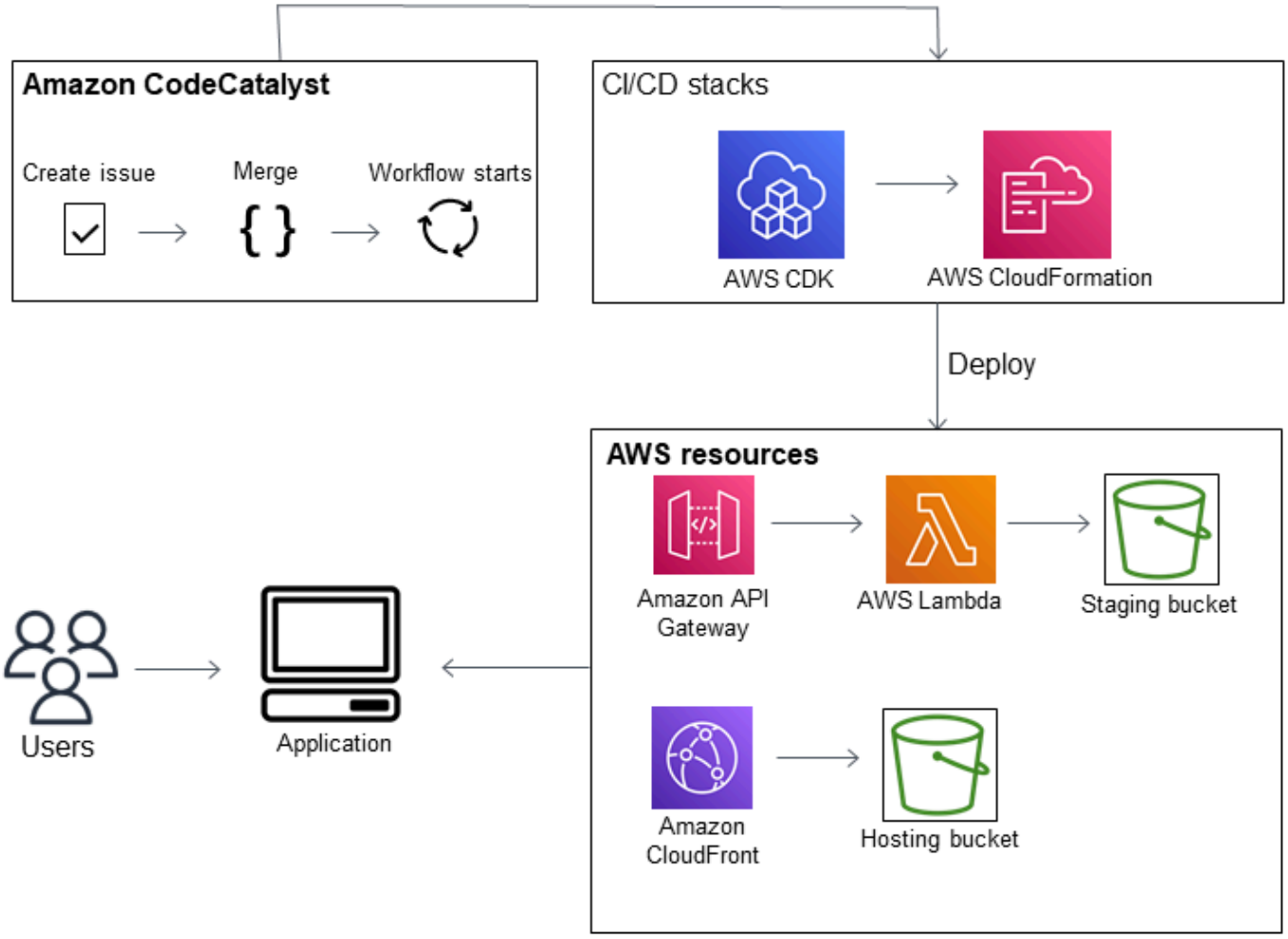
- Tutorial: Deploy an application to Amazon EKS
- Tutorial: Lint code using a GitHub Action in a workflow
- Tutorial: Creating and updating a React application

Tutorial: Creating a project with the Modern three-tier web application blueprint

You can get started more quickly with developing software by creating a project with a blueprint. A project created with a blueprint includes the resources that you need, including a source repository to manage your code, and a workflow to build and deploy the application. In this tutorial, we will walk you through using the **Modern three-tier web application** blueprint to create a project in Amazon CodeCatalyst. The tutorial also includes viewing the deployed sample, inviting other users to work on it, and making changes to the code with pull requests that are automatically built and deployed to resources in the connected AWS account when the pull request is merged. Where CodeCatalyst creates your project with reports, activity feeds, and other tools, your blueprint creates AWS resources in the AWS account associated with your project. Your blueprint files allow you to build and test a sample modern application and deploy it to infrastructure in the AWS Cloud.

The following illustration shows how tools in CodeCatalyst are used to create an issue for tracking, merge and automatically build the change, and then start a workflow in the CodeCatalyst project that runs actions to allow AWS CDK and AWS CloudFormation to provision your infrastructure.

The actions generate resources in the associated AWS account and deploy your application to a serverless AWS Lambdafunction with an API Gateway endpoint. The AWS Cloud Development Kit (AWS CDK) action converts one or more AWS CDK stacks to AWS CloudFormation templates and deploys stacks to your AWS account. Resources in your stacks include Amazon CloudFront resources to distribute dynamic web content, an Amazon DynamoDB instance for your application data, and the roles and policies that support the deployed application.



When you create a project with the **Modern three-tier web application** blueprint, your project is created with the following resources:

In the CodeCatalyst project:

- A source repository with sample code and workflow YAML
- A workflow that builds and deploys the sample code whenever a change is made to the default branch
- An issues board and backlog that you can use to plan and track work
- A test reports suite with automated reports included in the sample code

In the associated AWS account:

- Three AWS CloudFormation stacks that create the resources needed for the application.

For expanded details about the resources that will be created in AWS and CodeCatalyst as part of this tutorial, see Reference.

> **Note**
>
> The resources and samples included in a project depend on which blueprint you select. Amazon CodeCatalyst offers several project blueprints that define resources related to their defined language or framework. To learn more about blueprints, see Creating a comprehensive project with CodeCatalyst blueprints.

Topics

Prerequisites

To create a modern application project in this tutorial, you must have completed the tasks in Set up and sign in to CodeCatalyst as follows:

- Have an AWS Builder ID for signing in to CodeCatalyst.
- Belong to a space and have the **Space administrator** or **Power user** role assigned to you in that space. For more information, see Creating a space, Granting users space permissions, and Space administrator role.
- Have an AWS account associated with your space and have the IAM role you created during sign-up. For example, during sign-up, you have the option to choose to create a service role with a

role policy called the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role policy. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*spaceName* with a unique identifier appended. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role. For the steps to create the role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space.

Step 1: Create the Modern three-tier web application project

After you've created it, your project is where you will develop and test code, coordinate development tasks, and view project metrics. Your project also contains your development tools and resources.

In this tutorial, you will use the **Modern three-tier web application** blueprint to create an interactive application. The workflow that is created and run automatically as part of your project will build and deploy the application. The workflow only runs successfully after all roles and account information is configured for your space. After the workflow has run successfully, you can visit the endpoint URL to see the application.

To create a project with a blueprint

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to create a project.
3. Choose **Create project**.
4. Choose **Start with a blueprint**.
5. In the search bar, enter **`modern`**.
6. Select the **Modern three-tier web application** blueprint, and then choose **Next**.
7. In **Name your project**, enter a project name. For example:

   `MyExampleProject`.

   > **Note**
   >
   > The name must be unique in your space.

8. In **Account**, choose the AWS account you added during sign-up. The blueprint will install resources into this account.

9. In **Deployment Role**, choose the role that you added during sign-up. For example, choose `CodeCatalystWorkflowDevelopmentRole-spaceName`.

   If there are no roles listed, add one. To add a role, choose **Add IAM role** and add the role to your AWS account. For more information, see Allowing access to AWS resources with connected AWS accounts.
10. In **Compute platform**, choose **Lambda**.
11. In **Frontend Hosting Option**, choose **Amplify Hosting**. For information about AWS Amplify, see What is AWS Amplify Hosting? in the *AWS Amplify User Guide*.
12. In **Deployment Region**, enter the Region code of the AWS Region where you want the blueprint to deploy the Mysfits application and supporting resources. For a list of Region codes, see Regional endpoints in the *AWS General Reference*.
13. In **Application name**, leave the default of `mysfitsstring`.
14. (Optional) Under **Generate project preview**, choose **View code** to preview the source files that the blueprint will install. Choose **View workflow** to preview the CI/CD workflow definition files that the blueprint will install. The preview dynamically updates based on your selections.
15. Choose **Create project**.

The project workflow starts as soon as you create the project. It will take a little time to finish building and deploying the code. In the meantime, go ahead and invite someone else to your project.

Step 2: Invite someone to your project

Now that you've set up your project, invite others to work with you.

To invite someone to your project

1. Navigate to the project to which you want to invite users.
2. In the navigation pane, choose **Project settings**.
3. On the **Members** tab, choose **Invite**.
4. Type the email addresses of the people you want to invite as users for your project. You can type multiple email addresses separated by a space or a comma. You can also choose from members of your space who are not members of the project.
5. Choose the role for the user.

When you have finished adding users, choose **Invite**.

Step 3: Create issues to collaborate on and track work

CodeCatalyst helps you track features, tasks, bugs, and any other work involved in your project with issues. You can create issues to track needed work and ideas. By default, when you create an issue it is added to your backlog. You can move issues to a board where you track work in progress. You can also assign an issue to a specific project member.

To create an issue for a project

1. In the navigation pane, choose **Issues**.
2. Choose **Create issue**.
3. In **Issue title**, provide a name for the issue. Optionally, provide a description of the issue. In this example, use **`make a change in the src/mysfit_data.json file.`**
4. Choose the priority, estimate, status, and labels. Under **assignee**, choose **+Add me** to assign the issue to yourself.
5. Choose **Create issue**. The issue is now visible on the board. Choose the card to move the issue to the **In progress** column.

For more information, see Track and organize work with issues in CodeCatalyst.

Step 4: View your source repository

Your blueprint installs a source repository that contains files to define and support your application or service. A few noteworthy directories and files in the source repository are:

- **.cloud9** directory – Contains supporting files for the AWS Cloud9 Dev Environment.
- **.codecatalyst** directory – Contains the YAML workflow definition file for each workflow included in the blueprint.
- **.idea** directory – Contains supporting files for the JetBrains Dev Environments.
- **.vscode** directory – Contains supporting files for the Visual Studio Code Dev Environment.
- **cdkStacks** directory – Contains the AWS CDK stack files that define the infrastructure in the AWS Cloud.
- **src** directory – Contains the application source code.

- **tests** directory – Contains files for the integ and unit tests that are run as part of the automated CI/CD workflow that runs when you build and test your application.
- **web** directory – Contains the frontend source code. Other files include project files such as the `package.json` file that contains important metadata about your project, the `index.html` page for the website, the `.eslintrc.cjs` file for linting code, and the `tsconfig.json` file for specifying root files and compiler options.
- `Dockerfile` file – Describes the application's container.
- `README.md` file – Contains configuration information for the project.

To navigate to the source repositories for a project

1. Navigate to your project, and do one of the following:
   - On the summary page for your project, choose the repository you want from the list, and then choose **View repository**.
   - In the navigation pane, choose **Code**, and then choose **Source repositories**. In **Source repositories**, choose the name of the repository from the list. You can filter the list of repositories by typing part of the repository name in the filter bar.
2. On the home page for the repository, view the contents of the repository and information about the associated resources such as the number of pull requests and workflows. By default, the contents for the default branch are shown. You can change the view by choosing a different branch from the drop-down list.

Step 5: Create a Dev Environment with a test branch and make a quick code change

You can quickly work on the code in your source repository by creating a Dev Environment. For this tutorial, we assume you will:

- Create a AWS Cloud9 Dev Environment.
- Choose the option to work in a new branch off the **main** branch when creating the Dev Environment.
- Use the name `test` for this new branch.

In a later step, you will use the Dev Environment to make a code change and create a pull request.

To create a Dev Environment with a new branch

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to create a Dev Environment.
3. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, choose **Source repositories**, and choose the repository for which you want to create a Dev Environment.
4. On the repository home page, choose **Create Dev Environment**.
5. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.
6. Choose the repository to clone, choose **Work in new branch**, enter a branch name into the **Branch name** field, and choose a branch off of which to create the new branch from the **Create branch from** drop-down menu.
7. Optionally, add an alias for the Dev Environment.
8. Optionally, choose the **Dev Environment configuration** edit button to edit the Dev Environment's compute, storage, or timeout configuration.
9. Choose **Create**. While your Dev Environment is being created, the Dev Environment status column will display **Starting**, and the status column will display **Running** once the Dev Environment has been created. A new tab will open with your Dev Environment in the IDE of your choice. You can edit code and commit and push your changes.

In this section, you will work with your generated sample application in CodeCatalyst by making changes to the code with pull requests that are automatically built and deployed to resources in the connected AWS account when the pull request is merged.

To make a change in your `src/mysfit_data.json` file

1. Navigate to your project Dev Environment. In AWS Cloud9, expand the side navigation menu to browse the files. Expand `mysfits`, `src`, and open `src/mysfit_data.json`.
2. In the file, change the value for the `"Age":` field from 6 to 12. Your line should look like the following:

```
    {
        "Age": 12,
        "Description": "Twilight's personality sparkles like the night sky and is
looking for a forever home with a Greek hero or God. While on the smaller side
```

```
 at 14 hands, he is quite adept at accepting riders and can fly to 15,000 feet.
 Twilight needs a large area to run around in and will need to be registered with
 the FAA if you plan to fly him above 500 feet. His favorite activities include
 playing with chimeras, going on epic adventures into battle, and playing with a
 large inflatable ball around the paddock. If you bring him home, he'll quickly
 become your favorite little Pegasus.",
        "GoodEvil": "Good",
        "LawChaos": "Lawful",
        "Name": "Twilight Glitter",
        "ProfileImageUri": "https://www.mythicalmysfits.com/images/
pegasus_hover.png",
        "Species": "Pegasus",
        "ThumbImageUri": "https://www.mythicalmysfits.com/images/pegasus_thumb.png"
    },
```

3. Save the file.
4. Change to the mysfits repository with the **`cd /projects/mysfits`** command.
5. Add, commit and push your changes with the **git add**, **git commit**, and **git push** commands.

```
git add .
git commit -m "make an example change"
git push
```

Step 6: View the workflow that builds the modern application

After creating the modern application project, CodeCatalyst generates several resources on your behalf, including a workflow. A *workflow* is an automated procedure defined in a .yaml file that describes how to build, test, and deploy your code.

In this tutorial, CodeCatalyst created a workflow and started it automatically when you created your project. (The workflow might still be running depending on how long ago you created your project.) Use the following procedures to check on the workflow's progress, review the generated logs and test reports, and finally, navigate to the URL of the deployed application.

To check on the workflow progress

1. In the CodeCatalyst console, in the navigation pane, choose **CI/CD**, and then choose **Workflows**.

A list of workflows appears. These are the workflows that the CodeCatalyst blueprint generated and started when you created your project.

2. Observe the list of workflows. You should see four:

   - The two workflows at the top correspond to the `test` branch that you created earlier in Step 5: Create a Dev Environment with a test branch and make a quick code change. These workflows are clones the workflows on the `main` branch. The **ApplicationDeploymentPipeline** is not active because it is configured for use with the `main` branch. The **OnPullRequest** workflow did not run because no pull request has been made.
   - The two workflows at the bottom correspond to the `main` branch that was created when you ran the blueprint earlier. The **ApplicationDeploymentPipeline** workflow is active and has an in-progress (or finished) run.

   > **Note**
   >
   > If the **ApplicationDeploymentPipeline** run fails with a **Build@cdk_bootstrap** or **DeployBackend** error, it might be because you ran the Modern three-tier web application previously, and it left old resources behind that conflict with the current blueprint. You'll need to delete these old resources and then re-run the workflow. For more information, see Clean up resources.

3. Choose the **ApplicationDeploymentPipeline** workflow associated with the `main` branch, at the bottom. This workflow was run using the source code on the `main` branch.

   A workflow diagram appears. The diagram shows several blocks, each representing a task or *action*. Most actions are arranged vertically, with the actions at the top running before the ones below. Actions that are arranged side by side run in parallel. Actions that are grouped together must all run successfully before the action below them can start.

   The main blocks are:

   - **WorkflowSource** – This block represents your source repository. It shows, among other information, the source repository name (**mysfits**) and the commit that automatically started the workflow run. CodeCatalyst generated this commit when you created your project.
   - **Build** – This block represents a grouping of two actions which must both complete successfully for the next action to start.

- **DeployBackend** – This block represents an action that deploys the application's backend components into the AWS cloud.
- **Tests** – This block represents a grouping of two test actions which must both complete successfully for the next action to start.
- **DeployFrontend** – This block represents an action that deploys the application's frontend components into the AWS cloud.

4. Choose the **Definition** tab (near the top). The workflow definition file appears on the right. The file has the following noteworthy sections:

   - A `Triggers` section, at the top. This section indicates that the workflow must start whenever code is pushed to the source repository's `main` branch. Pushes to other branches (such as `test`) will not start this workflow. The workflow runs using the files on the `main` branch.
   - An `Actions` section, under `Triggers`. This section defines the actions that you see in the workflow diagram.
5. Choose the **Latest state** tab (near the top), and choose any action in the workflow diagram.
6. On the right, choose the **Configuration** tab to see the configuration settings used by the action during the latest run. Each configuration setting has a matching property in the workflow definition file.
7. Leave the console open and go to the next procedure.

To review the build logs and test reports

1. Choose the **Latest state** tab.
2. In the workflow diagram, choose the **DeployFrontend** action.
3. Wait for the action to finish. Watch for the "in-progress" icon
   ( )
   to change to a "success" icon
   ( ).
4. Choose the **build_backend** action.
5. Choose the **Logs** tab, and expand a couple of sections to view the log messages for these steps. You can see messages related to the backend setup.
6. Choose the **Reports** tab, and then choose the `backend-coverage.xml` report. CodeCatalyst displays the associated report. The report shows the code coverage tests that were run, and

indicates the proportion of lines of code that were successfully validated by testing, such as 80%.

For more information about test reports, see Testing with workflows.

> **Tip**
>
> You can also view your test reports by choosing **Reports** in the navigation pane.

7. Leave the CodeCatalyst console open, and go to the next procedure.

To confirm that the modern application was deployed successfully

1. Return to the **ApplicationDeploymentPipeline** workflow, and choose the **Run-*string*** link of the latest run.
2. In the workflow diagram, find the **DeployFrontend** action and choose the **View app** link. The Mysfit website appears.

> **Note**
>
> If you don't see the **View app** link inside the **DeployFrontend** action, make sure you chose the run ID link.

3. Search for the pegasus Mysfit named **Twilight Glitter**. Note the value for the age. It is 6. You will make a code change to update the age.

Step 7: Ask others to review your changes

Now that you have changes in a branch named `test`, you can ask others to review them by creating a pull request. Perform the following steps to create a pull request to merge the changes from the `test` branch into the `main` branch.

To create a pull request

1. Navigate to your project.
2. Do one of the following:
   - In the navigation pane, choose **Code**, choose **Pull requests**, and then choose **Create pull request**.

- On the repository home page, choose **More**, and then choose **Create pull request**.
- On the project page, choose **Create pull request**.

3. In **Source repository**, make sure that the specified source repository is the one that contains the committed code. This option only appears if you did not create the pull request from the repository's main page.
4. In **Destination branch**, choose the branch to merge the code into after it is reviewed.
5. In **Source branch**, choose the branch that contains the committed code.
6. In **Pull request title**, enter a title that helps other users understand what needs to be reviewed and why.
7. (Optional) In **Pull request description**, provide information such as a link to issues or a description of your changes.

> **Tip**
>
> You can choose **Write description for me** to have CodeCatalyst automatically generate a description of the changes contained in the pull request. You can make changes to the automatically generated description after you add it to the pull request.
> This functionality requires that generative AI features are enabled for the space and is not available for pull requests in linked repositories. For more information, see Managing generative AI features.

8. (Optional) In **Issues**, choose **Link issues**, and then either choose an issue from the list or enter its ID. To unlink an issue, choose the unlink icon.
9. (Optional) In **Required reviewers**, choose **Add required reviewers**. Choose from the list of project members to add them. Required reviewers must approve the changes before the pull request can be merged into the destination branch.

> **Note**
>
> You cannot add a reviewer as both a required reviewer and an optional reviewer. You cannot add yourself as a reviewer.

10. (Optional) In **Optional reviewers**, choose **Add optional reviewers**. Choose from the list of project members to add them. Optional reviewers do not have to approve the changes as a requirement before the pull request can be merged into the destination branch.

11. Review the differences between the branches. The difference displayed in a pull request is the changes between the revision in the source branch and the merge base, which is the head commit of the destination branch at the time the pull request was created. If no changes display, the branches might be identical, or you might have chosen the same branch for both the source and the destination.
12. When you are satisfied that the pull request contains the code and changes you want reviewed, choose **Create**.

> **Note**
>
> After you create the pull request, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resources, such as files, by using the @ sign followed by the name of the file.

When you create the pull request, the **OnPullRequest** workflow starts using the source files in the `test` branch. While your reviewers are approving your code change, you can observe the results by choosing the workflow and viewing the test output.

After you've had the change reviewed, you can merge the code. Merging the code to the default branch will automatically start the workflow that will build and deploy your changes.

To merge a pull request from the CodeCatalyst console

1. Navigate to your modern application project.
2. On the project page, under **Open pull requests**, choose the pull request you want to merge. If you do not see the pull request, choose **View all** and then choose it from the list. Choose **Merge**.
3. Choose from the available merge strategies for the pull request. Optionally select or deselect the option to delete the source branch after merging the pull request, and then choose **Merge**.

> **Note**
>
> If the **Merge** button is not active, or you see the **Not mergeable** label, either one or more required reviewers have not yet approved the pull request, or the pull request cannot be merged in the CodeCatalyst console. A reviewer who has not approved a pull request is indicated by a clock icon in **Overview** in the **Pull request details** area.

If all required reviewers have approved the pull request but the **Merge** button is still not active, you might have a merge conflict. You can resolve merge conflicts for the destination branch in the CodeCatalyst console and then merge the pull request, or you can resolve conflicts and merge locally, and then push the commit that contains the merge to CodeCatalyst. For more information, see Merging a pull request (Git) and your Git documentation.

Once you've merged the changes from the `test` branch into the **`main`** branch, the change automatically starts the **ApplicationDeploymentPipeline** workflow that builds and deploys your change.

To see the merged commit run through the ApplicationDeploymentPipeline workflow

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. In **Workflows**, in **ApplicationDeploymentPipeline**, expand **Recent runs**. You can see the workflow run started by the merge commit. Optionally choose it to watch the run progress.
3. When the run completes, reload the URL you visited earlier. View the pegasus to verify that the age changed.

Step 8: Close the issue

When an issue is resolved, it can be closed on the CodeCatalyst console.

To close an issue for a project

1. Navigate to your project.
2. In the navigation pane, choose **Issues**.
3. Drag-and-drop the issue to the **Done** column.

For more information, see Track and organize work with issues in CodeCatalyst.

Clean up resources

Clean up in CodeCatalyst and AWS to remove traces of this tutorial from your environment.

You can choose to keep using the project you used for this tutorial, or you can delete the project and its associated resources.

> **Note**
>
> Deleting this project will delete all repositories, issues, and artifacts in the project for all members.

To delete a project

1. Navigate to your project, and then choose **Project settings**.
2. Choose the **General** tab.
3. Under the project name, choose **Delete project**.

To delete resources in AWS CloudFormation and Amazon S3

1. Sign in to the AWS Management Console with the same account that you added to your CodeCatalyst space.
2. Go to the **AWS CloudFormation** service.
3. Delete the **mysfits*string*** stack.
4. Delete the **development-mysfits*string*** stack.

5. Choose (but do not delete) the **CDKToolkit** stack. Choose the **Resources** tab. Choose the **StagingBucket** link, and delete the bucket and bucket contents in Amazon S3.

   > **Note**
   >
   > If you don't delete this bucket manually, you may see an error when re-running the Modern three-tier web application blueprint.

6. (Optional) Delete the **CDKToolkit** stack.

Reference

The Modern three-tier web application blueprint deploys resources into your CodeCatalyst space and your AWS account in the AWS cloud. These resources are:

- **In your CodeCatalyst space**:
  - A CodeCatalyst project that includes the following resources:
    - A source repository – This repository contains sample code for a 'Mysfits' web application.
    - A workflow – This workflow builds and deploys the Mysfits application code whenever a change is made to the default branch
    - An issues board and backlog – This board and backlog can be used to plan and track work.
    - A test reports suite – This suite includes automated reports included in the sample code.
- **In the associated AWS account**:
  - A **CDKToolkit** stack – This stack deploys the following resources:
    - An Amazon S3 staging bucket, bucket policy, and the AWS KMS key used to encrypt the bucket.
    - An IAM deployment role for the deploy action.
    - AWS IAM roles and policies in support of the resources in the stack.

    > **Note**
    >
    > The **CDKToolkit** is not torn down and recreated for each deployment. This is a stack that is initiated in each account to support the AWS CDK.

  - A **development-mysfits*string*BackEnd** stack – This stack deploys the following backend resources:

- An Amazon API Gateway endpoint.
- AWS IAM roles and policies in support of the resources in the stack.
- An AWS Lambda function and layer provides the serverless compute platform for the modern application.
- An IAM policy and role for the bucket deployment and Lambda function.

- A **mysfits*string*** stack – This stack deploys the AWS Amplify frontend application.

See also

For more information about the AWS services where resources are created as part of this tutorial, see the following:

- **Amazon S3** – A service for storing your frontend assets on an object storage service offering industry-leading scalability, data high availability, security, and performance. For more information, see Amazon S3 User Guide.
- **Amazon API Gateway** – A service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale For more information, see API Gateway Developer Guide.
- **Amplify** – A service for hosting your frontend application. For more information, see AWS Amplify Hosting User Guide.
- **AWS Cloud Development Kit (AWS CDK)** – A framework for defining cloud infrastructure in code and provisioning it through AWS CloudFormation. The AWS CDK includes the AWS CDK Toolkit, which is a command line tool for interacting with AWS CDK apps and stacks. For more information, see AWS Cloud Development Kit (AWS CDK) Developer Guide.
- **Amazon DynamoDB** – A fully managed NoSQL database service for storing data. For more information, see Amazon DynamoDB Developer Guide.
- **AWS Lambda** – A service for invoking your code on a high availability compute infrastructure without provisioning or managing servers. For more information, see AWS Lambda Developer Guide.
- **AWS IAM** – A service for securely controlling access to AWS and its resources. For more information, see IAM User Guide.

Tutorial: Starting with an empty project and manually adding resources

You can create an empty project without any predefined resources inside it by choosing the **Empty project** blueprint when you create the project. After you create an empty project, you can create and add resources to it according to your project needs. Because projects created without a blueprint are empty on creation, this option requires more knowledge of creating and configuring CodeCatalyst resources to get started.

Topics

Prerequisites

To create an empty project, you must have the **Space administrator** or **Power user** role assigned to you. If this is your first time signing in to CodeCatalyst, see Set up and sign in to CodeCatalyst.

Create an empty project

Creating a project is the first step in being able to work together. If you want to create your own resources, such as source repositories and workflows, you can start with an empty project.

To create an empty project

1. Navigate to the space where you want to create a project.
2. On the space dashboard, choose **Create project**.
3. Choose **Start from scratch**.
4. Under **Give a name to your project**, enter the name that you want to assign to your project. The name must be unique within your space.

5. Choose **Create project**.

Now that you have an empty project, the next step is to create a source repository.

Create a source repository

Create a source repository to store and collaborate on your project's code. Project members can clone this repository to their local computers to work on code. Alternatively, you can choose to link a repository hosted in a supported service, but that is not covered in this tutorial. For more information, see Linking a source repository.

To create a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose **Add repository**, and then choose **Create repository**.
5. In **Repository name**, provide a name for the repository. In this guide, we use *`codecatalyst-source-repository`*, but you can choose a different name. Repository names must be unique within a project. For more information about requirements for repository names, see Quotas for source repositories in CodeCatalyst.
6. (Optional) In **Description**, add a description for the repository that will help other users in the project understand what the repository is used for.
7. Choose **Create repository (default)**. This option creates a repository that includes a default branch and a README.md file. Unlike an empty repository, you can use this repository as soon as it's created.
8. In **Default branch**, leave the name as *main* unless you have a reason to choose a different name. The examples in this guide all use the name *main* for the default branch.
9. (Optional) Add a `.gitignore` file for the type of code you plan to push.
10. Choose **Create**.

> **Note**
>
> CodeCatalyst adds a `README.md` file to your repository when you create it. CodeCatalyst also creates an initial commit for the repository in a default branch

named **main**. You can edit or delete the README.md file, but you can't delete the default branch.

You can quickly add code in your repository by creating a Dev Environment. For this tutorial, we recommend that you create a Dev Environment using AWS Cloud9, and choose the option to create a branch from the **main** branch when creating the Dev Environment. We use the name **`test`** for this branch, but you can enter a different branch name if you prefer.

To create a Dev Environment with a new branch

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to create a Dev Environment.
3. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, choose **Source repositories**, and choose the repository for which you want to create a Dev Environment.
4. On the repository home page, choose **Create Dev Environment**.
5. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.
6. Choose the repository to clone, choose **Work in new branch**, enter a branch name into the **Branch name** field, and choose a branch off of which to create the new branch from the **Create branch from** drop-down menu.
7. Optionally, add an alias for the Dev Environment.
8. Optionally, choose the **Dev Environment configuration** edit button to edit the Dev Environment's compute, storage, or timeout configuration.
9. Choose **Create**. While your Dev Environment is being created, the Dev Environment status column will display **Starting**, and the status column will display **Running** once the Dev Environment has been created. A new tab will open with your Dev Environment in the IDE of your choice. You can edit code and commit and push your changes.

Create a workflow to build, test, and deploy a code change

In CodeCatalyst, you organize the building, testing, and deployment of your applications or services in workflows. Workflows consist of actions and can be configured to run automatically after specified source repository events occur, such as code pushes or opening or updating a pull

request. For more information about workflows, see Build, test, and deploy with workflows in CodeCatalyst.

Follow the instructions in Getting started with workflows to create your first workflow.

Invite someone to your project

Now that you've set up your custom project, invite others to work with you.

To invite someone to your project

1. Navigate to the project to which you want to invite users.
2. In the navigation pane, choose **Project settings**.
3. On the **Members** tab, choose **Invite**.
4. Type the email addresses of the people you want to invite as users for your project. You can type multiple email addresses separated by a space or a comma. You can also choose from members of your space who are not members of the project.
5. Choose the role for the user.

   When you have finished adding users, choose **Invite**.

Create issues to collaborate on and track work

CodeCatalyst helps you track features, tasks, bugs, and any other work involved in your project with issues. You can create issues to track needed work and ideas. By default, when you create an issue it is added to your backlog. You can move issues to a board where you track work in progress. You can also assign an issue to a specific project member.

To create an issue for a project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

   Make sure that you are navigating in the project where you want to create issues. To view all projects, in the navigation pane, choose **Amazon CodeCatalyst**, and if needed, choose **View all projects**. Choose the project where you want to create or work with issues.
2. In the navigation pane, choose **Track**, and then choose **Backlog**.
3. Choose **Create issue**.

4. In **Issue title**, provide a name for the issue. Optionally provide a description of the issue. Choose the status, priority, and estimate for the issue if desired. You can also assign the issue to a project member from the list of project members.

   > **Tip**
   >
   > You can choose to assign an issue to **Amazon Q** to have Amazon Q try to solve the issue. If the attempt is successful, a pull request will be created and the status of the issue will change to **In review** so that you can review and test the code. For more information, see Tutorial: Using CodeCatalyst generative AI features to speed up your development work.
   > This functionality requires that generative AI features are enabled for the space. For more information, see Managing generative AI features.

5. Choose **Save**.

After you have created issues, you can assign them to project members, estimate them, and track them on a Kanban board. For more information, see Track and organize work with issues in CodeCatalyst.

Tutorial: Using CodeCatalyst generative AI features to speed up your development work

If you have a project and a source repository in Amazon CodeCatalyst in a space where generative AI features are enabled, you can use these features to help speed up software development. Developers frequently have more tasks to do than time to accomplish them. They often don't take the time to explain their code changes to their teammates when creating pull requests for review of those changes, expecting other users to find the changes self-explanatory. Pull request creators and reviewers also don't have time to find and read all the comments on a pull request thoroughly, particularly if the pull request has multiple revisions. CodeCatalyst integrates with the Amazon Q Developer Agent for software development to provide generative AI features that can both help team members accomplish their tasks more quickly, and increase the time they have to focus on the most important parts of their work.

Amazon Q Developer is a generative AI-powered conversational assistant that can help you to understand, build, extend, and operate AWS applications. To accelerate your building on AWS, the model that powers Amazon Q is augmented with high-quality AWS content to produce more

complete, actionable, and referenced answers. For more information, see What is Amazon Q Developer? in the *Amazon Q Developer User Guide*.

> **Note**
>
> **Powered by Amazon Bedrock**: AWS implements automated abuse detection. Because the **Write description for me**, **Create content summary**, and **Assign issues to Amazon Q** feature with Amazon Q Developer Agent for software development are built on Amazon Bedrock, users can take full advantage of the controls implemented in Amazon Bedrock to enforce safety, security, and the responsible use of artificial intelligence (AI).

In this tutorial, you'll learn how to use the generative AI features in CodeCatalyst to help you create projects with blueprints, as well as add blueprints to existing projects. Additionally, you'll learn how to summarize changes between branches when creating pull requests and to summarize comments left on a pull request. You'll also learn how to create issues with your ideas for code changes or improvements and assign them to Amazon Q. As part of working with issues assigned to Amazon Q, you'll learn how to allow Amazon Q to suggest tasks and how to assign and work on any tasks it creates as part of working on an issue.

Topics

Prerequisites

To work with the CodeCatalyst features in this tutorial, you must have first completed and have access to the following resources:

- You have an AWS Builder ID or a single sign-on (SSO) identity for signing in to CodeCatalyst.

- Your are in a space that has generative AI features enabled. For more information, see Managing generative AI features.
- You have the Contributor or Project administrator role in a project in that space.
- Unless you are creating a project with generative AI, your existing project has at least one source repository configured for it. Linked repositories are not supported.
- When assigning issues to have an initial solution created by generative AI, the project cannot be configured with the **Jira Software** extension. The extension is not supported for this feature.

For more information, see Creating a space, Track and organize work with issues in CodeCatalyst, Add functionality to projects with extensions in CodeCatalyst, and Granting access with user roles.

This tutorial is based on a project created using the **Modern three-tier web application** blueprint with Python. If you use a project created with a different blueprint, you can still follow the steps, but some specifics will vary, such as sample code and language.

Using Amazon Q to choose a blueprint when creating a project or adding functionality

As a project developer, you can collaborate with Amazon Q, a generative AI assistant, when creating new projects or adding components to existing projects. You can provide Amazon Q with requirements for your project by interacting with it in a chat-like interface. Based on your requirements, Amazon Q suggests a blueprint and also outlines requirements that can't be met. If your space has custom blueprints, Amazon Q learns and includes those blueprints in the recommendations as well. You can then proceed with Amazon Q's suggestion if you're satisfied, and it will create the necessary resources such as a source repository with code for your requirement. Amazon Q also creates issues for requirements that can't be satisfied with a blueprint. To learn more about available CodeCatalyst blueprints, see Creating a comprehensive project with CodeCatalyst blueprints. To learn more about using Amazon Q with blueprints, see Best practices when using Amazon Q to create projects or add functionality with blueprints.

To create a project with Amazon Q

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to create a blueprint.
3. On the space dashboard, choose **Create with Amazon Q**.

4. In the Amazon Q prompt text input field, provide instructions by writing a brief description about the project you want to build. For example, "`I want to create a project in Python that has a presentation layer responsible for how the data is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data.`"

   (Optional) Under **Try examples**, you can use a prewritten prompt by choosing a blueprint. For example, if you choose React app, the following prompt is provided : "`I want to create a project in Python that has a presentation layer responsible for how the data is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data. I also want to add authentication and authorization mechanisms for security and allowable actions.`"

5. Choose **Send** to submit your instructions to Amazon Q. The generative AI assistant provides a suggestion and outlines requirements that can't be met by the blueprint. For example, Amazon Q might suggest the following based on your criteria:

   ```
   I recommend using the Modern three-tier web application blueprint based on
                       your requirements. Blueprints are dynamic and can always be
    updated and edited later.

   Modern three-tier web application
   By Amazon Web Services

   This blueprint creates a Mythical Mysfits 3-tier web application with a modular
    presentation, application, and data layers.
   The application leverages containers, infrastructure as code (IaC), continuous
    integration and continuous delivery (CI/CD),
   and serverless code functions.

   Version: 0.1.163

   View details

   The following requirements could not be met so I will create issues for you.
   • Add authentication and authorization mechanisms for security and allowable
    actions.
   ```

6. (Optional) To view the in-depth details of the suggested blueprint, choose **View details**.

7. Do one of the following:

    a. Choose **Yes, use this blueprint** if you're satisfied with the suggestion.

    b. Choose **Edit prompt** if you want to modify the prompt.

    c. Choose **Start over** if you want to clear the prompt completely.

8. Do one of the following:

    a. Choose **Configure** if you want to configure the blueprint that is suggested. You can also configure the blueprint at a later time.

    b. Choose **Skip** if you don't want to modify the blueprint configurations at the moment.

9. If you chose to configure the blueprint, choose **Continue** after modifying the project resources.

10. When prompted, enter the name that you want to assign to your project and its associated resource names. The name must be unique within your space.

11. Choose **Create project** to create a project with the blueprint. Amazon Q creates resources using the blueprint. For example, if you create a project with the Single-page application blueprint, a source repository for relevant code and workflows for CI/CD are created.

12. (Optional) By default, Amazon Q also creates issues for the requirements that aren't satisfied by a blueprint. You can choose which items you don't want to create issues for. After you choose to let Amazon Q create issues, you can then assign an issue to Amazon Q as well. It'll analyze the issue in the context of the given source repositories, providing a summary of the relevant source files and code. For more information, see Finding and viewing issues, Create an issue and assign it to Amazon Q, and Best practices when creating and working with issues assigned to Amazon Q.

After creating a project with Amazon Q, you can also use Amazon Q to add new components as it suggests CodeCatalyst blueprints based on your requirements.

To add a blueprint with Amazon Q

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. In the CodeCatalyst console, navigate to the project where you want to add a blueprint.

3. Choose **Add with Amazon Q**.

4. In the Amazon Q prompt text input field, provide instructions by writing a brief description about the project you want to build. For example, "`I want to create a project in Python that has a presentation layer responsible for how the data`

is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data."

(Optional) Under **Try examples**, you can use a prewritten prompt by choosing a blueprint. For example, if you choose React app, the following prompt is provided : "I want to create a project in Python that has a presentation layer responsible for how the data is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data. I also want to add authentication and authorization mechanisms for security and allowable actions."

5. Choose **Send** to submit your instructions to Amazon Q. The generative AI assistant provides a suggestion and outlines requirements that can't be met by the blueprint. For example, Amazon Q might suggest the following based on your criteria:

```
I recommend using the Single-page application blueprint based on your requirements.
 Blueprints are dynamic and can always be updated and edited later.

Single-page application
By Amazon Web Services

This blueprint creates a SPA (single-page application) using React, Vue, or Angular
 frameworks and deploys to AWS Amplify Hosting.

Version: 0.2.15
View details

The following requirements could not be met so I will create issues for you.
• The application should have reusable UI components
• The application should support for client-side routing
• The application may require server-side rendering for improved performance and
 SEO
```

6. (Optional) To view the in-depth details of the suggested blueprint, choose **View details**.
7. Do one of the following:

   a. Choose **Yes, use this blueprint** if you're satisfied with the suggestion.

   b. Choose **Edit prompt** if you want to modify the prompt.

   c. Choose **Start over** if you want to clear the prompt completely.

8. Do one of the following:

    a. Choose **Configure** if you want to configure the blueprint that is suggested. You can also configure the blueprint at a later time.

    b. Choose **Skip** if you don't want to modify the blueprint configurations at the moment.

9. If you chose to configure the blueprint, choose **Continue** after modifying the project resources.

10. Choose **Add to project** to add resources to a project with the blueprint. Amazon Q creates resources using the blueprint. For example, if you add to a project with the Single-page application blueprint, a source repository for relevant code and workflows for CI/CD are created.

11. (Optional) By default, Amazon Q also creates issues for the requirements that aren't satisfied by a blueprint. You can choose which items you don't want to create issues for. After you choose to let Amazon Q create issues, you can then assign an issue to Amazon Q as well. It'll analyze the issue in the context of the given source repositories, providing a summary of the relevant source files and code. For more information, see Create an issue and assign it to Amazon Q and Best practices when creating and working with issues assigned to Amazon Q.

Create a summary of the code changes between branches when creating a pull request

A pull request is the primary way you and other project members can review, comment on, and merge code changes from one branch to another. You can use pull requests to review code changes collaboratively for minor changes or fixes, major feature additions, or new versions of your released software. Summarizing the code changes and the intent behind the changes as part of the pull request's description is helpful to others who will review the code, and also helps with a historical understanding of the changes to the code over time. However, developers often rely on their code to explain itself or provide ambiguous details rather than describing their changes with enough details for reviewers to understand what they are reviewing or what the intent was behind the changes in the code.

You can use the **Write description for me** feature when creating pull requests to have Amazon Q create a description of the changes contained in a pull request. When you choose this option, Amazon Q analyzes the differences between the source branch that contains the code changes and the destination branch where you want to merge these changes. It then creates a summary of what those changes are, as well as its best interpretation of the the intent and effect of those changes.

> **Note**
>
> This feature does not work with Git submodules. It will not summarize any changes in a Git submodule that is part of the pull request.
> This feature is not available for pull requests in linked repositories.

You can try this feature with any pull request you create, but in this tutorial, we'll test it out by making some simple changes to the code contained in a project created in a Python-based **Modern three-tier web application** blueprint.

> **Tip**
>
> If you are using a project created with a different blueprint or your own code, you can still follow this tutorial, but the examples in this tutorial will not match the code in your project. Instead of the suggested example below, make simple changes in your project's code in a branch, and then create a pull request to test the feature as shown in the following steps.

First, you will create a branch in the source repository. You'll then make a quick code change to a file in that branch using the text editor in the console. You'll then create a pull request, and use the **Write description for me** feature to summarize the changes you made.

To create a branch (console)

1. In the CodeCatalyst console, navigate to the project where your source repository resides.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.
3. Choose the repository where you want to create a branch.
4. On the overview page of the repository, choose **More**, and then choose **Create branch**.
5. Enter a name for the branch.
6. Choose a branch to create the branch from, and then choose **Create**.

Once you have a branch, edit a file in that branch with a simple change. In this example, you'll edit the `test_endpoint.py` file to change the number of retries for tests from **3** to **5**.

> **Tip**
>
> You can also choose to create or use a Dev Environment to make this code change. For more information, see Creating a Dev Environment.

To edit the `test_endpoint.py` file in the console

1. On the overview page for the **`mysfits`** source repository, choose the branch drop-down and choose the branch you created in the previous procedure.
2. In **Files**, navigate to the file you want to edit. For example, to edit the `test_endpoint.py` file, expand **tests**, expand **integ**, and then choose `test_endpoint.py`.
3. Choose **Edit**.
4. On line 7, change the number of times all tests will be retried from:

   ```
   def test_list_all(retry=3):
   ```

   to:

   ```
   def test_list_all(retry=5):
   ```

5. Choose **Commit** and commit your changes to your branch.

Now that you have a branch with a change, you can create a pull request.

Create a pull request with a summary of the changes

1. On the overview page of the repository, choose **More**, and then choose **Create pull request**.
2. In **Destination branch**, choose the branch to merge the code into after it is reviewed.

   > **Tip**
   >
   > Choose the branch that you created your branch from in the previous procedure for the simplest demonstration of this feature. For example, if you created your branch from the repository's default branch, choose that branch as the destination branch for your pull request.

3. In **Source branch**, choose the branch that contains the changes you just committed to the `test_endpoint.py` file.
4. In **Pull request title**, enter a title that helps other users understand what needs to be reviewed and why.
5. In **Pull request description**, choose **Write description for me** to have Amazon Q create a description of the changes contained in the pull request.
6. A summary of the changes appears. Review the suggested text and then choose **Accept and add to description**.
7. Optionally, modify the summary to better reflect the changes you made to the code. You can also choose to add reviewers or link issues to this pull request. When you have finished making any additional changes you want, choose **Create**.

Create a summary of comments left on code changes in a pull request

When users review a pull request, they often leave multiple comments on the changes in that pull request. If there are a lot of comments from a lot of reviewers, it can be difficult to pick out common themes in the feedback, or even be sure that you've reviewed all the comments in all revisions. You can use the **Create comment summary** feature to have Amazon Q analyze all the comments left on code changes in a pull request and create a summary of those comments.

> **Note**
>
> Comment summaries are transient. If you refresh a pull request, the summary will disappear. Content summaries do not include comments on the overall pull request, just comments left on differences in code in revisions of the pull request.
> This feature does not work with any comments left on code changes in Git submodules.
> This feature is not available for pull requests in linked repositories.

To create a summary of comments in a pull request

1. Navigate to the pull request you created in the previous procedure.

> **Tip**
>
> If you prefer, you can use any open pull request in your project. In the navigation bar, choose **Code**, choose **Pull requests**, and choose any open pull request.

2. Add a few comments to the pull request in **Changes** if the pull request does not already have comments.
3. In **Overview**, choose **Create comment summary**. When complete, the **Comment summary** section expands.
4. Review the summary of comments left on changes in the code in revisions of the pull request, and compare it to the comments in the pull request.

Create an issue and assign it to Amazon Q

Development teams create issues to track and manage their work, but sometimes an issue lingers because either it's not clear who should work on it, or the issue requires research into a particular part of the code base, or other urgent work must be attended to first. CodeCatalyst includes integration with Amazon Q Developer Agent for software development. You can assign issues to a generative AI assistant called **Amazon Q** that can analyze an issue based on its title and its description. If you assign the issue to Amazon Q, it will attempt to create a draft solution for you to evaluate. This can help you and your team to focus and optimize your work on issues that require your attention, while Amazon Q works on a solution for problems you don't have resources to address immediately.

> **Tip**
>
> **Amazon Q** performs best on simple issues and straightforward problems. For best results, use plain language to clearly explain what you want done.

When you assign an issue to **Amazon Q**, CodeCatalyst will mark the issue as blocked until you confirm how you want Amazon Q to work on the issue. It requires you to answer three questions before it can continue:

- Whether you want to confirm every step it takes or whether you want it to proceed without feedback. If you choose to confirm each step, you can reply to Amazon Q with feedback on

the approach it creates so it can iterate on its approach if needed. Amazon Q can also review feedback users leave on any pull request it creates if you choose this option. If you choose not to confirm each step, Amazon Q might complete its work more quickly, but it won't review any feedback you give it in the issue or in any pull request it creates.

- Whether you want to allow it to update workflow files as part of its work. Your project might have workflows configured to start runs on pull request events. If so, any pull request that Amazon Q creates that includes creating or updating workflow YAML might start a run of those workflows included in the pull request. As a best practice, don't choose to allow Amazon Q to work on workflow files unless you are sure there are no workflows in your project that will automatically run these workflows before you review and approve the pull request it creates.
- Whether you want to allow it to suggest creating tasks to break down the work in the issue into smaller increments that can be individually assigned to users, including Amazon Q itself. Allowing Amazon Q to suggest and create tasks can help accelerate development on complex issues by allowing multiple people to work on discrete portions of the issue. It can also help reduce the complexity of understanding the entirety of the work as the work needed to complete each task is ideally simpler than the issue it belongs to.
- What source repository you want it to work in. Even if your project has multiple source repositories, Amazon Q can only work on code in one source repository. Linked repositories are not supported.

Once you have made and confirmed your choices, **Amazon Q** will move the issue into the **In progress** state while it attempts to determine what the request is based on the issue title and its description, as well as the code in the specified repository. It will create a pinned comment where it will provide updates on the status of its work. After reviewing the data, Amazon Q will formulate a potential approach to a solution. Amazon Q records its actions by updating its pinned comment and commenting on its progress on the the issue at every stage. Unlike pinned comments and replies, it does not keep a strictly chronological record of its work. Rather, it puts the most relevant information about its work at the top-level of the pinned comment. It will attempt to create code based on its approach and its analysis of the code already in the repository. If it successfully generates a potential solution, it will create a branch and commit code to that branch. It then creates a pull request that will merge that branch with the default branch. When Amazon Q completes its work, it moves the issue to **In review** so that you and your team knows there is code ready for you to evaluate.

> **Note**
>
> This feature is only available through **Issues** in the US West (Oregon) Region. It is not available if you have configured your project to use Jira with the **Jira Software** extension. Additionally, if you have customized the layout of your board, the issue might not change states. For best results, only use this feature with projects that have a standard board layout.
> This feature does not work with Git submodules. It cannot make changes to any Git submodules included in the repository.
> Once you have assigned an issue to Amazon Q, you cannot change the title or description of the issue or assign it to anyone else. If you unassign **Amazon Q** from the issue, it will finish its current step and then stop work. It cannot resume work or be reassigned to the issue once it is unassigned.
> An issue can be automatically moved into the **In review** column if assigned to Amazon Q if a user chooses to allow it to create tasks. However, the issue in **In review** might still have tasks that are in a different state, such as in the **In progress** state.

In this portion of the tutorial, you will create three issues based on potential features for the code included in projects created with the **Modern three-tier web application** blueprint: one to add a to create a new mysfit creature, one to add a sort feature, and one to update a workflow to include a branch named **`test`**.

> **Note**
>
> If you are working in a project with different code, create issues with titles and descriptions that relate to that code base.

To create an issue and have a solution generated for you to evaluate

1. In the navigation pane, choose **Issues** and make sure you are in the **Board** view.
2. Choose **Create issue**.
3. Give the issue a title that explains what you want to do in plain language. For example, for this issue, enter a title of **`Create another mysfit named Quokkapus`**. In **Description**, provide the following details:

```
Expand the table of mysfits to 13, and give the new mysfit the following
 characteristics:

Name: Quokkapus

Species: Quokka-Octopus hybrid

Good/Evil: Good

Lawful/Chaotic: Chaotic

Age: 216

Description: Australia is full of amazing marsupials, but there's nothing there
 quite like the Quokkapus.
She's always got a friendly smile on her face, especially when she's using her
 eight limbs to wrap you up
in a great big hug. She exists on a diet of code bugs and caffeine. If you've got
 some gnarly code that needsa
assistance, adopt Quokkapus and put her to work - she'll love it! Just make sure
 you leave enough room for
her to grow, and keep that coffee coming.
```

4. (Optional) Attach an image to use as the thumbnail and profile picture for the mysfit to the issue. If you do this, update the description to include details of what images you want to use and why. For example, you might add the following to the description: "The mysfit requires image files to be deployed to the website. Add these images attached to this issue to the source repository as part of the work, and deploy the images to the website."

   > **Note**
   >
   > Attached images might or might not be deployed to the website during the interactions in this tutorial. You can add the images to the website yourself, and then leave comments for Amazon Q to update its code to point to the images you want it to use after it has created a pull request.

   Review the description and make sure it contains all the details that might be needed before you proceed to the next step.

5. In **Assignees**, choose **Assign to Amazon Q**.

6. In **Source repository**, choose the source repository that contains the project code.
7. Slide the **Require Amazon Q to stop after each step and await review of its work** selector to the active state if necessary.

> **Note**
>
> Choosing the option to have Amazon Q stop after every step allows you to comment on the issue or any created tasks to have the option to have Amazon Q change its approach up to three times based on your comments. If you choose the option to not have Amazon Q stop after every step so that you can review its work, work might proceed more quickly because Amazon Q isn't waiting for your feedback, but you won't be able to influence the direction Amazon Q takes by leaving comments. Amazon Q will also not respond to comments left in a pull request if you choose that option.

8. Leave the **Allow Amazon Q to modify workflow files** selector in the inactive state.
9. Slide the **Allow Amazon Q to suggest creating tasks** selector to the active state.
10. Choose **Create issue**. Your view changes to the Issues board.
11. Choose **Create issue** to create another issue, this time one with the title **`Change the get_all_mysfits() API to return mysfits sorted by the Age attribute`**. Assign this issue to **Amazon Q** and create the issue.
12. Choose **Create issue** to create another issue, this time one with the title **`Update the OnPullRequest workflow to include a branch named test in its triggers`**. Optionally link to the workflow in the description. Assign this issue to **Amazon Q** but this time make sure that the **Allow Amazon Q to modify workflow files** selector is set to the active state. Create the issue to return to the Issues board.

> **Tip**
>
> You can search for files, including workflow files, by entering the at symbol (@) and entering the file name.

Once you have created and assigned the issues, the issues will move into **In progress**. Amazon Q will add comments tracking its progress inside the issue in a pinned comment. If it is able to define an approach to a solution, it will update the issue's description with a **Background** section that contains its analysis of the code base and a **Approach** section that details its proposed approach

to creating a solution. If Amazon Q is successful in coming up with a solution to the problem described in the issue, it will create a branch and code changes in that branch that implement its proposed solution. If the proposed code contains similarities to open source code that Amazon Q is aware of, it will provide a file that includes links to that code so that you can review it. Once the code is ready, it creates a pull request so that you can review the suggested code changes, adds a link to that pull request to the issue, and moves the issue into **In review**.

> **Important**
>
> You should always review any code changes in a pull request before merging it. Merging code changes made by Amazon Q, like any other code changes, can negatively impact your code base and infrastructure code if the merged code is not properly reviewed and contains errors when merged.

To review an issue and linked pull request that contains changes made by Amazon Q

1. In **Issues**, choose an issue assigned to Amazon Q that is in **In progress**. Review the comments to monitor the progress of Amazon Q. If present, review the background and approach it records in the description of the issue. If you chose to allow Amazon Q to suggest tasks, review any proposed tasks and take any needed action. For example, if Amazon Q suggested tasks and you would like to change the order or assign tasks to specific users, choose **Change, add, or reorder tasks** and perform any updates necessary. When you are done viewing the issue, choose **X** to close the issue pane.

   > **Tip**
   >
   > To view progress on tasks, choose the task from the list of tasks in the issue. Tasks don't appear as separate items on the board and can only be accessed through an issue. If a task is assigned to Amazon Q, you must open the task to approve any actions it wants to perform. You must also open a task to see any linked pull requests as they will not appear as links in the issue, only in the task. To return to an issue from a task, choose the link to the issue.

2. Now choose an issue assigned to Amazon Q that is in **In review**. Review the background and approach it records in the description of the issue. Review the comments to understand the actions it performed. Review any tasks created for work related to this issue, including their

progress, any actions you might need to take, and any comments. In **Pull requests**, choose the link to the pull request next to the **Open** label to review the code.

> **Tip**
>
> Pull requests generated for tasks only appear as linked pull requests in the task view. They don't appear as linked pull requests for the issue.

3. In the pull request, review the code changes. For more information, see Reviewing a pull request. Leave comments on the pull request if you want Amazon Q to change any of its suggested code. Be specific when leaving comments for Amazon Q for best results.

   For example, when reviewing the pull request created for **`Create another mysfit named Quokkapus`**, you might notice that there's a typo in the description. You could leave a comment for Amazon Q that states "Change the description to fix the typo "needsa" by adding a space between "needs" and "a"." Alternatively, you could leave a comment that tells Amazon Q to update the description and provide the entire revised description for it to incorporate.

   If you uploaded images for the new mysfit to the website, you can leave a comment for Amazon Q to update the mysfit with pointers to the image and thumbnail to use for the new mysfit.

   > **Note**
   >
   > Amazon Q will not respond to individual comments. Amazon Q will only incorporate feedback left in comments in pull requests if you chose the default option of stopping after every step for approval when you created the issue.

4. (Optional) After you and other project users have left all the comments you want for changes to the code, choose **Create revision** to have Amazon Q create a revision of the pull request that incorporates the changes you requested in comments. Progress on the revision creation progress will be reported by Amazon Q in **Overview**, not in **Changes**. Make sure to refresh your browser to view the latest updates from Amazon Q on creating the revision.

   > **Note**
   >
   > Only the user who created the issue can create a revision of the pull request. You can only request one revision of a pull request. Make sure that you have addressed

all problems with comments, and that you are satisfied with the content of the comments, before you choose **Create revision**.

5. A workflow is run for each pull request in this example project. Make sure that you see a successful workflow run before you merge the pull request. You can also choose to create additional workflows and environments to test the code before you merge it. For more information, see Getting started with workflows.
6. When you are satisfied with the latest revision of the pull request, choose **Merge**.

Create an issue and have tasks recommended for it by Amazon Q

An issue can sometimes contain complex or lengthy amounts of work. CodeCatalyst includes integration with Amazon Q Developer Agent for software development. You can ask **Amazon Q** to analyze an issue based on its title and its description, and recommend a logical break down of the work into separate tasks. It will attempt to create a list of recommended tasks that can then review, modify, and choose whether to create. This can help you and your team to assign individual parts of the work to users in more managable ways that can be achieved more quickly.

To create and review a list of recommended tasks for an issue

1. In the navigation pane, choose **Issues** and make sure you are in the **Board** view.
2. Choose **Create issue**.
3. Give the issue a title that explains what you want to do in plain language. For example, for this issue, enter a title of **`Change the get_all_mysfits() API to return mysfits sorted by the Good/Evil attribute`**. In **Description**, provide the following details:

```
Update the API to allow sorting of mysfits by whether they are Good, Neutral, or
 Evil. Add a button on the website that allows users to quickly choose this sort
 and to exclude alignments that they don't want to see.
```

4. Review the description and make sure it contains all the details that might be needed before you proceed to the next step.
5. In **Assignees**, choose to assign the issue to yourself.
6. Choose **Create issue**. Your view changes to the Issues board.
7. Choose the issue you just created to open it. Choose **Recommend tasks**.

8. Choose the source repository that contains the code for the issuse. Choose **Start recomending tasks**.

The dialog will close and Amazon Q will begin analyzing the issue for complexity. If the issue is complex, it will suggest a break down of the work into separate, sequential tasks. When the list is ready, choose **View recommended tasks**. You can add additional tasks, modify the recommended tasks, and reorder the tasks. If you agree with the recommendations, choosing **Create tasks** will create the tasks. You can then assign those tasks to users to work on them, or even to Amazon Q itself.

Clean up resources

Once you've completed this tutorial, consider taking the following actions to clean up any resources you created during this tutorial that you no longer need.

- Unassign Amazon Q from any issues no longer being worked on. If Amazon Q has finished its work on an issue or could not find a solution, make sure to unassign Amazon Q to avoid reaching the maximum quota for generative AI features. For more information, see Managing generative AI features and Pricing.
- Move any issues where work is complete to **Done**.
- If the project is no longer needed, delete the project.

Tutorial: Creating a full-stack application with composable PDK blueprints

Amazon CodeCatalyst provides a number of different blueprints to help you get started with your projects quickly. A project created with a blueprint includes the resources that you need, including a source repository, sample source code, CI/CD workflows, build and test reports, and integrated issue tracking tools. However, sometimes you might want to gradually build out a project, or add functionality to an existing project created by a blueprint. You can also do this with blueprints. This tutorial demonstrates how you can get started with a a single blueprint that sets a foundation and allows you to store all your project code in a single repository. From there, you have the flexibility to incorporate additional resources and infrastructure by adding other blueprints on top of the initial blueprint at your convenience. Through this building-block method, you can address specific requirements across multiple projects.

This tutorial shows how to compose multiple AWS Project Development Kit (AWS PDK) blueprints together to create an application comprising of a React website, Smithy API, and the supporting CDK infrastructure to deploy it to AWS. The AWS PDK provides building blocks for common patterns along with development tools to manage and build your projects. For more information, see the AWS PDK GitHub source repository.

The following PDK blueprints are designed to be used with each other to build an application in a composable fashion:

- **Monorepo** - Creates a root-level project that manages interdependencies between projects within the monorepo. The project also provides build caching and dependency visualization.
- **Type Safe API** - Creates an API that can be defined in either Smithy or OpenAPI v3, and manages build-time code generation to allow you to implement and interact with your API in a type-safe manner. Vends a CDK construct that manages deploying your API to API Gateway and configures automatic input validation.
- **Cloudscape React website** - Creates a React-based website built using Cloudscape that comes preintegrated with Cognito Auth and (optionally) your created API, which provides you with the ability to call your API securely.
- **Infrastructure** - Creates a project that sets up all CDK-related infrastructure needed to deploy your application. It also comes preconfigured to generate a diagram based on your CDK code every time you build.
- **DevOps** - Creates DevOps workflows compatible with constructs found in the AWS Project Development Kit (AWS PDK).

The tutorial also includes steps on how to view the deployed application, invite other users to work on it, and make changes to the code with pull requests that are automatically built and deployed to resources in the connected AWS account when the pull request is merged.

When you create a project comprised of PDK blueprints, your project is created with the following resources in a CodeCatalyst project:

- A source repository configured as a monorepo.
- A workflow that runs static code analysis and license checking, as well as builds and deploys the sample code whenever a change is made to the default branch. An architecture diagram is generated each time you make changes to the code.
- An issues board and backlog that you can use to plan and track work.

- A test report suite with automated reports.

Topics

Prerequisites

To create and update a project, you must have completed the tasks in Set up and sign in to CodeCatalyst as follows:

- Have an AWS Builder ID for signing in to CodeCatalyst.
- Belong to a space and have the **Space administrator** or **Power user** role assigned to you in that space. For more information, see Creating a space, Granting users space permissions, and Space administrator role.
- Have an AWS account associated with your space and have the IAM role you created during sign-up. For example, during sign-up, you have the option to choose to create a service role with a role policy called the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role policy. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* with a unique identifier appended. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role. For the steps to create the role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space.

Step 1: Create a monorepo project

Begin with the **PDK - Monorepo** blueprint to create your monorepo codebase that acts as the foundation, allowing you to add additional PDK blueprints.

To create a project using the PDK - Monorepo blueprint

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to create a project.
3. On the space dashboard, choose **Create project**.
4. Choose **Start with a blueprint**.
5. Choose the **PDK - Monorepo** blueprint, and then choose **Next**.
6. Under **Name your project**, enter the name that you want to assign to your project and its associated resource names. The name must be unique within your space.
7. Under **Project resources**, do the following:

    a. Under **Primary programming language**, choose a language you want to develop your project code in. You can choose from TypeScript, Java, or Python.

    b. Choose **Code Configuration**

    c. In the **Source Repository** text input field, enter the name of a source repository, which will create a new repository, or select from an existing linked repository. The existing repository must be empty. For more information, see Linking a source repository.

    d. (Optional) From the **Package Manager** dropdown menu, choose a package manager. This is only necessary if you selected TypeScript as your **Primary programming language**.
8. (Optional) To preview code that will be generated based on the project parameter selections you made, choose **View code** from **Generate project preview**.
9. (Optional) Choose **View details** from the blueprint's card to view specific details about the blueprint, such as an overview of the blueprint's architecture, required connections and permissions, and the kind of resources the blueprint creates.
10. Choose **Create project** to create your monorepo project. The root-level project created manages interdependencies between projects within the monorepo, as well as provides build caching and dependency management.

    For more information about project blueprints, see Creating a comprehensive project with CodeCatalyst blueprints.

The **PDK - Monorepo** blueprint only generates the foundation of the project. To create a workable application using the blueprint, you need to add other PDK blueprints such as Type Safe API, Cloudscape React Website, Infrastructure, or DevOps. In the next step, you'll add a Type Safe API to the project.

Step 2: Add Type Safe API to the project

The **PDK - Type Safe API** blueprint allows you to define an API using Smithy or OpenAI v3. It generates runtime packages from your API definition, which include clients for interacting with your API and server-side code for implementing your API. The blueprint also generates a CDK construct with type safety for every API operation. You can add the blueprint to an existing PDK monorepo project to add API capabilities to the project.

To add the PDK - Type Safe API blueprint

1. In the navigation pane of your monorepo project, choose **Blueprints**, and then choose **Add blueprint**.
2. Choose the **PDK - Type Safe API** blueprint, and then choose **Next**.
3. Under **Configure blueprint**, configure the blueprint parameters:
   - Under **Model Language**, choose the language the API model is defined in.
   - In the **Namespace** text input field, enter a namespace for your API.
   - In the **API name** text input field, enter a name for your API.
   - Under **CDK language**, choose your preferred language to write CDK infrastructure to deploy the API in.
   - Choose the **Handler language(s)** dropdown menu, and then choose the languages you want to implement handlers for API operations in.
   - Choose the **Documentation format(s)** dropdown menu, and then choose the formats you want for generating API documentation.
4. In the **Code changes** tab, review the proposed changes. The difference displayed in a pull request shows the changes to your project at the time the pull request was created.
5. When you're satisfied with the proposed changes that will be made when the blueprint is applied, choose **Add blueprint**.

   After a pull request is created, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resource such as files by using the @ sign, followed by the name of the file.

> **Note**
>
> The blueprint won't be applied until the pull request is approved and merged. For more information, see Reviewing a pull request and Merging a pull request.

6. From the **Status** column, choose **Pending pull request** for the **PDK - Type Safe API** blueprint row, and then choose the link of the open pull request.
7. Choose **Merge**, choose your preferred merge strategy, and then choose **Merge** to incorporate changes from the applied blueprint.

   After the pull request is merged, a new `packages/apis/mypdkapi` folder is generated within your monorepo project, which contains all of the API related source code for your configured Type Safe API.
8. In the navigation pane, choose **Blueprints** to confirm that the **Status** of the **PDK - Type Safe API** shows **Up to date**.

Step 3: Add a Cloudscape React Website for the project

The **PDK - Cloudscape React Website** blueprint generates a website. You can associate an optional parameter (Type Safe APIs) to automatically configure your website to set up authenticated type safe clients along with an interactive API explorer to test your various API's.

To add the PDK - Cloudscape React Website blueprint

1. In the navigation pane of your monorepo project, choose **Blueprints**, and then choose **Add blueprint**.
2. Choose the **PDK - Cloudscape React Website** blueprint, and then choose **Next**.
3. Under **Configure blueprint**, configure the blueprint parameters:

   - In the **Website name** text input field, enter a name for your website.
   - Choose the **Type Safe APIs** dropdown menu, and then choose the API blueprints you want to integrate within the website. Passing in an API sets up authenticated clients and adds the required dependencies, API explorer, and other capabilities.
4. In the **Code changes** tab, review the proposed changes. The difference displayed in a pull request shows the changes to your project at the time the pull request was created.

5. When you're satisfied with the proposed changes that will be made when the blueprint is applied, choose **Add blueprint**.

   After a pull request is created, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resource such as files by using the @ sign, followed by the name of the file.

   > **Note**
   >
   > The blueprint won't be applied until the pull request is approved and merged. For more information, see Reviewing a pull request and Merging a pull request.

6. From the **Status** column, choose **Pending pull request** for the **PDK - Cloudscape React Website** blueprint row, and then choose the link of the open pull request.
7. Choose **Merge**, choose your preferred merge strategy, and then choose **Merge** to incorporate changes from the applied blueprint.

   After the pull request is merged, a new `packages/websites/my-website-name` folder is generated within your monorepo project, which contains all the source code for your new website.
8. In the navigation pane, choose **Blueprints** to confirm that the **Status** of the **PDK - Cloudscape React Website** shows **Up to date**.

Next, you'll add the **PDK - Infrastructure** blueprint to generate the infrastructure to deploy your website to the AWS cloud.

Step 4: Generate the infrastructure to deploy the application to AWS cloud

The **PDK - Infrastructure** blueprint sets up a package containing all of your CDK code to deploy your website and API. It also provides diagram generation and conformance to the prototyping nag pack by default.

To add the PDK - Infrastructure blueprint

1. In the navigation pane of your monorepo project, choose **Blueprints**, and then choose **Add blueprint**.
2. Choose the **PDK - Infrastructure** blueprint, and then choose **Next**.

3. Under **Configure blueprint**, configure the blueprint parameters:

   - Under **CDK language**, choose the language you want to develop your infrastructure with.
   - In the **Stack name** text input field, enter a name of the CloudFormation stack generated for your blueprint.

     > **Note**
     >
     > Keep note of this stack name for the next step, where you'll be setting a DevOps workflow.

   - Choose the **Type Safe APIs** dropdown menu, and then choose the API blueprints you want to integrate within the website.
   - Choose the **Cloudscape React TS Websites** dropdown menu, and then choose website blueprints you want to deploy within your infrastructure (for example, **PDK - Cloudscape React Website**).
4. In the **Code changes** tab, review the proposed changes. The difference displayed in a pull request shows the changes to your project at the time the pull request was created.
5. When you're satisfied with the proposed changes that will be made when the blueprint is applied, choose **Add blueprint**.

   After a pull request is created, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resource such as files by using the @ sign, followed by the name of the file.

   > **Note**
   >
   > The blueprint won't be applied until the pull request is approved and merged. For more information, see Reviewing a pull request and Merging a pull request.

6. From the **Status** column, choose **Pending pull request** for the **PDK - Infrastructure** blueprint row, and then choose the link of the open pull request.
7. Choose **Merge**, choose your preferred merge strategy, and then choose **Merge** to incorporate changes from the applied blueprint.

After the pull request is merged, a new `packages/infra` folder is generated within your monorepo project, which contains the infrastructure that will deploy your project into the AWS cloud.

8. In the navigation pane, choose **Blueprints** to confirm that the **Status** of the **PDK - Infrastructure** shows **Up to date**.

Next, you'll add the **PDK - DevOps** blueprint to deploy your application.

Step 5: Set up a DevOps workflow to deploy your project

The **PDK - DevOps** blueprint generates the required DevOps workflows to build and deploy your project using the AWS account and role specified in the configuration.

To add the PDK - DevOps blueprint

1. In the navigation pane of your monorepo project, choose **Blueprints**, and then choose **Add blueprint**.
2. Choose the **PDK - DevOps** blueprint, and then choose **Next**.
3. Under **Configure blueprint**, configure the blueprint parameters:

   - Choose **Bootstrap CDK** in the current environemnt.
   - In the **Stack name** text input field, enter a name of the CloudFormation stack you want to deploy. This should match the stack name configured in Step 4: Generate the infrastructure to deploy the application to AWS cloud for the **PDK - Infrastructure** blueprint.
   - Choose the **AWS account connection** dropdown menu, and then choose the AWS account you want to use for the resources. For more information, see Adding an AWS account to a space.
   - Choose **The role to use for deploying your application** dropdown menu, and then choose the IAM role you want to use for deploying your project application.

   > **Note**
   >
   > When creating an IAM role, restrict the `SourceArn` to the current `ProjectID` found within project settings. For more information, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role.

- Choose the **Region** dropdown menu, and then choose the region you want to deploy your monorepo project. Deployment only works in regions where the required AWS services exist. For more information, see AWS Services by Region.

4. In the **Code changes** tab, review the proposed changes. The difference displayed in a pull request shows the changes to your project at the time the pull request was created.
5. When you're satisfied with the proposed changes that will be made when the blueprint is applied, choose **Add blueprint**.

   After a pull request is created, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resource such as files by using the @ sign, followed by the name of the file.

   > **Note**
   >
   > The blueprint won't be applied until the pull request is approved and merged. For more information, see Reviewing a pull request and Merging a pull request.

6. From the **Status** column, choose **Pending pull request** for the **PDK - Infrastructure** blueprint row, and then choose the link of the open pull request.
7. Choose **Merge**, choose your preferred merge strategy, and then choose **Merge** to incorporate changes from the applied blueprint.

   After the pull request is merged, a new `.codecatalyst/workflows` folder is generated within your monorepo project.
8. In the navigation pane, choose **Blueprints** to confirm that the **Status** of the **PDK - DevOps** shows **Up to date**.

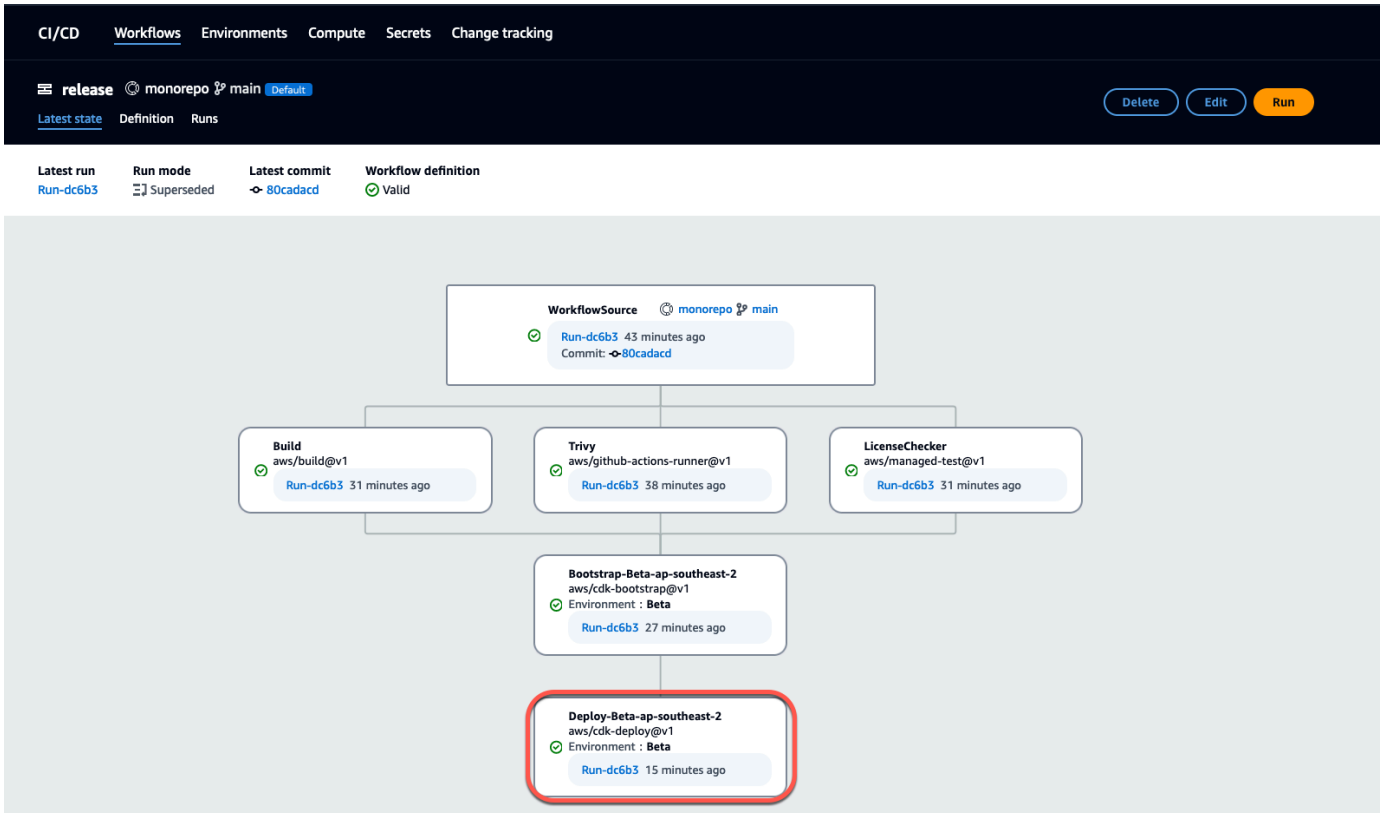
> **Note**
>
> The **PDK - DevOps** blueprint and all subsequent changes to PDK blueprints will be significantly slower from this point on because behind the scenes lockfiles are generated to ensure that builds and deployments are repeatable in future. It'll generate lockfiles for all packages across any of the supported languages.

Step 6: Confirm release workflow and view your website

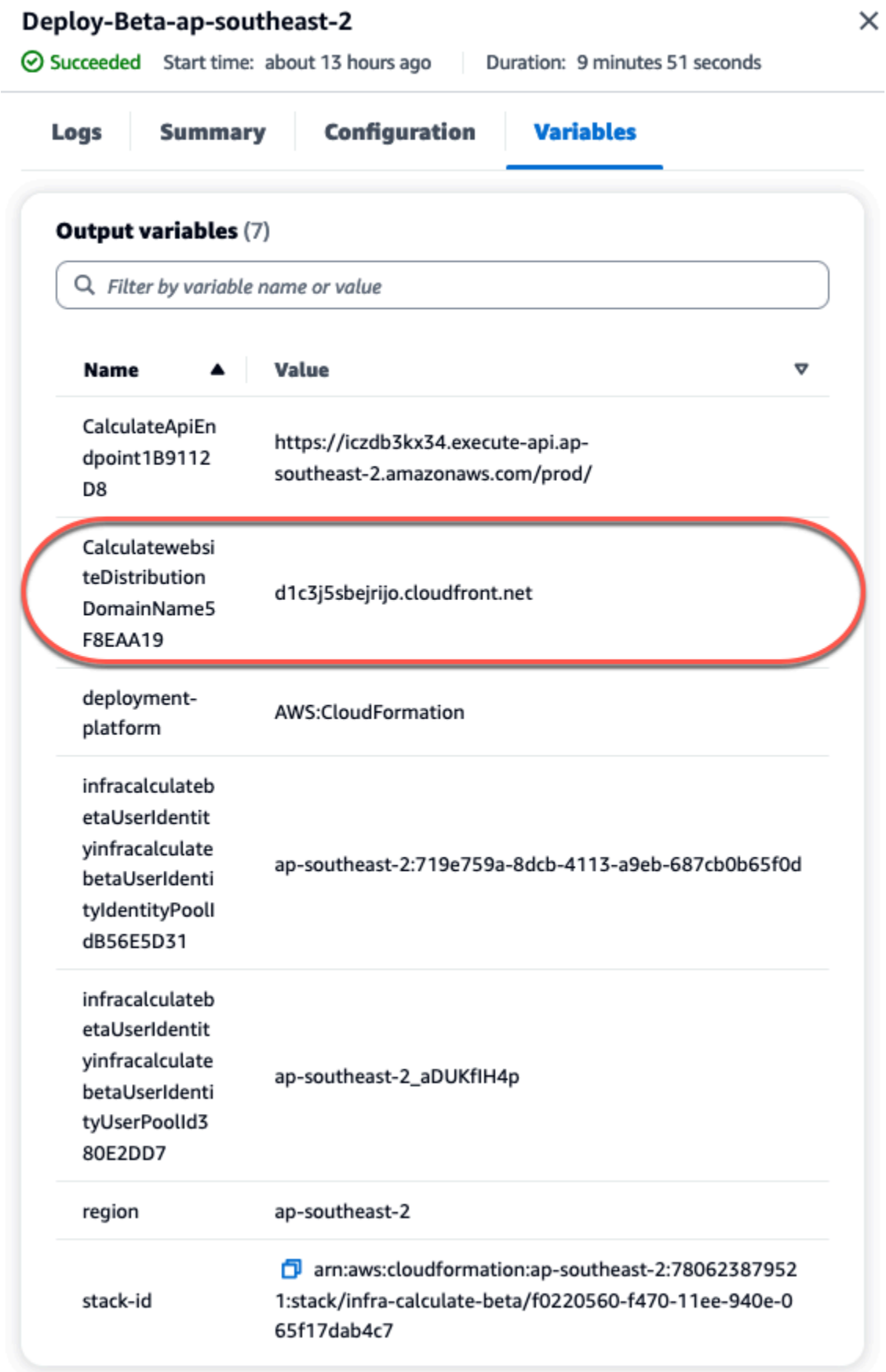
Once you have completed the previous steps, you can confirm the release workflow to ensure the project is being built.

To confirm the release workflow and view your website

1. In the navigation pane of your monorepo project, choose **CI/CD**, and then choose **Workflows**.
2. For the **release** workflow, choose the latest workflow run to view the details. For more information, see Viewing the status and details of a single run.
3. After the workflow run is successfully complete, choose the last action in the workflow (for example, **Deploy-Beta-ap-souteast-2**, and then choose **Variables**.



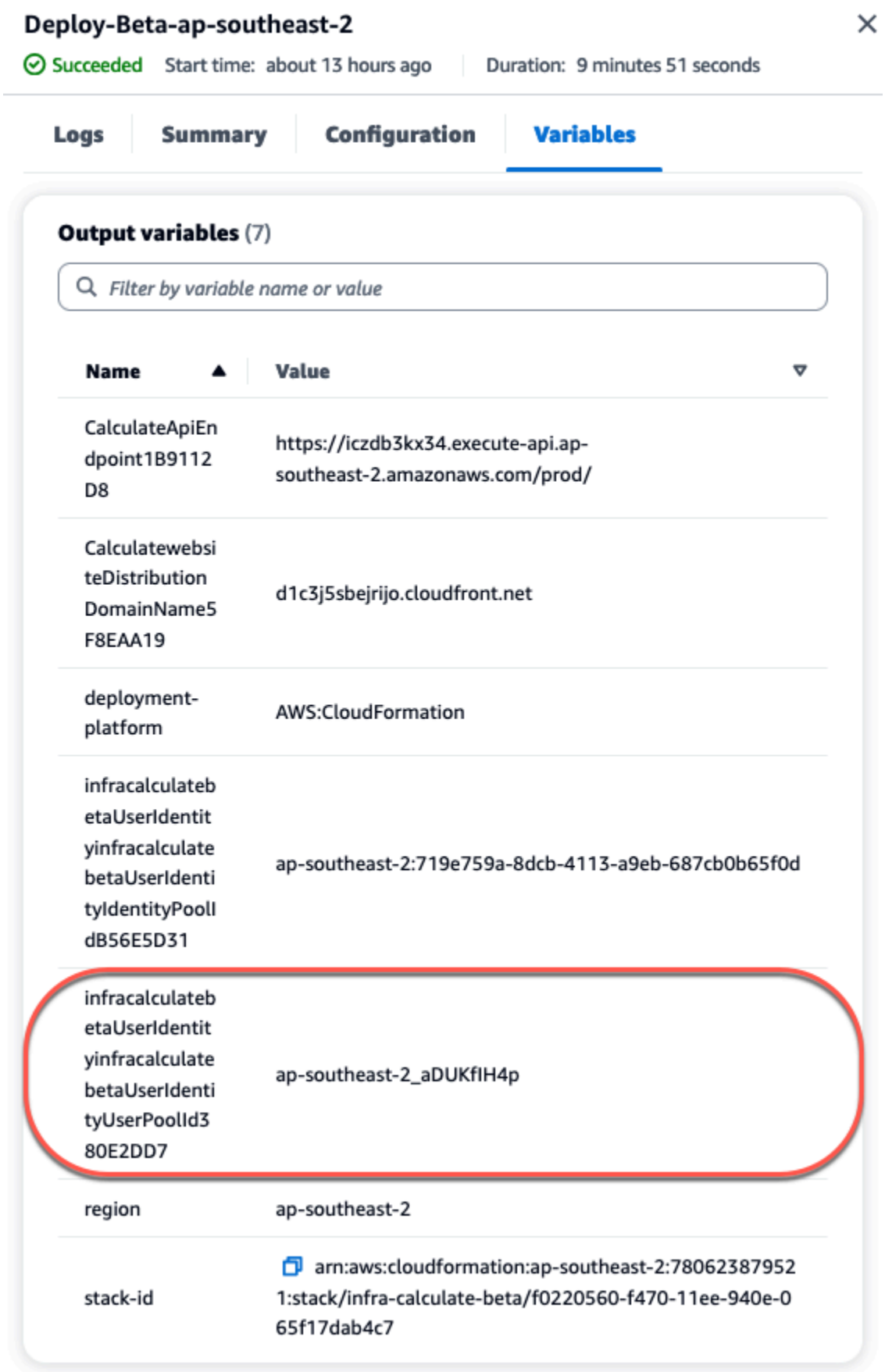
4. View the deployed website by copying and pasting the link found in the **Variables** table (for example, ***MyPDKApi*websiteDistributionDomainNameXXXXX**) into a new browser window.


Deploy-Beta-ap-southeast-2
Succeeded
Start time: about 13 hours ago
Duration: 9 minutes 51 seconds
Logs
Summary
Configuration
Variables
Output variables (7)
Filter by variable name or value
Name
Value
CalculateApiEndpoint1B9112D8
https://iczdb3kx34.execute-api.ap-southeast-2.amazonaws.com/prod/
CalculatewebsiteDistributionDomainName5F8EAA19
d1c3j5sbejrijo.cloudfront.net
deployment-platform
AWS:CloudFormation
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityIdentityPooldB56E5D31
ap-southeast-2:719e759a-8dcb-4113-a9eb-687cb0b65f0d
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityUserPoolId380E2DD7
ap-southeast-2_aDUKfIH4p
region
ap-southeast-2
stack-id
arn:aws:cloudformation:ap-southeast-2:780623879521:stack/infra-calculate-beta/f0220560-f470-11ee-940e-065f17dab4c7

You need an Amazon Cognito account to log in to your website. By default, the user pool isn't set up to allow self-registration.

a. Navigate to the AWS Cognito console.

b. From the **User pools** table, choose the **User pool name** that matches the user pool created by the **PDK - DevOps** blueprint, which can be found in the **Variables** table (for example, **infra*calculate*betaUserIdentityinfra*calculate*betaUserIdentityIdentityPoolIdXXXXX**. For more information, see Getting started with user pools.


Deploy-Beta-ap-southeast-2
Succeeded
Start time: about 13 hours ago
Duration: 9 minutes 51 seconds
Logs
Summary
Configuration
Variables
Output variables (7)
Filter by variable name or value
Name
Value
CalculateApiEndpoint1B9112D8
https://iczdb3kx34.execute-api.ap-southeast-2.amazonaws.com/prod/
CalculatewebsiteDistributionDomainName5F8EAA19
d1c3j5sbejrijo.cloudfront.net
deployment-platform
AWS:CloudFormation
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityIdentityPoolIdB56E5D31
ap-southeast-2:719e759a-8dcb-4113-a9eb-687cb0b65f0d
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityUserPoolId380E2DD7
ap-southeast-2_aDUKfIH4p
region
ap-southeast-2
stack-id
arn:aws:cloudformation:ap-southeast-2:780623879521:stack/infra-calculate-beta/f0220560-f470-11ee-940e-065f17dab4c7

c. Choose **Create user**.

d. Configure the **User information** parameters:

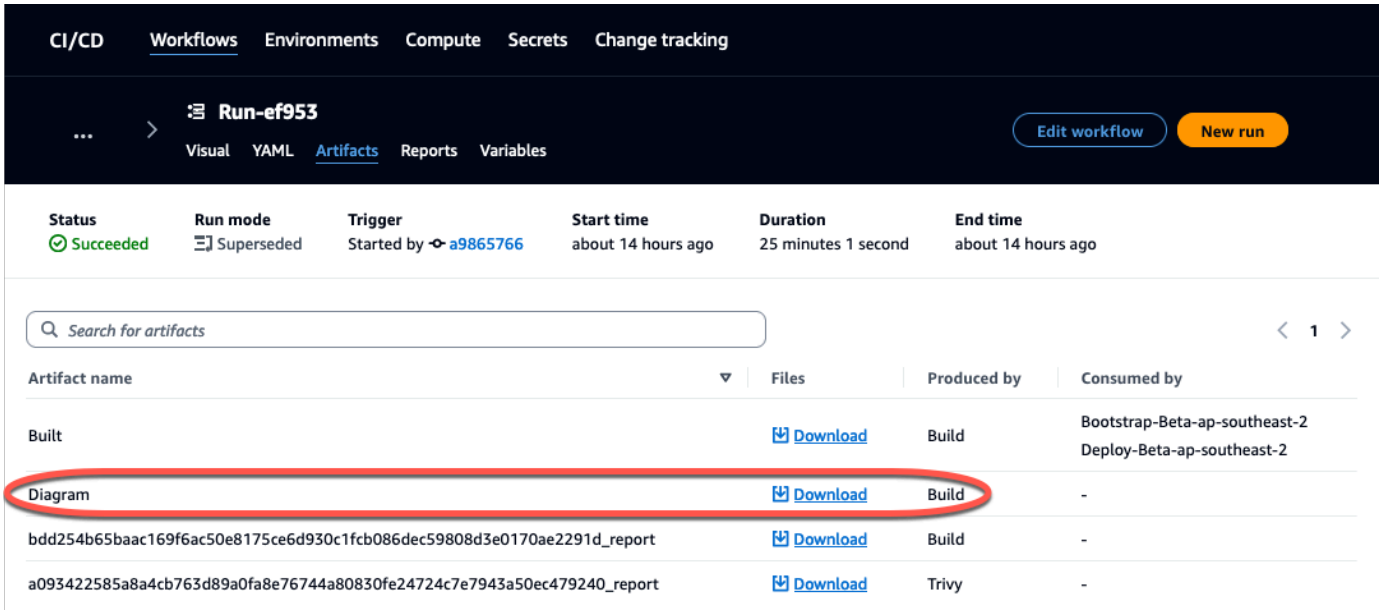
- Under **Invitation message**, choose **Send an email invitation**.
- In the **User name** text input field, enter a username.
- In the **Email address** text input field, enter a username.
- Under **Temporary password**, choose **Generate a password**.

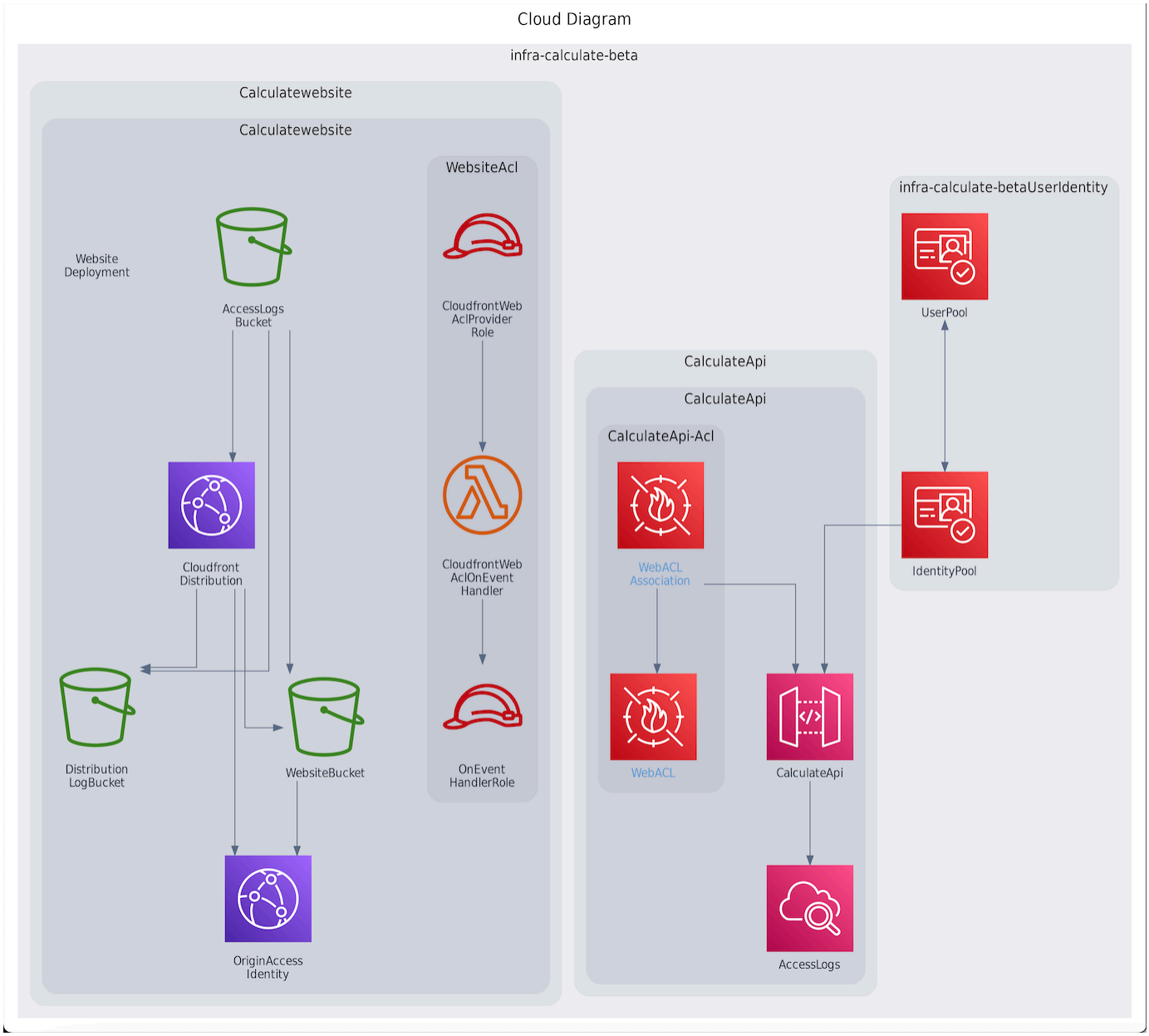
e. Choose **Create user**.

f. Navigate to the email account you entered for the **User information** parameters, open an email with a temporary password. Keep note of the password.

g. Navigate back to the deployed website, enter the username you created and the temporary password you received, and then choose **Sign in**.

5. (Optional) After the workflow run is successfully complete, you can also view the generated diagram. Choose the **Artifacts** tab in CodeCatalyst, choose **Download** for the **Diagram** row, and then open the downloaded file(s).





Collaborate and iterate on the PDK project

After your project is set up, you can make changes to the source code. You can also invite other space members to work on the project. PDK blueprints allow you to build your application iteratively, only adding on what you need, when you need it, while retaining full control of each blueprint's configuration.

Topics

- Step 1: Invite members to your project

- Step 2: Create issues to collaborate and track work
- Step 3: View your source repository
- Step 4: Create a Dev Environment and make code changes
- Step 5: Push and merge code changes

Step 1: Invite members to your project

You can use the console to invite users to your project. You can invite members of your space or add names from outside your space.

To invite users to your project, you must be signed in with the **Project administrator** or **Space administrator** role.

You do not need to invite a user with the **Space administrator** role to your project because they already have implicit access to all projects in the space.

When you invite a user to your project (without assigning the **Space administrator** role), the user will show in the **Project members** table under projects and in the **Project members** table under spaces.

To invite a member to your project from the Project settings tab

1. Navigate to your project.

   > **Tip**
   >
   > You can choose which project to view in the top navigation bar.

2. In the navigation pane, choose **Project settings**.
3. Choose the **Members** tab.
4. In **Project members**, choose **Invite new member**.
5. Type the new member's email address, choose the role for this member, and then choose **Invite**. For more information about roles, see Granting access with user roles.

To invite a member to your project from the Project overview page

1. Navigate to your project.

> **Tip**
>
> You can choose which project to view in the top navigation bar.

2. Choose the **Members +** button.
3. Type the new member's email address, choose the role for this member, and then choose **Invite**. For more information about roles, see Granting access with user roles.

Step 2: Create issues to collaborate and track work

CodeCatalyst helps you track features, tasks, bugs, and any other work involved in your project with issues. You can create issues to track needed work and ideas. By default, when you create an issue it is added to your backlog. You can move issues to a board where you track work in progress. You can also assign an issue to a specific project member. In this step, create an issue to make changes to your PDK project.

To create an issue

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your monorepo project where you want to create an issue.
3. On the project home page, choose **Create issue**. Alternatively, in the navigation pane, choose **Issues**.
4. Choose **Create issue**.

> **Note**
>
> You can also add issues inline when using a grid view.

5. Enter a title for the issue.
6. (Optional) Enter a **Description**. For this issue, enter the following description: `a change in the src/mysfit_data.json file.`. You can use Markdown to add formatting.
7. (Optional) Choose a **Status**, **Priority**, **Estimation** for the issue.
8. (Optional) Add an existing label or create a new label and add it by choosing **+ Add label**.
    a. To add an existing label, choose the label from the list. You can enter a search term in the field to search all labels containing that term in the project.

b. To create a new label and add it, enter the name of the label you want to create in the search field and press enter.

9. (Optional) Add an assignee by choosing **+ Add an assignee**. You can quickly add yourself as the assignee by choosing **+ Add me**.

> **Tip**
>
> You can choose to assign an issue to **Amazon Q** to have Amazon Q try to solve the issue. For more information, see Tutorial: Using CodeCatalyst generative AI features to speed up your development work.
> This functionality requires that generative AI features are enabled for the space. For more information, see Managing generative AI features.

10. (Optional) Add an existing **custom field** or create a new custom field. Issues can have multiple custom fields.

    a. To add an existing custom field, choose the custom field from the list. You can enter a search term in the field to search all custom fields containing that term in the project.

    b. To create a new custom field and add it, enter the name of the custom field you want to create in the search field and press enter. Then choose the type of custom field you want to create and set a value.

11. Choose **Create issue**. A notification appears in the lower right corner: If the issue was created successfully, a confirmation message appears saying the issue was successfully created. If the issue was not created successfully, an error message with the reason for the failure appears. You can then choose **Retry** to edit and retry creating the issue, or choose **Discard** to discard the issue. Both options will dismiss the notification.

> **Note**
>
> You can't link a pull request to an issue when you create it. However, you can edit it after you create it to add links to pull requests.

For more information, see Track and organize work with issues in CodeCatalyst.

Step 3: View your source repository

You can view the source repositories associated with a project in Amazon CodeCatalyst. For source repositories in CodeCatalyst, the overview page for a repository provides a quick overview of information and activity in that repository, including:

- The description of the repository, if any
- The number of branches in the repository
- The number of open pull requests for the repository
- The number of related workflows for the repository
- The files and folders in the default branch, or the branch that you choose
- The title, author, and date of the last commit to the displayed branch
- The contents of the README.md file rendered in Markdown, if any README.md file is included

This page also provides links to the commits, branches, and pull requests for the repository, as well as a quick way to open, view, and edit individual files.

> **Note**
>
> You can't view this information about linked repositories in the CodeCatalyst console. To view information about linked repositories, choose the link in the list of repositories to open that repository in the service that hosts it.

To navigate to the source repositories for a project

1. Navigate to your project, and do one of the following:
   - On the summary page for your project, choose the repository you want from the list, and then choose **View repository**.
   - In the navigation pane, choose **Code**, and then choose **Source repositories**. In **Source repositories**, choose the name of the repository from the list. You can filter the list of repositories by typing part of the repository name in the filter bar.
2. On the home page for the repository, view the contents of the repository and information about the associated resources such as the number of pull requests and workflows. By default, the contents for the default branch are shown. You can change the view by choosing a different branch from the drop-down list.

> **Tip**
>
> You can also quickly navigate to your project's repositories by choosing **See project code** from the project summary page.

Step 4: Create a Dev Environment and make code changes

In this step, create a Dev Environment and make code changes that are then merged into the main branch. While this tutorial walks you through a simple AWS PDK project, you can also follow a more complex example provided in the AWS PDK GitHub repository.

To create a Dev Environment with a new branch

1. In the navigation pane of your monorepo project, do one of the following:
   - Choose **Overview**, and then navigate to the **My Dev Environments** section.
   - Choose **Code**, and then choose **Dev Environments**.
   - Choose **Code**, choose **Source repositories**, and then choose the **monorepo** repository for which you want to create a Dev Environment.
2. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.
3. Choose **Clone a repository**.
4. Choose the repository to clone, choose **Work in new branch**, enter a branch name into the **Branch name** field, and choose a branch off of which to create the new branch from the **Create branch from** drop-down menu.

   > **Note**
   >
   > If you create a Dev Environment from the **Source repositories** page or from a specific source repository, you don't need to choose a repository. The Dev Environment will be created from the source repository you chose from the **Source repositories** page.

5. (Optional) In **Alias - optional**, enter an alias for the Dev Environment.
6. (Optional) Choose the **Dev Environment configuration** edit button to edit the Dev Environment's compute, storage, or timeout configuration.

7. (Optional) In **Amazon Virtual Private Cloud (Amazon VPC) - optional**, select a VPC connection that you'd like to associate with your Dev Environment from the drop-down menu.

   If a default VPC is set for your space, your Dev Environments will run connected to that VPC. You can override this by associating a different VPC connection. Also, note that VPC-connected Dev Environments don't support AWS Toolkit.

   > **Note**
   >
   > When you create a Dev Environment with a VPC connection, a new network interface is created inside the VPC. CodeCatalyst interacts with this interface using the associated VPC role. Also, make sure that your IPv4 CIDR block is **not** configured to the `172.16.0.0/12` IP address range.

8. Choose **Create**. While your Dev Environment is being created, the Dev Environment status column will display **Starting**, and the status column will display **Running** once the Dev Environment has been created.

After your Dev Environment is running, you can work with your generated sample application in CodeCatalyst by making changes to the code with pull requests that are automatically built and deployed to resources in the connected AWS account when the pull request is merged. The monorepo vends a devfile so all required global dependecies and runtimes are present automatically.

To change code in your project

1. In a working terminal of your Dev Environment, navigate to your monorepo project, and then install your project dependencies by running the following command:

   ```
   npx projen install
   ```

2. Navigate to the `packages/apis/mypdkapi/model/src/main/smithy/operations/say-hello.smithy`, which defines an example API operation. In this tutorial, you'll build a simple `Calculate` operation that adds two numbers together. Make a change to the code to define this operation, including its input and output.

   Example:

   ```
   $version: "2"
   ```

```
namespace com.aws

@http(method: "POST", uri: "/calculate")
@handler(language: "typescript")
operation Calculate {
    input := {
        @required
        numberA: Integer
        @required
        numberB: Integer
    }
    output := {
        @required
        result: Integer
    }
}
```

The `@handler` trait tells Type Safe API that you'll implement this operation as an AWS Lambda handler written in TypeScript. The Type Safe API will generate a stub for this operation for you to implement in TypeScript. The `@required` trait is added, which means it'll be enforced at runtime by the API gateway that gets deployed. For more information, see the Smithy documentation.

3. Rename the `/say-hello.smithy` filename with one that aligns with your code changes (for example, `calculate.smithy`).
4. Navigate to the `packages/apis/mypdkapi/model/src/main/smithy/main.smithy`, and make a change to the code to hook up the operation. You can expose the `Calculate` operation defined in the `/calculate.smithy` by listing it in the `operations` field of this file.

   Example:

```
$version: "2"
namespace com.aws

use aws.protocols#restJson1

/// A sample smithy api
@restJson1
service MyPDKApi {
    version: "1.0"
    operations: [Calculate]
```

```
    errors: [
      BadRequestError
      NotAuthorizedError
      InternalFailureError
    ]
}
```

5. Build the changes by running the following command:

```
npx projen build
```

> **Note**
>
> Optionally, you can pass in `--parallel X` flag, which will distribute the build across `X` cores.

Since the `@handler` trait was added, the following files are generated after the build is complete:

- `/packages/apis/mypdkapi/handlers/typescript/src/calculate.ts`
- `/packages/apis/mypdkapi/handlers/typescript/test/calculate.test.ts`

6. Navigate to the `packages/apis/mypdkapi/handlers/typescript/src/calculate.ts`, and make changes to the code. This file is the server handler that is invoked for the API.

```
import {
  calculateHandler,
  CalculateChainedHandlerFunction,
  INTERCEPTORS,
  Response,
  LoggingInterceptor,
} from 'mypdkapi-typescript-runtime';

/**
 * Type-safe handler for the Calculate operation
 */
export const calculate: CalculateChainedHandlerFunction = async (request) => {
  LoggingInterceptor.getLogger(request).info('Start Calculate Operation');

  const { input } = request;
```

```
  return Response.success({
    result: input.body.numberA + input.body.numberB,
  });
};

/**
 * Entry point for the AWS Lambda handler for the Calculate operation.
 * The calculateHandler method wraps the type-safe handler and manages marshalling
 inputs and outputs
 */
export const handler = calculateHandler(...INTERCEPTORS, calculate);
```

7. Navigate to the /packages/apis/*mypdkapi*/handlers/typescript/test/*calculate.test.ts* file, and make changes to the code to update the unit tests.

   Example:

```
import {
  CalculateChainedRequestInput,
  CalculateResponseContent,
} from 'mypdkapi-typescript-runtime';
import {
  calculate,
} from '../src/calculate';

// Common request arguments
const requestArguments = {
  chain: undefined as never,
  event: {} as any,
  context: {} as any,
  interceptorContext: {
    logger: {
      info: jest.fn(),
    },
  },
} satisfies Omit<CalculateChainedRequestInput, 'input'>;

describe('Calculate', () => {

  it('should return correct sum', async () => {
    const response = await calculate({
      ...requestArguments,
```

```
      input: {
        requestParameters: {},
        body: {
          numberA: 1,
          numberB: 2
        }
      },
    });

    expect(response.statusCode).toBe(200);
    expect((response.body as CalculateResponseContent).result).toEqual(3);
  });

});
```

8. Navigate to the `/packages/infra/main/src/constructs/apis/mypdkapi.ts` file, and make changes to the code to add an integration for the `Calculate` operation in your CDK infrastructure. The API construct has an integrations property, where you can pass in the implementation you added previously. Since you're using the `@handler` trait in your Smithy model for the `Calculate` operation, you can use the generated `CalculateFunction` CDK construct, which is preconfigured, to point to your handler implementation.

   Example:

```
import { UserIdentity } from "@aws/pdk/identity";
import { Authorizers, Integrations } from "@aws/pdk/type-safe-api";
import { Stack } from "aws-cdk-lib";
import { Cors } from "aws-cdk-lib/aws-apigateway";
import {
  AccountPrincipal,
  AnyPrincipal,
  Effect,
  PolicyDocument,
  PolicyStatement,
} from "aws-cdk-lib/aws-iam";
import { Construct } from "constructs";
import { Api, CalculateFunction } from "calculateapi-typescript-infra";

/**
 * Api construct props.
 */
export interface CalculateApiProps {
  /**
```

```
   * Instance of the UserIdentity.
   */
  readonly userIdentity: UserIdentity;
}

/**
 * Infrastructure construct to deploy a Type Safe API.
 */
export class CalculateApi extends Construct {
  /**
   * API instance
   */
  public readonly api: Api;

  constructor(scope: Construct, id: string, props?: CalculateApiProps) {
    super(scope, id);

    this.api = new Api(this, id, {
      defaultAuthorizer: Authorizers.iam(),
      corsOptions: {
        allowOrigins: Cors.ALL_ORIGINS,
        allowMethods: Cors.ALL_METHODS,
      },
      integrations: {
        calculate: {
          integration: Integrations.lambda(new CalculateFunction(this,
 "CalculateFunction"))
        }
      },
      policy: new PolicyDocument({
        statements: [
          // Here we grant any AWS credentials from the account that the prototype
 is deployed in to call the api.
          // Machine to machine fine-grained access can be defined here using more
 specific principals (eg roles or
          // users) and resources (ie which api paths may be invoked by which
 principal) if required.
          // If doing so, the cognito identity pool authenticated role must still
 be granted access for cognito users to
          // still be granted access to the API.
          new PolicyStatement({
            effect: Effect.ALLOW,
            principals: [new AccountPrincipal(Stack.of(this).account)],
            actions: ["execute-api:Invoke"],
```

```
            resources: ["execute-api:/*"],
          }),
          // Open up OPTIONS to allow browsers to make unauthenticated preflight
 requests
          new PolicyStatement({
            effect: Effect.ALLOW,
            principals: [new AnyPrincipal()],
            actions: ["execute-api:Invoke"],
            resources: ["execute-api:/*/OPTIONS/*"],
          }),
        ],
      }),
    });

    // Grant authenticated users access to invoke the api
    props?.userIdentity.identityPool.authenticatedRole.addToPrincipalPolicy(
      new PolicyStatement({
        effect: Effect.ALLOW,
        actions: ["execute-api:Invoke"],
        resources: [this.api.api.arnForExecuteApi("*", "/*", "*")],
      }),
    );
  }
}
```

9. Build the changes by running the following command:

```
npx projen build
```

After your project is finished building, you can view the updated generated diagram, which can be found in the `/packages/infra/main/cdk.out/cdkgraph/diagram.png`. The diagram shows how the function is added and hooked up to the API created. As the CDK code is modified, this diagram is also updated.

You can now deploy your changes by pushing and merging them to your repository's main branch.

Step 5: Push and merge code changes

Commit and push your code changes, which can then be merged into your source repository's main branch.

To push changes to your feature branch

- Commit and push changes to your feature branch by running the following commands:

```
git add .
```

```
git commit -m "my commit message"
```

```
git push
```

Pushing changes triggers a new workflow run for your feature branch, which you can view in the CodeCatalyst console. You can then create a pull request to merge the changes to your source repository's main branch. Merging the feature branch to your main branch triggers the release workflow. You can also link the pull request to your issue.

To create a pull request and link it to your issue

1. In your monorepo project, do one of the following:
   - In the navigation pane, choose **Code**, choose **Pull requests**, and then choose **Create pull request**.
   - On the repository home page, choose **More**, and then choose **Create pull request**.
   - On the project page, choose **Create pull request**.
2. In **Source repository**, make sure that the specified source repository is the one that contains the committed code. This option only appears if you didn't create the pull request from the repository's main page.
3. In **Destination branch**, choose the **main** branch to merge the code into after it is reviewed.
4. In **Source branch**, choose the feature branch that contains the committed code.
5. In **Pull request title**, enter a title that helps other users understand what needs to be reviewed and why.
6. (Optional) In **Pull request description**, provide information such as a link to issues or a description of your changes.

> **Tip**
>
> You can choose **Write description for me** to have CodeCatalyst automatically generate a description of the changes contained in the pull request. You can make changes to the automatically generated description after you add it to the pull request.
> This functionality requires that generative AI features are enabled for the space. For more information, see Managing generative AI features in Amazon CodeCatalyst.

7. In **Issues**, choose **Link issues**, and then choose the issue you created in Step 2: Create issues to collaborate and track work. To unlink an issue, choose the unlink icon.
8. (Optional) In **Required reviewers**, choose Add required reviewers. Choose from the list of project members to add them. Required reviewers must approve the changes before the pull request can be merged into the destination branch.

   > **Note**
   >
   > You can't add a reviewer as both a required reviewer and an optional reviewer. You can't add yourself as a reviewer.

9. (Optional) In **Optional reviewers**, choose **Add optional reviewers**. Choose from the list of project members to add them. Optional reviewers do not have to approve the changes as a requirement before the pull request can be merged into the destination branch.
10. Your pull request must reviewed and merged into the main branch by reviewers or yourself. For more information, see Merging a pull request.

    When your changes are merged into the main branch of your source repository, a new workflow is automatically triggered.
11. After the merge is complete, you can move your issue to **Done**.

    a. In the navigation pane, choose **Issues**.

    b. Choose the issue created in Step 2: Create issues to collaborate and track work, choose the **Status** dropdown, and then choose **Done**.

The release workflow deploys your application after a successful run, so you can view the changes.

To confirm the release workflow and view your website

1. In the navigation pane of your monorepo project, choose **CI/CD**, and then choose **Workflows**.
2. For the **release** workflow, choose the latest workflow run to view the details. For more information, see Viewing the status and details of a single run.
3. After the workflow run is successfully complete, choose the last action in the workflow (**Deploy-Beta-ap-souteast-2**), and then choose **Variables**.
4. View the deployed website by copying and pasting the link from the ***MyPDKApi*****websiteDistributionDomainNameXXXXX** row into a new browser window.
5. Enter the username and password you created in Step 6: Confirm release workflow and view your website, and then choose **Sign in**.
6. (Optional) Test the changes in your application.

    a. Choose the **POST** dropdown menu.

    b. Enter two values for `numberA` and `number B`, and then choose **Execute**.

    c. Confirm the results in **Response body**.

Over time, the PDK blueprints' catalog versions can change. You can change your project's blueprints to the catalog versions to stay up to date with the latest changes. You can view the code changes and affected environments before changing your project's blueprint versions. For more information, see Changing blueprint versions in a project.

Organize resources with spaces in CodeCatalyst

You create a space that represents you, your company, department, or group, and provides a place where your development teams can manage projects. You must create a space to add projects, members, and the associated cloud resources you create in Amazon CodeCatalyst.

> **Note**
>
> Space names must be unique across CodeCatalyst. You cannot reuse names of deleted spaces.

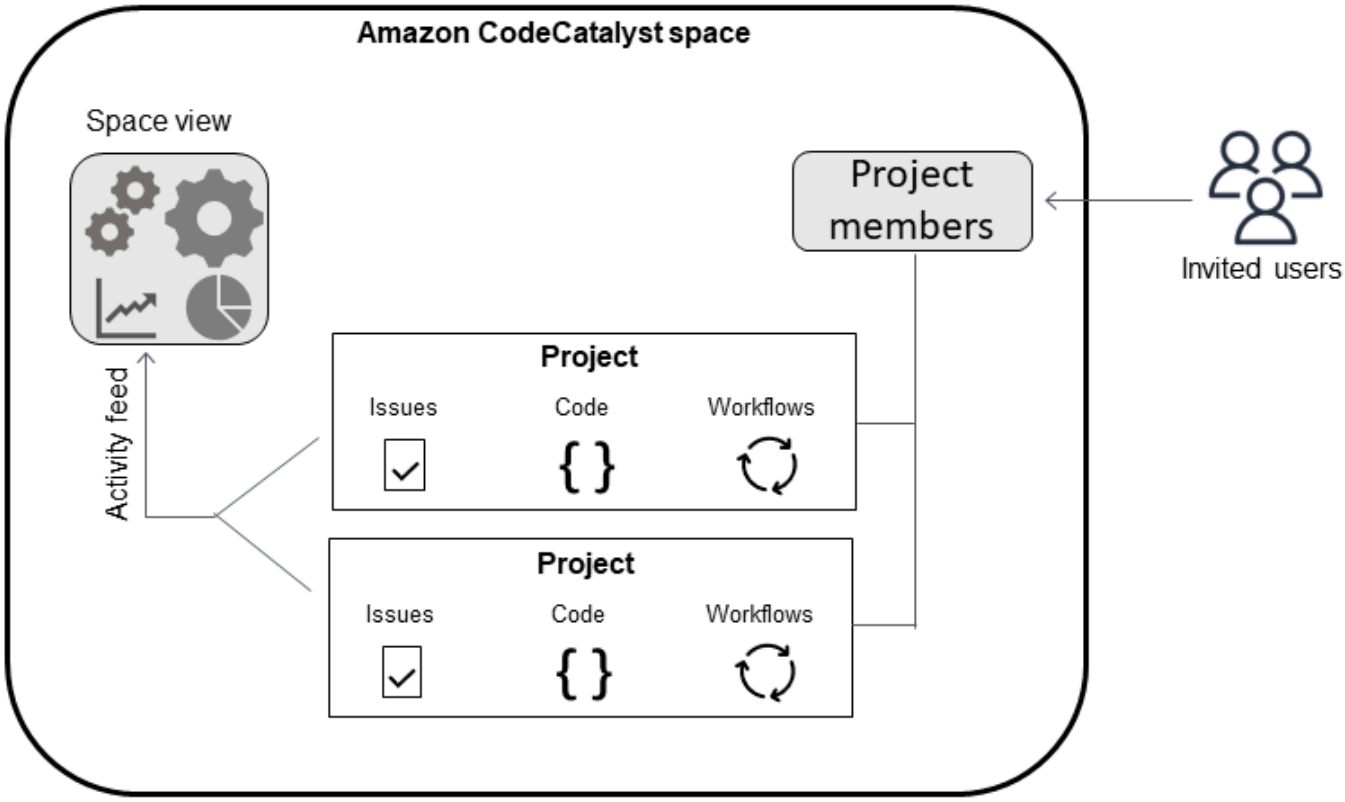
When you create a space, you are automatically assigned the **Space administrator** role. You can add this role to other users in the space.

With the **Space administrator** role, you can manage the space as follows:

- Add other space administrators to the space
- Change member roles and permissions
- Edit or delete the space
- Create projects and invite members to the project
- View a list of all projects in the space
- View the activity feed for all projects in the space

When you create a space, you are automatically added to the space with two roles: the **Space administrator** role, and the **Project administrator** role for the project you created as part of creating the space. Additional users are added as members to the space automatically when they accept invitations to projects. This membership in the space does not grant any permissions in the space. What users can do in a space is determined by the role the user has in a specific project.

For more information about roles, see Granting access with user roles.



The following are additional considerations for added accounts:

- AWS accounts added to a CodeCatalyst space can be used in any project in that space.
- While each environment can support multiple AWS accounts, you can only use one account per environment in an action.
- Billing is configured at the space level. Multiple accounts can be configured for billing, but only one can be active in a CodeCatalyst space. An AWS account can be used as a billing account for more than one space in CodeCatalyst. The AWS account that is specified as the billing account for your CodeCatalyst space has different quotas from other account connections for a space. For more information, see Quotas for CodeCatalyst.
- After you create a connection, you must add AWS IAM roles to your connection if your workflow must access those IAM roles with your CodeCatalyst environment. For more information about how environments are used, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.

Topics

- Creating a space

- Editing a space
- Deleting a space
- Monitoring activity for users and resources in a space
- Allowing access to AWS resources with connected AWS accounts
- Configuring IAM roles for connected accounts
- Granting users space permissions
- Allowing space access using teams
- Allowing space access for machine resources
- Administering Dev Environments for a space
- Quotas for spaces

Creating a space

When you first sign up in Amazon CodeCatalyst with your AWS Builder ID, you are required to create a space. For more information, see Set up and sign in to CodeCatalyst. You can choose to create additional spaces to meet your business needs.

> **Note**
>
> Space names must be unique across CodeCatalyst. You cannot reuse names of deleted spaces.

The information in this guide is provided for creating spaces in CodeCatalyst that support AWS Builder ID users. The steps to set up and administer a space that supports identity federation are provided in the *CodeCatalyst Administrator Guide*. To work with spaces that are set up for identity federation, see Setup and administration for CodeCatalyst spaces in the *Amazon CodeCatalyst Administrator Guide*.

To create additional spaces that support AWS Builder ID users, you must be assigned the Space administrator role.

> **Note**
>
> When you create an additional space, you are not prompted to create a project. To learn how to create projects in a space, see Creating a project.

To create another space

1. In the AWS Management Console, make sure you are signed in with the same AWS account that you want to associate with your CodeCatalyst space.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your space.

   > **Tip**
   >
   > If you belong to more than one space, choose a space in the top navigation bar.

4. Choose **Create space**.
5. On the **Create a space** page, in **Space name**, enter a name for the space. You cannot change this later.

   > **Note**
   >
   > Space names must be unique across CodeCatalyst. You cannot reuse names of deleted spaces.

6. In **AWS Region**, choose the Region where you want to store your space and project data. You cannot change this later.
7. In **AWS account ID**, enter the twelve-digit ID for the account you want to connect to your space.

   In **AWS account verification token**, copy the generated token ID. The token is automatically copied for you, but you might want to store it while you approve the AWS connection request.
8. Choose **Verify in AWS**.
9. The **Verify Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst Spaces** page. You might need to sign in to access the page.

In the AWS Management Console, make sure to choose the same AWS Region where you want to create your space.

To directly access the page, sign in to the Amazon CodeCatalyst Spaces in the AWS Management Console at https://console.aws.amazon.com/codecatalyst/home/.

The verification token is automatically entered in **Verification token**. A success banner shows a message that the token is a valid token.

10. Choose **Verify space**.

    An **Account verified** success message displays to show that the account has been added to the space.

11. Remain on the **Verify Amazon CodeCatalyst space** page. Choose the following link: **To add IAM roles for this space, view space details.**

    The **CodeCatalyst space details** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst Spaces** page. You might need to log in to access the page.

12. Under **IAM roles available to CodeCatalyst**, choose **Add IAM role**.

    The **Add IAM roles available to CodeCatalyst** page displays.

13. Choose **Create CodeCatalyst development administrator role in IAM**. This option creates a service role that contains the permissions policy and trust policy for the development role.

    The developer role is an AWS IAM role that enables your CodeCatalyst workflows to access AWS resources such as Amazon S3, Lambda, and AWS CloudFormation. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* with a unique identifier appended. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role.

14. Choose **Create development role**.

15. On the connection page, under **IAM roles available to CodeCatalyst**, view the developer role in the list of IAM roles added to your account.

16. Choose **Go to Amazon CodeCatalyst**.

17. On the creation page in CodeCatalyst, choose **Create space**.

Editing a space

You can change the description of a space to help users better understand what it's for.

You must have the **Space administrator** role to edit space details.

The information in this guide is provided for editing spaces in CodeCatalyst that support AWS Builder ID users. To learn more about the steps to set up and administer a space that supports identity federation, see Setup and administration for CodeCatalyst spaces in the *Amazon CodeCatalyst Administrator Guide*.

To edit a space description

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space.

   > **Tip**
   >
   > If you belong to more than one space, choose a space in the top navigation bar.

3. On the **Space settings** tab, choose **Edit**. Make the changes you want to the space description, and then choose **Save**.

Deleting a space

You can delete a space to remove access to all of the space's resources. You must have the **Space administrator** role to delete a space.

> **Note**
>
> You cannot undo a space deletion.

After you have deleted a space, all space members will be unable to access space resources. Billing for space resources will also stop, and any workflows that are prompted by third-party source repositories will be stopped.

> **Note**
>
> Space names must be unique across CodeCatalyst. You cannot reuse names of deleted spaces.

The information in this guide is provided for deleting spaces in CodeCatalyst that support AWS Builder ID users. To learn more about the steps to set up and administer a space that supports identity federation, see Setup and administration for CodeCatalyst spaces in the *Amazon CodeCatalyst Administrator Guide*.

To delete a space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space.

   > **Tip**
   >
   > If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Delete**.
4. Type **`delete`** to confirm the deletion.
5. Choose **Delete**.

   > **Note**
   >
   > If you belong to more than one space, you're redirected to the space overview page. If you belong to one space, you're redirected to the space creation page.

Monitoring activity for users and resources in a space

To see recently created projects and status updates, you can use the CodeCatalyst console to view an activity feed that shows updates for space resources.

In the activity feed, you can view metrics such as failed workflow runs and created projects.

To view activity in your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Activity**.
4. View the information in **Activity**.
5. To filter by activity, choose the selector on the upper right.
6. To view all activity in your space, choose **Any activity type**.

Allowing access to AWS resources with connected AWS accounts

You can use resources from your AWS accounts in Amazon CodeCatalyst spaces. To do so, you must set up a connection between the AWS accounts and your space in CodeCatalyst. Creating a connection like this means that projects and workflows within your CodeCatalyst space can interact with resources in your AWS accounts. You must create one connection for each AWS account you want to use with your CodeCatalyst space.

After you create a connection, you can choose to associate AWS IAM roles with it.

Topics

You can set up CodeCatalyst to use authorized AWS accounts by adding the accounts to your space. By adding AWS accounts to your CodeCatalyst space, you can give your project workflows access to AWS account resources and your billing configuration.

Adding an AWS account creates a connection that authorizes CodeCatalyst to use this account. You can use added AWS accounts to do the following:

- Set up billing for a CodeCatalyst space. See Managing billing in the Amazon CodeCatalyst Administrator Guide. The AWS account that is specified as the billing account for your CodeCatalyst space has different quotas from other account connections for a space. For more information, see Quotas for CodeCatalyst.
- Allow CodeCatalyst to assume IAM roles to access AWS resources and deploy to AWS services in the account. See Configuring IAM roles for connected accounts.

Account connections are created by completing authorization with the AWS account. After the connection is created, you further configure the connection for workflows and projects to use by adding IAM roles.

For the steps to configure account connections in the AWS Management Console page for CodeCatalyst as the administrator for the AWS account and the space, see Managing connected accounts in the *CodeCatalyst Administrator Guide*.

Adding an AWS account to a space

You use the CodeCatalyst console and the AWS Management Console to connect your space to an AWS account.

Before adding an AWS account to a space in CodeCatalyst, complete the following prerequisites:

- Create an AWS account and acquire permissions to create AWS IAM roles in the account you want to connect.
- Create the IAM role or roles you want to associate with your account connection, including the IAM policies with permissions for the roles.
- Acquire the **Space administrator** role in the CodeCatalyst space where you want to create the connection.

Topics

Step 1: Creating a connection request

Creating a connection request in the CodeCatalyst console generates a connection token that you can use to complete authorization.

You must have the **Space administrator** or **Power user** role in the CodeCatalyst space where you want to create the connection. You must also have administrative permissions for the AWS account you want to add.

To create a connection

1. In the AWS Management Console, make sure you are logged in with the same account that you want to create a connection with.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
4. Choose **Add an AWS account**.
5. On the **Associate AWS account with Amazon CodeCatalyst** page, in **AWS account ID**, enter the twelve-digit ID for the account you want to connect to your space. For information about finding your AWS account ID, see Your AWS account ID and its alias.
6. In **Amazon CodeCatalyst display name**, enter a reference name for the account.
7. (Optional) In **Connection description**, enter a description for the account that will help you choose the projects where the account and role or roles will apply.
8. Choose **Associate AWS account**.
9. The page returns to the **AWS account details** page where a success banner displays.

Step 2: Accepting an account connection request

After you submit a request in the CodeCatalyst console to connect to your AWS account, you work with your AWS administrator to accept the connection request by submitting it with the provided connection token.

Make sure you have administrator permissions for your account, and you're signed in to the AWS Management Console with the same AWS account for which you're creating the connection.

To approve a connection request (console)

1. In the AWS Management Console, make sure you are logged in with the same account that you want to create a connection with.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
4. On the **AWS account details** page, choose **Complete setup in the AWS Management Console**.
5. The **Verify Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst Spaces** page. You might need to log in to access the page.

   To directly access the page, sign in to the Amazon CodeCatalyst Spaces in the AWS Management Console at https://console.aws.amazon.com/codecatalyst/home/.

   The verification token is automatically entered in **Verification token**. A success message shows a message that the token is a valid token.
6. (Optional) Under **Authorized paid tiers**, choose **Authorize paid tiers (Standard, Enterprise)** to turn on the paid tiers for your billing account.

   > **Note**
   >
   > This does not upgrade the billing tier to a paid tier. However, this configures the AWS account so that you can change the billing tier for your space at any time in CodeCatalyst. You can turn on the paid tiers at any time. Without making this change, the space is only able to use the Free tier.
7. Choose **Verify space**.

   An **Account verified** success message displays to show that the account has been added to the space.

Step 3: Review an approved connection

After getting a connection approved, you can view the connection in the console, along with the IAM roles you added to it.

To review an approved connection

1. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
2. The account connection is listed with the date it was created.
3. Choose the account display name. The **AWS account details** page displays.

Step 4: Add IAM roles to your connection

If you're using an IAM role configured for a CodeCatalyst deploy action, add the role to your deployment environment. For more information, see Adding IAM roles to account connections.

Next steps: Create additional IAM roles for your account connection

After you create a connection, you can create additional IAM roles to add to it. The IAM roles that you add are dependent on your workflows. For example, a CodeCatalyst build action requires the CodeCatalyst build role. .

To connect your account, you will need the Amazon Resource Name (ARN) for the roles you created. Copy the ARN for your role or roles as detailed here. For more information about working with ARNs for IAM roles, see Amazon Resource Name (ARN).

To access your IAM role ARN

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. In the search box, enter the name of the role you want to add.
4. Choose the role from the list.

   The role's **Summary** page appears.
5. At the top, copy the **Role ARN** value.

Adding IAM roles to account connections

Part of creating your account connection includes adding the IAM role or roles you want to use with projects in your CodeCatalyst space.

> **Note**
>
> To use IAM roles with an account connection, make sure that the trust policy is updated to use the CodeCatalyst service principal.

Add IAM roles to an account connection (console)

1. In the AWS Management Console, make sure you are logged in with the same account that you want to manage.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
4. Choose the **Amazon CodeCatalyst display name** of your account connection, and then choose **Manage roles from AWS Management Console**.

   The **Add IAM role to Amazon CodeCatalyst space** page displays.
5. Do one of the following:

   - To create a service role that contains the permissions policy and trust policy for the developer role, choose **Create CodeCatalyst development administrator role in IAM**. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* with a unique identifier appended. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role.

     Choose **Create development role**.
   - To add a role that you have already created in IAM, choose **Add an existing IAM role**. In **Select existing IAM role**, choose the role from the drop-down list.

     Choose **Add role**.

   The page opens in the AWS Management Console. You might need to log in to access the page.
6. In the **Amazon CodeCatalyst spaces** page navigation pane, choose **Spaces**.

   To directly access the page, sign in to the Amazon CodeCatalyst Spaces in the AWS Management Console at https://console.aws.amazon.com/codecatalyst/home/.
7. Choose the account added for your CodeCatalyst space. The connection page is shown.

8. On the connection page, under **IAM roles available to CodeCatalyst**, view the list of IAM roles added to your account. Choose **Associate IAM role to CodeCatalyst**.
9. On the **Associate an IAM role** pop-up, in **Role ARN**, enter the Amazon Resource Name (ARN) of the IAM role you want to associate with your CodeCatalyst space.

   Under **Purpose**, choose a role purpose that describes how you want to use the role in your account connection. Specify `RUNNER` for roles that you use to run actions in workflows. Specify `SERVICE` for roles that you use to access another service.

   You can specify more than one purpose.

   > **Note**
   >
   > Choosing a purpose for the role ARN is required.

10. Choose **Associate an IAM role**. Repeat these steps for additional IAM roles.

Adding the account connection and IAM roles to your deploy environment

To access AWS resources, such as Amazon ECS or AWS Lambda resources for deployments, CodeCatalyst build and deploy actions require IAM roles with permissions to access those resources. With the **Space administrator** or **Power user** role, you can connect your CodeCatalyst account to the AWS account where your resources are created. You then add the IAM role to your account connection. For deploy actions, you must then add the IAM role to a CodeCatalyst environment.

You must add the IAM roles that you want to use with deployment environments in your projects. Adding the roles to the account connection does not add the roles and the connection to the project deploy environments. To add your account connection and IAM roles to your deploy environment, make sure that the account connection and roles are created as detailed in Step 4: Add IAM roles to your connection.

Then, use the **Environments** page in the CodeCatalyst console to add your account connection and IAM role to a deploy environment in a project.

> **Note**
>
> You only add an IAM role to an environment if the IAM role is used for a CodeCatalyst action that requires an IAM role. All workflow actions that require IAM roles, including build actions, must use a CodeCatalyst environment.

To add your account connection and IAM roles to your deploy environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project with the deployment environment where you want to add the account connection and IAM roles.
3. Expand **CI/CD**, and then choose **Environments**.
4. Choose your environment, and then the additional tabs display.
5. Choose the **AWS account connections** tab. Under **Connection name**, the accounts that have been added to the environment, if any, are listed.
6. Choose **Associate AWS account**. The **Associate AWS account with <environment_name>** page displays.
7. Under **Connection**, choose the name of the account connection with the IAM roles that you want to add. Choose **Associate**.

Viewing account connections

You can view a list of your connections and view details about each connection.

You must have the **Space administrator** or **Power user** role to manage connections for your space.

To view all connections for a CodeCatalyst space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space with the account connection that you want to view.
3. Choose the **AWS accounts** tab.
4. Under **AWS accounts**, view the list of account connections for the space, including the account ID and status for each connection.

To view account connection details

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
3. In **Amazon CodeCatalyst display name**, choose the connection name. On the **Details** page, view the list of IAM roles associated with the connection along with other details.

Deleting account connections (in CodeCatalyst)

You can delete an account connection that you no longer need. For this procedure, you will use CodeCatalyst to delete an account connection that you have previously added to your space. This deletes the account connection from your space, provided that the account is not the billing account for the space.

> **Important**
>
> After an account connection is deleted, you cannot reconnect it. You must create a new account connection and then associate IAM roles and environments, or set up billing, as needed.

A billing account must be designated for your CodeCatalyst space, even if usage for the space will not exceed the Free tier. Before you can remove a space for an account that is a designated billing account, you will need to add another account for your space. See Managing billing in the Amazon CodeCatalyst Administrator Guide.

> **Important**
>
> While you can use these steps to remove an account, this is not recommended. The account might also be set up to support workflows in CodeCatalyst.

To manage account connections for your space, you must have the **Space administrator** or **Power user** role.

An account that has been removed can be added again later, but you must create a new connection between the account and the space. You will need to re-associate any IAM roles to the added account.

To delete an account connection

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
3. Under **Amazon CodeCatalyst display name**, choose the selector next to the account connection that you want to remove.
4. Choose **Remove AWS account**. Confirm the deletion by entering the name in the field, and then choose **Remove**.

   A success banner displays, and the account connection is removed from the list of connections.

Configuring a billing account for a space

A billing account must be designated for your CodeCatalyst space, even if usage for the space will not exceed the Free tier.

To configure a billing account, see Billing in the *CodeCatalyst Administrator Guide*. The AWS account that is specified as the billing account for your CodeCatalyst space has different quotas from other account connections for a space. For more information, see Quotas for CodeCatalyst.

To remove an account that is a designated billing account for your CodeCatalyst space, make sure to first specify a new billing account.

Configuring IAM roles for connected accounts

You create roles in AWS Identity and Access Management (IAM) for the account that you want to add to CodeCatalyst. If you are adding a billing account, you do not need to create roles.

In your AWS account, you must have permissions to create roles for the AWS account you want to add to your space. For more information about IAM roles and policies, including IAM references and example policies, see Identity and Access Management and Amazon CodeCatalyst. For more information about the trust policy and service principals used in CodeCatalyst, see Understanding the CodeCatalyst trust model.

In CodeCatalyst, you must be signed in with the Space administrator role to complete the steps to add accounts (and the roles, if applicable) to your space.

You can add roles to your account connections by using one of the following methods.

- To create a service role that contains the permissions policy and trust policy for the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role, see **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role.
- For an example of creating a role and adding a policy to create a project from a blueprint, see Creating an IAM role and using the CodeCatalyst trust policy.
- For a list of sample role policies to use when creating your IAM roles, see Grant access to project AWS resources with IAM roles.
- For detailed steps to create roles for workflow actions, see the workflow tutorial for that action as follows:
  - Tutorial: Upload artifacts to Amazon S3
  - Tutorial: Deploy a serverless application using AWS CloudFormation
  - Tutorial: Deploy an application to Amazon ECS
  - Tutorial: Lint code using a GitHub Action in a workflow

Topics

CodeCatalystWorkflowDevelopmentRole-*spaceName* role

You create the developer role as a 1-click role in IAM. You must have the **Space administrator** or **Power user** role in the space where you want to add the account. You must also have administrative permissions for the AWS account you want to add.

Before you start the procedure below, you must log in to the AWS Management Console with the same account that you want to add to your CodeCatalyst space. Otherwise, the console will return an unknown account error.

To create and add the CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*

1. Before you start in the CodeCatalyst console, open the AWS Management Console, and then make sure you are logged in with the same AWS account for your space.

2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
4. Choose the link for the AWS account where you want to create the role. The **AWS account details** page displays.
5. Choose **Manage roles from AWS Management Console**.

   The **Add IAM role to Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst spaces** page. You might need to log in to access the page.
6. Choose **Create CodeCatalyst development administrator role in IAM**. This option creates a service role that contains the permissions policy and trust policy for the development role. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role.

   > **Note**
   >
   > This role is only recommended for use with developer accounts and uses the `AdministratorAccess` AWS managed policy, giving it full access to create new policies and resources in this AWS account.
7. Choose **Create development role**.
8. On the connections page, under **IAM roles available to CodeCatalyst**, view the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role in the list of IAM roles added to your account.
9. To return to your space, choose **Go to Amazon CodeCatalyst**.

AWSRoleForCodeCatalystSupport role

You create the support role as a 1-click role in IAM. You must have the **Space administrator** or **Power user** role in the space where you want to add the account. You must also have administrative permissions for the AWS account you want to add.

Before you start the procedure below, you must log in to the AWS Management Console with the same account that you want to add to your CodeCatalyst space. Otherwise, the console will return an unknown account error.

To create and add the CodeCatalyst AWSRoleForCodeCatalystSupport

1. Before you start in the CodeCatalyst console, open the AWS Management Console, and then make sure you are logged in with the same AWS account for your space.
2. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
3. Choose the link for the AWS account where you want to create the role. The **AWS account details** page displays.
4. Choose **Manage roles from AWS Management Console**.

   The **Add IAM role to Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst Spaces** page. You might need to sign in to access the page.
5. Under **CodeCatalyst space details**, choose **Add CodeCatalyst Support role**. This option creates a service role that contains the permissions policy and trust policy for the preview development role. The role will have a name **AWSRoleForCodeCatalystSupport** with a unique identifier appended. For more information about the role and role policy, see Understanding the **AWSRoleForCodeCatalystSupport** service role.
6. On the **Add role for CodeCatalyst Support** page, leave the default selected, and then choose **Create role**.
7. Under **IAM roles available to CodeCatalyst**, view the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role in the list of IAM roles added to your account.
8. To return to your space, choose **Go to Amazon CodeCatalyst**.

Creating an IAM role and using the CodeCatalyst trust policy

IAM roles to be used in CodeCatalyst with AWS account connections must be configured to use the trust policy provided here. Use these steps to create an IAM role and attach a policy that allows you to create projects from blueprints in CodeCatalyst.

As an alternative, you can create a service role that contains the permissions policy and trust policy for the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role. For more information, see Adding IAM roles to account connections.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. Choose **Roles**, and then choose **Create role**.
3. Choose **Custom trust policy**.
4. Under the **Custom trust policy** form, paste the following trust policy.

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
             "Principal": {
                "Service": [
                    "codecatalyst-runner.amazonaws.com",
                    "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/
*"
                }
            }
        }
    ]
```

5. Choose **Next**.
6. Under **Add permissions**, search for and select a custom policy that you have already created in IAM.
7. Choose **Next**.
8. For **Role name**, enter a name for the role, for example: `codecatalyst-project-role`
9. Choose **Create role**.
10. Copy the role Amazon Resource Name (ARN). You'll need to provide this information when adding the role to your account connection or environment.

Granting users space permissions

You can manage members for a space by viewing, adding, removing, or changing roles for users who join the space.

The information in this guide is provided for inviting and managing users in spaces in CodeCatalyst that support AWS Builder ID users. To learn more about the steps to set up and administer a space that supports identity federation, see Setup and administration for CodeCatalyst spaces in the *Amazon CodeCatalyst Administrator Guide*.

Viewing members in a space

You can view the users in your space, including information about their display names, aliases, and the role they have for the space. There are three roles for members in a space:

- **Space administrator** – This role has all permissions in CodeCatalyst, including creating projects. Only assign this role to users who need to administer every aspect of a space, such as accessing all projects in the space.

  You cannot change this role later without removing the user first. For more information, see Space administrator role.
- **Power user** – This role is the second-most powerful role in Amazon CodeCatalyst spaces, but it has no access to projects in a space. It is designed for users who need to be able to create projects in a space and help manage the users and resources for the space. For more information, see Power user role.
- **Limited access** – This role is assigned by default for users who join the space by accepting invitations to projects in the space. Project members are assigned a role in a project. For information about managing project members, see Granting users project permissions.

The **Space administrators** table shows users with the **Space administrator** role. These users are not shown in the **Space members** because they are automatically (implicitly) assigned to all projects in the space and do not have a role in a project.

The **Space members** table shows all members in the space that have a role in a project while not having the **Space administrator** role.

Users are shown based on whether the user has the **Space administrator** role in CodeCatalyst as follows:

- A user with the **Space administrator** role who later accepts a project invitation and role will not show in the **Space members** table under spaces or on the **Project members** table under projects. They will continue to be shown in the **Space administrators** table in both places. In each project,

all users with the **Space administrator** role are shown in the project **Space administrators** table for that project.

- A user who accepts a project invitation to join with a project role is added to the space with the **Limited access** role. If the user's role later changes to the **Space administrator** role, but will also move from the **Space members** table to the **Space administrators** table. Under the project, the user will move from the **Project members** table to the **Space administrators** table.

To view users and roles in your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space.

   > **Tip**
   >
   > If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Members**.

   Users who are members of the space are shown in the **Space members** table.

   > **Tip**
   >
   > If you have the **Space administrator** role, you can view which projects you have been directly invited to. Navigate to **Project settings** for the project, and then choose **My projects**.

   In the **Status** column, the following are valid values:

   - **Invited** – CodeCatalyst sent the invitation but the user has not yet accepted or declined.
   - **Member** – The user accepted the invitation.

Inviting a user directly to a space

You can invite users directly to your CodeCatalyst space. This is useful when you want to invite that user to help you manage the space by assigning them the **Space administrator** or **Power user** role.

Assigning one of those roles to other users can help you distribute the responsibilities of managing the space across more people without having to invite these users to any projects.

> **Note**
>
> You must have the **Space administrator** or **Power user** role to invite members.

The **Space administrators** table shows users with the **Space administrator** role. These users are not shown in the **Space members** table because they are automatically (implicitly) assigned to all projects in the space and do not have a role in a project.

Members who accept a project invitation are added to the space by default. The **Project members** table shows all members in the space that have a role in a project.

For more information about how to accept an invitation and sign in for the first time, see Set up and sign in to CodeCatalyst.

To invite a user to your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space.
3. Choose **Settings**, and then choose **Members**.
4. Choose **Invite**.
5. Enter the email of the person you would like to invite to join your space. In **Role**, choose the role you want to assign that user in the space.
6. Choose **Invite**

Canceling an invitation for a space

If you want to cancel an invitation to join a space that you sent recently, and it has not yet been accepted, you can cancel it.

To manage space invitations, you must have the **Space administrator** or **Power user** role.

To cancel a space member invitation

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. Navigate to your space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Members**.
4. Verify that the member has a status of **Invited**.

> **Note**
>
> You can only cancel an invitation that has not yet been accepted.

5. Choose the option next to the row with the invited member, and then choose **Cancel invitation**.
6. A confirmation window displays. Choose **Cancel invitation** to confirm.

Changing the role for a space member

You can change the assigned role for a member of your space. You must have the **Space administrator** role to change the role of a user in the space.

The **Space administrators** table shows users with the **Space administrator** role. These users are not shown in the **Space members** table because they are automatically (implicitly) assigned to all projects in the space.

To change the role for a user in your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Members**.
4. In the **Space members** table, choose the user whose role you want to change. Choose **Change role**.

Removing a space member

You can remove a member of your space when they do not need to access any of the space resources. You must have the **Space administrator** role to remove a member from a space.

The **Space administrators** table shows users with the **Space administrator** role. These users are not shown in the **Space members** table because they are automatically (implicitly) assigned to all projects in the space and do not have a role in a project. You can only directly remove a member of your space in this table.

To remove a user from the Project members table

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Members**.
4. Choose the user in the **Project members** table. Choose **Remove**.

> **Note**
>
> Removing a member from the space will remove the user from all projects in the space, along with permissions associated with the resources in those projects.

Removing or changing the role for a user with the Space administrator role

You can remove or change the role for a user with the **Space administrator** role for your space.

You must have the **Space administrator** role to remove a user with the **Space administrator** role from a space. Changing the role for a user with the **Space administrator** role essentially removes the user from the **Space administrators** table. If that user does not have a project role in any projects in the space, removing the **Space administrator** role from the user will remove the user from the space.

> **Note**
>
> As a user with the **Space administrator** role, you cannot remove yourself. Contact another user with the **Space administrator** role.

To remove a user with the Space administrator role from the Space members table

> **Note**
>
> For a user who has not been added explicitly to a project, they do not have any project roles (**Project administrator** or **Contributor**). If the **Space administrator** role is the user's only role, then the user is removed from the space entirely.

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space where you want to remove or change the role for a user with the **Space administrator** role.
3. Choose **Settings**, and then choose **Members**.
4. View the invitation status for the list of members, and make sure that the list contains no unauthorized pending invites to the space (a status of **Invited**).

   > **Important**
   >
   > Before removing a user with the **Space administrator** role, you must verify that no pending invites have been initiated.

5. Choose the **Members** tab. In the **Space administrators** table, choose the user, and then choose **Remove**.

   On the **Remove member** dialog box, do one of the following.

   - Choose the option to remove only the user's **Space administrator** role. Choose **Remove**.

> **Important**
>
> If the user does not have any other role assigned, then changing the role from **Space administrator** removes the user from the space.

- Choose the option to remove a user with the **Space administrator** role from the space and all its projects. Choose **Remove**.

6. Refresh the **Members** tab. The user is automatically added to the list of project members in any project where the user had membership through project roles. If the **Space administrator** role was the user's only role, then the user is removed from the space entirely.

Allowing space access using teams

After you create a space, you can add teams. Teams allow you to group users so that they can share permissions and manage projects, issue tracking, roles, and resources in CodeCatalyst.

You must have the **Space administrator** role to manage teams.

Teams are also managed at the project/space level in CodeCatalyst. To learn more about teams in spaces/projects, see Allowing space access using teams.

Topics

Creating a team

A team can have role permissions, such as **Power user**, in a space. A team can also have project permissions, such as **Project administrator**, in a project. Teams can be associated with many

projects with different roles for each project. You can manage teams where the team members are either individual users for an AWS Builder ID space or SSO groups for a space that supports identity federation.

On the members page for space and project users, users can have multiple roles. Users with multiple roles will show an indicator when they have multiple roles, and they will be displayed with the role with the most permissions first.

> **Note**
>
> If your space supports identity federation, you must already have your SSO users or your SSO groups set up in IAM Identity Center.

How you manage team members depends on how you will add and remove users. There are two options for managing team members:

- **Adding users directly** — You add or remove users individually. For example, you add users to a team by choosing either AWS Builder ID users or SSO users that are already set up in IAM Identity Center. When you choose to manage team members by adding AWS Builder ID users or SSO users directly, the option to use **SSO groups** will no longer be available.
- **Use SSO groups** — You manage team members through SSO groups already set up in IAM Identity Center. When you choose to manage team members by using **SSO groups**, the option to add users directly will no longer be available.

You must have the **Space administrator** role to manage teams.

To create a team

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. Choose **Create team**.
4. In **Team name**, enter a descriptive name for your team.

   > **Note**
   >
   > The team name must be unique in your space.

(Optional) In **Team description**, enter a description for your team.

5. Under **Space role**, choose a role from the list of space roles available in CodeCatalyst that you want to assign to the team. The role will be inherited by all members of the team.

   - **Space administrator** - For details, see Space administrator role.
   - **Limited access** - For details, see Limited access role.
   - **Power user** - For details, see Power user role.

6. In **Team membership**, choose one of the following to choose the method for adding members to the team.

   - Choose **Add members directly** to manage users individually. This includes adding AWS Builder ID users for a space or adding SSO users for a space that supports identity federation.
   - Choose **Use SSO Groups** to choose SSO groups that you have already set up in IAM Identity Center.

     In **SSO Groups** , choose the box next to the groups that you want to add. You can add up to five SSO groups.

   > **Note**
   >
   > You cannot change this later. When you choose to manage team members by adding AWS Builder ID users or SSO users directly, the option to use **SSO groups** will no longer be available. When you choose to manage team members by using **SSO groups**, the option to add users directly will no longer be available.

7. Choose **Create**.

   > **Note**
   >
   > When you choose to use SSO groups, note that the users in the SSO group are not pulled upon creation of the team. The users will need to have signed in to CodeCatalyst before they are visible in the list.

Viewing a team

In CodeCatalyst, you can view the projects and roles for your team. On the members page, you can view project roles and a list of users. For SSO group type teams, you will also be able to see a list of SSO groups associated with the team.

To view a team

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. In **Space role**, view the role assigned to the team for this space.
4. On the **Project roles** tab, view the project and project role assigned to the team for each CodeCatalyst project in the space where the team has been added as a member (for an AWS Builder ID space only).
5. On the **Members** tab, view the list of members assigned to the team.
6. On the **SSO Groups** tab, view the list of SSO groups assigned to the team (for a space that supports identity federation only).

Granting space roles for a team

Teams are a way to group users so that you can grant and manage team access to projects in CodeCatalyst. As an example, you can use teams to quickly manage roles and permissions for users by giving a team the ability to manage a space for users.

A team can have role permissions, such as **Power user**, in a space. You can change the space role for a team, but note that all members of the team will inherit those permissions.

You must have the **Space administrator** role to manage teams.

Changing the space role for a team

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. In **Actions**, choose **Change space role**. You can change the space role to one of the following. This changes the role for all members of the team.

   - **Space administrator** - For details, see Space administrator role.

- **Limited access** - For details, see Limited access role.
- **Power user** - For details, see Power user role.

4. Choose **Save**.

Granting project roles for a team at the space level

A team in CodeCatalyst is similar to a user in that the team members can have role permissions, such as **Project administrator**, in a project. A role change will be applied to the team, and all members of the team will inherit those permissions. You can choose one role for each project that will be automatically granted to the team.

You must have the **Space administrator** role to manage teams.

To add or change a project role

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. Choose the **Project roles** tab.
4. To change a role, choose the selector next to the project in this list, and then choose **Change role**. To add a role, choose **Add project role**. In **Project**, choose the project you want to add and in **Role**, choose the role. Choose one of the available project roles:
   - **Project administrator** - For details, see Project administrator role.
   - **Contributor** - For details, see Contributor role.
   - **Reviewer** - For details, see Reviewer role.
   - **Read only** - For details, see Read only role.
5. Choose **Save**.

To remove a project role

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. Choose the **Project roles** tab.
4. Choose the role you want to remove.

> **Important**
>
> Removing a role from a team removes the associated permissions for all users in the team.

5. Choose **Save**.

Adding a user to a team directly

You can add team members to your team. When you add a user, the new user will inherit permissions from all existing roles on the team.

Whether your space is set up for AWS Builder ID user support or identity federation, you can set up your space to add users directly.

> **Note**
>
> When your space is set up to manage team members by using SSO groups, the option to use **Add users directly** is not available. To use SSO groups, see Adding an SSO group to a team.

You must have the **Space administrator** role to manage teams.

To add a user directly

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. Choose the **Members** tab.
4. Choose **Add member**.

> **Note**
>
> Users being added to a team must already be members of a space. You cannot add or invite a team member who is not a member of the space.

5. Choose a user in the drop-down field, and then choose **Save**. Choose either AWS Builder ID users or SSO users that are already set up in IAM Identity Center.

Removing a user from a team directly

You can remove team members from your team. All permissions will no longer be inherited by the user. You can add the user back to the team later.

> **Note**
>
> When you remove a team member, the associated permissions will be removed for the user from all projects and resources in the space.

You must have the **Space administrator** role to manage teams.

To remove a team member

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. Choose the **Members** tab.
4. Choose the selector next to the user you want to remove, and then choose **Remove**.
5. Enter *remove* in the input field, and then choose **Remove**.

Adding an SSO group to a team

If your space is configured as a space with SSO users and groups managed in IAM Identity Center, you can add an SSO group that will join the space as a separate team.

> **Note**
>
> When you choose to manage team members by adding AWS Builder ID users or SSO users directly, the option to use **SSO groups** is not available. To add users directly, see Adding a user to a team directly.

You must have the **Space administrator** role to manage teams.

To add an SSO group as a team

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the page for your space, choose **Teams**. Choose the **SSO groups** tab.
3. Choose the SSO groups you want to add. You can add up to five SSO groups.

Deleting a team

You can delete a team that you no longer need.

> **Note**
>
> When you delete a team, the associated permissions will be removed for all team members from all projects and resources in the space.

You must have the **Space administrator** role to manage teams.

Delete a team

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Settings**, and then choose **Teams**.
3. In **Actions**, choose **Delete team**. This changes the role for the entire team.
4. Choose **Delete**.

Allowing space access for machine resources

Machine resources are specific resources in CodeCatalyst that are granted permissions for projects or spaces in CodeCatalyst.

> **Note**
>
> The term machine resource does not refer to cloud infrastructure such as an Amazon EC2 instance, but it is instead meant to refer to a blueprint or workflow resource with permissions for a space or project.

A machine resource represents your identity from your authorized resource when accessing CodeCatalyst through SSO. Machine resources are used to grant permissions to resources in the space, such as **blueprints** and **workflows**. You can view the machine resources in your space, and you can choose to enable or disable machine resources for your space. For example, you might want to disable a machine resource to manage access and then re-enable it later.

These operations are available for machine resources in cases where a machine resource needs to be revoked or disabled. For example, if you suspect credentials might have been compromised, you can disable the machine resource. Generally, these operations will not need to be used.

You must have the **Space administrator** role to view this page and to manage machine resources at the space level.

Machine resources are also managed at the project level in CodeCatalyst. To learn more about teams in projects, see Allowing space access for machine resources .

Topics

Viewing space access for machine resources

You can view a listing of the machine resources that are in use in your space.

You must have the **Space administrator** role to manage machine resources.

To view machine resources

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space, and then choose **Settings**. Choose **Machine resources**.
3. In the drop-down, choose **Workflow action** to view only the machine resources for workflows. Choose **Blueprint** to view only the machine resources for blueprints.

   You can also filter on a name using the **Filter** field.

Disabling space access for machine resources

You can choose to disable machine resources that are in use in your space.

> **Important**
>
> Disabling machine resources will remove all permissions to all associated blueprints or workflows in the space.

You must have the **Space administrator** role to manage machine resources.

To disable machine resources

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space, and then choose **Settings**. Choose **Machine resources**.
3. Choose one of the following.

   > **Important**
   >
   > Disabling machine resources will remove all permissions to all associated blueprints or workflows in the space.

   - To disable individually, choose the selector next to one or more machine resources you want to disable. Choose **Disable**, and then choose **This resource**.
   - To disable all resources, choose **Disable**, and then choose **All resources**.
   - To disable all workflow actions, choose **Disable**, and then choose **All workflow actions**.
   - To disable all blueprints, choose **Disable**, and then choose **All blueprints**.

Enabling space access for machine resources

You can choose to enable machine resources that are in use in your space and that have been disabled.

You must have the **Space administrator** role to manage machine resources.

To enable machine resources

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space, and then choose **Settings**. Choose **Machine resources**.
3. Choose one of the following.

   - To enable individually, choose the selector next to one or more machine resources you want to enable. Choose **Enable**, and then choose **This resource**.
   - To enable all resources, choose **Enable**, and then choose **All resources**.
   - To enable all workflow actions, choose **Enable**, and then choose **All workflow actions**.
   - To enable all blueprints, choose **Enable**, and then choose **All blueprints**.

Administering Dev Environments for a space

All Dev Environments are created as part of a project within a space. Space members can create their own Dev Environments within a project at the source repository level. Space administrators can then use the Amazon CodeCatalyst console to view, edit, delete, and stop Dev Environments on behalf of space members. In short, space administrators maintain Dev Environments at the space level.

Considerations for administering Dev Environments

- You must have the **Space administrator** role to view the **Dev Environments** page under **Settings** and to manage Dev Environments at the space level.
- Space members manage the Dev Environments that they create in projects through their CodeCatalyst accounts. When administering Dev Environments as a space administrator, you are maintaining these resources on behalf of space members.
- Dev Environments default to a specific compute and storage configuration. For information about billing and rates for upgrading your configuration, see the Amazon CodeCatalyst pricing page.

For other considerations about Dev Environments, including stopping running instances, default compute configuration, upgrading your compute, incurring costs, and configuring timeouts, see Write and modify code with Dev Environments in CodeCatalyst.

Topics

Viewing Dev Environments for your space

You can view the type, status, and details for all Dev Environments in your space. For more information about creating and running Dev Environments, see Creating a Dev Environment.

You must have the **Space administrator** role to view this page and to manage Dev Environments at the space level.

To view Dev Environments in your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

   > **Tip**
   >
   > If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Dev Environments**.

   The page lists all Dev Environments in your space. You can view the **Resource** name, the resource **alias** if applicable, the type of **IDE**, the default or configured **Compute** and **Storage**, and the configured **Timeout** for each Dev Environment.

Editing a Dev Environment for your space

You can edit the configuration for a Dev Environment, such as the configured length of timeout, if any, for an idle Dev Environment to stop running. For more information about editing a Dev Environment, see Editing a Dev Environment.

You must have the **Space administrator** role to view this page and to manage Dev Environments at the space level.

To edit Dev Environments in your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Dev Environments**.
4. Choose the selector next to the Dev Environment you want to manage. Choose **Edit**.
5. Make the changes you want to the compute or inactivity timeout for the Dev Environment.
6. Choose **Save**.

Stopping a Dev Environment for your space

You can stop a running Dev Environment before it becomes idle if the Dev Environment is configured to have a timeout. Otherwise, a Dev Environment with an elapsed timeout will already be stopped. For more information about stopping a Dev Environment, see Stopping a Dev Environment.

You must have the **Space administrator** role to view this page and to manage Dev Environments at the space level.

To stop a Dev Environments in your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Dev Environments**.
4. Choose the selector next to the Dev Environment you want to manage. Choose **Stop**.

Deleting a Dev Environment for your space

You can delete a Dev Environment that is no longer needed or that no longer has an owner. For more information about considerations for deleting a Dev Environment, see Deleting a Dev Environment.

You must have the **Space administrator** role to view this page and to manage Dev Environments at the space level.

To delete Dev Environments in your space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

   > **Tip**
   >
   > If you belong to more than one space, choose a space in the top navigation bar.

3. Choose **Settings**, and then choose **Dev Environments**.
4. Choose the selector next to the Dev Environment you want to manage. Choose **Delete**. To confirm, type `delete`, and then choose **Delete**.

Quotas for spaces

The following table describes quotas and limits for spaces in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Maximum number of Slack channels for a space	500
Maximum number of invitations for an email address	25
Maximum number of invitations for a user	500
Maximum number of active spaces per user per AWS Region	5

Maximum number of space creations per Region per month per user	5
Maximum number of SSO groups for a team	5
Maximum number of teams for a spaces	100
Maximum number of users for a team	1000
Space descriptions	Space descriptions are optional. If specified, they must be between 0 and 200 characters in length. They can contain any combination of letters, numbers, spaces, periods, underscores, commas, dashes, and the following special characters: `? & $ % + = / \ ; : \n \t \r`
Space names	Space names must be unique across CodeCatalyst. You cannot reuse names of deleted spaces. Space names must be between 3 and 63 characters in length. They must also begin with an alphanumeric character. Space names can contain any combination of letters, numbers, periods, underscores, and dashes. They cannot contain any of the following characters: `! ? @ # $ % ^ & * ( ) + = { } [ ] | /\ > < ~ ` ' " ; :`

Organize work with projects in CodeCatalyst

You use projects in Amazon CodeCatalyst to establish a collaboration space where development teams can conduct development tasks with shared continuous integration/continuous delivery (CI/CD) workflows and repositories. When you create a project, you can add, update, or remove resources. You can also monitor the progress of your team's work. You can have multiple projects within a space.

Spaces in CodeCatalyst are made up of projects. You can see every project within your space, but you can only use the projects of which you are a member. When you create a project, default roles for your project are generated, which you assign to users that you invite to your project.

- Anyone assigned to the project with a project role, such as the **Contributor role**, can access project resources, such as a source repository.
- Anyone with the **Space administrator Project administrator**or role can send invitations to join a project.
- Users with the **Project administrator** role can track activity, status, and other settings across shared resources.
- Users with the **Limited access** role can manage project assignments for features, code fixes, and tests as part of CI/CD workflows.

  Workflows are used to build, test, and release or update applications as a CI/CD pipeline. You can assemble workflows by adding actions that transfer and work on your source artifacts. When you run actions, your project cloud resources are used to provide on-demand compute ability for your workflow actions. You might configure more CI/CD workflows based on the activity and output you want to set up. For example, you might create a workflow for build and test actions only, where you can view test results and complete the workflow without a deployment while you fix bugs. Then, you might create another workflow to build and deploy your application to a staging environment.

When you create a project, you can use a blueprint to create a project that contains sample code and creates resources, or you can start with an empty project. If you create a project using a blueprint, the blueprint you choose determines which resources are added to your project and the tools that CodeCatalyst creates or configures so you can track and use your project resources. You can manually add or remove resources after you have created a project.

Each project tracks project activity as a list of events by user, such as when a project is created or a resource is modified. Project activity is monitored and aggregated at the space level. For more information about working with activity data, see Viewing all projects in a space.

If your project uses AWS resources, you can connect your CodeCatalyst account to an AWS account where you have administrative permissions to integrate resources for your project.

You can add source repositories, issues, and other resources to your project after you create it. You must have the **Space administrator** role to create projects.

Creating a project

With CodeCatalyst projects, you can conduct development tasks with shared continuous integration/continuous delivery (CI/CD) workflows and repositories, manage resources, track issues, and add users.

Before you create a project, you must have the **Space administrator** or **Power user** role.

Topics

- Creating an empty project in Amazon CodeCatalyst
- Creating a project with a linked third-party repository
- Creating a project with a blueprint
- Best practices when using Amazon Q to create projects or add functionality with blueprints
- Best practices for using blueprints with projects
- Adding resources and tasks to created projects

Creating an empty project in Amazon CodeCatalyst

You can create an empty project with no resources and manually add the resources you want at a later time.

Before you create a project, you must have the **Space administrator** or **Power user** role.

To create an empty project

1. Navigate to the space where you want to create a project.

2. On the space dashboard, choose **Create project**.
3. Choose **Start from scratch**.
4. Under **Give a name to your project**, enter the name that you want to assign to your project. The name must be unique within your space.
5. Choose **Create project**.

Creating a project with a linked third-party repository

You can keep your project's source code in a preferred third-party provider and still use all the CodeCatalyst features such as blueprints, lifecycle management, workflows, and more. To do this, you can create a new CodeCatalyst project that links to a GitHub repository, Bitbucket repository, or a GitLab project repository. You can then use your linked source repository in your CodeCatalyst project.

Before you create a CodeCatalyst project, you must have the **Space administrator** or **Power user** role. For more information, see Creating a space and Inviting a user directly to a space .

To create a project in CodeCatalyst that links to a source repository in your GitHub account, you'll need to complete the following three tasks:

1. Install the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension. You're prompted in an external site to connect and provide CodeCatalyst with access to your repository, which is done as part of the next step.

   > **Important**
   >
   > To install the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension to your CodeCatalyst space, you must be signed in with an account that has the **Space administrator** role in the space.

2. Connect your GitHub account or Bitbucket workspace, or GitLab user to CodeCatalyst.

   > **Important**
   >
   > To connect your GitHub account, Bitbucket workspace, GitLab user to your CodeCatalyst space, you must be both the third-party source's administrator and the CodeCatalyst **Space administrator**.

> **Important**
>
> After you install a repository extension, any repositories you link to CodeCatalyst will have their code indexed and stored in CodeCatalyst. This will make the code searchable in CodeCatalyst. To better understand the data protection for your code when using linked repositories in CodeCatalyst, see Data protection in the *Amazon CodeCatalyst User Guide*.

3. Create a CodeCatalyst project linked to your GitHub repository, Bitbucket repository, or GitLab project repository.

> **Important**
>
> While you can link a GitHub repository, Bitbucket repository, or GitLab project repository as a **Contributor**, you can only unlink a third-party repository as the **Space administrator** or the **Project administrator**. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

> **Important**
>
> CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
> As a best practice, always make sure you have the latest version of the extension before you link a repository.

> **Note**
>
> - A GitHub repository, Bitbucket repository, or GitLab project repository can only be linked to one CodeCatalyst project in a space.

- You can't use empty or archived GitHub repositories, Bitbucket repositories, or GitLab project repositories with CodeCatalyst projects.
- You can't link a GitHub repository, Bitbucket repository, or GitLab project repository that has the same name as a repository in a CodeCatalyst project.
- The **GitHub repositories** extension isn't compatible with GitHub Enterprise Server repositories.
- The **Bitbucket repositories** extension isn't compatible with Bitbucket Data Center repositories.
- The **GitLab repositories** extension isn't compatible with GitLab self-managed project repositories.
- You can't use the **Write description for me** or **Summarize comments** features with linked repositories. These features are only available in pull requests in CodeCatalyst.

For more information, see Add functionality to projects with extensions in CodeCatalyst.

To install the third-party extension

1. Navigate to the space where you want to create a project.
2. On the space dashboard, choose **Create project**.
3. Choose **Bring your own code**.
4. Under **Link existing repository**, choose **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories** depending on the third-party repository provider you want to use. You're prompted to connect your GitHub account, Bitbucket workspace, or GitLab account if you didn't do so previously. If the third-party extension of your choice isn't already installed, an install prompt displays.
5. If prompted, choose **Install**. Review the permissions required by the extension, and if you want to continue, choose **Install** again.

After you install the third-party extension, the next step is to connect your GitHub account, Bitbucket workspace, or GitLab user to your CodeCatalyst space.

To connect your GitHub account, Bitbucket workspace, or GitLab user to CodeCatalyst

Do one of the following depending on the third-party extension you chose to configure:

- **GitHub repositories**: Connect to a GitHub account.

  1. Choose **Connect GitHub account** to go to the external site for GitHub.
  2. Sign in to your GitHub account using your GitHub credentials, and then choose the account where you want to install Amazon CodeCatalyst.

     > **Tip**
     >
     > If you have previously connected a GitHub account to the space, you will not be prompted to reauthorize. You will instead see a dialog box asking you where you would like to install the extension if you are a member or collaborator in more than one GitHub space, or the configuration page for the Amazon CodeCatalyst application if you only belong to one GitHub space. Configure the application for the repository access that you want to allow, and then choose **Save**. If the **Save** button is not active, make a change to the configuration, and then try again.

  3. Choose whether you want to allow CodeCatalyst to access all current and future repositories, or choose the specific GitHub repositories you want to use in CodeCatalyst. The default option is to include all GitHub repositories in the GitHub account, including future repositories that will be accessed by CodeCatalyst.
  4. Review the permissions given to CodeCatalyst, and then choose **Install**.

  After connecting your GitHub account to CodeCatalyst, you're taken to the **GitHub repositories** extension details page, where you can view and manage connected GitHub accounts and linked GitHub repositories.

- **Bitbucket repositories**: Connect to a Bitbucket workspace.

  1. Choose **Connect Bitbucket workspace** to go to the external site for Bitbucket.
  2. Sign into your Bitbucket workspace using your Bitbucket credentials and review the permissions given to CodeCatalyst.
  3. From the **Authorize for workspace** dropdown menu, choose the Bitbucket workspace you want to provide CodeCatalyst access to, and then choose **Grant access**.

     > **Tip**
     >
     > If you have previously connected a Bitbucket workspace to the space, you will not be prompted to reauthorize. You will instead see a dialog asking you where you

would like to install the extension if you're a member or collaborator in more than one Bitbucket workspace, or the configuration page for the Amazon CodeCatalyst application if you only belong to one Bitbucket workspace. Configure the application for the workspace access you want to allow, and then choose **Grant access**. If the **Grant access** button is not active, make a change to the configuration, and then try again.

After connecting your Bitbucket workspace to CodeCatalyst, you're taken to the **Bitbucket repositories** extension details page, where you can view and manage connected Bitbucket workspaces and linked Bitbucket repositories.

- **GitLab repositories**: Connect to a GitLab user.

  1. Choose **Connect GitLab user** to go to the external site for GitLab.
  2. Sign in to your GitLab user using your GitLab credentials and review the permissions given to CodeCatalyst.

     Tip

     If you have previously connected a GitLab user to the space, you will not be prompted to reauthorize. You will instead be navigated back to the CodeCatalyst console.

  3. Choose **Authorize AWS Connector for GitLab**.

  After connecting your GitLab user to CodeCatalyst, you're taken to the **GitLab repositories** extension details page, where you can view and manage connected GitLab user and linked GitLab project repositories.

After connecting your third-party source to CodeCatalyst, you can link the third-party repositories to your CodeCatalyst projects.

To create your project

1. On the **Create project** page, choose the GitHub account you connected.
2. Depending on the third-party repository provider you connected, choose the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** repository dropdown menu to

view the third-party repositories, and then choose the repository that you want to link to your project.

3. In the **Name your project** text input field, enter the name that you want to assign to your project. The name must be unique within your space.
4. Choose **Create project**.

After installing the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension, connecting your resource provider, and linking your third-party repository to your CodeCatalyst project, you can use it in CodeCatalyst workflows and Dev Environments. You can also create third-party repositories in the connected GitHub account, Bitbucket workspace, or GitLab user with code generated from a blueprint. You can also use the linked repositories with Amazon Q Developer, blueprints, and more. For more information, see Automatically starting a workflow run after third-party repository events and Creating a Dev Environment.

Creating a project with a blueprint

You can provision all of your project resources and sample code with a project blueprint. For information about blueprints, see the Creating a comprehensive project with CodeCatalyst blueprints.

To create a project with a blueprint

1. In the CodeCatalyst console, navigate to the space where you want to create a project.
2. On the space dashboard, choose **Create project**.
3. Choose **Start with a blueprint**.

> **Tip**
>
> You can choose to add a blueprint by giving **Amazon Q** your project requirements to have Amazon Q suggest a blueprint to you. For more information, see Using Amazon Q to choose a blueprint when creating a project or adding functionality and Best practices when using Amazon Q to create projects or add functionality with blueprints.
> This feature is only available in the US West (Oregon) Region.
> This functionality requires that generative AI features are enabled for the space. For more information, see Managing generative AI features.

4. From the **CodeCatalyst blueprints** or **Space blueprints** tab, choose a blueprint, and then choose **Next**.
5. Under **Name your project**, enter the name that you want to assign to your project and its associated resource names. The name must be unique within your space.
6. (Optional) By default, the source code created by the blueprint is stored in a CodeCatalyst repository. Alternatively, you can choose to store the blueprint's source code in a third-party repository. For more information, see Add functionality to projects with extensions in CodeCatalyst.

   > **Important**
   >
   > CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
   > As a best practice, always make sure you have the latest version of the extension before you link a repository.

   Do one of the following depending on the third-party repository provider you want to use:

   - **GitHub repositories**: Connect a GitHub account.

     Choose the **Advanced** dropdown menu, choose GitHub as the repository provider, and then choose the GitHub account where you want to store the source code created by the blueprint.

     > **Note**
     >
     > If you're connecting a GitHub account, you must create a personal connection to establish identity mapping between your CodeCatalyst identity and your GitHub identity. For more information, see Personal connections and Accessing GitHub resources with personal connections.

   - **Bitbucket repositories**: Connect a Bitbucket workspace.

Choose the **Advanced** dropdown menu, choose Bitbucket as the repository provider, and then choose the Bitbucket workspace where you want to store the source code created by the blueprint.

- **GitLab repositories**: Connect a GitLab user.

  Choose the **Advanced** dropdown menu, choose GitLab as the repository provider, and then choose the GitLab user where you want to store the source code created by the blueprint.

7. Under **Project resources**, configure the blueprint parameters. Depending on the blueprint, you may have the option to name the source repository name.
8. (Optional) To view definition files with updates based on the project parameter selections you made, choose **View code** or **View workflow** from **Generate project preview**.
9. (Optional) Choose **View details** from the blueprint's card to view specific details about the blueprint, such as an overview of the blueprint's architecture, required connections and permissions, and the kind of resources the blueprint creates.
10. Choose **Create project**.

Best practices when using Amazon Q to create projects or add functionality with blueprints

When you create a project or want to add new components to an existing project, you might be unsure about which blueprint to use or how to integrate capabilities. CodeCatalyst includes integration with a generative AI assistant called Amazon Q that can analyze your project requirements and suggest a blueprint that best fits your needs.

You can use Amazon Q to help you create a project with a blueprint that creates components based on your requirements, or you can use Amazon Q to help you add a blueprint to an existing project. For example, to add resources for a web application or modern application to a project, specify your requirements and then the resources will be added with a recommended blueprint. Issues for remaining components can be created for you.

Amazon Q also creates issues for requirements that can't be addressed by a suggested blueprint. Additionally, you can assign those issues to Amazon Q. If you assign the issue to Amazon Q, it will attempt to create a draft solution for you to evaluate. This can help you and your team to focus and optimize work on issues that require your attention, while Amazon Q works on a solution for problems you don't have resources to address immediately.

> **Note**
>
> **Powered by Amazon Bedrock**: AWS implements automated abuse detection. Because the **Use Amazon Q to create or add features to a project** feature is built on Amazon Bedrock, users can take full advantage of the controls implemented in Amazon Bedrock to enforce safety, security, and the responsible use of artificial intelligence (AI).

The following are some best practices to help you create projects and add blueprints with Amazon Q.

> **Important**
>
> Generative AI features are only available in the US West (Oregon) Region.

- **Use the default prompts provided by Amazon Q** . Amazon Q does best with choosing blueprints from the provided prompts.
- **Use the configuration options suggested by Amazon Q to preview the blueprints**. Choose a blueprint to preview the sample code and resources that will be created by the blueprint.
- **Use a space that is enabled for Amazon Q** . To create a project with Amazon Q, or to add functionality to a project with blueprints using Amazon Q, use a space that is enabled for generative AI features. For more information, see Enabling or disabling generative AI features for a space.
- **Get more information about blueprints recommended by Amazon Q**. You might want to find out more about the kind of project resources, sample code, and components that are created with a specific recommended blueprint. For more information about available blueprints in CodeCatalyst, see Creating a comprehensive project with CodeCatalyst blueprints.
- **Allow Amazon Q to work with issues**. Allow Amazon Q to create issues for you, assign those issues, and track them. For more information, see Tutorial: Using CodeCatalyst generative AI features to speed up your development work.
- **Unassign Amazon Q from issues that are no longer worked on**. After you complete the example, unassign Amazon Q from any issues no longer being worked on. If Amazon Q has finished its work on an issue or could not find a solution, make sure to unassign Amazon Q to avoid reaching the maximum quota for generative AI features. For more information, see Managing generative AI features and Pricing.

- **View usage for Amazon Q**. You can view usage of generative AI features at the user level. Go to **My settings** to manage generative AI quotas and view usage by your Builder ID or single sign-on (SSO) identity. For more information, see Viewing usage of generative AI features in a space.

> **Important**
>
> The generative AI features in CodeCatalyst are subject to quotas. For more information, see Amazon Q Developer Pricing, Enabling or disabling generative AI features for a space, and Billing.

Best practices for using blueprints with projects

The following are some best practices to help you create a project with blueprints or add blueprints.

- **Use blueprints provided by CodeCatalyst to create or add to projects**. You can use blueprints to create a full project with source code and resources for developers. For example, the web application blueprint creates application and infrastructure resources and deploys a web application. You can create a project with a blueprint or add a custom blueprint to an existing project. For more information, see Creating a project with a blueprint. View any blueprint in CodeCatalyst to preview the sample code and resources that will be created by the blueprint.
- **Use custom blueprints designed by your organization**. You can use custom blueprints to create a full project in your space. The custom blueprint designed by your organization can provide standardization and best practices, which can also help to cut down on efforts to set up a new project. As a custom blueprint author, you can view details about which projects are using your blueprint throughout your space. Lifecycle management allows you to centrally manage the software development lifecycle of every project, and blueprint users can utilize lifecycle management to regenerate a codebase from updated options or versions of a blueprint. For more information, see Working with lifecycle management as a blueprint author.
- **Add the developer role or appropriate IAM roles to the account for your project**. During or after you complete the project creation steps, you can configure your blueprint permissions by choosing or creating IAM roles in an AWS account that is connected to the space.

Adding resources and tasks to created projects

After your project is ready, you can add resources and tasks.

- To learn about the CI/CD workflows created with your project, see Getting started with workflows.
- To work with build actions similar to those in your new project that deploy build artifacts to an Amazon S3 bucket, see Building with workflows and Tutorial: Upload artifacts to Amazon S3.
- To start with an empty project and work with deploying a similar serverless application with an AWS CloudFormation stack deployment, see Tutorial: Deploy a serverless application using AWS CloudFormation.
- To add an issues planning board, see Track and organize work with issues in CodeCatalyst.
- To view the project overview, project status, recent team activity, and assigned work, see Getting a list of projects.
- To view source code or create a pull request, see Store and collaborate on code with source repositories in CodeCatalyst.
- To set up notifications that send status alerts for workflow run success or failure, see Managing notifications in Amazon CodeCatalyst.
- To invite members to your project, see Granting users project permissions.
- To set up Dev Environments, see Write and modify code with Dev Environments in CodeCatalyst.

Getting a list of projects

From your CodeCatalyst space, you can view details of each project where you have project permissions.

To view a project, you must be a member of the project or have the **Space administrator** role for the space.

If you have not created a project yet, see Creating a project. You must have the **Space administrator** role for the space where you want to create a project.

- On the project overview, you can view project members, source repositories, workflow runs, open pull requests, project Dev Environments, and issues.
- Under project settings, you can view and manage project details, delete the project, invite new members to the project, manage project members, and configure notifications.

Viewing project tasks and Dev Environments

To view a summary of project tasks, such as open issues and pull requests that are assigned to you or created by you, and the project's associated Dev Environments, use the console.

To view a project, you must be a member of the project or have the **Space administrator** role for the space.

To view your source repositories, workflow runs, issues, pull requests, Dev Environments, and issues

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space with the project you want to view. Under **Projects**, choose your project.
3. In the navigation pane, choose **Overview**.
4. View the project tasks assigned to you and created by you.

   - View the **Members + View all** list to view a list of the project members.
   - View the **Repositories** card to view the source repositories that are associated with the project.
   - View the **Workflow runs** card to view the workflows that are associated with the project.
   - View the **Open pull requests** card to view a summary of code repository status, in addition to pull requests assigned to you and created by you.
   - View the **My Dev Environments** card to view a summary of the Dev Environments associated with the project.
   - View the **Issues** card to view a summary of your assigned tasks or tasks that you created.

Viewing all projects in a space

In the **Projects** list for your space, you can view all projects where you have permissions.

To view a summary of project tasks, such as open issues and pull requests that are assigned to you or created by you, and the project's associated Dev Environments, use the console.

To view a project, you must be a member of the project or have the **Space administrator** role for the space.

To view your source repositories, workflow runs, issues, pull requests, Dev Environments, and issues

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space with the project you want to view. Under **Projects**, choose your project.
3. In the navigation pane, choose **Project settings**.
4. View the project name, path, project ID, and description.

Viewing project settings

In the **Project settings**, you can view project members, source repositories, workflow runs, open pull requests, project Dev Environments, and issues.

To view a summary of project tasks, such as open issues and pull requests that are assigned to you or created by you, and the project's associated Dev Environments, use the console.

To view your source repositories, workflow runs, issues, pull requests, Dev Environments, and issues

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space with the project you want to view. Under **Projects**, choose your project.
3. In the navigation pane, choose **Project settings**.
4. View the project name, path, project ID, and description.

Changing to a different project in CodeCatalyst

To change to a different project, use the console to choose from a list of projects you have access to.

To change to a different project

1. In the CodeCatalyst console, choose the project selector at the top.
2. Expand the drop-down and choose the project you want to navigate to.

Deleting a project

You can delete a project to remove all access to the project's resources. You must have the Space administrator or Project administrator role to delete a project. Once you have deleted a project, project members will be unable to access project resources, and any workflows that are prompted by third-party source repositories will be stopped.

To delete your project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space with the project you want to view. Under **Projects**, choose your project.
3. In the navigation pane, choose **Project settings**.
4. Choose **Delete project**.
5. Enter **`delete`** to confirm the deletion.
6. Choose **Delete project**.

Granting users project permissions

You can manage the members in your projects using the Amazon CodeCatalyst console. You can add or remove users, manage current members' roles, send invitations to join your project, and cancel invitations that have not yet been accepted.

On the members page for space and project users, users can have multiple roles. Users with multiple roles will show an indicator when they have multiple roles, and they will be displayed with the role with the most permissions first.

Getting a list of members and their project roles

When you add a user to your project, you assign a role that grants project permissions as follows:

- The **Project administrator** role has all permissions in a project. Only assign this role to users who need to administer every aspect of a project, including editing project settings, managing project permissions, and deleting the project. For more information, see Project administrator role.
- The **Contributor** role has the permissions required to work in a project. Assign this role to those users who need to work with code, workflows, issues, and actions in a project. For more information, see Contributor role.
- The **Reviewer** role has review permissions. For details, see Reviewer role.

- The **Read only** role has read permissions. For details, see Read only role.

You do not need to invite a user with the **Space administrator** role to your project because they already have implicit access to all projects in the space.

When you invite a user to your project (without assigning the **Space administrator** role), the user will show in the **Project members** table under projects and in the**Project members** table under spaces.

To view users and roles in a space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space with the project you want to view. Under **Projects**, choose your project.
3. In the navigation pane, choose **Project settings**.
4. Choose the **Members** tab.

   The **Project members** table shows all members that have a role in a project.

   > **Tip**
   >
   > If you have the **Space administrator** role, you can view which projects you have been directly invited to. Navigate to **Project settings** for the project, and then choose **My projects**.

   The **Space administrators** table shows users with the **Space administrator** role. These users are automatically (implicity) assigned to all projects in the space and do not have a role in a project.

   In the **Status** column, the following are valid values:

   - **Invited** – CodeCatalyst sent the invitation but the user has not yet accepted or declined.
   - **Member** – The user accepted the invitation.

Topics

- Inviting a user to a project
- Canceling an invitation

- Removing a user from your project
- Accepting or declining an invitation for a project

Inviting a user to a project

You can use the console to invite users to your project. You can invite members of your space or add names from outside your space.

To invite users to your project, you must be signed in with the **Project administrator** or **Space administrator** role.

You do not need to invite a user with the **Space administrator** role to your project because they already have implicit access to all projects in the space.

When you invite a user to your project (without assigning the **Space administrator** role), the user will show in the **Project members** table under projects and in the**Project members** table under spaces.

To invite a member to your project from the Project settings tab

1. Navigate to your project.

   > **Tip**
   >
   > You can choose which project to view in the top navigation bar.

2. In the navigation pane, choose **Project settings**.
3. Choose the **Members** tab.
4. In **Project members**, choose **Invite new member**.
5. Type the new member's email address, choose the role for this member, and then choose **Invite**. For more information about roles, see Granting access with user roles.

To invite a member to your project from the Project overview page

1. Navigate to your project.

> **Tip**
>
> You can choose which project to view in the top navigation bar.

2. Choose the **Members +** button.
3. Type the new member's email address, choose the role for this member, and then choose **Invite**. For more information about roles, see Granting access with user roles.

Canceling an invitation

If you recently sent an invitation, you can cancel it as long as the invitation hasn't yet been accepted.

To manage project invitations, you must have the **Project administrator** or **Space administrator** role.

To cancel a project member invitation

1. Navigate to the project where you have sent an invitation that you want to cancel.
2. In the navigation pane, choose **Project settings**.
3. View the **Members** tab and verify that the member has a status of **Invited**.

> **Note**
>
> You can only cancel an invitation that has not yet been accepted.

4. Choose the option next to the row with the invited member, and then choose **Cancel invitation**.
5. A confirmation window displays. Choose **Cancel invitation** to confirm.

Removing a user from your project

You can use the console to remove a user from your project.

To remove a user from your project, you must be signed in with the **Project administrator** or **Space administrator** role.

> **Note**
>
> Removing a user from all projects within a space automatically removes the user from that space.

To remove a user from a project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space with the project you want to view. Under **Projects**, choose your project.
3. In the navigation pane, choose **Project settings**.
4. Choose the **Members** tab.
5. Choose the selector next to the profile you want to remove, and then choose **Remove**.
6. Confirm that you want to remove the user, and then choose **Remove**.

Accepting or declining an invitation for a project

You might receive an email invitation to join an Amazon CodeCatalyst project. You can accept or decline the invitation.

To accept or decline an invitation

1. Open the invitation email.
2. Choose the project link in the email.
3. Choose **Accept** or **Decline**.

   If you choose **Decline**, an email is sent to the project management account notifying them that you declined the invitation.

Allowing project access using teams

After you create a project, you can add teams. Teams allow you to group users so that they can share permissions and manage projects, issue tracking, roles, and resources in CodeCatalyst as project and space members.

You must have the **Project administrator** role to manage teams for your project.

Teams are also managed at the space level in CodeCatalyst. To learn more about teams in spaces, see Allowing space access using teams.

Topics

Adding a team to a project

You can manage teams where the team members can access resources in your project.

On the members page for space and project users, users can have multiple roles. Users with multiple roles will show an indicator when they have multiple roles, and they will be displayed with the role with the most permissions first.

To add a team

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project. Choose **Project settings**, and then choose **Teams**.
3. Choose **Add team**.
4. In **Team**, choose a team from the list of teams available.
5. Under **Project role**, choose a role from the list of project roles available in CodeCatalyst.
   - **Project administrator** — For details, see Project administrator role.
   - **Contributor** — For details, see Contributor role.
   - **Reviewer** — For details, see Reviewer role.
   - **Read only** — For details, see Read only role.
6. Choose **Add team**.

Granting project roles for a team

A team can have role permissions, such as **Power user**, in a space. You can change the space role for a team, but note that all members of the team will inherit those permissions.

To add or change a project role

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Project settings**, and then choose **Teams**.
3. To change a role, choose the selector next to the team in this list, and then choose **Change role**. To add a role, choose **Add project role**. In **Project**, choose the project you want to add and in **Role**, choose the role. Choose one of the available project roles:
   - **Project administrator** - For details, see Project administrator role.
   - **Contributor** - For details, see Contributor role.
   - **Reviewer** - For details, see Reviewer role.
   - **Read only** - For details, see Read only role.
4. Choose **Save**.

Removing a project role for a team

In CodeCatalyst, you can view the project roles for your team. You can also view the members in a team. You can remove the project role for a team.

To remove a project role

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your space. Choose **Project settings**, and then choose **Teams**.
3. Choose the **Project roles** tab.
4. Choose the role you want to remove.

> **Important**
>
> Removing a role from a team removes the associated permissions for all users in the team.

5. Choose **Save**.

Allowing project access for machine resources

Machine resources are specific resources in CodeCatalystthat are granted permissions for projects or spaces in CodeCatalyst.

> **Note**
>
> The term machine resource does not refer to cloud infrastructure such as an EC2 instance, but it is instead meant to refer to a blueprint or workflow resource with permissions for a space or project.

An example of working with machine resources in projects includes enabling a blueprint resource to access a project on your behalf.

A machine resource represents your identity from your authorized resource when accessing CodeCatalyst through SSO. Machine resources are used to grant permissions to resources in your project, such as **blueprints** and **workflows**. You can view the machine resources in your project, and you can choose to enable or disable machine resources for your project. For example, you might want to disable a machine resource to manage access and then re-enable it later.

These operations are available for machine resources in cases where a machine resource needs to be revoked or disabled. For example, if you suspect credentials might have been compromised, you can disable the machine resource. Generally, these operations will not need to be used.

You must have the **Space administrator** role or the **Project administrator** role to view this page and to manage machine resources at the project level.

Machine resources are also managed at the space level in CodeCatalyst. To learn more about teams in spaces/projects, see Allowing space access for machine resources .

Topics

Viewing project access for machine resources

You can view a listing of the machine resources that are in use in your project.

You must have the **Space administrator** role or the **Project administrator** role.

To view machine resources

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, and then choose **Project settings**. Choose **Machine resources**.
3. In the drop-down, choose **Workflow action** to view only the machine resources for workflows. Choose **Blueprint** to view only the machine resources for blueprints.

   You can also filter on a name using the **Filter** field.

Disabling project access for machine resources

You can choose to disable machine resources that are in use in your project.

> **Important**
>
> Disabling machine resources will remove all permissions to all associated blueprints or workflows in the space.

You must have the **Space administrator** role or the **Project administrator** role.

To disable machine resources

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, and then choose **Project settings**. Choose **Machine resources**.
3. Choose one of the following.

> **Important**
>
> Disabling machine resources will remove all permissions to all associated blueprints or workflows in the space.

- To disable individually, choose the selector next to one or more machine resources you want to disable. Choose **Disable**, and then choose **This resource**.
- To disable all resources, choose **Disable**, and then choose **All resources**.
- To disable all workflow actions, choose **Disable**, and then choose **All workflow actions**.
- To disable all blueprints, choose **Disable**, and then choose **All blueprints**.

Enabling project access for machine resources

You can choose to enable machine resources that are in use in your project and that have been disabled.

You must have the **Space administrator** role or the **Project administrator** role.

To enable machine resources

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, and then choose **Project settings**. Choose **Machine resources**.
3. Choose one of the following.

   - To enable individually, choose the selector next to one or more machine resources you want to enable. Choose **Enable**, and then choose **This resource**.
   - To enable all resources, choose **Enable**, and then choose **All resources**.
   - To enable all workflow actions, choose **Enable**, and then choose **All workflow actions**.
   - To enable all blueprints, choose **Enable**, and then choose **All blueprints**.

Quotas for projects

The following table describes quotas and limits for projects in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Maximum number of projects per space	100
Maximum number of projects to which a user can belong	1,000

Maximum number of members that can belong to a project.	10,000
Project names	Project names must be unique within a space. Names must be between 3 and 63 character s. Names are case sensitive. Project names must begin with an alphanumeric character. Valid characters: **A-Z, a-z, 0-9, spaces,** and **. , _ (underscore) - (hyphen)** Project names cannot contain any of the following characters: `! ? @ # $ % ^ & * ( ) + = { } [ ] | \ / > < ~ ` ' " ; :`
Project descriptions	Project descriptions can be up to 200 characters. Valid characters: **A-Z, a-z, 0-9, spaces,** and **. , _ (underscore) - (hyphen)**. Project descriptions are optional.

Working with notifications in CodeCatalyst

You can set up notifications to monitor your projects and resources in CodeCatalyst. Users can choose the project events about which they want to receive emails in any project where they are a member. You can also choose to configure notifications sent to an entire team in a team messaging application, such as Slack, by configuring access between a CodeCatalyst space and a Slack workspace, and then configuring notifications for a project to be sent to one or more channels in that Slack workspace. Once you've configured access between a CodeCatalyst space and a Slack workspace, project members will also have the option to add their own Slack member IDs so that they can be notified directly about CodeCatalyst events in connected Slack workspaces and channels.

> **Note**
>
> The set of project events that can be sent to Slack are not the same set of events that users can choose to be notified about in email.

Topics

How do notifications work?

You can set up your project to provide notifications to your team messaging application, such as Slack.

What permissions are necessary for notifications?

Any project member can configure, view, update, or delete notification settings for a channel in CodeCatalyst. However, only users with the **Space administrator** role can add or delete Slack workspaces. All users can configure what project events they want to receive emails about for the projects they belong to in CodeCatalyst.

What CodeCatalyst events can I configure notifications about?

You can configure CodeCatalyst to deliver notifications to one or more Slack channels about workflow events. Once notifications have been configured between a CodeCatalyst project and Slack, project users can choose to add their own Slack member ID in order to receive direct messages in Slack channels about CodeCatalyst events. Users who add their Slack member IDs will receive direct mentions to their IDs in the Slack channels configured for their projects, helping raise awareness about events they care about.

You can also choose what events you want to receive emails about. These emails are sent to the email address configured for your AWS Builder ID.

How are notifications surfaced?

You can configure CodeCatalyst to deliver notifications to one or more Slack channels. You need to authorize CodeCatalyst to grant permissions to access your Slack workspace. Once the authorization is provided, CodeCatalyst can deliver notifications to the Slack channels you configure. If a project member chooses to add their Slack member ID, they can receive mentions about CodeCatalyst events in the Slack channels configured for that project.

How do I set up notifications?

Email notifications are configured as part of CodeCatalyst. Project users can choose what events they'd like to receive emails about in their **My settings** page.

To set up Slack notifications for your project resources, you must complete the following high-level tasks.

To set up notifications (high-level tasks)

1. In CodeCatalyst, you **set up a connection** between CodeCatalyst and a messaging client, such as Slack. Once a Slack workspace is connected, it will be available to all projects in the space.

   > **Note**
   >
   > Only users with a Space administrator role can add or delete a Slack workspace.

2. In your project in CodeCatalyst, **add the channel** where you want your team to receive notifications.
3. In CodeCatalyst, you **turn on notifications** for various events, such as workflow run failure, and specify the channel where you want them sent.

For detailed steps, see Getting started with Slack notifications.

Once notifications have been configured between a CodeCatalyst space and Slack, users can choose to add their own Slack member IDs to receive direct messages about CodeCatalyst events in the Slack channels configured for their projects,

Getting started with Slack notifications

After you create a project, you can set up Slack notifications that help your team to monitor project resources.

These steps walk you through setting up Slack notifications for the first time in CodeCatalyst. If you have already configured notifications, see Managing notifications in Amazon CodeCatalyst.

> **Note**
>
> The set of project events that can be sent to notification channels are not the same set of events that users can choose to be notified about in email. For more information, see Managing notifications in Amazon CodeCatalyst.

Topics

Prerequisites

Before you begin, you need the following:

- A CodeCatalyst space. For information about creating a CodeCatalyst space and signing in for the first time, see Set up and sign in to CodeCatalyst.
- A CodeCatalyst project. For more information, see Creating a project.
- A CodeCatalyst account with the **Project administrator** or **Space administrator** role. For more information, see Granting access with user roles.
- A Slack account and Slack workspace that can be accessed by CodeCatalyst.
- A Slack channel where CodeCatalyst will send notifications. The channel can be public or private.

Step 1: Connect CodeCatalyst to your Slack workspace

Only users with the **Space administrator** role can add or delete Slack workspaces. Adding or deleting a Slack workspace affects all projects in the space. To establish the connection between CodeCatalyst and Slack, CodeCatalyst performs a secure OAuth authentication handshake with your Slack workspace.

Use the following instructions to connect CodeCatalyst to your Slack workspace.

> **Note**
>
> This only needs to be done once for each Slack workspace. You can then set up notifications by Slack channel.

To connect CodeCatalyst to your Slack workspace

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project.
3. In the navigation pane, choose **Project settings**.
4. Choose the **Notifications** tab.
5. Choose **Configure notifications**.
6. Choose **Connect to Slack workspace**.
7. Read the dialog box contents, and then choose **Connect to Slack workspace**.
8. On the **AWS Chatbot** message:

   a. In the upper right, choose the Slack workspace that contains your channel.

   b. Choose **Allow**.

   You are returned to the CodeCatalyst console.
9. Continue to Step 2: Add your Slack channel to CodeCatalyst.

Step 2: Add your Slack channel to CodeCatalyst

You need the Slack channel ID to add your channel to CodeCatalyst.

To get your Slack channel ID

1. Sign in to Slack. For more information, see Sign in to Slack.
2. Go to the Slack workspace that contains the channel where you want notifications to go. For more information, see Switch between Slack workspaces or Sign in to additional Slack workspaces.
3. In the navigation pane, open the context (right click) menu for the channel where you want notifications to go, and choose **Open channel details**.

The channel ID is displayed at the bottom of the dialog box.

4. Copy the **Channel ID** value. You'll need it in the next step.

Using the channel ID you just copied, you can now connect your Slack channel to CodeCatalyst.

To add your Slack channel to CodeCatalyst

1. Before you begin, if your Slack channel is private, add the AWS Chatbot app to the channel as follows:

   a. In your Slack channel's message box, enter **@aws** and choose **aws app** from the dialog box.

   b. Press Enter.

      A Slackbot message appears, indicating that AWS Chatbot is not in the private channel.

   c. Choose **Invite Them** to invite AWS Chatbot to the channel.
2. In the CodeCatalyst console, choose **Next**.
3. In **Channel ID**, paste the Slack channel ID you obtained earlier.
4. In **Channel name**, enter a name. We recommend using the Slack channel name.
5. Choose **Next**.
6. In **Select notification events**, choose the type of event you want to receive notifications for.
7. Choose **Finish**.

Step 3: Test notifications from CodeCatalyst to Slack

After your project is configured to send notifications for workflow status, you can view your notifications in Slack.

To view your notifications in Slack

1. In your CodeCatalyst project, start a workflow manually in order to complete a workflow run and receive a status notification when the run finishes.
2. In Slack, view the channel you set up for notifications. Your notifications show the latest status from each workflow run, and whether it failed or succeeded.

Step 4: Next steps

Once a Slack workspace is configured for your CodeCatalyst space, you can add additional Slack channels existing CodeCatalyst projects, and add them for new projects after you create them. You can also let project users know that they can configure personal Slack notifications for their Slack member IDs, and configure the events for which they'll receive emails. For more information, see Managing notifications in Amazon CodeCatalyst.

Managing notifications in Amazon CodeCatalyst

You can configure CodeCatalyst to send notifications about events in your project. You can send notifications to messaging clients such as Slack channels. Project users can choose what project events they will be notified about by emails sent to the email address configured for their profile.

> **Note**
>
> The set of project events that can be sent to notification channels are not the same set of events that users can choose to be notified about in email.

Topics

Managing notifications sent directly to you

You can choose to have email notifications sent to you about events in any project where you are a member. These emails will be sent to the email address configured in your AWS Builder ID. By default, you will receive emails about all project events for which emails can be sent.

If a project has been configured to send notifications to a Slack channel, you can add your Slack member ID to receive direct mentions about CodeCatalyst events in that Slack channel. This can help raise your awareness of events happening in the projects where you have a role.

To configure email notifications for project events

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. In the top menu bar, choose your profile badge, and then choose **My settings**. The CodeCatalyst **My settings** page opens.

> **Tip**
>
> You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

3. In **Email notifications**, find the project in the list where you want to configure email notifications, and choose **Edit**.
4. Select the events for which you want to receive emails, and then choose **Save**.

To configure personal Slack notifications

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the top menu bar, choose your profile badge, and then choose **My settings**. The CodeCatalyst **My settings** page opens.

> **Tip**
>
> You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

3. In **Personal Slack notifications**, choose **Connect Slack ID**, and then choose **Connect to Slack workspace**. A separate window will open.

> **Tip**
>
> This option is not configurable unless a user with the **Space administrator** role has added a Slack workspace for your CodeCatalyst space. For more information, see Getting started with Slack notifications and Managing notifications sent to channels.

4. In the permissions request window, make sure that the name of the workspace matches the Slack workspace configured for the CodeCatalyst space. Choose **Allow** to allow AWS Chatbot access to the workspace. The window will close, and the Slack workspace will show the **Connnection status** as **Connected**.

> **Tip**
>
> If the connection status does not change, check to see if an error occurred connecting the Slack workspace. You might have to scroll up to see the error.

5. To stop receiving personal Slack notifications, choose the connected Slack workspace, and then choose **Disconnect Slack ID**.

Managing notifications sent to channels

You can choose to add and manage notifications about project events in CodeCatalyst sent to team resources, such as a team Slack channel. By doing this, you can help ensure that your entire team is aware of important events, such as when a workflow run fails.

> **Note**
>
> Any member of a project can manage notifications sent to channels for that project. However, only users with the **Space administrator** role can add or delete Slack workspaces.

Adding a notification channel for a project

You can add a channel where you want to receive notifications, such as your team's Slack channel.

To add a Slack channel for notifications

1. If you're adding your first Slack channel, see instead Getting started with Slack notifications.

   After setting up your first channel, return to this procedure to set up additional channels.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your project.
4. In the navigation pane, choose **Project settings**.
5. Choose the **Notifications** tab.
6. Choose **Add channel**.
7. Choose **Choose workspace**, and then select the Slack workspace that contains the channel where you want to send notifications.

If your Slack workspace is not in the list, you can add it by following the instructions in Getting started with Slack notifications.

8. Before entering a **Channel ID**, if the Slack channel you want to add is private, complete these steps:

   a. In your Slack channel's message box, enter **@aws** and choose **aws app** from the pop-up.

   b. Press Enter.

      A Slackbot message appears, indicating that AWS Chatbot is not in the private channel.

   c. Choose **Invite Them** to invite AWS Chatbot to the channel.

9. In CodeCatalyst's **Channel ID** field, enter the Slack channel ID. To find the ID, go to Slack, and in the navigation pane, right-click the channel and choose **Open channel details**.

   The channel ID is displayed at the bottom of the dialog box.

10. In **Channel name**, enter a name. We recommend using the Slack channel name.
11. In **Select notification events**, choose the type of event you want to receive notifications for.
12. Choose **Add**.

Editing notifications for a notification channel

You can change which channels notifications go to, and you can turn off specific notifications altogether.

To edit notifications

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project.
3. In the navigation pane, choose **Project settings**.
4. Choose the **Notifications** tab.
5. Choose **Edit notifications**.
6. Do one of the following:

   - To send a notification to a specific channel, choose the channel from the drop-down list.
   - To turn off a notification globally, choose the toggle next to the notification.
   - To stop sending a notification to a specific channel, choose the **X** on the channel.

7. Choose **Save**.

Removing a channel

To remove a channel

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project. In the navigation pane, choose **Project settings**.
3. On the **Project settings** page, choose the **Notifications** tab.
4. Choose the indicator next to the channel you want to remove and then choose **Remove channel**. When prompted, choose **Ok** in the confirmation window.

Set up CodeCatalyst projects with blueprints

Blueprints are arbitrary code generators that represent an architectural component of a CodeCatalyst project. The component can consist of anything from a workflow in a single file to the entire project complete with sample code. Blueprints take an arbitrary set of options and use those to generate an arbitrary set of output code that gets forwarded into a project. As the blueprint gets updated with the latest best practices or new options, it can regenerate the relevant parts of your codebase in projects containing that blueprint.

You can use an Amazon CodeCatalyst blueprint to create a full project with a source repository, sample source code, CI/CD workflows, build and test reports, and integrated issue tracking tools. A CodeCatalyst blueprint generates resources and source code based on configuration parameters set. When using a CodeCatalyst-managed blueprint, the blueprint you choose determines which resources are added to your project, as well as the tools that CodeCatalyst creates or configures, so you can track and use your project resources. As a blueprint user, you can create a project with a blueprint or add them to an existing CodeCatalyst project. You can add multiple blueprints in your project, and each can be applied as an independent component. For example, you can have project that was created with a web application blueprint, and then you add a security blueprint at a later time. When one of the blueprints are updated, you can incorporate the changes or fixes in your project through lifecycle management. For more information, see Creating a comprehensive project with CodeCatalyst blueprints and Working with lifecycle management as a blueprint user.

As a blueprint author, you can also create and publish custom blueprints for your CodeCatalyst space members to use your project resources. The custom blueprints can be developed to meet specified needs for your space's projects. After adding a custom blueprint to your space's blueprints catalog, you can manage the blueprint and continue to make updates so your space's projects stay up to date with the latest best practices. For more information, see Standardizing projects custom blueprints in CodeCatalyst. To view the blueprints SDK and sample blueprints, see the open-source GitHub repository.

You may already have standardization and best practices in place. Instead of creating and developing a custom blueprint from scratch, you can choose to convert an existing source repository with source code into a custom blueprint. For more information, see Converting source repositories to custom blueprints.

Topics

- Adding a blueprint in a project to integrate resources
- Disassociating a blueprint from a project to stop updates
- Changing blueprint versions in a project
- Editing a desciption for a blueprint in a project
- Working with lifecycle management as a blueprint user
- Creating a comprehensive project with CodeCatalyst blueprints
- Standardizing projects custom blueprints in CodeCatalyst
- Quotas for blueprints in CodeCatalyst

Creating a project with a blueprint

You can quickly create a project using a blueprint from the Amazon CodeCatalyst blueprints catalog or your team's space catalog with custom blueprints. Depending on the blueprint, your project is created with specific resources. You can also collaborate with Amazon Q, a generative AI assistant, when creating new projects or adding components to existing projects. You can provide Amazon Q with requirements for your project by interacting with it in a chat-like interface. Based on your requirements, Amazon Q suggests a blueprint and also outlines requirements that can't be met. You can then proceed with Amazon Q's suggestion if you're satisfied, and it will create the necessary resources such as a source repository with code for your requirement. For more information, see Creating a project with a blueprint, Best practices when using Amazon Q to create projects or add functionality with blueprints, and Creating a comprehensive project with CodeCatalyst blueprints.

After creating a project, you can add additional blueprints to your CodeCatalyst project from the CodeCatalyst catalog or your space's catalog with custom blueprints. Blueprints represent architectural components, so multiple blueprints can be used together in your project to incorporate your team's best practices. This also gives you the ability to make sure your project is up to date with the latest changes to the evolving components. To learn more about working with blueprints in your project, see Working with lifecycle management as a blueprint user.

Adding a blueprint in a project to integrate resources

You can add multiple blueprints in a project to incorporate functional components, resources, and governance. Your projects can support various elements that are managed independently

in separate blueprints. Adding blueprints to a project reduces the need to manually create resources and make software components functional. Your projects can also stay current as requirements evolve. To learn more about adding blueprints in your project, see Working with lifecycle management as a blueprint user.

While configuring a blueprint's details, you can also choose to store the blueprint's source code in a preferred third-party repository, where you can still manage the blueprint and utilize the lifecycle management capabilities to keep your project up to date. For more information, see Add functionality to projects with extensions in CodeCatalyst and Working with lifecycle management as a blueprint user.

Important

To add a blueprint in your CodeCatalyst project, you must be signed in with an account that has the **Space administrator**, **Power user**, or **Project administrator** role in the space.

Tip

After adding a blueprint to your project, you can configure your email and Slack notifications to provide updates for the latest changes to the blueprint. For more information, see Working with notifications in CodeCatalyst.

To add a blueprint to your project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space, and then choose the project where you want to add a blueprint.
3. In the navigation pane, choose **Blueprints**, and then choose **Add blueprint**.

   Tip

   You can choose to add a blueprint by giving **Amazon Q** your project requirements to have Amazon Q suggest a blueprint to you. For more information, see Using Amazon Q to choose a blueprint when creating a project or adding functionality and Best practices when using Amazon Q to create projects or add functionality with blueprints. This feature is only available in the US West (Oregon) Region.

This functionality requires that generative AI features are enabled for the space. For more information, see Managing generative AI features.

4. Choose a blueprint from the **CodeCatalyst blueprints** tab or a custom blueprint from the **Space blueprints** tab, and then choose **Next**.
5. Under **Blueprint details**, choose a blueprint version from the **Target version** dropdown menu. The latest catalog version is automatically selected.
6. (Optional) By default, the source code created by the blueprint is stored in a CodeCatalyst repository. Alternatively, you can choose to store the blueprint's source code in a third-party repository. For more information, see Add functionality to projects with extensions in CodeCatalyst.

   Do one of the following depending on the third-party repository provider you want to use:

   - **GitHub repositories**: Connect a GitHub account.

     Choose the **Advanced** dropdown menu, choose GitHub as the repository provider, and then choose the GitHub account where you want to store the source code created by the blueprint.

     Note

     If you're using a connection to a GitHub account, you must create a personal connection to establish identity mapping between your CodeCatalyst identity and your GitHub identity. For more information, see Personal connections and Accessing GitHub resources with personal connections.

   - **Bitbucket repositories**: Connect a Bitbucket workspace.

     Choose the **Advanced** dropdown menu, choose Bitbucket as the repository provider, and then choose the Bitbucket workspace where you want to store the source code created by the blueprint.

   - **GitLab repositories**: Connect a GitLab user.

     Choose the **Advanced** dropdown menu, choose GitLab as the repository provider, and then choose the GitLab user where you want to store the source code created by the blueprint.
7. Under **Configure blueprint**, configure the blueprint parameters. Depending on the blueprint, you may have the option to name the source repository.

8. Review the differences between the current blueprint version and your updated version. The difference displayed in a pull request shows the changes between the current version and the latest version, which is the desired version at the time the pull request was created. If no changes display, the versions might be identical, or you might have chosen the same version for both the current version and the desired version.
9. When you're satisfied that the pull request contains the code and changes that you want reviewed, choose **Add blueprint**. After you create the pull request, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resourcesuch as filesby using the @ sign, followed by the name of the file.

> **Note**
>
> The blueprint won't be applied until the pull request is approved and merged. For more information, see Reviewing a pull request and Merging a pull request.

Blueprint authors can also add a custom blueprint to projects in specified spaces that don't have the blueprint available to create new projects or add to existing projects. For more information, see Publishing and adding a custom blueprint in specified spaces and projects.

If you no longer want to receive updates for a blueprint, you can disassociate the blueprint from your project. For more information, see Disassociating a blueprint from a project to stop updates.

Disassociating a blueprint from a project to stop updates

If you don't want new updates from a blueprint, you can disassociate the blueprint from your project. Resources and functional software components added to your project from the blueprint will remain in your project.

> **Important**
>
> To disassociate a blueprint from your CodeCatalyst project, you must be signed in with an account that has the **Space administrator**, **Power user**, or **Project administrator** role in the space.

To disassociate a blueprint from your project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space, and then choose the project where you want to disassociate a blueprint.
3. In the navigation pane, choose **Blueprints**.
4. Choose the blueprint with the resources your want to disassociate, choose the **Actions** dropdown menu, and then choose **Disassociate blueprint**.
5. Enter `confirm` to confirm the disassociation.
6. Choose **Confirm**.

Changing blueprint versions in a project

If you created a project with a blueprint or added a blueprint to an existing project, then you're notified of new versions of the blueprint. Before the blueprint version is updated through an approved pull request, you can view the code changes and affected environments. Lifecycle management allows you to change versions of one or more applied blueprints in your project, so each blueprint version can be changed without impacting other areas of your project. You can also override blueprint updates. For more information, see Working with lifecycle management as a blueprint user.

> **Important**
>
> To change the version of a blueprint in your CodeCatalyst project, you must be signed in with an account that has the **Space administrator**, **Power user**, or **Project administrator** role in the space.

> **Tip**
>
> After adding a blueprint to your project, you can configure your email and Slack notifications to provide updates for the latest changes to the blueprint. For more information, see Working with notifications in CodeCatalyst.

To update a blueprint to the latest version

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. In the CodeCatalyst console, navigate to the space where you want to update a blueprint's version.
3. On the space dashboard, choose the project with the blueprint that you want to update.
4. In the navigation pane, choose **Blueprints**.
5. In the **Status** column, choose the link to change the catalog version (for example, **Change catalog version 0.3.109)**.
6. Choose the **Actions** dropdown menu, and then choose **Update version**. The latest version is automatically selected.
7. (Optional) Under **Configure blueprint**, configure the blueprint parameters.
8. (Optional) In the **Code changes** tab, review the differences between the current blueprint version and the updated version. The difference displayed in a pull request is the changes between the current version and the latest version, which is the desired version at the time the pull request is created. If no changes display, the versions might be identical, or you might have chosen the same version for both the current version and the desired version.
9. When you're satisfied that the pull request contains the code and changes that you want reviewed, choose **Apply update**. After you create the pull request, you can add comments. Comments can be added to the pull request or to individual lines in files and to the overall pull request. You can add links to resources such as files by using the @ sign, followed by the name of the file.

> **Note**
>
> The blueprint won't update until the pull request is approved and merged. For more information, see Reviewing a pull request and Merging a pull request.

> **Note**
>
> If you have existing pull requests open for updating a blueprint version, close the previous pull requests before creating a new one. When you choose **Update version**, you will be directed to the list of pending pull requests for the blueprint. You can also view pending pull requests from the **Blueprints** tab in the project **Settings** and the project summary page. For more information, see Viewing pull requests.

To change a blueprint version

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to update a blueprint's version.
3. On the space dashboard, choose the project with the blueprint that you want to update.
4. In the navigation pane, choose **Blueprints**, and then choose the radio button for the blueprint you want to update.
5. Choose the **Actions** dropdown menu, and then choose **Configure blueprint**.
6. From the **Target version** dropdown menu, choose the version that you want to use. The latest version is automatically selected.
7. (Optional) Under **Configure blueprint**, configure the blueprint parameters.
8. (Optional) In the **Code changes** tab, review the differences between the current blueprint version and the updated version. The difference displayed in a pull request is the changes between the current version and the latest version, which is the desired version at the time the pull request is created. If no changes display, the versions might be identical, or you might have chosen the same version for both the current version and the desired version.
9. When you're satisfied that the pull request contains the code and changes that you want reviewed, choose **Apply update**. After you create the pull request, you can add comments. Comments can be added to the pull request or to individual lines in files and to the overall pull request. You can add links to resources such as files by using the @ sign, followed by the name of the file.

> **Note**
>
> The blueprint won't update until the pull request is approved and merged. For more information, see Reviewing a pull request and Merging a pull request.

> **Note**
>
> If you have existing pull requests open for updating a blueprint version, close the previous pull requests before creating a new one. When you choose **Update version**, you will be directed to the list of pending pull requests for the blueprint. You can also

view pending pull requests from the **Blueprints** tab in the project **Settings** and the project summary page. For more information, see Viewing pull requests.

Editing a desciption for a blueprint in a project

You can edit the description of a blueprint that you used to create a project or applied after a project was created. A blueprint can be used more than once in a project. To differentiate the purpose of blueprints in your project, you can use descriptions for those blueprints. Descriptions can also be used to identify the components that you're adding from a specific blueprint.

Important

To edit a custom blueprint's description in your space, you must be signed in with an account that has the **Space administrator**, **Power user**, or **Project administrator** role in the space.

To edit a blueprint's description in your project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to your space, and then choose the project with the blueprint settings that you want to update.
3. In the navigation pane, choose **Blueprints**.
4. Choose the blueprint with the description that you want to update, choose the **Actions** dropdown menu, and then choose **Settings**.
5. In the **Blueprint description** text input field, enter a description to identify the blueprint in your project.
6. Choose **Save**.

Working with lifecycle management as a blueprint user

Lifecycle management is the ability to regenerate a codebase from updated options or versions of a blueprint. This allows a blueprint author to centrally manage the software development lifecycle of every project that contains a particular blueprint. For example, pushing a security fix to a web application blueprint will allow every project containing or created from the web application

blueprint to pick up that fix automatically. This same management framework also allows you as a blueprint user to change blueprint options after they have been selected.

Topics

Using lifecycle management on existing projects

You can use lifecycle management for projects created from blueprints or on existing projects not associated with any blueprints. For example, you can add a standard security practices blueprint into a five-year-old Java application that was never created from a blueprint. The blueprint generates a security scanning workflow and other related code. That portion of the codebase in the Java application will now be kept up to date automatically with your team's best practices any time changes are made to the blueprint.

Using lifecycle management on multiple blueprints in a project

Because blueprints represent architectural components, multiple blueprints can often be used together in the same project. For example, a project could be made up of a central web API blueprint built by a company platform engineer, along with a release check blueprint built by the app-security team. Each of those blueprints can be updated independently and will remember merge resolutions applied to them in the past.

> **Note**
>
> As arbitrary architectural components, not all blueprints make sense together or will logically work together, even though they will still attempt to merge into each other.

Working with conflicts in lifecycle pull requests

Occasionally, lifecycle pull requests might generate merge conflicts. These can be manually resolved. Resolutions are remembered on subsequent blueprint updates.

Opting out of lifecycle management changes

Users can remove a blueprint from a project to disassociate all references to the blueprint and opt out of lifecycle updates. For safety reasons, this doesn't remove or impact any of the project's code or resources, including what was added from the blueprint. For more information, see Disassociating a blueprint from a project to stop updates.

Overriding a blueprint's lifecycle management in a project

If you want to to override a blueprint's updates to specific files in your project, you can include an ownership file in your repository. GitLab's Code Owners spec is the recommended guidelines. The blueprint always respects the code owners file over everything else and can generate a sample one like the following:

```
new BlueprintOwnershipFile(sourceRepo, {
      resynthesis: {
        strategies: [
          {
            identifier: 'dont-override-sample-code',
            description: 'This strategy is applied accross all sample code. The
 blueprint will create sample code, but skip attempting to update it.',
            strategy: MergeStrategies.neverUpdate,
            globs: [
              '**/src/**',
              '**/css/**',
            ],
          },
        ],
      },
    });
```

This generates a `.ownership-file` with the following contents:

```
[dont-override-sample-code] @amazon-codecatalyst/blueprints.import-from-git
# This strategy is applied accross all sample code. The blueprint will create sample
 code, but skip attempting to update it.
# Internal merge strategy: neverUpdate
**/src/**
**/css/**
```

Creating a comprehensive project with CodeCatalyst blueprints

When you create a project using a blueprint, CodeCatalyst creates a full project with a source repository, sample source code, CI/CD workflows, build and test reports, and integrated issue tracking tools. A project blueprint uses code to provision cloud infrastructure, resources, and sample source artifacts for different types of applications and frameworks.

For more information, see Creating a project. You must be the Space administrator to create a project.

Topics

Available blueprints

Blueprint name	Blueprint description
ASP.NET Core web API	This blueprint creates a .NET 6 ASP.NET Core web API application. The blueprint uses the AWS Deployment tool for .NET and provides an option to configure Amazon Elastic Container Service, AWS App Runner, or AWS Elastic Beanstalk as a deployment target.
AWS Glue ETL	This blueprint creates a sample extract transform load (ETL) reference implement ation using AWS CDK, AWS Glue, AWS Lambda, and Amazon Athena to convert comma-separated values (CSVs) to Apache Parquet.
DevOps deployment pipeline	This blueprint creates a deployment pipeline using the AWS Deployment Pipeline Reference Architecture that deploys a reference applicati on to AWS across multiple stages.

Blueprint name	Blueprint description
Java API with AWS Fargate	This blueprint creates a containerized web service project. The project uses AWS Copilot CLI to build and deploy a containerized Spring Boot Java web service backed by Amazon DynamoDB on Amazon ECS. The project deploys a containerized app to an Amazon ECS cluster on AWS Fargate serverless compute. The app stores data in a DynamoDB table. After your workflow runs successfully, the sample web service is publicly available through the Application Load Balancer.
Modern three-tier web application	This blueprint generates code in Python for the application layer and Vue front-end framework to build and deploy a well-arch itected 3-tier modern web application.
.NET serverless application	This blueprint creates AWS Lambda functions using .NET CLI Lambda tools. The blueprint provides options for the AWS Lambda functions, including a choice of C# or F#.
Node.js API with AWS Fargate	This blueprint creates a containerized web service project. The project uses AWS Copilot CLI to build and deploy a containerized Express/Node.js web service on Amazon Elastic Container Service. The project deploys a containerized app to an Amazon ECS cluster on AWS Fargate serverless compute. After your workflow runs successfully, the sample web service is publicly available through the Application Load Balancer.

Blueprint name	Blueprint description
Serverless Application Model (SAM)	This blueprint creates a project that uses a serverless application model (SAM) to create and deploy an API. You can choose SDK for Java, TypeScript, or SDK for Python as the programming language.
Serverless image handler	This blueprint creates an application for high-speed image processing without reducing image quality.
Serverless RESTful microservice	This blueprint creates a REST API that uses AWS Lambda and Amazon API Gateway with a To Do service reference. You can choose SDK for Java, TypeScript, or SDK for Python as the programming language.
Single-page application	This blueprint creates a single-page applicati on (SPA) that uses React, Vue, and Angular frameworks. For hosting, choose from AWS Amplify Hosting or Amazon CloudFront and Amazon S3.
Static website	This blueprint creates a static website using Hugo or Jekyll static site generators. Static site generators use text input files (such as Markdown) to generate static web pages. They are ideal for rarely-changing, informati ve content, such as product pages, documenta tion, and blogs. The blueprint uses the AWS CDK to deploy static web pages to either AWS Amplify or Amazon S3 + CloudFront.

Blueprint name	Blueprint description
To Do web application	This blueprint creates a To Do serverless web application with frontend and backend components. You can choose SDK for Java, TypeScript, or SDK for Python as the programming language.
Video-on-demand web service	This blueprint creates a video-on-demand service that provides the ability to take in, transcode, and deliver content. The blueprint uses AWS Lambda, Amazon S3, Amazon CloudWatch, and AWS Elemental MediaConvert.
Subscribe to external blueprint	This blueprint creates a workflow for each imported package. These workflows run once a day to check NPM for new versions of the packages. If a new version exists, the workflow attempts to add it to your CodeCatalyst space as a custom blueprint. The action will fail if a package can't be found or isn't a blueprint. The target package must be on NPM, and the package must be a blueprint. The space must be subscribed at a tier that supports custom blueprints.
Bedrock GenAI chatbot	This blueprint creates a generative AI chatbot with Amazon Bedrock and Anthropic's Claude. With this blueprint, you can build and deploy your own secure, login-protected LLM playground that can be customized to your data. For more information, see the Bedrock GenAI Chatbot documentation.

Blueprint name	Blueprint description
AWS Project Development Kit (AWS PDK) blueprints	These PDK blueprints can be composed together to create an application comprisin g of a React website, Smithy API and the supporting CDK infrastructure to deploy it to AWS. The AWS PDK provides building blocks for common patterns along with development tools to manage and build your projects. For more information, see the AWS PDK GitHub source repository and Tutorial: Creating a full-stack application with composable PDK blueprints.

Finding project blueprint information

Several project blueprints are available in CodeCatalyst. For each blueprint, there is an accompanying summary and README file. The summary describes the resources that are installed by the blueprint, while the README file explains the blueprint in detail and provides instructions on how to use it.

Standardizing projects custom blueprints in CodeCatalyst

You can standardize the development and best practices for your CodeCatalyst space's projects with custom blueprints. Custom blueprints can be used to define various aspects of a CodeCatalyst project, such as workflow definitions and application code. After a custom blueprint is used to create a new project or applied to existing projects, any changes to the blueprint are available to those projects as pull request updates. As a blueprint author, you can view details about which projects are using your blueprints throughout your space, so you can see how standards are being applied across projects. Lifecycle mangement of a blueprint allows you to centrally manage the software development lifecycle of every project, giving you ability to make sure the projects in your space continue to follow best practices with the latest changes or fixes. For more information, see Working with lifecycle management as a blueprint author.

Custom blueprints provide the ability to update blueprint versions against the prior project through resynthesis. Resynthesis is the process of rerunning blueprint synthesis with updated

versions or the ability to incorporate fixes and changes into existing projects. For more information, see Custom blueprints concepts.

You may already have standardization and best practices in place. Instead of creating and developing a custom blueprint from scratch, you can choose to convert an existing source repository with source code into a custom blueprint. For more information, see Converting source repositories to custom blueprints.

To view the blueprints SDK and sample blueprints, see the open-source GitHub repository.

Topics

Custom blueprints concepts

Here are some concepts and terms that you should know when working with custom blueprints in CodeCatalyst.

Topics

- Space blueprints
- Space blueprints catalog
- Synthesis
- Resynthesis
- Partial options
- Projen

Blueprint project

A blueprint project gives you the ability to develop and publish blueprints to your space. A source repository is created during the project creation process, and the name of the repository is the one you chosewhen entering the **Project resources** details. During the blueprint creation process, if you choose to generate a workflow release, a publishing workflow is created in your blueprint with the **Blueprint Builder** blueprint. The workflow automatically publishes your latest version.

Space blueprints

You can view and manage all blueprints from the **Space blueprints** table when you navigate to the **Blueprints** section of your space. After blueprints are published to your space, they are made available as a space blueprint to be added and removed from your space's blueprints catalog. You can also manage publishing permissions and delete blueprints from in the **Blueprints** section of your space. For more information, see Viewing details, versions, and projects of a custom blueprint.

Space blueprints catalog

You can view all added custom blueprints from a space's blueprints catalog. This is where a space member can choose your custom blueprint to create a new project. This catalog is different to the CodeCatalyst catalog, which already has available blueprints for all space members. For more information, see Creating a comprehensive project with CodeCatalyst blueprints.

Synthesis

Synthesis is the process of generating a CodeCatalyst project bundle that represents the source code, configuration, and resources in a project. The bundle is then used by CodeCatalyst deployment API operations to deploy into a project. The process can be run locally while developing your custom blueprint to emulate project creation without having to create a project in CodeCatalyst. The following commands can be used to perform synthesis:

```
yarn blueprint:synth             # fast mode
yarn blueprint:synth --cache     # wizard emulation mode
```

The blueprint starts itself by calling the main `blueprint.ts` class with that option merged on `defaults.json`. A new project bundle is generated under the `synth/synth.[options-name]/proposed-bundle/` folder. The output includes the project bundle that a custom blueprint generates, given the options you set, including the partial options that you may have configured.

Resynthesis

Resynthesis is the process of regenerating a blueprint with different blueprint options or blueprint versions of existing projects. As a blueprint author, you can define custom merge strategies in the custom blueprint code. You can also define ownership boundaries in an `.ownership-file` to specify in which parts of the codebase a blueprint is permitted to be updated. While the custom blueprint can propose updates to the `.ownership-file`, project developers using the custom blueprint can determine ownership boundaries for their projects. You can run resynthesis locally, and test and update before publishing your custom blueprint. Use the following commands to perform resynthesis:

```
yarn blueprint:resynth             # fast mode
yarn blueprint:resynth --cache     # wizard emulation mode
```

The blueprint starts itself by calling the main `blueprint.ts` class with that option merged on `defaults.json`. A new project bundle is generated under the `synth/resynth.[options-name]/` folder. The output includes the project bundle that a custom blueprint generates, given the options you set, including the partial options that you may have configured.

The following contents are created after the synthesis and resynthesis processes:

- **proposed-bundle** - The output of synthesis when it is run with new options for the target blueprint version.
- **existing-bundle** - A mock of your existing project. If there's nothing in this folder, it's generated with the same output as he `proposed-bundle`.
- **ancestor-bundle** - A mock of what your blueprint would generate when run with either a prior version, prior options, or a combination. If there's nothing in this folder, it's generated with the same output as the `proposed-bundle`.

- **resolved-bundle** - The bundle is always regenerated and defaults to a three-way merge between the `proposed-bundle`, `existing-bundle`, and the `ancestor-bundle`. This bundle provides an emulation of what a resynthesis would output locally.

To learn more about blueprint output bundles, see Generating files with resynthesis.

Partial options

You can add option variations under `src/wizard-configuration/` that don't have to enumerate the entirety of the `Options` interface, and the options are merged on top of the `defaults.json` file. That allows you to tailor test cases across particular options.

Example:

`Options` interface:

```
{
  language: "Python" | "Java" | "Typescript",
  repositoryName: string
  ...
}
```

`defaults.json` file:

```
{
  language: "Python",
  repositoryName: "Myrepo"
  ...
}
```

Additional configuration tests:

- ```
  #wizard-config-typescript-test.json
  {
    language: "Typescript",
  }
  ```
- ```
  #wizard-config-java-test.json
  {
    language: "Java",
  ```

```
}
```

Projen

Projen is an open-source tool that custom blueprints use to keep themselves updated and consistent. Blueprints come as Projen packages because this framework provides you with the ability to build, bundle, and publish projects, and you can use the interface to manage a project's configurations and settings.

You can use Projen to update blueprints at scale, even after they're created. The Projen tool is the underlying technology behind the blueprint synthesis that generates a project bundle. Projen owns the configuration for a project, and it shouldn't impact you as a blueprint author. You can run `yarn projen` to regenerate the configuration of your project after adding dependencies, or you can change options in the `projenrc.ts` file. Projen is also the underlying generation tool for custom blueprints to synthesize a project. For more information, see the projen GitHub page. To learn more about working with Projen, see the Projen documentation and How to simplify project setup with Projen.

Getting started with custom blueprints

During the process of creating a blueprint, you can configure the blueprint and generate a preview of the project resources. Each custom blueprint is managed by a CodeCatalyst project, which contains a workflow by default for publishing to the space's blueprints catalog.

While configuring your custom blueprint's details, you can also choose to store your blueprint's source code in a third-party repository, where you can still manage the custom blueprint and utilize the lifecycle management capabilities to keep your space's projects synchronized when the custom blueprint is modified. For more information, see Add functionality to projects with extensions in CodeCatalyst and Working with lifecycle management as a blueprint author.

If you already have a source repository with standardization and best practices in place, you can choose to convert that source repository into a custom blueprint. For more information, see Converting source repositories to custom blueprints.

Topics

- Step 2: Develop a custom blueprint with components
- Step 3: Preview a custom blueprint
- (Optional) Step 4: Publish a custom blueprint preview version

Prerequisites

Before creating a custom blueprint, consider the following requirements:

- Your CodeCatalyst space must be the **Enterprise** tier. For more information, see Managing billing in the Amazon CodeCatalyst Administrator Guide.
- You need to have the **Space administrator** or the **Power user** role to create custom blueprints. For more information, see Granting access with user roles.

Step 1: Create a custom blueprint in CodeCatalyst

When you create a custom blueprint from your space's settings, a repository is created for you. The repository includes all the required resources that you must have to develop your blueprint before publishing it to the space's blueprints catalog.

To create a custom blueprint

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to create a custom blueprint.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.
4. Choose **Create blueprint**.
5. Under **Name your blueprint**, enter the name that you want to assign to your project and its associated resource names. The name must be unique within your space.
6. (Optional) By default, the source code created by the blueprint is stored in a CodeCatalyst repository. Alternatively, you can choose to store the blueprint's source code in a third-party repository. For more information, see Add functionality to projects with extensions in CodeCatalyst.

   Do one of the following depending on the third-party repository provider you want to use:

   - **GitHub repositories**: Connect a GitHub account.

Choose the **Advanced** dropdown menu, choose GitHub as the repository provider, and then choose the GitHub account where you want to store the source code created by the blueprint.

> **Note**
>
> If you're using a connection to a GitHub account, you must create a personal connection to establish identity mapping between your CodeCatalyst identity and your GitHub identity. For more information, see Personal connections and Accessing GitHub resources with personal connections.

- **Bitbucket repositories**: Connect a Bitbucket workspace.

  Choose the **Advanced** dropdown menu, choose Bitbucket as the repository provider, and then choose the Bitbucket workspace where you want to store the source code created by the blueprint.

- **GitLab repositories**: Connect a GitLab user.

  Choose the **Advanced** dropdown menu, choose GitLab as the repository provider, and then choose the GitLab user where you want to store the source code created by the blueprint.

7. Under **Blueprint details**, do the following:

   a. In the **Blueprint display name** text input field, enter a name that will appear in your space's blueprints catalog.

   b. In the **Description** text input field, enter a description for your custom blueprint.

   c. In the **Author name** text input field, enter an author name for your custom blueprint.

   d. (Optional) Choose the **Advanced settings**.

      i. Choose **+ Add** to add tags that are added to the `package.json` file.

      ii. Choose the **License** dropdown menu, and then choose a license for your custom blueprint.

      iii. In the **Blueprint package name** text input field, enter a name to identify your blueprint package.

      iv. By default, a release workflow is generated using a publishing blueprint within your project called **Blueprint Builder**. The workflow publishes the latest blueprint version to your space when you push changes since publishing permissions are enabled by the

release workflow. To turn off the workflow generation, uncheck the **Release workflow** checkbox.

8. (Optional) A blueprint project comes with predefined code to support the publishing of the blueprint to the space's blueprints catalog. To view definition files with updates based on the project parameter selections you made, choose **View code** or **View workflow** from **Generate blueprint preview**.
9. Choose **Create blueprint**.

If you didn't turn off the workflow generation for your custom blueprint, the workflow automatically begins to run when your blueprint is created. When the workflow run is complete, your custom blueprint is available to be added to your space's blueprints catalog by default. You can turn off publishing permissions if you don't want the latest blueprint version to be published automatically to your space. For more information, see Setting publishing permissions for a custom blueprint and Running a workflow.

Since the publishing workflow called `blueprint-release` is created using a blueprint, the blueprint can be found as an applied blueprint in your project. For more information, see Adding a blueprint in a project to integrate resources and Disassociating a blueprint from a project to stop updates.

Step 2: Develop a custom blueprint with components

A blueprint wizard is generated when you create a custom blueprint, and it can be modified with components when developing the custom blueprint. You can update the `src/blueprints.js` and `src/defaults.json` files to modify the wizard.

> **Important**
>
> If you want to use blueprint packages from external sources, consider the risks that may come with those packages. You're responsible for the custom blueprints that you add to your space and the code they generate.

Create a Dev Environment in your CodeCatalyst project with a supported integrated development environment (IDE) before configuring your blueprint code. A Dev Environment is necessary to work with the required tools and packages.

To create a Dev Environment

1. In the navigation pane, do one of the following:

   a. Choose **Overview**, and then navigate to the **My Dev Environments** section.

   b. Choose **Code**, and then choose **Dev Environments**.

   c. Choose **Code**, choose **Source repositories**, and choose the repository that you created when creating your blueprint.

2. Choose **Create Dev Environment**.
3. Choose a supported IDE from the dropdown menu. See Supported integrated development environments for Dev Environments for more information.
4. Choose **Work in existing branch**, and from the **Existing branch** dropdown menu, choose the feature branch you created.
5. (Optional) In the **Alias - *optional*** text input field, enter an alias to identify the Dev Environment.
6. Choose **Create**. While your Dev Environment is being created, the Dev Environment status column displays **Starting**, and the status column displays **Running** when the Dev Environment has been created.

For more information, see Write and modify code with Dev Environments in CodeCatalyst.

To develop your custom blueprint

1. In a working terminal, use the following `yarn` command to install dependencies:

   ```
   yarn
   ```

   The required tools and packages are made available through the CodeCatalyst Dev Environment, including Yarn. If you're working on a custom blueprint without a Dev Environment, install Yarn to your system first. For more information, see Yarn's installation documentation.

2. Develop your custom blueprint so that it's configured to your preferences. You can modify your blueprint's wizard by adding components. For more information, see Developing a custom blueprint to meet project requirements, Modifying blueprint features with a front-end wizard, and Publishing a custom blueprint to a space.

Step 3: Preview a custom blueprint

After setting up and developing your custom blueprint, you can preview and publish the preview version of your blueprint to your space. A preview version gives you the ability to check that the blueprint is what you want before it's used to create new projects or applied to existing projects.

To preview a custom blueprint

1. In a working terminal, use the following `yarn` command:

   ```
   yarn blueprint:preview
   ```

2. Navigate to the `See this blueprint at:` link provided to preview your custom blueprint.
3. Check that the UI, including text, appears as you expected based on your configuration. If you want to change your custom blueprint, you can edit the `blueprint.ts` file, resynthesize the blueprint, and then publish a preview version again. For more information, see Resynthesis.

(Optional) Step 4: Publish a custom blueprint preview version

You can publish a preview version of your custom blueprint to your space if you want to add it to your space's blueprints catalog. This allows you to view the blueprint as a user before adding the non-preview version to the catalog. The preview version allows you to publish without taking up an actual version. For example, if you work on a `0.0.1` version, you can publish and add a preview version, so new updates for a second version can be published and added as `0.0.2`.

To publish a preview version of a custom blueprint

Navigate to the `Enable version` *`[version number]`* `at:` link provided to enable your custom blueprint. This link is provided when running the `yarn` command in Step 3: Preview a custom blueprint.

After creating, developing, previewing, and publishing your custom blueprint, you can publish and add the final blueprint version to your space's blueprints catalog. For more information, see Publishing a custom blueprint to a space and Adding a custom blueprint to a space blueprints catalog.

Tutorial: Creating and updating a React application

As a blueprint author, you can develop and add custom blueprints to your space's blueprints catalog. These blueprints can then be used by space members to create new projects or add them

to existing projects. You can continue to make changes to your blueprints that are then made available as updates through pull requests.

This tutorial provides a walkthrough from a blueprint author's perspective and a blueprint user's perspective. The tutorial shows how to create a React single-page web application blueprint. The blueprint is then used to create a new project. When the blueprint is updated with changes, the project created from the blueprint incorporates those changes through a pull request.

Topics

Prerequisites

To create and update a custom blueprint, you must have completed the tasks in Set up and sign in to CodeCatalyst as follows:

- Have an AWS Builder ID for signing in to CodeCatalyst.
- Belong to a space and have the **Space administrator** or **Power user** role assigned to you in that space. For more information, see Creating a space, Granting users space permissions, and Space administrator role.

Step 1: Create a custom blueprint

When you create a custom blueprint, a CodeCatalyst project is created that contains your blueprint source code and development tools and resources. Your project is where you will develop, test, and publish the blueprint.

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to create a blueprint.
3. Choose **Settings** to navigate to the space settings.
4. In the **Space settings** tab, choose **Blueprints** and choose **Create blueprint**.
5. Update the fields in the blueprint creation wizard with the following values:
   - In **Blueprint name**, enter `react-app-blueprint`.
   - In **Blueprint Display Name**, enter `react-app-blueprint`.
6. Optionally, choose **View code** to preview the blueprint source code for your blueprint. Likewise, choose **View workflow** to preview the workflow that will be created in the project that builds and publishes the blueprint.
7. Choose **Create blueprint**.
8. Once your blueprint is created, you are taken to the blueprint's project. This project contains the blueprint source code, along with the tools and resources you need to develop, test, and publish the blueprint. A release workflow was generated and it automatically published your blueprint to the space.
9. Now that your blueprint and blueprint project is created, the next step is to configure it by updating the source code. You can use Dev Environments to open and edit your source repository directly in your browser.

   In the navigation pane, choose **Code**, and then choose **Dev Environments**.
10. Choose **Create Dev Environment** and then choose **AWS Cloud9 (in browser)**.
11. Keep the default settings and choose **Create**.
12. In the AWS Cloud9 terminal, navigate to your blueprint project directory by running the following command:

    ```
    cd react-app-blueprint
    ```
13. A `static-assets` folder is created and filled automatically when a blueprint is created. In this tutorial, you will delete the default folder and generate a new one for a react app blueprint.

    Delete the static-assets folder by running the following command:

    ```
    rm -r static-assets
    ```

AWS Cloud9 is built on a Linux-based platform. If you're using a Windows operating system, you can use the following command instead:

```
rmdir /s /q static-assets
```

14. Now that the default folder is deleted, create a `static-assets` folder for a react-app blueprint by running the following command:

    ```
    npx create-react-app static-assets
    ```

    If prompted, enter `y` to proceed.

    A new react application was created in the `static-assets` folder with necessary packages. The changes need to pushed to your remote CodeCatalyst source repository.

15. Ensure you have the latest changes, and then commit and push the changes to the blueprint's CodeCatalyst source repository by running the following commands:

    ```
    git pull
    ```

    ```
    git add .
    ```

    ```
    git commit -m "Add React app to static-assets"
    ```

    ```
    git push
    ```

When a change is pushed to the blueprint's source repository, the release workflow is automatically started. This workflow increments the blueprint version, builds the blueprint, and publishes it to your space. In the next step, you'll navigate to the release workflow run to see how it's doing.

Step 2: View release workflow

1. In the CodeCatalyst console, in the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the **blueprint-release** workflow.
3. You can see the workflow has actions to build and publish the blueprint.

4. Under **Latest run**, choose the workflow run link to view the run from the code change you made.
5. Once the run is completed, your new blueprint version is published. Published blueprint versions can be seen in your space **Settings**, but can't be used in projects until it's added to the space's blueprints catalog. In the next step, you'll add the blueprint to the catalog.

Step 3: Add blueprint to catalog

Adding a blueprint to the space's blueprints catalog makes the blueprint available for use in all projects in a space. Space members can use the blueprint to create new projects or add them to existing projects.

1. In the CodeCatalyst console, navigate back to the space.
2. Choose **Settings**, and then choose **Blueprints**.
3. Choose **react-app-blueprint**, and then choose **Add to catalog**.
4. Choose **Save**.

Step 4: Create project with blueprint

Now that the blueprint is added to the catalog, it can be used in projects. In this step, you'll create a project with the blueprint you just created. In a later step, you'll update this project by updating and publishing a new version of the blueprint.

1. Choose the **Projects** tab and then choose **Create project**.
2. Choose **Space blueprints**, and then choose **react-app-blueprint**.

   > **Note**
   >
   > Once the blueprint is chosen, you can see the contents of the blueprint's `README.md` file.

3. Choose **Next**.
4.

   > **Note**
   >
   > The contents of this project creation wizard can be configured in the blueprint.

Enter the project name as a blueprint user. For this tutorial, enter `react-app-project`. For more information, see Developing a custom blueprint to meet project requirements.

Next, you'll make an update to the blueprint and add the new version to the catalog, which you will use to update this project.

Step 5: Update blueprint

After a blueprint is used to create a new project or applied to existing projects, you can continue to make updates as a blueprint author. In this step, you'll make changes to the blueprint and automatically publish a new version to the space. The new version can then be added as the catalog version.

1. Navigate to the **react-app-blueprint** project created in Tutorial: Creating and updating a React application.
2. Open the Dev Environment created in Tutorial: Creating and updating a React application.
   a. In the navigation pane, choose **Code**, and then choose **Dev Environments**.
   b. From the table, find the Dev Environment, and then choose **Open in AWS Cloud9 (in browser)**.
3. When the blueprint release workflow was run, it incremented the blueprint version by updating the `package.json` file. Pull that change in by running the following command in the AWS Cloud9 terminal:

   ```
   git pull
   ```
4. Navigate to the `static-assets` folder by running the following command:

   ```
   cd /projects/react-app-blueprint/static-assets
   ```
5. Create a `hello-world.txt` file in `static-assets` folder by running the following command:

   ```
   touch hello-world.txt
   ```

   AWS Cloud9 is built on a Linux-based platform. If you're using a Windows operating system, you can use the following command instead:

```
echo > hello-world.text
```

6. In the left-hand navigation, double-click the `hello-world.txt` file to open it in the editor, and add the following contents:

   ```
   Hello, world!
   ```

   Save the file.

7. Ensure you have the latest changes, and then commit and push the changes to the blueprint's CodeCatalyst source repository by running the following commands:

   ```
   git pull
   ```

   ```
   git add .
   ```

   ```
   git commit -m "prettier setup"
   ```

   ```
   git push
   ```

Pushing the changes started the release workflow, which will automatically publish the new version of the blueprint to the space.

Step 6: Update the blueprint's published catalog version to the new version

After a blueprint is used to create a new project or applied to existing projects, you can still update the blueprint as a blueprint author. In this step, you'll make changes to the blueprint and change the blueprint's catalog version.

1. In the CodeCatalyst console, navigate back to the space.
2. Choose **Settings**, and then choose **Blueprints**.
3. Choose **react-app-blueprint**, and then choose **Manage catalog version**.
4. Choose the new version, and then choose **Save**.

Step 7: Update project with new blueprint version

A new version is now available in the space's blueprints catalog. As a blueprint user, you can update the version for the project created in Step 4: Create project with blueprint. This ensures you have the latest changes and fixes needed to meet best practices.

1. In the CodeCatalyst console, navigate to **react-app-project** project created in Step 4: Create project with blueprint.
2. In the navigation pane, choose **Blueprints**.
3. Choose **Update blueprint** in the info box.
4. In the right-side **Code changes** panel, you can see the `hello-world.txt` and `package.json` updates.
5. Choose **Apply update**.

Choosing **Apply update** creates a pull request in the project with the changes from the updated blueprint version. To make the updates to the project, you must merge the pull request. For more information, see Reviewing a pull request and Merging a pull request.

1. In the **Blueprints** table, find the blueprint. In the **Status** column, choose **Pending pull request**, and then choose the link to the open pull request.
2. Review the pull request, and then choose **Merge**.
3. Choose **Fast forward merge** to keep the the default values, and then choose **Merge**.

Step 8: View the changes in the project

Changes to the blueprint are now available in your project after Step 7: Update project with new blueprint version. As a blueprint user, you can view the changes in the source repository.

1. In the navigation pane, choose **Source repositories**, and then choose the name of the source repository created when the project was created.
2. Under **Files**, you can view the `hello-world.txt` file that was created in Step 5: Update blueprint.
3. Choose the `hello-world.txt` to view the file's content.

Lifecycle management provides blueprint authors the ability centrally manage the software development lifecycle of every project that contains a particular blueprint. As seen in this tutorial, you can push updates to the blueprint that can then be incorporated by projects that used the blueprint to create a new project or applied it to an existing project. For more information, see Working with lifecycle management as a blueprint author.

Converting source repositories to custom blueprints

Custom blueprints allow you to incorporate best practices or new options across multiple projects in a CodeCatalyst space. While you can create and develop a new custom blueprint from scratch, you can also convert an existing CodeCatalyst or third-party source repository with code and established best practices into a blueprint project. You can avoid copying relevant artifacts from that existing repository into a blueprint project. After converting a source repository into a custom blueprint, you can update, publish, and add the blueprint just like any other custom blueprint.

After converting a source repository into a custom blueprint, the source repository is restructured to be a blueprint project, the contents of the repository are moved into a `static-assets` folder within the repository, and relevant assets needed for a blueprint are added into the repository. When the custom blueprint is used to create projects or added to an existing project, workflow definitions stored in the converted source repository is also added to projects by the blueprint.

> **Note**
>
> Resources such as environments and secrets aren't included when converting a source repository into a custom blueprint. You need to manually copy or add those resources after converting a source repository into a custom blueprint.

> **Important**
>
> To convert a source repository into a custom blueprint, you must be signed in with an account that has the **Project administrator**, **Space administrator**, or **Power user** role in the space.

To convert a source repository into a custom blueprint from the source repository list

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. In the CodeCatalyst console, navigate to the space, and then choose the project where you want to convert a source repository into a custom blueprint.
3. In the navigation pane, choose **Code**, choose **Source repositories**, and then choose the radio button of the source repository you want to convert into a custom blueprint.
4. Choose **Convert to blueprint** to convert your source repository into a custom blueprint.

You can also convert CodeCatalyst repositories into custom blueprints from the source repository page in the CodeCatalyst console.

To convert a source repository into a custom blueprint from the source repository page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space, and then choose the project where you want to convert a source repository into a custom blueprint.
3. In the navigation pane, choose **Code**, choose **Source repositories**, and then choose the name of the CodeCatalyst source repository you want to convert into a custom blueprint.
4. Choose the **More** dropdown menu, and then choose **Convert to blueprint** to convert your source repository into a custom blueprint.

Once your source repository is converted into a custom blueprint, a release workflow is automatically run. After the run is successfully complete, the custom blueprint is published to your space's custom blueprints list. From there, you can add your converted custom blueprint to your space's catalog to create new projects or to be added to existing projects. For more information, see Publishing a custom blueprint to a space and Adding a custom blueprint to a space blueprints catalog.

> **Important**
>
> To add a custom blueprint to your space's blueprints catalog, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

Working with lifecycle management as a blueprint author

Lifecycle management allows you to keep a large number of projects synchronized from a single common source of best-practices. This scales the propagation of fixes and the maintenance of

any number of projects across their entire software development lifecycle. Lifecycle management streamlines internal campaigns, security fixes, audits, runtime upgrades, changes in best practices, and other maintenance practices because those standards are defined in one place and automatically kept up to date centrally when new standards are published.

When a new version of your blueprint is published, all projects containing that blueprint are prompted to update to the latest version. As a blueprint author, you can also see the version of a particular blueprint that each project contains for compliance purposes. When there are conflicts in an existing source repository, lifecycle management creates pull requests. For all other resources, such as Dev Environment, all lifecycle management updates strictly create new resources. Users are free to merge or not merge those pull requests. When the pending pull requests are merged, the version of the blueprint, including options, used in your project are then updated. To learn about working with lifecycle management as a blueprint user, see Using lifecycle management on existing projects and Using lifecycle management on multiple blueprints in a project.

Topics

- Testing lifecycle management for bundle outputs and merge conflicts
- Using merge strategies to generate bundles and specifying files
- Accessing context objects for project details

Testing lifecycle management for bundle outputs and merge conflicts

You can locally test your blueprint's lifecycle management and merge conflict resolution. A series of bundles under the `synth/` directory that represent the various phases of a lifecycle update is generated. To test the lifecycle management, you can run the following yarn command on your blueprint:`yarn blueprint: resynth`. To learn more about resynthesis and bundles, see Resynthesis and Generating files with resynthesis.

Using merge strategies to generate bundles and specifying files

Topics

- Generating files with resynthesis
- Using merge strategies
- Specifying files for lifecycle management updates
- Writing merge strategies

Generating files with resynthesis

Resynthesis can merge the source code produced by a blueprint with source code that was previously generated by same the blueprint, allowing changes to a blueprint to be propagated to existing projects. Merges are run from the `resynth()` function across blueprint output bundles. Resynthesis first generates three bundles representing different aspects of the blueprint and project state. It can be manually run locally with the `yarn blueprint:resynth` command, which will create the bundles if they don't already exist. Manually working with the bundles will allow you to mock and test resynthesis behavior locally. By default, blueprints only run resynthesis across the repositories under `src/*` since only that portion of the bundle is typically under source control.

- `existing-bundle` - This bundle is a representation of the existing project state. This is artificially constructed by the synthesis compute to give the blueprint context about what's in the project it's deploying into (if anything). If something already exists at this location when running resynthesis locally, it will be reset and respected as a mock. Otherwise, it will be set to the contents of the `ancestor-bundle`.
- `ancestor-bundle` - This is the bundle that represents the blueprint output if it was synthesized with some previous options and/or version. If this is the first time this blueprint is being added to a project, then the ancestor doesn't exist, so it's set to the same contents as the `existing-bundle`. Locally, if this bundle already exists at this location, it will be respected as a mock.
- `proposed-bundle` - This is the bundle that mocks the blueprint if it was synthesized with some new options and/or version. This is the same bundle that would be produced by the `synth()` function. Locally, this bundle is always overridden.

Each bundle is created during a resynthesis phase that can be accessed from the blueprint class under `this.context.resynthesisPhase`.

- `resolved-bundle` - This is the final bundle, which is a representation of what gets packaged and deployed to a CodeCatalyst project. You can view which files and diffs are sent to the deployment mechanisms. This is the output of the `resynth()` function resolving merges between the three other bundles.

Three-way merge is applied by taking the difference between the `ancestor-bundle` and `proposed-bundle` and adding that to the `existing-bundle` to generate the `resolved-bundle`. All merge strategies resolve files to the `resolved-bundle`. Resynthesis resolves reach of

these bundles with the blueprint's merge strategies during `resynth()` and produces the resolved bundle from the result.

Using merge strategies

You can use a merge strategy vended by the blueprints library. These strategies provide ways to resolve file outputs and conflicts for files mentioned in the Generating files with resynthesis section.

- `alwaysUpdate` - A strategy that always resolves to the proposed file.
- `neverUpdate` - A strategy that always resolves to the existing file.
- `onlyAdd` - A strategy that resolves to the proposed file when an existing file doesn't exist already. Otherwise, resolves to the existing file.
- `threeWayMerge` - A strategy that performs a three-way merge between the existing, proposed, and common ancestor files. The resolved file may contain conflict markers if the files can't be cleanly merged. The provided files' contents must be UTF-8 encoded in order for the strategy to produce meaningful output. The strategy attempts to detect if the input files are binary. If the strategy detects a merge conflict in a binary file, it always returns the proposed file.
- `preferProposed` - A strategy that performs a three-way merge between the existing, proposed, and common ancestor files. This strategy resolves conflicts by selecting the proposed file's side of each conflict.
- `preferExisting` - A strategy that performs a three-way merge between the existing, proposed, and common ancestor files. This strategy resolves conflicts by selecting the existing file's side of each conflict.

To view the source code for the merge strategies, see the open-source GitHub repository.

Specifying files for lifecycle management updates

During resynthesis, blueprints control how changes are merged into an existing source repository. However, you might not want to push updates to every single file in your blueprint. For example, sample code like CSS stylesheets are intended to be project specific. The three-way merge strategy is the default option if you don't specify another strategy. Blueprints can specify which files they own and which files they don't by specifying merge strategies on the repository construct itself. Blueprints can update their merge strategies, and the latest strategies can be used during resynthesis.

```
const sourceRepo = new SourceRepository(this, {
```

```
      title: 'my-repo',
    });
    sourceRepo.setResynthStrategies([
      {
        identifier: 'dont-override-sample-code',
        description: 'This strategy is applied accross all sample code. The blueprint
 will create sample code, but skip attempting to update it.',
        strategy: MergeStrategies.neverUpdate,
        globs: [
          '**/src/**',
          '**/css/**',
        ],
      },
    ]);
```

Multiple merge strategies can be specified, and the last strategy takes precedence. Uncovered files default to a three-way-merge similar to Git. There are several merge strategies provided through the `MergeStrategies` construct, but you can write your own. The provided strategies adhere to the git merge strategy driver.

Writing merge strategies

In addition to using one of the provided build merge strategies, you can also write your own strategies. Strategies must adhere to a standard strategy interface. You must write a strategy function that takes versions of a file from the `existing-bundle`, `proposed-bundle`, and `ancestor-bundle`, and merges them into a single resolved file. For example:

```
type StrategyFunction = (
   /**
   * file from the ancestor bundle (if it exists)
   */
    commonAncestorFile: ContextFile | undefined,
   /**
   * file from the existing bundle (if it exists)
   */
    existingFile: ContextFile | undefined,
   /**
   * file from the proposed bundle (if it exists)
   */
    proposedFile: ContextFile | undefined,
    options?: {})
    /**
```

```
    * Return: file you'd like in the resolved bundle
    * passing undefined will delete the file from the resolved bundle
    */
=> ContextFile | undefined;
```

If the files don't exist (are undefined), then that file path doesn't exist in that particular location bundle.

Example:

```
strategies: [
          {
            identifier: 'dont-override-sample-code',
            description: 'This strategy is applied across all sample code. The
 blueprint will create sample code, but skip attempting to update it.',
            strategy: (ancestor, existing, proposed) => {
                const resolvedfile = ...
                ...
                // do something
                ...
                return resolvedfile
            },
            globs: [
              '**/src/**',
              '**/css/**',
            ],
          },
        ],
```

Accessing context objects for project details

As a blueprint author, you can access context from the blueprint's project during synthesis to get information like space and project names, or existing files in a project's source repository. You can also get details like the phase of resynthesis that the blueprint is generating. For example, you can access context to know if you're resynthesizing to generate an ancestor bundle or proposed bundle. Existing code context can then be used to transform your code in your repository. For example, you can write your own resynthesis strategy to set specific code standards. The strategy can be added to the `blueprint.ts` file for small blueprints, or you can create a separate file for strategies.

The following example shows how you can find files in a project's context, set a workflow builder, and set a blueprint-vended resynthesis strategy for a particular file:

```
const contextFiles = this.context.project.src.findAll({
      fileGlobs: ['**/package.json'],
    });

    // const workflows = this.context.project.src.findAll({
    //   fileGlobs: ['**/.codecatalyst/**.yaml'],
    // });

    const security = new WorkflowBuilder(this, {
      Name: 'security-workflow',
    });
    new Workflow(this, repo, security.getDefinition());
    repo.setResynthStrategies([
      {
        identifier: 'force-security',
        globs: ['**/.codecatalyst/security-workflow.yaml'],
        strategy: MergeStrategies.alwaysUpdate,
      },
    ]);


    for (const contextFile of contextFiles) {
      const packageObject = JSON.parse(contextFile.buffer.toString());
      new SourceFile(internalRepo, contextFile.path, JSON.stringify({
        ...packageObject,
      }, null, 2));
    }
  }
```

Developing a custom blueprint to meet project requirements

Before publishing a custom blueprint, you can develop the blueprint to meet specific requirements. You can develop your custom blueprint and test the blueprint by creating a project when previewing. You can develop the custom blueprint to include project components, such as specific source code, account connections, workflows, issues, or any other component that can be created in CodeCatalyst.

Important

If you want to use blueprint packages from external sources, consider the risks that may come with those packages. You're responsible for the custom blueprints that you add to your space and the code they generate.

Important

To develop a custom blueprint in your CodeCatalyst space, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

To develop or update a custom blueprint

1. Resume your Dev Environment. For more information, see Resuming a Dev Environment.

   If you don't have a Dev Environment, you must first create one. For more information, see Creating a Dev Environment.
2. Open a working terminal in your Dev Environment.
3. If you opted in for a release workflow when creating your blueprint, the latest blueprint version is automatically published. Pull the changes to make sure the `package.json` file has the incremented version. Use the following command:

   ```
   git pull
   ```
4. In the `src/blueprint.ts` file, edit the options of your custom blueprint. The `Options` interface is interpreted by the CodeCatalyst wizard dynamically to generate a selection user interface (UI). You can develop your custom blueprint by adding components and supported tags. For more information, see Modifying blueprint features with a front-end wizard, Adding environment components to a blueprint, Adding region components to a blueprint, Adding repository and source code components to a blueprint, Adding workflow components to a blueprint, and Adding Dev Environments components to a blueprint.

   You can also view the blueprints SDK and sample blueprints for additional support when developing your custom blueprint. For more information, see the open-source GitHub repository.

Custom blueprints provide preview bundles as a result of a successful synthesis. The project bundle represents the source code, configuration, and resources in a project, and it's used by CodeCatalyst deployment API operations to deploy into a project. If you want to continue developing your custom blueprint, rerun the blueprint synthesis process. For more information, see Custom blueprints concepts.

Modifying blueprint features with a front-end wizard

A blueprint selection wizard on CodeCatalyst is auto-generated by the `Options` interface in the `blueprint.ts` file. The front-end wizard supports modifications and features of a blueprint's `Options` using JSDOC style comments and tags. You can use JSDOC style comments and tags to perform tasks. For example, you can select the text displayed above an option, enable features such as input validation, or make an option collapsible. The wizard works by interpreting an abstract syntax tree (AST) generated from the TypeScript type from the `Options` interface. The wizard configures itself automatically to the type described as best as it can. Not all types are supported. Other supported types include the region selector and environment selector.

The following is an example of a wizard that uses JSDOC comments and tags with blueprint's `Options`:

```
export interface Options {
  /**
   * What do you want to call your new blueprint?
   * @validationRegex /^[a-zA-Z0-9_]+$/
   * @validationMessage Must contain only upper and lowercase letters, numbers and
 underscores
   */
  blueprintName: string;

  /**
   * Add a description for your new blueprint.
   */
   description?: string;

   /**
    * Tags for your Blueprint:
    * @collapsed true
    */
  tags?: string[];
}
```

The display name of each option of the `Options` interface appears in `camelCase` by default. Plain text in the JSDOC style comment is displayed as text above the option in the wizard.

Topics

Supported tags

The following JSDOC tags are supported by a custom blueprint's `Options` in the front-end wizard.

@inlinePolicy ./path/to/policy/file.json

- **Requires** - Option to be a type `Role`.
- **Usage** - Allows you to communicate the inline policies a role needs. The `policy.json` path is expected to be under source code. Use this tag when you need a custom policy for a role.
- **Dependencies** - `blueprint-cli 0.1.12` and above
- **Example** - `@inlinePolicy ./deployment-policy.json`

```
environment: EnvironmentDefinition{
    awsAccountConnection: AccountConnection{
      /**
       * @inlinePolicy ./path/to/deployment-policy.json
       */
      cdkRole: Role[];
    };
   };
```

@trustPolicy ./path/to/policy/file.json

- **Requires** - Option to be a type `Role`.
- **Usage** - Allows you to communicate the trust policies a role needs. The `policy.json` path is expected to be under source code. Use this tag when you need a custom policy for a role.
- **Dependencies** - `blueprint-cli 0.1.12` and above
- **Example** - `@trustPolicy ./trust-policy.json`

```
environment: EnvironmentDefinition{
    awsAccountConnection: AccountConnection{
      /**
       * @trustPolicy ./path/to/trust-policy.json
       */
      cdkRole: Role[];
    };
   };
```

@validationRegex Regex expression

- **Requires** - Option to be a string.
- **Usage** - Performs input validation on the option by using the given regex expression and displays `@validationMessage`.
- **Example** - `@validationRegex /^[a-zA-Z0-9_]+$/`
- **Recommendation** - Use with `@validationMessage`. Validation message is empty by default.

@validationMessage string

- **Requires** - `@validationRegex` or other errors to review usage.
- **Usage** - Displays validation message on `@validation*` failure.
- **Example** - `@validationMessage Must contain only upper and lowercase letters, numbers, and underscores`.
- **Recommendation** - Use with `@validationMessage`. Validation message is empty by default.

@collapsed boolean (optional)

- **Requires** - N/A
- **Usage** - Boolean that allows a suboption to be collapsible. If the collapsed annotation is present, its default value is true. Setting the value to `@collapsed false` creates a collapsible section that is initially open.
- **Example** - `@collapsed true`

@displayName string

- **Requires** - N/A

- **Usage** - Changes option display name. Allows formats other than camelCase for the display name.
- **Example** - `@displayName Blueprint Name`

@displayName string

- **Requires** - N/A
- **Usage** - Changes option display name. Allows formats other than camelCase for the display name.
- **Example** - `@displayName Blueprint Name`

@defaultEntropy number

- **Requires** - Option to be a string.
- **Usage** - Appends a randomized alphanumeric string of a specified length to the option.
- **Example** - `@defaultEntropy 5`

@placeholder string (optional)

- **Requires** - N/A
- **Usage** - Changes default text field placeholder.
- **Example** - `@placeholder type project name here`

@textArea number (optional)

- **Requires** - N/A
- **Usage** - Converts string input into a text area component for larger sections of text. Adding a number defines the number of rows. The default is five rows.
- **Example** - `@textArea 10`

@hidden boolean (optional)

- **Requires** - N/A
- **Usage** - Hides file from user unless validation check fails. Default value is true.

- **Example** - `@hidden`

@button boolean (optional)

- **Requires** - N/A
- **Usage** - Annotation must be on a Boolean property. Adds a button that will synthesize as true when chosen. Not a toggle.
- **Example** - `buttonExample: boolean;`

```
/**
  * @button
  */
buttonExample: boolean;
```

@showName boolean (optional)

- **Requires** - N/A
- **Usage** - Can only be used on an account connection type. Shows hidden name input. Defaults to `default_environment`.
- **Example** - `@showName true`

```
/**
  * @showName true
  */
accountConnection: AccountConnection<{
    ...
}>;
```

@showEnvironmentType boolean (optional)

- **Requires** - N/A
- **Usage** - Can only be used on an account connection type. Shows hidden environment type dropdown menu. All connections default to `production`. Options are **Non-production** or **Production**.
- **Example** - `@showEnvironmentType true`

```
/**
  * @showEnvironmentType true
  */
accountConnection: AccountConnection<{
    ...
}>;
```

@forceDefault boolean (optional)

- **Requires** - N/A
- **Usage** - Uses the default value provided by the blueprint author instead of the value that was used previously by the user.
- **Example** - `forceDeafultExample: any;`

```
/**
  * @forceDefault
  */
forceDeafultExample: any;
```

@requires blueprintName

- **Requires** - Annotates the `Options` interface
- **Usage** - Warns user to add specified `blueprintName` to project as a requirement for the current blueprint.
- **Example** - `@requires '@amazon-codecatalyst/blueprints.blueprint-builder'`

```
/*
 * @requires '@amazon-codecatalyst/blueprints.blueprint-builder'
 */
export interface Options extends ParentOptions {
...
```

Supported TypeScript types

The following TypeScript types are supported by a custom blueprint's `Options` in the front-end wizard.

Number

- **Requires** - Option to be a type `number`.
- **Usage** - Generate a number input field.
- **Example** - `age: number`

```
{
  age: number
  ...
}
```

String

- **Requires** - Option to be a type `string`.
- **Usage** - Generate a string input field.
- **Example** - `name: string`

```
{
  age: string
  ...
}
```

String list

- **Requires** - Option to be a type `boolean`.
- **Usage** - Generate a checkbox.
- **Example** - `isProduction: boolean`

```
{
  isProduction: boolean
  ...
}
```

Radio

- **Requires** - Option to be a union of three or fewer strings.

- **Usage** - Generate a radio selected.

> **Note**
>
> When there are four or more items, this type renders as a dropdown.

- **Example** - `color: 'red' | 'blue' | 'green'`

```
{
  color: 'red' | 'blue' | 'green'
  ...
}
```

Dropdown

- **Requires** - Option to be a union of four or more strings.
- **Usage** - Generate a dropdown.
- **Example** - `runtimes: 'nodejs' | 'python' | 'java' | 'dotnetcore' | 'ruby'`

```
{
  runtimes: 'nodejs' | 'python' | 'java' | 'dotnetcore' | 'ruby'
  ...
}
```

Expandable section

- **Requires** - Option to be an object.
- **Usage** - Generate an expandable section. Options in the object will be nested inside the expandable section in the wizard.
- **Example** -

```
{
     expandableSectionTitle: {
         nestedString: string;
         nestedNumber: number;
     }
```

```
}
```

Tuple

- **Requires** - Option to be of type `Tuple`.
- **Usage** - Generate a key-value paid input.
- **Example** - `tuple: Tuple[string, string]>`

```
{
    tuple: Tuple[string, string]>;
    ...
}
```

Tuple list

- **Requires** - Option to be an array of type `Tuple`.
- **Usage** - Generate a tuple list input.
- **Example** - `tupleList: Tuple[string, string]>[]`

```
{
  tupleList: Tuple[string, string]>[];
  ...
}
```

Selector

- **Requires** - Option to be of type `Selector`.
- **Usage** - Generate a dropdown of source repositories or blueprints applied to a project.
- **Example** - `sourceRepo: Selector<SourceRepository>`

```
{
    sourceRepo: Selector<SourceRepository>;
    sourceRepoOrAdd: Selector<SourceRepository | string>;
    blueprintInstantiation: Selector<BlueprintInstantiation>;
  ...
}
```

Multiselect

- **Requires** - Option to be of type `Selector`.
- **Usage** - Generate a multiselect input.
- **Example** - `multiselect: MultiSelect['A' | 'B' | 'C' | 'D' | 'E']>`

```
{
  multiselect: MultiSelect['A' | 'B' | 'C' | 'D' | 'E']>;
  ...
}
```

Communicating to the user during synthesis

As a blueprint author, you can communicate back to users beyond only validation messages. For example, a space member might view a combination of options that produces a blueprint that isn't clear. Custom blueprints supports the ability to communicate error messages back to users by invoking the synthesis. The base blueprint implements a `throwSynthesisError(...)` function that expects a clear error message. You can invoke the message by using the following:

```
//blueprint.ts
this.throwSynthesisError({
   name: BlueprintSynthesisErrorTypes.BlueprintSynthesisError,
   message: 'hello from the blueprint! This is a custom error communicated to the
 user.'
})
```

Adding environment components to a blueprint

The custom blueprint wizard is dynamically generated from the `Options` interface exposed through the wizard. Blueprints support generating user-interface (UI) components from exposed types.

To import Amazon CodeCatalyst blueprints environment components

In your `blueprint.ts` file, add the following:

```
import {...} from '@amazon-codecatalyst/codecatalyst-environments'
```

Topics

- Creating development environments
- Mock interface examples

Creating development environments

The following example shows how to deploy your application to the cloud:

```
export interface Options extends ParentOptions {
        ...
        myNewEnvironment:  EnvironmentDefinition{
            thisIsMyFirstAccountConnection: AccountConnection{
                thisIsARole: Role['lambda', 's3', 'dynamo'];
             };
        };
    }
```

The interface generates a UI component that asks for a new environment (`myNewEnvironment`) with a single account connection (`thisIsMyFirstAccountConnection`. A role on the account connection (`thisIsARole`) is also generated with `['lambda', 's3', 'dynamo']` as the minimum required role capabilities. Not all users have account connections, so you should check for the case where a user doesn't connect an account or doesn't connect an account with a role. Roles can also be annotated with `@inlinePolicies`. For more information, see @inlinePolicy ./path/to/policy/file.json.

The environment component requires a `name` and `environmentType`. The following code is the minimum required default shape:

```
{
  ...
  "myNewEnvironment": {
    "name": "myProductionEnvironment",
    "environmentType": "PRODUCTION"
  },
}
```

The UI component then prompts you for various fields. As you fill in the fields, the blueprint gets a fully expanded shape. It can be helpful for you to include the full mock in the `defaults.json` file for testing and development purposes.

Mock interface examples

Simple mock interface

```
{
    ...
    "thisIsMyEnvironment": {
        "name": "myProductionEnvironment",
        "environmentType": "PRODUCTION",
        "thisIsMySecondAccountConnection": {
            "id": "12345678910",
            "name": "my-account-connection-name",
            "secondAdminRole": {
                "arn": "arn:aws:iam::12345678910:role/ConnectedQuokkaRole",
                "name": "ConnectedQuokkaRole",
                "capabilities": [
                    "lambda",
                    "s3",
                    "dynamo"
                ]
            }
        }
    }
}
```

Complex mock interface

```
export interface Options extends ParentOptions {
  /**
   * The name of an environment
   * @displayName This is a Environment Name
   * @collapsed
   */
  thisIsMyEnvironment: EnvironmentDefinition{
    /**
     * comments about the account that is being deployed into
     * @displayName This account connection has an overriden name
     * @collapsed
     */
    thisIsMyFirstAccountConnection: AccountConnection{
      /**
       * Blah blah some information about the role that I expect
       * e.g. here's a copy-pastable policy: [to a link]
```

```
       * @displayName This role has an overriden name
       */
      adminRole: Role['admin', 'lambda', 's3', 'cloudfront'];
      /**
       * Blah blah some information about the second role that I expect
       * e.g. here's a copy-pastable policy: [to a link]
       */
      lambdaRole: Role['lambda', 's3'];
    };
    /**
     * comments about the account that is being deployed into
     */
    thisIsMySecondAccountConnection: AccountConnection{
      /**
         * Blah blah some information about the role that I expect
         * e.g. here's a copy-pastable policy: [to a link]
         */
      secondAdminRole: Role['admin', 'lambda', 's3', 'cloudfront'];
      /**
         * Blah blah some information about the second role that I expect
         * e.g. here's a copy-pastable policy: [to a link]
         */
      secondLambdaRole: Role['lambda', 's3'];
    };
  };
}
```

Complete mock interface

```
{
  ...
  "thisIsMyEnvironment": {
    "name": "my-production-environment",
    "environmentType": "PRODUCTION",
    "thisIsMySecondAccountConnection": {
      "id": "12345678910",
      "name": "my-connected-account",
      "secondAdminRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "admin",
          "lambda",
```

```
          "s3",
          "cloudfront"
        ]
      },
      "secondLambdaRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "lambda",
          "s3"
        ]
      }
    },
    "thisIsMyFirstAccountConnection": {
      "id": "12345678910",
      "name": "my-connected-account",
      "adminRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "admin",
          "lambda",
          "s3",
          "cloudfront"
        ]
      },
      "lambdaRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "lambda",
          "s3"
        ]
      }
    }
  },
}
```

Adding secrets components to a blueprint

Secrets can be used in CodeCatalyst to store sensitive data that can be referenced in workflows. You can add a secret to your custom blueprint and reference it in your workflow. For more information, see Configuring and using secrets in a workflow.

To import Amazon CodeCatalyst blueprints region type

In your `blueprint.ts` file, add the following:

```
import { Secret, SecretDefinition } from '@amazon-codecatalyst/blueprint-
component.secrets'
```

Topics

- Creating a secret
- Referencing a secret in a workflow

Creating a secret

The following example creates a UI component that prompts the user to enter a secret value and optional description:

```
export interface Options extends ParentOptions {
    ...
    mySecret: SecretDefinition;
}


export class Blueprint extends ParentBlueprint {
  constructor(options_: Options) {
    new Secret(this, options.secret);
}
```

The secret component requires a `name`. The following code is the minimum required default shape:

```
{
    ...
    "secret": {
        "name": "secretName"
    },

}
```

Referencing a secret in a workflow

The following example blueprint creates a secret and a workflow that references the secret value. For more information, see Referencing a secret in a workflow.

```
export interface Options extends ParentOptions {
    ...
/**
*
* @validationRegex /^\w+$/
*/
  username: string;


  password: SecretDefinition;
}


export class Blueprint extends ParentBlueprint {
  constructor(options_: Options) {
    const password = new Secret(this, options_.password);

    const workflowBuilder = new WorkflowBuilder(this, {
      Name: 'my_workflow',
    });


    workflowBuilder.addBuildAction({
      actionName: 'download_files',
      input: {
        Sources: ['WorkflowSource'],
      },
      output: {
        Artifacts: [{ Name: 'download', Files: ['file1'] }],
      },
      steps: [
        `curl -u ${options_.username}:${password.reference} https://example.com`,
      ],
    });

    new Workflow(
      this,
      repo,
      workflowBuilder.getDefinition(),
```

```
    );

}
```

To learn more about using secrets in CodeCatalyst, see Configuring and using secrets in a workflow.

Adding region components to a blueprint

The region type can be added to your custom blueprint's `Options` interface to generate a component in the blueprint wizard you can input one or more AWS gions. The gion type can be imported from your base blueprint in your `blueprint.ts` file. For more information, see AWS regions.

To import Amazon CodeCatalyst blueprints region type

In your `blueprint.ts` file, add the following:

```
import { Region } from '@amazon-codecatalyst/blueprints.blueprint'
```

The region type parameter is an array of AWS region codes to choose from, or you can use `*` to include all supported AWS regions.

Topics

- Annotations
- Region components examples

Annotations

JSDoc tags can be added to each field in the `Options` interface to customize how a field appears and behaves in the wizard. For the region type, the following tags are supported:

- The `@displayName` annotation can be used to change the field's label in the wizard.

  Example: `@displayName AWS Region`
- The `@placeholder` annotation can be used to change the select/multiselect component's placeholder.

  Example: `@placeholder Choose AWS Region`

Region components examples

Choosing a region from a specified list

```
export interface Options extends ParentOptions {
    ...
  /**
   * @displayName Region
   */
  region: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>;
}
```

Choosing one or more regions from a specified list

```
export interface Options extends ParentOptions {
    ...
  /**
   * @displayName Regions
   */
  multiRegion: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>[];
}
```

Choosing one AWS egion

```
export interface Options extends ParentOptions {
    ...
  /**
   * @displayName Region
   */
  region: Region<['*']>;
}
```

Choosing one or more regions from a specified list

```
export interface Options extends ParentOptions {
    ...
  /**
   * @displayName Regions
   */
  multiRegion: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>[];
}
```

Adding repository and source code components to a blueprint

A repository is used by Amazon CodeCatalyst to store code. The repository takes a name as an input. Most components are stored in a repository, such as source code files, workflows, and other components like managed development environments (MDE). The source repository component also exports components used for managing files and static assets. Repositories have name constraints. For more information, see Store and collaborate on code with source repositories in CodeCatalyst.

```
const repository = new SourceRepository(this, {
  title: 'my-new-repository-title',
});
```

To import Amazon CodeCatalyst blueprints repository and source code components

In your `blueprint.ts` file, add the following:

```
import {...} from '@caws-blueprint-component/caws-source-repositories'
```

Topics

- Adding a file
- Adding a generic file
- Copying files
- Targeting multiple files
- Creating a new repository and adding files

Adding a file

You can write a text file to a repository with the `SourceFile` construct. The operation is one of the most common use cases and takes a repository, a filepath, and text contents. If the file path doesn't exist within a repository, the component creates all the required folders.

```
new SourceFile(repository, `path/to/my/file/in/repo/file.txt`, 'my file contents');
```

> **Note**
>
> If you write two files to the same location within the same repository, the most recent implementation overwrites the previous one. You can use the feature to layer generated code, and it's especially useful for extending over the code that the custom blueprints may have generated.

Adding a generic file

You can write arbitrary bits to your repository. You can read from a buffer and use the `File` construct.

```
new File(repository, `path/to/my/file/in/repo/file.img`, new Buffer(...));

new File(repository, `path/to/my/file/in/repo/new-img.img`, new StaticAsset('path/to/
image.png').content());
```

Copying files

You can get started with generated code by copying and pasting starter code and then generating more code on top of that base. Place the code inside the `static-assets` directory, and then target that code with the `StaticAsset` construct. The path in this case always begins at the root of the `static-assets` directory.

```
const starterCode = new StaticAsset('path/to/file/file.txt')
const starterCodeText = new StaticAsset('path/to/file/file.txt').toString()
const starterCodeRawContent = new StaticAsset('path/to/image/hello.png').content()

const starterCodePath = new StaticAsset('path/to/image/hello.png').path()
// starterCodePath is equal to 'path/to/image/hello.png'
```

A subclass of `StaticAsset` is `SubstitutionAsset`. The subclass functions exactly the same, but instead you can run a mustache substitution over the file instead. It can be useful for performing copy-and-replace style generation.

Static asset substitution uses a mustache templating engine to render the static files that are seeded into the generated source repository. Mustache templating rules are applied during the rendering, which means that all values are HTML-encoded by default. To render unescaped HTML,

use the triple mustache syntax {{{name}}}. For more information, see the mustache templating rules.

> **Note**
>
> Running a substitute over files that aren't text-interpretable can produce errors.

```
const starterCodeText = new SubstitionAsset('path/to/file/file.txt').subsitite({
  'my_variable': 'subbed value1',
  'another_variable': 'subbed value2'
})
```

Targeting multiple files

Static assets support glob targeting through a static function on `StaticAsset` and its subclasses called `findAll(...)`, which returns a list of static assets preloaded with their paths, contents, and more. You can chain the list with `File` constructions to copy and paste contents in the `static-assets` directory.

```
new File(repository, `path/to/my/file/in/repo/file.img`, new Buffer(...));

new File(repository, `path/to/my/file/in/repo/new-img.img`, new StaticAsset('path/to/
image.png').content());
```

Creating a new repository and adding files

You can use a repository component to create a new repository in a generated project. You can then add files or workflows to the created repository.

```
import { SourceRepository } from '@amazon-codecatalyst/codecatalyst-source-
repositories';
...
const repository = new SourceRepository(this, { title: 'myRepo' });
```

The following example shows how to add files and workflows to an existing repository:

```
import { SourceFile } from '@amazon-codecatalyst/codecatalyst-source-repositories';
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows';
```

```
...
new SourceFile(repository, 'README.md', 'This is the content of my readme');
new Workflow(this, repository, {/**...workflowDefinition...**/});
```

Combining the two pieces of code generates a single repository named myRepo with a source file README.md and a CodeCatalyst workflow at the root.

Adding workflow components to a blueprint

A workflow is used by Amazon CodeCatalyst projects to run actions based on triggers. You can use workflow components to build and put together workflow YAML files. For more information, see Workflow YAML definition.

To import Amazon CodeCatalyst blueprints workflows components

In your blueprint.ts file, add the following:

```
import { WorkflowBuilder, Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'
```

Topics

- Workflow components examples
- Connecting to an environment

Workflow components examples

WorkflowBuilder component

You can use a class to build a workflow definition. The definition can be given to a workflow component for rendering in a repository.

```
import { WorkflowBuilder } from '@amazon-codecatalyst/codecatalyst-workflows'

const workflowBuilder = new WorkflowBuilder({} as Blueprint, {
  Name: 'my_workflow',
});

// trigger the workflow on pushes to branch 'main'
workflowBuilder.addBranchTrigger(['main']);

// add a build action
```

```
workflowBuilder.addBuildAction({
  // give the action a name
  actionName: 'build_and_do_some_other_stuff',

  // the action pulls from source code
  input: {
    Sources: ['WorkflowSource'],
  },

  // the output attempts to autodiscover test reports, but not in the node modules
  output: {
    AutoDiscoverReports: {
      Enabled: true,
      ReportNamePrefix: AutoDiscovered,
      IncludePaths: ['**/*'],
      ExcludePaths: ['*/node_modules/**/*'],
    },
  },
  // execute some arbitrary steps
  steps: [
    'npm install',
    'npm run myscript',
    'echo hello-world',
  ],
  // add an account connection to the workflow
  environment: convertToWorkflowEnvironment(myEnv),
});
```

Workflow Projen component

The following example shows how a Projen component can be used to write a workflow YAML to a repository:

```
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'

...

const repo = new SourceRepository
const blueprint = this;
const workflowDef = workflowBuilder.getDefinition()

// creates a workflow.yaml at .aws/workflows/${workflowDef.name}.yaml
new Workflow(blueprint, repo, workflowDef);
```

```
// can also pass in any object and have it rendered as a yaml. This is unsafe and may
 not produce a valid workflow
new Workflow(blueprint, repo, {... some object ...});
```

Connecting to an environment

Many workflows need to run in an AWS account connection. Workflows handle this by allowing actions to connect to environments with account and role name specifications.

```
import { convertToWorkflowEnvironment } from '@amazon-codecatalyst/codecatalyst-
workflows'


const myEnv = new Environment(...);

// can be passed into a workflow constructor
const workflowEnvironment = convertToWorkflowEnvironment(myEnv);


// add a build action
workflowBuilder.addBuildAction({
  ...
  // add an account connection to the workflow
  environment: convertToWorkflowEnvironment(myEnv),
});
```

Adding Dev Environments components to a blueprint

Managed development environments (MDE) are used to create and stand up MDE Workspaces in CodeCatalyst. The component generates a `devfile.yaml` file. For more information, see Introduction to Devfile and Editing a repository devfile for a Dev Environment.

```
new Workspace(this, repository, SampleWorkspaces.default);
```

To import Amazon CodeCatalyst blueprints workspaces components

In your `blueprint.ts` file, add the following:

```
import {...} from '@amazon-codecatalyst/codecatalyst-workspaces'
```

Adding issues components to a blueprint

In CodeCatalyst, you can monitor features, tasks, bugs, and any other work involved in your project. Each piece of work is kept in a distinct recordcalled an issue. Each issue can have a description, assignee, status, and other properties, which you can search for, group and filter on. You can view your issues using the default views, or you can create your own views with custom filtering, sorting, or grouping. For more information about concepts related to issues, see Issues concepts and Quotas for issues in CodeCatalyst.

The issue component generates a JSON representation of an issue. The component takes in an ID field and issue definition as input.

To import Amazon CodeCatalyst blueprints issues components

In your `blueprint.ts` file, add the following:

```
import {...} from '@amazon-codecatalyst/blueprint-component.issues'
```

Topics

- Issues components examples

Issues components examples

Creating an issue

```
import { Issue } from '@amazon-codecatalyst/blueprint-component.issues';
...
new Issue(this, 'myFirstIssue', {
  title: 'myFirstIssue',
  content: 'This is an example issue.',
});
```

Creating a high-priority issue

```
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'
...
const repo = new SourceRepository
const blueprint = this;
const workflowDef = workflowBuilder.getDefinition()
```

```
// Creates a workflow.yaml at .aws/workflows/${workflowDef.name}.yaml
new Workflow(blueprint, repo, workflowDef);

// Can also pass in any object and have it rendered as a yaml. This is unsafe and may
 not produce a valid workflow
new Workflow(blueprint, repo, {... some object ...});
```

Creating a low-priority issue with labels

```
import { Issue } from '@amazon-codecatalyst/blueprint-component.issues';
...
new Issue(this, 'myThirdIssue', {
  title: 'myThirdIssue',
  content: 'This is an example of a low priority issue with a label.',
  priority: 'LOW',
  labels: ['exampleLabel'],
});
```

Working with blueprint tooling and CLI

The blueprint CLI provides tooling to manage and work with your custom blueprints.

Topics

- Working with blueprint tooling
- Image upload tool

Working with blueprint tooling

To work with the blueprint tools

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Resume your Dev Environment. For more information, see Resuming a Dev Environment.

   If you don't have a Dev Environment, you must first create one. For more information, see Creating a Dev Environment.
3. In a working terminal, run the following the command to install the blueprint CLI:

   ```
   npm install -g @amazon-codecatalyst/blueprint-util.cli
   ```

4. In the `blueprint.ts` file, import the tools you want to use in the following format:

```
import { <tooling-function-name> } from '@amazon-codecatalyst/blueprint-util.cli/
lib/<tooling-folder-name>/<tooling-file-name>;
```

> **Tip**
>
> You can to the CodeCatalyst blueprints GitHub repository to find the name of the tooling you want to use.

If you want to use the image uploading tool, add the following to your script:

```
import { uploadImagePublicly } from '@amazon-codecatalyst/blueprint-util.cli/lib/
image-upload-tool/upload-image-to-aws';
```

Examples

- **If you want to use the publishing function, add the following to your script:**

  ```
  import { publish } from '@amazon-codecatalyst/blueprint-util.cli/lib/publish/
  publish';
  ```

- **If you want to use the image uploading tool, add the following to your script:**

  ```
  import { uploadImagePublicly } from '@amazon-codecatalyst/blueprint-util.cli/lib/
  image-upload-tool/upload-image-to-aws';
  ```

5. Call the function.

Examples:

- **If you want to use the publishing function, add the following to your script:**

  ```
  await publish(logger, config.publishEndpoint, {<your publishing options>});
  ```

- **If you want to use the image uploading tool, add the following to your script:**

  ```
  const {imageUrl, imageName} = await uploadImagePublicly(logger, 'path/to/
  image'));
  ```

Image upload tool

The image upload tool provides you with the ability to upload your own image to an S3 bucket in your AWS account and then distribute that image publicly behind CloudFront. The tool takes an image path in the local storage (and optional bucket name) as input, and returns the URL to the image that is publicly available. For more information, see What is Amazon CloudFront? and What is Amazon S3?

To work with the image upload tool

1. Clone the open-source blueprints GitHub repository that provides access to the blueprints SDK and sample blueprints. In a working terminal, run the following command:

```
git clone https://github.com/aws/codecatalyst-blueprints.git
```

2. Run the following command to navigate to the blueprints GitHub repository:

```
cd codecatalyst-blueprints
```

3. Run the following command to install dependencies:

```
yarn && yarn build
```

4. Run the following command to make sure the latest blueprint CLI version is installed:

```
yarn upgrade @amazon-codecatalyst/blueprint-util.cli
```

5. Log in to the AWS account with the S3 bucket you want to upload your image to. For more information, see Configure the AWS CLI, and Sign in through the AWS Command Line Interface.
6. Run the following command from the root of your CodeCatalyst repository to navigate to the directory with the blueprint CLI:

```
cd packages/utils/blueprint-cli
```

7. Run the following command to upload your image to an S3 bucket:

```
yarn blueprint upload-image-public <./path/to/your/image>
      <optional:optional-bucket-name>
```

A URL to your image is generated. The URL won't be available immediately since it requires some time for the CloudFront distribution to be deployed. Check the distribution status to get the latest deployment status. For more information, see Working with distributions.

Assessing interface changes with snapshot testing

Generated snapshot tests across multiple configurations of your blueprint are supported.

Blueprints support snapshot testing on configurations provided by you as a blueprint author. The configurations are partial overrides that are merged on top of the defaults.json file at the root of a blueprint. When snapshot testing is enabled and configured, the build and test process synthesizes the given configurations and verifies that the synthesized outputs haven't changed from the reference snapshot. To view the snapshot testing code, see the CodeCatalyst blueprints GitHub repository.

To enable snapshot testing

1. In the `.projenrc.ts` file, update the input object to ProjenBlueprint with the files you want to snapshot. For example:

```
{
  ....
  blueprintSnapshotConfiguration: {
    snapshotGlobs: ['**', '!environments/**', '!aws-account-to-environment/**'],
  },
}
```

2. Resynthesize the blueprint to create TypeScript files in your blueprint project. Don't edit the source files since they're maintained and regenerated by Projen. Use the following command:

```
yarn projen
```

3. Navigate to the `src/snapshot-configurations` directory to view the `default-config.json` file with an empty object. Update or replace the file with one or more of your own test configurations. Each test configuration is then merged with the project's `defaults.json` file, synthesized, and compared to snapshots when testing. Use the following command to test:

```
yarn test
```

The first time you use a test command, the following message is displayed: `Snapshot Summary › NN snapshots written from 1 test suite`. Subsequent test runs verify that the synthesized output hasn't changed from the snapshots and display the following message: `Snapshots: NN passed, NN total`.

If you intentionally change your blueprint to produce a different output, then run the following command to update the reference snapshots:

```
yarn test:update
```

Snapshots expect synthesized outputs to be constant between each run. If your blueprint generates files that vary, you must exclude those files from the snapshot testing. Update the `blueprintSnapshotConfiguration` object of your `ProjenBluerpint` input object to add the `snapshotGlobs` property. The `snapshotGlobs` property is an array of globs that determines which files are included or excluded from snapshotting.

Note

There is a default list of globs. If you specify your own list, you may need to explicitly bring back the default entries.

Publishing a custom blueprint to a space

Before you can a custom blueprint to your space's blueprints catalog, you must publish it to the space. You can also view the blueprint in the CodeCatalyst console before publishing. You can publish a preview version or a normal version of your blueprint.

Important

If you want to use blueprint packages from external sources, consider the risks that may come with those packages. You're responsible for the custom blueprints that you add to your space and the code they generate.

> **Important**
>
> To publish a custom blueprint to your CodeCatalyst space, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

Topics

Viewing and publishing a preview version of a custom blueprint

You can publish a preview version of your custom blueprint to your space if you want to add it to your space's blueprints catalog. This allows you to view the blueprint as a user before adding the non-preview version to the catalog. The preview version allows you to publish without taking up an actual version. For example, if you work on a `0.0.1` version, you can publish and add a preview version, so new updates for a second version can be published and added as `0.0.2`.

After making changes, rebuild your custom blueprint's package by running the `package.json` file, and preview your changes.

To view and publish a preview version of a custom blueprint

1. Resume your Dev Environment. For more information, see Resuming a Dev Environment
2. Open a working terminal in your Dev Environment.
3. (Optional) In a working terminal, install necessary dependencies for your project if you didn't install them already. Use the following command:

   ```
   yarn
   ```

4. (Optional) If you made changes to the `.projenrc.ts` file, regenerate the configuration of your project before building and previewing your blueprint. Use the following command:

   ```
   yarn projen
   ```

5. Rebuild and preview your custom blueprint using the following command. Use the followng command:

```
yarn blueprint:preview
```

Navigate to the `See this blueprint at:` link provided to preview your custom blueprint. Check that the UI, including text, appears as you expected based on your configuration. If you want to change your custom blueprint, you can edit the `blueprint.ts` file, resynthesize the blueprint, and then publish a preview version again. For more information, see Resynthesis.

6. (Optional) You can publish a preview version of your custom blueprint, which can then be added to your space's blueprints catalog. Navigate to the `Enable version` *`[preview version number]`* `at:` link to publish a preview version to your space.

You can emulate project creation without having to create a project in CodeCatalyst. To synthesize your project, use the following command:

```
yarn blueprint:synth
```

A blueprint is generated in the `synth/synth.`*`[options-name]`*`/proposed-bundle/` folder. For more information, see Synthesis.

If you're updating your custom blueprint, instead, use the following command to resynthesize your project:

```
yarn blueprint:resynth
```

A blueprint is generated in the `synth/synth.`*`[options-name]`*`/proposed-bundle/` folder. For more information, see Resynthesis.

After publishing your preview version, you can then add the blueprint so space members can use it to create new projects or add in existing projects. For more information, see Adding a custom blueprint to a space blueprints catalog.

Viewing and publishing a normal version of a custom blueprint

After you're done developing and previewing your custom blueprint, you can view and publish the new version that you want to add to your space's blueprints catalog. The release workflow generated when creating a project automatically publishes changes that are pushed. If you turned off the workflow generation when creating the blueprint, you're blueprint isn't automatically made

available to be added to your space's blueprints catalog. You can still publish your custom blueprint to your space after running a `yarn` command.

To view and publish a custom blueprint

1. Resume your Dev Environment. For more information, see Resuming a Dev Environment
2. Open a working terminal in your Dev Environment.
3.
   - If you opted out of the release workflow generation when creating your blueprint, use the following command:

     ```
     yarn blueprint:release
     ```

     You can still navigate to the `See this blueprint at:` link provided to view your custom blueprint.

     Publish the updated version of your custom blueprint, which can then be added to your space's blueprints catalog. Navigate to the `Enable version` *`[release version number]`* `at:` link to publish the latest version to your space.
   - If you opted in for a release workflow when creating your blueprint, the latest blueprint version is automatically published when changes are pushed. Use the following commands:

     ```
     git add .
     ```

     ```
     git commit -m "commit message"
     ```

     ```
     git push
     ```

After publishing your normal version, you can then add the blueprint so space members can use it to create new projects or add in existing projects. For more information, see Adding a custom blueprint to a space blueprints catalog.

Publishing and adding a custom blueprint in specified spaces and projects

By default, the `blueprint:preview` and `blueprint:release` commands publish into the CodeCatalyst space you created the blueprint in. If you have multiple Enterprise spaces, you can

preview and publish the same blueprint in those spaces as well. You can also add a blueprint to an existing project of another space.

To publish or add a custom blueprint in a specified space

1. Resume your Dev Environment. For more information, see Resuming a Dev Environment.
2. Open a working terminal in your Dev Environment.
3. (Optional) Install necessary dependencies for your project if you didn't install them already. Use the following command:

```
yarn
```

4. Use the `--space` tag to publish a preview or normal version to a specified space. For example:

   - ```
     yarn blueprint:preview --space my-awesome-space # publishes under a "preview"
      version tag to 'my-awesome-space'
     ```

     Example output:

     ```
     Enable version 0.0.1-preview.0 at: https://codecatalyst.aws/spaces/my-awesome-
     space/blueprints
     Blueprint applied to [NEW]: https://codecatalyst.aws/spaces/my-awesome-space/
     blueprints/%40amazon-codecatalyst%2Fmyspace.my-blueprint/publishers/1524817d-
     a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1-preview.0/projects/create
     ```

   - ```
     yarn blueprint:release --space my-awesome-space # publishes normal version to
      'my-awesome-space'
     ```

     Example output:

     ```
     Enable version 0.0.1 at: https://codecatalyst.aws/spaces/my-awesome-space/
     blueprints
     Blueprint applied to [NEW]: https://codecatalyst.aws/spaces/my-awesome-space/
     blueprints/%40amazon-codecatalyst%2Fmyspace.my-blueprint/publishers/1524817d-
     a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1/projects/create
     ```

   Use the `--project` to add a preview version of a custom blueprint to an existing project in a specified space. For example:

```
yarn blueprint:preview --space my-awesome-space --project my-project # previews
 blueprint application to an existing project
```

Example output:

```
Enable version 0.0.1-preview.1 at: https://codecatalyst.aws/spaces/my-awesome-
space/blueprints
Blueprint applied to [my-project]: https://codecatalyst.aws/spaces/my-awesome-
space/projects/my-project/blueprints/%40amazon-codecatalyst%2FmySpace.my-blueprint/
publishers/1524817d-a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1-preview.1/add
```

Viewing details, versions, and projects of a custom blueprint

You can view your space's published custom blueprints, including a blueprint's details, versions, and the projects that are created with or adding the blueprint.

Topics

- Viewing a space's custom blueprints
- Viewing projects created with or added custom blueprints

Viewing a space's custom blueprints

To view a space's custom blueprints

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to view a custom blueprint.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints** to view the **Space blueprints**. The following details are displayed in the table:

   - **Name** - Name of the custom blueprint.
   - **Catalog status** - Whether the custom blueprint is published to the space's blueprints catalog.
   - **Latest version** - The atest version of the custom blueprint.
   - **Latest modified** - The date when the space blueprint was last updated.

Viewing projects created with or added custom blueprints

To view projects created with or added custom blueprints

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to view a custom blueprint.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.
4. From the **Space blueprints** table, choose the name of a custom blueprint to view the **Projects using blueprint** and **Projects not using blueprint** tables.

Adding a custom blueprint to a space blueprints catalog

After you publish a custom blueprint to your space, it can be added to your space's blueprints catalog. If you add a custom blueprint to your CodeCatalyst space's blueprints catalog, then the blueprint is available to all space members to use when creating a project or adding it to an existing project. Before adding a custom blueprint to the space's blueprints catalog, the blueprint's publishing permission must be enabled. If you opted in for workflow release generation, then publishing permissions are enabled by default. For more information, see Setting publishing permissions for a custom blueprint and Publishing a custom blueprint to a space.

> **Important**
>
> To add a custom blueprint to your CodeCatalyst space's blueprints catalog, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

To add a blueprint to the space's blueprints catalog

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. The blueprint can only be added from the default branch of the source repository. If you developed the blueprint on a feature branch, merge your feature branch with the changes to the default branch. Create a pull request to merge any changes to the default branch. For more information, see Reviewing code with pull requests in Amazon CodeCatalyst.
3. In the CodeCatalyst console, navigate to the space dashboard with your custom blueprint.
4. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.

5. Choose the blueprint name you want to add, and then choose **Add to catalog**. If you have more than one version, choose a version from the **Catalog version** dropdown menu
6. Choose **Save**.

Removing a custom blueprint from a space blueprints catalog

A custom blueprint can be removed from your space's blueprints catalog if you no longer want it being used to create new projects or applied to existing projects.

> **Note**
>
> If you remove a custom blueprint from a space's blueprints catalog, it doesn't affect a project created from the blueprint or a project that applied the blueprint. Resources of the blueprint aren't removed from the project.

> **Important**
>
> To remove a custom blueprint from your CodeCatalyst space's blueprints catalog, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

To remove a custom blueprint from a space blueprints catalog

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space dashboard with your custom blueprint
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.
4. Choose the blueprint name you want to remove, and then choose **Remove blueprint from catalog**.

Setting publishing permissions for a custom blueprint

By default, a custom blueprint's permission is enabled if a workflow release was generated during project creation. When publishing permissions are enabled, the blueprint can be published to the space. You can disable the permission so that the blueprint can't be published. When the

permission is disabled, the release workflow that is generated during blueprint creation doesn't run. New changes to a blueprint can't be published unless the blueprint's permissions are enabled.

> **Important**
>
> To enable or disable a custom blueprint project's publishing permissions, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

To set a blueprint project's publishing permissions

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to manage a custom blueprint's publishing permissions.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.
4. Choose the **Project publishing permissions** tab to view the publishing permissions for all your space's blueprints.
5. Choose the blueprint that you want to manage, and then choose **Enable** or **Disable** to change the publishing permissions. If you're enabling the permissions, review the permission change details, and then choose **Enable blueprint publishing** to confirm the change.

Changing catalog versions for a custom blueprint

As a blueprint author, you can manage the version that you want to publish to the space's blueprints catalog. Changing the catalog version of a blueprint doesn't affect the projects that are using a different blueprint version.

> **Important**
>
> To change catalog versions for a custom blueprint in your Amazon CodeCatalyst space's blueprints catalog, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

To manage a custom blueprint version

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to change a custom blueprint's version.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.
4. On **Space blueprints** table, choose the radio button for the custom blueprint that you want to manage.
5. Choose **Make catalog version**, and then choose the a version from the **Catalog version** dropdown menu.
6. Choose **Save**.

Deleting a published custom blueprint or version

When you delete a custom blueprint's version or the blueprint itself from your Amazon CodeCatalyst space, all your access is removed to the resources of the blueprint project or blueprint version. When you have deleted a blueprint version or the blueprint, project members will be unable to access project resources, and any workflows that are prompted by third-party source repositories will be stopped.

> **Note**
>
> If you delete a blueprint, it doesn't affect a project that is applied the blueprint. Resources of the blueprint aren't removed from the project.

If a blueprint version is published to the space's blueprint catalog, choose a new version for the catalog before you delete the published version.

> **Important**
>
> To delete a published custom blueprint or a custom blueprint's catalog version from your space, you must be signed in with an account that has the **Space administrator** or **Power user** role in the space.

To delete a custom blueprint's catalog version

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. In the CodeCatalyst console, navigate to the space where you want to delete a custom blueprint's catalog version.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.
4. Choose the name of the blueprint with the catalog version that you want to delete.
5. Choose the radio button for the catalog version that you want to delete, and then choose **Delete version**.
6. Review the details, and then choose another blueprint version from the **Choose a new blueprint catalog version** dropdown menu.
7. Enter `delete` to confirm the deletion of the blueprint catalog version.
8. Choose **Delete**.

If a blueprint version isn't in the space's blueprints catalog, you can delete the version without choosing a new version.

To delete a custom blueprint version

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to delete a custom blueprint version.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.
4. Choose the name of the blueprint with the version that you want to delete.
5. Choose the radio button for the version that you want to delete, and then choose **Delete version**.
6. Enter `delete` to confirm the blueprint version deletion.
7. Choose **Delete**.

Deleting a blueprint from the space's blueprints catalog deletes all versions of the blueprint. The space's projects that are using the blueprint aren't affected by the deletion.

To delete a custom blueprint version

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the CodeCatalyst console, navigate to the space where you want to delete a custom blueprint.
3. On the space dashboard, choose the **Settings** tab, and then choose **Blueprints**.

4. On the **Space blueprints** table, choose the radio button for the custom blueprint that you want to delete, and then choose **Delete blueprint**.
5. Enter `delete` to confirm the custom blueprint deletion.
6. Choose **Delete**.

Handling dependencies, mismatches, and tooling

Topics

Adding dependencies

As a blueprint author, you might need to add packages to your blueprint, such as `@amazon-codecatalyst/blueprint-component.environments`. You need to update the `projen.ts` file with that package, and then regenerate the configuration of your project with Projen. Projen acts as the project model for each blueprint codebase, which provides the ability to push backwards-compatible tooling updates by changing the way that the model renders configuration files. The `package.json` file is a file that is partially owned by the Projen model. Projen acknowledges dependency versions included in the package.json file, but other options need to originate from the model.

To add a dependency and update a `projenrc.ts` file

1. In the `projen.ts` file, navigate to the deps section.
2. Add the dependency you want to use in your blueprint.
3. Use the following command to regenerate the configuration of your project:

```
yarn projen && yarn
```

Handling dependency type mismatches

After a Yarn update, you might get the following error regarding a repository parameter:

```
Type 'SourceRepository' is missing the following properties from type
 'SourceRepository': synthesisSteps, addSynthesisStep
```

The error is due a dependency mismatch that happens when one component relies on a newer version of another component, but the relying component is pinned to an older version. The error can be fixed by making all your components rely on the same version so that the version synchronized between them. It's best to keep al blueprint-vended packages under the same latest version (`0.0.x`), unless you're certain about how you're handling the versions. The following example shows how the `package.json` file can be configured so all the dependencies rely on the same version:

```
...
"@caws-blueprint-component/caws-environments": "^0.1.12345",
"@caws-blueprint-component/caws-source-repositories": "^0.1.12345",
"@caws-blueprint-component/caws-workflows": "^0.1.12345",
"@caws-blueprint-component/caws-workspaces": "^0.1.12345",
"@caws-blueprint-util/blueprint-utils": "^0.1.12345",
...
"@caws-blueprint/blueprints.blueprint": "*",
```

After configuring the versions for all dependencies, use the following command:

```
yarn install
```

Using yarn and npm

Blueprints use Yarn for tooling. Using npm and Yarn will cause tooling problems because the way that dependency trees are resolved by each is different. To avoid such issues, it's best to use Yarn only.

If you accidentally installed dependencies using npm, you can remove the generated `package-lock.json` file, and make sure your `.projenrc.ts` file is updated with the dependencies you need. You regenerate the configuration of your project with Projen.

Use the following to regenerate from the model:

```
yarn projen
```

After making sure your .projenrc.ts file is updated with the necessary dependencies, use the following command:

```
yarn
```

Upgrading tooling and components

Occasionally, you might want to upgrade your tooling and components to bring in new features available. You're recommended to keep all the components on the same version unless you're certain about how you're handling the versions. Versions are synchronized between components, so the same versions for all components ensures proper dependency between them.

Using Yarn workspace monorepo

Use the following command to upgrade utils and components from the root of a custom blueprint's repository:

```
yarn upgrade @amazon-codecatalyst/*
```

Use the following command if you're not using a monorepo:

```
yarn upgrade –pattern @amazon-codecatalyst/*
```

Other options you can use to upgrade tooling and components:

- Use npm view `@caws-blueprint-component/<some-component>` to get the latest version.
- Manually increase to the latest version by setting the version in your package.json file and using the following command:`yarn`. All components and utils should have the same version.

Contribute

The blueprints software development kit (SDK) is an open-source library that you can contribute to. As a contributor, consider the contribution guidelines, feedback, and defects. For more information, see the blueprints GitHub repository.

Quotas for blueprints in CodeCatalyst

The following table describes quotas and limits for blueprints in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Maximum number of blueprints applied per CodeCatalyst project	100

Store and collaborate on code with source repositories in CodeCatalyst

CodeCatalyst source repositories are Git repositories hosted in Amazon CodeCatalyst. You can use source repositories in CodeCatalyst to securely store, version, and manage assets for a project.

Assets in a CodeCatalyst repository can include:

- documents
- source code
- binary files

CodeCatalyst also uses the source repository for a project to store configuration information for your project, such as workflow configuration files.

You can have more than one source repository in a CodeCatalyst project. For example, you might want to have separate source repositories for front-end source code, back-end source code, utilities, and documentation.

Here is one possible workflow for working with code in source repositories, pull requests, and Dev Environments in CodeCatalyst:

Mary Major creates a web application project in CodeCatalyst using a blueprint, which creates a source repository with sample code in it. She invites her friends Li Juan, Saanvi Sarkar, and Jorge Souza to work on the project with her. Li Juan looks at the sample code in the source repository and decides to make some quick changes to add a test to the code. Li creates a Dev Environment, chooses AWS Cloud9 as the IDE, and specifies a new branch, `test-code`. The Dev Environment opens. Li quickly adds the code, then commits and pushes the branch with the changes to the source repository in CodeCatalyst. Li then creates a pull request. As part of creating that pull request, Li adds Jorge Souza and Saanvi Sarkar as reviewers to ensure that the code is reviewed.

While reviewing the code, Jorge Souza remembers that he has his own project repository on GitHub that contains a prototype of the app they're working on. He asks Mary Major to install and configure the extension that will allow him to link the GitHub repository to the project as an additional source repository. Mary reviews the repository on GitHub and works with Jorge to configure the GitHub extension so that he can link the GitHub repository as an additional source repository for the project.

CodeCatalyst source repositories support the standard functionality of Git and work with your existing Git-based tools. You can create and use personal access tokens (PATs) as an application-specific password when cloning and working with source repositories from a Git client or integrated development environments (IDEs). These PATs are associated with your CodeCatalyst user identity. For more information, see Grant users repository access with personal access tokens.

CodeCatalyst source repositories support pull requests. This is a simple way for you and other project members to review and comment on code changes before you merge them from one branch to another. You can view the changes in the CodeCatalyst console and comment on lines of code.

Pushes to branches in a CodeCatalyst source repository can automatically start a run in a workflow, where changes can be built, tested, and deployed. If your source repository was created as part of a project using a project template, one or more workflows are configured for you as part of the project. You can add additional workflows for repositories at any time. The YAML configuration files for workflows in a project are stored in the source repositories configured in the source action for those workflows. For more information, see Getting started with workflows.

Topics

Source repository concepts

Here are some concepts to know about as you work with CodeCatalyst source repositories.

Topics

- Source repositories
- Dev Environments
- Personal access tokens (PATs)
- Branches
- Default branches
- Commits
- Pull requests
- Revisions
- Workflows

Projects

A *project* represents a collaborative effort in CodeCatalyst that supports development teams and tasks. After you have a project, you can add, update, or remove users and resources, customize your project dashboard, and monitor the progress of your team's work. You can have multiple projects within a space.

Source repositories are specific to the projects where you create or link them in a space. You can't share a repository between projects, and you can't link a repository to more than one project in a space. Users with the **Contributor** or **Project administrator** role in a project can interact with the source repositories associated with that project according to the permissions granted to those roles. For more information, see Granting access with user roles.

Source repositories

A *source repository* is where you securely store code and files for your project. It also stores the version history of your files. By default, a source repository is shared with the other users in your CodeCatalyst project. You can have more than one source repository for a project. You can create source repositories for projects in CodeCatalyst, or you can choose to link an existing source repository hosted by another service if that service is supported by an installed extension. For example, you can link a GitHub repository to a project after you install the **GitHub Repositories** extension. For more information, see Storing source code in repositories for a project in CodeCatalyst and Quickstart: Installing extensions, connecting providers, and linking resources in CodeCatalyst.

Dev Environments

A *Dev Environment* is a cloud-based development environment that you can use in CodeCatalyst to quickly work on the code stored in the source repositories of your project. The project tools and application libraries included in your Dev Environment are defined by a devfile in the source repository of your project. If you do not have a devfile in your source repository, a default devfile will be applied automatically. The default devfile includes tools for the most frequently used programming languages and frameworks. By default, a Dev Environment is configured to have a 2-core processor, 4 GB of RAM, and 16 GiB of persistent storage.

You can choose to clone an existing branch of your source repository into your Dev Environment, or you can choose to create a new branch as part of creating the Dev Environment.

Personal access tokens (PATs)

A *personal access token* (PAT) is similar to a password. It is associated with your user identity for use across all spaces and projects in CodeCatalyst. You use PATs to access CodeCatalyst resources that include integrated development environments (IDEs) and Git-based source repositories. PATs represent you in CodeCatalyst and you can manage them in your user settings. A user can have more than one PAT. Personal access tokens only display once. As a best practice, be sure to store them securely on your local computer. By default, PATs expire after one year.

When working with integrated development environments (IDEs), PATs are the equivalent of a Git password. Provide the PAT when asked for a password when setting up your IDE to work with a Git repository. For more information about how to connect your IDE with a Git-based repository, see the documentation for your IDE.

Branches

A *branch* is a pointer or reference to a commit in Git and in CodeCatalyst. You can use branches to organize your work. For example, you can use branches to work on a new or different version of files without affecting files in other branches. You can use branches to develop new features, store a specific version of your project, and more. A source repository can have one branch or many branches. When you create a project using a template, the source repository created for the project contains sample files in a branch called **main**. The **main** branch is the default branch for the repository.

Default branches

Source repositories in CodeCatalyst have a default branch regardless of how you create them. If you choose to create a project using a template, the source repository created for that project includes a README.md file in addition to sample code, workflow definitions, and other resources. If you create a source repository without using a template, a README.md file is added for you as a first commit and a default branch is created for you as part of creating the repository. This default branch is named *main*. This default branch is the one used as the base or default branch in local repositories (repos) when users clone the repository. You can change which branch is used as the default branch. For more information, see Managing the default branch for a repository.

You can't delete the default branch for a source repository. Search results only include results from the default branch.

Commits

A *commit* is a change to a file or set of files. In the Amazon CodeCatalyst console, a commit saves your changes and pushes them to a source repository. The commit includes information about the change, including the identity of the user who made the change, the time and date of the change, the commit title, and any message included about the change. For more information, see Understanding changes in source code with commits in Amazon CodeCatalyst.

In the context of a source repository in CodeCatalyst, commits are snapshots of the contents and changes to the contents of your repository. You can also add Git tags to commits, to identify specific commits.

Pull requests

A *pull request* is the primary way you and other users review, comment on, and merge code changes from one branch to another in a source repository. You can use pull requests to review code changes collaboratively for minor changes or fixes, major feature additions, or new versions of your released software. In a pull request, you can review the changes between the source and destination branches or the differences between revisions of those branches. You can add comments to individual lines of code changes as well as comments on the pull request overall.

> **Tip**
>
> While you are creating a pull request, the difference displayed is the difference between the tip of the source branch and the tip of the destination branch. Once the pull request

has been created, the displayed difference will be between the revision of the pull request you choose and the commit that was the tip of the destination branch when you created the pull request. For more information about differences and merge bases in Git, see git-merge-base in the Git documentation.

Revisions

A *revision* is an updated version of a pull request. Each push to the source branch of a pull request creates a revision that contains the changes made in the commits included in that push. You can view the differences between revisions of a pull request in addition to the differences between the source and destination branches. For more information, see Reviewing code with pull requests in Amazon CodeCatalyst.

Workflows

A *workflow* is an automated procedure that describes how to build, test, and deploy your code as part of a continuous integration and continuous delivery (CI/CD) system. A workflow defines a series of steps, or *actions*, to take during a workflow run. A workflow also defines the events, or *triggers*, that cause the workflow to start. To set up a workflow, you create a *workflow definition file* using the CodeCatalyst console's visual or YAML editor.

Tip

For a quick look at how you might use workflows in a project, create a project with a blueprint. Each blueprint deploys a functioning workflow that you can review, run, and experiment with.

A source repository can also store the configuration files and other information for workflows, notifications, issues, and other configuration information for the project. The configuration files are created and stored in the source repository when you create resources that require configuration files, or when you specify the repository as a source action for a workflow. If you create a project from a blueprint, you will have configuration files already stored in the source repository created for you as part of the project. This configuration information is stored in a folder named `.codecatalyst` in the default branch of your repository. Whenever you create a branch of the default branch, you create a copy of this folder and its configuration in addition to all the other files and folders in that branch.

Setting up for working with source repositories

When you work with source repositories in Amazon CodeCatalyst on your local machine, you can use Git either on its own or in a supported integrated development environment (IDE) to make code changes and push and pull your code. As a best practice, we recommend that you use the latest versions of Git and other software.

> **Note**
>
> If you use Dev Environments, you do not have to install Git. A recent version of Git is included in your Dev Environment.

Version compatibility information for CodeCatalyst

Component	Version
Git	latest

Install Git

To work with files, commits, branches, and other information in source repositories from a Git client without an IDE, install Git on your local machine.

To install Git, we recommend websites such as Git Downloads.

Create a personal access token

To clone source repositories to your local machine or to your preferred IDE, you must create a personal access token (PAT).

To create a personal access token (PAT)

1. In the top menu bar, choose your profile badge, and then choose **My settings**.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

2. In **PAT name**, enter a descriptive name for your PAT.
3. In **Expiration date**, leave the default date or choose the calendar icon to select a custom date. The expiration date defaults to one year from the current date.
4. Choose **Create**.

   You can also create this token when you choose **Clone repository** for a source repository.
5. Save the PAT secret in a secure location.

> **Important**
>
> The PAT secret only displays once. You cannot retrieve it after you close the window.

Getting started with CodeCatalyst source repositories and the Single-page application blueprint

Follow the steps in this tutorial to learn how to work with source repositories in Amazon CodeCatalyst.

The quickest way to get started working with source repositories in Amazon CodeCatalyst is to create a project using a template. When you create a project using a template, resources are created for you, including a source repository that includes sample code. You can use this repository and code example to learn how to:

- View a project's source repositories and browse their contents
- Create a Dev Environment with a new branch where you can work on code
- Change a file, and commit and push your changes
- Create a pull request and review your code changes with other project members
- See the workflow for your project automatically build and test the changes in the source branch of the pull request
- Merge your changes from your source branch into the destination branch and close the pull request
- See the merged changes automatically built and deployed

To get the most out of this tutorial, invite others to your project so you can work together on a pull request. You can also explore additional features in CodeCatalyst, such as creating issues and associating them with a pull request, or configuring notifications and getting alerts when the associated workflow runs. For a full exploration of CodeCatalyst, see Getting started tutorials.

Creating a project with a blueprint

Creating a project is the first step in being able to work together. You can use a blueprint to create your project, which will also create a source repository with sample code and a workflow that will automatically build and deploy your code when you change it. In this tutorial, we'll walk you through a project created with the **Single-page application** blueprint, but you can follow the procedures for any project with a source repository. Make sure to choose an IAM role or add an IAM role if you don't have one as part of creating the project. We recommend that you use the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role for this project.

If you already have a project, you can skip ahead to Viewing the repositories for a project.

> **Note**
>
> Only users with the Space administrator or **Power user** role can create projects in CodeCatalyst. If you do not have this role and you need a project to work on for this tutorial, ask someone with one of these roles to create a project for you and add you to the created project. For more information, see Granting access with user roles.

To create a project with a blueprint

1. In the CodeCatalyst console, navigate to the space where you want to create a project.
2. On the space dashboard, choose **Create project**.
3. Choose **Start with a blueprint**.

> **Tip**
>
> You can choose to add a blueprint by giving **Amazon Q** your project requirements to have Amazon Q suggest a blueprint to you. For more information, see Using Amazon Q to choose a blueprint when creating a project or adding functionality and Best practices when using Amazon Q to create projects or add functionality with blueprints. This feature is only available in the US West (Oregon) Region.

This functionality requires that generative AI features are enabled for the space. For more information, see Managing generative AI features.

4. From the **CodeCatalyst blueprints** or **Space blueprints** tab, choose a blueprint, and then choose **Next**.
5. Under **Name your project**, enter the name that you want to assign to your project and its associated resource names. The name must be unique within your space.
6. (Optional) By default, the source code created by the blueprint is stored in a CodeCatalyst repository. Alternatively, you can choose to store the blueprint's source code in a third-party repository. For more information, see Add functionality to projects with extensions in CodeCatalyst.

   Important

   CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
   As a best practice, always make sure you have the latest version of the extension before you link a repository.

   Do one of the following depending on the third-party repository provider you want to use:

   - **GitHub repositories**: Connect a GitHub account.

     Choose the **Advanced** dropdown menu, choose GitHub as the repository provider, and then choose the GitHub account where you want to store the source code created by the blueprint.

     Note

     If you're connecting a GitHub account, you must create a personal connection to establish identity mapping between your CodeCatalyst identity and your GitHub identity. For more information, see Personal connections and Accessing GitHub resources with personal connections.

- **Bitbucket repositories**: Connect a Bitbucket workspace.

  Choose the **Advanced** dropdown menu, choose Bitbucket as the repository provider, and then choose the Bitbucket workspace where you want to store the source code created by the blueprint.

- **GitLab repositories**: Connect a GitLab user.

  Choose the **Advanced** dropdown menu, choose GitLab as the repository provider, and then choose the GitLab user where you want to store the source code created by the blueprint.

7. Under **Project resources**, configure the blueprint parameters. Depending on the blueprint, you may have the option to name the source repository name.
8. (Optional) To view definition files with updates based on the project parameter selections you made, choose **View code** or **View workflow** from **Generate project preview**.
9. (Optional) Choose **View details** from the blueprint's card to view specific details about the blueprint, such as an overview of the blueprint's architecture, required connections and permissions, and the kind of resources the blueprint creates.
10. Choose **Create project**.

The project overview page opens as soon as you create a project or accept an invitation to a project and complete the sign-in process. The project overview page for a new project contains no open issues or pull requests. You can optionally choose to create an issue and assign it to yourself. You can also choose to invite someone else to your project. For more information, see Creating an issue in CodeCatalyst and Inviting a user to a project.

Viewing the repositories for a project

As a member of a project, you can view the source repositories for the project. You can also choose to create additional repositories. If someone with the **Space administrator** role has installed and configured the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension, you can also add links to third-party repositories in the GitHub accounts, Bitbucket workspaces, or GitLab users configured for the extension. For more information, see Creating a source repository and Quickstart: Installing extensions, connecting providers, and linking resources in CodeCatalyst.

> **Note**
>
> For projects created with the Single-page application blueprint, the default name for the source repository that contains the sample code is ***`spa-app`***.

To navigate to the source repositories for a project

1. Navigate to your project, and do one of the following:
   - On the summary page for your project, choose the repository you want from the list, and then choose **View repository**.
   - In the navigation pane, choose **Code**, and then choose **Source repositories**. In **Source repositories**, choose the name of the repository from the list. You can filter the list of repositories by typing part of the repository name in the filter bar.
2. On the home page for the repository, view the contents of the repository and information about the associated resources such as the number of pull requests and workflows. By default, the contents for the default branch are shown. You can change the view by choosing a different branch from the drop-down list.

The overview page for the repository includes information about the workflows and pull requests that are configured for the branches of this repository and its files. If you just created the project, the initial workflows to build, test, and deploy the code will still be running, as they take a few minutes to complete. You can view the related workflows and their status by choosing the number beneath **Related workflows**, but this opens the **Workflows** page in **CI/CD**. For this tutorial, stay on the overview page and explore the code in the repository. The contents of the `README.md` file are rendered on this page below the repository files. In **Files**, the contents of the default branch are shown. You can change the file view to show the contents of another branch if you have one. The `.codecatalyst` folder contains code used for other parts of the project, such as workflow YAML files.

To view the content of folders, choose the arrow next to the folder name to expand it. For example, choose the arrow next to `src` to view the files for the single-page web application contained in that folder. To view the contents of a file, choose it from the list. This will open **View files**, where you can browse the contents of multiple files. You can edit single files in the console as well, but to edit multiple files, you'll want to create a Dev Environment.

Creating a Dev Environment

You can add and change files in a source repository in the Amazon CodeCatalyst console. However, to work effectively with multiple files and branches, we recommend using a Dev Environment or cloning the repository to your local computer. In this tutorial, we'll create an AWS Cloud9 Dev Environment with a branch named **`develop`**. You can choose a different branch name, but by naming the branch **`develop`**, a workflow will automatically run to build and test your code when you create a pull request later in this tutorial.

> **Tip**
>
> If you decide to clone a repository locally instead of or in addition to using a Dev Environment, make sure that you have Git on your local computer or that your IDE includes Git. For more information, see Setting up for working with source repositories.

To create a Dev Environment with a new branch

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to create a Dev Environment.
3. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, choose **Source repositories**, and choose the repository for which you want to create a Dev Environment.
4. On the repository home page, choose **Create Dev Environment**.
5. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.
6. Choose the repository to clone, choose **Work in new branch**, enter a branch name into the **Branch name** field, and choose a branch off of which to create the new branch from the **Create branch from** drop-down menu.
7. Optionally, add an alias for the Dev Environment.
8. Optionally, choose the **Dev Environment configuration** edit button to edit the Dev Environment's compute, storage, or timeout configuration.
9. Choose **Create**. While your Dev Environment is being created, the Dev Environment status column will display **Starting**, and the status column will display **Running** once the Dev Environment has been created. A new tab will open with your Dev Environment in the IDE of your choice. You can edit code and commit and push your changes.

Once you've created the Dev Environment, you can edit files, commit your changes, and push your changes to the **`test`** branch. For this tutorial, edit the content between the <p> tags in `App.tsx` file in the `src` folder to change the text that's displayed on the webpage. Commit and push your change, and then return to the CodeCatalyst tab.

To make and push a change from an AWS Cloud9 Dev Environment

1. In AWS Cloud9, expand the side navigation menu to browse the files. Expand `src`, and open `App.tsx`.
2. Make a change the text inside the <p> tags.
3. Save the file, and then commit and push your changes by using the Git menu. Alternatively, in the terminal window, commit and push your changes with the **git commit** and **git push** commands.

```
git commit -am "Making an example change"
git push
```

> **Tip**
>
> You might need to change directories in the terminal to the Git repository directory before you can successfully run the Git commands.

Creating a pull request

You can use pull requests to review code changes collaboratively for minor changes or fixes, major feature additions, or new versions of your released software. In this tutorial, you will create a pull request to review the changes you made to the *`test`* branch compared to the **main** branch. Creating a pull request in a project created with a template will also start a run of its associated workflows, if any.

To create a pull request

1. Navigate to your project.
2. Do one of the following:
   - In the navigation pane, choose **Code**, choose **Pull requests**, and then choose **Create pull request**.

- On the repository home page, choose **More**, and then choose **Create pull request**.
- On the project page, choose **Create pull request**.

3. In **Source repository**, make sure that the specified source repository is the one that contains the committed code. This option only appears if you did not create the pull request from the repository's main page.
4. In **Destination branch**, choose the branch to merge the code into after it is reviewed.
5. In **Source branch**, choose the branch that contains the committed code.
6. In **Pull request title**, enter a title that helps other users understand what needs to be reviewed and why.
7. (Optional) In **Pull request description**, provide information such as a link to issues or a description of your changes.

> **Tip**
>
> You can choose **Write description for me** to have CodeCatalyst automatically generate a description of the changes contained in the pull request. You can make changes to the automatically generated description after you add it to the pull request.
> This functionality requires that generative AI features are enabled for the space and is not available for pull requests in linked repositories. For more information, see Managing generative AI features.

8. (Optional) In **Issues**, choose **Link issues**, and then either choose an issue from the list or enter its ID. To unlink an issue, choose the unlink icon.
9. (Optional) In **Required reviewers**, choose **Add required reviewers**. Choose from the list of project members to add them. Required reviewers must approve the changes before the pull request can be merged into the destination branch.

> **Note**
>
> You cannot add a reviewer as both a required reviewer and an optional reviewer. You cannot add yourself as a reviewer.

10. (Optional) In **Optional reviewers**, choose **Add optional reviewers**. Choose from the list of project members to add them. Optional reviewers do not have to approve the changes as a requirement before the pull request can be merged into the destination branch.

11. Review the differences between the branches. The difference displayed in a pull request is the changes between the revision in the source branch and the merge base, which is the head commit of the destination branch at the time the pull request was created. If no changes display, the branches might be identical, or you might have chosen the same branch for both the source and the destination.
12. When you are satisfied that the pull request contains the code and changes you want reviewed, choose **Create**.

> **Note**
>
> After you create the pull request, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resources, such as files, by using the @ sign followed by the name of the file.

You can view information about associated workflows started by the creation of this pull request by choosing **Overview** and then reviewing the information in the **Pull request details** area under **Workflow runs**. To view the workflow run, choose the run.

> **Tip**
>
> If you named your branch something other than **`develop`**, a workflow will not automatically run to build and test your changes. If you want to configure that, edit the YAML file for the **onPullRequestBuildAndTest** workflow. For more information, see Creating a workflow.

You can comment on this pull request and ask other project members to comment on it. You can also choose to add or change optional or required reviewers. You can choose to make more changes to the source branch for the repository, and see how those committed changes create revisions for the pull request. For more information, see Reviewing a pull request, Updating a pull request, Reviewing code with pull requests in Amazon CodeCatalyst, and Viewing workflow run status and details .

Merging a pull request

Once a pull request has been reviewed and has received approvals from required reviewers, you can merge its source branch to the destination branch in the CodeCatalyst console. Merging a pull request will also start a run of the changes through any workflows associated with the destination branch. In this tutorial, you'll merge the test branch into main, which will start a run of the **onPushToMainDeployPipeline** workflow.

To merge a pull request (console)

1. In **Pull requests**, choose the pull request you created in the previous step. In the pull request, choose **Merge**.
2. Choose from the available merge strategies for the pull request. Optionally select or deselect the option to delete the source branch after merging the pull request, and then choose **Merge**. After the merge completes, the status of the pull request changes to **Merged** and no longer appears in the default view of pull requests. The default view shows pull requests with a status of **Open**. You can still view a merged pull request, but you cannot approve it or change its status.

> **Note**
>
> If the **Merge** button is not active, or you see the **Not mergeable** label, either a required reviewer has not yet approved the pull request, or the pull request cannot be merged in the CodeCatalyst console. A reviewer who has not approved a pull request is indicated by a clock icon in **Overview** in the **Pull request details** area. If all required reviewers have approved the pull request but the **Merge** button is still not active, you might have a merge conflict, or you have exceeded the storage quota for the space. You can resolve merge conflicts for the destination branch in a Dev Environment, push the changes, and then merge the pull request, or you can resolve conflicts and merge locally, and then push the commit that contains the merge to CodeCatalyst. For more information, see Merging a pull request (Git) and your Git documentation.

Viewing the deployed code

Now it's time to view the originally deployed code that was in the default branch, and your merged changes once they are automatically built, tested, and deployed. To do so, you can return to the

overview page for the repository and choose the number next to the related workflows icon, or in the navigation pane, choose **CI/CD**, and then choose **Workflows**.

To view the deployed code

1. In **Workflows**, in `onPushToMainDeployPipeline`, expand **Recent runs**.

   > **Note**
   >
   > This is the default name of the workflow for projects created with the **Single-page application** blueprint.

2. The most recent run is the one started by your merged pull request commit to the `main` branch and will likely show a status of **In progress**. Choose a successfully completed run from the list to open the details of that run.
3. Choose **Variables**. Copy the value for **AppURL**. This is the URL for the deployed single page web application. Open a new browser tab and paste in the value to view the built and deployed code. Leave the tab open.
4. Return to the list of workflow runs and wait for the most recent run to complete. When it does, return to the tab you opened to view the web application and refresh your browser. You should see the changes that you made in your merged pull request.

Cleaning up resources

Once you've explored working with a source repository and pull request, you might want to remove any resources you don't need. You cannot delete pull requests, but you can close them. You can delete any branches you created.

If you no longer need the source repository or the project, you can also delete those resources. For more information, see Deleting a source repository and Deleting a project.

Storing source code in repositories for a project in CodeCatalyst

A source repository is where you securely store code and files for your project. It also stores your source history, from the first commit through the latest changes. If you choose a blueprint that includes a source repository, that repository also contains the configuration files and other information for workflows and notifications for the project. This configuration information is stored in a folder named **.codecatalyst**.

You can create a source repository in CodeCatalyst either by creating a project with a blueprint that creates a source repository as part of creating a project, or by creating a source repository in an existing project. Project users will automatically see and be able to use the repositories you create for a project. You can also choose to link a Git repository hosted on GitHub, Bibucket, or GitLab to your project. When you do so, your project users can view and access that linked repository in the list of repositories for the project.

> **Note**
>
> Before you can link the repository, you must install the extension for the service that hosts it. You cannot link an archived repository. While you can link an empty repository, you can't use it in CodeCatalyst until you have initialized it with an initial commit that creates a default branch. For more information, see Installing an extension in a space.

By default, a source repository is shared with other members of your Amazon CodeCatalyst project. You can create additional source repositories for a project or link repositories to the project. All members of a project can view, add, edit, and delete files and folders in the project's source repositories.

To quickly work on code in a source repository, you can create a Dev Environment that clones a specified repository and branch into it where you can work on the code in the integrated development environment (IDE) you choose for the Dev Environment. You can clone a source repository on your local computer and pull and push changes between your local repo and the remote repository in CodeCatalyst. You can also work with source repositories by configuring access to them in your preferred IDE as long as that IDE supports credential management.

Repository names must be unique within a CodeCatalyst project.

Topics

- Deleting a source repository

Creating a source repository

When you create a project using a blueprint in Amazon CodeCatalyst, CodeCatalyst creates a source repository for you. That source repository contains sample code in addition to configuration information for the workflows and other resources created for you. This is the recommended way to get started with repositories in CodeCatalyst. You can choose to create repositories for a project. Those repositories will contain a **README.md** file that you can edit or delete at any time. Depending on your choices when you create a source repository, repositories might also contain a `.gitignore` file.

If you want to clone an existing Git repository into a CodeCatalyst source repository, consider creating an empty repository instead. This repository will be unavailable for use in CodeCatalyst until you add content to it, which you can do with a few simple Git commands. Alternatively, you can add content to the empty repository directly from the CodeCatalyst console. Alternatively, you can link a source repository in a supported Git repository provider. For more information, see Linking a source repository.

To create a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose **Add repository**, and then choose **Create repository**.
5. In **Repository name**, provide a name for the repository. In this guide, we use *`codecatalyst-source-repository`*, but you can choose a different name. Repository names must be unique within a project. For more information about requirements for repository names, see Quotas for source repositories in CodeCatalyst.
6. (Optional) In **Description**, add a description for the repository that will help other users in the project understand what the repository is used for.
7. Choose **Create repository (default)**. This option creates a repository that includes a default branch and a README.md file. Unlike an empty repository, you can use this repository as soon as it's created.
8. In **Default branch**, leave the name as *main* unless you have a reason to choose a different name. The examples in this guide all use the name *main* for the default branch.

9. (Optional) Add a `.gitignore` file for the type of code you plan to push.
10. Choose **Create**.

> **Note**
>
> CodeCatalyst adds a `README.md` file to your repository when you create it. CodeCatalyst also creates an initial commit for the repository in a default branch named **main**. You can edit or delete the README.md file, but you can't delete the default branch.

To create an empty source repository

1. In the CodeCatalyst console, navigate to the project where you want to create an empty repository.
2. On the summary page for your project, in **Source repositories**, choose **Add repository**, and then choose **Create repository**. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**. Choose **Add repository**, and then choose **Create repository**.
3. In **Repository name**, provide a name for the repository. In this guide, we use *`codecatalyst-source-repository`*, but you can choose a different name. Repository names must be unique within a project. For more information about requirements for repository names, see Quotas for source repositories in CodeCatalyst.
4. (Optional) In **Description**, add a description for the repository that will help other users in the project understand what the repository is used for.
5. Choose **Create empty repository**, and then choose **Create**.

Cloning an existing Git repository into a source repository

You can clone an existing Git repository to an empty source repository in Amazon CodeCatalyst. This is a quick way to get started in CodeCatalyst with code that was previously hosted in another Git repository provider. You can clone the contents of the repository by creating a mirror clone and then pushing the mirror to CodeCatalyst. Alternatively, if you have a local repo of the repository whose contents you want to add to CodeCatalyst, you can add the CodeCatalyst source repository as another remote to the local repo, and then push to the empty source repository. Both approaches are equally valid. Using a mirror clone not only maps branches, it maps all refs. It's a simple and clean way to create a working copy of the repository in CodeCatalyst. Adding a remote

to a local repo that points to an empty CodeCatalyst source repository will add the repository contents to CodeCatalyst, but it will also allow you to make pushes from the local repo to both the CodeCatalyst source repository and the original Git remote repository. This can be useful if you want to maintain the code in different remote repositories, but can lead to conflicts if other developers are committing code to only one of the remotes.

The following procedures use basic Git commands to accomplish this task. There are many ways to accomplish tasks in Git, including cloning. For more information, see the Git documentation.

> **Important**
>
> You must create an empty repository in CodeCatalyst before you can clone content to it. You must also have a personal access token. For more information, see To create an empty source repository and Create a personal access token.

To use `git clone --mirror` to clone an existing Git repository into CodeCatalyst

1. In the CodeCatalyst console, navigate to the project where you created an empty repository.
2. On the summary page for your project, choose the empty repository from the list, and then choose **View repository**. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**. Choose the name of the empty repository from the list of source repositories for the project.
3. Copy the HTTPS clone URL of the empty repository. You'll need this for pushing the mirror clone. For example, if you named the source repository MyExampleRepo in the MyExampleProject project in the ExampleCorp space, and your user name is LiJuan, your clone URL might look like the following:

   ```
   https://LiJuan@git.us-west-2.codecatalyst.aws/
   v1/ExampleCorp/MyExampleProject/MyExampleRepo
   ```

4. At a command line or terminal window, use the `git clone --mirror` command to create a mirror clone of the Git repository you want to clone to CodeCatalyst. For example, if you want to create a mirror clone of the codecatalyst-blueprints repository in GitHub, you would enter the following command:

   ```
   git clone --mirror https://github.com/aws/codecatalyst-blueprints.git
   ```

5. Change directories to the directory where you made the clone.

```
cd codecatalyst-blueprints.git
```

6. Run the **git push** command, specifying the URL and name of the destination CodeCatalyst source repository and the **--all** option. (This is the URL you copied in Step 3.) For example:

```
git push https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo --all
```

To add a remote and push a local repo into CodeCatalyst

1. In the CodeCatalyst console, navigate to the project where you created an empty repository.
2. On the summary page for your project, choose the empty repository from the list, and then choose **View repository**. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**. Choose the name of the empty repository from the list of source repositories for the project.
3. Copy the HTTPS clone URL of the empty repository. You'll need this for pushing the mirror clone. For example, if you named the source repository MyExampleRepo in the MyExampleProject project in the ExampleCorp space, and your user name is LiJuan, your clone URL might look like the following:

```
https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

4. At a command line or terminal window, change directories to the local repo you want to push to CodeCatalyst.
5. Run the git remote -v command to see the existing remotes for the local repository. For example, if you are cloning a local repo of a AWS CodeCommit repository named **MyDemoRepo** in the US East (Ohio) Region, your command output might look like this:

```
origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

   Copy the remote URL if you want to continue using the repository.

6. Use the `git remote remove` command to remove the CodeCommit repository URLs for fetch and push for origin:

```
git remote remove origin
```

7. Use the **git remote add** command to add the CodeCatalyst source repository URL as the fetch and push remote for your local repo. For example:

```
git remote add origin https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

This replaces the CodeCommit repository push URL with CodeCatalyst source repository URL, but does not change the fetch URL. So if you run the git remote -v command again, you'll see that you're now pulling (fetch) code from the CodeCommit remote repository, but you're configured to push changes from your local repo to the CodeCatalyst source repository:

```
origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin  https://LiJuan@git.us-west-2.codecatalyst.aws/v1/ExampleCorp/
MyExampleProject/MyExampleRepo (push)
```

You can optionally add back the CodeCommit remote URL if you want to push to both repositories with the `git remote set-url` command:

```
git remote set-url --add --push origin https://git-codecommit.us-
east-2.amazonaws.com/v1/repos/MyDemoRepo
```

8. Run the `git push` command to push the local repo to all configured push remotes. Alternatively, run the **git push -u -origin** command, specifying the **--all** option to push the local repo to both repositories. For example:

```
git push -u -origin --all
```

Tip

Depending on your version of Git, --all might not work to push all branches of the local repo to the empty repository. You might have to check out and push each branch separately.

Linking a source repository

When linking a source repository to a project, you can include repositories that have a CodeCatalyst extension for the service that hosts the repository, if that extension is installed for your space. Only users with the Space administrator role can install extensions. Once the extension is installed, you can link to repositories configured for access by that extension. For more information, see Installing an extension in a space or follow Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

> **Important**
>
> After you install a repository extension, any repositories you link to CodeCatalyst will have their code indexed and stored in CodeCatalyst. This will make the code searchable in CodeCatalyst. To better understand the data protection for your code when using linked repositories in CodeCatalyst, see Data protection in the *Amazon CodeCatalyst User Guide*.

You can link a repository to only one project in a space. You cannot link an archived repository. While you can link an empty repository, you can't use it in CodeCatalyst until you have initialized it with an initial commit that creates a default branch. Additionally:

- A GitHub repository, Bitbucket repository, or GitLab project repository can only be linked to one CodeCatalyst project in a space.
- You can't use empty or archived GitHub repositories, Bitbucket repositories, or GitLab project repositories with CodeCatalyst projects.
- You can't link a GitHub repository, Bitbucket repository, or GitLab project repository that has the same name as a repository in a CodeCatalyst project.
- The **GitHub repositories** extension isn't compatible with GitHub Enterprise Server repositories.
- The **Bitbucket repositories** extension isn't compatible with Bitbucket Data Center repositories.
- The **GitLab repositories** extension isn't compatible with GitLab self-managed project repositories.
- You can't use the **Write description for me** or **Summarize comments** features with linked repositories. These features are only available in pull requests in CodeCatalyst.

While you can link a GitHub repository, Bitbucket repository, or GitLab project repository as a **Contributor**, you can only unlink a third-party repository as the **Space administrator** or the

Project administrator. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

> **Important**
>
> CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
> As a best practice, always make sure you have the latest version of the extension before you link a repository.

To link a source repository

1. Navigate to the project where you want to link a repository.

   > **Note**
   >
   > Before you can link a repository, a user with the Space administrator role must first install the extension for the provider that hosts the repository. For more information, see Installing an extension in a space.

2. In the navigation pane, choose **Code**, and then choose **Source repositories**.
3. Choose **Add repository**, and then choose **Link repository**.
4. From the **Repository provider** dropdown menu, choose one of the following third-party repository providers: **GitHub** or **Bitbucket**.
5. Do one of the following depending on the third-party repository provider you chose to link:

   - **GitHub repositories**: Link a GitHub repository.

     1. From the **GitHub account** dropdown menu, choose the GitHub account that contains the repository that you want to link.
     2. From the **GitHub repository** dropdown menu, choose the GitHub account you want to link your CodeCatalyst project.
     3. (Optional) If you don't see a GitHub repository in the list of repositories, it might not have been configured for repository access in the Amazon CodeCatalyst application in

GitHub. You can configure which GitHub repositories can be used in CodeCatalyst in the connected account.

a. Navigate to your GitHub account, choose **Settings**, and then choose **Applications**.

b. In the **Installed GitHub Apps** tab, choose **Configure** for the Amazon CodeCatalyst application.

c. Do one of the following to configure access of GitHub repositories you want to link in CodeCatalyst:

- To provide access to all current and future repositories, choose **All repositories**.
- To provide access to specific repositories, choose **Only select repositories**, choose the **Select repositories** dropdown, and then choose a repository you want to allow to link in CodeCatalyst.

- **Bitbucket repositories**: Link a Bitbucket repository.

1. From the **Bitbucket workspace** dropdown menu, choose the Bitbucket workspace that contains the repository that you want to link.
2. From the **Bitbucket repository** dropdown menu, choose the Bitbucket repository you want to link your CodeCatalyst project.

> **Tip**
>
> If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the Amazon CodeCatalyst.

6. Choose **Link**.

If you no longer want to use a GitHub repository, Bitbucket repository, or GitLab project repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a repository is unlinked, events in that repository will not start workflow runs, and you will not be able to use that repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

Viewing a source repository

You can view the source repositories associated with a project in Amazon CodeCatalyst. For source repositories in CodeCatalyst, the overview page for a repository provides a quick overview of information and activity in that repository, including:

- The description of the repository, if any
- The number of branches in the repository
- The number of open pull requests for the repository
- The number of related workflows for the repository
- The files and folders in the default branch, or the branch that you choose
- The title, author, and date of the last commit to the displayed branch
- The contents of the README.md file rendered in Markdown, if any README.md file is included

This page also provides links to the commits, branches, and pull requests for the repository, as well as a quick way to open, view, and edit individual files.

> **Note**
>
> You cannot view this information about linked repositories in the CodeCatalyst console. To view information about linked repositories, choose the link in the list of repositories to open that repository in the service that hosts it.

To navigate to the source repositories for a project

1. Navigate to your project, and do one of the following:
   - On the summary page for your project, choose the repository you want from the list, and then choose **View repository**.
   - In the navigation pane, choose **Code**, and then choose **Source repositories**. In **Source repositories**, choose the name of the repository from the list. You can filter the list of repositories by typing part of the repository name in the filter bar.
2. On the home page for the repository, view the contents of the repository and information about the associated resources such as the number of pull requests and workflows. By default, the contents for the default branch are shown. You can change the view by choosing a different branch from the drop-down list.

> **Tip**
>
> You can also quickly navigate to your project's repositories by choosing **See project code** from the project summary page.

Editing the settings for a source repository

You can manage the settings for your repository, including editing the description of a repository, choosing the default branch, creating and managing branch rules, and creating and managing approval rules for pull requests in CodeCatalyst. This can help project members understand what the repository is used for, and help you enforce best practices and processes used by the team.

> **Note**
>
> You can't edit the name of a source repository.
> You can't edit the name, description, or other information for a linked repository in CodeCatalyst. To modify information about a linked repository, you must edit it in the provider that hosts the linked repository. For more information, see the documentation for the service that hosts the linked repository.

To edit the settings of a repository

1. In the CodeCatalyst console, navigate to the project that contains the source repository whose settings you want to edit.
2. On the summary page for your project, choose the repository you want from the list, and then choose **View repository**. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**. Choose the name of the repository from the list of source repositories for the project.
3. On the overview page for the repository, choose **More**, and then choose **Manage settings**.
4. Do one or more of the following:
   - Edit the description of the repository and then choose **Save**.
   - To change the default branch for the repository, in **Default branch**, choose **Edit**. For more information, see Managing the default branch for a repository.

- To add, remove, or change a rule for what project roles have permission to perform certain actions in a branch, in **Branch rules**, choose **Edit**. For more information, see Manage allowed actions for a branch with branch rules.
- To add, remove, or change an approval rule for merging pull reuqests to a branch, in **Approval rules**, choose **Edit**. For more information, see Managing requirements for merging a pull request with approval rules.

Cloning a source repository

To work effectively with multiple files, branches, and commits in source repositories, clone the source repository to your local computer and use a Git client or an integrated development environment (IDE) to make changes. Commit and push your changes to the source repository in order to work with CodeCatalyst features such as issues and pull requests. You can also choose to create a Dev Environment to work on code. Creating a Dev Environment automatically clones the repository and branch you specify into the Dev Environment.

> **Note**
>
> You cannot clone linked repositories in the CodeCatalyst console or create Dev Environments for them. To clone a linked repository locally, choose the link in the list of repositories to open that repository in the service that hosts it, and then clone it. For more information, see the documentation for the service that hosts the linked repository.

To create a Dev Environment from a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **Code**, and then choose **Source repositories**.
3. Choose the source repository where you want to work on code.
4. Choose **Create Dev Environment**.
5. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.
6. Do one of the following:

   - Choose **Work in existing branch**, and then choose a branch from the **Existing branch** drop-down menu.

- Choose **Work in new branch**, enter a branch name into the **Branch name** field, and choose a branch off of which to create the new branch from the **Create branch from** drop-down menu.

7. Optionally add a name for the Dev Environment or edit its configuration.
8. Choose **Create**.

To clone a source repository

1. Navigate to your project.
2. On the summary page for your project, choose the repository you want from the list, and then choose **View repository**. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**. Choose the name of the repository from the list of source repositories for the project. You can filter the list of repositories by typing part of the repository name in the filter bar.
3.
4. Choose **Clone repository**. Copy the clone URL for the repository.

> **Note**
>
> If you do not have a personal access token (PAT), choose **Create token**. Copy the token and save it in a secure location. You will use this PAT when prompted for a password by your Git client or integrated development environment (IDE).

5. Do one of the following:

- To clone a repository to your local computer, open a terminal or command line and run the **git clone** command with the clone URL after the command. For example:

```
git clone https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

When prompted for a password, paste the PAT you saved earlier.

> **Note**
>
> If your operating system provides credential management or you have installed a credential management system, you only need to provide the PAT once. If not, you

might have to provide the PAT for every Git operation. As a best practice, make sure that your credential management system securely stores your PAT. Do not include the PAT as part of the clone URL string.

- To clone a repository using an IDE, follow the documentation for your IDE. Choose the option to clone a Git repository and provide the URL. When prompted for a password, provide the PAT.

Deleting a source repository

If a source repository for an Amazon CodeCatalyst project is no longer needed, you can delete it. Deleting a source repository also deletes any project information stored in the repository. If any workflows depend on the source repository, those workflows will be deleted from the list of project workflows after the repository is deleted. Issues that reference the source repository will not be deleted or altered, but any links to the source repository added to issues will fail once the repository is deleted.

Important

Deleting a source repository cannot be undone. After you delete a source repository, you are no longer able to clone it, pull data from it, or push data to it. Deleting a source repository does not delete any local copies of that repository (local repos). To delete a local repo, use your local computer's directory and file management tools.

Note

You cannot delete a linked repository in the CodeCatalyst console. To delete a linked repository, choose the link in the list of repositories to open that repository in the service that hosts it, and then delete it. For more information, see the documentation for the service that hosts the linked repository.
To remove a linked repository from a project, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

To delete a source repository

1. Navigate to the project that contains the source repository you want to delete.
2. On the summary page for your project, choose the repository you want from the list, and then choose **View repository**. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**. Choose the name of the repository from the list of source repositories for the project.
3. On the home page for the repository, choose **More**, and then choose **Delete repository**.
4. Review the branch, pull request, and related workflow information to help ensure that you are not deleting a repository that is still in use or has unfinished work. If you want to continue, type **delete**, and then choose **Delete**.

Organizing your source code work with branches in Amazon CodeCatalyst

In Git, branches are pointers or references to a commit. In development, they're a convenient way to organize your work. You can use branches to separate work on a new or different version of files without affecting work in other branches. You can use branches to develop new features, store a specific version of your project, and more. You can configure rules for branches in source repositories to limit certain actions on a branch to specific roles in that project.

Source repositories in Amazon CodeCatalyst have contents and a default branch regardless of how you create them. Linked repositories might not have a default branch or content, but are not usable by CodeCatalyst until you initialize them and create a default branch. When you create a project using a blueprint, CodeCatalyst creates a source repository for that project that includes a README.md file, sample code, workflow definitions, and other resources. When you create a source repository without using a blueprint, a README.md file is added for you as a first commit, and a *default branch* is created for you. This default branch is named *main*. This default branch is the one used as the base or default branch in local repositories (repos) when users clone the repository.

> **Note**
>
> You can't delete the default branch. The first branch created for a source repository is the default branch for that repository. Additionally, search only displays results from the default branch. You can't search for code in other branches.

Creating a repository in CodeCatalyst also creates a first commit, which creates a *default branch* with a README.md file included in it. The name of that default branch is *main*. This is the default branch name used in the examples in this guide.

Topics

Creating a branch

You can use the CodeCatalyst console to create branches in a CodeCatalyst repository. The branches you create will be visible to other users the next time they pull changes from the repository.

> **Tip**
>
> You can also create branches as part of creating a Dev Environment to work on your code. For more information, see Creating a Dev Environment.

You can also use Git to create branches. For more information, see Common Git commands for branches.

To create a branch (console)

1. In the CodeCatalyst console, navigate to the project where your source repository resides.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.
3. Choose the repository where you want to create a branch.
4. On the overview page of the repository, choose **More**, and then choose **Create branch**.
5. Enter a name for the branch.
6. Choose a branch to create the branch from, and then choose **Create**.

Managing the default branch for a repository

You can specify which branch to use as the *default branch* in a source repository in Amazon CodeCatalyst. All source repositories in CodeCatalyst have contents and a default branch regardless of how you create them. If you use a blueprint to create a project, the default branch in the source repository created for that project is named *main*. The contents of the default branch are displayed automatically on the overview page for that repository.

Important

CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
As a best practice, always make sure you have the latest version of the extension before you link a repository.

The default branch is treated a little differently than all other branches in a source repository. It has a special label next to its name, **Default**. The default branch is the one used as the base or default branch in local repositories (repos) when users clone the repository to local computers with a Git client. It is also the default used when creating workflows for storing workflow YAML files, and for storing information for issues. When using search in CodeCatalyst, only the default branch of a repository is searched. Because the default branch is fundamental to so many aspects of projects, you cannot delete a branch if it is specified as the default branch. However, you can choose to use a different branch as the default branch. If you do, any branch rules that were applied to the former default branch will be applied automatically to the branch you specify as the default branch.

Note

You must have the Project administrator role to change the default branch for source repositories in CodeCatalyst projects. This does not apply to linked repositories.

To view and change the default branch for a repository

1. Navigate to the project where your repository resides.

2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to view the settings, including the default branch.
3. On the overview page of the repository, choose **More**, and then choose **Manage settings**.
4. In **Default branch**, the name of the branch specified as the default branch is displayed along with a label called **Default** next to the name. This same label appears next to the branch name in the list of branches in **Branches**.
5. To change the default branch, choose **Edit**.

   > **Note**
   >
   > You must have the Project administrator role in the project to change the default branch.

6. Choose the name of the branch you want to make the default branch from the drop-down list and then choose **Save**.

Manage allowed actions for a branch with branch rules

When you create a branch, certain actions are allowed for that branch based on the permissions for that role. You can change what actions are allowed for a specific branch by configuring branch rules. Branch rules are based on the role a user has in your project. You can choose to limit some predefined actions, such as pushing commits to a branch, to users with a particular role in a project. This can help you protect specific branches in a project by limiting which roles are allowed to perform certain actions. For example, if you configure a branch rule to only allow users with the **Project administrator** role to merge or push to that branch, users with other roles in the project will not be able to make changes to the code in the that branch.

You should carefully consider all the implications of creating a rule for a branch. For example, if you choose to limit pushes to a branch to users with the **Project administrator** role, users with the **Contributor** role will not be able to create or edit workflows in that branch, because the workflow YAML is stored in that branch, and those users cannot commit and push changes to the YAML. As a best practice, test any branch rules after you create them in order to make sure that they do not have any impacts that you did not intend. You can also use branch rules in conjunction with approval rules for pull requests. For more information, see Managing requirements for merging a pull request with approval rules.

> **Note**
>
> You must have the Project administrator role to manage branch rules for source repositories in CodeCatalyst projects. You cannot create branch rules for linked repositories. You can only create branch rules that are more restrictive than the default permissions for the role. You cannot create branch rules that are more permissive than what a user's role in the project allows. For example, you cannot create a branch rule that allows users with the Reviewer role to push to the branch.

Branch rules that are applied to the default branch of your source repository will behave a little differently than branch rules applied to other branches. Any rule applied to the default branch will be automatically applied to any branch you specify as the default branch. The branch that was formerly set as the default branch will still keep the rules applied to it, except that it will no longer have protection against deletion. That protection is only applied to the current default branch.

Branch rules have two states, **Standard** and **Custom**. **Standard** indicates that the actions allowed on a branch are those that match the permissions for the role the user has in CodeCatalyst for branch actions. To learn more about what roles have which permissions, see Granting access with user roles. **Custom** indicates that one or more branch actions have actions that have a specific list of roles allowed to perform that action that differ from the default permissions granted by a user's roe in the project.

> **Note**
>
> If you create a branch rule to restrict one or more actions for a branch, the **Delete the branch** action is automatically set to only allow users with the Project administrator role to delete that branch.

The following table lists the actions and the default settings for roles allowed to perform these actions on a branch.

Branch actions and roles

Branch action	Roles allowed to perform this action when no branch rules are applied
Merge to the branch (this includes merging a pull request to the branch)	Project administrator, Contributor
Push to the branch	Project administrator, Contributor
Delete the branch	Project administrator, Contributor
Delete the branch (default branch)	Not allowed

You cannot delete branch rules, but you can update them to allow actions from all the roles that would be allowed to perform this action on a branch, which effectively removes the rule.

> **Note**
>
> You must have the Project administrator role to configure branch rules for source repositories in CodeCatalyst projects. This does not apply to linked repositories. Linked repositories do not support the branch rules in CodeCatalyst.

To view and edit branch rules for a repository

1. Navigate to the project where your repository resides.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to view branch rules.
3. On the overview page of the repository, choose **Branches**.
4. In the **Branch rules** column, view the status of rules for each branch of the repository. **Standard** indicates that the rules for branch action are the default ones for any branch created in a source repository and match the permissions granted to those roles in a project. **Custom** indicates that one or more branch actions have rules that restrict one or more actions allowed for that branch to a different set of roles.

To view the specifics of the branch rules for a branch, choose the word **Standard** or **Custom** next to the branch you want to review.

5. To create or change a branch rule, choose **Manage settings**. On the settings page for the source repository, in **Branch rules**, choose **Edit**.
6. In **Branch**, choose the name of the branch for which you want to configure a rule from the drop-down list. For each of the allowed action types, choose the roles you want to allow to perform that action from the drop-down list, and then choose **Save**.

Git commands for branches

You can use Git to create, manage, and delete branches in the clone of the source repository you have on your computer (your local repo) or in your Dev Environments, and then commit and push your changes to your CodeCatalyst source repository (the remote repository). For example:

Common Git commands for branches

Lists all branches in the local repo with an asterisk (*) displayed next to your current branch.	`git branch`
Pulls information about all existing branches in the remote repository to the local repo.	`git fetch`
Lists all branches in the local repo and remote tracking branches in the local repo.	`git branch -a`
Lists only remote tracking branches in the local repo.	`git branch -r`
Creates a branch in the local repo using the specified branch name. This branch will not appear in the remote repository until you commit and push the change.	`git branch` *`branch-name`*
Creates a branch in the local repo using the specified branch name, and then switches to it.	`git checkout -b` *`branch-name`*

Switches to another branch in the local repo using the specified branch name.	`git checkout other-branch-name`
Pushes a branch from the local repo to the remote repository using the local repo's specified nickname for the remote repository and the specified branch name. Also sets up upstream tracking information for the branch in the local repo.	`git push -u remote-name branch-name`
Merges changes from another branch in the local repo to the current branch in the local repo.	`git merge from-other-branch-name`
Deletes a branch in the local repo unless it contains work that has not been merged.	`git branch -d branch-name`
Deletes a branch in the remote repository using the specified nickname the local repo has for the remote repository and the specified branch name. (Note the use of the colon (`:`).) Alternatively, specify `--delete` as part of the command.	`git push remote-name :branch-name` `git push remote-name --delete branch-name`

For more information, see your Git documentation.

Viewing branches and details

You can view information about remote branches in Amazon CodeCatalyst, including specifics of the files, folders, and most recent commit for a specific branch, in the Amazon CodeCatalyst console. You can also use Git commands and your local operating system to view this information for remote and local branches.

To view branches (console)

1. In the CodeCatalyst console, navigate to the project that contains the source repository where you want to view branches. Choose **Code**, choose **Source repositories**, and then choose the source repository.

2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to view a branch.
3. The default branch of the repository is displayed. You can see a list of files and folders in the branch, information about the most recent commit, and the contents of the README.md file, if it exists in the branch. To view the information for a different branch, choose it from the drop-down list of branches for the repository.
4. To view all the branches for a repository, choose **View all**. The Branches page displays information about the name, most recent commit, and rules for each branch.

For information about how to use Git and your operating system to view branches and details, see Common Git commands for branches, your Git documentation, and your operating system documentation.

Deleting a branch

If you no longer need a branch, you can delete it. For example, if you've merged a branch with a feature change into the default branch and that feature has been released, you might want to delete the original feature branch, as the changes are already part of the default branch. Keeping the number of branches low can help users find the branch that contains the changes they want to work on. When you delete a branch, copies of that branch remain in the clones of the repository on local computers until users pull and synchronize those changes.

To delete a branch (console)

1. Navigate to the project where your repository resides.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to delete a branch.
3. On the overview page of the repository, choose the drop-down selector next to the branch name, and then choose **View all**.
4. Choose the branch that you want to delete and then choose **Delete branch**.

> **Note**
>
> You cannot delete the default branch for a repository.

5. A confirmation dialog box appears. It shows the repository, number of open pull requests, and number of workflows associated with the branch.
6. To confirm deletion of the branch, type **delete** into the text box, and then choose **Delete**.

You can also use Git to delete branches. For more information, see Common Git commands for branches.

Managing source code files in Amazon CodeCatalyst

In Amazon CodeCatalyst, a file is a version-controlled, self-contained piece of information available to you and other users of the source repository and branch where the file is stored. You can organize your repository files with a directory structure. CodeCatalyst automatically tracks every committed change to a file. You can store different versions of a file in different repository branches.

To add or edit multiple files in a source repository, you can use a Git client, a Dev Environment, or an integrated development environment (IDE). To add or edit a single file, you can use the CodeCatalyst console.

Topics

Creating or adding a file

To create and add files to a source repository, you can use the Amazon CodeCatalyst console, a Dev Environment, a connected integrated development environment (IDE), or a Git client. The CodeCatalyst console includes a code editor for creating files. This editor is a convenient way to

create or edit a simple file, such as a README.md file, in a branch in a repository. When working on more than one file, consider creating a Dev Environment.

To create a Dev Environment from a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **Code**, and then choose **Source repositories**.
3. Choose the source repository where you want to work on code.
4. Choose **Create Dev Environment**.
5. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.
6. Do one of the following:
   - Choose **Work in existing branch**, and then choose a branch from the **Existing branch** drop-down menu.
   - Choose **Work in new branch**, enter a branch name into the **Branch name** field, and choose a branch off of which to create the new branch from the **Create branch from** drop-down menu.
7. Optionally add a name for the Dev Environment or edit its configuration.
8. Choose **Create**.

To create a file in the CodeCatalyst console

1. Navigate to the project where you want to create a file. For more information about how to navigate to a repository, see Viewing a source repository.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to create the file.
3. (Optional) Choose the branch where you want to create the file, if you want to create the file in a different branch than the default branch.
4. Choose **Create file**.
5. Enter the name of the file in **File name**. Add the contents of the file in the editor.

> **Tip**
>
> If you want to create the file in a sub-folder or subdirectory of the root of the branch, include that structure as part of the file name.

When you are satisfied with your changes, choose **Commit**.

6. In **File name**, review the name of the file and make any changes you might want to it. Optionally choose the branch where you want to create the file from the list of available branches in **Branch**. In **Commit message**, optionally enter a brief but informative description of why you made this change. This will be displayed as the basic commit information for the commit that adds the file to the source repository.
7. Choose **Commit** to commit and push the file to the source repository.

You can also add files to a source repository by cloning it to your local computer and using a Git client or connected integrated development environment (IDE) to push your files and changes.

> **Note**
>
> If you want to add a Git submodule, you must use a Git client or a Dev Environment and run the **git submodule add** command. You cannot add or view Git submodules in the CodeCatalyst console or view the differences in Git submodules in pull requests. For more information about Git submodules, see the Git documentation.

To add a file using a Git client or connected integrated development environment (IDE)

1. Clone your source repository to your local computer. For more information, see Cloning a source repository.
2. Create files in your local repo or copy files into your local repo.
3. Create and push a commit by doing one of the following:

   - If you are using a Git client, at the terminal or command line, run the **git add** command, specifying the names of the files you want to add. Alternatively, to add all added or changed files, run the **git add** command followed by either a single or double period to indicate whether you want to include all the changes at the current directory level (single period) or

all changes in the current directory and all subdirectories (double period). To commit the changes, run the **git commit -m** command and provide a commit message. To push your changes to the source repository in CodeCatalyst, run **git push**. For more information about Git commands, see your Git documentation and Git commands for branches.

- If you are using a Dev Environment or an IDE, create files and add files in the IDE, and then commit and push your changes. For more information, see see Write and modify code with Dev Environments in CodeCatalyst or consult your IDE documentation.

Viewing a file

You can view the files in your source repository in the Amazon CodeCatalyst console. You can view the files in the default branch and in any other branches. File contents might vary depending on the branch you choose to view.

To view files in the CodeCatalyst console

1. Navigate to the project where you want to view files. For more information, see Viewing a source repository.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to view files.
3. A list of files and folders are displayed for the default branch. Files are indicated by a paper icon, while folders are indicated by a folder icon.
4. Do any of the following:
   - To view files and folders in a different branch, choose it from the list of branches.
   - To expand a folder, choose it from the list.
5. To view the contents of a specific file, choose it from the list. The contents of the file will be displayed in the branch. To view the contents of the file in a different branch, choose the branch you want from the branch selector.

> **Tip**
>
> When viewing the contents of a file, you can choose additional files to view from **View files**. To edit a file, choose **Edit**.

You can view multiple files in the console. You can also view files that you have cloned to your local computer using a Git client or an integrated development environment (IDE). For more information, see the documentation for your Git client or IDE.

> **Note**
>
> You cannot view Git submodules in the CodeCatalyst console. For more information about Git submodules, see the Git documentation.

Viewing the history of changes to a file

You can view the the history of changes to a file in your source repository in the Amazon CodeCatalyst console. This can help you understand the changes made to the file by various commits to the branch where you choose to view the history of the file. For example, if you view the history of changes to the **`readme.md`** file in the **`main`** branch of the source repository, you will see a list of commits that included changes to that file in that branch.

> **Note**
>
> You can't view the history of a file in a linked repository in the CodeCatalyst console.

To view the history of a file in the CodeCatalyst console

1. Navigate to the project where you want to view the history of a file. For more information, see Viewing a source repository.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.
3. Choose the repository where you want to view the history of a file. Choose the branch where you want to view the history the file, and then choose the file from the list. Choose **View history**.
4. Review the list of commits that included changes to this file in the specified branch. To view the details of the changes included in a particular commit, choose the commit message for that commit in the list. The differences between that commit and its parent commit are displayed.

5. To review the history of changes to the file in another branch, use the branch selector to change views to that branch, choose the file from the file list, and then choose **View history**.

> **Note**
>
> You cannot view the history of changes to Git submodules in the CodeCatalyst console. For more information about Git submodules, see the Git documentation.

Editing a file

You can edit individual files in the Amazon CodeCatalyst console. To edit multiple files at once, either create a Dev Environment or clone the repository and make your changes using a Git client or integrated development environment (IDE). For more information, see Write and modify code with Dev Environments in CodeCatalyst or Cloning a source repository.

To edit a file in the CodeCatalyst console

1. Navigate to the project where you want to edit a file. For more information about how to navigate to a repository, see Viewing a source repository.
2. Choose the repository where you want to edit the file. Choose **View branches** and then choose the branch you want to work in. Choose the file from the list of files and folders in that branch.

   The contents of the file are displayed.
3. Choose **Edit**.
4. In the editor, edit the contents of the file and then choose **Commit**. Optionally, in **Commit changes**, add more information about the change in **Commit message**. When you are satisfied with your changes, choose **Commit**.

Renaming or deleting a file

You can rename or delete files in a Dev Environment, locally on your computer, or in an integrated development environment (IDE). Once you have renamed or deleted the files, commit and push those changes to the source repository. You can't rename or delete files in the Amazon CodeCatalyst console.

Reviewing code with pull requests in Amazon CodeCatalyst

A pull request is the primary way you and other project members can review, comment on, and merge code changes from one branch to another. You can use pull requests to review code changes collaboratively for minor changes or fixes, major feature additions, or new versions of your released software. If you use issues to track work on your project, you can link specific issues to pull requests to help you track what issues are being addressed by the code changes in the pull request. When you create, update, comment on, merge, or close a pull request, an email is automatically sent to the author of the pull request as well as any required or optional reviewers for the pull request.

Tip

You can configure what pull request events you that will receive emails about as part of your profile. For more information, see Managing notifications in Amazon CodeCatalyst.

Pull requests require two branches in a source repository: a source branch that contains the code that you want reviewed, and a destination branch, where you want to merge the reviewed code. The source branch contains the AFTER commit, which is the commit that contains the changes you want to merge into the destination branch. The destination branch contains the BEFORE commit, which represents the state of the code before the pull request branch is merged into the destination branch.

Note

While you are creating a pull request, the difference displayed is the difference between the tip of the source branch and the tip of the destination branch. Once you create the pull request, the displayed difference will be between the revision of the pull request you choose and the commit that was the tip of the destination branch when you created the pull request. For more information about differences and merge bases in Git, see git-merge-base in the Git documentation.

While a pull request is created for a specific source repository and branches, you can create, view, review, and close them as part of working with your project. You do not have to view the source repository in order to view and work with pull requests. A pull request state is set to **Open** when you create it. The pull request remains open until you either merge it in the CodeCatalyst console, which changes the state to **Merged**, or close it, which changes the state to **Closed**.

When your code has been reviewed, you can change the pull request state in one of several ways:

- Merge the pull request in the CodeCatalyst console. The code in the source branch of the pull request will be merged into the destination branch. The pull request status will change to **Merged**. It can't be changed back to **Open**.
- Merge the branches locally and push your changes, and then close the pull request in the CodeCatalyst console.
- Use the CodeCatalyst console to close the pull request without merging. This will change the status to **Closed**, and it will not merge the code from the source branch into the destination branch.

Before you create a pull request:

- Commit and push the code changes you want reviewed to a branch (the source branch).
- Set up notifications for your project, so other users can be notified about any workflows that run when you create a pull request. (This step is optional but recommended.)

Topics

Creating a pull request

Creating pull requests helps other users see and review your code changes before you merge them into another branch. First, you create a branch for your code changes. This is referred to as the source branch for a pull request. After you commit and push changes to the repository, you can create a pull request that compares the contents of the source branch to the contents of the destination branch.

You can create a pull request in the Amazon CodeCatalyst console from a specific branch, from the pull requests page, or from the project overview. Creating a pull request from a specific branch automatically provides the repository name and source branch on the pull request creation page. When you create a pull request, you will automatically receive emails about any updates to the pull request, as well as when the pull request is merged or closed.

> **Note**
>
> While you are creating a pull request, the difference displayed is the difference between the tip of the source branch and the tip of the destination branch. Once the pull request has been created, the displayed difference will be between the revision of the pull request you choose and the commit that was the tip of the destination branch when you created the pull request. For more information about differences and merge bases in Git, see git-merge-base in the Git documentation.

You can use the **Write description for me** feature when creating pull requests to have Amazon Q automatically create a description of the changes contained in a pull request. When you choose this option, Amazon Q analyzes the differences between the source branch that contains the code changes and the destination branch where you want to merge these changes. It then creates a summary of what those changes are, as well as its best interpretation of the intent and effect of those changes. This feature is only available in the US West (Oregon) Region for CodeCatalyst pull requests. The **Write description for me** feature is not available for pull requests in linked repositories.

> **Note**
>
> **Powered by Amazon Bedrock**: AWS implements automated abuse detection. Because the **Write description for me** and **Create content summary** features are built on Amazon Bedrock, users can take full advantage of the controls implemented in Amazon Bedrock to enforce safety, security, and the responsible use of artificial intelligence (AI).

To create a pull request

1. Navigate to your project.
2. Do one of the following:

- In the navigation pane, choose **Code**, choose **Pull requests**, and then choose **Create pull request**.
- On the repository home page, choose **More**, and then choose **Create pull request**.
- On the project page, choose **Create pull request**.

3. In **Source repository**, make sure that the specified source repository is the one that contains the committed code. This option only appears if you did not create the pull request from the repository's main page.
4. In **Destination branch**, choose the branch to merge the code into after it is reviewed.
5. In **Source branch**, choose the branch that contains the committed code.
6. In **Pull request title**, enter a title that helps other users understand what needs to be reviewed and why.
7. (Optional) In **Pull request description**, provide information such as a link to issues or a description of your changes.

> **Tip**
>
> You can choose **Write description for me** to have CodeCatalyst automatically generate a description of the changes contained in the pull request. You can make changes to the automatically generated description after you add it to the pull request.
> This functionality requires that generative AI features are enabled for the space and is not available for pull requests in linked repositories. For more information, see Managing generative AI features.

8. (Optional) In **Issues**, choose **Link issues**, and then either choose an issue from the list or enter its ID. To unlink an issue, choose the unlink icon.
9. (Optional) In **Required reviewers**, choose **Add required reviewers**. Choose from the list of project members to add them. Required reviewers must approve the changes before the pull request can be merged into the destination branch.

> **Note**
>
> You cannot add a reviewer as both a required reviewer and an optional reviewer. You cannot add yourself as a reviewer.

10. (Optional) In **Optional reviewers**, choose **Add optional reviewers**. Choose from the list of project members to add them. Optional reviewers do not have to approve the changes as a requirement before the pull request can be merged into the destination branch.
11. Review the differences between the branches. The difference displayed in a pull request is the changes between the revision in the source branch and the merge base, which is the head commit of the destination branch at the time the pull request was created. If no changes display, the branches might be identical, or you might have chosen the same branch for both the source and the destination.
12. When you are satisfied that the pull request contains the code and changes you want reviewed, choose **Create**.

> **Note**
>
> After you create the pull request, you can add comments. Comments can be added to the pull request or to individual lines in files as well as to the overall pull request. You can add links to resources, such as files, by using the @ sign followed by the name of the file.

To create a pull request from a branch

1. Navigate to the project where you want to create a pull request.
2. In the navigation pane, choose **Source repositories**, and then choose the repository that contains the branch where you have code changes to review.
3. Choose the drop-down arrow next to the default branch name, and then choose the branch you want from the list. To view all the branches for a repository, choose **View all**.
4. Choose **More**, and then choose **Create pull request**.
5. The repository and the source branch are preselected for you. In **Destination branch**, choose the branch where you will merge the code once it has been reviewed. In **Pull request title**, enter a title that will help other project users understand what must be reviewed and why. Optionally, provide more information in **Pull request description**, such as pasting in a link to related issues in CodeCatalyst, or adding a description of the changes you made.

> **Note**
>
> Workflows that are configured to run for pull request create events will run after the pull request is created, if the destination branch for the pull request matches one of the branches specified in the workflow.

6. Review the differences between the branches. If no changes are displayed, the branches might be identical, or you might have chosen the same branch for both the source and the destination.
7. (Optional) In **Issues**, choose **Link issues**, and then either choose an issue from the list or enter its ID. To unlink an issue, choose the unlink icon.
8. (Optional) In **Required reviewers**, choose **Add required reviewers**. Choose from the list of project members to add them. Required reviewers must approve the changes before the pull request can be merged into the destination branch.

> **Note**
>
> You can't add a reviewer as both required and optional. You can't add yourself as a reviewer.

9. (Optional) In **Optional reviewers**, choose **Add optional reviewers**. Choose from the list of project members to add them. Optional reviewers do not have to approve the changes before the pull request can be merged into the destination branch.
10. When you are satisfied that the pull request contains the changes that you want reviewed and includes the required reviewers, choose **Create**.

If you have any workflows configured to run where the branch matches the destination branch in the pull request, you will see information about those workflow runs in **Overview** in the **Pull request details** area after the pull request is created. For more information, see Adding a push, pull, or schedule trigger.

Viewing pull requests

You can view pull requests for a project in the Amazon CodeCatalyst console. The project summary page displays all open pull requests for a project. To view all pull requests regardless of state,

navigate to the pull requests page for your project. When viewing a pull request, you can choose to have a summary of all comments left on changes to the pull request created for you.

> **Note**
>
> **Powered by Amazon Bedrock**: AWS implements automated abuse detection. Because the **Write description for me** and **Create content summary** features are built on Amazon Bedrock, users can take full advantage of the controls implemented in Amazon Bedrock to enforce safety, security, and the responsible use of artificial intelligence (AI).

To view open pull requests

1. Navigate to the project where you want to view pull requests.
2. On the project page, open pull requests are displayed, including information about who created the pull request, what repository contains the branches for the pull request, and the date the pull request was created. You can filter the open pull request view by source repository.
3. To view all pull requests, choose **View all**. You can use the selectors to choose between the options. For example, to view all pull requests, choose **Any status** and **Any author**.

   Alternatively, in the navigation pane, choose **Code**, and then choose **Pull requests**, and then use the selectors to refine your view.
4. On the **Pull requests** page, you can sort pull requests by ID, title, status, and more. To customize what information and how much information is displayed on the pull requests page, choose the gear icon.
5. To view a specific pull request, choose it from the list.
6. To view the status of workflows runs associated with this pull request, if any, choose **Overview** and review the information in the **Pull request details** area of the pull request under **Workflow runs**.

   A workflow run will occur if the workflow is configured with pull request create or revision events, and if the destination branch requirements in the workflow match the destination branch specified in the pull request. For more information, see Adding a push, pull, or schedule trigger.
7. To view linked issues, if any, choose **Overview** and review the information in the **Pull request details** under **Issues**. If you want to view a linked issue, choose its ID from the list.

8. (Optional) To create a summary of comments left on the code changes in revisions of the pull request, choose **Create content summary**. The summary will not include any comments left on the overall pull request.

> **Note**
>
> This functionality requires that generative AI features are enabled for the space, is not available for linked repositories, and is only available in the US West (Oregon) Region. For more information, see Managing generative AI features.

9. To view the code changes in the pull request, choose **Changes**. You can quickly view how many files have changes in the pull request, and what files in the pull request have comments on them, in **Files changed**. The number of comments shown next to a folder indicates the number of comments on files in that folder. Expand the folder to view the number of comments for each file in the folder. You can also view any comments left on specific lines of code.

> **Note**
>
> Not all changes in a pull request can be displayed in the console. For example, you cannot view Git submodules in the console, so you cannot view differences in a submodule in a pull request. Some differences might be too large to display. For more information, see Quotas for source repositories in CodeCatalyst and Viewing a file.

10. To view quality reports for this pull request, choose **Reports**.

> **Note**
>
> A workflow must be configured to generate reports in order for them to show up in your pull requests. For more information, see Testing with workflows.

Managing requirements for merging a pull request with approval rules

When you create a pull request, you can choose to add required or optional reviewers to that individual pull request. However, you can also create requirements that all pull requests must meet when merging to a specific destination branch. These requirements are called approval rules. Approval rules are configured for branches in a repository. When you create a pull request whose

destination branch has an approval rule configured for it, the requirements for that rule must be met in addition to approvals from any required reviewers before you can merge the pull request to that branch. Creating approval rules can help you maintain quality standards for merges to branches such as your default branch.

Approval rules that are applied to the default branch of your source repository will behave a little differently than approval rules applied to other branches. Any rule applied to the default branch will be automatically applied to any branch you specify as the default branch. The branch that was formerly set as the default branch will still keep the rules applied to it.

When you create approval rules, you should consider how that rule will be met by your project users both in the present and in the future. For example, if you have six users in your project, and you create an approval rule that requires five approvals before it can be merged to the destination branch, you have effectively created a rule that requires everyone except the person who created the pull requets to approve that pull request before it can be merged.

> **Note**
>
> You must have the Project administrator role to create and manage approval rules in CodeCatalyst projects. You cannot create approval rules for linked repositories.

You cannot delete approval rules, but you can update them to require zero approvals, which effectively removes the rule.

To view and edit approval rules for destination branches for pull requests

1. Navigate to the project where your repository resides.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to view approval rules.
3. On the overview page of the repository, choose **Branches**.
4. In the **Approval rules** column, choose **View** to see the status of any rules for each branch of the repository.

   In **Minimum number of approvals**, the number corresponds to the number of approvals required before a pull request can be merged to that branch.

5. To create or change an approval rule, choose **Manage settings**. On the settings page for the source repository, in **Approval rules**, choose **Edit**.

   > **Note**
   >
   > You must have the **Project administrator** role to edit approval rules.

6. In **Branch**, choose the name of the branch for which you want to configure an approval rule from the drop-down list. In **Minimum number of approvals**, enter a number, and then choose **Save**.

Reviewing a pull request

You can use the Amazon CodeCatalyst console to collaboratively review and comment on the changes included in a pull request. You can add comments to individual lines of code in the difference between the source and destination branches, or in the difference between revisions of the pull request. You can choose to create a summary of comments left on the code changes in the pull request to help you quickly understand the feedback left by other users. You can also choose to create a Dev Environment to work on code.

> **Note**
>
> **Powered by Amazon Bedrock**: AWS implements automated abuse detection. Because the **Write description for me** and **Create content summary** features are built on Amazon Bedrock, users can take full advantage of the controls implemented in Amazon Bedrock to enforce safety, security, and the responsible use of artificial intelligence (AI).

> **Tip**
>
> You can configure what pull request events you that will receive emails about as part of your profile. For more information, see Managing notifications in Amazon CodeCatalyst.

Pull requests show what the difference will be between the revision of the pull request and the commit that was the tip of the destination branch when you created the pull request. This is called

the merge base. For more information about differences and merge bases in Git, see git-merge-base in the Git documentation.

> **Tip**
>
> When working in the console, particularly if you have had a pull request open for a while, consider refreshing your browser to ensure you have the latest revision available for a pull request before you start to review it.

To review a pull request in the CodeCatalyst console

1. Navigate to your project.
2. Navigate to the pull requests by doing one of the following:
   - If the pull request is listed on the project page, choose it from the list.
   - If the pull request is not listed on the project page, choose **View all**. Use the filters and sort to find the pull request, and then choose it from the list.
   - In the navigation pane, choose **Code**, and then choose **Pull requests**.
3. Choose the pull request you want to review from the list. You can filter the list of pull requests by typing part of its name in the filter bar.
4. In **Overview**, you can review the name and title of the pull request. You can create and view comments left on the pull request itself. You can also view the details of the pull request, including information about workflow runs, linked issues, reviewers, the author of the pull request, and feasible merge strategies.

   > **Note**
   >
   > Comments left on specific lines of code appear in **Changes**.

5. (Optional) To add a comment that applies to the entire pull request, expand **Comments on pull request**, and then choose **Create comment**.
6. (Optional) To view a summary of all comments left on changes in revisions of this pull request, choose **Create comment summary**.

> **Note**
>
> This functionality requires that generative AI features are enabled for the space and is only available in the US West (Oregon) Region. For more information, see Managing generative AI features.

7. In **Changes**, you can see the differences between the destination branch and the most recent revision of the pull request. If there is more than one revision, you can change what revisions are compared in the difference between them. For more information about revisions, see Revisions.

> **Tip**
>
> You can quickly view how many files have changes in the pull request, and what files in the pull request have comments on them, in **Files changed**. The number of comments shown next to a folder indicates the number of comments on files in that folder. Expand the folder to view the number of comments for each file in the folder.

8. To change the way differences are displayed, choose between **Unified** and **Split**.
9. To add a comment to a line in the pull request, go to the line you want to comment on. Choose the comment icon that appears for that line, enter a comment, and then choose **Save**.
10. To view changes between revisions in a pull request, or between its source and destination branches, choose from the available options in **Comparing**. Comments on lines in revisions are preserved in those revisions.
11. If you've configured your workflow to generate a code coverage report on pull request triggers, you can view the line and branch coverage findings in the relevant pull request. To hide code coverage findings, choose **Hide code coverage**. For more information, see Code coverage reports.
12. If you want to make code changes to the pull request, you can create a Dev Environment from the pull request. Choose **Create Dev Environment**. Optionally add a name for the Dev Environment or edit its configuration and then choose **Create**.
13. In **Reports**, you can view the quality reports in this pull request. If there is more than one revision, you can change what revisions are compared in the difference between them. You can filter the reports by name, status, workflow, action, and type.

> **Note**
>
> A workflow must be configured to generate reports in order for them to show up in your pull requests. For more information, see Configuring quality reports in an action.

14. To view a specific report, choose it from the list. For more information, see Testing with workflows.
15. If you are listed as a reviewer of this pull request and want to approve the changes, make sure that you are viewing the most recent revision, and then choose **Approve**.

> **Note**
>
> All required reviewers must approve a pull request before it can be merged.

Updating a pull request

You can make it easier for other project members to review code by updating the pull request. You can update a pull request to change its reviewers, its links to issues, the title of the pull request, or its description. For example, you might want to change the required reviewers for a pull request to remove someone who's away on vacation, and add someone else. You can also update a pull request with further code changes by pushing commits to the source branch of an open pull request. Each push to the source branch of a pull request in the CodeCatalyst source repository creates a revision. Project members can view the differences between revisions in a pull request.

To update the reviewers for a pull request

1. Navigate to the project where you want to update the reviewers of a pull request.
2. On the project page, under **Open pull requests**, choose the pull request where you want to update reviewers. Alternatively, in the navigation pane, choose **Code**, choose **Pull requests**, and then choose the pull request you want to update.
3. (Optional) In **Overview**, in the **Pull request details** area, choose the plus sign to add required or optional reviewers. Choose the **X** next to a reviewer to remove them as an optional or required reviewer.

4. (Optional) In **Overview**, in the **Pull request details** area, choose **Link issues** to link an issue to the pull request, and then either choose an issue from the list or enter its ID. To unlink an issue, choose the unlink icon next to the issue you want to unlink.

To update files and code in the source branch of a pull request

1. To update multiple files, either create a Dev Environment, or clone the repository and its source branch and use a Git client or an integrated development environment (IDE) to make changes to the files in the source branch. Commit and push the changes to the source branch in the CodeCatalyst source repository to automatically update the pull request with the changes. For more information, see Cloning a source repository and Understanding changes in source code with commits in Amazon CodeCatalyst.
2. To update an individual file in a source branch, you can use a Git client or IDE as you would for multiple files. You can also edit it directly in the CodeCatalyst console. For more information, see Editing a file.

To update the title and description of a pull request

1. Navigate to the project where you want to update the title or description of a pull request.
2. The project page displays open pull requests, including information about who created the pull request, what repository contains the branches for the pull request, and when the pull request was created. You can filter the open pull request view by source repository. Choose the pull request that you want to change from the list.
3. To view all pull requests, choose **View all**. Alternatively, in the navigation pane, choose **Code**, and then choose **Pull requests**. Use the filter box or sort functions to find the pull request you want to change, and then choose it.
4. In **Overview**, choose **Edit**.
5. Change the title or description, and then choose **Save**.

Merging a pull request

After your code has been reviewed and all required reviewers have approved it, you can merge a pull request in the CodeCatalyst console using a supported merge strategy, such as fast-forward. Not all merge strategies supported in the CodeCatalyst console are available as choices for all pull requests. CodeCatalyst evaluates the merge and only allows you to choose between merge

strategies that are available in the console and capable of merging the source branch into the destination branch. You can also merge a pull request with your choice of Git merge strategies by running the **git merge** command on your local computer or a Dev Environment to merge the source branch into the destination branch. You can then push those changes in the destination branch to the source repository in CodeCatalyst.

> **Note**
>
> Merging the branch and pushing the changes in Git does not automatically close the pull request.

If you have the Project administrator role, you can also choose to merge a pull request that has not yet met all the requirements for approvals and approval rules.

Merging a pull request (console)

You can merge a pull request in the CodeCatalyst console if there are no merge conflicts between the source and destination branches and if all required reviewers have approved the pull request. If there are conflicts, or if the merge can't be completed, the merge button is inactive, and a **Not mergeable** label is displayed. In that case, you must obtain approval from any required approvers, resolve conflicts locally if necessary, and push those changes before you can merge. Merging a pull request will automatically send an email to the creator of the pull request as well as any required or optional reviewers. It will not automatically close or change the status of any issues linked to the pull request.

> **Tip**
>
> You can configure what pull request events you that will receive emails about as part of your profile. For more information, see Managing notifications in Amazon CodeCatalyst.

To merge a pull request

1. Navigate to the project where you want to merge a pull request.
2. On the project page, under **Open pull requests**, choose the pull request you want to merge. If you do not see the pull request, choose **View all pull requests** and then choose it from the list. Alternatively, in the navigation pane, choose **Code**, choose **Pull requests**, and then choose the pull request you want to merge. Choose **Merge**.

3. Choose from the available merge strategies for the pull request. Optionally, select or deselect the option to delete the source branch after merging the pull request, and then choose **Merge**.

> **Note**
>
> If the **Merge** button is inactive, or you see the **Not mergeable** label, either required reviewers have not yet approved the pull request, or the pull request can't be merged in the CodeCatalyst console. A reviewer who has not approved a pull request is indicated by a clock icon in the **Pull request details** area in **Overview**. If all required reviewers have approved the pull request but the **Merge** button is still inactive, you might have a merge conflict. Choose the underlined **Not mergeable** label to see more details about why the pull request can't be merged. You can resolve merge conflicts for the destination branch in a Dev Environment or the CodeCatalyst console and then merge the pull request, or you can resolve conflicts and merge locally, and then push the commit that contains the merge to the source branch in CodeCatalyst. For more information, see Merging a pull request (Git) and your Git documentation.

Override merge requirements

If you have the **Project administrator** role, you can choose to merge a pull request that has not yet met all the requirements for required approvals and approval rules. This is referred to as overriding the requirements for a pull request. You might choose to do this if a required reviewer is unavailable, or if an urgent need arises to merge a specific pull request into a branch that has approval rules that cannot be met quickly.

To merge a pull request

1. In the pull request where you want to override requirements and merge, choose the drop-down arrow next to the **Merge** button. Choose **Override approval requirements**.
2. In **Override reason**, provide details of why you are merging this pull request without it meeting the approval rules and required reviewer requirements. While this is optional, this is highly recommended.
3. Optionally choose a merge strategy, or accept the default. You can also choose to update the automatically-generated commit message with more details.

4. Select or deselect the option to delete the source branch on merge. We recommend that you retain the source branch when overriding the requirements for merging a pull request until you've had a chance to review the decision with other team members.
5. Choose **Merge**.

Merging a pull request (Git)

Git supports many options for merging and managing branches. The following commands are some of the options that you can use. For more information, see the available documentation on the Git website. Once you have merged and pushed your changes, manually close the pull request. For more information, see Closing a pull request.

Common Git commands for merging branches

Merges changes from the source branch in the local repo to the destination branch in the local repo.	`git checkout destination-branch-name`
	
	`git merge source-branch-name`

Merges the source branch into the destination branch, specifying a fast-forward merge. This merges the branches and moves the destination branch pointer to the tip of the source branch.	`git checkout destination-branch-name`
	
	`git merge --ff-only  source-branch-name`

Merges the source branch into the destination branch, specifying a squash merge. This combines all commits from the source branch into a single merge commit in the destination branch.	`git checkout destination-branch-name`
	
	`git merge --squash source-branch-name`

Merges the source branch into the destination branch, specifying a three-way merge. This creates a merge commit and adds the individual commits from the source branch to the destination branch.	`git checkout destination-branch-name`
	
	`git merge --no-ff source-branch-name`

Deletes the source branch in the local repo. This is useful to do as a clean-up for your	`git branch -d source-branch-name`

local repo after merging to the destination branch and pushing the changes to the source repository.	
Deletes the source branch in the remote repository (the source repository in CodeCatalyst) using the local repo's specified nickname for the remote repository. (Note the use of the colon (`:`).) Alternatively, specify `--delete` as part of the command.	`git push` *`remote-name`* `:`*`source-branch-name`* `git push` *`remote-name`* `--delete` *`source-branch-name`*

Closing a pull request

You can mark a pull request as **Closed**. This does not merge the pull request, but it can help you determine which pull requests require action and which pull requests are no longer relevant. We recommend closing a pull request if you no longer plan to merge those changes, or if the changes were merged by another pull request.

Closing a pull request will automatically send an email to the creator of the pull request as well as any required or optional reviewers. It will not automatically change the status of any issues linked to the pull request.

> **Note**
>
> You cannot re-open a pull request after it has been closed.

To close a pull request

1. Navigate to the project where you want to close a pull request.
2. On the project page, open pull requests are displayed. Choose the pull request that you want to close.
3. Choose **Close**.
4. Review the information, and then choose **Close pull request**.

Understanding changes in source code with commits in Amazon CodeCatalyst

Commits are snapshots of the contents and changes to the contents of your repository. Every time a user commits and pushes a change to a branch, that information is saved. Git commit information includes the commit author, the person who committed the change, the date and time, and the changes made. Similar information is automatically included when you create or edit a file in the Amazon CodeCatalyst console, but the author name is your CodeCatalyst user name. You can also add Git tags to commits to help you identify specific commits.

In Amazon CodeCatalyst, you can:

- View a list of commits for a branch.
- View individual commits, including the changes made in a commit when compared to its parent or parents.

You can also view files and folders. For more information, see Managing source code files in Amazon CodeCatalyst.

Topics

Viewing commits to a branch

You can view the history of changes made to a branch by reviewing the branch's commits in the CodeCatalyst console. This helps you understand who made changes to the branch and when. You can also review the changes made in a specific commit.

> **Tip**
>
> You can also view the history of commits that made changes to a specific file. For more information see Viewing a file.

You can also view commits by using your Git client. For more information, see your Git documentation.

To view commits (console)

1. Navigate to the project that contains the source repository where you want to view commits.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to view commits to a branch.
3. The default branch of the repository is displayed, including information about the most recent commit to the branch. Choose **Commits**. Alternatively, choose **More**, and then choose **View commits**.
4. To view commits for a different branch, choose the branch selector, and then choose the name of the branch.
5. To view details about a particular commit, choose its title from **Commit title**. Details of the commit are displayed, including information about its parent commit and the changes made to the code by comparing the parent commit to the specified commit.

   > **Tip**
   >
   > If a commit has more than one parent, you can choose which parent commit to view information and display changes for by choosing the drop-down icon next to the parent commit ID.

Changing how commits are displayed (CodeCatalyst console)

You can change what information is displayed in the **Commits** view. You can choose to hide or display columns such as author and commit ID.

To change how commits are displayed (console)

1. Navigate to the project that contains the source repository where you want to view commits.
2. Choose the name of the repository from the list of source repositories for the project. Alternatively, in the navigation pane, choose **Code**, and then choose **Source repositories**.

   Choose the repository where you want to change how you view commits.
3. The default branch of the repository is displayed, including information about the most recent commit to the branch. Choose **Commits**.

4. Choose the gear icon.
5. In **Preferences**, choose the number of commits to display, and choose whether to display information about commit author, commit date, and the commit ID.

> **Note**
>
> You can't hide the commit title in the display of information.

6. When you have made your changes, choose **Save** to save them, or **Cancel** to discard them.

Quotas for source repositories in CodeCatalyst

The following table describes quotas and limits for source repositories in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Resource	Information
Branch names	Any combination of allowed characters between 1 and 256 characters in length and must be unique within a repository. Branch names cannot: • begin or end with a slash (/) or period (.) • consist of the single character @ • contain two or more consecutive periods (..), forward slashes (//), or the following character combination: @{ • contain spaces or any of the following characters: ? ^ * [\ ~ : Branch names are references. Many of the limitations on branch names are based on the Git reference standard. For more information, see Git Internals and git-check-ref-format.
Comments on a pull request	Maximum of 1,000 on a pull request.

Resource	Information
Commit message	Maximum of 1024 characters.

File paths	Any combination of allowed characters between 1 and 4,096 characters in length. File paths must be an unambiguous name that specifies the file and the exact location of the file. File paths cannot exceed 20 directories in depth. In addition, file paths cannot:
	• contain empty strings
	• be a relative file path
	• include any of the following character combinations:
	`/./`
	`/../`
	`//`
	• end with a trailing slash or backslash
	
	File names and paths must be fully qualified . The name and path to a file on your local computer must follow the standards for that operating system. When specifying the path to a file in a repository, use the standards for Amazon Linux.

File size	Maximum of 6 MB for any individual file when using the CodeCatalyst console.
File size viewable in the CodeCatalyst console	Maximum of 6 MB for any individual file when using the CodeCatalyst console.

Resource	Information
Git blob size	Maximum of 2 GB. **Note** There is no limit on the number or the total size of all files in a single commit, as long as the metadata does not exceed 6 MB and a single blob does not exceed 2 GB. However, as a best practice, consider making multiple smaller commits rather than one large commit.
Metadata for a commit	Maximum of 6 MB for the combined metadata for a commit (for example, the combination of author information, date, parent commit list, and commit messages). **Note** There is no limit on the number or the total size of all files in a single commit, as long as the data does not exceed 20 MB, an individual file does not exceed 6 MB, and a single blob does not exceed 2 GB.
Number of CodeCatalyst issues that can be linked to a pull request	50
Number of Jira issues that can be linked to a pull request	50
Number of open pull requests in a space	Maximum of 1,000 for an Amazon CodeCatal yst space.

Resource	Information
Number of total pull requests in a space	Maximum of 10,000 for an Amazon CodeCatalyst space.
Number of references in a single push	Maximum of 4,000, including create, delete, and update. There is no limit on the overall number of references in the repository.
Number of repositories in a space	Maximum of 5,000 for an Amazon CodeCatalyst space.
Repository descriptions	Any combination of characters between 0 and 1,000 characters in length. Repository descriptions are optional.
Repository names	Repository names must be unique within a project. They can contain any combination of letters, numbers, periods, underscores, and dashes between 1 and 100 characters in length. Names are not case sensitive. Repository names cannot end in .git, cannot contain spaces, and cannot contain any of the following characters: `! ? @ # $ % ^ & * ( ) + = { } [ ] | \ / > < ~ ` ' " ; :`
Repository size	Repository sizes are impacted by the overall storage limits for your space. For more information, see Pricing and Troubleshooting problems with source repositories.
Reviewers for a pull request	Maximum of 100 reviewers total (optional or required) for a pull request.

Resource	Information
Written summaries for pull requests	The maximum number of written summaries for pull requests depends on the billing tier for your space. For more information, see Pricing.

Write and modify code with Dev Environments in CodeCatalyst

Dev Environments are cloud-based development environments. In Amazon CodeCatalyst, you use Dev Environments to work on the code stored in the source repositories of your project. When creating a Dev Environments, you have several options:

- Create a project-specific Dev Environment in CodeCatalyst to work on code with a supported integrated development environment (IDE).
- Create an empty Dev Environment, clone code into it from a source repository, and work on that code with a supported IDE.
- Create a Dev Environment in an IDE and clone a source repository into the Dev Environment

A *devfile* is an open standard YAML file that standardizes your Dev Environments. In other words, this file codifies the required development tools for your Dev Environment. As a result, you can quickly set up a Dev Environment, switch between projects, and replicate the Dev Environment configuration across team members. Dev Environments minimize the time that you spend creating and maintaining a local development environment, because they use a devfile that configures all of the tools you need to code, test, and debug for a given project.

The project tools and application libraries included in your Dev Environment are defined by the devfile in the source repository of your project. If you don't have a devfile in your source repository, CodeCatalyst automatically applies a default devfile. This default devfile includes tools for the most frequently used programming languages and frameworks. If your project was created using a blueprint, a devfile is automatically created by CodeCatalyst. For more information about the devfile, see https://devfile.io.

After you've created a Dev Environment, only you can access it. In your Dev Environment, you can view and work on your source repository's code in a supported IDE.

By default, a Dev Environment is configured to have a 2-core processor, 4 GB of RAM, and 16 GB of persistent storage. If you have Space administrator permissions, you can change the billing tier for your space to use different Dev Environment configuration options and manage compute and storage limits.

Topics

Creating a Dev Environment

You can create a Dev Environment in multiple ways:

- Create a Dev Environment in CodeCatalyst with a CodeCatalyst source repository or a linked source repository from the **Overview**, **Dev Environments** or **Source repositories** pages
- Create an empty Dev Environment in CodeCatalyst that is not connected to a source repository from the Dev Environments page
- Create a Dev Environment in your IDE of choice and clone any source repository into the Dev Environment

You can create one Dev Environment per branch of a repository. A project can have multiple repositories. The Dev Environments you create can only be managed with your CodeCatalyst account, but you can open the Dev Environment and work in it with any of the supported IDEs. You must have the AWS Toolkit installed to use Dev Environments in your IDE. For more information, see Supported integrated development environments for Dev Environments. By default, Dev Environments are created with a 2-core processor, 4 GB of RAM, and 16 GB of persistent storage.

> **Note**
>
> If you created a Dev Environment that is associated with a source repository, the **Resource** column always shows the branch you specified when creating this Dev Environment. This applies even if you create another branch, switch to another branch within the Dev

Environment, or clone an additional repository. If you created an empty Dev Environment, the **Resource** column will be blank.

Supported integrated development environments for Dev Environments

You can use Dev Environments with the following supported integrated development environments (IDEs):

- AWS Cloud9
- JetBrains IDEs
  - IntelliJ IDEA Ultimate
  - GoLand
  - PyCharm Professional
- Visual Studio Code

Creating a Dev Environment in CodeCatalyst

To get started working with Dev Environment in CodeCatalyst, authenticate and sign in with either your AWS Builder ID or SSO.

To create a Dev Environment from a branch

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to create a Dev Environment.
3. In the navigation pane, do one of the following:
   - Choose **Overview**, and then navigate to the **My Dev Environments** section.
   - Choose **Code**, and then choose **Dev Environments**.
   - Choose **Code**, choose **Source repositories**, and choose the repository for which you want to create a Dev Environment.
4. Choose **Create Dev Environment**.
5. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.

6. Choose **Clone a repository**.
7. Do one of the following:

   a. Choose the repository to clone, choose **Work in existing branch**, and then choose a branch from the **Existing branch** drop-down menu.

   > **Note**
   >
   > If you choose a third-party repository, you must work in an existing branch.

   b. Choose the repository to clone, choose **Work in new branch**, enter a branch name into the **Branch name** field, and choose a branch off of which to create the new branch from the **Create branch from** drop-down menu.

   > **Note**
   >
   > If you create a Dev Environment from the **Source repositories** page or from a specific source repository, you do not need to choose a repository. The Dev Environment will be created from the source repository you chose from the **Source repositories** page.

8. (Optional) In **Alias - optional**, enter an alias for the Dev Environment.
9. (Optional) Choose the **Dev Environment configuration** edit button to edit the Dev Environment's compute, storage, or timeout configuration.
10. (Optional) In **Amazon Virtual Private Cloud (Amazon VPC) - optional**, select a VPC connection that you'd like to associate with your Dev Environment from the drop-down menu.

    If a default VPC is set for your space, your Dev Environments will run connected to that VPC. You can override this by associating a different VPC connection. Also, note that VPC-connected Dev Environments don't support AWS Toolkit.

    > **Note**
    >
    > When you create a Dev Environment with a VPC connection, a new network interface is created inside the VPC. CodeCatalyst interacts with this interface using the associated VPC role. Also, make sure that your IPv4 CIDR block is **not** configured to the `172.16.0.0/12` IP address range.

11. Choose **Create**. While your Dev Environment is being created, the Dev Environment status column will display **Starting**, and the status column will display **Running** once the Dev Environment has been created.

To create an empty Dev Environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to create a Dev Environment.
3. In the navigation pane, do one of the following:
   - Choose **Overview**, and then navigate to the **My Dev Environments** section.
   - Choose **Code**, and then choose **Dev Environments**.
4. Choose **Create Dev Environment**.
5. Choose a supported IDE from the drop-down menu. See Supported integrated development environments for Dev Environments for more information.
6. Choose **Create an empty Dev Environment**.
7. (Optional) In **Alias - optional**, enter an alias for the Dev Environment.
8. (Optional) Choose the **Dev Environment configuration** edit button to edit the Dev Environment's compute, storage, or timeout configuration.
9. (Optional) In **Amazon Virtual Private Cloud (Amazon VPC) - optional**, select a VPC connection that you'd like to associate with your Dev Environment from the drop-down menu.

   If a default VPC is set for your space, your Dev Environments will run connected to that VPC. You can override this by associating a different VPC connection. Also, note that VPC-connected Dev Environments don't support AWS Toolkit.

   > **Note**
   >
   > When you create a Dev Environment with a VPC connection, a new network interface is created inside the VPC. CodeCatalyst interacts with this interface using the associated VPC role. Also, make sure that your IPv4 CIDR block is **not** configured to the `172.16.0.0/12` IP address range.

10. Choose **Create**. While your Dev Environment is being created, the Dev Environment status column will display **Starting**, and the status column will display **Running** once the Dev Environment has been created.

Note

Creating and opening a Dev Environment for the first time might take one to two minutes.

Note

After the Dev Environment opens in the IDE, you might need to change the directory to the source repository before you commit and push changes to your code.

Creating a Dev Environment in an IDE

You can use Dev Environments to quickly work on the code stored in the source repositories of your project. Dev Environments increase your development velocity because you can start coding immediately in a project-specific, fully functioning cloud development environment with a supported integrated development environment (IDE).

For information about working with CodeCatalyst from an IDE, see the following documentation.

- Amazon CodeCatalyst for JetBrains IDEs
- Amazon CodeCatalyst for VS Code
- Amazon CodeCatalyst for AWS Cloud9

Stopping a Dev Environment

The `/projects` directory of a Dev Environment stores the files that are pulled from the source repository and the devfile that is used to configure the Dev Environment. The `/home` directory, which is empty upon Dev Environment creation, stores the files you create while using your Dev Environment. Everything in the `/projects` and `/home` directories of a Dev Environment is stored persistently, so you can stop working in a Dev Environment if you need to switch to another Dev Environment, repository, or project.

Warning

A Dev Environment will not timeout if any instances, including web browsers, remote shells, and IDEs, remain connected. So, make sure to close all connected instances to avoid incurring additional costs.

A Dev Environment will automatically stop if it is idle for the amount of time that was selected in the **Timeout** fields during Dev Environment creation. You can stop the Dev Environment before it goes idle. If you chose **No timeout** when you created your Dev Environment, the Dev Environment will not stop automatically. Instead, it will run continuously.

Warning

If you stop a Dev Environment that's associated with a deleted VPC connection, it can't be resumed.

To stop a Dev Environment from the Dev Environments page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to stop a Dev Environment.
3. In the navigation pane, choose **Code**.
4. Choose **Dev Environments**.
5. Choose the radio button for the Dev Environment you want to stop.
6. From the **Actions** menu, choose **Stop**.

Note

Compute use is billed only while the Dev Environment is running, but storage use is billed for the entire time that the Dev Environment exists. Stop your Dev Environment when it is not in use to stop compute billing.

Resuming a Dev Environment

The `/projects` directory of a Dev Environment stores the files that are pulled from the source repository and the devfile that is used to configure the Dev Environment. The `/home` directory, which is empty upon Dev Environment creation, stores the files you create while using your Dev Environment. Everything in the `/projects` and `/home` directories of a Dev Environment is stored persistently, so you can stop working in a Dev Environment if you need to switch to another Dev Environment, repository, or project and resume working in your Dev Environment at a later time.

A Dev Environment will automatically stop if it is idle for the amount of time that was selected in the **Timeout** fields during Dev Environment creation. You must close the AWS Cloud9 browser tab for the Dev Environment to be idle.

> **Note**
>
> The Dev Environment is still available and running even if you delete the branch with which you created the Dev Environment. If you want to resume working in a Dev Environment for which you deleted the branch, create a new branch and push your changes to it.

To resume a Dev Environment from the overview page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to resume a Dev Environment, and navigate to the **My Dev Environments** section.
3. Choose **Resume in (IDE)**.
   - For JetBrains IDEs, choose JetBrains Gateway-EAP when prompted to **Choose an application to open the JetBrains-gateway link**. Choose **Open Link** to confirm when prompted.
   - For the VS Code IDE, choose VS Code when prompted to **Choose an application to open the VS Code link**. Choose **Open Link** to confirm.

To resume a Dev Environment from the source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to resume a Dev Environment.
3. In the navigation pane, choose **Code**.

4. Choose **Source Repositories**.
5. Choose the source repository that contains the Dev Environment you want to resume.
6. Choose the branch name to view a drop-down menu of your branches, then choose your branch.
7. Choose **Resume Dev Environment**.

   - For JetBrains IDEs, choose **Open Link** to confirm when prompted to **Allow this site to open the JetBrains-gateway link with JetBrains Gateway?**.
   - For the VS Code IDE, choose **Open Link** to confirm when prompted to **Allow this site to open the VS Code link with Visual Studio Code?**.

To resume a Dev Environment from the Dev Environments page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to resume a Dev Environment.
3. In the navigation pane, choose **Code**.
4. Choose **Dev Environments**.
5. From the **IDE** column, choose **Resume in (IDE)** for the Dev Environment.

   - For JetBrains IDEs, choose **Open Link** to confirm when prompted to **Allow this site to open the JetBrains-gateway link with JetBrains Gateway?**.
   - For the VS Code IDE, choose **Open Link** to confirm when prompted to **Allow this site to open the VS Code link with Visual Studio Code?**.

> **Note**
>
> Resuming a Dev Environment may take a few minutes.

Editing a Dev Environment

While your IDE is running, you can edit the Dev Environment. If you edit your compute or inactivity timeout, your Dev Environment will restart after you've saved your changes.

To edit a Dev Environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to edit a Dev Environment.
3. In the navigation pane, choose **Code**.
4. Choose **Dev Environments**.
5. Choose the Dev Environment you want to edit.
6. Choose **Edit**.
7. Make the changes you want to the compute or inactivity timeout.
8. Choose **Save**.

Deleting a Dev Environment

When you have finished working on the content that is stored in your Dev Environment, you can delete the Dev Environment. Create a new Dev Environment to work on new content. If you delete your Dev Environment, the persisted content will be permanently deleted. Before you delete your Dev Environment, make sure you commit and push your code changes to the Dev Environment's original source repository. After you have deleted your Dev Environment, compute and storage billing for the Dev Environment will stop.

After you delete your Dev Environment, it may take a few minutes for the storage quota to be updated. If you've reached the storage quota, you will be unable to create a new Dev Environment during this time.

> **Important**
>
> Deleting a Dev Environment cannot be undone. After you delete a Dev Environment, you are no longer able to recover it.

To delete a Dev Environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to delete a Dev Environment.
3. In the navigation pane, choose **Code**.

4. Choose **Dev Environments**.
5. Choose the Dev Environment you want to delete.
6. Choose **Delete**.
7. Enter **`delete`** to confirm the Dev Environment deletion.
8. Choose **Delete**.

> **Note**
>
> Before deleting a VPC connection in your space, make sure to remove the Dev Environment associated to that VPC.
> Even if you delete a Dev Environment, you might not delete the network interface in the VPC. Make sure to clean up your resources as needed. If an error occurs when you delete a VPC-connected Dev Environment, you must detach your stale connection, and delete it after confirming that it's not being used.

Connecting to a Dev Environment using SSH

You can connect to your Dev Environment using SSH to perform actions without limitations, such as port forwarding, uploading and downloading files, and using other IDEs.

> **Note**
>
> If you want to continue using SSH for an extended time after closing the IDE tab or window, make sure to set a high timeout for your Dev Environment so that it doesn't stop due to inactivity in the IDE.

Prerequisites

- You need one of the following operating systems:
  - Windows 10 or newer and OpenSSH enabled
  - macOS and Bash version 3 or higher
  - Linux with `yum`, `dpkg` or `rpm` package managers and Bash version 3 or higher
- You also need AWS CLI version 2.9.4 or higher.

To connect to a Dev Environment using SSH

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to connect to a Dev Environment using SSH.
3. In the navigation pane, choose **Code**.
4. Choose **Dev Environments**.
5. Choose a running Dev Environment you want to connect to using SSH.
6. Choose **Connect via SSH**, choose your desired operating system, and do the following:

   - If you haven't done so already, paste and execute the first command in your specified terminal. The command downloads a script and executes the following modifications in your local environment so that you can connect to your Dev Environment using SSH:
     - Installs the Session Manager plugin for the AWS CLI
     - Modifies your local AWS Config and adds a CodeCatalyst profile so that you're able to perform the SSO login. For more information, see Setting up to use the AWS CLI with CodeCatalyst.
     - Modifies your local SSH config and adds the required configuration for connecting to your Dev Environment using SSH.
     - Adds a script in the `~/.aws/codecatalyst-dev-env` directory that is used by the SSH client to connect to your Dev Environment. This script calls the CodeCatalyst StartDevEnvironmentSession API and uses AWS Systems Manager Session Manager plugin to establish an AWS Systems Manager session with your Dev Environment which is used by the local SSH client to securely connect to the remote Dev Environment.
   - Sign-in to Amazon CodeCatalyst using AWS SSO using the second command. This command requests and retrieves credentials so that the script in the `~/.aws/codecatalyst-dev-env` directory can call CodeCatalyst StartDevEnvironmentSession API. This command should be executed every time your credentials expire. When you execute the last command in the modal(ssh <destination>) you will get an error if your credentials are expired or you haven't performed the SSO login as instructed in this step.
   - Connect to your specified Dev Environment using SSH using the third command. This command has the following structure:

     ```
     ssh codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
     ```

You can also use this command to perform other actions allowed by the SSH client, such as port forwarding or uploading and downloading files:

- Port forwarding:

```
ssh -L <local-port>:127.0.0.1:<remote-port> codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

- Uploading a file to the home directory in your Dev Environment:

```
scp -O </path-to-local-file> codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>:</path-to-remote-file-or-directory>
```

Configuring a devfile for a Dev Environment

A *devfile* is an open standard that helps you to customize your development Dev Environments across your team. A devfile is a YAML file that codifies your required development tools. By configuring a devfile, you can pre-determine the project tools and application libraries you need and Amazon CodeCatalyst installs them to your Dev Environment for you. The devfile is specific to the repository for which it was created, and you can create a separate devfile for each repository. Your Dev Environment supports commands and events, and provides a default universal devfile image.

If you create a project using the empty blueprint, you can create a devfile manually. If you create a project using a different blueprint, CodeCatalyst creates a devfile automatically. The `/projects` directory of a Dev Environment stores the files that are pulled from the source repository and the devfile. The `/home` directory, which is empty when you first create a Dev Environment, stores the files you create while using your Dev Environment. Everything in the `/projects` and `/home` directories of a Dev Environment is stored persistently.

> **Note**
>
> The `/home` folder only changes if you change the name of the devfile or devfile component name. If you change the devfile or devfile component name, the contents of the `/home` directory are replaced and your previous `/home` directory data cannot be recovered.

If you create a Dev Environment with a source repository that does not contain a devfile in its root, or if you create a Dev Environment without a source repository, a default universal devfile is applied to the source repository automatically. The same default universal devfile image is used for all IDEs. CodeCatalyst currently supports devfile version 2.0.0. For more information about the devfile, see Devfile schema - Version 2.0.0.

> **Note**
>
> You can only include public container images in your devfile.

Note that VPC-connected Dev Environments only support the following devfile images:

- Universal image
- Private Amazon ECR images, if the repository is in the same region as the VPC

Topics

Editing a repository devfile for a Dev Environment

Use the following procedure to edit a repository devfile for a Dev Environment.

Editing a repository devfile for a Dev Environment in CodeCatalyst

To edit the repository devfile

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. Navigate to the project that contains the source repository for which you want to edit the devfile.
3. In the navigation pane, choose **Code**.
4. Choose **Source Repositories**.
5. Choose the source repository that contains the devfile that you want to edit.
6. From the list of files, choose the `devfile.yaml` file.
7. Choose **Edit**.
8. Edit the devfile.
9. Choose **Commit**, or create a pull request so a team member can review and approve the changes.

> **Note**
>
> If you edit your devfile, you have to restart the devfile for the changes to take effect. This can be done by running `/aws/mde/mde start --location devfile.yaml`. If there's a problem starting your devfile, it will enter recovery mode. However, if you edit a devfile associated to a VPC-connected Dev Environment, you have to restart the Dev Environment instead for the changes to take effect.

You can review which devfile is being used by running `/aws/mde/mde status`. The location field has the path of the devfile relative to the environment's `/projects` folder.

```
{
            "status": "STABLE",
            "location": "devfile.yaml"
        }
```

You can also move the default devfile in `/projects/devfile.yaml` to your source code repository. To update the location of the devfile, use following command: `/aws/mde/mde start --location` *`repository-name`*`/devfile.yaml`.

Editing a repository devfile for a Dev Environment in an IDE

To change the configuration of a Dev Environment, you must edit the devfile. We recommend that you edit the devfile in a supported IDE and then update your Dev Environment, but you can also

edit the devfile from the root of the source repository in CodeCatalyst. If you edit the devfile in a supported IDE, you must commit and push your changes to the source repository or create a pull request so a team member can review and approve the devfile edits.

- Editing the repository devfile for a Dev Environment in AWS Cloud9
- Editing the repository devfile for a Dev Environment in VS Code
- Editing the repository devfile for a Dev Environment in JetBrains

Devfile features supported by CodeCatalyst

CodeCatalyst supports the following devfile features on version 2.0.0. For more information about the devfile, see Devfile schema - Version 2.0.0.

Feature	Type
`exec`	Command
`postStart`	Event
`container`	Component
`args`	Component Properties
`env`	Component Properties
`mountSources`	Component Properties
`volumeMounts`	Component Properties

Example of a devfile for a Dev Environment

The following is an example of a simple devfile.

```
schemaVersion: 2.0.0
metadata:
  name: al2
components:
  - name: test
```

```
    container:
      image: public.ecr.aws/amazonlinux/amazonlinux:2
      mountSources: true
      command: ['sleep', 'infinity']
  - name: dockerstore
commands:
  - id: setupscript
    exec:
      component: test
      commandLine: "chmod +x script.sh"
      workingDir: /projects/devfiles
  - id: executescript
    exec:
      component: test
      commandLine: "/projects/devfiles/script.sh"
  - id: yumupdate
    exec:
      component: test
      commandLine: "yum -y update --security"
events:
  postStart:
    - setupscript
    - executescript
    - yumupdate
```

Devfile startup, command, and event logs are captured and stored in `/aws/mde/logs`. To debug devfile behaviour, start up your Dev Environment using a working devfile and access the logs.

Troubleshooting a repository devfile using recovery mode

If there's a problem starting your devfile, it will enter recovery mode so that you can still connect to your environment and fix your devfile. While in recovery mode, running `/aws/mde/mde status` won't contain the location of your devfile.

```
{
            "status": "STABLE"
        }
```

You can check the error in the logs under `/aws/mde/logs`, fix the devfile, and try running `/aws/mde/mde start` again.

Specifying universal devfile images for a Dev Environment

The default *universal image* includes the most commonly used programming languages and related tools that can be used for your IDE. If no image is specified, CodeCatalyst provides this image and contains tools that are maintained by CodeCatalyst. To remain notified of new image releases, see Subscribing to universal image notifications with SNS.

> **Note**
>
> The `public.ecr.aws/aws-mde/universal-image:latest` image gets the `public.ecr.aws/aws-mde/universal-image:3.0` image.

Amazon CodeCatalyst supports the following devfile images.

Image version	Image identifier
`Universal image 1.0`	`public.ecr.aws/aws-mde/universal-image:1.0`
`Universal image 2.0`	`public.ecr.aws/aws-mde/universal-image:2.0`
`Universal image 3.0`	`public.ecr.aws/aws-mde/universal-image:3.0`

> **Note**
>
> If you're using AWS Cloud9, auto-complete will not work for PHP, Ruby and CSS after upgrading to `universal-image:3.0`.

Topics

Subscribing to universal image notifications with SNS

CodeCatalyst provides a universal image notification service. You can use it to subscribe to an Amazon Simple Notification Service (SNS) topic that notifies you when CodeCatalyst universal image updates have been released. For more information about SNS topics, see What is Amazon Simple Notification Service?.

Whenever new universal images are released, we send notifications to subscribers; this section describes how to subscribe to CodeCatalyst universal image updates.

Sample message

```
{
    "Type": "Notification",
    "MessageId": "123456789",
    "TopicArn": "arn:aws:sns:us-east-1:1234657890:universal-image-updates",
    "Subject": "New Universal Image Release",
    "Message": {
        "v1": {
            "Message": "A new version of the Universal Image has been released. You are
 now able to launch new DevEnvironments using this image.",
            "image ": {
                "release_type": "MAJOR VERSION",
                "image_name": "universal-image",
                "image_version": "2.0",
                "image_uri": "public.ecr.aws/amazonlinux/universal-image:2.0"
            }
        }
    },
    "Timestamp": "2021-09-03T19:05:57.882Z",
    "UnsubscribeURL": "example url"
}
```

To subscribe to CodeCatalyst universal image updates using the Amazon SNS console

1. Open the Amazon SNS console to the Dashboard.
2. In the navigation bar, choose your AWS Region.
3. In the navigation pane, choose **Subscriptions**, and then choose **Create subscription**.
4. In **Topic ARN**, enter `arn:aws:sns:us-east-1:089793673375:universal-image-updates`.

5. In **Protocol**, choose **Email**.
6. In **Endpoint**, provide an email address. This email address will be used to receive notifications.
7. Choose **Create subscription**.
8. You will receive a confirmation email with the subject line "AWS Notification - Subscription Confirmation". Open the email and choose **Confirm subscription**.

To unsubscribe from CodeCatalyst universal image updates using the Amazon SNS console

1. Open the Amazon SNS console to the Dashboard.
2. In the navigation bar, choose your AWS Region.
3. In the navigation pane, choose **Subscriptions** and then select the subscription you want to unsubscribe from.
4. Choose **Actions**, and then choose **Delete subscriptions**.
5. Choose **Delete**.

Universal image 1.0 runtime versions

The following table lists the available runtimes for `universal-image:1.0`.

`universal-image:1.0` runtime versions

Runtime name	Version	Specific major and latest minor version
aws cli	2.11	`aws-cli: 2.x`
docker compose	2.16	`docker-compose: 2.x`
dotnet	6.0	`dotnet: 6.x`
	7.0	`dotnet: 7.x`
golang	1.19	`golang: 1.x`
java	corretto11	`java: corretto11.x`
	corretto17	`java: corretto17.x`
nodejs	14.20	`nodejs: 14.x`

Runtime name	Version	Specific major and latest minor version
	16.19	`nodejs: 16.x`
openssl	1.0	`openssl: 1.x`
	1.1	
php	7.2	`php: 7.x`
python	3.9	`python: 3.x`
	3.10	
ruby	3.1	`ruby: 3.x`
terraform	1.4	`terraform: 1.x`

Universal image 2.0 runtime versions

The following table lists the available runtimes for `universal-image:2.0`.

`universal-image:2.0` runtime versions

Runtime name	Version	Specific major and latest minor version
aws cli	2.11	`aws-cli: 2.x`
docker compose	2.17	`docker-compose: 2.x`
dotnet	6.0	`dotnet: 6.x`
	7.0	`dotnet: 7.x`
golang	1.20	`golang: 1.x`
java	corretto11	`java: corretto11.x`
	corretto17	`java: corretto17.x`
nodejs	16.19	`nodejs: 16.x`

Runtime name	Version	Specific major and latest minor version
openssl	1.0	`openssl: 1.x`
	1.1	
php	7.2	`php: 7.x`
python	3.9	`python: 3.x`
	3.10	
ruby	3.2	`ruby: 3.x`
terraform	1.4	`terraform: 1.x`

Universal image 3.0 runtime versions

The following table lists the available runtimes for `universal-image:3.0`.

`universal-image:3.0` runtime versions

Runtime name	Version	Specific major and latest minor version
aws cli	2.11	`aws-cli: 2.x`
docker compose	2.17	`docker-compose: 2.x`
dotnet	6.0	`dotnet: 6.x`
	7.0	`dotnet: 7.x`
golang	1.21	`golang: 1.x`
java	corretto11	`java: corretto11.x`
	corretto17	`java: corretto17.x`
nodejs	18.17	`nodejs: 18.x`
	20.6	`nodejs: 20.x`

Runtime name	Version	Specific major and latest minor version
openssl	3.0	`openssl: 3.x`
php	8.2	`php: 8.x`
python	3.9	`python: 3.x`
	3.11	
ruby	3.2	`ruby: 3.x`
terraform	1.5	`terraform: 1.x`

Devfile commands

Currently, CodeCatalyst only supports `exec` commands in your devfile. For more information, see Adding commands in the Devfile.io documentation.

The following example shows you how to specify `exec` commands in your devfile.

```
commands:
  - id: setupscript
    exec:
      component: test
      commandLine: "chmod +x script.sh"
      workingDir: /projects/devfiles
  - id: executescript
    exec:
      component: test
      commandLine: "./projects/devfiles/script.sh"
  - id: updateyum
    exec:
      component: test
      commandLine: "yum -y update --security"
```

After you're connected to your Dev Environment, you can execute defined commands through the terminal.

```
/aws/mde/mde command <command-id>
```

```
/aws/mde/mde command executescript
```

For long running commands, you can use the streaming flag `-s` to output the execution of the command in real time.

```
/aws/mde/mde -s command <command-id>
```

> **Note**
>
> `command-id` must be lower case.

Exec parameters supported by CodeCatalyst

CodeCatalyst supports the following `exec` parameters on devfile version 2.0.0.

- `commandLine`
- `component`
- `id`
- `workingDir`

Devfile events

Currently, CodeCatalyst only supports `postStart` events in your devfile. For more information, see postStartObject in the Devfile.io documentation.

The following example shows you how to add `postStart` event bindings in your devfile.

```
commands:
  - id: executescript
    exec:
      component: test
      commandLine: "./projects/devfiles/script.sh"
  - id: updateyum
    exec:
```

```
      component: test
      commandLine: "yum -y update --security"
events:
  postStart:
    - updateyum
    - executescript
```

After startup, your Dev Environment will execute the specified `postStart` commands in the order they are defined. If a command fails, the Dev Environment will continue running and the execution output is stored in the logs under `/aws/mde/logs`.

Devfile components

Currently, CodeCatalyst only supports `container` components in your devfile. For more information, see Adding components in the Devfile.io documentation.

The following example shows you how to add a startup command to your container in your devfile.

```
components:
  - name: test
    container:
      image: public.ecr.aws/amazonlinux/amazonlinux:2
      command: ['sleep', 'infinity']
```

> **Note**
>
> When the container has short lived entry command, you must include `command: ['sleep', 'infinity']` to keep the container running.

CodeCatalyst also supports the following properties in your container component: `args`, `env`, `mountSources`, and `volumeMounts`.

Associating a VPC connection to a Dev Environment

A *VPC connection* is a CodeCatalyst resource which contains all of the configurations needed for your workflow to access a VPC. Space administrators can add their own VPC connections in the Amazon CodeCatalyst console on behalf of space members. By adding a VPC connection, space members can run workflow actions and create Dev Environments that adhere to network rules and can access resources in the associated VPC.

You can only associate a Dev Environment to a VPC connection upon Dev Environment creation. You can't change the VPC connection associated to your Dev Environment after you create it. If you'd like to use a different VPC connection, you have to delete your current Dev Environment and create new one.

> **Important**
>
> Dev Environments with a VPC connection do not support third-party source repositories linked to CodeCatalyst.

Note that Dev Environments utilize several AWS resources and services upon creation. This means that Dev Environments connect to the following AWS services:

- Amazon CodeCatalyst
- AWS SSM
- AWS KMS
- Amazon ECR
- Amazon CloudWatch
- Amazon ECS

> **Note**
>
> AWS Toolkit doesn't support Dev Environments creation with an associated VPC connection. Also note that if you use an IDE other than AWS Cloud9, you may experience loading times of about five minutes.

You must have the **Space administrator** role or **Power user** role to manage VPC connections at the space level. For more information about VPCs, see Managing Amazon VPCs in CodeCatalyst in the *CodeCatalyst Administrator Guide*.

Quotas for Dev Environments in CodeCatalyst

The following table describes quotas and limits for Dev Environments in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Number of Dev Environment hours per month	Dev Environment hours are impacted by the overall storage limits for your space. For more information, see Pricing and Troubleshooting problems with Dev Environments.
Amount of Dev Environment storage per space	Dev Environment storage us impacted by the overall storage limits for your space. For more information, see Pricing and Troubleshooting problems with Dev Environments.
Amount of Dev Environment compute	Dev Environment compute is impacted by the overall storage limits for your space. For more information, see Pricing and Troubleshooting problems with Dev Environments.

Publish and share software packages in CodeCatalyst

Amazon CodeCatalyst contains a fully managed package repository service that makes it easy for your development team to securely store and share software packages used for application development. These packages are stored in package repositories, which are created and organized within projects in CodeCatalyst.

A single package repository can store packages of every supported package type. CodeCatalyst supports the following package formats:

- npm
- Maven
- NuGet
- Python

Packages in a package repository can be discovered and shared between members of the project that contains the repository.

To publish packages to, and consume packages from a repository, configure a package manager to use the repository endpoint (URL). You can then use the package manager to publish packages to the repository. You can use package managers such as Maven, Gradle, npm, yarn, nuget, dotnet, pip, and twine.

You can also configure CodeCatalyst workflows to use CodeCatalyst package repositories. For more information about using packages in workflows, see Publishing and importing packages using a workflow.

You can make packages in one package repository available to another repository in the same project by adding it as an upstream repository. All package versions available to the upstream repository are also available to the downstream repository. For more information, see Configuring and using upstream repositories.

You can make open-source packages available to your CodeCatalyst repository by creating a special type of repository called a **gateway**. Upstreaming to a gateway repository allows you to consume packages from popular public repositories such as npmjs.com and pypi.org, and automatically cache them in your CodeCatalyst repository. For more information, see Connecting to public external repositories.

Topics

Packages concepts

Here are some concepts and terms to know when managing, publishing, or consuming packages in CodeCatalyst.

Packages

A *package* is a bundle that includes both software and the metadata that is required to install the software and resolve any dependencies. CodeCatalyst supports the npm package format.

A package consists of:

- A name (for example, `webpack` is the name of a popular npm package)
- An optional namespace (for example, `@types` in `@types/node`)
- A set of versions (for example, `1.0.0`, `1.0.1`, `1.0.2`)
- Package-level metadata (for example, npm dist tags)

Package namespaces

Some package formats support hierarchical package names to organize packages into logical groups and to help avoid name collisions. Packages that have the same name can be stored in different namespaces. For example, npm supports scopes, and the npm package `@types/node`

has a scope of `@types` and a name of node. There are many other package names in the `@types` scope. In CodeCatalyst, the scope ("types") is referred to as the package namespace, and the name ("node") is referred to as the package name. For Maven packages, the package namespace corresponds to the Maven groupID. The Maven package `org.apache.logging.log4j:log4j` has a groupID (package namespace) of `org.apache.logging.log4j` and the artifactID (package name) `log4j`. Some package formats such as Python don't support hierarchical names with a concept similar to npm scope or Maven groupID. If you don't have a way to group package names, it can be more difficult to avoid name collisions.

Package versions

A *package version* identifies the specific version of a package, such as `@types/node@12.6.9`. The version number format and semantics vary for different package formats. For example, npm package versions must conform to the Semantic Versioning specification. In CodeCatalyst, a package version consists of the version identifier, package-version-level metadata, and a set of assets.

Assets

An *asset* is an individual file stored in CodeCatalyst that is associated with a package version, such as an npm `.tgz` file or a Maven POM or JAR file.

Package repositories

A CodeCatalyst *package repository* contains a set of packages, which contain package versions, each of which maps to a set of assets. Package repositories are polyglot, meaning a single repository can contain packages of any supported type. Each package repository exposes endpoints for fetching and publishing packages using tools like the NuGet CLIs (`nuget`, `dotnet`), the `npm` CLI, the Maven CLI (`mvn`), and the Python CLIs (`pip` and `twine`). For information about packages quotas in CodeCatalyst, including how many package repositories can be created in each space, see Quotas for packages.

You can link a package repository to another by setting it as an upstream. When a repository is set as an upstream, you can use any package from the upstream as well as any additional upstream repositories in the chain. For more information, see Upstream repositories.

Gateway repositories are a special type of package repository that pulls and stores packages from official external package authorities. For more information, see Gateway repositories.

Upstream repositories

You can use CodeCatalyst to create an upstream relationship between two package repositories. A package repository is an *upstream* of another when the package versions it contains can be accessed from the package repository endpoint of the downstream repository. With an upstream relationship, the contents of the two package repositories are effectively merged from the point of view of a client.

For example, if a package manager requests a package version that does not exist in a repository, CodeCatalyst will then search configured upstream repositories for the package version. Upstream repositories are searched in the order they are configured, and once a package is found, CodeCatalyst will stop searching.

Gateway repositories

A *gateway repository* is a special type of package repository that is connected to a supported external, official package authority. When you add a gateway repository as an upstream repository, you can consume packages from the corresponding official package authority. Your downstream repository does not communicate to the public repository, instead, everything is intermediated by the gateway repository. Packages consumed in this manner are stored in both the gateway repository and the downstream repository that received the original request.

Gateway repositories are predefined, but they must be created in each project to be used. The following list contains every gateway repository that can be created in CodeCatalyst and the package authority they are connected to.

- **npm-public-registry-gateway** provides npm packages from npmjs.com.
- **maven-central-gateway** provides Maven packages from the Maven Central repository.
- **google-android-gateway** provides Maven packages from Google Android.
- **commonsware-gateway** provides Maven packages from CommonsWare.
- **gradle-plugins-gateway** provides Maven packages from Gradle Plugins.
- **nuget-gallery-gateway** provides NuGet packages from the NuGet Gallery.
- **pypi-gateway** provides Python packages from the Python Package Index.

Configuring and using package repositories

In CodeCatalyst, packages are stored and managed inside package repositories. To publish packages to CodeCatalyst or to consume packages from a CodeCatalyst (or any supported public package repositories), you must create a package repository and connect your package manager to it.

Topics

Creating a package repository

Perform the following steps to create a package repository in CodeCatalyst.

To create a package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project in which you want to create a package repository.
3. From the navigation pane, choose **Packages**.
4. On the **Package repositories** page, choose **Create package repository**.
5. In the **Package repository details** section, add the following:

    a. **Repository name**. Consider using a descriptive name with details such as your project or team name, or how the repository will be used.

    b. (Optional) **Description**. A repository description is especially helpful when you have multiple repositories across multiple teams in a project.
6. In the **Upstream repositories** section, choose **Select upstream repositories** to add any package repositories that you want to access through your CodeCatalyst package repository. You can add **Gateway repositories** to connect to external package repositories or other **CodeCatalyst repositories**.

    - When a package is requested from a package repository, upstream repositories will be searched in the order they appear in this list. Once a package is found, CodeCatalyst will

stop searching. To change the order of the upstream repositories, you can drag and drop the repositories in the list.

7. Choose **Create** to create your package repository.

Connecting to a package repository

To publish to, or consume packages from CodeCatalyst, you must configure your package manager with your package repository endpoint information and CodeCatalyst credentials. If you haven't created a repository, you can do so by following the instructions in Creating a package repository.

For instructions on how to connect a package manager to a CodeCatalyst package repository, see the following documentation.

- Configuring and using Gradle Groovy
- Configuring and using mvn
- Configuring and using the nuget or dotnet CLI
- Configuring and using npm
- Configuring pip and installing Python packages
- Configuring Twine and publishing Python packages

Deleting a package repository

Perform the following steps to delete a package repository in CodeCatalyst.

To delete a package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project that contains the package repository that you want to delete.
3. From the navigation pane, choose **Packages**.
4. On the **Package repositories** page, choose the repository you want to delete.
5. Choose **Delete**.
6. Review the information provided about the effects of deleting a package repository.
7. Enter `delete` into the input field and choose **Delete**.

Configuring and using upstream repositories

You can connect both gateway repositories, and other CodeCatalyst package repositories, as upstreams to your package repositories. This enables a package manager client to access the packages that are contained in more than one package repository by using a single package repository endpoint. The following are the main benefits of using upstream repositories:

- You only have to configure your package manager with a single repository endpoint to pull from multiple sources.
- Packages consumed from an upstream repository are stored in your downstream repository, which ensures your packages are available even if the upstream repository experiences unexpected outages or packages in the upstream repository are deleted.

You can add upstream repositories when you create a package repository. You can also add or remove upstream repositories from existing package repositories in the CodeCatalyst console.

When you add a gateway repository as an upstream repository, the package repository is connected to the gateway repository's corresponding public package repository. For a list of supported public package repositories, see Supported external package repositories and their gateway repositories.

You can link multiple repositories together as upstream repositories. For example, suppose that your team creates a repository named `project-repo` and is already using another repository named `team-repo` that has the **npm-public-registry-gateway** added as an upstream repository, which is connected to the public npm repository, `npmjs.com`. You can add `team-repo` as an upstream repository to `project-repo`. In this case, you only have to configure your package manager to use `project-repo` to pull packages from `project-repo`, `team-repo`, `npm-public-registry-gateway`, and `npmjs.com`.

Topics

Adding an upstream repository

Adding a public package repository or another CodeCatalyst package repository as an upstream repository to your downstream repository makes all of the packages in the upstream repository available to package managers that are connected to the downstream repository.

To add an upstream repository

1. In the navigation pane, choose **Packages**.
2. On the **Package repositories** page, choose the package repository that you want to add an upstream repository to.
3. Under the package repository's name, choose **Upstreams** and choose **Select upstream repositories**.
4. In **Select upstream type**, choose one of the following:
   - **Gateway repositories**

     You can choose from a list of available gateway repositories.

     > **Note**
     >
     > To connect to public external package authorities such as Maven Central, npmjs.com, or Nuget Gallery, CodeCatalyst uses gateway repositories as intermediary repositories that search and store packages pulled from external repositories. This saves time and data transfer as all package repositories in a project will use packages from the gateway intermediary repository. For more information, see Connecting to public external repositories.

   - **CodeCatalyst repositories**

     You can choose from a list of available CodeCatalyst package repositories in your project.
5. When you've selected all of the repositories you want to add as upstream repositories, choose **Select**, and then choose **Save**.

   For more information about changing the search order of upstream repositories, see Editing the search order of upstream repositories.

When you've added an upstream repository, you can use a package manager that is connected to your local repository to fetch packages from the upstream repository. You do not need to update your package manager configuration. For more information about requesting package versions from an upstream repository, see Requesting a package version with upstream repositories.

Editing the search order of upstream repositories

CodeCatalyst searches upstream repositories in their configured search order. When a package is found, CodeCatalyst stops searching. You can change the order in which the upstream repositories are searched for packages.

To edit the search order of upstream repositories

1. In the navigation pane, choose **Packages**.
2. On the **Package repositories** page, choose the package repository whose upstream repository search order you want to edit.
3. Under the package repository's name, choose **Upstreams**.
4. In the **Upstream repositories** section, you can view the upstream repositories and their search order. To change the search order, drag and drop the repositories in the list.
5. When you're finished editing the search order of upstream repositories, choose **Save**.

Requesting a package version with upstream repositories

The following example shows the possible scenarios when a package manager requests a package from a CodeCatalyst package repository that has upstream repositories.

For this example, a package manager, such as `npm`, requests a package version from a package repository named `downstream` that has multiple upstream repositories. When the package is requested, the following can occur:

- If `downstream` contains the requested package version, it is returned to the client.
- If `downstream` does not contain the requested package version, CodeCatalyst searches for it in `downstream`'s upstream repositories, in their configured search order. If the package version is found, a reference to it is copied to `downstream`, and the package version is returned to the client.
- If neither `downstream` or its upstream repositories contain the package version, an HTTP 404 `Not Found` response is returned to the client.

The maximum number of direct upstream repositories allowed for one repository is 10. The maximum number of repositories CodeCatalyst searches in when a package version is requested is 25.

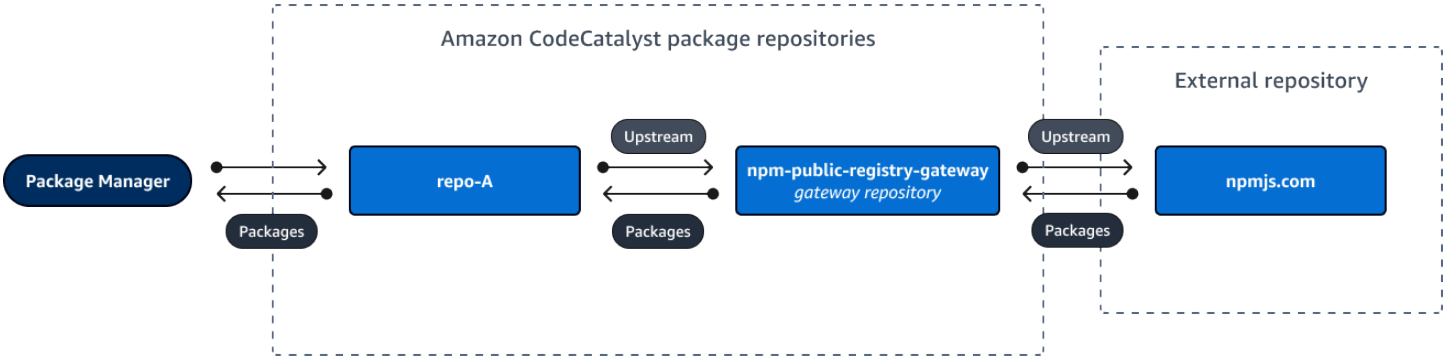
Package retention from upstream repositories

If a requested package version is found in an upstream repository, a reference to it is retained and is always available in the repository that requested it. This ensures that you have access to your packages if there is an unexpected outage of the upstream repository. The retained package version is not affected by any of the following:

- Deleting the upstream repository.
- Disconnecting the upstream repository from the downstream repository.
- Deleting the package version from the upstream repository.
- Editing the package version in the upstream repository (for example, by adding a new asset to it).

Fetching packages through an upstream relationship

CodeCatalyst can fetch packages through multiple linked repositories called upstream repositories. If a CodeCatalyst package repository has an upstream connection to another CodeCatalyst package repository that has an upstream connection to a gateway repository, requests for packages not in the upstream repository are copied from the external repository. For example, consider the following configuration: a repository named `repo-A` has an upstream connection to the gateway repository, `npm-public-registry-gateway`. `npm-public-registry-gateway` has an upstream connection to the public package repository, https://npmjs.com.



If `npm` is configured to use the `repo-A` repository, running `npm install` initiates the copying of packages from https://npmjs.com into `npm-public-registry-gateway`. The versions installed are also pulled into `repo-A`. The following example installs `lodash`.

```
$ npm config get registry
https://packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/
$ npm install lodash
+ lodash@4.17.20
added 1 package from 2 contributors in 6.933s
```

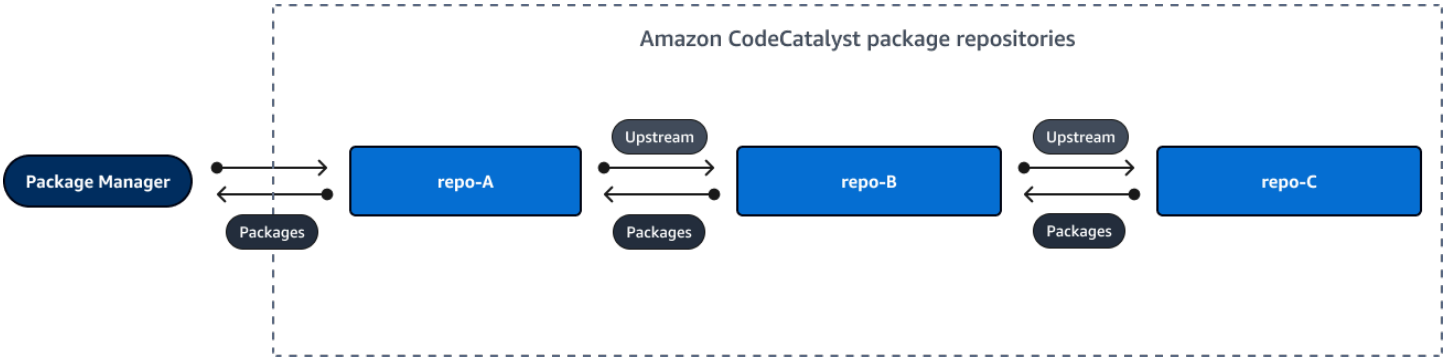
After running `npm install`, `repo-A` contains only the latest version (`lodash 4.17.20`) because that's the version that was fetched by `npm` from `repo-A`.

Because `npm-public-registry-gateway` has an external upstream connection to https://npmjs.com, all the package versions that are imported from https://npmjs.com are stored in `npm-public-registry-gateway`. These package versions could have been fetched by any downstream repository with an upstream connection that leads to `npm-public-registry-gateway`.

The contents of `npm-public-registry-gateway` provide a way for you to see all the packages and package versions imported from https://npmjs.com over time.

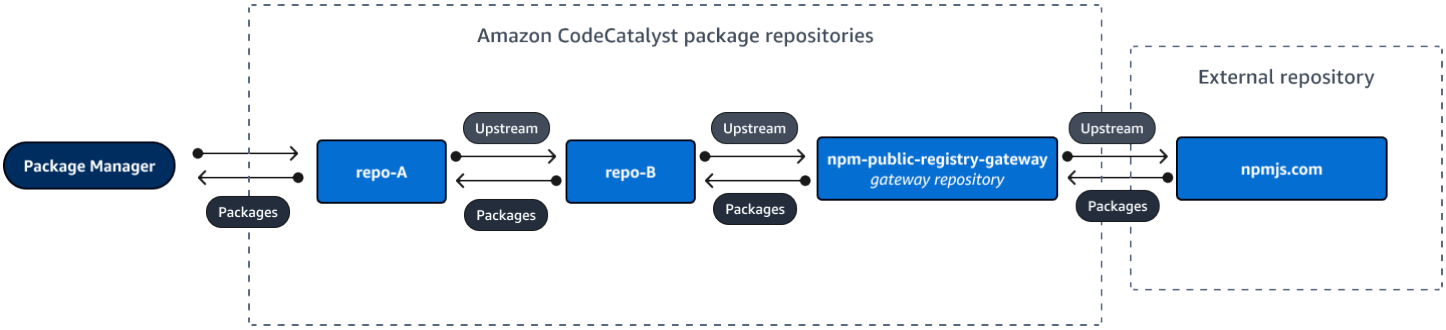
Package retention in intermediate repositories

CodeCatalyst allows you to chain upstream repositories. For example, `repo-A` can have `repo-B` as an upstream repository and `repo-B` can have `repo-C` as an upstream repository. This configuration makes the package versions in `repo-B` and `repo-C` available from `repo-A`.



When a package manager connects to repository `repo-A` and fetches a package version from repository `repo-C`, the package version is not retained in repository `repo-B`. The package version is only retained in the furthest downstream repository, which in this example is `repo-A`. It is not retained in any intermediate repositories. This is also true for longer chains; for example, if there were four repositories: `repo-A`, `repo-B`, `repo-C`, and `repo-D`, and a package manager connected to `repo-A` fetched a package version from `repo-D`, the package version would be retained in `repo-A` but not in `repo-B` or `repo-C`.

Package retention behavior is similar when pulling a package version from a public package repository, except that the package version is always retained in the gateway repository that has the direct upstream connection to the public repository. For example, `repo-A` has `repo-B` as an upstream repository. `repo-B` has `npm-public-registry-gateway` as an upstream repository, which has an upstream connection to the public repository, **npmjs.com**; see the diagram below.



If a package manager connected to `repo-A` requests a specific package version, *lodash 4.17.20* for example, and the package version is not present in any of the three repositories, it will be fetched from **npmjs.com**. When *lodash 4.17.20* is fetched, it is retained in `repo-A` as that is the furthest downstream repository and `npm-public-registry-gateway` as it has the upstream connection to the public external repository, **npmjs.com**. *lodash 4.17.20* is not retained in `repo-B` because that is an intermediate repository.

Removing an upstream repository

If you no longer want to access the packages within an upstream repository, you can remove the upstream repository from a package repository.

> **Warning**
>
> When you remove an upstream repository, you could break upstream relationship chains, which could break your projects or builds.

To remove an upstream repository

1. In the navigation pane, choose **Packages**.
2. On the **Package repositories** page, choose the package repository from which you want to remove an upstream repository.
3. Under the package repository's name, choose **Upstreams**.

4. In the **Edit upstream repositories** section, find the upstream repository you want to remove and choose

.
5. When you're finished removing upstream repositories, choose **Save**.

Connecting to public external repositories

You can connect CodeCatalyst package repositories to supported public, external repositories by adding the corresponding gateway repository as an upstream repository. Gateway repositories act as intermediary repositories that search and store packages pulled from external repositories. This saves time and data transfer because all package repositories in a project can use stored packages from the gateway repository.

To connect to a public repository using gateway repositories

1. In the navigation pane, choose **Packages**.
2. In **Packages**, choose the **Gateway repositories** page. You can view a list of supported gateway repositories and their descriptions.
3. To use a gateway repository, first you must create it. If the gateway repository has been created, the date and time it was created is shown. If it hasn't, choose **Create** to create it.
4. To use packages from the gateway repository, you must set an upstream connection to it from a CodeCatalyst repository. Choose **Package repositories** and choose the package repository that you want to connect to.
5. To connect to the public repository, choose **Upstreams** and choose **Select upstream repositories**.
6. Choose **Gateway repositories** select the gateway repository that corresponds to the public repository that you want to connect to as an upstream repository .
7. When you've selected all of the gateway repositories you want to add as upstream repositories, choose **Select**.
8. When you're finished ordering upstream repositories, choose **Save**.

For more information about upstream repositories, see Configuring and using upstream repositories.

When you've added a gateway repository as an upstream repository, you can use a package manager that is connected to your local repository to fetch packages from the public, external package repository that corresponds to it. You do not need to update your package manager configuration. Packages consumed in this manner are stored in both the gateway repository and your local package repository. For more information about requesting package versions from an upstream repository, see Requesting a package version with upstream repositories.

Supported external package repositories and their gateway repositories

CodeCatalyst supports adding an upstream connection to the following official package authorities with gateway repositories.

Repository package type	Description	Gateway repository name
npm	npm public registry	`npm-public-registry-gateway`
Python	Python Package Index	`pypi-gateway`
Maven	Maven Central	`maven-central-gateway`
Maven	Google Android repository	`google-android-gateway`
Maven	CommonsWare	`commonsware-gateway`
Maven	Gradle plugins repository	`gradle-plugins-gateway`
NuGet	NuGet Gallery	`nuget-gallery-gateway`

Publishing and modifying packages

A *package* in CodeCatalyst is a bundle of software and the metadata that is required to resolve dependencies and install the software. For a list of supported package formats in CodeCatalyst,

see Publish and share software packages in CodeCatalyst. This section provides information about publishing, viewing, deleting packages, and updating a package version's status.

Topics

Publishing packages to a CodeCatalyst package repository

You can publish versions of any supported package type to a CodeCatalyst package repository by using package manager tools. The steps to publish a package version are as follows:

To publish a package version to a CodeCatalyst package repository

1. If you haven't, create a package repository.
2. Connect your package manager to your package repository. For instructions on how to connect the npm package manager to a CodeCatalyst package repository, see Configuring and using npm.
3. Use your connected package manager to publish your package versions.

Contents

Publishing and upstream repositories

In CodeCatalyst, you cannot publish package versions that are present in reachable upstream repositories or public repositories. For example, suppose that you want to publish an npm package, `lodash@1.0`, to a package repository, `myrepo`, and `myrepo` is connected to npmjs.com through a gateway repository configured as an upstream repository. If `lodash@1.0` is present in the upstream repository or in npmjs.com, CodeCatalyst rejects any attempt to publish to it in `myrepo`

by issuing a 409 conflict error. This helps prevent you from accidentally publishing a package with the same name and version as a package in an upstream repository, which can result in unexpected behavior.

You can still publish different versions of a package name that exist in an upstream repository. For example, if `lodash@1.0` is present in an upstream repository, but `lodash@1.1` is not, you can publish `lodash@1.1` to the downstream repository.

Private packages and public repositories

CodeCatalyst does not publish packages stored in CodeCatalyst repositories to public repositories, such as npmjs.com or Maven Central. CodeCatalyst imports packages from public repositories into a CodeCatalyst repository, but it doesn't move packages in the opposite direction. Packages that you publish to CodeCatalyst repositories remain private and are only available to the CodeCatalyst project in which the repository belongs.

Overwriting package assets

You can't republish a package asset that already exists with different content. For example, suppose that you already published a Maven package with a JAR asset `mypackage-1.0.jar`. You can only publish that asset again if the checksum of the old and new assets are identical. To republish the same asset with new content, delete the package version first. Trying to republish the same asset name with different content will result in an HTTP 409 conflict error.

For package formats that support multiple assets (Python and Maven), you can add new assets with different names to an existing package version at any time, assuming you have the required permissions. Because npm and NuGet only support a single asset per package version, to modify a published package version you must first delete it.

If you try to republish an asset that already exists (for example, `mypackage-1.0.jar`), and the content of the published asset and the new asset are identical, the operation will succeed because the operation is idempotent.

Viewing package version details

You can use the CodeCatalyst console to view details about a specific package version.

To view package version details

1. In the navigation pane, choose **Packages**.

2. On the **Package repositories** page, choose the repository that contains the package version that you want to view the details of.
3. Search for the package version in the **Packages** table. You can use the search bar to filter packages by package name and format. Choose the package from the list.
4. In the **Package details** page, choose **Versions**, and then choose the version you want to view.

Deleting a package version

You can delete a package version from the **Package version details** page in the CodeCatalyst console.

To delete a package version

1. In the navigation pane, choose **Packages**.
2. On the **Package repositories** page, choose the repository that contains the package version that you want to delete.
3. Search and choose the package from the table.
4. On the **Package details** page, choose **Versions** and choose the version you want to delete.
5. On the **Package version details** page, choose **Version actions** and then choose **Delete**.
6. Enter *delete* into the text field and choose **Delete**.

Updating a package version's status

Every package version in CodeCatalyst has a status that describes the current state and availability of the package version. You can change the package version status in the CodeCatalyst console. For more information about the possible status values of package versions and their meanings, see Package version status.

To update a package version's status

1. In the navigation pane, choose **Packages**.
2. On the **Package repositories** page, choose the repository that contains the package version that you want to update the status of.
3. Search and choose the package from the table.

4. On the **Package details** page, choose **Versions** and then choose the version that you want to view.
5. On the **Package version details** page, choose **Actions** and then choose **Unlist**, **Archive**, or **Dispose**. For information about each package version status, see Package version status.
6. Enter the confirmation text into the text field, and then choose **Unlist**, **Archive**, or **Dispose**, depending on which status you are updating to.

Package version status

The following are possible values for package version status. You can change the package version status in the console. For more information, see Updating a package version's status.

- **Published**: The package version is successfully published and can be requested by a package manager. The package version will be included in package version lists returned to package managers; for example, in the output of `npm view <package-name> versions`. All assets of the package version are available from the repository.
- **Unfinished**: The last attempt to publish did not complete. Currently only Maven package versions can have a status of **Unfinished**. This can occur when the client uploads one or more assets for a package version but does not publish a `maven-metadata.xml` file for the package that includes that version.
- **Unlisted**: The package version assets are available for download from the repository, but the package version is not included in the list of versions returned to package managers. For example, for an npm package, the output of `npm view <package-name> versions` does not include the package version. This means that the npm dependency resolution logic does not select the package version because the version does not appear in the list of available versions. However, if the **Unlisted** package version is already referenced in an `npm package-lock.json` file, it can still be downloaded and installed; for example, when running `npm ci`.
- **Archived**: The package version assets cannot be downloaded. The package version will not be included in the list of versions returned to package managers. Because the assets are not available, consumption of the package version by clients is blocked. If your application build depends on a version that is updated to **Archived**, the build will fail, unless the package version has been locally cached. You can't use a package manager or build tool to republish an **Archived** package version because it is still present in the repository. However, you can change the package version status back to **Unlisted** or **Published** in the console.
- **Disposed**: The package version doesn't appear in listings, and the assets cannot be downloaded from the repository. The key difference between **Disposed** and **Archived** is that with a status of

Disposed, the assets of the package version are permanently deleted by CodeCatalyst. For this reason, you cannot move a package version from **Disposed** to **Archived**, **Unlisted**, or **Published**. The package version cannot be used because the assets have been deleted. When a package version has been marked as **Disposed**, you are not billed for storage of the package assets.

In addition to the statuses in the preceding list, a package version can also be deleted. After it is deleted, a package version is not in the repository and you can freely republish that package version by using a package manager or build tool.

Package name, package version, and asset name normalization

CodeCatalyst normalizes package names, package versions, and asset names before storing them, which means the names or versions in CodeCatalyst may be different than the name or version provided when the package was published. For more information about how names and versions are normalized in CodeCatalyst for each package type, see the following documentation.

- Python package name normalization
- NuGet package name, version, and asset name normalization

CodeCatalyst does not perform normalization on other package formats.

Editing package origin controls

In Amazon CodeCatalyst, package versions can be added to a package repository by directly publishing them, pulling them down from an upstream repository, or ingesting them from an external, public repository through a gateway. If you allow versions of a package to be added both by direct publishing and ingesting from public repositories, then you are vulnerable to a dependency substitution attack. For more information, see Dependency substitution attacks. To protect yourself against a dependency substitution attack, configure package origin controls on a package in a repository to limit how versions of that package can be added to the repository.

You should consider configuring package origin controls to make new versions of different packages come from both internal sources, such as direct publishing, and external sources, such as public repositories. By default, package origin controls are configured based on how the first version of a package is added to the repository.

Package origin control settings

With package origin controls, you can configure how package versions can be added to a repository. The following lists include the available package origin control settings and values.

Publish

This setting configures whether package versions can be published directly to the repository using package managers or similar tools.

- **ALLOW**: Package versions can be published directly.
- **BLOCK**: Package versions cannot be published directly.

Upstream

This setting configures whether package versions can be ingested from external, public repositories, or retained from upstream repositories when requested by a package manager.

- **ALLOW**: Any package version can be retained from other CodeCatalyst repositories configured as upstream repositories or ingested from a public source with an external connection.
- **BLOCK**: Package versions cannot be retained from other CodeCatalyst repositories configured as upstream repositories or ingested from a public source with an external connection.

Default package origin control settings

The default package origin controls for a package will be based on how the first version of that package is added to the package repository.

- If the first package version is published direcly by a package manager, the settings will be **Publish: ALLOW** and **Upstream: BLOCK**.
- If the first package version is ingested from a public source, the settings will be **Publish: BLOCK** and **Upstream: ALLOW**.

Common package access control scenarios

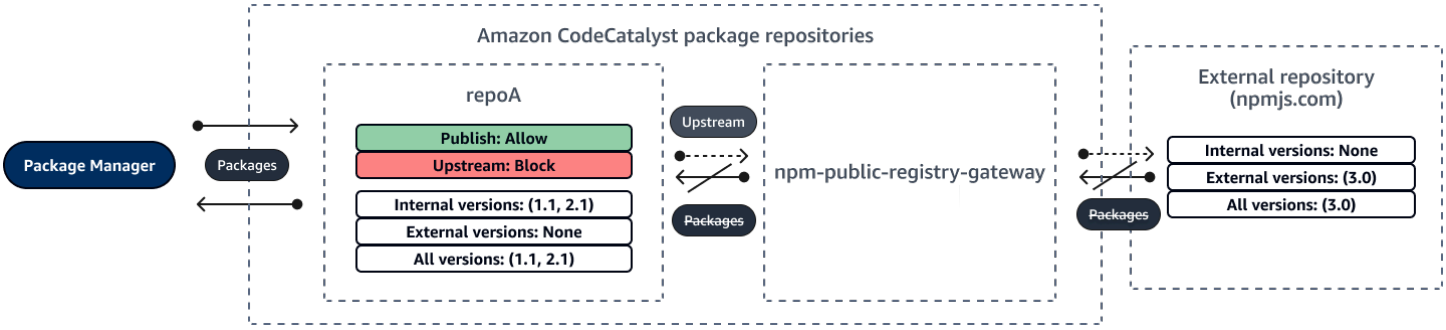
This section describes some common scenarios of when a package version is added to a CodeCatalyst package repository. Package origin control settings are set for new packages depending on how the first package version is added.

In the following scenarios, an *internal package* is published directly from a package manager to your repository, such as a package that you maintain. An *external package* is a package that exists in a public repository that can be ingested into your repository through a gateway repository upstream.

An external package version is published for an existing internal package

In this scenario, consider an internal package, *packageA*. Your team publishes the first package version for *packageA* to a CodeCatalyst package repository. Because this is the first package version for that package, the package origin control settings are automatically set to **Publish: Allow** and **Upstream: Block**. After the package is published in your repository, a package with the same name is published to a public repository that is connected to your CodeCatalyst package repository. This could be an attempted dependency substitution attack against the internal package, or it could be a coincidence. Regardless, package origin controls are configured to block the ingestion of the new external version to protect themselves against a potential attack.

In the following image, *repoA* is your CodeCatalyst package repository with an upstream connection to the `npm-public-registry-gateway` repository. Your repository contains versions 1.1 and 2.1 of *packageA*, but version 3.0 is published to the public repository. Normally, *repoA* would ingest version 3.0 after the package is requested by a package manager. Because package ingestion is set to **Block**, version 3.0 is not ingested into your CodeCatalyst package repository and is not available to package managers connected to it.

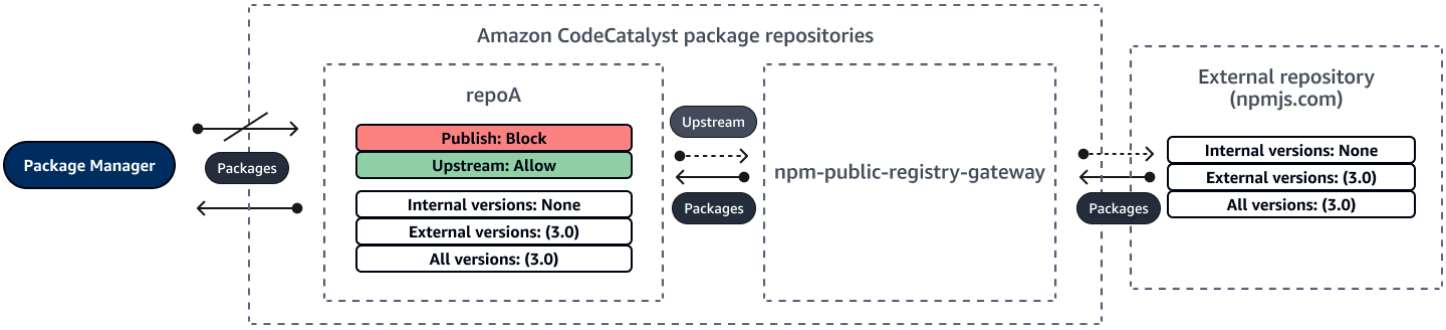


An internal package version is published for an existing external package

In this scenario, a package, *packageB*, exists externally in a public repository that you have connected to your repository. When a package manager connected to your repository requests *packageB*, the package version is ingested into your repository from the public repository. Because this is the first package version of *packageB* added to your repository, the package origin settings are configured to **Publish: BLOCK** and **Upstream: ALLOW**. Later, you try to publish a version with the same package name to the repository. You may not be aware of the public package and trying

to publish an unrelated package by same name, or you may be trying to publish a patched version, or you may be trying to directly publish the exact package version that already exists externally. CodeCatalyst rejects the version that you are trying to publish, but you can explicitly override the rejection and publish the version, if necessary.

In the following image, *repoA* is your CodeCatalyst package repository with an upstream connection to the `npm-public-registry-gateway` repository. Your package repository contains version 3.0 that it ingested from the public repository. You want to publish version 1.2 to your package repository. Typically, you could publish version 1.2 to *repoA*, but because publishing is set to **Block**, version 1.2 cannot be published.



Publishing a patched package version of an existing external package

In this scenario, a package, *packageB*, exists externally in a public repository that you have connected to your package repository. When a package manager connected to your repository requests *packageB*, the package version is ingested into your repository from the public repository. Because this is the first package version of *packageB* added to your repository, the package origin settings are configured to **Publish: BLOCK** and **Upstream: ALLOW**. Your team decides to publish patched package versions of this package to the repository. To be able to publish package versions directly, your team changes the package origin control settings to **Publish: ALLOW** and **Upstream: BLOCK**. Versions of this package can now be published directly to your repository and ingested from public repositories. After your team publishes the patched package versions, your team reverts the package origin settings to **Publish: BLOCK** and **Upstream: ALLOW**.

Editing package origin controls

Package origin controls are configured automatically based on how the first package version of a package is added to the package repository. For more information, see Default package origin control settings. To add or edit package origin controls for a package in a CodeCatalyst package repository, perform the steps in the following procedure.

To add or edit package origin controls

1. In the navigation pane, choose **Packages**.
2. Choose the package repository that contains the package that you want to edit.
3. In the **Packages** table, search for and choose the package that you want to edit.
4. From the package summary page, choose **Origin controls**.
5. In **Origin controls**, choose the package origin controls that you want to set for this package. Both package origin control settings, **Publish** and **Upstream**, must be set at the same time.
   - To allow publishing package versions directly, in **Publish**, choose **Allow**. To block publishing of package versions, choose **Block**.
   - To allow ingestion of packages from external repositories and pulling packages from upstream repositories, in **Upstream sources**, choose **Allow**. To block all ingestion and pulling of package versions from external and upstream repositories, choose **Block**.
6. Choose **Save**.

Publishing and upstream repositories

In CodeCatalyst, you cannot publish package versions that are present in reachable upstream repositories or public repositories. For example, suppose that you want to publish an npm package `lodash@1.0` to a repository, `myrepo`, and `myrepo` has an upstream repository with an external connection to npmjs.com. Consider the following scenarios.

1. The package origin control settings on `lodash` are **Publish: ALLOW** and **Upstream: ALLOW**. If `lodash@1.0` is present in the upstream repository or in npmjs.com, CodeCatalyst rejects any attempt to publish to it in `myrepo` by issuing a 409 conflict error. You could still publish a different version, such as `lodash@1.1`.
2. The package origin control settings on `lodash` are **Publish: ALLOW** and **Upstream: BLOCK**. You can publish any version of `lodash` to your repository that does not already exist because package versions are not reachable.
3. The package origin control settings on `lodash` are **Publish: BLOCK** and **Upstream: ALLOW**. You cannot publish any package versions directly to your repository.

Dependency substitution attacks

Package managers simplify the process of packaging and sharing reusable code. These packages may be private packages developed by an organization for use in their applications, or they may be public, typically open-source packages that are developed outside an organization and distributed by public package repositories. When requesting packages, developers rely on their package manager to fetch new versions of their dependencies. Dependency substitution attacks, also known as dependency confusion attacks, exploit the fact that a package manager typically has no way to distinguish legitimate versions of a package from malicious versions.

Dependency substitution attacks belong to a subset of attacks known as software supply chain attacks. A software supply chain attack is an attack that takes advantage of vulnerabilities anywhere in the software supply chain.

A dependency substitution attack can target anyone who uses both internally developed packages and packages fetched from public repositories. The attackers identify internal package names and then strategically place malicious code with the same name in public package repositories. Typically, the malicious code is published in a package with a high version number. Package managers fetch the malicious code from these public feeds because they believe that the malicious packages are the latest versions of the package. This causes a "confusion" or "substitution" between the desired package and the malicious package, which leads to the code being compromised.

To prevent dependency substitution attacks, Amazon CodeCatalyst provides package origin controls. Package origin controls are settings that control how packages can be added to your repositories. The controls are configured automatically when the first package version of a new package is added to a CodeCatalyst repository The controls can ensure package versions cannot be both published directly to your repository and ingested from public sources, protecting you from dependency substitution attacks. For more information about package origin controls and how to change them, see Editing package origin controls.

Using npm

These topics describe how you can use `npm`, the Node.js package manager, with CodeCatalyst.

> **Note**
>
> CodeCatalyst supports `node v4.9.1` and later and `npm v5.0.0` and later.

Topics

Configuring and using npm

To use npm with CodeCatalyst, you must connect npm to your package repository and provide a personal access token (PAT) for authentication. You can view instructions for connecting npm to your package repository in the CodeCatalyst console.

Contents

Configuring npm with CodeCatalyst

The following instructions explain how to authenticate and connect npm to your CodeCatalyst package repository. For more information about npm, see the official npm documentation.

To connect npm to your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project.
3. In the navigation pane, choose **Packages**.
4. Choose your package repository from the list.
5. Choose **Connect to repository**.
6. In **Configuration details**, in **Package manager client**, choose **npm client**.
7. Choose your operating system to view the corresponding configuration steps.

8. A personal access token (PAT) is required to authenticate npm with CodeCatalyst. If you already have a token, you can use it. If not, you can create one using the following steps.

   a. **(Optional):** Update the **PAT name** and **Expiration date**.

   b. Choose **Create token**.

   c. Copy and store your PAT in a safe location.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box. Credentials should be short lived to minimize the length of time an attacker can use the credentials after misappropriating them.

9. Run the following commands from your project's root directory to configure npm with your package repository. The commands will do the following:

   - Create a project-level `.npmrc` file if your project does not have one.
   - Add the package repository endpoint information to your project-level `.npmrc` file.
   - Add your credentials (PAT) to your user-level `.npmrc` file.

   Replace the following values.

   > **Note**
   >
   > If you are copying from the console instructions, the values in the following commands are updated for you and do not need to be changed.

   - Replace *`username`* with your CodeCatalyst user name.
   - Replace *`PAT`* with your CodeCatalyst PAT.
   - Replace *`space_name`* with your CodeCatalyst space name.
   - Replace *`proj_name`* with your CodeCatalyst project name.
   - Replace *`repo_name`* with your CodeCatalyst package repository name.

```
npm set registry=https://packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/ --location project
npm set //packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/:_authToken=username:PAT
```

For npm 6 or lower: To make npm always pass the auth token to CodeCatalyst, even for `GET` requests, set the always-auth configuration variable with `npm config set` as follows.

```
npm set //packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/:always-auth=true --location project
```

Installing npm packages from a CodeCatalyst package repository

After you connect npm to your repository by following the steps in Configuring npm with CodeCatalyst, you can run `npm` commands on your repository.

You can install an npm package that is in your CodeCatalyst package repository or one of its upstream repositories with the `npm install` command.

```
npm install lodash
```

Installing npm packages from npmjs through CodeCatalyst

You can install npm packages from npmjs.com through a CodeCatalyst repository by configuring the repository with an upstream connection to the gateway repository connected to npmjs.com, **npm-public-registry-gateway**. Packages installed from npmjs are ingested and stored in the gateway repository, and the farthest downstream package repository.

To install packages from npmjs

1. If you haven't already done so, configure `npm` with your CodeCatalyst package repository by following the steps in Configuring npm with CodeCatalyst.
2. Check that your repository has added the gateway repository, **npm-public-registry-gateway**, as an upstream connection. You can check which upstream sources are added or add **npm-public-registry-gateway** as an upstream source by following the instructions in Adding an upstream repository and choosing the **npm-public-registry-gateway** repository.

3. Install packages with the `npm install` command.

```
npm install package_name
```

For more information about requesting packages from upstream repositories, see Requesting a package version with upstream repositories.

Publishing npm packages to your CodeCatalyst package repository

After you have completed Configuring npm with CodeCatalyst, you can run `npm` commands.

You can publish an npm package to a CodeCatalyst package repository with the `npm publish` command.

```
npm publish
```

For information about how to create npm packages, see Creating Node.js Modules on *npm Docs*.

npm command support

The following sections summarize the `npm` commands that are supported by CodeCatalyst package repositories, in addition to listing specific commands that are not supported.

Topics

- Supported commands that interact with a package repository
- Supported client-side commands
- Unsupported commands

Supported commands that interact with a package repository

This section lists `npm` commands where the `npm` client makes one or more requests to the registry to which it is configured (for example, `npm config set registry`). These commands have been verified to function correctly when invoked against a CodeCatalyst package repository.

Command	Description
bugs	Guesses the location of a package's bug tracker URL, and then it attempts to open it.

Command	Description
ci	Installs a project with a clean slate.
deprecate	Deprecates a version of a package.
dist-tag	Modifies package distribution tags.
docs	Guesses the location of a package's documentation URL, and then it attempts to open it by using the `--browser` config parameter.
doctor	Runs a set of checks to validate that your npm installation can manage your JavaScript packages.
install	Installs a package.
install-ci-test	Installs a project with a clean slate and runs tests. Alias: `npm cit`. This command runs an `npm ci`, followed immediately by an `npm test`.
install-test	Installs package and runs tests. Runs an `npm install`, followed immediately by an `npm test`.
outdated	Checks the configured registry to determine if any installed packages are outdated.
ping	Pings the configured or given npm registry and verifies authentication.
publish	Publishes a package version to the registry.
update	Guesses the location of a package's repository URL, and then it attempts to open it by using the `--browser` config parameter.

Command	Description
view	Displays package metadata. Can also be used to print metadata properties.

Supported client-side commands

These commands don't require any direct interaction with a package repository, so CodeCatalyst does not require anything to support them.

Command	Description
bin (legacy)	Displays the npm `bin` directory.
build	Builds a package.
cache	Manipulates the packages cache.
completion	Enables tab completion in all npm commands.
config	Updates the contents of the user and global `npmrc` files.
dedupe	Searches the local package tree and attempts to simplify the structure by moving dependencies further up the tree where they can be more effectively shared by multiple dependent packages.
edit	Edits an installed package. Selects a dependency in the current working directory and opens the package directory in the default editor.
explore	Browses an installed package. Spawns a subshell in the directory of the specified installed package. If a command is specified

Command	Description
	, then it is run in the subshell, which then immediately shuts down.
help	Gets help on npm.
help-search	Searches npm help documentation.
init	Creates a `package.json` file.
link	Symlinks a package directory.
ls	Lists installed packages.
pack	Creates a tarball from a package.
prefix	Displays a prefix. This is the closest parent directory to contain a `package.json` file, unless `-g` is also specified.
prune	Removes packages that are not listed on the parent package's dependencies list.
rebuild	Runs the `npm build` command on the matched folders.
restart	Runs a package's stop, restart, and start scripts and associated pre-scripts and post-scripts.
root	Prints the effective `node_modules` directory to standard out.
run-script	Runs arbitrary package scripts.
shrinkwrap	Locks down dependency versions for publicati on.
uninstall	Uninstalls a package.

Unsupported commands

These npm commands are not supported by CodeCatalyst package repositories.

Command	Description	Notes
access	Sets the access level on published packages.	CodeCatalyst uses a permissio n model that is different from the public npmjs repository.
adduser	Adds a registry user account	CodeCatalyst uses a user model that is different from the public npmjs repository.
audit	Runs a security audit.	CodeCatalyst does not currently vend security vulnerability data.
hook	Manages npm hooks, including adding, removing, listing, and updating.	CodeCatalyst does not currently support any change notification mechanism.
login	Authenticates a user. This is an alias for `npm adduser`.	CodeCatalyst uses an authentication model that is different from the public npmjs repository. For information, see Configuring npm with CodeCatalyst.
logout	Signs out of the registry.	CodeCatalyst uses an authentication model that is different from the public npmjs repository. There is no way to sign out from a CodeCatalyst repository, but authentication tokens expire after their configurable

Command	Description	Notes
		expiration time. The default token duration is 12 hours.
owner	Manages package owners.	CodeCatalyst uses a permissio ns model that is different from the public npmjs repository.
profile	Changes settings on your registry profile.	CodeCatalyst uses a user model that is different from the public npmjs repository.
search	Searches the registry for packages matching the search terms.	CodeCatalyst does not support the `search` command.
star	Marks your favorite packages.	CodeCatalyst currently does not support any favorites mechanism.
stars	Views packages marked as favorites.	CodeCatalyst currently does not support any favorites mechanism.
team	Manages teams and team memberships.	CodeCatalyst uses a user and group membership model that is different from the public npmjs repository.
token	Manages your authentication tokens.	CodeCatalyst uses a different model for getting authentic ation tokens. For informati on, see Configuring npm with CodeCatalyst.

Command	Description	Notes
unpublish	Removes a package from the registry.	CodeCatalyst does not support removing a package version from a repository by using the npm client. You can delete a package in the console.
whoami	Displays the npm user name.	CodeCatalyst uses a user model that is different from the public npmjs repository.

npm tag handling

npm registries support *tags*, which are string aliases for package versions. You can use tags to provide an alias instead of using version numbers. For example, you have a project with multiple streams of development and you use a different tag for each stream (for example, `stable`, `beta`, `dev`, `canary`). For more information, see dist-tag on *npm Docs*.

By default, npm uses the `latest` tag to identify the current version of a package. `npm install pkg` (without `@version` or `@tag` specifier) installs the latest tag. Typically, projects only use the latest tag for stable release versions. Other tags are used for unstable or prerelease versions.

Editing tags with the npm client

The three `npm dist-tag` commands (`add`, `rm`, and `ls`) function the same way in CodeCatalyst package repositories as they function in the default npm registry.

npm tags and upstream repositories

When `npm` requests the tags for a package and versions of that package are also present in an upstream repository, CodeCatalyst merges the tags before returning them to the client. For example, a repository named `R` has an upstream repository named `U`. The following table shows the tags for a package named `web-helper` that's present in both repositories.

Repository	Package name	Package tags
R	`web-helper`	*latest* (alias for version 1.0.0)
U	`web-helper`	*alpha* (alias for version 1.0.1)

In this case, when the npm client fetches the tags for the `web-helper` package from repository R, it receives both the *latest* and *alpha* tags. The versions the tags point to won't change.

When the same tag is present on the same package in both the upstream and local repository, CodeCatalyst uses the tag that was *last updated*. For example, suppose that the tags on *webhelper* have been modified to look like the following.

Repository	Package name	Package tags	Last updated
R	`web-helper`	*latest* (alias for version 1.0.0)	January 1, 2023
U	`web-helper`	*latest* (alias for version 1.0.1)	June 1, 2023

In this case, when the npm client fetches the tags for package *web-helper* from repository R, the *latest* tag will alias the version *1.0.1* because it was updated last. This makes it easy to consume new package versions in an upstream repository that are not yet present in a local repository by running `npm update`.

Using Maven

The Maven repository format is used by many different languages, including Java, Kotlin, Scala, and Clojure. It's supported by many different build tools, including Maven, Gradle, Scala SBT, Apache Ivy, and Leiningen.

We have tested and confirmed compatibility with CodeCatalyst for the following versions:

- Latest **Maven** version: 3.6.3.
- Latest **Gradle** version: 6.4.1. Version 5.5.1 has also been tested.

Topics

Configuring and using Gradle Groovy

To use Gradle Groovy with CodeCatalyst, you must connect Gradle Groovy to your package repository and provide a personal access token (PAT) for authentication. You can view instructions for connecting Gradle Groovy to your package repository in the CodeCatalyst console.

Contents

Fetching dependencies from CodeCatalyst

The following instructions explain how to configure Gradle Groovy to fetch dependencies your CodeCatalyst package repository.

To use Gradle Groovy to fetch dependencies from your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project.
3. In the navigation pane, choose **Packages**.
4. Choose your package repository from the list of package repositories.
5. Choose **Connect to repository**.

6. In the **Connect to repository** dialog box, choose **Gradle Groovy** from the list of package manager clients.
7. You will need a personal access token (PAT) to authenticate Gradle Groovy with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.

   a. Choose **Create token**.

   b. Choose **Copy** to copy your PAT.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box.

8. Update your gradle properties file with your access credentials. Replace *username* with your CodeCatalyst username and replace *PAT* with your CodeCatalyst personal access token. You can use any value for *spaceUsername* and *spacePassword* as long as you use the same values in the following steps.

```
spaceUsername=username
spacePassword=PAT
```

9. To fetch dependencies from CodeCatalyst in a Gradle build, copy the `maven` code snippet and add it to the `repositories` section in your project's `build.gradle` file. Replace the following values. You can use any value for *spaceName* as long as you use the same values in the following steps.

   > **Note**
   >
   > If copying from the console instructions, the following values should be updated for you and should not be changed.

   - Replace *space_name* with your CodeCatalyst space name.
   - Replace *proj_name* with your CodeCatalyst project name.
   - Replace *repo_name* with your CodeCatalyst package repository name.

```
maven {
  name = 'spaceName'
```

```
  url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
  credentials(PasswordCredentials)
}
```

10. (Optional) To use the CodeCatalyst package repository as the only source for your project dependencies, remove any other sections in repositories from the `build.gradle` file. If you have more than one repository, Gradle searches each repository for dependencies in the order they are listed.

Fetching plugins from CodeCatalyst

By default Gradle will resolve plugins from the public Gradle Plugin Portal. The following steps configure your Gradle project to resolve plugins from your CodeCatalyst package repository.

To use Gradle to fetch plugins from your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project.
3. In the navigation pane, choose **Packages**.
4. Choose your package repository from the list of package repositories.
5. Choose **Connect to repository**.
6. In the **Connect to repository** dialog box, choose **Gradle** from the list of package manager clients.
7. You will need a personal access token (PAT) to authenticate Gradle with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.
   a. Choose **Create token**.
   b. Choose **Copy** to copy your PAT.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box.

8. Update your gradle properties file with your access credentials. Replace *username* with your CodeCatalyst username and replace *PAT* with your CodeCatalyst personal access token. You

can use any value for *spaceUsername* and *spacePassword* as long as you use the same values in the following steps.

```
spaceUsername=username
spacePassword=PAT
```

9. Add a `pluginManagement` block to your `settings.gradle` file. The `pluginManagement` block must appear before any other statements in `settings.gradle`. Replace the following values.

> **Note**
>
> If copying from the console instructions, the following values should be updated for you and should not be changed.

- Replace *spaceName* with the name value used in the previous step.
- Replace *space_name* with your CodeCatalyst space name.
- Replace *proj_name* with your CodeCatalyst project name.
- Replace *repo_name* with your CodeCatalyst package repository name.

```
pluginManagement {
    repositories {
        maven {
            name = 'spaceName'
            url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
            credentials(PasswordCredentials)
        }
    }
}
```

This will ensure that Gradle resolves plugins from the specified repository. The repository must have an upstream connection configured to the Gradle Plugin Portal (`gradle-plugins-store`) so that commonly required Gradle plugins are available to the build. For more information, see the Gradle documentation.

Fetching packages from external package repositories through CodeCatalyst

You can install Maven packages from public repositories through a CodeCatalyst repository by configuring it with an upstream connection to the gateway that represents the gateway repository. Packages installed from the gateway repository are ingested and stored in your CodeCatalyst repository.

CodeCatalyst supports the following public Maven package repositories.

- maven-central-gateway
- google-android-gateway
- gradle-plugins-gateway
- commonsware-gateway

To install packages from public Maven package repositories

1. If you haven't already, configure Gradle with your CodeCatalyst package repository by following the steps in Fetching dependencies from CodeCatalyst or Fetching plugins from CodeCatalyst.
2. Ensure that your repository has added the gateway repository you want to install from as an upstream connection. You can do this by following the instructions in Adding an upstream repository and choosing the public package repository you want to add as an upstream.

For more information about requesting packages from upstream repositories, see Requesting a package version with upstream repositories.

Publishing packages to CodeCatalyst

This section describes how to publish a Java library built with Gradle Groovy to a CodeCatalyst repository.

To use Gradle Groovy to publish packages to a CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the overview page for your project, choose **Packages**.
3. Choose your package repository from the list of package repositories.
4. Choose **Connect to repository**.

5. In the **Connect to repository** dialog box, choose **Gradle Groovy** from the list of package manager clients.
6. You will need a personal access token (PAT) to authenticate Gradle with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.

   a. Choose **Create token**.

   b. Choose **Copy** to copy your PAT.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box.

7. Update your gradle properties file with your access credentials. Replace *username* with your CodeCatalyst username and replace *PAT* with your CodeCatalyst personal access token. You can use any value for *spaceUsername* and *spacePassword* as long as you use the same values in the following steps.

```
spaceUsername=username
spacePassword=PAT
```

8. Add the `maven-publish` plugin to the `plugins` section of the project's `build.gradle` file.

```
plugins {
    id 'java-library'
    id 'maven-publish'
}
```

9. Next, add a `publishing` section to the project `build.gradle` file. Replace the following values.

   > **Note**
   >
   > If copying from the console instructions, the following values should be updated for you and should not be changed.

   - Replace *space_name* with your CodeCatalyst space name.
   - Replace *proj_name* with your CodeCatalyst project name.

- Replace *repo_name* with your CodeCatalyst package repository name.

```
publishing {
    publications {
        mavenJava(MavenPublication) {
            groupId = 'group-id'
            artifactId = 'artifact-id'
            version = 'version'
            from components.java
        }
    }
    repositories {
        maven {
            name = 'spaceName'
            url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
            credentials(PasswordCredentials)
        }
    }
}
```

The `maven-publish` plugin generates a POM file based on the `groupId`, `artifactId`, and `version` specified in the `publishing` section.

10. After these changes to `build.gradle` are complete, run the following command to build the project and upload it to the repository.

```
./gradlew publish
```

11. Navigate to your package repository in the CodeCatalyst console to check that the package was successfully published. You should see the package in the **Packages** list of your package repository.

For more information, see these topics on the Gradle website:

- Building Java Libraries
- Publishing a project as a module

Running a Gradle build in IntelliJ IDEA

You can run a Gradle build in IntelliJ IDEA that pulls dependencies from CodeCatalyst. To authenticate Gradle with CodeCatalyst, you must use a personal access token (PAT). You can store your CodeCatalyst PAT in `gradle.properties` or a separate file of your choice.

Method 1: Put the PAT in `gradle.properties`

Use this method if you are not using the `gradle.properties` file and can overwrite its contents with your PAT. If you are using `gradle.properties`, you can modify this method to add the PAT instead of overwriting the file's contents.

> **Note**
>
> The example shows the `gradle.properties` file located in `GRADLE_USER_HOME`.

First, create a PAT if you do not have one.

To create a personal access token (PAT)

1. In the top menu bar, choose your profile badge, and then choose **My settings**.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

2. In **PAT name**, enter a descriptive name for your PAT.
3. In **Expiration date**, leave the default date or choose the calendar icon to select a custom date. The expiration date defaults to one year from the current date.
4. Choose **Create**.

   You can also create this token when you choose **Clone repository** for a source repository.

5. Save the PAT secret in a secure location.

   > **Important**
   >
   > The PAT secret only displays once. You cannot retrieve it after you close the window.

Next, update your `build.gradle` file with the following snippet:

```
repositories {
    maven {
        name = 'spaceName'
        url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
        credentials(PasswordCredentials)
    }
}
```

Method 2: Put the PAT in a separate file

Use this method if you do not want to modify your `gradle.properties` file.

First, create a PAT if you do not have one.

To create a personal access token (PAT)

1. In the top menu bar, choose your profile badge, and then choose **My settings**.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

2. In **PAT name**, enter a descriptive name for your PAT.
3. In **Expiration date**, leave the default date or choose the calendar icon to select a custom date. The expiration date defaults to one year from the current date.
4. Choose **Create**.

   You can also create this token when you choose **Clone repository** for a source repository.

5. Save the PAT secret in a secure location.

   > **Important**
   >
   > The PAT secret only displays once. You cannot retrieve it after you close the window.

To put your PAT in a separate file

1. Update your `build.gradle` file with the following snippet. Replace *space_name*, *proj_name*, and *repo_name* with your CodeCatalyst user name, space name, project name, and package repository name.

```
def props = new Properties()
file("fileName").withInputStream { props.load(it) }

repositories {
        maven {
            name = 'spaceName'
            url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
            credentials(PasswordCredentials)
        }
    }
}
```

2. Write your PAT into the file that was specified in your `build.gradle` file:

```
echo "codecatalystArtifactsToken=PAT" > fileName
```

Configuring and using mvn

You use the `mvn` command to run Maven builds. You must configure `mvn` to use your package repository and provide a personal access token (PAT) for authentication.

Contents

Fetching dependencies from CodeCatalyst

To configure `mvn` to fetch dependencies from a CodeCatalyst repository, you must edit the Maven configuration file, `settings.xml` and optionally, your project's Project Model Object (POM) file.

The POM file contains information about the project and configuration information for Maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, and goals.

To use `mvn` to fetch dependencies from your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the overview page for your project, choose **Packages**.
3. Choose your package repository from the list of package repositories.
4. Choose **Connect to repository**.
5. In the **Connect to repository** dialog box, choose **mvn** from the list of package manager clients.
6. You will need a personal access token (PAT) to authenticate `mvn` with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.

   a. Choose **Create token**.

   b. Choose **Copy** to copy your PAT.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box.

7. Add a profile containing your repository to your `settings.xml` file. Replace the following values.

   > **Note**
   >
   > If copying from the console instructions, the following values should be updated for you and should not be changed.

   - Replace *`space_name`* with your CodeCatalyst space name.
   - Replace *`proj_name`* with your CodeCatalyst project name.
   - Replace *`repo_name`* with your CodeCatalyst package repository name.

```
<profiles>
  <profile>
```

```
    <id>repo_name</id>
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
        <repository>
          <id>repo_name</id>
          <url>https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/</url>
        </repository>
    </repositories>
  </profile>
</profiles>
```

8. Add your server to the list of servers in your `settings.xml` file. Replace the following values.

> **Note**
>
> If copying from the console instructions, the following values should be updated for you and should not be changed.

- Replace *repo_name* with your CodeCatalyst package repository name.
- Replace *username* with your CodeCatalyst user name.
- Replace *PAT* with your CodeCatalyst PAT.

```
<servers>
  <server>
    <id>repo_name</id>
    <username>username</username>
    <password>PAT</password>
  </server>
</servers>
```

9. (Optional) Set a mirror in your `settings.xml` file that captures all connections and routes them to your repository instead of a gateway repository.

> **Note**
>
> If copying from the console instructions, the following values should be updated for you and should not be changed.

- Replace *space_name* with your CodeCatalyst space name.
- Replace *proj_name* with your CodeCatalyst project name.
- Replace *repo_name* with your CodeCatalyst package repository name.

```
<mirrors>
  <mirror>
    <id>repo_name</id>
    <name>repo_name</name>
    <url>https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/</url>
    <mirrorOf>*</mirrorOf>
  </mirror>
</mirrors>
```

> **Important**
>
> You can use any value in the `<id>` element, but it must be the same in both the `<server>` and `<repository>` elements. This enables the specified credentials to be included in requests to CodeCatalyst.

After you make these configuration changes, you can build the project.

```
mvn compile
```

Fetching packages from external package repositories through CodeCatalyst

You can install Maven packages from public repositories through a CodeCatalyst repository by configuring it with an upstream connection to the gateway that represents the gateway repository.

Packages installed from the gateway repository are ingested and stored in your CodeCatalyst repository.

Currently, CodeCatalyst supports the following public Maven package repositories.

- maven-central-gateway
- google-android-gateway
- gradle-plugins-gateway
- commonsware-gateway

To install packages from public Maven package repositories

1. If you haven't already, configure `mvn` with your CodeCatalyst package repository by following the steps in Fetching dependencies from CodeCatalyst.
2. Ensure your repository has added the gateway repository you want to install from as an upstream connection. To check which upstream sources are added or to add a gateway repository as an upstream source, followthe instructions in Adding an upstream repository.

For more information about requesting packages from upstream repositories, see Requesting a package version with upstream repositories.

Publishing packages to CodeCatalyst

To publish a Maven package with `mvn` to a CodeCatalyst repository, you must also edit `~/.m2/settings.xml` and the project POM.

To use `mvn` to publish packages to your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the overview page for your project, choose **Packages**.
3. Choose your package repository from the list of package repositories.
4. Choose **Connect to repository**.
5. In the **Connect to repository** dialog box, choose **mvn** from the list of package manager clients.
6. You will need a personal access token (PAT) to authenticate `mvn` with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.

    a. Choose **Create token**.

b. Choose **Copy** to copy your PAT.

> **Warning**
>
> You will not be able to see or copy your PAT again after you close the dialog box.

7. Configure an environment variable on your local machine with your PAT. You will use this environment variable in your `setting.xml` file.

```
export CODECATALYST_ARTIFACTS_TOKEN=your_PAT
```

8. Add a `<servers>` section to `settings.xml` with a reference to the `CodeCatalyst_ARTIFACTS_TOKEN` environment variable so that Maven passes the token in HTTP requests.

```
<settings>
...
    <servers>
        <server>
            <id>repo-name</id>
            <username>username</username>
            <password>${env.CodeCatalyst_ARTIFACTS_TOKEN}</password>
        </server>
    </servers>
...
</settings>
```

9. Add a `<distributionManagement>` section to your project's `pom.xml`.

```
<project>
...
     <distributionManagement>
         <repository>
             <id>repo_name</id>
             <name>repo_name</name>
             <url>https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/</url>
         </repository>
     </distributionManagement>
...
</project>
```

After you make these configuration changes, you can build the project and publish it to the specified repository.

```
mvn deploy
```

You can navigate to your package repository in the CodeCatalyst console to check that the package was successfully published.

Publishing third-party packages

You can publish third-party Maven packages to a CodeCatalyst repository with `mvn deploy:deploy-file`. This can be helpful to users that want to publish packages and only have JAR files and don't have access to package source code or POM files.

The `mvn deploy:deploy-file` command will generate a POM file based on the information passed in the command line.

First, create a PAT if you do not have one.

To create a personal access token (PAT)

1. In the top menu bar, choose your profile badge, and then choose **My settings**.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

2. In **PAT name**, enter a descriptive name for your PAT.
3. In **Expiration date**, leave the default date or choose the calendar icon to select a custom date. The expiration date defaults to one year from the current date.
4. Choose **Create**.

   You can also create this token when you choose **Clone repository** for a source repository.
5. Save the PAT secret in a secure location.

   > **Important**
   >
   > The PAT secret only displays once. You cannot retrieve it after you close the window.

To publish third-party Maven packages

1. Create a `~/.m2/settings.xml` file with the following contents:

```
<settings>
    <servers>
        <server>
            <id>repo_name</id>
            <username>username</username>
            <password>PAT}</password>
        </server>
    </servers>
</settings>
```

2. Run the `mvn deploy:deploy-file` command:

```
mvn deploy:deploy-file -DgroupId=commons-cli          \
-DartifactId=commons-cli       \
-Dversion=1.4                  \
-Dfile=./commons-cli-1.4.jar   \
-Dpackaging=jar                \
-DrepositoryId=repo-name      \
-Durl=https://packages.region.codecatalyst.aws/maven/space-name/proj-name/repo-
name/
```

> **Note**
>
> The preceding example publishes `commons-cli 1.4`. Modify the groupId, artifactID, version, and file arguments to publish a different JAR.

These instructions are based on examples in the Guide to deploying 3rd party JARs to remote repository from the *Apache Maven documentation*.

For more information, see these topics on the Apache Maven Project website:

- Setting up Multiple Repositories
- Settings Reference

- Distribution Management
- Profiles

Publishing packages with curl

This section shows how to use the HTTP client `curl` to publish Maven packages to a CodeCatalyst package repository. Publishing packages with `curl` can be useful if you do not have or want to install the Maven client in your environments.

To publish a Maven package with `curl`

1. You must store a personal access token (PAT) into an environment variable to authenticate `curl` with CodeCatalyst. If you already have one, you can use that. If not, you can create one and configure the environment variable.

   a. Create a PAT by following the steps in Grant users repository access with personal access tokens. Copy the PAT to store it in an environment variable.

   b. On your local machine's command line, configure an environment variable with your PAT.

   ```
   export CodeCatalyst_ARTIFACTS_TOKEN=your_PAT
   ```

2. Use the following `curl` command to publish the JAR to a CodeCatalyst repository. Replace *username*, *space_name*, *proj_name*, and *repo_name* with your CodeCatalyst user name, space name, project name, and package repository name.

   ```
   curl --request PUT https://packages.region.codecatalyst.aws/maven/space-name/proj-
   name/repo-name/com/mycompany/app/my-app/1.0/my-app-1.0.jar \
        --user "username:CodeCatalyst_ARTIFACTS_TOKEN" --header "Content-Type:
    application/octet-stream" \
        --data-binary @target/path/to/my-app-1.0.jar
   ```

3. Use the following `curl` command to publish the POM to a CodeCatalyst repository. Replace *username*, *space_name*, *proj_name*, and *repo_name* with your CodeCatalyst user name, space name, project name, and package repository name.

   ```
   curl --request PUT https://packages.region.codecatalyst.aws/maven/space-name/proj-
   name/repo-name/com/mycompany/app/my-app/1.0/my-app-1.0.pom \
        --user "username:CodeCatalyst_ARTIFACTS_TOKEN" --header "Content-Type:
    application/octet-stream" \
   ```

```
    --data-binary @target/my-app-1.0.pom
```

4. At this point, the Maven package will be in your CodeCatalyst repository with a status of `Unfinished`. To be able to consume the package, it must be in the `Published` state. You can move the package from `Unfinished` to `Published` by either uploading a `maven-metadata.xml` file to your package, or changing the status in the CodeCatalyst console.

   a. Option 1: Use the following `curl` command to add a `maven-metadata.xml` file to your package. Replace *`username`*, *`space_name`*, *`proj_name`*, and *`repo_name`* with your CodeCatalyst user name, space name, project name, and package repository name.

   ```
   curl --request PUT https://packages.region.codecatalyst.aws/maven/space-
   name/proj-name/repo-name/com/mycompany/app/my-app/maven-metadata.xml \
       --user "username:CodeCatalyst_ARTIFACTS_TOKEN" --header "Content-Type:
    application/octet-stream" \
       --data-binary @target/maven-metadata.xml
   ```

   Following is an example of the contents of a `maven-metadata.xml` file:

   ```
   <metadata modelVersion="1.1.0">
       <groupId>com.mycompany.app</groupId>
       <artifactId>my-app</artifactId>
       <versioning>
           <latest>1.0</latest>
           <release>1.0</release>
           <versions>
               <version>1.0</version>
           </versions>
           <lastUpdated>20200731090423</lastUpdated>
       </versioning>
   </metadata>
   ```

   b. Option 2: Update the package status to `Published` in the CodeCatalyst console. For information about how to update a package version's status, see Updating a package version's status.

If you only have a package's JAR file, you can publish a consumable package version to a CodeCatalyst repository using `mvn`. This can be useful if you do not have access to the package's source code or POM. See Publishing third-party packages for details.

Using Maven checksums and snapshots

The following sections describe how to use Maven checksums and Maven snapshots in CodeCatalyst.

Using Maven checksums

When a Maven package is published to a CodeCatalyst package repository, the checksum associated with each *asset* or file in the package is used to validate the upload. Examples of assets are *jar*, *pom*, and *war* files. For each asset, the Maven package contains multiple checksum files that use the asset name with an additional extension, such as `md5` or `sha1`. For example, the checksum files for a file named `my-maven-package.jar` might be `my-maven-package.jar.md5` and `my-maven-package.jar.sh1`.

Every Maven package also contains a `maven-metadata.xml` file. This file must be uploaded for a publish to succeed. If a checksum mismatch is detected during the upload of any package file, the publish stops. This might prevent the `maven-metadata.xml` from being uploaded. When that happens, the status of the Maven package is set to `Unfinished`. You cannot download assets that are part of a package with this status.

Keep the following in mind in the event of a checksum mismatch when you publish a Maven package:

- If the checksum mismatch occurs before `maven-metadata.xml` is uploaded, the status of the package is not set to `Unfinished`. The package is not visible and its assets cannot be consumed. When this happens, try one of the following, and then try to download the asset again.
  - Run the command that publishes the Maven package again. This might work if a network issue corrupted the checksum file during download. If the network issue is resolved for the retry, the checksum matches and the download is successful.
  - If republishing the Maven package doesn't work, delete the package and then republish it.
- If the checksum mismatch occurs after `maven-metadata.xml` is uploaded, the status of the package is set to `Published`. You can consume any asset from the package, including those with checksum mismatches. When you download an asset, the checksum generated by CodeCatalyst is downloaded with it. If the downloaded file is associated with a checksum mismatch, its downloaded checksum file might not match the checksum that was uploaded when the package was published.

Using Maven snapshots

A Maven *snapshot* is a special version of a Maven package that refers to the latest production branch code. It is a development version that precedes the final release version. You can identify a snapshot version of a Maven package by the suffix `SNAPSHOT` that is appended to the package version. For example, the snapshot of version `1.1` is `1.1-SNAPSHOT`. For more information, see What is a SNAPSHOT version? on the Apache Maven Project website.

CodeCatalyst supports publishing and consuming Maven snapshots. You can publish a Maven snapshot to a CodeCatalyst repository or, if you are directly connected, to an upstream repository. However, a snapshot version in both a package repository and one of its upstream repositories is not supported. For example, if you upload a Maven package with version `1.2-SNAPSHOT` to your package repository, CodeCatalyst does not support uploading a Maven package with the same snapshot version to one of its upstream repositories. This scenario might return unpredictable results.

When a Maven snapshot is published, its previous version is preserved in a new version called a *build*. Each time a Maven snapshot is published, a new build version is created. All previous versions of a snapshot are maintained in its build versions. When a Maven snapshot is published, its status is set to `Published` and the status of the build that contains the previous version is set to `Unlisted`.

If you request a snapshot, the version with status `Published` is returned. This is always the most recent version of the Maven snapshot. You can also request a particular build of a snapshot.

To delete all build versions of a Maven snapshot, use the CodeCatalyst console.

Using NuGet

These topics describe how to consume and publish `NuGet` packages using CodeCatalyst.

> **Note**
>
> CodeCatalyst supports NuGet version 4.8 and higher.

Topics

- Using CodeCatalyst with Visual Studio

- Configuring and using the nuget or dotnet CLI
- NuGet package name, version, and asset name normalization
- NuGet compatibility

Using CodeCatalyst with Visual Studio

You can consume packages from CodeCatalyst directly in Visual Studio.

To configure and use NuGet with CLI tools such as `dotnet` or `nuget`, see Configuring and using the nuget or dotnet CLI.

Contents

Configuring Visual Studio with CodeCatalyst

Windows

To configure Visual Studio with CodeCatalyst

1. A personal access token (PAT) is required to authenticate with CodeCatalyst. If you already have one, you can use that. If not, follow the instructions in Grant users repository access with personal access tokens to create one.
2. Use `nuget` or `dotnet` to configure your package repository and credentials.

   dotnet

   Linux and macOS users: Because encryption is not supported on non-Windows platforms, you must add the `--store-password-in-clear-text` flag to the following command. Note that this will store your password as plaintext in your configuration file.

   ```
   dotnet nuget add source https://packages.region.codecatalyst.aws/nuget/space-name/proj-name/repo-name/v3/index.json --name repo_name --password PAT --username user_name
   ```

nuget

```
nuget sources add -name repo_name -Source https://
packages.region.codecatalyst.aws/nuget/space-name/proj-name/repo-name/v3/
index.json -password PAT --username user_name
```

Example output:

```
Package source with Name: repo_name added successfully.
```

3. Configure Visual Studio to use your new package source. In Visual Studio, choose **Tools**, and then choose **Options**.
4. In the **Options** menu, expand the **NuGet Package Manager** section and choose **Package Sources**.
5. In the **Available package sources** list, make sure that your *repo_name* source is enabled. If you have configured your package repository with an upstream connection to the NuGet Gallery, disable the **nuget.org** source .

macOS

To configure Visual Studio with CodeCatalyst

1. A personal access token (PAT) is required to authenticate with CodeCatalyst. If you already have one, you can use that. If not, follow the instructions in Grant users repository access with personal access tokens to create one.
2. Choose **Preferences** from the menu bar.
3. In the **NuGet** section, choose **Sources**.
4. Choose **Add** and add your repository information.
   a. For **Name**, enter your CodeCatalyst package repository name.
   b. For **Location**, enter your CodeCatalyst package repository endpoint. The following snippet shows an example endpoint. Replace *space-name*, *proj-name*, and *repo-name* with your CodeCatalyst space name, project name, and repository name.

```
https://packages.region.codecatalyst.aws/nuget/space-name/proj-name/repo-name/
```

c. For **Username**, enter any valid value.

d. For **Password**, enter your PAT.

5. Choose **Add source**.
6. If you have configured your package repository with an upstream connection to the NuGet Gallery, disable the **nuget.org** source.

After configuration, Visual Studio can consume packages from your CodeCatalyst repository, any of its upstream repositories, or from NuGet.org if you have it configured as an upstream source. For more information about browsing and installing NuGet packages in Visual Studio, see Install and manage packages in Visual Studio using the NuGet Package Manager in the *NuGet documentation*.

Configuring and using the nuget or dotnet CLI

You can use CLI tools such as `NuGet` and `dotnet` to publish and consume packages from CodeCatalyst. This document provides information about configuring the CLI tools and using them to publish or consume packages.

Contents

Configuring NuGet with CodeCatalyst

To configure NuGet with CodeCatalyst, add a repository endpoint and personal access token to your NuGet configuration file to allow `nuget` or `dotnet` to connect to your CodeCatalyst package repository.

To configure NuGet with your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the overview page for your project, choose **Packages**.
3. Choose your package repository from the list of package repositories.
4. Choose **Connect to repository**.

5. In the **Connect to repository** dialog box, choose **NuGet** or **dotnet** from the list of package manager clients.
6. You will need a personal access token (PAT) to authenticate NuGet with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.

   a. Choose **Create token**.

   b. Choose **Copy** to copy your PAT.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box.

7. Configure `nuget` or `dotnet` to use your repository's NuGet endpoint and CodeCatalyst PAT. Replace the following values.

   > **Note**
   >
   > If copying from the console instructions, the following values should be updated for you and should not be changed.

   - Replace *`username`* with your CodeCatalyst user name.
   - Replace *`PAT`* with your CodeCatalyst PAT.
   - Replace *`space_name`* with your CodeCatalyst space name.
   - Replace *`proj_name`* with your CodeCatalyst project name.
   - Replace *`repo_name`* with your CodeCatalyst package repository name.

   a. For `nuget`, use the `nuget sources add` command.

   ```
   nuget sources add -name "repo_name" -Source "https://
   packages.region.codecatalyst.aws/nuget/space_name/proj_name/repo_name/v3/
   index.json" -username "username" -password "PAT"
   ```

   b. For `dotnet`, use the `dotnet nuget add source` command.

Linux and macOS users: Because encryption is not supported on non-Windows platforms, you must add the `--store-password-in-clear-text` flag to the following command. Note that this will store your password as plaintext in your configuration file.

```
dotnet nuget add source "https://packages.region.codecatalyst.aws/
nuget/space_name/proj_name/repo_name/v3/index.json" -n "proj_name/repo_name" -u
 "username" -p "PAT" --store-password-in-clear-text
```

Once you have configured NuGet with CodeCatalyst, you can consume NuGet packages that are stored in your CodeCatalyst repository or one of its upstream repositories and publish NuGet packages to your CodeCatalyst repository.

Consuming NuGet packages from a CodeCatalyst repository

Once you have configured NuGet with CodeCatalyst, you can consume NuGet packages that are stored in your CodeCatalyst repository or one of its upstream repositories.

To consume a package version from a CodeCatalyst repository or one of its upstream repositories with nuget or dotnet, run the following command. Replace *packageName* with the name of the package you want to consume and *packageSourceName* with the source name for your CodeCatalyst package repository in your NuGet configuration file, which should be the repository name.

To install a package with `dotnet`

```
dotnet add packageName --source packageSourceName
```

To install a package with `nuget`

```
nuget install packageName --source packageSourceName
```

For more information, see Manage packages using the nuget CLI or Install and manage packages using the dotnet CLI in the *Microsoft Documentation*.

Consuming NuGet packages from NuGet.org through CodeCatalyst

You can consume NuGet packages from NuGet.org through a CodeCatalyst repository by configuring the repository with an upstream connection to **NuGet.org**. Packages consumed from **NuGet.org** are ingested and stored in your CodeCatalyst repository.

To consume packages from NuGet.org

1. If you haven't already, configure your NuGet package manager with your CodeCatalyst package repository by following the steps in Configuring NuGet with CodeCatalyst.
2. Ensure that your repository has added **NuGet.org** as an upstream connection. You can check which upstream sources are added or add **Nuget.org** as an upstream source by following the instructions in Adding an upstream repository and choosing the **NuGet store** repository.

Publishing NuGet packages to CodeCatalyst

Once you have configured NuGet with CodeCatalyst, you can use `nuget` or `dotnet` to publish package versions to CodeCatalyst repositories.

To push a package version to a CodeCatalyst repository, run the following command with the full path to your `.nupkg` file and the source name for your CodeCatalyst repository in your NuGet configuration file.

To publish a package with `dotnet`

```
dotnet nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

To publish a package with `nuget`

```
nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

NuGet package name, version, and asset name normalization

CodeCatalyst normalizes package and asset names and package versions before storing them, which means the names or versions in CodeCatalyst may be different than the ones provided when the package or asset was published.

Package name normalization: CodeCatalyst normalizes NuGet package names by converting all letters to lowercase.

Package version normalization: CodeCatalyst normalizes NuGet package versions using the same pattern as NuGet. The following information is from Normalized version numbers from the NuGet documentation.

- Leading zeroes are removed from version numbers:
  - `1.00` is treated as `1.0`
  - `1.01.1` is treated as `1.1.1`
  - `1.00.0.1` is treated as `1.0.0.1`
- A zero in the fourth part of the version number will be omitted:
  - `1.0.0.0` is treated as `1.0.0`
  - `1.0.01.0` is treated as `1.0.1`
- SemVer 2.0.0 build metadata is removed:
  - `1.0.7+r3456` is treated as `1.0.7`

Package asset name normalization: CodeCatalyst constructs the NuGet package asset name from the normalized package name and package version.

NuGet compatibility

This guide contains information about CodeCatalyst's compatibility with different NuGet tools and versions.

Topics

General NuGet compatibility

CodeCatalyst supports NuGet 4.8 and higher.

CodeCatalyst only supports V3 of the NuGet HTTP protocol. This means that some CLI commands that rely V2 of the protocol are not supported. See the following nuget command support section for more information.

CodeCatalyst does not support PowerShellGet 2.x.

NuGet command line support

CodeCatalyst supports the NuGet (`nuget`) and .NET Core (`dotnet`) CLI tools.

nuget command support

Because CodeCatalyst only supports V3 of NuGet's HTTP protocol, the following commands will not work when used against CodeCatalyst resources:

- `list`: The `nuget list` command displays a list of packages from a given source. To get a list of packages in a CodeCatalyst package repository, navigate to the repository in the CodeCatalyst console.

Using Python

These topics describe how to use `pip`, the Python package manager, and `twine`, the Python package publishing utility, with CodeCatalyst.

Topics

Configuring pip and installing Python packages

To use `pip` with CodeCatalyst, you must connect `pip` to your package repository and provide a personal access token for authentication. You can view instructions for connecting `pip` to your package repository in the CodeCatalyst console. After you authenticate and connect `pip` to CodeCatalyst, you can run `pip` commands.

Contents

- Supported client-side commands

Installing Python packages from CodeCatalyst with pip

The following instructions explain how to configure `pip` to install Python packages from your CodeCatalyst package repository or one of its upstream repositories.

To configure and use `pip` to install Python packages from your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the overview page for your project, choose **Packages**.
3. Choose your package repository from the list of package repositories.
4. Choose **Connect to repository**.
5. In the **Connect to repository** dialog box, choose **pip** from the list of package manager clients.
6. You will need a personal access token (PAT) to authenticate pip with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.

   a. Choose **Create token**.

   b. Choose **Copy** to copy your PAT.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box.

7. Use the `pip config` command to set the CodeCatalyst registry URL and credentials. Replace the following values.

   > **Note**
   >
   > If copying from the console instructions, the following values should be updated for you and should not be changed.

   - Replace *`username`* with your CodeCatalyst user name.
   - Replace *`PAT`* with your CodeCatalyst PAT.

- Replace *space_name* with your CodeCatalyst space name.
- Replace *proj_name* with your CodeCatalyst project name.
- Replace *repo_name* with your CodeCatalyst package repository name.

```
pip config set global.index-url https://username:PAT@https://
packages.region.codecatalyst.aws/pypi/space_name/proj_name/repo_name/simple/
```

8. Assuming that a package is present in your repository or one of its upstream repositories, you can install it with `pip install`. For example, use the following command to install the `requests` package.

```
pip install requests
```

Use the `-i` option to revert temporarily to installing packages from https://pypi.org instead of your CodeCatalyst package repository.

```
pip install -i https://pypi.org/simple requests
```

Consuming Python packages from PyPI through CodeCatalyst

You can consume Python packages from the Python Package Index (PyPI) through a CodeCatalyst repository by configuring the repository with an upstream connection to **PyPI**. Packages consumed from **PyPI** are ingested and stored in your CodeCatalyst repository.

To consume packages from PyPI

1. If you haven't already, configure pip with your CodeCatalyst package repository by following the steps in Installing Python packages from CodeCatalyst with pip.
2. Ensure that your repository has added **PyPI** as an upstream source. You can check which upstream sources are added or add **PyPI** as an upstream source by following the instructions in Adding an upstream repository and choosing the **PyPI store** repository.

For more information about requesting packages from upstream repositories, see Requesting a package version with upstream repositories.

pip command support

The following sections summarize the pip commands that are supported, by CodeCatalyst repositories, in addition to specific commands that are not supported.

Topics

Supported commands that interact with a repository

This section lists `pip` commands where the `pip` client makes one or more requests to the registry it's been configured with. These commands have been verified to function correctly when invoked against a CodeCatalyst package repository.

Command	Description
install	Install packages.
download	Download packages.

CodeCatalyst does not implement `pip search`. If you have configured `pip` with a CodeCatalyst package repository, running `pip search` will search and show packages from PyPI.

Supported client-side commands

These commands don't require any direct interaction with a repository, so CodeCatalyst does not need to do anything to support them.

Command	Description
uninstall	Uninstall packages.
freeze	Output installed packages in requirements format.
list	List installed packages.

Command	Description
show	Show information about installed packages.
check	Verify that installed packages have compatible dependencies.
config	Manage local and global configuration.
wheel	Build wheels from your requirements.
hash	Compute hashes of package archives.
completion	Helps with command completion.
debug	Show information useful for debugging.
help	Show help for commands.

Configuring Twine and publishing Python packages

To use `twine` with CodeCatalyst, you must connect `twine` to your package repository and provide a personal access token for authentication. You can view instructions for connecting `twine` to your package repository in the CodeCatalyst console. After you authenticate and connect `twine` to CodeCatalyst, you can run `twine` commands.

Publishing packages to CodeCatalyst with Twine

The following instructions explain how to authenticate and connect `twine` to your CodeCatalyst package repository.

To configure and use `twine` to publish packages to your CodeCatalyst package repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. On the overview page for your project, choose **Packages**.
3. Choose your package repository from the list of package repositories.
4. Choose **Connect to repository**.
5. In the **Connect to repository** dialog box, choose **Twine** from the list of package manager clients.

6. You will need a personal access token (PAT) to authenticate twine with CodeCatalyst. If you already have one, you can use that. If not, you can create one here.

   a. Choose **Create token**.

   b. Choose **Copy** to copy your PAT.

   > **Warning**
   >
   > You will not be able to see or copy your PAT again after you close the dialog box.

7. You can configure twine with a `.pypirc` file, or with environment variables.

   a. **To configure with a `.pypirc` file.**

   Open `~/.pypirc` in your editor of choice.

   Add an index server for CodeCatalyst, including the repository, user name, and PAT that you created and copied in a previous step. Replace the following values.

   > **Note**
   >
   > If copying from the console instructions, the following values should be updated for you and should not be changed.

   - Replace *`username`* with your CodeCatalyst user name.
   - Replace *`PAT`* with your CodeCatalyst PAT.
   - Replace *`space_name`* with your CodeCatalyst space name.
   - Replace *`proj_name`* with your CodeCatalyst project name.
   - Replace *`repo_name`* with your CodeCatalyst package repository name.

```
[distutils]
index-servers = proj-name/repo-name

[proj-name/repo-name]
repository = https://packages.region.codecatalyst.aws/
pypi/space_name/proj_name/repo_name/
password = PAT
```

```
username = username
```

b. **To configure with environment variables.**

Set the following environment variables. In the `TWINE_REPOSITORY_URL` value, update *space_name*, *proj_name*, and *repo_name* with your CodeCatalyst space, project, and package repository names.

```
export TWINE_USERNAME=username
```

```
export TWINE_PASSWORD=PAT
```

```
export TWINE_REPOSITORY_URL="https://packages.region.codecatalyst.aws/
pypi/space_name/proj_name/repo_name/"
```

8. Publish a Python distribution with the `twine upload` command.

Python package name normalization

CodeCatalyst normalizes package names before storing them, which means the package names in CodeCatalyst may be different than the name provided when the package was published.

For Python packages, when performing normalization the package name is lowercased and all instances of the characters `.`, `-`, and `_` are replaced with a single `-` character. So the package names `pigeon_cli` and `pigeon.cli` are normalized and stored as `pigeon-cli`. The non-normalized name can be used by pip and twine. For more information about Python package name normalization, see PEP 503 in the Python documentation.

Python compatibility

While CodeCatalyst does not support the `/simple/` API, it does support the `Legacy` API operations. CodeCatalyst does not support PyPI's `XML-RPC` or `JSON` API operations.

For more information, see the following on the Python Packaging Authority's GitHub repository.

- Legacy API
- XML-RPC API
- JSON API

Quotas for packages

The following table describes quotas and limits for packages in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Resource	Default quota
Package repositories	Maximum of 1000 per space.
Direct upstream repositories	Maximum of 10 per package repository.
Upstream package repositories searched	Maximum of 25 upstream repositories searched per requested package version.
Package asset file size	Maximum of 5GB per package asset.
Package assets	Maximum of 150 per package version.

Build, test, and deploy with workflows in CodeCatalyst

After writing your application code in a CodeCatalyst Dev Environment and pushing it to your CodeCatalyst source repository, you're ready to deploy it. The way to do this automatically is through a workflow.

A *workflow* is an automated procedure that describes how to build, test, and deploy your code as part of a continuous integration and continuous delivery (CI/CD) system. A workflow defines a series of steps, or *actions*, to take during a workflow run. A workflow also defines the events, or *triggers*, that cause the workflow to start. To set up a workflow, you create a *workflow definition file* using the CodeCatalyst console's visual or YAML editor.

> **Tip**
>
> For a quick look at how you might use workflows in a project, create a project with a blueprint. Each blueprint deploys a functioning workflow that you can review, run, and experiment with.

About the workflow definition file

A *workflow definition file* is a YAML file that describes your workflow. The file is stored in a `~/.codecatalyst/workflows/` folder in the root of your source repository. The file can have a .yml or .yaml extension.

The following is an example of a simple workflow definition file. We explain each line of this example in the table that follows.

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
```

```
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: docker build -t MyApp:latest .
```

Line	Description
`Name: MyWorkflow`	Specifies the name of the workflow. For more information about the `Name` property, see Top-level properties.
`SchemaVersion: 1.0`	Specifies the workflow schema version. For more information about the `SchemaVersion` property, see Top-level properties.
`RunMode: QUEUED`	Indicates how CodeCatalyst handles multiple runs. For more information about the run mode, see Configuring the queuing behavior of runs.
`Triggers:`	Specifies the logic that will cause a workflow run to start. For more information about triggers, see Starting a workflow run automatically with triggers.

`- Type: PUSH`
`  Branches:`
`    - main`
	Indicates that the workflow must start whenever you push code to the `main` branch of the default source repository. For more information about the workflow source, see Connecting a workflow to a source repository.

`Actions:`	Defines the tasks to perform during a workflow run. In this example, the `Actions` section defines a single action called `Build`. For more information about actions, see Configuring the actions that a workflow performs.

Line	Description
`Build:`	Defines the properties for the `Build` action. For more information about the build action, see Building with workflows.
`Identifier: aws/build@v1`	Specifies the unique, hard-coded identifier for the build action.

`Inputs:`
`  Sources:`
`  - WorkflowSource`
	Indicates that the build action should look in the `WorkflowSource` source repository to find the files it needs to complete its processin g. For more information, see Connecting a workflow to a source repository.

`Configuration:`	Contains the configuration properties that are specific to the build action.

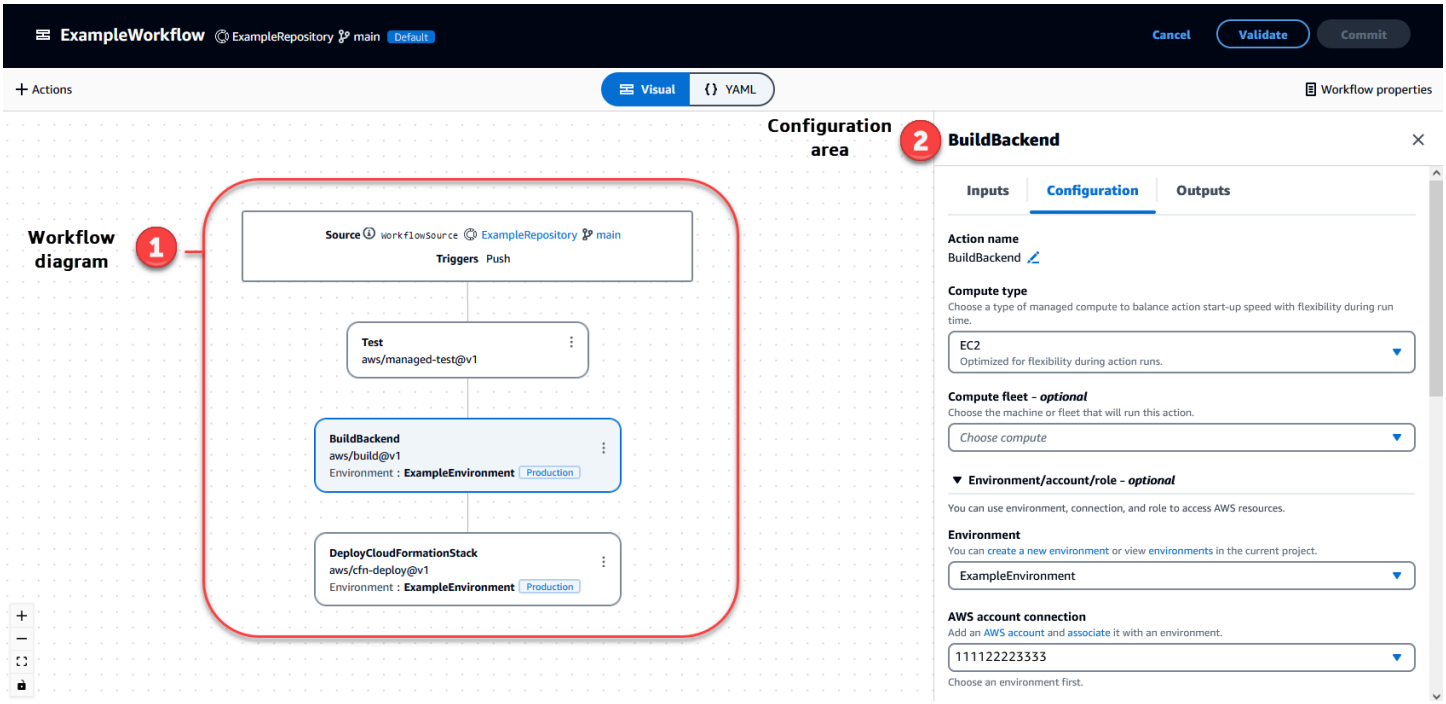
`Steps:`
`  - Run: docker build -t MyApp:lat`
`est .`
	Tells the build action to build a Docker image called `MyApp` and tag it with `latest`.

For a complete list of all the properties available in the workflow definition file, see the Workflow YAML definition.

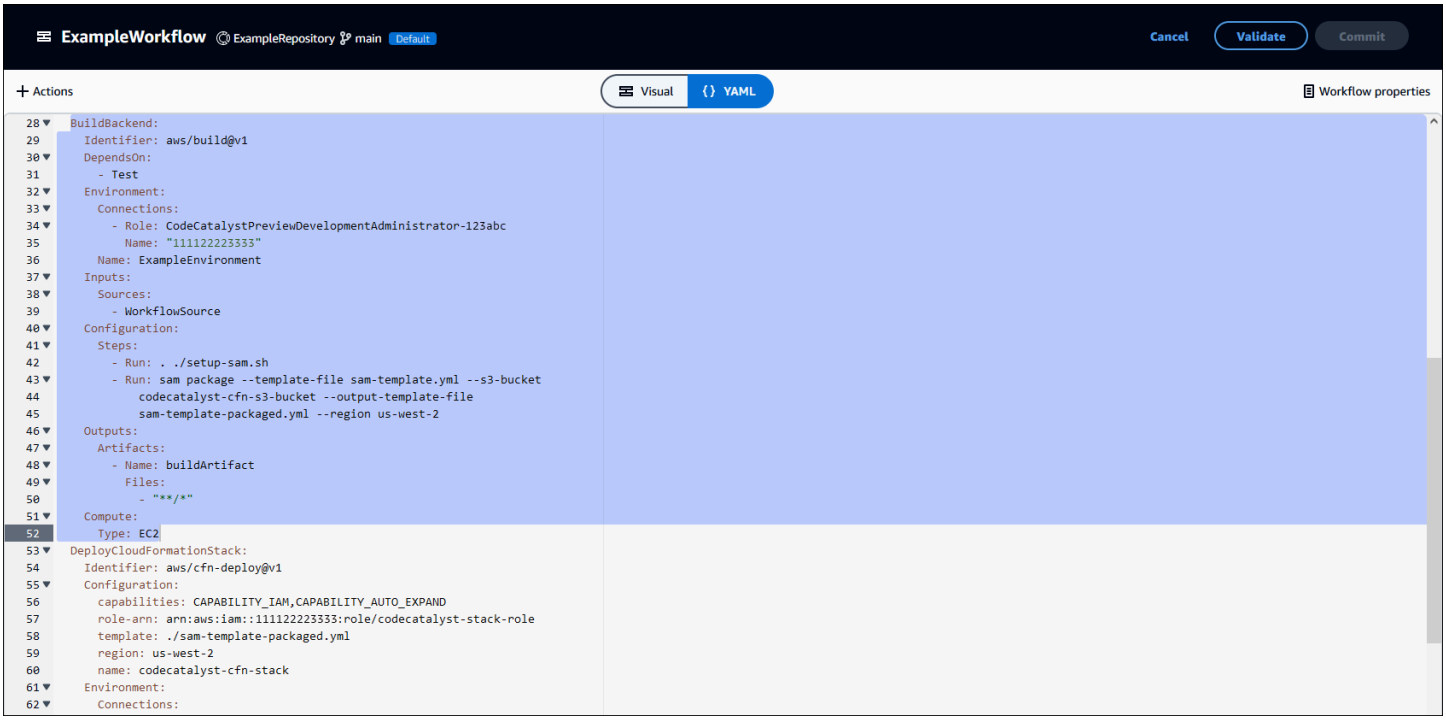
Using the CodeCatalyst console's visual and YAML editors

To create and edit the workflow definition file, you can use your preferred editor, but we recommend using the CodeCatalyst console's visual editor or YAML editor. These editors offer helpful file validation to help ensure YAML property names, values, nesting, spacing, capitalization, and so on, are correct.

The following image shows a workflow in the visual editor. The visual editor offers you a complete user interface through which to create and configure your workflow definition file. The visual editor includes a workflow diagram (1) showing the workflow's main components, and a configuration area (2).



Alternatively, you can use the YAML editor, shown in the next image. Use the YAML editor to paste in large code blocks (from a tutorial, for example), or to add advanced properties that are not offered through the visual editor.

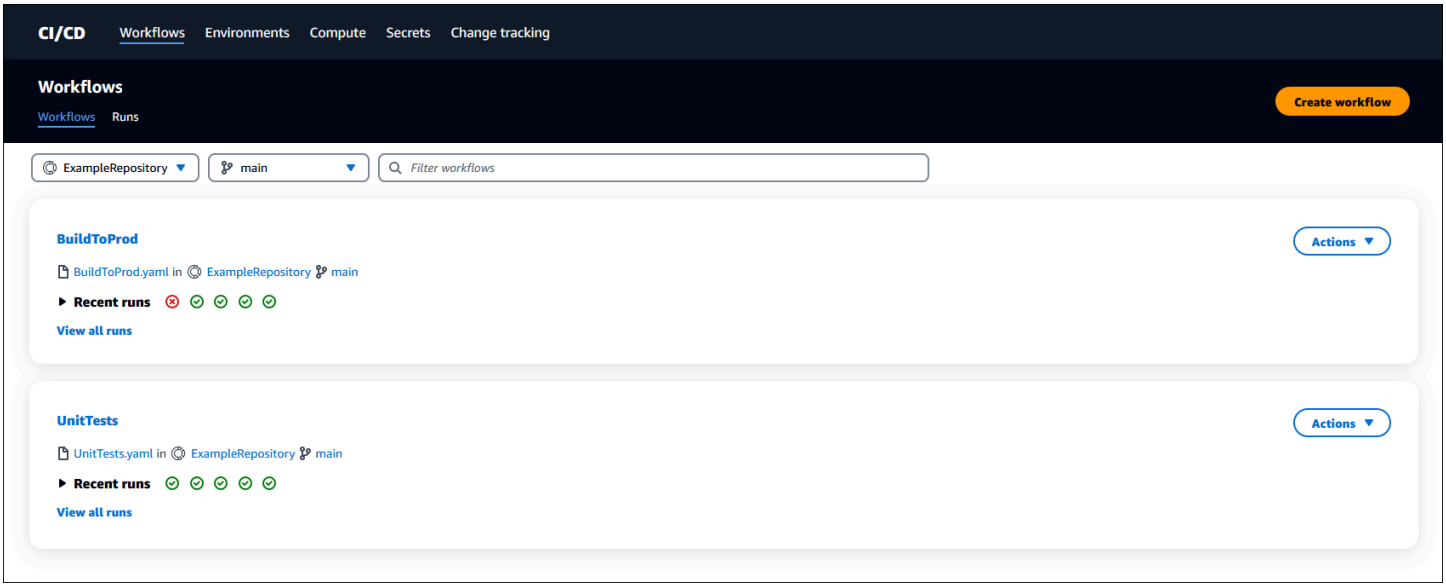


You can toggle from the visual editor to the YAML editor to see the effect that your configurations have on the underlying YAML code.

Discovering workflows

You can view your workflow on the **Workflows** summary page, along with other workflows you've set up in the same project.

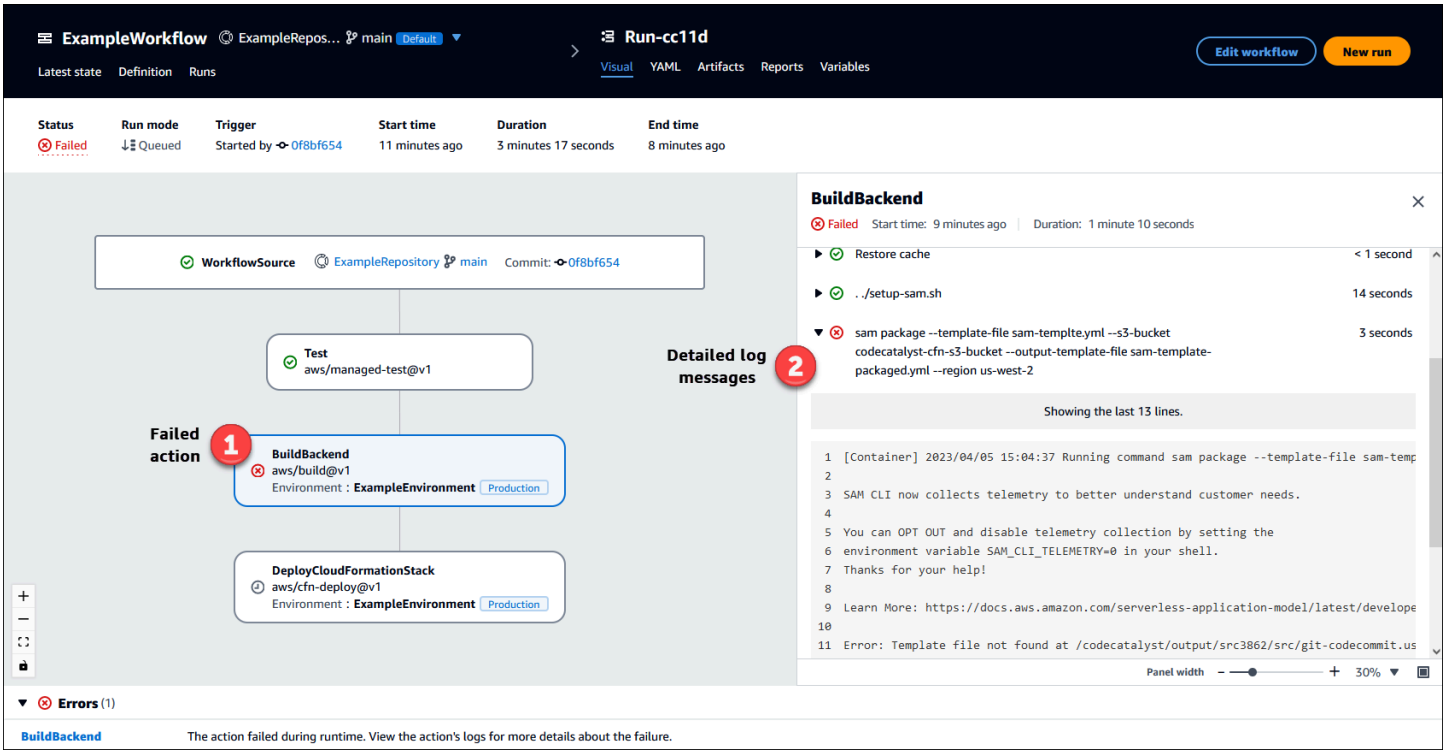
The following image shows the **Workflows** summary page. It is populated with two workflows: **BuildToProd** and **UnitTests**. You can see that both have been run a few times. You can choose **Recent runs** to quickly see the run history, or choose the name of the workflow to see the workflow's YAML code and other detailed information.



Viewing workflow run details

You can view the details of a workflow run by choosing the run in the **Workflows** summary page.

The following image shows the details of a workflow run called **Run-cc11d** that was started automatically on a commit to source. The workflow diagram indicates that an action has failed (1). You can navigate to the logs (2) to view the detailed log messages and troubleshoot issues. For more information about workflow runs, see Running a workflow.



Next steps

To learn more about workflows concepts, see Workflows concepts.

To create your first workflow, see Getting started with workflows.

Workflows concepts

Here are some concepts and terms to know when building, testing, or deploying your code with workflows in CodeCatalyst.

Workflows

A *workflow* is an automated procedure that describes how to build, test, and deploy your code as part of a continuous integration and continuous delivery (CI/CD) system. A workflow defines a series of steps, or *actions*, to take during a workflow run. A workflow also defines the events, or *triggers*, that cause the workflow to start. To set up a workflow, you create a *workflow definition file* using the CodeCatalyst console's visual or YAML editor.

> **Tip**
>
> For a quick look at how you might use workflows in a project, create a project with a blueprint. Each blueprint deploys a functioning workflow that you can review, run, and experiment with.

Workflow definition files

A *workflow definition file* is a YAML file that describes your workflow. The file is stored in a `~/.codecatalyst/workflows/` folder in the root of your source repository. The file can have a .yml or .yaml extension.

For more information about the workflow definition file, see Workflow YAML definition.

Actions

An *action* is the main building block of a workflow, and defines a logical unit of work, or task, to perform during a workflow run. Typically, a workflow includes multiple actions that run sequentially or in parallel depending on how you've configured them.

For more information about actions, see Configuring the actions that a workflow performs.

Action groups

An *action group* contains one or more actions. Grouping actions into action groups helps you keep your workflow organized, and also allows you to configure dependencies between different groups.

For more information about action groups, see Grouping actions into action groups.

Artifacts

An *artifact* is the output of a workflow action, and typically consists of a folder or archive of files. Artifacts are important because they allow you to share files and information between actions.

For more information about artifacts, see Sharing data between actions in a workflow using artifacts.

Compute

Compute refers to the computing engine (the CPU, memory, and operating system) managed and maintained by CodeCatalyst to run workflow actions.

For more information about compute, see Configuring the compute and runtime environment Docker images for a workflow.

Environments

A CodeCatalyst *environment*, not to be confused with a Dev Environment, defines the target AWS account and optional Amazon VPC that a CodeCatalyst workflow connects to. An environment also defines the IAM role that a workflow needs to access the AWS services and resources within the target account.

You can set up multiple environments and give them names such as development, test, staging, and production. When you deploy into these environments, information about the deployments appears on the CodeCatalyst **Deployment activity** and **Deployment targets** tabs in the environment.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.

Gates

A *gate* is a workflow component that you can use to prevent a workflow run from proceeding unless certain conditions are met. An example of a gate is the **Approval** gate where users must submit an approval in the CodeCatalyst console before the workflow run is allowed to continue.

You can add gates between sequences of actions in a workflow, or before the first action (which runs immediately after the **Source** downloads). You can also add gates after the last action, if you have a need to do so.

For more information about gates, see Gating a workflow run.

Reports

A *report* contains details about tests that occur during a workflow run. You can create reports such as a test report, a code coverage report, a software composition analysis report, and a static analysis report. You can use a report to help troubleshoot a problem during a workflow. If you have

many reports from multiple workflows, you can use your reports to view trends and failure rates to help you optimize your applications and deployment configurations.

For more information about reports, see Quality report types.

Runs

A *run* is a single iteration of a workflow. During a run, CodeCatalyst performs the actions defined in the workflow configuration file and outputs the associated logs, artifacts, and variables.

For more information about runs, see Running a workflow.

Sources

A *source*, also called an *input source*, is a source repository to which a workflow action connects in order to obtain the files it needs to carry out its operations. For example, a workflow action might connect to a source repository to obtain application source files in order to build an application.

For more information about sources, see Connecting a workflow to a source repository.

Variables

A *variable* is a key-value pair that contains information that you can reference in your CodeCatalyst workflow.

For more information about variables, see Configuring and using variables in a workflow.

Workflow triggers

A *workflow trigger*, or simply a *trigger*, allows you to start a workflow run automatically when certain events occur, like a code push. You might want to configure triggers to free your software developers from having to start workflow runs manually through the CodeCatalyst console.

You can use three types of trigger:

- **Push** – A code push trigger causes a workflow run to start whenever a commit is pushed.
- **Pull request** – A pull request trigger causes a workflow run to start whenever a pull request is either created, revised, or closed.
- **Schedule** – A schedule trigger causes a workflow run to start on a schedule that you define. Consider using a schedule trigger to run nightly builds of your software so that the latest build is ready for your software developers to work on the next morning.

You can use push, pull request, and schedule triggers alone or in combination in the same workflow.

Triggers are optional—if you don't configure any, you can only start a workflow manually.

For more information about triggers, see Starting a workflow run automatically with triggers.

Getting started with workflows

In this tutorial, you'll learn how to create and configure your first workflow.

> **Tip**
>
> Prefer to start with a preconfigured workflow? See Creating a project with a blueprint, which includes instructions for setting up a project with a functioning workflow, sample application, and other resources.

Topics

Prerequisites

Before you begin:

- You need a CodeCatalyst **space**. For more information, see Creating a space.
- In your CodeCatalyst space, you need an empty, **Start from scratch** CodeCatalyst **project** called:

  ```
  codecatalyst-project
  ```

  For more information, see Creating an empty project in Amazon CodeCatalyst.

- In your project, you need a CodeCatalyst **repository** called:

```
codecatalyst-source-repository
```

For more information, see Creating a source repository.

> **Note**
>
> If you have an existing project and source repository, you can use them; however, creating new ones makes cleanup easier at the end of this tutorial.

Step 1: Create and configure your workflow

In this step, you create and configure a workflow that automatically builds and tests your source code when changes are made.

To create your workflow

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose **Create workflow**.

   The workflow definition file appears in the CodeCatalyst console's YAML editor.

To configure your workflow

You can configure your workflow in the **Visual** editor, or the **YAML** editor. Let's start with the YAML editor and then switch to the visual editor.

1. Choose **+ Actions** to see a list of workflow actions that you can add to your workflow.
2. In the **Build** action, choose **+** to add the action's YAML to your workflow definition file. Your workflow now looks similar to the following.

```
Name: Workflow_fe47
SchemaVersion: "1.0"

# Optional - Set automatic triggers.
Triggers:
```

```
  - Type: Push
    Branches:
      - main

# Required - Define action configurations.
Actions:
  Build_f0:
    Identifier: aws/build@v1

    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this workflow as
 a source

    Outputs:
      AutoDiscoverReports:
        Enabled: true
        # Use as prefix for the report files
        ReportNamePrefix: rpt

    Configuration:
      Steps:
        - Run: echo "Hello, World!"
        - Run: echo "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" >> report.xml
        - Run: echo "<testsuite tests=\"1\" name=\"TestAgentJunit\" >" >>
 report.xml
        - Run: echo "<testcase classname=\"TestAgentJunit\" name=\"Dummy
            Test\"/></testsuite>" >> report.xml
```

The workflow copies the files in the `WorkflowSource` source repository to the compute machine running the `Build_f0` action, prints `Hello, World!` to the logs, discovers test reports on the compute machine, and outputs them to the CodeCatalyst console's **Reports** page.

3. Choose **Visual** to view the workflow definition file in the visual editor. The fields in the visual editor let you configure the YAML properties shown in the YAML editor.

Step 2: Save your workflow with a commit

In this step, you save your changes. Because workflows are stored as `.yaml` files in your repository, you save your changes with commits.

To commit your workflow changes

1. (Optional) Choose **Validate** to make sure the workflow's YAML code is valid.
2. Choose **Commit**.
3. In **Workflow file name**, enter a name for your workflow configuration file, like **`my-first-workflow`**.
4. In **Commit message**, enter a message to identify your commit, like **`create my-first-workflow.yaml`**.
5. In **Repository**, choose the repository you want to save the workflow in (`codecatalyst-repository`).
6. In **Branch name**, choose the branch you want to save the workflow in (`main`).
7. Choose **Commit**.

Your new workflow appears in the list of workflows. It might take several moments to appear.

Because workflows are saved with commits, and because the workflow has a code push trigger configured, saving the workflow starts a workflow run automatically.

Step 3: View run results

In this step, you navigate to the run that was started from your commit and view the results.

To view run results

1. Choose the name of your workflow, for example, `Workflow_fe47`.

   A workflow diagram showing the label of your source repository (**WorkflowSource**) and the build action (for example, **Build_f0**).
2. In the workflow run diagram, choose the build action (for example, **Build_f0**).
3. Review the contents of the **Logs**, **Reports**, **Configuration**, and **Variables** tabs. These tabs show you the results of your build action.

   For more information, see Viewing the results of a build action.

(Optional) Step 4: Clean up

In this step, you clean up the resources that you created in this tutorial.

To delete resources

- If you created a new project for this tutorial, delete it. For instructions, see Deleting a project. Deleting the project also deletes the source repository and workflow.

Building with workflows

Using CodeCatalyst workflows, you can build applications and other resources.

Topics

How do I build an application?

To build an application or resource in CodeCatalyst, you first create a workflow, and then specify a build action inside it.

A *build action* is a workflow building block that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy.

You add a build action to your workflow using the CodeCatalyst console's visual editor or YAML editor.

The high-level steps to build an application or resource are as follows.

To build an application (high-level tasks)

1. In CodeCatalyst, you **add source code** for an application you want to build. For more information, see Storing source code in repositories for a project in CodeCatalyst.
2. In CodeCatalyst, you **create a workflow**. The workflow is where you define how to build, test, and deploy your application. For more information, see Getting started with workflows.

3. (Optional) In the workflow, you **add a trigger** that indicates the events that will cause the workflow to start automatically. For more information, see Starting a workflow run automatically with triggers
4. In the workflow, you add a **build action** that compiles and packages your application or resource source code. Optionally, you can also have the build action run unit tests, generate reports, and deploy your application if you don't want to use a test or deploy action for these purposes. For more on the test and deploy actions, see Adding the build action.
5. (Optional) In the workflow, you **add a test action** and a **deploy action** to test and deploy your application or resource. You can choose from several pre-configured actions to deploy your application to different targets, such as Amazon ECS. For more information, see Testing with workflows, and Deploying with workflows.
6. You **start the workflow** either manually or automatically through a trigger. The workflow runs the build, test, and deploy actions in sequence to build, test, and deploy your application and resources to the target. For more information, see Starting a workflow run manually.

Benefits of the build action

Using the build action within a workflow has the following benefits:

- **Fully managed** – The build action eliminates the need to set up, patch, update, and manage your own build servers.
- **On demand** – The build action scales on demand to meet your build needs. You pay only for the number of build minutes you consume. For more information, see Configuring the compute and runtime environment Docker images for a workflow.
- **Out of the box** – CodeCatalyst includes prepackaged runtime environment Docker images that are used to run all your workflow actions, including build actions. These images come preconfigured with useful tools for building applications such as the AWS CLI and Node.js. You can configure CodeCatalyst to use a build image that you supply from a public or private registry. For more information, see Specifying runtime environment Docker images.

Alternatives to the build action

If you're using a build action to deploy your application, consider using a CodeCatalyst *deploy action* instead. Deploy actions perform behind-the-scenes configuration that you would otherwise

have to write manually if you're using a build action. For more information on the available deploy actions, see List of deploy actions.

You can also use AWS CodeBuild to build your applications. For more information, see What is CodeBuild?.

Adding the build action

Use the following procedure to add a build action to your CodeCatalyst workflow.

Visual

To add a build action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
4. Choose **Edit**.
5. Choose **Visual**.
6. Choose **Actions**.
7. In **Actions**, choose **Build**.
8. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the Build and test action YAML definition. This reference provides detailed information on each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add a build action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

4. Choose **Edit**.
5. Choose **YAML**.
6. Choose **Actions**.
7. In **Actions**, choose **Build**.
8. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the Build and test action YAML definition.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

Build action definition

The build action is defined as a set of YAML properties inside your workflow definition file. For information on these properties, see Build and test action YAML definition in the Workflow YAML definition.

Viewing the results of a build action

Use the following instructions to view the results of a build action, including the generated logs, reports, and variables.

To view the results of a build action

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
3. In the workflow diagram, choose the name of your build action, for example, **Build**.
4. To view the logs for the build run, choose **Logs**. The logs for the various build phases are displayed. You can expand or collapse the logs as needed.
5. To view the test reports produced by the build action, choose **Reports**, or in the navigation pane, choose **Reports**. For more information, see Quality report types.
6. To view the configuration used for the build action, choose **Configuration**. For more information, see Adding the build action.
7. To view the variables used by the build action, choose **Variables**. For more information, see Configuring and using variables in a workflow.

Tutorial: Upload artifacts to Amazon S3

In this tutorial, you learn how to upload artifacts to an Amazon S3 bucket using a CodeCatalyst workflow that includes a couple of build actions. These actions run in series when the workflow starts. The first build action generates two files, `Hello.txt` and `Goodbye.txt`, and bundles them into a build artifact. The second build action uploads the artifact to Amazon S3. You'll configure the workflow to run every time you push a commit to your source repository.

Topics

Prerequisites

Before you begin, you need the following:

- You need a CodeCatalyst **space** with a connected AWS account. For more information, see Creating a space.
- In your space, you need an empty, **Start from scratch** CodeCatalyst **project** called:

  ```
  codecatalyst-artifact-project
  ```

  For more information, see Creating an empty project in Amazon CodeCatalyst.
- In your project, you need a CodeCatalyst **environment** called:

  ```
  codecatalyst-artifact-environment
  ```

  Configure this environment as follows:
  - Choose any type, such as **Development**.
  - Connect your AWS account to it.

- For the **Default IAM role**, choose any role. You'll specify a different role later.

For more information, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.

Step 1: Create an AWS role

In this step, you create an AWS IAM role which you will later assign to the build action in your workflow. This role grants the CodeCatalyst build action permission to access your AWS account and write to Amazon S3 where your artifact will be stored. The role is called the **Build role**.

> **Note**
>
> If you already have a build role that you created for another tutorial, you can use it for this tutorial too. Just make sure it has the permissions and trust policy shown in the following procedure.

For more information on IAM roles, see IAM roles in the *AWS AWS Identity and Access Management User Guide*.

To create a build role

1. Create a policy for the role, as follows:

    a. Sign in to AWS.
    b. Open the IAM console at https://console.aws.amazon.com/iam/.
    c. In the navigation pane, choose **Policies**.
    d. Choose **Create policy**.
    e. Choose the **JSON** tab.
    f. Delete the existing code.
    g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
```

```
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-s3-build-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the build role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. Choose **Next**.

f. In **Permissions policies**, search for `codecatalyst-s3-build-policy` and select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-s3-build-role
```

i. For **Role description**, enter:

```
CodeCatalyst build role
```

j. Choose **Create role**.

You have now created a build role with a trust policy and permissions policy.

Step 2: Create an Amazon S3 bucket

In this step, you create an Amazon S3 bucket where the `Hello.txt` and `Goodbye.txt` artifacts will be uploaded.

To create an Amazon S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the main pane, choose **Create bucket**.
3. For **Bucket name**, enter:

```
codecatalyst-artifact-bucket
```

4. For **AWS Region**, choose a Region. This tutorial assumes you chose **US West (Oregon) us-west-2**. For information about Regions supported by Amazon S3, see Amazon Simple Storage Service endpoints and quotas in the *AWS General Reference*.
5. At the bottom of the page, choose **Create bucket**.
6. Copy the name of the bucket you just created, for example:

```
codecatalyst-artifact-bucket
```

You have now created a bucket called **`codecatalyst-artifact-bucket`** in the US West (Oregon) us-west-2 Region.

Step 3: Create a source repository

In this step, you create a source repository in CodeCatalyst. This repository is used to store the tutorial's workflow definition file.

For more information on source repositories, see Creating a source repository.

To create a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, `codecatalyst-artifact-project`.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose **Add repository**, and then choose **Create repository**.
5. In **Repository name**, enter:

```
codecatalyst-artifact-source-repository
```

6. Choose **Create**.

You have now created a repository called `codecatalyst-artifact-source-repository`.

Step 4: Create a workflow

In this step, you create a workflow that consists of the following building blocks that run sequentially:

- A trigger – This trigger starts the workflow run automatically when you push a change to your source repository. For more information on triggers, see Starting a workflow run automatically with triggers.
- A build action called `GenerateFiles` – On trigger, the `GenerateFiles` action creates two files, `Hello.txt` and `Goodbye.txt`, and packages them into an output artifact called `codecatalystArtifact`.
- Another build action called `Upload` – On completion of the `GenerateFiles` action, the `Upload` action runs the AWS CLI command `aws s3 sync` to upload the files in the `codecatalystArtifact` and in your source repository to your Amazon S3 bucket. The AWS CLI comes pre-installed and pre-configured on the CodeCatalyst compute platform, so you don't need to install or configure it.

  For more information on the pre-packaged software on the CodeCatalyst compute platform, see Specifying runtime environment Docker images. For more information on the AWS CLI's `aws s3 sync` command, see sync in the *AWS CLI Command Reference*.

For more information on the build action, see Building with workflows.

To create a workflow

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose **Create workflow**.
3. Delete the YAML sample code.
4. Add the following YAML code:

> **Note**
>
> In the YAML code that follows, you can omit the `Connections:` section if you want. If you omit this section, you must ensure that the role specified in the **Default IAM role** field in your environment includes the permissions and trust policies described in Step

1: Create an AWS role. For more information about setting up an environment with a default IAM role, see Creating an environment.

```
Name: codecatalyst-artifact-workflow
SchemaVersion: 1.0

Triggers:
  - Type: Push
    Branches:
      - main
Actions:
  GenerateFiles:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        # Create the output files.
        - Run: echo "Hello, World!" > "Hello.txt"
        - Run: echo "Goodbye!" > "Goodbye.txt"
    Outputs:
      Artifacts:
        - Name: codecatalystArtifact
          Files:
            - "**/*"
  Upload:
    Identifier: aws/build@v1
    DependsOn:
      - GenerateFiles
    Environment:
      Name: codecatalyst-artifact-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-s3-build-role
    Inputs:
      Artifacts:
        - codecatalystArtifact
    Configuration:
      Steps:
        # Upload the output artifact to the S3 bucket.
        - Run: aws s3 sync . s3://codecatalyst-artifact-bucket
```

In the code above, replace:

- *codecatalyst-artifact-environment* with the name of the environment you created in Prerequisites.
- *codecatalyst-account-connection* with the name of the account connection you created in Prerequisites.
- *codecatalyst-s3-build-role* with the name of the build role that you created in Step 1: Create an AWS role.
- *codecatalyst-artifact-bucket* with the name of the Amazon S3 you created in Step 2: Create an Amazon S3 bucket.

For information about the properties in this file, see the Build and test action YAML definition.

5. (Optional) Choose **Validate** to make sure the YAML code is valid before committing.
6. Choose **Commit**.
7. On the **Commit workflow** dialog box, enter the following:

   a. For **Workflow file name**, leave the default, `codecatalyst-artifact-workflow`.

   b. For **Commit message**, enter:

   ```
   add initial workflow file
   ```

   c. For **Repository**, choose **codecatalyst-artifact-source-repository**.

   d. For **Branch name**, choose **main**.

   e. Choose **Commit**.

   You have now created a workflow. A workflow run starts automatically because of the trigger defined at the top of the workflow. Specifically, when you committed (and pushed) the `codecatalyst-artifact-workflow.yaml` file to your source repository, the trigger started the workflow run.

To view the workflow run in progress

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the workflow you just created: `codecatalyst-artifact-workflow`.
3. Choose **GenerateFiles** to see the first build action progress.

4. Choose **Upload** to see the second build action progress.
5. When the **Upload** action finishes, do the following:

   - If the workflow run succeeded, go to the next procedure.
   - If the workflow run failed, choose **Logs** to troubleshoot the issue.

Step 5: Verify the results

After the workflow runs, go to the Amazon S3 service and look in your *codecatalyst-artifact-bucket* bucket. It should now include the following files and folders:

```
.
|— .aws/
|— .git/
|Goodbye.txt
|Hello.txt
|REAME.md
```

The `Goodbye.txt` and `Hello.txt` files were uploaded because they were part of the `codecatalystArtifact` artifact. The `.aws/`, `.git/`, and `README.md` files were uploaded because they were in your source repository.

Clean up

Clean up in CodeCatalyst and AWS to avoid being charged for these services.

To clean up in CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Delete the `codecatalyst-artifact-source-repository` source repository.
3. Delete the `codecatalyst-artifact-workflow` workflow.

To clean up in AWS

1. Clean up in Amazon S3, as follows:

   a. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   b. Delete the files in the `codecatalyst-artifact-bucket` bucket.

c. Delete the `codecatalyst-artifact-bucket` bucket.

2. Clean up in IAM, as follows:

   a. Open the IAM console at https://console.aws.amazon.com/iam/.

   b. Delete the `codecatalyst-s3-build-policy`.

   c. Delete the `codecatalyst-s3-build-role`.

Build and test action YAML definition

The following is the YAML definition of the build and test actions. There is one reference for two actions because their YAML properties are very similar.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

Choose a YAML property in the following code to see a description if it.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  action-name:
    Identifier: aws/build@v1 | aws/managed-test@v1
    DependsOn:
      - dependent-action-name-1
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
```

```
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Caching:
      FileCaching:
        key-name-1:
          Path: file1.txt
          RestoreKeys:
            - restore-key-1
    Inputs:
      Sources:
        - source-name-1
        - source-name-2
      Artifacts:
        - artifact-name
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Outputs:
      Artifacts:
        - Name: output-artifact-1
          Files:
            - build-output/artifact-1.jar
            - "build-output/build*"
        - Name: output-artifact-2
          Files:
            - build-output/artifact-2.1.jar
            - build-output/artifact-2.2.jar
      Variables:
        - variable-name-1
        - variable-name-2
      AutoDiscoverReports:
        Enabled: true | false
        ReportNamePrefix: AutoDiscovered
        IncludePaths:
          - "**/*"
        ExcludePaths:
          - node_modules/cdk/junit.xml
        SuccessCriteria:
          PassRate: percent
```

```
        LineCoverage: percent
        BranchCoverage: percent
        Vulnerabilities:
          Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
          Number: whole-number
        StaticAnalysisBug:
          Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
          Number: whole-number
        StaticAnalysisSecurity:
          Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
          Number: whole-number
        StaticAnalysisQuality:
          Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
          Number: whole-number
        StaticAnalysisFinding:
          Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
          Number: whole-number
    Reports:
      report-name-1:
        Format: format
        IncludePaths:
          - "*.xml"
        ExcludePaths:
          - report2.xml
          - report3.xml
        SuccessCriteria:
          PassRate: percent
          LineCoverage: percent
          BranchCoverage: percent
          Vulnerabilities:
            Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
            Number: whole-number
          StaticAnalysisBug:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
          StaticAnalysisSecurity:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
          StaticAnalysisQuality:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
          StaticAnalysisFinding:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
```

```
    Configuration:
      Container:
        Registry: registry
        Image: image
      Steps:
        - Run: "step 1"
        - Run: "step 2"
      Packages:
        NpmConfiguration:
          PackageRegistries:
            - PackagesRepository: package-repository
              Scopes:
                - "@scope"
        ExportAuthorizationToken: true | false
```

action-name

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Corresponding UI: Configuration tab/**Action name**

Identifier

(*action-name*/**Identifier**)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Use `aws/build@v1` for build actions.

Use `aws/managed-test@v1` for test actions.

Corresponding UI: Workflow diagram/Action-name/*aws/build@v1|aws/managed-test@v1* label

DependsOn

(*action-name*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*action-name*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*action-name*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/**Compute type**

Fleet

(*action-name*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/**Compute fleet**

Timeout

(*action-name*/**Timeout**)

(Optional)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Environment

(*action-name*/**Environment**)

(Optional)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the

default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*action-name*/Environment/**Name**)

(Optional)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*action-name*/Environment/**Connections**)

(Optional)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**

Name

(*action-name*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**

Role

(*action-name*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that this action uses in order to access and operate in AWS services such as Amazon S3 and Amazon ECR. Make sure this role is added to your AWS account connection in your space. To add an IAM role to an account connection, see Adding IAM roles to account connections.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role with this action. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

> **Warning**
>
> Limit the permissions to those required by the build and test actions. Using a role with broader permissions might pose a security risk.

Corresponding UI: Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**

Caching

(*action-name*/**Caching**)

(Optional)

A section where you can specify a cache to save on-disk files and restore them from that cache in subsequent workflow runs.

For more information about file caching, see Caching files between workflow runs.

Corresponding UI: Configuration tab/**File caching - optional**

FileCaching

(*action-name*/Caching/**FileCaching**)

(Optional)

A section that specifies the configuration for a sequence of caches.

Corresponding UI: Configuration tab/File caching - optional/**Add cache**

key-name-1

(*action-name*/Caching/FileCaching/***key-name-1***)

(Optional)

Specify the name of your primary cache property name. Cache property names must be unique within your workflow. Each action can have up to five entries in `FileCaching`.

Corresponding UI: Configuration tab/File caching - optional/Add cache/**Key**

Path

(*action-name*/Caching/FileCaching/*key-name-1*/**Path**)

(Optional)

Specify the associated path for your cache.

Corresponding UI: Configuration tab/File caching - optional/Add cache/**Path**

RestoreKeys

(*action-name*/Caching/FileCaching/*key-name-1*/**RestoreKeys**)

(Optional)

Specify the restore key to use as a fallback when the primary cache property can't be found. Restore key names must be unique within your workflow. Each cache can have up to five entries in `RestoreKeys`.

Corresponding UI: Configuration tab/File caching - optional/Add cache/**Restore keys - optional**

Inputs

(*action-name*/**Inputs**)

(Optional)

The `Inputs` section defines the data that an action needs during a workflow run.

> **Note**
>
> A maximum of four inputs (one source and three artifacts) are allowed per build action or test action. Variables do not count towards this total.

If you need to refer to files residing in different inputs (say a source and an artifact), the source input is the primary input, and the artifact is the secondary input. References to files in secondary

inputs take a special prefix to distiguish them from the primary. For details, see Example: Referencing files in multiple artifacts.

Corresponding UI: **Inputs** tab

Sources

(*action-name*/Inputs/**Sources**)

(Optional)

Specify the labels that represent the source repositories that will be needed by the action. Currently, the only supported label is `WorkflowSource`, which represents the source repository where your workflow definition file is stored.

If you omit a source, then you must specify at least one input artifact under *action-name*/`Inputs/Artifacts`.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: *none*

Artifacts - input

(*action-name*/Inputs/**Artifacts**)

(Optional)

Specify artifacts from previous actions that you want to provide as input to this action. These artifacts must already be defined as output artifacts in previous actions.

If you do not specify any input artifacts, then you must specify at least one source repository under *action-name*/`Inputs/Sources`.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

> **Note**
>
> If the **Artifacts - optional** drop-down list is unavailable (visual editor), or if you get errors in when you validate your YAML (YAML editor), it might be because the action only supports one input. In this case, try removing the source input.

Corresponding UI: Inputs tab/**Artifacts - optional**

Variables - input

(*action-name*/Inputs/**Variables**)

(Optional)

Specify a sequence of name/value pairs that define the input variables that you want to make available to the action. Variable names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in variable names.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Inputs tab/**Variables - optional**

Outputs

(*action-name*/**Outputs**)

(Optional)

Defines the data that is output by the action during a workflow run.

Corresponding UI: **Outputs** tab

Artifacts - output

(*action-name*/Outputs/**Artifacts**)

(Optional)

Specify the name of an artifact generated by the action. Artifact names must be unique within a workflow, and are limited to alphanumeric characters (a-z, A-Z, 0-9) and underscores (_). Spaces, hyphens (-), and other special characters are not allowed. You cannot use quotation marks to enable spaces, hyphens, and other special characters in output artifact names.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/**Artifacts**

Name

(*action-name*/Outputs/Artifacts/**Name**)

(Required if Artifacts - output is included)

Specify the name of an artifact generated by the action. Artifact names must be unique within a workflow, and are limited to alphanumeric characters (a-z, A-Z, 0-9) and underscores (_). Spaces, hyphens (-), and other special characters are not allowed. You cannot use quotation marks to enable spaces, hyphens, and other special characters in output artifact names.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/Artifacts/New output/**Build artifact name**

Files

(*action-name*/Outputs/Artifacts/**Files**)

(Required if Artifacts - output is included)

Specify the files that CodeCatalyst includes in the artifact that is output by the action. These files are generated by the workflow action when it runs, and are also available in your source repository. File paths can reside in a source repository or an artifact from a previous action, and are relative to the source repository or artifact root. You can use glob patterns to specify paths. Examples:

- To specify a single file that is in the root of your build location or source repository location, use `my-file.jar`.
- To specify a single file in a subdirectory, use `directory/my-file.jar` or `directory/subdirectory/my-file.jar`.
- To specify all files, use `"**/*"`. The `**` glob pattern indicates to match any number of subdirectories.
- To specify all files and directories in a directory named `directory`, use `"directory/**/*"`. The `**` glob pattern indicates to match any number of subdirectories.
- To specify all files in a directory named `directory`, but not any of its subdirectories, use `"directory/*"`.

> **Note**
>
> If your file path includes one or more asterisks (*) or other special character, enclose the path with double quotation marks (" "). For more information about special characters, see Syntax guidelines and conventions.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

> **Note**
>
> You may need to add a prefix to the file path to indicate which artifact or source to find it in. For more information, see Referencing files in a source repository and Referencing files in an artifact.

Corresponding UI: Outputs tab/Artifacts/New output/**Files produced by build**

Variables - output

(*action-name*/Outputs/**Variables**)

(Optional)

Specify the variables that you want the action to export so that they are available for use by subsequent actions.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Outputs tab/Variables/**Add variable**

variable-name-1

(*action-name*/Outputs/Variables/*variable-name-1*)

(Optional)

Specify the name of a variable that you want the action to export. This variable must already be defined in the `Inputs` or `Steps` section of the same action.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Outputs tab/Variables/Add variable/**Name**

AutoDiscoverReports

(*action-name*/Outputs/**AutoDiscoverReports**)

(Optional)

Defines the configuration for the auto-discovery feature.

When you enable auto-discovery, CodeCatalyst searches all `Inputs` passed into the action as well as all files generated by the action itself, looking for test, code coverage, and software composition analysis (SCA) reports. For each report that is found, CodeCatalyst transforms it into a CodeCatalyst report. A *CodeCatalyst report* is a report that is fully integrated into the CodeCatalyst service and can be viewed and manipulated through the CodeCatalyst console.

> **Note**
>
> By default, the auto-discover feature inspects all files. You can limit which files are inspected using the IncludePaths or ExcludePaths properties.

Corresponding UI: Outputs tab/Reports/**Auto-discover reports**

Enabled

(*action-name*/Outputs/AutoDiscoverReports/**Enabled**)

(Optional)

Enable or disable the auto-discovery feature.

Valid values are `true` or `false`.

If `Enabled` is omitted, the default is `true`.

Corresponding UI: Outputs tab/Reports/**Auto-discover reports**

ReportNamePrefix

(*action-name*/Outputs/AutoDiscoverReports/**ReportNamePrefix**)

(Required if AutoDiscoverReports is included and enabled)

Specify a prefix that CodeCatalyst prepends to all the reports it finds in order to name their associated CodeCatalyst reports. For example, if you specify a prefix of `AutoDiscovered`, and CodeCatalyst auto-discovers two test reports, `TestSuiteOne.xml` and `TestSuiteTwo.xml`, then the associated CodeCatalyst reports will be named `AutoDiscoveredTestSuiteOne` and `AutoDiscoveredTestSuiteTwo`.

Corresponding UI: Outputs tab/Reports/**Prefix name**

IncludePaths

(*action-name*/Outputs/AutoDiscoverReports/**IncludePaths**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/**IncludePaths**)

(Required if AutoDiscoverReports is included and enabled, or if Reports is included)

Specify the files and file paths that CodeCatalyst includes when searching for raw reports. For example, if you specify `"/test/report/*"`, CodeCatalyst searches the entire build image used by the action looking for the `/test/report/*` directory. When it finds that directory, CodeCatalyst then looks for reports in that directory.

> **Note**
>
> If your file path includes one or more asterisks (*) or other special characters, enclose the path with double quotation marks (""). For more information about special characters, see Syntax guidelines and conventions.

If this property is omitted, the default is `"**/*"`, meaning the search includes all files at all paths.

> **Note**
>
> For manually configured reports, `IncludePaths` must be a glob pattern that matches a single file.

Corresponding UI:

- Outputs tab/Reports/Auto-discover reports/Include/exclude paths/**Include paths**
- Outputs tab/Reports/Manually configure reports/*report-name-1*/Include/exclude paths/**Include paths**

ExcludePaths

(*action-name*/Outputs/AutoDiscoverReports/**ExcludePaths**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/**ExcludePaths**)

(Optional)

Specify the files and file paths that CodeCatalyst excludes when searching for raw reports. For example, if you specify `"/test/my-reports/**/*"`, CodeCatalyst will not search for files in the `/test/my-reports/` directory. To ignore all files in a directory, use the `**/*` glob pattern.

> **Note**
>
> If your file path includes one or more asterisks (*) or other special characters, enclose the path with double quotation marks (""). For more information about special characters, see Syntax guidelines and conventions.

Corresponding UI:

- Outputs tab/Reports/Auto-discover reports/Include/exclude paths/**Exclude paths**
- Outputs tab/Reports/Manually configure reports/*report-name-1*/Include/exclude paths/**Exclude paths**

SuccessCriteria

(*action-name*/Outputs/AutoDiscoverReports/**SuccessCriteria**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/**SuccessCriteria**)

(Optional)

Specify the success criteria for the test, code coverage, software composition analysis (SCA), and static analysis (SA) reports.

For more information, see Configuring success criteria for reports.

Corresponding UI: Output tab/Reports/**Success criteria**

PassRate

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**PassRate**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**PassRate**)

(Optional)

Specify the percentage of tests in a test report that must pass for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. The pass rate criteria are applied only to test reports. For more information about test reports, see Test reports.

Corresponding UI: Output tab/Reports/Success criteria/**Pass rate**

LineCoverage

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**LineCoverage**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**LineCoverage**)

(Optional)

Specify the percentage of lines in a code coverage report that must be covered for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. Line coverage criteria are applied only to code coverage reports. For more information about code coverage reports, see Code coverage reports.

Corresponding UI: Output tab/Reports/Success criteria/**Line coverage**

BranchCoverage

(*`action-name`*/Outputs/AutoDiscoverReports/SuccessCriteria/**BranchCoverage**)

Or

(*`action-name`*/Outputs/Reports/*`report-name-1`*/SuccessCriteria/**BranchCoverage**)

(Optional)

Specify the percentage of branches in a code coverage report that must be covered for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. Branch coverage criteria are applied only to code coverage reports. For more information about code coverage reports, see Code coverage reports.

Corresponding UI: Output tab/Reports/Success criteria/**Branch coverage**

Vulnerabilities

(*`action-name`*/Outputs/AutoDiscoverReports/SuccessCriteria/**Vulnerabilities**)

Or

(*`action-name`*/Outputs/Reports/*`report-name-1`*/SuccessCriteria/**Vulnerabilities**)

(Optional)

Specify the maximum number and severity of vulnerabilities permitted in the SCA report for the associated CodeCatalyst report to be marked as passed. To specify vulnerabilities, you must specify:

- The minimum severity of the vulnerabilities you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` vulnerabilities will be tallied.

- The maximum number of vulnerabilities of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Vulnerabilities criteria are applied only to SCA reports. For more information about SCA reports, see Software composition analysis reports.

To specify the minimum severity, use the `Severity` property. To specify the maximum number of vulnerabilities, use the `Number` property.

Corresponding UI: Output tab/Reports/Success criteria/**Vulnerabilities**

StaticAnalysisBug

(*`action-name`*/Outputs/AutoDiscoverReports/SuccessCriteria/**StaticAnalysisBug**)

Or

(*`action-name`*/Outputs/Reports/*`report-name-1`*/SuccessCriteria/**StaticAnalysisBug**)

(Optional)

Specify the maximum number and severity of bugs permitted in the SA report for the associated CodeCatalyst report to be marked as passed. To specify bugs, you must specify:

- The minimum severity of the bugs you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` bugs will be tallied.
- The maximum number of bugs of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Bugs criteria are applied only to PyLint and ESLint SA reports. For more information about SA reports, see Static analysis reports.

To specify the minimum severity, use the `Severity` property. To specify the maximum number of vulnerabilities, use the `Number` property.

Corresponding UI: Output tab/Reports/Success criteria/**Bugs**

StaticAnalysisSecurity

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**StaticAnalysisSecurity**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**StaticAnalysisSecurity**)

(Optional)

Specify the maximum number and severity of security vulnerabilities permitted in the SA report for the associated CodeCatalyst report to be marked as passed. To specify security vulnerabilities, you must specify:

- The minimum severity of the security vulnerabilities you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` security vulnerabilities will be tallied.
- The maximum number of security vulnerabilities of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Security vulnerabilities criteria are applied only to PyLint and ESLint SA reports. For more information about SA reports, see Static analysis reports.

To specify the minimum severity, use the `Severity` property. To specify the maximum number of vulnerabilities, use the `Number` property.

Corresponding UI: Output tab/Reports/Success criteria/**Security vulnerabilities**

StaticAnalysisQuality

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**StaticAnalysisQuality**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**StaticAnalysisQuality**)

(Optional)

Specify the maximum number and severity of quality issues permitted in the SA report for the associated CodeCatalyst report to be marked as passed. To specify quality issues, you must specify:

- The minimum severity of the quality issues you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` quality issues will be tallied.
- The maximum number of quality issues of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Quality issues criteria are applied only to PyLint and ESLint SA reports. For more information about SA reports, see Static analysis reports.

To specify the minimum severity, use the `Severity` property. To specify the maximum number of vulnerabilities, use the `Number` property.

Corresponding UI: Output tab/Reports/Success criteria/**Quality issues**

StaticAnalysisFinding

(*`action-name`*/Outputs/AutoDiscoverReports/SuccessCriteria/**StaticAnalysisFinding**)

Or

(*`action-name`*/Outputs/Reports/*`report-name-1`*/SuccessCriteria/**StaticAnalysisFinding**)

(Optional)

Specify the maximum number and severity of findings permitted in the SA report for the associated CodeCatalyst report to be marked as passed. To specify findings, you must specify:

- The minimum severity of the findings you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` findings will be tallied.
- The maximum number of findings of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Findings are applied only to SARIF SA reports. For more information about SA reports, see Static analysis reports.

To specify the minimum severity, use the `Severity` property. To specify the maximum number of vulnerabilities, use the `Number` property.

Corresponding UI: Output tab/Reports/Success criteria/**Findings**

Reports

(*`action-name`*/Outputs/**Reports**)

(Optional)

A section that specifies the configuration for test reports.

Corresponding UI: Outputs tab/**Reports**

report-name-1

(*`action-name`*/Outputs/Reports/**report-name-1**)

(Required if Reports is included)

The name you want to give to the CodeCatalyst report that will be generated from your raw reports.

Corresponding UI: Outputs tab/Reports/Manually configure reports/**Report name**

Format

(*`action-name`*/Outputs/Reports/*`report-name-1`*/**Format**)

(Required if Reports is included)

Specify the file format that you're using for your reports. Possible values are as follows.

- For test reports:
  - For Cucumber JSON, specify **Cucumber** (visual editor) or `CUCUMBERJSON` (YAML editor).
  - For JUnit XML, specify **JUnit** (visual editor) or `JUNITXML` (YAML editor).
  - For NUnit XML, specify **NUnit** (visual editor) or `NUNITXML` (YAML editor).
  - For NUnit 3 XML, specify **NUnit3** (visual editor) or `NUNIT3XML` (YAML editor).

  - For Visual Studio TRX, specify **Visual Studio TRX** (visual editor) or `VISUALSTUDIOTRX` (YAML editor).
  - For TestNG XML, specify **TestNG** (visual editor) or `TESTNGXML` (YAML editor).
- For code coverage reports:
  - For Clover XML, specify **Clover** (visual editor) or `CLOVERXML` (YAML editor).
  - For Cobertura XML, specify **Cobertura** (visual editor) or `COBERTURAXML` (YAML editor).
  - For JaCoCo XML, specify **JaCoCo** (visual editor) or `JACOCOXML` (YAML editor).
  - For SimpleCov JSON generated by simplecov, not simplecov-json, specify **Simplecov** (visual editor) or `SIMPLECOV` (YAML editor).
- For software composition analysis (SCA) reports:
  - For SARIF, specify **SARIF** (visual editor) or `SARIFSCA` (YAML editor).

Corresponding UI: Outputs tab/Reports/Manually configure reports/Add/configure reports/*`report-name-1`*/**Report type** and **Report format**

Configuration

(*`action-name`*/**Configuration**)

(Required) A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

Container

(*`action-name`*/Configuration/**Container**)

(Optional)

Specify the Docker image, or *container*, that the action uses to complete its processing. You can specify one of the active images that come with CodeCatalyst, or you can use your own image. If you choose to use your own image, it can reside in Amazon ECR, Docker Hub, or another registry. If you don't specify a Docker image, the action uses one of the active images for its processing. For information about which active image is used by default, see Active images.

For more information about specifying your own Docker image, see Assigning a custom runtime environment Docker image to an action.

Corresponding UI: **Runtime environment Docker image - optional**

Registry

(*action-name*/Configuration/Container/**Registry**)

(Required if `Container` is included)

Specify the registry where your image is stored. Valid values include:

- `CODECATALYST` (YAML editor)

  The image is stored in the CodeCatalyst registry.
- **Docker Hub** (visual editor) or `DockerHub` (YAML editor)

  The image is stored in the Docker Hub image registry.
- **Other registry** (visual editor) or `Other` (YAML editor)

  The image is stored in a custom image registry. Any publicly available registry can be used.
- **Amazon Elastic Container Registry** (visual editor) or `ECR` (YAML editor)

  The image is stored in an Amazon Elastic Container Registry image repository. To use an image in an Amazon ECR repository, this action needs access to Amazon ECR. To enable this access, you must create an IAM role that includes the following permissions and custom trust policy. (You can modify an existing role to include the permissions and policy, if you want.)

  The IAM role must include the following permissions in its role policy:
  - `ecr:BatchCheckLayerAvailability`
  - `ecr:BatchGetImage`
  - `ecr:GetAuthorizationToken`
  - `ecr:GetDownloadUrlForLayer`

  The IAM role must include the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
```

```
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

For more information about creating IAM roles, see Creating a role using custom trust policies (console) in the *IAM User Guide*.

Once you have created the role, you must assign it to the action through an environment. For more information, see Associating an environment with a workflow action.

Corresponding UI: **Amazon Elastic Container Registry**, **Docker Hub**, and **Other registry** options

Image

(*action-name*/Configuration/Container/**Image**)

(Required if `Container` is included)

Specify one of the following:

- If you are using a `CODECATALYST` registry, set the image to one of the of the following active images:
  - `CodeCatalystLinux_x86_64:2024_03`
  - `CodeCatalystLinux_x86_64:2022_11`
  - `CodeCatalystLinux_Arm64:2024_03`
  - `CodeCatalystLinux_Arm64:2022_11`
  - `CodeCatalystLinuxLambda_x86_64:2024_03`
  - `CodeCatalystLinuxLambda_x86_64:2022_11`
  - `CodeCatalystLinuxLambda_Arm64:2024_03`
  - `CodeCatalystLinuxLambda_Arm64:2022_11`
  - `CodeCatalystWindows_x86_64:2022_11`

- If you are using a Docker Hub registry, set the image to the Docker Hub image name and optional tag.

  Example: `postgres:latest`
- If you are using an Amazon ECR registry, set the image to the Amazon ECR registry URI.

  Example: `111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo`
- If you are using a custom registry, set the image to the value expected by the custom registry.

Corresponding UI: **Runtime environment docker image** (if the registry is `CODECATALYST`), **Docker Hub image** (if the registry is **Docker Hub**), **ECR image URL** (if the registry is **Amazon Elastic Container Registry**), and **Image URL** (if the registry is **Other registry**).

Steps

(*`action-name`*/Configuration/**Steps**)

(Required)

Specify the shell commands that you want to run during the action to install, configure, and run your build tools.

Here is an example of how to build an npm project:

```
Steps:
  - Run: npm install
  - Run: npm run build
```

Here is an example of how to specify file paths:

```
Steps:
  - Run: cd $ACTION_BUILD_SOURCE_PATH_WorkflowSource/app  && cat file2.txt
  - Run: cd $ACTION_BUILD_SOURCE_PATH_MyBuildArtifact/build-output/  && cat file.txt
```

For more information about specifying file paths, see Referencing files in a source repository and Referencing files in an artifact.

Corresponding UI: Configuration tab/**Shell commands**

Packages

(*action-name*/Configuration/**Packages**)

(Optional)

A section where you can specify a package repository that the action uses to resolve dependencies. Packages allow you to securely store and share software packages used for application development.

For more information about packages, see Publish and share software packages in CodeCatalyst.

Corresponding UI: Configuration tab/**Packages**

NpmConfiguration

(*action-name*/Configuration/Packages/**NpmConfiguration**)

(Required if Packages is included)

A section which defines the configuration for the npm package format. This configuration is used by an action during a workflow run.

For more information about the npm package configuration, see Using npm.

Corresponding UI: Configuration tab/Packages/Add configuration/**npm**

PackageRegistries

(*action-name*/Configuration/Packages/NpmConfiguration/**PackageRegistries**)

(Required if Packages is included)

A section where you can define the configuration properties of a sequence of package repositories.

Corresponding UI: Configuration tab/Packages/Add configuration/npm/**Add package repository**

PackagesRepository

(*action-name*/Configuration/Packages/NpmConfiguration/
PackageRegistries/**PackagesRepository**)

(Required if Packages is included)

Specify the name of your CodeCatalyst *package repository* that you want the action to use.

If you specify multiple default repositories, the last repository will take priority.

For more information about package repositories, see Package repositories.

Corresponding UI: Configuration tab/Packages/Add configuration/npm/Add package repository/**Package repository**

Scopes

(*action-name*/Configuration/Packages/NpmConfiguration/PackageRegistries/**Scopes**)

(Optional)

Specify a sequence of *scopes* that you want to define in your package registry. When defining scopes, the specified package repository is configured as the registry for all listed scopes. If a package with the scope is requested through the npm client, it will use that repository instead of the default. Each scope name must be prefixed with "@".

If you include overriding scopes, the last repository will take priority.

If `Scopes` is omitted, then the specified package repository is configured as the default registry for all packages used by the action.

For more information about scopes, see Package namespaces and Scoped packages.

Corresponding UI: Configuration tab/Packages/Add configuration/npm/Add package repository/**Scopes - optional**

ExportAuthorizationToken

(*action-name*/Configuration/Packages/**ExportAuthorizationToken**)

(Optional)

Enable or disable the export authorization token feature. If enabled, exported authorization tokens can be used to manually configure a package manager to authenticate with CodeCatalyst package repositories. You can use the token as an environment variable that can be referenced in your actions.

Valid values are `true` or `false`.

If `ExportAuthorizationToken` is omitted, the default is `false`.

For more information about the export authorization token, see Using authorization tokens in workflow actions.

Corresponding UI: Configuration tab/Packages/**Export authorization token**

Testing with workflows

In CodeCatalyst, you can run tests as part of different workflow actions, such as build and test. These workflow actions can all generate quality reports. A *test action* is a workflow action that produces test, code coverage, software composition analysis, and static analysis reports. These reports are displayed in the CodeCatalyst console.

Topics

Quality report types

The Amazon CodeCatalyst test action supports the following types of quality reports. For an example on how to format these reports in your YAML, see Quality reports YAML example.

Topics

- Software composition analysis reports
- Static analysis reports

Test reports

In CodeCatalyst, you can configure unit tests, integration tests, and system tests that run during builds. Then CodeCatalyst can create reports that contain the results of your tests.

You can use a test report to help troubleshoot problems with your tests. If you have many test reports from multiple builds, you can use your test reports to view failure rates to help you optimize your builds.

You can use the following test report file formats:

- Cucumber JSON (.json)
- JUnit XML (.xml)
- NUnit XML (.xml)
- NUnit3 XML (.xml)
- TestNG XML (.xml)
- Visual Studio TRX (.trx, .xml)

Code coverage reports

In CodeCatalyst, you can generate code coverage reports for your tests. CodeCatalyst provides the following code coverage metrics:

Line coverage

Measures how many statements your tests cover. A statement is a single instruction, not including comments.

```
line coverage = (total lines covered)/(total number of lines)
```

Branch coverage

Measures how many branches your tests cover out of every possible branch of a control structure such as an `if` or `case` statement.

```
branch coverage = (total branches covered)/(total number of branches)
```

The following code coverage report file formats are supported:

- JaCoCo XML (.xml)
- SimpleCov JSON (generated by simplecov, not simplecov-json, .json)
- Clover XML (version 3, .xml)
- Cobertura XML (.xml)
- LCOV (.info)

Software composition analysis reports

In CodeCatalyst, you can use software composition analysis (SCA) tools to analyze components of your application and check for known security vulnerabilities. You can discover and parse SARIF reports that detail vulnerabilities with varying severities and ways to fix them. Valid severity values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

The following SCA report file formats are supported:

- SARIF (.sarif, .json)

Static analysis reports

You can use static analysis (SA) reports to identify source-level code defects. In CodeCatalyst, you can generate SA reports to help resolve issues in your code before you deploy it. These issues include bugs, security vulnerabilities, quality issues, and other vulnerabilities. Valid severity values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, and `INFORMATIONAL`.

CodeCatalyst provides the following SA metrics:

Bugs

Identifies a number of possible bugs found in your source code. These bugs can include issues regarding memory safety. The following is an example of a bug.

```
// The while loop will inadvertently index into array x out-of-bounds
int x[64];
while (int n = 0; n <= 64; n++) {
```

```
  x[n] = 0;
}
```

Security vulnerabilities

Identifies a number of possible security vulnerabilities found in your source code. These security vulnerabilities can include issues such as storing your secret tokens in plaintext.

Quality issues

Identifies a number of possible quality issues found in your source code. These quality issues can include issues regarding style conventions. The following is an example of a quality issue.

```
// The function name doesn't adhere to the style convention of camelCase
int SUBTRACT(int x, int y) {
  return x-y
}
```

Other vulnerabilities

Identifies a number of possible other vulnerabilities found in your source code.

CodeCatalyst supports the following SA report file formats:

- PyLint (.py)
- ESLint (.js, .jsx, .ts, .tsx)
- SARIF (.sarif, .json)

Adding the test action

Use the following procedure to add a test action to your CodeCatalyst workflow.

Visual

To add a test action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

4. Choose **Edit**.
5. Choose **Visual**.
6. Choose **Actions**.
7. In **Actions**, choose **Test**.
8. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the Build and test action YAML definition. This reference provides detailed information on each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add a build action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
4. Choose **Edit**.
5. Choose **YAML**.
6. Choose **Actions**.
7. In **Actions**, choose **Test**.
8. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the Build and test action YAML definition.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

Test action definition

The test action is defined as a set of YAML properties inside your workflow definition file. For information about these properties, see Build and test action YAML definition in the Workflow YAML definition.

Viewing the results of a test action

Use the following instructions to view the results of a test action, including the generated logs, reports, and variables.

To view the results of a test action

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
3. In the workflow diagram, choose the name of your test action, for example, **Test**.
4. To view the logs generated by an action, choose **Logs**. The logs for the various action phases are displayed. You can expand or collapse the logs as needed.
5. To view the test reports produced by the test action, choose **Reports**, or in the navigation pane, choose **Reports**. For more information, see Quality report types.
6. To view the configuration used for the test action, choose **Configuration**. For more information, see Adding the test action.
7. To view the variables used by the test action, choose **Variables**. For more information, see Configuring and using variables in a workflow.

Skipping failed tests in an action

If your action has more than one test command, you might want to allow subsequent test commands in the action to run even if a previous command fails. For example, in the following commands, you may want `test2` to run always, even if `test1` fails.

```
Steps:
- Run: npm install
- Run: npm run test1
- Run: npm run test2
```

Normally, when a step returns an error, Amazon CodeCatalyst stops the workflow action and marks it as failed. You can allow the action steps to continue to run by redirecting the error output to `null`. You can do this by adding `2>/dev/null` to the command. With this modification, the preceding example would look like the following.

```
Steps:
```

```
- Run: npm install
- Run: npm run test1 2>/dev/null
- Run: npm run test2
```

In the second code snippet, the status of the `npm install` command will be honored, but any error returned by the `npm run test1` command will be ignored. As a result the `npm run test2` command is run. By doing this, you're able to view both reports at once regardless of whether an error occurs.

Integrating universal-test-runner into a test action

Test actions integrate with the open-source command line tool `universal-test-runner`. This tool provides advanced testing features, such as retrying one or more test cases from a test report. `universal-test-runner` uses the Test Execution Protocol to run your tests for any language in a given framework. `universal-test-runner` supports the following frameworks:

- Gradle
- Jest
- Maven
- pytest
- .NET

`universal-test-runner` is installed only on the curated images for test actions. If you configure a test action to use a custom Docker Hub or Amazon ECR, you must manually install `universal-test-runner` to enable advanced testing features. To do so, install Node.js (14 or higher) on the image, then install `universal-test-runner` through `npm` using the shell command `- Run: npm install -g @aws/universal-test-runner`. For more information about installing Node.js in your container through shell commands, see Installing and Updating Node Version Manager.

For more information about `universal-test-runner`, see What is universal-test-runner?

Visual

To use universal-test-runner in the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.

3. Choose the name of your workflow.
4. Choose **Edit**.
5. Choose **Visual**.
6. Choose **Actions**.
7. In **Actions**, choose **Test**.
8. On the **Configuration** tab, complete the **Shell commands** field by updating the sample code with your choice of the supported frameworks. For example, to use a supported framework, you would use a Run command similar to the following.

   ```
   - Run: run-tests <framework>
   ```

   If the framework you want is not supported, consider contributing a custom adapter or runner. For a description of the **Shell commands** field, see Steps.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To use universal-test-runner in the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose the name of your workflow.
4. Choose **Edit**.
5. Choose **YAML**.
6. Choose **Actions**.
7. In **Actions**, choose **Test**.
8. Modify the YAML code according to your needs. For example, to use a supported framework, you would use a Run command similar to the following.

   ```
   Configuration:
     Steps:
       - Run: run-tests <framework>
   ```

If the framework you want is not supported, consider contributing a custom adapter or runner. For a description of the **Steps** property, see Steps.

9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

Configuring quality reports in an action

This section describes how to configure a quality report in an action.

Topics

Auto-discovery and manual reports

When auto-discovery is enabled, CodeCatalyst searches all inputs passed into the action, and all files generated by the action itself, looking for test, code coverage, software composition analysis (SCA), and static analysis (SA) reports. You can view and manipulate each of these reports in CodeCatalyst.

You can also manually configure which reports are generated. You can specify the type of report you'd like to generate as well as the file format. For more information, see Quality report types.

Configuring success criteria for reports

You can set the values that determine the success criteria for a test, code coverage, software composition analysis (SCA), or static analysis (SA) report.

Success criteria are thresholds that determine whether a report passes or fails. CodeCatalyst first generates your report, which can be a test, code coverage, SCA, or SA report, and then applies the success criteria to the generated reports. It then shows whether the success criteria were met, and to what extent. If any report does not meet the specified success criteria, the CodeCatalyst action that specified the success criteria fails.

For example, when you set the success criteria for your SCA report, the valid vulnerability values ranging from most to least severe are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`. If you

set the criteria to scan for one vulnerability at `HIGH` severity, the report will fail if there is either at least one vulnerability at `HIGH` severity or no vulnerabilities at `HIGH` severity, but at least one vulnerability at a higher severity level, such as one vulnerability at `CRITICAL` severity.

If you do not specify success criteria, then:

- The CodeCatalyst report that is generated based on your raw reports will not display success criteria.
- Success criteria will not be used to determine whether the associated workflow action passes or fails.

Visual

To configure success criteria

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose a workflow containing an action that generates a report. This is the report for which you want to apply success criteria. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
3. Choose **Edit**.
4. Choose **Visual**.
5. In the workflow diagram, choose the action that you have configured to generate CodeCatalyst reports.
6. Choose the **Outputs** tab.
7. Under **Auto-discover reports** or under **Manually configure reports**, choose **Success criteria**.

   Success criteria appear. Depending on your previous selections, you may see any or all of these options:

   Pass rate

   Specify the percentage of tests in a test report that must pass for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. The pass rate criteria are applied only to test reports. For more information about test reports, see Test reports.

Line coverage

Specify the percentage of lines in a code coverage report that must be covered for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. Line coverage criteria are applied only to code coverage reports. For more information about code coverage reports, see Code coverage reports.

Branch coverage

Specify the percentage of branches in a code coverage report that must be covered for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. Branch coverage criteria are applied only to code coverage reports. For more information about code coverage reports, see Code coverage reports.

Vulnerabilities (SCA)

Specify the maximum number and severity of vulnerabilities permitted in the SCA report for the associated CodeCatalyst report to be marked as passed. To specify vulnerabilities, you must specify:

- The minimum severity of the vulnerabilities you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` vulnerabilities will be tallied.
- The maximum number of vulnerabilities of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Vulnerabilities criteria are applied only to SCA reports. For more information about SCA reports, see Software composition analysis reports.

Bugs

Specify the maximum number and severity of bugs permitted in the SA report for the associated CodeCatalyst report to be marked as passed. To specify bugs, you must specify:

- The minimum severity of the bugs you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` bugs will be tallied.

- The maximum number of bugs of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Bugs criteria are applied only to PyLint and ESLint SA reports. For more information about SA reports, see Static analysis reports.

Security vulnerabilities

Specify the maximum number and severity of security vulnerabilities permitted in the SA report for the associated CodeCatalyst report to be marked as passed. To specify security vulnerabilities, you must specify:

- The minimum severity of the security vulnerabilities you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` security vulnerabilities will be tallied.
- The maximum number of security vulnerabilities of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Security vulnerabilities criteria are applied only to PyLint and ESLint SA reports. For more information about SA reports, see Static analysis reports.

Quality issues

Specify the maximum number and severity of quality issues permitted in the SA report for the associated CodeCatalyst report to be marked as passed. To specify quality issues, you must specify:

- The minimum severity of the quality issues you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` quality issues will be tallied.

- The maximum number of quality issues of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Quality issues criteria are applied only to PyLint and ESLint SA reports. For more information about SA reports, see Static analysis reports.

8. Choose **Commit**.
9. Run your workflow to have CodeCatalyst apply success criteria to your raw reports, and regenerate the associated CodeCatalyst reports with success criteria information included. For more information, see Starting a workflow run manually.

YAML

To configure success criteria

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose a workflow containing an action that generates a report. This is the report for which you want to apply success criteria. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
3. Choose **Edit**.
4. Choose **YAML**.
5. In the workflow diagram, choose the action that you have configured to generate CodeCatalyst reports.
6. In the details pane, choose the **Outputs** tab.
7. In the action, in `AutoDiscoverReports` section, or in the `Reports` section, add a **SuccessCriteria** property, along with `PassRate`, `LineCoverage`, `BranchCoverage`, `Vulnerabilities`, `StaticAnalysisBug`, `StaticAnalysisSecurity`, and `StaticAnalysisQuality` properties.

   For an explanation of each of these properties, consult the Build and test action YAML definition.
8. Choose **Commit**.
9. Run your workflow to have CodeCatalyst apply success criteria to your raw reports, and regenerate the associated CodeCatalyst reports with the success criteria information

included. For more information on starting a workflow, see Starting a workflow run manually.

Quality reports YAML example

The following example shows how to manually configure four reports: a test report, a code coverage report, a software composition analysis report, and a static analysis report.

```
Reports:
  MyTestReport:
    Format: JUNITXML
    IncludePaths:
      - "*.xml"
    ExcludePaths:
      - report1.xml
      SuccessCriteria:
        PassRate: 90
  MyCoverageReport:
    Format: CLOVERXML
    IncludePaths:
      - output/coverage/jest/clover.xml
      SuccessCriteria:
        LineCoverage: 75
        BranchCoverage: 75
  MySCAReport:
    Format: SARIFSCA
    IncludePaths:
      - output/sca/reports.xml
      SuccessCriteria:
        Vulnerabilities:
          Number: 5
          Severity: HIGH
  MySAReport:
    Format: ESLINTJSON
    IncludePaths:
      - output/static/eslint.xml
      SuccessCriteria:
        StaticAnalysisBug:
          Number: 10
          Severity: MEDIUM
        StaticAnalysisSecurity:
          Number: 5
```

```
      Severity: CRITICAL
    StaticAnalysisQuality:
      Number: 0
      Severity: INFORMATIONAL
```

Retrying test cases of a report

If your report fails because of several test cases, you can retry only those individual tests. This allows you to quickly check the quality of your test cases and determine the next steps to resolve your issues, like engaging a broken dependency, or initiating a workflow rerun. Your test action incorporates the `universal-test-runner` to retry only selected test cases, instead of the whole action. You can only retry one set of selected test cases per action at a time, and only retry five times per test report. For more information, see Integrating universal-test-runner into a test action.

> **Note**
>
> If you retry test cases on a report, it will have no effect on the status of the workflow that generated the original report.

Use the following instructions to retry the test cases in your reports.

To retry the test cases of a report

1. In the navigation pane, choose **Reports**.
2. Choose the name of your report. You can filter by the name, status, repository, branch, or type of report.
3. Under the name of the report, choose **Results**.
4. Select the test cases that you want to retry, choose **Rerun**, then choose **Selected test cases**.
5. When your retry is finished, choose **Refresh** on the banner and view the updated results.

Best practices for testing in CodeCatalyst

When using the testing features provided by CodeCatalyst, we recommend that you follow these best practices.

Topics

- Auto-discovery
- Success criteria
- Include/exclude paths

Auto-discovery

When configuring actions in CodeCatalyst, auto-discovery lets you automatically discover outputs of various tools, such as JUnit test reports, and generate relevant CodeCatalyst reports from them. Auto-discovery helps ensure that reports continue to be generated even if names or paths to discovered outputs change. When new files are added, CodeCatalyst automatically discovers them and produces relevant reports. However, if you use auto-discovery, it is important to factor in some of the following aspects of this feature:

- When you activate auto-discovery in your action, all automatically discovered reports of the same type will share the same success criteria. For example, a shared criteria such as minimum pass rate would apply to all auto-discovered test reports. If you need different criteria for reports of the same type, you must explicitly configure each of these reports.
- Auto-discovery can also find reports that are produced by your dependencies and, if success criteria are configured, might fail the action on these reports. This issue can be addressed by updating the exclude path configuration.
- Auto-discovery is not guaranteed to produce the same list of reports every time, because it scans the action at runtime. In the case where you want a particular report to always be produced, you should configure reports explicitly. For example, if tests were to stop running as part of your build, the test framework would not produce any outputs and, as a result, no test report would be produced and the action might succeed. If you want the success of your action to depend on that particular test, then you must explicitly configure that report.

> **Tip**
>
> When getting started on a new or existing project, use auto-discovery for the entire project directory (include `**/*`). This invokes report generation across all files in your project, including those within subdirectories.

For more information, see Configuring quality reports in an action.

Success criteria

You can enforce quality thresholds on your reports by configuring success criteria. For example, if two code coverage reports were auto-discovered, one with a line coverage of 80% and the other with a line coverage of 60%, you have the following options:

- Set the auto-discovery success criteria for line coverage at 80%. This would cause the first report to pass and the second report to fail, which would result in the overall action failing. To unblock the workflow, add new tests to your project until the line coverage for the second report exceeds 80%.
- Set the auto-discovery success criteria for line coverage at 60%. This would cause both reports to pass, which would result in the action succeeding. You could then work on increasing the code coverage in the second report. However, with this approach, you cannot guarantee that the coverage in the first report is not dropping below 80%.
- Explicitly configure one or both of the reports by using the visual editor or adding an explicit YAML section and path for each report. This would allow you to configure separate success criteria and custom names for each report. However, with this approach, the action could fail if the report paths change.

For more information, see Configuring success criteria for reports.

Include/exclude paths

When reviewing action results, you can adjust the list of reports that are generated by CodeCatalyst by configuring `IncludePaths` and `ExcludePaths`.

- Use `IncludePaths` to specify the files and file paths you want CodeCatalyst to include when searching for reports. For example, if you specify `"/test/report/*"`, CodeCatalyst searches the entire build image used by the action looking for the `/test/report/` directory. When it finds that directory, CodeCatalyst then looks for reports in that directory.

  > **Note**
  >
  > For manually configured reports, `IncludePaths` must be a glob pattern that matches a single file.

- Use `ExcludePaths` to specify the files and file paths you want CodeCatalyst to exclude when searching for reports. For example, if you specify `"/test/reports/**/*"`, CodeCatalyst will

not search for files in the `/test/reports/` directory. To ignore all files in a directory, use the `**/*` glob pattern.

The following are examples of possible glob patterns.

Pattern	Description
`*.*`	Matches all object names in the current directory that contain a dot
`*.xml`	Matches all object names in the current directory ending with `.xml`
`*.{xml,txt}`	Matches all object names in the current directory ending with `.xml` or `.txt`
`**/*.xml`	Matches object names across all directories ending with `.xml`
`testFolder`	Matches an object called `testFolder` , treating it as a file
`testFolder/*`	Matches objects in one level of the subfolder from `testFolder` , such as `testFolder/file.xml`
`testFolder/*/*`	Matches objects in two levels of the subfolder from `testFolder` , such as `testFolder/reportsFolder/file.xml`
`testFolder/**`	Matches subfolder `testFolder` as well as files below `testFolder` , such as `testFolder/file.xml` and `testFolder/otherFolder/file.xml`

CodeCatalyst interprets the glob patterns as follows:

- The slash (/) character separates directories in file paths.

- The asterisk (*) character matches zero or more characters of a name component without crossing folder boundaries.
- A double asterisk (**) matches zero or more characters of a name component across all directories.

> **Note**
>
> `ExcludePaths` takes precedence over `IncludePaths`. If both `IncludePaths` and `ExcludePaths` include the same folder, that folder is not scanned for reports.

SARIF properties supported in software composition analysis and static analysis reports

SARIF (Static Analysis Results Interchange Format) is an output file format which is available in software composition analysis and static analysis reports in CodeCatalyst. The following example shows how to manually configure SARIF in a static analysis report:

```
Reports:
MySAReport:
Format: SARIFSA
IncludePaths:
    - output/sa_report.json
SuccessCriteria:
    StaticAnalysisFinding:
    Number: 25
    Severity: HIGH
```

CodeCatalyst supports the following SARIF properties which can be used to optimize how the analysis results will appear in your reports.

Topics

- sarifLog object
- run object
- toolComponent object
- reportingDescriptor object
- result object

- location object
- physicalLocation object
- logicalLocation object
- fix object

`sarifLog` object

Name	Required	Description
`$schema`	Yes	The URI of the SARIF JSON schema for version 2.1.0.
`version`	Yes	CodeCatalyst only supports SARIF version 2.1.0.
`runs[]`	Yes	A SARIF file contains an array of one or more runs, each of which represents a single run of the analysis tool.

`run` object

Name	Required	Description
`tool.driver`	Yes	A `toolComponent` object that describes the analysis tool.
`tool.name`	No	A property that indicates the name of the tool used to perform analysis.
`results[]`	Yes	The results of the analysis tool that are displayed on CodeCatalyst.

`toolComponent` object

Name	Required	Description
`name`	Yes	The name of the analysis tool.
`properties.artifactScanned`	No	A total number of artifacts analyzed by the tool.
`rules[]`	Yes	An array of `reportingDescriptor` objects that represent rules. Based on these rules, the analysis tool finds problems in the code that is analyzed.

`reportingDescriptor` object

Name	Required	Description
`id`	Yes	The unique identifier for the rule that is used to reference a finding. Maximum length: 1,024 characters
`name`	No	The display name of the rule. Maximum length: 1,024 characters
`shortDescription.text`	No	A shortened description of the rule. Maximum length: 3,000 characters

Name	Required	Description
`fullDescription.text`	No	A complete description of the rule. Maximum length: 3,000 characters
`helpUri`	No	A string that can be localized to contain the absolute URI of the primary documentation for the rule. Maximum length: 3,000 characters
`properties.unscore`	No	A flag that indicates if the scan finding has been scored.
`properties.score.severity`	No	A fixed set of strings that specify the severity level of the finding. Maximum length: 1,024 characters
`properties.cvssv3_baseSeverity`	No	A qualitative severity rating of Common Vulnerability Scoring System v3.1.
`properties.cvssv3_baseScore`	No	A CVSS v3 Base Score ranging from 0.0 - 10.0.
`properties.cvssv2_severity`	No	If CVSS v3 values are not available, CodeCatalyst searches for CVSS v2 values.
`properties.cvssv2_score`	No	A CVSS v2 Base Score ranging from 0.0 - 10.0.

Name	Required	Description
`properties.severity`	No	A fixed set of strings that specify the severity level of the finding. Maximum length: 1,024 characters
`defaultConfiguration.level`	No	The default severity of a rule.

`result` object

Name	Required	Description
`ruleId`	Yes	The unique identifier for the rule that is used to reference a finding. Maximum length: 1,024 characters
`ruleIndex`	Yes	The index of the associated rule in the tool component `rules[]`.
`message.text`	Yes	A message that describes the result and displays the message for each finding. Maximum length: 3,000 characters
`rank`	No	A value between 0.0 to 100.0 inclusive that represents the priority or importance of the result. This scale values

Name	Required	Description
		0.0 being the lowest priority and 100.0 being the highest priority.
`level`	No	The severity of the result. Maximum length: 1,024 characters
`properties.unscore`	No	A flag that indicates if the scan finding has been scored.
`properties.score.severity`	No	A fixed set of strings that specify the severity level of the finding. Maximum length: 1,024 characters
`properties.cvssv3_baseSeverity`	No	A qualitative severity rating of Common Vulnerability Scoring System v3.1.
`properties.cvssv3_baseScore`	No	A CVSS v3 Base Score ranging from 0.0 - 10.0.
`properties.cvssv2_severity`	No	If CVSS v3 values are not available, CodeCatalyst searches for CVSS v2 values.
`properties.cvssv2_score`	No	A CVSS v2 Base Score ranging from 0.0 - 10.0.

Name	Required	Description
`properties.severity`	No	A fixed set of strings that specify the severity level of the finding. Maximum length: 1,024 characters
`locations[]`	Yes	The set of locations where the result was detected. Only one location should be included unless the problem can only be corrected by making a change at every specified location. CodeCatalyst uses the first value in the location array to annotate the result. Maximum number of `location` objects: 10
`relatedLocations[]`	No	A list of additional locations references in the finding. Maximum number of `location` objects: 50
`fixes[]`	No	An array of `fix` objects that represent the recommend ation provided by the scanning tool. CodeCatalyst uses the first recommend ation in the `fixes` array.

`location` object

Name	Required	Description
`physicalLocation`	Yes	Identifies the artifact and region.
`logicalLocations[]`	No	The set of locations described by name without reference to the artifact.

`physicalLocation` object

Name	Required	Description
`artifactLocation.uri`	Yes	The URI indicating the location of an artifact, usually a file either in the repository or generated during a build.
`fileLocation.uri`	No	The fall back URI indicating the location of the file. This is used if `artifactLocation.uri` returns empty.
`region.startLine`	Yes	The line number of the first character in the region.
`region.startColumn`	Yes	The column number of the first character in the region.
`region.endLine`	Yes	The line number of the last character in the region.
`region.endColumn`	Yes	The column number of the last character in the region.

`logicalLocation` object

Name	Required	Description
`fullyQualifiedName`	No	Additional information that describes the location of the result. Maximum length: 1,024 characters

`fix` object

Name	Required	Description
`description.text`	No	A message that displays a recommendation for each finding. Maximum length: 3,000 characters
`artifactChanges.[0].artifactLocation.uri`	No	The URI indicating the location of the artifact that needs to be updated.

Deploying with workflows

Using CodeCatalyst workflows, you can deploy applications and other resources to various targets such as Amazon ECS, AWS Lambda, and more.

How do I deploy an application?

To deploy an application or resource through CodeCatalyst, you first create a workflow, and then specify a deploy action inside of it. A *deploy action* is a workflow building block that defines *what* you want to deploy, *where* you want to deploy it, and *how* you want to deploy it (for example, using

a blue/green scheme). You add a deploy action to your workflow using the CodeCatalyst console's visual editor, or YAML editor.

The high-level steps to deploy an application or resource are as follows.

To deploy an application (high-level tasks)

1. In your CodeCatalyst project, you **add source code** for an application you want to deploy. For more information, see Storing source code in repositories for a project in CodeCatalyst.
2. In your CodeCatalyst project, you **add an environment** that defines the target AWS account and optional Amazon Virtual Private Cloud (VPC) that you want to deploy to. For more information, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.
3. In your CodeCatalyst project, you **create a workflow**. The workflow is where you define how to build, test, and deploy your application. For more information, see Getting started with workflows.
4. In the workflow, you **add a trigger**, a **build action**, and optionally, a **test action**. For more information, see Starting a workflow run automatically with triggers, Adding the build action, and Adding the test action.
5. In the workflow, you **add a deploy action**. You can choose from several CodeCatalyst-provided deploy actions to your application to different targets, such as Amazon ECS. (You can also use a build action or a GitHub Action to deploy your application. For more information about the build action and GitHub Actions, see Alternatives to deploy actions.)
6. You **start the workflow** either manually or automatically through a trigger. The workflow runs the build, test, and deploy actions in sequence to deploy your application and resources to the target. For more information, see Starting a workflow run manually.

List of deploy actions

The following deploy actions are available:

- Deploy AWS CloudFormation stack – This action creates a CloudFormation stack in AWS based on an AWS CloudFormation template or AWS Serverless Application Model template that you provide. For more information, see Deploying an AWS CloudFormation stack with a workflow.
- Deploy to Amazon ECS – This action registers a task definition file that you provide. For more information, see Deploying an application to Amazon Elastic Container Service (ECS) with a workflow.

- Deploy to Kubernetes cluster – This action deploys an application to an Amazon Elastic Kubernetes Service cluster. For more information, see Deploying an application to Amazon Elastic Kubernetes Service with a workflow.
- AWS CDK deploy – This action deploys an AWS CDK app into AWS. For more information, see Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow.

> **Note**
>
> There are other CodeCatalyst actions that can deploy resources; however, they are not considered *deploy* actions because their deployment information doesn't appear on the **Environments** page. To learn more about the **Environments** page and viewing deployments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Viewing deployment status, commits, and pull requests.

Benefits of deploy actions

Using deploy actions within a workflow has the following benefits:

- **Deployment history** – View a history of your deployments to help manage and communicate changes in your deployed software.
- **Traceability** – Track the status of your deployments through the CodeCatalyst console, and see when and where each application revision was deployed.
- **Rollbacks** – Roll back deployments automatically if there are errors. You can also configure alarms to activate deployment rollbacks.
- **Monitoring** – Watch your deployment as it progresses through the various stages of your workflow.
- **Integration with other CodeCatalyst features** – Store source code and then build, test, and deploy it, all from one application.

Alternatives to deploy actions

You don't have to use deploy actions, although they are recommended because they offer the benefits outlined in the preceding section. Instead, you can use the following CodeCatalyst actions :

- A **build** action.

Typically, you use build actions if you want to deploy to a target for which a corresponding deploy action does not exist, or if you want more control over the deployment procedure. For more information about using build actions to deploy resources, see Building with workflows.

- A **GitHub Action**.

  You can use a GitHub Action inside a CodeCatalyst workflow to deploy applications and resources (instead of a CodeCatalyst action). For information about how to use GitHub Actions inside a CodeCatalyst workflow, see Integrating GitHub Actions into a workflow

You can also use the following AWS services to deploy your application, if you don't want to use a CodeCatalyst workflow to do so:

- AWS CodeDeploy – see What is CodeDeploy?
- AWS CodeBuild and AWS CodePipeline – see What is AWS CodeBuild? and What is AWS CodePipeline?
- AWS CloudFormation – see What is AWS CloudFormation?

Use CodeDeploy, CodeBuild, CodePipeline, and CloudFormation services for complex, enterprise deployments.

Topics

- Deploying an application to Amazon Elastic Container Service (ECS) with a workflow
- Deploying an application to Amazon Elastic Kubernetes Service with a workflow
- Deploying an AWS CloudFormation stack with a workflow
- Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow
- Bootstrapping an AWS CDK app with a workflow
- Publishing files to Amazon S3 with a workflow
- Deploying into AWS accounts and VPCs with CodeCatalyst environments
- Displaying the URL of the deployed application in the workflow diagram
- Removing a deployment target
- Tracking deployment status by commit
- Viewing the deployment logs
- Viewing deployment status, commits, and pull requests

Deploying an application to Amazon Elastic Container Service (ECS) with a workflow

This section describes how to deploy a containerized application into an Amazon ECS cluster using a CodeCatalyst workflow. To accomplish this, you must add the **Deploy to Amazon ECS** action to your workflow. This action registers a task definition file that you provide. Upon registration, the task definition is instantiated by your Amazon ECS service running in your Amazon ECS cluster. "Instantiating a task definition" is equivalent to deploying an application into Amazon ECS.

To use this action, you must have an Amazon ECS cluster, service, and task definition file ready.

For more information about Amazon ECS, see the *Amazon Elastic Container Service Developer Guide*.

> **Tip**
>
> For a tutorial that shows you how to use the **Deploy to Amazon ECS** action, see Tutorial: Deploy an application to Amazon ECS.

> **Tip**
>
> For a working example of the **Deploy to Amazon ECS** action, create a project with either the **Node.js API with AWS Fargate** or **Java API with AWS Fargate** blueprint. For more information, see Creating a project with a blueprint.

Topics

Tutorial: Deploy an application to Amazon ECS

In this tutorial, you learn how to deploy a serverless application into Amazon Elastic Container Service (Amazon ECS) using a workflow, Amazon ECS, and a few other AWS services. The deployed

application is a simple Hello World website built on an Apache web server Docker image. The tutorial walks you through the required preparation work such as setting up a cluster, and then describes how to create a workflow to build and deploy the application.

> **Tip**
>
> Instead of working your way through this tutorial, you can use a blueprint that does a complete Amazon ECS setup for you. You'll need to use either the **Node.js API with AWS Fargate** or **Java API with AWS Fargate** blueprint. For more information, see Creating a project with a blueprint.

Topics

Prerequisites

Before you begin:

- You need a CodeCatalyst **space** with a connected AWS account. For more information, see Creating a space.
- In your space, you need an empty, **Start from scratch** CodeCatalyst **project** called:

```
codecatalyst-ecs-project
```

For more information, see Creating an empty project in Amazon CodeCatalyst.

- In your project, you need a CodeCatalyst **environment** called:

```
codecatalyst-ecs-environment
```

Configure this environment as follows:

- Choose any type, such as **Non-production**.
- Connect your AWS account to it.
- For the **Default IAM role**, choose any role. You'll specify a different role later.

For more information, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.

Step 1: Set up an AWS user and AWS CloudShell

The first step in this tutorial is to create a user in AWS IAM Identity Center, and launch an AWS CloudShell instance as this user. For the duration of this tutorial, CloudShell is your development computer and is where you configure AWS resources and services. Delete this user after completing the tutorial.

> **Note**
>
> Do not use your root user for this tutorial. You must create a separate user or else you may experience problems when performing actions in the AWS Command Line Interface (CLI) later on.

For more information about IAM Identity Center users and CloudShell, see the *AWS IAM Identity Center User Guide* and *AWS CloudShell User Guide*.

To create an IAM Identity Center user

1. Sign in to the AWS Management Console and open the AWS IAM Identity Center console at https://console.aws.amazon.com/singlesignon/.

> **Note**
>
> Make sure you sign in using the AWS account that is connected to your CodeCatalyst space. You can verify which account is connected by navigating to your space and choosing the **AWS accounts** tab. For more information, see Creating a space.

2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. In **Username**, enter:

```
CodeCatalystECSUser
```

4. Under **Password**, choose **Generate a one-time password that you can share with this user**.
5. In **Email address** and **Confirm email address**, enter an email address that doesn't already exist in IAM Identity Center.
6. In **First name** and **Last name**, enter:

```
CodeCatalystECSUser
```

7. In **Display name**, keep the automatically generated name:

```
CodeCatalystECSUser CodeCatalystECSUser
```

8. Choose **Next**.
9. On the **Add user to groups** page, choose **Next**.
10. On the **Review and add user** page, review the information and choose **Add user**.

    A **One-time password** dialog box appears.
11. Choose **Copy** and then paste the sign-in information, including the AWS access portal URL and the one-time password.
12. Choose **Close**.

To create a permission set

You'll assign this permission set to `CodeCatalystECSUser` later.

1. In the navigation pane, choose **Permission sets**, and then choose **Create permission set**.

2. Choose **Predefined permission set** and then select **AdministratorAccess**. This policy provides full permissions to all AWS services.
3. Choose **Next**.
4. In **Permission set name**, enter:

   ```
   CodeCatalystECSPermissionSet
   ```

5. Choose **Next**.
6. On the **Review and create** page, review the information and choose **Create**.

To assign the permission set to CodeCatalystECSUser

1. In the navigation pane, choose **AWS accounts**, and then select the check box next to the AWS account that you're currently signed in to.
2. Choose **Assign users or groups**.
3. Choose the **Users** tab.
4. Select the check box next to `CodeCatalystECSUser`.
5. Choose **Next**.
6. Select the check box next to `CodeCatalystECSPermissionSet`.
7. Choose **Next**.
8. Review the information and choose **Submit**.

   You have now assigned `CodeCatalystECSUser` and `CodeCatalystECSPermissionSet` to your AWS account, binding them together.

To sign out and sign back in as CodeCatalystECSUser

1. Before you sign out, make sure you have the AWS access portal URL and the username and one-time password for `CodeCatalystECSUser`. You should have copied this information to a text editor earlier.

> **Note**
>
> If you do not have this information, go to the `CodeCatalystECSUser` details page in IAM Identity Center, choose **Reset password**, **Generate a one-time password [...]**, and **Reset password** again to display the information on the screen.

2. Sign out of AWS.
3. Paste the AWS access portal URL into your browser's address bar.
4. Sign in with the username and one-time password for `CodeCatalystECSUser`.
5. In **New password**, enter a password, and choose **Set new password**.

   An **AWS account** box appears on the screen.
6. Choose **AWS account**, and then choose the name of the AWS account to which you assigned the `CodeCatalystECSUser` user and permission set.
7. Next to the `CodeCatalystECSPermissionSet`, choose **Management console**.

   The AWS Management Console appears. You are now signed in as `CodeCatalystECSUser` with the appropriate permissions.

To launch an AWS CloudShell instance

1. As `CodeCatalystECSUser`, in the top navigation bar, choose the AWS icon ().

   The main page of the AWS Management Console appears.
2. In the top navigation bar, choose the AWS CloudShell icon ().

   CloudShell opens. Wait while the CloudShell environment is created.

> **Note**
>
> If you don't see the CloudShell icon, make sure that you're in a Region supported by CloudShell. This tutorial assumes you are in the US West (Oregon) Region.

To verify that the AWS CLI is installed

1. In the CloudShell terminal, enter:

```
aws --version
```

2. Check that a version appears.

   The AWS CLI is already configured for the current user, `CodeCatalystECSUser`, so there is no need to configure AWS CLI keys and credentials, as is normally the case.

Step 2: Deploy a placeholder application into Amazon ECS

In this section, you manually deploy a placeholder application into Amazon ECS. This placeholder application will be replaced by the Hello World application deployed by your workflow. The placeholder application is Apache Web Server.

For more information about Amazon ECS, see the *Amazon Elastic Container Service Developer Guide*.

Complete the following series of procedures to deploy the placeholder application.

To create the task execution role

This role grants Amazon ECS and AWS Fargate (Fargate) permission to make API calls on your behalf.

1. Create a trust policy:

   a. In AWS CloudShell, enter the following command:

```
cat > codecatalyst-ecs-trust-policy.json
```

   A blinking prompt appears in the CloudShell terminal.

   b. Enter the following code at the prompt:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

c. Place your cursor after the last curly bracket (}).

d. Press **Enter** and then **Ctrl+d** to save the file and exit cat.

2. Create a task execution role:

```
aws iam create-role \
      --role-name codecatalyst-ecs-task-execution-role \
      --assume-role-policy-document file://codecatalyst-ecs-trust-policy.json
```

3. Attach the AWS managed `AmazonECSTaskExecutionRolePolicy` policy to the role:

```
aws iam attach-role-policy \
      --role-name codecatalyst-ecs-task-execution-role \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy
```

4. Display the role's details:

```
aws iam get-role \
      --role-name codecatalyst-ecs-task-execution-role
```

5. Note the role's `"Arn":` value, for example, `arn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role`. You will need this Amazon Resource Name (ARN) later.

To create an Amazon ECS cluster

This cluster will contain the Apache placeholder application, and later, the Hello World application.

1. As `CodeCatalystECSUser`, in AWS CloudShell, create an empty cluster:

```
aws ecs create-cluster --cluster-name codecatalyst-ecs-cluster
```

2. (Optional) Verify that the cluster was created successfully:

```
aws ecs list-clusters
```

The ARN of the `codecatalyst-ecs-cluster` cluster should appear in the list, indicating a successful creation.

To create a task definition file

The task definition file indicates to run the Apache 2.4 Web server Docker image (`httpd:2.4`) which is pulled from DockerHub.

1. As `CodeCatalystECSUser`, in AWS CloudShell, create a task definition file:

```
cat > taskdef.json
```

2. Paste the following code at the prompt:

```
{
    "executionRoleArn": "arn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role",
    "containerDefinitions": [
        {
            "name": "codecatalyst-ecs-container",
            "image": "httpd:2.4",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "family": "codecatalyst-ecs-task-def",
    "memory": "512",
```

```
    "networkMode": "awsvpc"
}
```

In the preceding code, replace *arn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role*

with the ARN of the task execution role that you noted in To create the task execution role.

3. Place your cursor after the last curly bracket (}).
4. Press **Enter** and then **Ctrl+d** to save the file and exit cat.

To register the task definition file with Amazon ECS

1. As `CodeCatalystECSUser`, in AWS CloudShell, register the task definition:

```
aws ecs register-task-definition \
    --cli-input-json file://taskdef.json
```

2. (Optional) Verify that the task definition was registered:

```
aws ecs list-task-definitions
```

The `codecatalyst-ecs-task-def` task definition should appear in the list.

To create the Amazon ECS service

The Amazon ECS service runs the tasks (and associated Docker containers) of the Apache placeholder application, and later, the Hello World application.

1. As `CodeCatalystECSUser`, switch to the Amazon Elastic Container Service console if you haven't done so already.
2. Choose the cluster you created earlier, `codecatalyst-ecs-cluster`.
3. In the **Services** tab, choose **Create**.
4. In the **Create** page, do the following:
   a. Keep all default settings except for those listed next.
   b. For **Launch type**, choose **FARGATE**.
   c. Under **Task definition**, in the **Family** drop-down list, choose:

```
codecatalyst-ecs-task-def
```

d. For **Service name**, enter:

```
codecatalyst-ecs-service
```

e. For **Desired tasks**, enter:

```
3
```

In this tutorial, each task launches a single Docker container.

f. Expand the **Networking** section.

g. For **VPC**, choose any VPC.

h. For **Subnets**, choose any subnet.

> **Note**
>
> Only specify one subnet. That's all that is needed for this tutorial.

> **Note**
>
> If you don't have a VPC and subnet, create them. See Create a VPC, and Create a subnet in your VPC in the *Amazon VPC User Guide*.

i. For **Security group**, choose **Create a new security group**, and then do the following:

i. For **Security group name**, enter:

```
codecatalyst-ecs-security-group
```

ii. For **Security group description**, enter:

```
CodeCatalyst ECS security group
```

iii. Choose **Add rule**. For **Type**, choose **HTTP**, and for **Source**, choose **Anywhere**.

j. At the bottom, choose **Create**.

k. Wait while the service is created. This may take a few minutes.

5. Choose the **Tasks** tab, and then choose the refresh button. Verify that all three tasks have their **Last Status** column set to **Running**.

(Optional) To verify that your Apache placeholder application is running

1. In the **Tasks** tab, choose any one of the three tasks.
2. In the **Public IP** field, choose **open address**.

   An `It Works!` page appears. This indicates that the Amazon ECS service successfully started a task that launched a Docker container with the Apache image.

   At this point in the tutorial, you have manually deployed an Amazon ECS cluster, service, and task definition, as well as an Apache placeholder application. With all these items in place, you are now ready to create a workflow that will replace the Apache placeholder application with the tutorial's Hello World application.

Step 3: Create an Amazon ECR image repository

In this section, you create a private image repository in Amazon Elastic Container Registry (Amazon ECR). This repository stores the tutorial's Docker image that will replace the Apache placeholder image you deployed previously.

For more information about Amazon ECR, see the *Amazon Elastic Container Registry User Guide*.

To create an image repository in Amazon ECR

1. As `CodeCatalystECSUser`, in AWS CloudShell, create an empty repository in Amazon ECR:

   ```
   aws ecr create-repository --repository-name codecatalyst-ecs-image-repo
   ```

2. Display the Amazon ECR repository's details:

   ```
   aws ecr describe-repositories \
         --repository-names codecatalyst-ecs-image-repo
   ```

3. Note the "`repositoryUri`": value, for example, `111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo`.

   You need it later when adding the repository to your workflow.

Step 4: Create AWS roles

In this section, you create AWS IAM roles that your CodeCatalyst workflow will need in order to function. These roles are:

- **Build role** – Grants the CodeCatalyst build action (in the workflow) permission to access your AWS account and write to Amazon ECR and Amazon EC2.
- **Deploy role** – Grants the CodeCatalyst **Deploy to ECS** action (in the workflow) permission to access your AWS account, Amazon ECS, and a few other AWS services.

For more information about IAM roles, see IAM roles in the *AWS Identity and Access Management User Guide*.

> **Note**
>
> To save time, you can create a single role, called the `CodeCatalystWorkflowDevelopmentRole-spaceName` role, instead of the two roles listed previously. For more information, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-spaceName` role has very broad permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern. This tutorial assumes you are creating the two roles listed previously.

To create the build and deploy roles, you can use either the AWS Management Console or the AWS CLI.

AWS Management Console

To create the build and deploy roles, complete the following series of procedures.

To create a build role

1. Create a policy for the role, as follows:
   a. Sign in to AWS.
   b. Open the IAM console at https://console.aws.amazon.com/iam/.
   c. In the navigation pane, choose **Policies**.

d. Choose **Create policy**.

e. Choose the **JSON** tab.

f. Delete the existing code.

g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-ecs-build-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the build role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. Choose **Next**.

f. In **Permissions policies**, search for `codecatalyst-ecs-build-policy`, select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-ecs-build-role
```

i. For **Role description**, enter:

```
CodeCatalyst ECS build role
```

j. Choose **Create role**.

You have now created a build role with a permissions policy and a trust policy.

3. Obtain the build role ARN, as follows:

a. In the navigation pane, choose **Roles**.

b. In the search box, enter the name of the role you just created (`codecatalyst-ecs-build-role`).

c. Choose the role from the list.

The role's **Summary** page appears.

d. At the top, copy the **ARN** value. You need it later.

To create a deploy role

1. Create a policy for the role, as follows:

a. Sign in to AWS.

b. Open the IAM console at https://console.aws.amazon.com/iam/.

c. In the navigation pane, choose **Policies**.

d. Choose **Create Policy**.

e. Choose the **JSON** tab.

f. Delete the existing code.

g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action":[
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs:DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
```

```
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
   ],
   "Resource":"*",
   "Effect":"Allow"
},{"Action":[
      "iam:PassRole"
   ],
   "Effect":"Allow",
   "Resource":"*",
   "Condition":{"StringLike":{"iam:PassedToService":[
            "ecs-tasks.amazonaws.com",
            "codedeploy.amazonaws.com"
         ]
      }
   }
}]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement. You can then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-ecs-deploy-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the deploy role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
```

```
    ]
}
```

e. Choose **Next**.

f. In **Permissions policies**, search for `codecatalyst-ecs-deploy-policy`, select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-ecs-deploy-role
```

i. For **Role description**, enter:

```
CodeCatalyst ECS deploy role
```

j. Choose **Create role**.

You have now created a deploy role with a trust policy.

3. Obtain the deploy role ARN, as follows:

a. In the navigation pane, choose **Roles**.

b. In the search box, enter the name of the role you just created (`codecatalyst-ecs-deploy-role`).

c. Choose the role from the list.

The role's **Summary** page appears.

d. At the top, copy the **ARN** value. You need it later.

AWS CLI

To create the build and deploy roles, complete the following series of procedures.

To create a trust policy for both roles

As `CodeCatalystECSUser`, in AWS CloudShell, create a trust policy file:

1. Create the file:

```
cat > codecatalyst-ecs-trust-policy.json
```

2. At the terminal prompt, paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

3. Place your cursor after the last curly bracket (}).
4. Press **`Enter`** and then **`Ctrl+d`** to save the file and exit cat.

To create the build policy and build role

1. Create the build policy:

   a. As `CodeCatalystECSUser`, in AWS CloudShell, create a build policy file:

   ```
   cat > codecatalyst-ecs-build-policy.json
   ```

   b. At the prompt, enter the following code:

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "ecr:*",
   ```

```
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

c. Place your cursor after the last curly bracket (}).

d. Press **Enter** and then **Ctrl+d** to save the file and exit cat.

2. Add the build policy to AWS:

```
aws iam create-policy \
    --policy-name codecatalyst-ecs-build-policy \
    --policy-document file://codecatalyst-ecs-build-policy.json
```

3. In the command output, note the `"Arn":` value, for example, `arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy`. You need this ARN later.

4. Create the build role and attach the trust policy to it:

```
aws iam create-role \
      --role-name codecatalyst-ecs-build-role \
      --assume-role-policy-document file://codecatalyst-ecs-trust-policy.json
```

5. Attach the build policy to the build role:

```
aws iam attach-role-policy \
      --role-name codecatalyst-ecs-build-role \
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-
policy
```

Where *arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy* is replaced with the ARN of the build policy you noted earlier.

6. Display the build role's details:

```
aws iam get-role \
      --role-name codecatalyst-ecs-build-role
```

7. Note the role's `"Arn":` value, for example, `arn:aws:iam::111122223333:role/codecatalyst-ecs-build-role`. You need this ARN later.

To create the deploy policy and deploy role

1. Create a deploy policy:

   a. In AWS CloudShell, create a deploy policy file:

   ```
   cat > codecatalyst-ecs-deploy-policy.json
   ```

   b. At the prompt, enter the following code:

   ```
   {
       "Version": "2012-10-17",
       "Statement": [{
       "Action":[
         "ecs:DescribeServices",
         "ecs:CreateTaskSet",
         "ecs:DeleteTaskSet",
         "ecs:ListClusters",
         "ecs:RegisterTaskDefinition",
         "ecs:UpdateServicePrimaryTaskSet",
         "ecs:UpdateService",
         "elasticloadbalancing:DescribeTargetGroups",
         "elasticloadbalancing:DescribeListeners",
         "elasticloadbalancing:ModifyListener",
         "elasticloadbalancing:DescribeRules",
         "elasticloadbalancing:ModifyRule",
         "lambda:InvokeFunction",
         "lambda:ListFunctions",
         "cloudwatch:DescribeAlarms",
         "sns:Publish",
         "sns:ListTopics",
         "s3:GetObject",
         "s3:GetObjectVersion",
         "codedeploy:CreateApplication",
         "codedeploy:CreateDeployment",
         "codedeploy:CreateDeploymentGroup",
         "codedeploy:GetApplication",
         "codedeploy:GetDeployment",
         "codedeploy:GetDeploymentGroup",
         "codedeploy:ListApplications",
         "codedeploy:ListDeploymentGroups",
         "codedeploy:ListDeployments",
         "codedeploy:StopDeployment",
   ```

```
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
   ],
   "Resource":"*",
   "Effect":"Allow"
},{"Action":[
      "iam:PassRole"
   ],
   "Effect":"Allow",
   "Resource":"*",
   "Condition":{"StringLike":{"iam:PassedToService":[
            "ecs-tasks.amazonaws.com",
            "codedeploy.amazonaws.com"
         ]
      }
   }
}]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

c. Place your cursor after the last curly bracket (}).

d. Press **`Enter`** and then **`Ctrl+d`** to save the file and exit cat.

2. Add the deploy policy to AWS:

```
aws iam create-policy \
    --policy-name codecatalyst-ecs-deploy-policy \
    --policy-document file://codecatalyst-ecs-deploy-policy.json
```

3. In the command output, note the deploy policy's `"arn":` value, for example, `arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy`. You need this ARN later.
4. Create the deploy role and attach the trust policy to it:

```
aws iam create-role \
      --role-name codecatalyst-ecs-deploy-role \
      --assume-role-policy-document file://codecatalyst-ecs-trust-policy.json
```

5. Attach the deploy policy to the deploy role, where *arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy* is replaced with the ARN of the deploy policy you noted earlier.

```
aws iam attach-role-policy \
      --role-name codecatalyst-ecs-deploy-role \
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-
policy
```

6. Display the deploy role's details:

```
aws iam get-role \
      --role-name codecatalyst-ecs-deploy-role
```

7. Note the role's `"Arn":` value, for example, `arn:aws:iam::111122223333:role/codecatalyst-ecs-deploy-role`. You need this ARN later.

Step 5: Add AWS roles to CodeCatalyst

In this step, you add the build role (`codecatalyst-ecs-build-role`) and deploy role (`codecatalyst-ecs-deploy-role`) to the CodeCatalyst account connection in your space.

To add build and deploy roles to your account connection

1. In CodeCatalyst, navigate to your space.
2. Choose **AWS accounts**. A list of account connections appears.

3. Choose the account connection that represents the AWS account where you created your build and deploy roles.
4. Choose **Manage roles from AWS management console**.

   The **Add IAM role to Amazon CodeCatalyst space** page appears. You might need to sign in to access the page.
5. Select **Add an existing role you have created in IAM**.

   A drop-down list appears. The list displays all IAM roles with a trust policy that includes the `codecatalyst-runner.amazonaws.com` and `codecatalyst.amazonaws.com` service principals.
6. In the drop-down list, choose `codecatalyst-ecs-build-role`, and choose **Add role**.

   > **Note**
   >
   > If you see `The security token included in the request is invalid`, it might be because you do not have the right permissions. To fix this issue, sign out of AWS as sign back in with the AWS account that you used when you created your CodeCatalyst space.
7. Choose **Add IAM role**, choose **Add an existing role you have created in IAM**, and in the drop-down list, choose `codecatalyst-ecs-deploy-role`. Choose **Add role**.

   You have now added the build and deploy roles to your space.
8. Copy the value of the **Amazon CodeCatalyst display name**. You'll need this value later, when creating your workflow.

Step 6: Create a source repository

In this step, you create a source repository in CodeCatalyst. This repository stores the tutorial's source files, such as the task definition file.

For more information about source repositories, see Creating a source repository.

To create a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, `codecatalyst-ecs-project`.

3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose **Add repository**, and then choose **Create repository**.
5. In **Repository name**, enter:

```
codecatalyst-ecs-source-repository
```

6. Choose **Create**.

Step 7: Add source files

In this section, you add the Hello World source files to your CodeCatalyst repository, `codecatalyst-ecs-source-repository`. They consist of:

- An `index.html` file – Displays a Hello World message in the browser.
- A Dockerfile – Describes the base image to use for your Docker image and the Docker commands to apply to it.
- A `taskdef.json` file – Defines the Docker image to use when launching tasks into your cluster.

The folder structure is as follows:

```
.
|– public-html
|   |– index.html
|– Dockerfile
|– taskdef.json
```

> **Note**
>
> The following instructions show you how to add the files using the CodeCatalyst console but you can use Git if you prefer. For details, see Cloning a source repository.

Topics

- index.html
- Dockerfile
- taskdef.json

index.html

The `index.html` file displays a Hello World message in the browser.

To add the index.html file

1. In the CodeCatalyst console, go to your source repository, `codecatalyst-ecs-source-repository`.
2. In **Files**, choose **Create file**.
3. For **File name**, enter:

   ```
   public-html/index.html
   ```

   > **Important**
   >
   > Make sure to include the `public-html/` prefix to create a folder of the same name. The `index.html` is expected to be in this folder.

4. In the text box, enter the following code:

   ```
   <html>
     <head>
       <title>Hello World</title>
       <style>
         body {
         background-color: black;
         text-align: center;
         color: white;
         font-family: Arial, Helvetica, sans-serif;
         }
       </style>
     </head>
     <body>
       <h1>Hello World</h1>
     </body>
   </html>
   ```

5. Choose **Commit**, and then choose **Commit** again.

   The `index.html` is added to your repository in a `public-html` folder.

Dockerfile

The Dockerfile describes the base Docker image to use and the Docker commands to apply to it. For more information about the Dockerfile, see the Dockerfile Reference.

The Dockerfile specified here indicates to use the Apache 2.4 base image (`httpd`). It also includes instructions for copying a source file called `index.html` to a folder on the Apache server that serves webpages. The `EXPOSE` instruction in the Dockerfile tells Docker that the container is listening on port 80.

To add the Dockerfile

1. In your source repository, choose **Create file**.
2. For **File name**, enter:

   ```
   Dockerfile
   ```

   Do not include a file extension.

   > **Important**
   >
   > The Dockerfile must reside in your repository's root folder. The workflow's `Docker build` command expects it to be there.

3. In the text box, enter the following code:

   ```
   FROM httpd:2.4
   COPY ./public-html/index.html /usr/local/apache2/htdocs/index.html
   EXPOSE 80
   ```

4. Choose **Commit**, and then choose **Commit** again.

   The Dockerfile is added to your repository.

taskdef.json

The `taskdef.json` file that you add in this step is the same as the one you already specified in Step 2: Deploy a placeholder application into Amazon ECS with the following difference:

Instead of specifying a hardcoded Docker image name in the `image:` field (`httpd:2.4`), the task definition here uses a couple of variables to denote the image: `$REPOSITORY_URI` and `$IMAGE_TAG`. These variables will be replaced with real values generated by the workflow's build action when you run the workflow in a later step.

For details on the task definition parameters, see Task definition parameters in the *Amazon Elastic Container Service Developer Guide*.

To add the taskdef.json file

1. In your source repository, choose **Create file**.
2. For **File name**, enter:

```
taskdef.json
```

3. In the text box, enter the following code:

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-
execution-role",
    "containerDefinitions": [
        {
            "name": "codecatalyst-ecs-container",
            # The $REPOSITORY_URI and $IMAGE_TAG variables will be replaced
            # by the workflow at build time (see the build action in the
            # workflow)
            "image": $REPOSITORY_URI:$IMAGE_TAG,
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
```

```
    "memory": "512",
    "family": "codecatalyst-ecs-task-def"
}
```

In the preceding code, replace

arn:aws:iam::account_ID:role/codecatalyst-ecs-task-execution-role

with the ARN of the task execution role that you noted in To create the task execution role.

4. Choose **Commit**, and then choose **Commit** again.

   The `taskdef.json` file is added to your repository.

Step 8: Create and run a workflow

In this step, you create a workflow that takes your source files, builds them into a Docker image, and then deploys the image to your Amazon ECS cluster. This deployment replaces the existing Apache placeholder application.

The workflow consists of the following building blocks that run sequentially:

- A trigger – This trigger starts the workflow run automatically when you push a change to your source repository. For more information about triggers, see Starting a workflow run automatically with triggers.
- A build action (`BuildBackend`) – On trigger, the action builds the Docker image using the Dockerfile and pushes the image to Amazon ECR. The build action also updates the `taskdef.json` with the correct `image` field value, and then creates an output artifact of this file. This artifact is used as the input for the deploy action, which is next.

  For more information about the build action, see Building with workflows.
- A deploy action (`DeployToECS`) – On completion of the build action, the deploy action looks for the output artifact generated by the build action (`TaskDefArtifact`), finds the `taskdef.json` inside of it, and registers it with your Amazon ECS service. The service then follows the instructions in the `taskdef.json` file to run three Amazon ECS tasks—and associated Hello World Docker containers—inside your Amazon ECS cluster.

To create a workflow

1. In the CodeCatalyst console, in the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose **Create workflow**.
3. For **Source repository**, choose `codecatalyst-ecs-source-repository`.
4. For **Branch**, choose `main`.
5. Choose **Create**.
6. Delete the YAML sample code.
7. Add the following YAML code:

> **Note**
>
> In the YAML code that follows, you can omit the `Connections:` sections if you want. If you omit these sections, you must ensure that the role specified in the **Default IAM role** field in your environment includes the permissions and trust policies of both roles described in Step 5: Add AWS roles to CodeCatalyst. For more information about setting up an environment with a default IAM role, see Creating an environment.

```
Name: codecatalyst-ecs-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildBackend:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-build-role
    Inputs:
      Sources:
        - WorkflowSource
      Variables:
```

```
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-
image-repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      Steps:
        #pre_build:
        - Run: echo Logging in to Amazon ECR...
        - Run: aws --version
        - Run: aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
        #build:
        - Run: echo Build started on `date`
        - Run: echo Building the Docker image...
        - Run: docker build -t $REPOSITORY_URI:latest .
        - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
        #post_build:
        - Run: echo Build completed on `date`
        - Run: echo Pushing the Docker images...
        - Run: docker push $REPOSITORY_URI:latest
        - Run: docker push $REPOSITORY_URI:$IMAGE_TAG
        # Replace the variables in taskdef.json
        - Run: find taskdef.json -type f | xargs sed -i "s|\$REPOSITORY_URI|
$REPOSITORY_URI|g"
        - Run: find taskdef.json -type f | xargs sed -i "s|\$IMAGE_TAG|$IMAGE_TAG|
g"
        - Run: cat taskdef.json
        # The output artifact will be a zip file that contains a task definition
 file.
    Outputs:
      Artifacts:
        - Name: TaskDefArtifact
          Files:
            - taskdef.json
  DeployToECS:
    DependsOn:
      - BuildBackend
    Identifier: aws/ecs-deploy@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-deploy-role
```

```
    Inputs:
      Sources: []
      Artifacts:
        - TaskDefArtifact
    Configuration:
      region: us-west-2
      cluster: codecatalyst-ecs-cluster
      service: codecatalyst-ecs-service
      task-definition: taskdef.json
```

In the preceding code, replace:

- Both instances of *codecatalyst-ecs-environment* with the name of the environment you created in Prerequisites.
- Both instances of *codecatalyst-account-connection* with the display name of your account connection. The display name might be a number. For more information, see Step 5: Add AWS roles to CodeCatalyst.
- *codecatalyst-ecs-build-role* with the name of the build role you created in Step 4: Create AWS roles.
- *111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo* (in the `Value:` property) with the URI of the Amazon ECR repository you created in Step 3: Create an Amazon ECR image repository.
- *111122223333.dkr.ecr.us-west-2.amazonaws.com* (in the `Run: aws ecr` command) with the URI of the Amazon ECR repository without the image suffix (`/codecatalyst-ecs-image-repo`).
- *codecatalyst-ecs-deploy-role* with the name of the deploy role you created in Step 4: Create AWS roles.
- Both instances of *us-west-2* with your AWS Region code. For a list of Region codes, see Regional endpoints in the *AWS General Reference*.

> **Note**
>
> If you decided not to create build and deploy roles, replace *codecatalyst-ecs-build-role* and *codecatalyst-ecs-deploy-role* with the name of the `CodeCatalystWorkflowDevelopmentRole-`*spaceName* role. For more information about this role, see Step 4: Create AWS roles.

> **Tip**
>
> Instead of using the `find` and `sed` commands shown in the previous workflow code to update the repository and image name, you can use the **Render Amazon ECS task definition** action for this purpose. For more information, see Modifying an Amazon ECS task definition file using a workflow.

8. (Optional) Choose **Validate** to make sure that the YAML code is valid before committing.
9. Choose **Commit**.
10. In the **Commit workflow** dialog box, enter the following:

    a. For **Commit message**, remove the text and enter:

    ```
    Add first workflow
    ```

    b. For **Repository**, choose `codecatalyst-ecs-source-repository`.

    c. For **Branch name**, choose main.

    d. Choose **Commit**.

    You have now created a workflow. A workflow run starts automatically because of the trigger defined at the top of the workflow. Specifically, when you committed (and pushed) the `workflow.yaml` file to your source repository, the trigger started the workflow run.

To view the workflow run progress

1. In the navigation pane of the CodeCatalyst console, choose **CI/CD**, and then choose **Workflows**.
2. Choose the workflow you just created, `codecatalyst-ecs-workflow`.
3. Choose **BuildBackend** to see the build progress.
4. Choose **DeployToECS** to see the deployment progress.

   For more information about viewing run details, see Viewing workflow run status and details.

To verify the deployment

1. Open the Amazon ECS classic console at https://console.aws.amazon.com/ecs/.
2. Choose your cluster, `codecatalyst-ecs-cluster`.
3. Choose the **Tasks** tab.
4. Choose any one of the three tasks.
5. In the **Public IP** field, choose **open address**.

   A "Hello World" page appears in the browser, indicating that the Amazon ECS service successfully deployed your application.

Step 9: Make a change to your source files

In this section, you make a change to the `index.html` file in your source repository. This change causes the workflow to build a new Docker image, tag it with a commit ID, push it to Amazon ECR, and deploy it to Amazon ECS.

To change the index.html

1. In the CodeCatalyst console, in the navigation pane, choose **Code**, then choose **Source repositories**, and then choose your repository, `codecatalyst-ecs-source-repository`.
2. Choose `public-html`, and then choose `index.html`.

   The contents of `index.html` appear.
3. Choose **Edit**.
4. On line 14, change the `Hello World` text to `Tutorial complete!`.
5. Choose **Commit**, and then choose **Commit** again.

   The commit causes a new workflow run to start.
6. (Optional) Go to your source repository's main page, choose **View commits**, and then note the commit ID for the `index.html` change.
7. Watch the deployment progress:

   a. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.

   b. Choose `codecatalyst-ecs-workflow` to view the latest run.

   c. Choose **BuildBackend**, and **DeployToECS** to see the workflow run progress.

8. Verify that your application was updated, as follows:

   a. Open the Amazon ECS classic console at https://console.aws.amazon.com/ecs/.

   b. Choose your cluster, `codecatalyst-ecs-cluster`.

   c. Choose the **Tasks** tab.

   d. Choose any one of the three tasks.

   e. In the **Public IP** field, choose **open address**.

   A `Tutorial complete!` page appears.

9. (Optional) In AWS, switch to the Amazon ECR console and verify that the new Docker image was tagged with the commit ID from step 6.

Clean up

Clean up the files and services used in this tutorial to avoid being charged for them.

In the AWS Management Console, clean up in this order:

1. In Amazon ECS, do the following:
   a. Delete `codecatalyst-ecs-service`.
   b. Delete `codecatalyst-ecs-cluster`.
   c. Deregister `codecatalyst-ecs-task-definition`.
2. In Amazon ECR, delete `codecatalyst-ecs-image-repo`.
3. In Amazon EC2, delete `codecatalyst-ecs-security-group`.
4. In IAM Identity Center, delete:
   a. `CodeCatalystECSUser`
   b. `CodeCatalystECSPermissionSet`

In the CodeCatalyst console, clean up as follows:

1. Delete `codecatalyst-ecs-workflow`.
2. Delete `codecatalyst-ecs-environment`.
3. Delete `codecatalyst-ecs-source-repository`.
4. Delete `codecatalyst-ecs-project`.

In this tutorial, you learned how to deploy an application to an Amazon ECS service using a CodeCatalyst workflow and a **Deploy to Amazon ECS** action.

Adding the "Deploy to Amazon ECS" action

Use the following instructions to add the **Deploy to Amazon ECS** action to your workflow.

Visual

To add the "Deploy to Amazon ECS" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Deploy to Amazon ECS** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **Deploy to Amazon ECS**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **Download** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the "Deploy to Amazon ECS" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.

12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add the "Deploy to Amazon ECS" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Deploy to Amazon ECS** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **Deploy to Amazon ECS**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **Download** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "Deploy to Amazon ECS" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

Variables produced by the "Deploy to Amazon ECS" action

The **Deploy to Amazon ECS** action produces and sets the following variables at run time. These are known as *predefined variables*.

For information about referencing these variables in a workflow, see Using predefined variables.

Key	Value
cluster	The name of the Amazon ECS cluster that was deployed to during the workflow run. Example: `codecatalyst-ecs-cluster`
deployment-platform	The name of the deployment platform. Hardcoded to `AWS:ECS`.
service	The name of the Amazon ECS service that was deployed to during the workflow run. Example: `codecatalyst-ecs-service`
task-definition-arn	The Amazon Resource Name (ARN) of the task definition that was registered during the workflow run. Example: `arn:aws:ecs:us-west-2:111122223333:task-definition/codecatalyst-task-def:8` The `:8` in the preceding example indicates the revision that was registered.
deployment-url	A link to the Amazon ECS console's **Events** tab, where you can view details of the Amazon ECS deployment associated with the workflow run. Example: `https://console.aws.amazon.com/ecs/home?region=us-west-2#/clusters/codecatalyst-ecs-cluster/services/codecatalyst-ecs-service/events`

Key	Value
region	The region code of the AWS Region that was deployed to during the workflow run. Example: `us-west-2`

"Deploy to Amazon ECS" action YAML definition

The following is the YAML definition of the **Deploy to Amazon ECS** action. To learn how to use this action, see Deploying an application to Amazon Elastic Container Service (ECS) with a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  ECSDeployAction_nn:
    Identifier: aws/ecs-deploy@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
```

```
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - task-definition-artifact
    Configuration:
      region: us-east-1
      cluster: ecs-cluster
      service: ecs-service
      task-definition: task-definition-path
      force-new-deployment: false|true
      codedeploy-appspec: app-spec-file-path
      codedeploy-application: application-name
      codedeploy-deployment-group: deployment-group-name
      codedeploy-deployment-description: deployment-description
```

ECSDeployAction

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `ECSDeployAction_nn`.

Corresponding UI: Configuration tab/**Action display name**

Identifier

(*ECSDeployAction*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/ecs-deploy@v1`.

Corresponding UI: Workflow diagram/ECSDeployAction_nn/**aws/ecs-deploy@v1** label

DependsOn

(*ECSDeployAction*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*ECSDeployAction*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*ECSDeployAction*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/Advanced - optional/**Compute type**

Fleet

(*ECSDeployAction*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/Advanced - optional/**Compute fleet**

Timeout

(*ECSDeployAction*/**Timeout**)

(Optional)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Environment

(*ECSDeployAction*/**Environment**)

(Required)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the

default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*ECSDeployAction*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*ECSDeployAction*/Environment/**Connections**)

(Optional in newer versions of the action; required in older versions)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Name

(*ECSDeployAction*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Role

(*ECSDeployAction*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that the **Deploy to Amazon ECS** action uses to access AWS. Make sure that you have added the role to your CodeCatalyst space, and that the role includes the following policies.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

- The following permissions policy:

Warning

Limit the permissions to those shown in the following policy. Using a role with broader permissions might pose a security risk.

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action":[
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs:DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
```

```
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
   ],
   "Resource":"*",
   "Effect":"Allow"
},{"Action":[
      "iam:PassRole"
   ],
   "Effect":"Allow",
   "Resource":"*",
   "Condition":{"StringLike":{"iam:PassedToService":[
            "ecs-tasks.amazonaws.com",
            "codedeploy.amazonaws.com"
         ]
      }
   }
}]
}
```

Note

The first time the role is used, use the following wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

- The following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
```

```
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *`my-environment`*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**Role**

Inputs

(*`ECSDeployAction`*/**Inputs**)

(Optional)

The `Inputs` section defines the data that the `ECSDeployAction` needs during a workflow run.

> **Note**
>
> Only one input (either a source or an artifact) is allowed per **Deploy to Amazon ECS** action.

Corresponding UI: **Inputs** tab

Sources

(*ECSDeployAction*/Inputs/**Sources**)

(Required if your task definition file is stored in a source repository)

If your task definition file is stored in a source repository, specify the label of that source repository. Currently, the only supported label is `WorkflowSource`.

If your task definition file is not contained within a source repository, it must reside in an artifact generated by another action.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*ECSDeployAction*/Inputs/**Artifacts**)

(Required if your task definition file is stored in an output artifact from a previous action)

If the task definition file that you want to deploy is contained in an artifact generated by a previous action, specify that artifact here. If your task definition file is not contained within an artifact, it must reside in your source repository.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Configuration tab/**Artifacts - optional**

Configuration

(*ECSDeployAction*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

region

(Configuration/**region**)

(Required)

Specify the AWS Region where your Amazon ECS cluster and service reside. For a list of Region codes, see Regional endpoints in the *AWS General Reference*.

Corresponding UI: Configuration tab/**Region**

cluster

(*ECSDeployAction*/Configuration/**cluster**)

(Required)

Specify the name of an existing Amazon ECS cluster. The **Deploy to Amazon ECS** action will deploy your containerized application as a task into this cluster. For more information about Amazon ECS clusters, see Clusters in the *Amazon Elastic Container Service Developer Guide*.

Corresponding UI: Configuration tab/**Cluster**

service

(*ECSDeployAction*/Configuration/**service**)

(Required)

Specify the name of an existing Amazon ECS service that will instantiate the task definition file. This service must reside under the cluster specified in the `cluster` field. For more information about Amazon ECS services, see Amazon ECS services in the *Amazon Elastic Container Service Developer Guide*.

Corresponding UI: Configuration tab/**Service**

task-definition

(*ECSDeployAction*/Configuration/**task-definition**)

(Required)

Specify the path to an existing task definition file. If the file resides in your source repository, the path is relative to the source repository root folder. If your file resides in an artifact from a previous workflow action, the path is relative to the artifact root folder. For more information about task definition files, see Task definitions in the *Amazon Elastic Container Service Developer Guide*.

Corresponding UI: Configuration tab/**Task definition**

force-new-deployment

(*ECSDeployAction*/Configuration/**force-new-deployment**)

(Required)

If enabled, the Amazon ECS service is able to start new deployments without service definition changes. Forcing a deployment causes the service to stop all currently running tasks and launch new tasks. For more information about forcing new deployments, see Updating a service in the *Amazon Elastic Container Service Developer Guide*.

Default: `false`

Corresponding UI: Configuration tab/**Force a new deployment of the service**

codedeploy-appspec

(*ECSDeployAction*/Configuration/**codedeploy-appspec**)

(Required if you have configured your Amazon ECS service to use blue/green deployments, otherwise, omit)

Specify the name and path to an existing CodeDeploy application specification (AppSpec) file. This file must reside in the root of your CodeCatalyst source repository. For more information about AppSpec files, see CodeDeploy application specification (AppSpec) files in the *AWS CodeDeploy User Guide*.

> **Note**
>
> Only supply CodeDeploy information if you have configured your Amazon ECS service to perform blue/green deployments. For rolling update deployments (the default), omit CodeDeploy information. For more information about Amazon ECS deployments, see Amazon ECS deployment types in the *Amazon Elastic Container Service Developer Guide*.

> **Note**
>
> The **CodeDeploy** fields may be hidden in the visual editor. To get them to appear, see Why are CodeDeploy fields missing from the visual editor?.

Corresponding UI: Configuration tab/**CodeDeploy AppSpec**

codedeploy-application

(*ECSDeployAction*/Configuration/**codedeploy-application**)

(Required if `codedeploy-appspec` is included)

Specify the name of an existing CodeDeploy application. For more information about CodeDeploy applications, see Working with applications in CodeDeploy in the *AWS CodeDeploy User Guide*.

Corresponding UI: Configuration tab/**CodeDeploy application**

codedeploy-deployment-group

(*ECSDeployAction*/Configuration/**codedeploy-deployment-group**)

(Required if `codedeploy-appspec` is included)

Specify the name of an existing CodeDeploy deployment group. For more information about CodeDeploy deployment groups, see Working with deployment groups in CodeDeploy in the *AWS CodeDeploy User Guide*.

Corresponding UI: Configuration tab/**CodeDeploy deployment group**

codedeploy-deployment-description

(*ECSDeployAction*/Configuration/**codedeploy-deployment-description**)

(Optional)

Specify a description of the deployment that this action will create. For more information, see Working with deployments in CodeDeploy in the *AWS CodeDeploy User Guide*.

Corresponding UI: Configuration tab/**CodeDeploy deployment description**

Deploying an application to Amazon Elastic Kubernetes Service with a workflow

> **Tip**
>
> For a tutorial that shows you how to use the **Deploy to Kubernetes cluster** action, see Tutorial: Deploy an application to Amazon EKS.

This section describes how to deploy a containerized application into a Kubernetes cluster using a CodeCatalyst workflow. To accomplish this, you must add the **Deploy to Kubernetes cluster** action to your workflow. This action deploys your application to a Kubernetes cluster that you have set up in Amazon Elastic Kubernetes Service (EKS) using one or more Kubernetes manifest files. For a sample manifest, see deployment.yaml in Tutorial: Deploy an application to Amazon EKS.

For more information about Kubernetes, see the Kubernetes Documentation.

For more information about Amazon EKS, see What is Amazon EKS? in the *Amazon EKS User Guide*.

How the "Deploy to Kubernetes cluster" action works

The **Deploy to Kubernetes cluster** works as follows:

1. At runtime, the action installs the Kubernetes `kubectl` utility to the CodeCatalyst compute machine where the action is running. The action configures `kubectl` to point to the Amazon EKS cluster you provided when you configured the action. The `kubectl` utility is necessary to run the `kubectl apply` command, next.
2. The action runs the `kubectl apply -f` *`my-manifest.yaml`* command, which carries out the instructions in *`my-manifest.yaml`* to deploy your application as a set of containers and pods into the configured cluster. For more information on this command, see the kubectl apply topic in the *Kubernetes Reference Documentation*.

Topics

- "Deploy to Kubernetes cluster" action YAML definition

Tutorial: Deploy an application to Amazon EKS

In this tutorial, you learn how to deploy a containerized application into Amazon Elastic Kubernetes Service using an Amazon CodeCatalyst workflow, Amazon EKS, and a few other AWS services. The deployed application is a simple 'Hello, World!' website built on an Apache web server Docker image. The tutorial walks you through the required preparation work such as setting up a development machine and an Amazon EKS cluster, and then describes how to create a workflow to build the application and deploy it into the cluster.

After the initial deployment is complete, the tutorial instructs you to make a change to your application source. This change causes a new Docker image to be built and pushed to your Docker image repository with new revision information. The new revision of the Docker image is then deployed into Amazon EKS.

> **Tip**
>
> Instead of working your way through this tutorial, you can use a blueprint that does a complete Amazon EKS setup for you. You'll need to use the **EKS App Deployment** blueprint. For more information, see Creating a project with a blueprint.

Topics

Prerequisites

Before you begin this tutorial:

- You need an Amazon CodeCatalyst **space** with a connected AWS account. For more information, see Creating a space.
- In your space, you need an empty, **Start from scratch** CodeCatalyst **project** called:

  ```
  codecatalyst-eks-project
  ```

  For more information, see Creating an empty project in Amazon CodeCatalyst.
- In your project, you need an empty CodeCatalyst **source repository** called:

  ```
  codecatalyst-eks-source-repository
  ```

  For more information, see Store and collaborate on code with source repositories in CodeCatalyst.
- In your project, you need a CodeCatalyst CI/CD **environment** (not a Dev Environment) called:

  ```
  codecatalyst-eks-environment
  ```

  Configure this environment as follows:
  - Choose any type, such as **Non-production**.
  - Connect your AWS account to it.
  - For the **Default IAM role**, choose any role. You'll specify a different role later.

  For more information, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.

Step 1: Set up your development machine

The first step in this tutorial is to configure a development machine with a few tools that you'll use throughout this tutorial. These tools are:

- the `eksctl` utility – for cluster creation
- the `kubectl` utility – a prerequisite for `eksctl`
- the AWS CLI – also a prerequisite for `eksctl`

You can install these tools on your existing development machine if you have one, or you can use a CodeCatalyst Dev Environment, which is Cloud-based. The benefit of a CodeCatalyst Dev Environment is that it's easy to spin up and take down, and is integrated with other CodeCatalyst services, allowing you to work through this tutorial in fewer steps.

This tutorial assumes you'll be using a CodeCatalyst Dev Environment.

The following instructions describe a quick way to launch a CodeCatalyst Dev Environment and configure it with the required tools, but if you want detailed instructions, see:

- Creating a Dev Environment in this guide.
- Installing kubectl in the Amazon EKS User Guide.
- Installing or upgrading eksctl in the Amazon EKS User Guide.
- Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

To launch a Dev Environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, `codecatalyst-eks-project`.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose the name of your source repository, `codecatalyst-eks-source-repository`.
5. Near the top choose **Create Dev Environment**, and then choose **AWS Cloud9 (in browser)**.
6. Make sure that **Work in existing branch** and **main** are selected, and then choose **Create**.

   Your Dev Environment launches in a new browser tab, and your repository (`codecatalyst-eks-source-repository`) is cloned into it.

To install and configure kubectl

1. In the Dev Environment terminal, enter:

   ```
   curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/bin/linux/amd64/kubectl
   ```

2. Enter:

```
chmod +x ./kubectl
```

3. Enter:

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
```

4. Enter:

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

5. Enter:

```
kubectl version --short --client
```

6. Check that a version appears.

   You have now installed `kubectl`.

To install and configure eksctl

> **Note**
>
> `eksctl` is not strictly required because you can use `kubectl` instead. However, `eksctl` has the benefit of automating much of the cluster configuration, and is therefore the tool recommended for this tutorial.

1. In the Dev Environment terminal, enter:

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/
download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

2. Enter:

```
sudo cp /tmp/eksctl /usr/bin
```

3. Enter:

```
eksctl version
```

4. Check that a version appears.

   You have now installed `eksctl`.

To verify that the AWS CLI is installed

1. In the Dev Environment terminal, enter:

   ```
   aws --version
   ```

2. Check that a version appears to verify that the AWS CLI is installed.

   Complete the remaining procedures to configure the AWS CLI with the necessary permissions to access AWS.

To configure the AWS CLI

You must configure the AWS CLI with access keys and a session token to give it access to AWS services. The following instructions provide a quick way to configure the keys and token, but if you want detailed instructions, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

1. Create an IAM Identity Center user, as follows:

   a. Sign in to the AWS Management Console and open the AWS IAM Identity Center console at https://console.aws.amazon.com/singlesignon/.

      (You might need to choose **Enable** if you've never signed in to IAM Identity Center before.)

      > **Note**
      >
      > Make sure you sign in using the AWS account that is connected to your CodeCatalyst space. You can verify which account is connected by navigating to your space and choosing the **AWS accounts** tab. For more information, see Creating a space.

   b. In the navigation pane, choose **Users**, and then choose **Add user**.

   c. In **Username**, enter:

```
codecatalyst-eks-user
```

d. Under **Password**, choose **Generate a one-time password that you can share with this user**.

e. In **Email address** and **Confirm email address**, enter an email address that doesn't already exist in IAM Identity Center.

f. In **First name**, enter:

```
codecatalyst-eks-user
```

g. In **Last name**, enter:

```
codecatalyst-eks-user
```

h. In **Display name**, keep:

```
codecatalyst-eks-user codecatalyst-eks-user
```

i. Choose **Next**.

j. On the **Add user to groups** page, choose **Next**.

k. On the **Review and add user** page, review the information and choose **Add user**.

A **One-time password** dialog box appears.

l. Choose **Copy** and then paste the sign-in information to a text file. The sign-in information consists of the AWS access portal URL, a user name, and a one-time password.

m. Choose **Close**.

2. Create a permission set, as follows:

a. In the navigation pane, choose **Permission sets**, and then choose **Create permission set**.

b. Choose **Predefined permission set** and then select **AdministratorAccess**. This policy provides full permissions to all AWS services.

c. Choose **Next**.

d. In **Permission set name**, remove `AdministratorAccess` and enter:

```
codecatalyst-eks-permission-set
```

e. Choose **Next**.

f. On the **Review and create** page, review the information and choose **Create**.

3. Assign the permission set to `codecatalyst-eks-user`, as follows:

   a. In the navigation pane, choose **AWS accounts**, and then select the check box next to the AWS account that you're currently signed in to.

   b. Choose **Assign users or groups**.

   c. Choose the **Users** tab.

   d. Select the check box next to `codecatalyst-eks-user`.

   e. Choose **Next**.

   f. Select the check box next to `codecatalyst-eks-permission-set`.

   g. Choose **Next**.

   h. Review the information and choose **Submit**.

   You have now assigned `codecatalyst-eks-user` and `codecatalyst-eks-permission-set` to your AWS account, binding them together.

4. Obtain `codecatalyst-eks-user`'s access keys and session token, as follows:

   a. Make sure you have the AWS access portal URL and the username and one-time password for `codecatalyst-eks-user`. You should have copied this information to a text editor earlier.

   > **Note**
   >
   > If you do not have this information, go to the `codecatalyst-eks-user` details page in IAM Identity Center, choose **Reset password**, **Generate a one-time password [...]**, and **Reset password** again to display the information on the screen.

   b. Sign out of AWS.

   c. Paste the AWS access portal URL into your browser's address bar.

   d. Sign in with:

   - **Username**:

     ```
     codecatalyst-eks-user
     ```

   - **Password**:

one-time-password

e. In **Set new password**, enter a new password and choose **Set new password**.

An **AWS account** box appears on the screen.

f. Choose **AWS account**, and then choose the name of the AWS account to which you assigned the `codecatalyst-eks-user` user and permission set.

g. Next to `codecatalyst-eks-permission-set`, choose **Command line or programmatic access**.

h. Copy the commands in the middle of the page. They look similar to the following:

```
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
export AWS_SESSION_TOKEN="session-token"
```

...where *session-token* is a long random string.

5. Add the access keys and session token to the AWS CLI, as follows:

a. Return to your CodeCatalyst Dev Environment.

b. At the terminal prompt, paste the commands you copied. Press Enter.

You have now configured the AWS CLI with access keys and a session token. You can now use AWS CLI to complete the tasks required by this tutorial.

> **Important**
>
> If at any time during this tutorial you see messages similar to:
> `Unable to locate credentials. You can configure credentials by running "aws configure".`
> Or:
> `ExpiredToken: The security token included in the request is expired`
> ...it's because your AWS CLI session has expired. In this case, do *not* run the `aws configure` command. Instead, use the instructions in step 4 of this procedure that starts with `Obtain codecatalyst-eks-user's access key and session token` to refresh your session.

Step 2: Create an Amazon EKS cluster

In this section, you create a cluster in Amazon EKS. The instructions below describe a quick way to create the cluster using `eksctl`, but if you want detailed instructions, see:

- Getting started with eksctl in the Amazon EKS User Guide

  or

- Getting started with the console and AWS CLI in the Amazon EKS User Guide (this topic provides `kubectl` instructions for creating the cluster)

> **Note**
>
> Private clusters are not supported by the CodeCatalyst integration with Amazon EKS.

Before you begin

Make sure you have completed the following tasks on your development machine:

- Installed the `eksctl` utility.
- Installed the `kubectl` utility.
- Installed the AWS CLI and configured it with access keys and a session token.

For information on how to complete these tasks, see Step 1: Set up your development machine.

To create a cluster

> **Important**
>
> Do not use the Amazon EKS service's user interface to create the cluster because the cluster won't be configured correctly. Use the `eksctl` utility, as described in the following steps.

1. Go to your Dev Environment.
2. Create a cluster and nodes:

```
eksctl create cluster --name codecatalyst-eks-cluster --region us-west-2
```

Where:

- *codecatalyst-eks-cluster* is replaced with the name you want to give your cluster.
- *us-west-2* is replaced with your Region.

After 10-20 minutes, a message similar to the following appears:

```
EKS cluster "codecatalyst-eks-cluster" in "us-west-2" region is ready
```

> **Note**
>
> You will see multiple `waiting for CloudFormation stack` messages while AWS creates your cluster. This is expected.

3. Verify that your cluster was created successfully:

```
kubectl cluster-info
```

You will see a message similar to the following, indicating a sucessful cluster creation:

```
Kubernetes master is running at https://long-string.gr7.us-west-2.eks.amazonaws.com
CoreDNS is running at https://long-string.gr7.us-west-2.eks.amazonaws.com/api/v1/
namespaces/kube-system/services/kube-dns:dns/proxy
```

Step 3: Create an Amazon ECR image repository

In this section, you create a private image repository in Amazon Elastic Container Registry (Amazon ECR). This repository stores the Docker image for the tutorial.

For more information about Amazon ECR, see the *Amazon Elastic Container Registry User Guide*.

To create an image repository in Amazon ECR

1. Go to your Dev Environment.
2. Create an empty repository in Amazon ECR:

```
aws ecr create-repository --repository-name codecatalyst-eks-image-repo
```

Replace *codecatalyst-eks-image-repo* with the name you want to give the Amazon ECR repository.

This tutorial assumes you named your repository codecatalyst-eks-image-repo.

3. Display the Amazon ECR repository's details:

```
aws ecr describe-repositories \
      --repository-names codecatalyst-eks-image-repo
```

4. Note the "repositoryUri": value, for example, 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-image-repo.

   You need it later when adding the repository to your workflow.

Step 4: Add source files

In this section, you add application source files to your source repository (codecatalyst-eks-source-repository). They consist of:

- An index.html file – Displays a 'Hello, World!' message in the browser.
- A Dockerfile – Describes the base image to use for your Docker image and the Docker commands to apply to it.
- A deployment.yaml file – The Kubernetes manifest that defines the Kubernetes service and deployment.

The folder structure is as follows:

```
|– codecatalyst-eks-source-repository
   |– Kubernetes
      |– deployment.yaml
   |– public-html
   |  |– index.html
   |– Dockerfile
```

Topics

- index.html
- Dockerfile

- deployment.yaml

index.html

The `index.html` file displays a 'Hello, World!' message in the browser.

To add the index.html file

1. Go to your Dev Environment.
2. In `codecatalyst-eks-source-repository`, create a folder called `public-html`.
3. In `/public-html`, create a file called `index.html` with the following contents:

```
<html>
  <head>
    <title>Hello World</title>
    <style>
      body {
      background-color: black;
      text-align: center;
      color: white;
      font-family: Arial, Helvetica, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

4. At the terminal prompt, enter:

```
cd /projects/codecatalyst-eks-source-repository
```

5. Add, commit, and push:

```
git add .
git commit -m "add public-html/index.html"
git push
```

   The `index.html` is added to your repository in a `public-html` folder.

Dockerfile

The Dockerfile describes the base Docker image to use and the Docker commands to apply to it. For more information about the Dockerfile, see the Dockerfile Reference.

The Dockerfile specified here indicates to use the Apache 2.4 base image (`httpd`). It also includes instructions for copying a source file called `index.html` to a folder on the Apache server that serves webpages. The `EXPOSE` instruction in the Dockerfile tells Docker that the container is listening on port 80.

To add the Dockerfile

1. In `codecatalyst-eks-source-repository`, create a file called `Dockerfile` with the following contents:

   ```
   FROM httpd:2.4
   COPY ./public-html/index.html /usr/local/apache2/htdocs/index.html
   EXPOSE 80
   ```

   Do not include a file extension.

   > **Important**
   >
   > The Dockerfile must reside in your repository's root folder. The workflow's `Docker build` command expects it to be there.

2. Add, commit, and push:

   ```
   git add .
   git commit -m "add Dockerfile"
   git push
   ```

   The Dockerfile is added to your repository.

deployment.yaml

In this section, you add a `deployment.yaml` file to your repository. The `deployment.yaml` file is a Kubernetes manifest that defines two Kubernetes resources types or *kinds* to run: a 'service' and a 'deployment'.

- The 'service' deploys a load balancer into Amazon EC2. The load balancer provides you with an Internet-facing public URL and standard port (port 80) that you can use to browse to the 'Hello, World!' application.
- The 'deployment' deploys three pods, and each pod will contain a Docker container with the 'Hello, World!' application. The three pods are deployed onto the nodes that were created when you created the cluster.

The manifest in this tutorial is short; however, a manifest can include any number of Kubernetes resource types, such as pods, jobs, ingresses, and network policies. Further, you can use multiple manifest files if your deployment is complex.

To add a deployment.yaml file

1. In `codecatalyst-eks-source-repository`, create a folder called `Kubernetes`.
2. In `/Kubernetes`, create a file called `deployment.yaml` with the following contents:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    app: my-app
spec:
  type: LoadBalancer
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: codecatalyst-eks-container
        # The $REPOSITORY_URI and $IMAGE_TAG placeholders will be replaced by
 actual values supplied by the build action in your workflow
        image: $REPOSITORY_URI:$IMAGE_TAG
        ports:
        - containerPort: 80
```

3. Add, commit, and push:

```
git add .
git commit -m "add Kubernetes/deployment.yaml"
git push
```

The `deployment.yaml` file is added to your repository in a folder called `Kubernetes`.

You have now added all your source files.

Take a moment to double-check your work and make sure you placed all the files in the correct folders. The folder structure is as follows:

```
|— codecatalyst-eks-source-repository
   |— Kubernetes
      |— deployment.yaml
   |— public-html
   |  |— index.html
   |— Dockerfile
```

Step 5: Create AWS roles

In this section, you create AWS IAM roles that your CodeCatalyst workflow will need in order to function. These roles are:

- **Build role** – Grants the CodeCatalyst build action (in the workflow) permission to access your AWS account and write to Amazon ECR and Amazon EC2.

- **Deploy role** – Grants the CodeCatalyst **Deploy to Kubernetes cluster** action (in the workflow) permission to access your AWS account and Amazon EKS.

For more information about IAM roles, see IAM roles in the *AWS Identity and Access Management User Guide*.

> **Note**
>
> To save time, you can create a single role, called the `CodeCatalystWorkflowDevelopmentRole-spaceName` role, instead of the two roles listed previously. For more information, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-spaceName` role has very broad permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern. This tutorial assumes you are creating the two roles listed previously.

To create the build and deploy roles, complete the following series of procedures.

1. To create a trust policy for both roles

1. Go to your Dev Environment.
2. In the `Cloud9-long-string` directory, create a file called `codecatalyst-eks-trust-policy.json` with the following contents:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
```

```
        }
    ]
}
```

2. To create the build policy for the build role

- In the `Cloud9-`*`long-string`* directory, create a file called `codecatalyst-eks-build-policy.json` with the following contents:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

3. To create the deploy policy for the deploy role

- In the `Cloud9-`*`long-string`* directory, create a file called `codecatalyst-eks-deploy-policy.json` with the following contents:

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

You have now added three policy documents to your Dev Environment. Your directory structure now looks like this:

```
|— Cloud9-long-string
   |— .c9
   |— codecatalyst-eks-source-repository
      |— Kubernetes
      |— public-html
      |— Dockerfile
   codecatalyst-eks-build-policy.json
   codecatalyst-eks-deploy-policy.json
   codecatalyst-eks-trust-policy.json
```

4. To add the build policy to AWS

1. In the Dev Environment terminal, enter:

```
cd /projects
```

2. Enter:

```
aws iam create-policy \
    --policy-name codecatalyst-eks-build-policy \
    --policy-document file://codecatalyst-eks-build-policy.json
```

3. Press **Enter**.
4. In the command output, note the `"arn":` value, for example, `arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy`. You need this ARN later.

5. To add the deploy policy to AWS

1. Enter:

```
aws iam create-policy \
    --policy-name codecatalyst-eks-deploy-policy \
    --policy-document file://codecatalyst-eks-deploy-policy.json
```

2. Press **Enter**.
3. In the command output, note the deploy policy's `"arn":` value, for example, `arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy`. You need this ARN later.

6. To create the build role

1. Enter:

```
aws iam create-role \
      --role-name codecatalyst-eks-build-role \
      --assume-role-policy-document file://codecatalyst-eks-trust-policy.json
```

2. Press **Enter**.
3. Enter:

```
aws iam attach-role-policy \
      --role-name codecatalyst-eks-build-role \
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy
```

Where *arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy* is replaced with the ARN of the build policy you noted earlier.

4. Press **Enter**.
5. At the terminal prompt, enter:

```
aws iam get-role \
      --role-name codecatalyst-eks-build-role
```

6. Press **Enter**.
7. Note the role's `"Arn":` value, for example, `arn:aws:iam::111122223333:role/codecatalyst-eks-build-role`. You need this ARN later.

7. To create the deploy role

1. Enter:

```
aws iam create-role \
      --role-name codecatalyst-eks-deploy-role \
      --assume-role-policy-document file://codecatalyst-eks-trust-policy.json
```

2. Press **Enter**.
3. Enter:

```
aws iam attach-role-policy \
      --role-name codecatalyst-eks-deploy-role \
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy
```

Where *arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy* is replaced with the ARN of the deploy policy you noted earlier.

4. Press **Enter**.
5. Enter:

```
aws iam get-role \
      --role-name codecatalyst-eks-deploy-role
```

6. Press **Enter**.

7. Note the role's `"Arn":` value, for example, `arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role`. You need this ARN later.

You have now created build and deploy roles and noted their ARNs.

Step 6: Add AWS roles to CodeCatalyst

In this step, you add the build role (`codecatalyst-eks-build-role`) and deploy role (`codecatalyst-eks-deploy-role`) to the AWS account that you connected to your space. This makes the roles available for use in your workflow.

To add build and deploy roles to your AWS account

1. In the CodeCatalyst console, navigate to your space.
2. At the top, choose **Settings**.
3. In the navigation pane, choose **AWS accounts**. A list of accounts appears.
4. In the **Amazon CodeCatalyst display name** column, copy the display name of the AWS account where you created your build and deploy roles. (It might be a number.) You'll need this value later, when creating your workflow.
5. Choose the display name.
6. Choose **Manage roles from AWS management console**.

   The **Add IAM role to Amazon CodeCatalyst space** page appears. You might need to sign in to access the page.
7. Select **Add an existing role you have created in IAM**.

   A drop-down list appears. The list displays the build and deploy roles, and any other IAM roles with a trust policy that includes the `codecatalyst-runner.amazonaws.com` and `codecatalyst.amazonaws.com` service principals.
8. From the drop-down list, add:

   - `codecatalyst-eks-build-role`
   - `codecatalyst-eks-deploy-role`

> **Note**
>
> If you see `The security token included in the request is invalid`, it might be because you do not have the right permissions. To fix this issue, sign out of AWS as sign back in with the AWS account that you used when you created your CodeCatalyst space.

9. Return to the CodeCatalyst console and refresh the page.

   The build and deploy roles should now appear under **IAM roles**.

   These roles are now available for use in CodeCatalyst workflows.

Step 7: Update the ConfigMap

You must add the deploy role that you created in Step 5: Create AWS roles to the Kubernetes `ConfigMap` file to give the **Deploy to Kubernetes cluster** action (in your workflow) the ability to access and interact with your cluster. You can use `eksctl` or `kubectl` to perform this task.

To configure the Kubernetes ConfigMap file using eksctl

- In the Dev Environment terminal, enter:

```
eksctl create iamidentitymapping --cluster codecatalyst-eks-cluster --
arn arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role --group
 system:masters --username codecatalyst-eks-deploy-role --region us-west-2
```

  Where:

  - *`codecatalyst-eks-cluster`* is replaced with the cluster name of the Amazon EKS cluster.
  - *`arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role`* is replaced with the ARN of the deploy role that you created in Step 5: Create AWS roles.
  - *`codecatalyst-eks-deploy-role`* (next to `--username`) is replaced with the name of the deploy role that you created in Step 5: Create AWS roles.

> **Note**
>
> If you decided not to create a deploy role, replace *codecatalyst-eks-deploy-role* with the name of the CodeCatalystWorkflowDevelopmentRole-*spaceName* role. For more information about this role, see Step 5: Create AWS roles.

- *us-west-2* is replaced with your Region.

For details on this command, see Manage IAM users and roles.

A message similar to the following appears:

```
2023-06-09 00:58:29 [#]  checking arn arn:aws:iam::111122223333:role/codecatalyst-
eks-deploy-role against entries in the auth ConfigMap
2023-06-09 00:58:29 [#]  adding identity "arn:aws:iam::111122223333:role/
codecatalyst-eks-deploy-role" to auth ConfigMap
```

To configure the Kubernetes ConfigMap file using kubectl

1. In the Dev Environment terminal, enter:

```
kubectl edit configmap -n kube-system aws-auth
```

The ConfigMap file appears on the screen.

2. Add the text in red italics:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
 will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
```

```
      rolearn: arn:aws:iam::111122223333:role/eksctl-codecatalyst-eks-cluster-n-
NodeInstanceRole-16BC456ME6YR5
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:masters
      rolearn: arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role
      username: codecatalyst-eks-deploy-role
  mapUsers: |
    []
kind: ConfigMap
metadata:
  creationTimestamp: "2023-06-08T19:04:39Z"
  managedFields:
  ...
```

Where:

- *arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role* is replaced with the ARN of the deploy role that you created in Step 5: Create AWS roles.
- *codecatalyst-eks-deploy-role* (next to `username:`)is replaced with the name of the deploy role that you created in Step 5: Create AWS roles.

> **Note**
>
> If you decided not to create a deploy role, replace *codecatalyst-eks-deploy-role* with the name of the `CodeCatalystWorkflowDevelopmentRole-`*spaceName* role. For more information about this role, see Step 5: Create AWS roles.

For details, see Enabling IAM principal access to your cluster in the Amazon EKS User Guide.

You have now given the deploy role, and by extension the **Deploy to Amazon EKS** action, `system:masters` permissions to your Kubernetes cluster.

Step 8: Create and run a workflow

In this step, you create a workflow that takes your source files, builds them into a Docker image, and then deploys the image into tree pods in your Amazon EKS cluster.

The workflow consists of the following building blocks that run sequentially:

- A trigger – This trigger starts the workflow run automatically when you push a change to your source repository. For more information about triggers, see Starting a workflow run automatically with triggers.
- A build action (`BuildBackend`) – On trigger, the action builds the Docker image using the Dockerfile and pushes the image to Amazon ECR. The build action also updates the `$REPOSITORY_URI` and `$IMAGE_TAG` variables in the `deployment.yaml` file with the correct values, and then creates an output artifact of this file and any others in the `Kubernetes` folder. In this tutorial, the only file in the `Kubernetes` folder is `deployment.yaml` but you could include more files. The artifact is used as the input for the deploy action, which is next.

  For more information about the build action, see Building with workflows.
- A deploy action (`DeployToEKS`) – On completion of the build action, the deploy action looks for the output artifact generated by the build action (`Manifests`), and finds the `deployment.yaml` file inside of it. The action then follows the instructions in the `deployment.yaml` file to run three pods—each containing a single 'Hello, World!' Docker container—inside your Amazon EKS cluster.

To create a workflow

1. Go to the CodeCatalyst console.
2. Navigate to your project (`codecatalyst-eks-project`).
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose **Create workflow**.
5. For **Source repository**, choose `codecatalyst-eks-source-repository`.
6. For **Branch**, choose `main`.
7. Choose **Create**.
8. Delete the YAML sample code.
9. Add the following YAML code to create a new workflow definition file:

> **Note**
>
> For more information about the workflow definition file, see Workflow YAML definition.

> **Note**
>
> In the YAML code that follows, you can omit the `Connections:` sections if you want. If you omit these sections, you must ensure that the role specified in the **Default IAM role** field in your environment includes the permissions and trust policies of both roles described in Step 6: Add AWS roles to CodeCatalyst. For more information about setting up an environment with a default IAM role, see Creating an environment.

```
Name: codecatalyst-eks-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildBackend:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-eks-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-eks-build-role
    Inputs:
      Sources:
        - WorkflowSource
      Variables:
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-
image-repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      Steps:
        #pre_build:
        - Run: echo Logging in to Amazon ECR...
        - Run: aws --version
        - Run: aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
```

```
        #build:
        - Run: echo Build started on `date`
        - Run: echo Building the Docker image...
        - Run: docker build -t $REPOSITORY_URI:latest .
        - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
        #post_build:
        - Run: echo Build completed on `date`
        - Run: echo Pushing the Docker images...
        - Run: docker push $REPOSITORY_URI:latest
        - Run: docker push $REPOSITORY_URI:$IMAGE_TAG
        # Replace the variables in deployment.yaml
        - Run: find Kubernetes/ -type f | xargs sed -i "s|\$REPOSITORY_URI|
$REPOSITORY_URI|g"
        - Run: find Kubernetes/ -type f | xargs sed -i "s|\$IMAGE_TAG|$IMAGE_TAG|g"
        - Run: cat Kubernetes/*
        # The output artifact will be a zip file that contains Kubernetes manifest
 files.
    Outputs:
      Artifacts:
        - Name: Manifests
          Files:
            - "Kubernetes/*"
  DeployToEKS:
    DependsOn:
      - BuildBackend
    Identifier: aws/kubernetes-deploy@v1
    Environment:
      Name: codecatalyst-eks-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-eks-deploy-role
    Inputs:
      Artifacts:
        - Manifests
    Configuration:
      Namespace: default
      Region: us-west-2
      Cluster: codecatalyst-eks-cluster
      Manifests: Kubernetes/
```

In the preceding code, replace:

- Both instances of *codecatalyst-eks-environment* with the name of the environment you created in Prerequisites.
- Both instances of *codecatalyst-account-connection* with the display name of your account connection. The display name might be a number. For more information, see Step 6: Add AWS roles to CodeCatalyst.
- *codecatalyst-eks-build-role* with the name of the build role you created in Step 5: Create AWS roles.
- *111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-image-repo* (in the `Value:` property) with the URI of the Amazon ECR repository you created in Step 3: Create an Amazon ECR image repository.
- *111122223333.dkr.ecr.us-west-2.amazonaws.com* (in the `Run: aws ecr` command) with the URI of the Amazon ECR repository without the image suffix (`/codecatalyst-eks-image-repo`).
- *codecatalyst-eks-deploy-role* with the name of the deploy role you created in Step 5: Create AWS roles.
- Both instances of *us-west-2* with your AWS Region code. For a list of Region codes, see Regional endpoints in the *AWS General Reference*.

> **Note**
>
> If you decided not to create build and deploy roles, replace *codecatalyst-eks-build-role* and *codecatalyst-eks-deploy-role* with the name of the `CodeCatalystWorkflowDevelopmentRole-`*spaceName* role. For more information about this role, see Step 5: Create AWS roles.

10. (Optional) Choose **Validate** to make sure that the YAML code is valid before committing.
11. Choose **Commit**.
12. In the **Commit workflow** dialog box, enter the following:

    a. For **Commit message**, remove the text and enter:

    ```
    Add first workflow
    ```

    b. For **Repository**, choose `codecatalyst-eks-source-repository`.

c. For **Branch name**, choose main.

d. Choose **Commit**.

You have now created a workflow. A workflow run starts automatically because of the trigger defined at the top of the workflow. Specifically, when you committed (and pushed) the `workflow.yaml` file to your source repository, the trigger started the workflow run.

To view the workflow run progress

1. In the navigation pane of the CodeCatalyst console, choose **CI/CD**, and then choose **Workflows**.
2. Choose the workflow you just created, `codecatalyst-eks-workflow`.
3. Choose **BuildBackend** to see the build progress.
4. Choose **DeployToEKS** to see the deployment progress.

   For more information about viewing run details, see Viewing workflow run status and details.

To verify the deployment

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the left, near the bottom, choose **Load Balancers**.
3. Select the load balancer that was created as part of your Kubernetes deployment. If you're not sure which load balancer to choose, look for the following tags under the **Tags** tab:
   - `kubernetes.io/service-name`
   - `kubernetes.io/cluster/ekstutorialcluster`
4. With the correct load balancer selected, choose the **Description** tab.
5. Copy and paste the **DNS name** value into your browser's address bar.

   The 'Hello, World!' webpage appears in your browser, indicating that you successfully deployed your application.

Step 9: Make a change to your source files

In this section, you make a change to the `index.html` file in your source repository. This change causes the workflow to build a new Docker image, tag it with a commit ID, push it to Amazon ECR, and deploy it to Amazon ECS.

To change the index.html

1. Go to your Dev Environment.
2. At the terminal prompt, change to your source repository:

   ```
   cd /projects/codecatalyst-eks-source-repository
   ```

3. Pull the latest workflow changes:

   ```
   git pull
   ```

4. Open `codecatalyst-eks-source-repository/public-html/index.html`.
5. On line 14, change the `Hello, World!` text to `Tutorial complete!`.
6. Add, commit, and push:

   ```
   git add .
   git commit -m "update index.html title"
   git push
   ```

   A workflow run starts automatically.
7. (Optional) Enter:

   ```
   git show HEAD
   ```

   Note the commit ID for the `index.html` change. This commit ID will be tagged to the Docker image that will be deployed by the workflow run that you just started.
8. Watch the deployment progress:

   a. In the CodeCatalyst console, in the navigation pane, choose **CI/CD**, and then choose **Workflows**.

   b. Choose `codecatalyst-eks-workflow` to view the latest run.

   c. Choose **BuildBackend**, and **DeployToEKS** to see the workflow run progress.

9. Verify that your application was updated, as follows:

    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

    b. On the left, near the bottom, choose **Load Balancers**.

    c. Select the load balancer that was created as part of your Kubernetes deployment.

    d. Copy and paste the **DNS name** value into your browser's address bar.

    The 'Tutorial Complete!' webpage appears in your browser, indicating that you successfully deployed a new revision of your application.

10. (Optional) In AWS, switch to the Amazon ECR console and verify that the new Docker image was tagged with the commit ID from step 7 of this procedure.

Clean up

You should clean up your environment so that you're not charged unnecessarily for the storage and compute resources used by this tutorial.

To clean up

1. Delete your cluster:

    - In the Dev Environment terminal, enter:

    ```
    eksctl delete cluster --region=us-west-2 --name=codecatalyst-eks-cluster
    ```

    Where:

    - *us-west-2* is replaced with your Region.
    - *codecatalyst-eks-cluster* is replaced with the name of the cluster you created.

    After 5-10 minutes, the cluster and associated resources are deleted, including but not limited to AWS CloudFormation stacks, nodes groups (in Amazon EC2), and load balancers.

> **Important**
>
> If the `eksctl delete cluster` command doesn't work, you may need to refresh your AWS credentials or your `kubectl` credentials. If you're not sure which credentials

to refresh, refresh the AWS credentials first. To refresh your AWS credentials, see How do I fix "Unable to locate credentials" and "ExpiredToken" errors?. To refresh your `kubectl` credentials, see How do I fix "Unable to connect to the server" errors?.

2. In the AWS console, clean up as follows:
    1. In Amazon ECR, delete `codecatalyst-eks-image-repo`.
    2. In IAM Identity Center, delete:
        a. `codecatalyst-eks-user`
        b. `codecatalyst-eks-permission-set`
    3. In IAM, delete:
        - `codecatalyst-eks-build-role`
        - `codecatalyst-eks-deploy-role`
        - `codecatalyst-eks-build-policy`
        - `codecatalyst-eks-deploy-policy`
3. In the CodeCatalyst console, clean up as follows:
    1. Delete `codecatalyst-eks-workflow`.
    2. Delete `codecatalyst-eks-environment`.
    3. Delete `codecatalyst-eks-source-repository`.
    4. Delete your Dev Environment.
    5. Delete `codecatalyst-eks-project`.

In this tutorial, you learned how to deploy an application to an Amazon EKS service using a CodeCatalyst workflow and a **Deploy to Kubernetes cluster** action.

Adding the "Deploy to Kubernetes cluster" action

Use the following instructions to add the **Deploy to Kubernetes cluster** action to your workflow.

Before you begin

Before you add the **Deploy to Kubernetes cluster** action to your workflow, you must have the following prepared:

> **Tip**
>
> To set up these prerequisites quickly, follow the instructions in Tutorial: Deploy an application to Amazon EKS.

- A Kubernetes cluster in Amazon EKS. For information about clusters, see Amazon EKS clusters in the Amazon EKS User Guide.
- At least one Dockerfile that describes how to assemble your application into a Docker image. For more information about Dockerfiles, see the Dockerfile reference.
- At least one Kubernetes manifest file, which is called a *configuration file* or *configuration* in the Kubernetes documentation. For more information, see Managing resources in the Kubernetes documentation.
- An IAM role that gives the **Deploy to Kubernetes cluster** action the ability to access and interact with your Amazon EKS cluster. For more information, see the Role topic in the "Deploy to Kubernetes cluster" action YAML definition.

  After creating this role, you must add it to:

  - Your Kubernetes ConfigMap file. To learn how to add a role to a ConfigMap file, see Enabling IAM principal access to your cluster in the Amazon EKS User Guide.
  - CodeCatalyst. To learn how to add an IAM role to CodeCatalyst, see Adding IAM roles to account connections.
- A CodeCatalyst space, project, and environment. The space and environment must both be connected to the AWS account into which you will be deploying your application. For more information, see Creating a space, Creating an empty project in Amazon CodeCatalyst, and Deploying into AWS accounts and VPCs with CodeCatalyst environments.
- A source repository supported by CodeCatalyst. The repository stores your application source files, Dockerfiles, and Kubernetes manifests. For more information, see Store and collaborate on code with source repositories in CodeCatalyst.

Visual

To add the "Deploy to Kubernetes cluster" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.

3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Deploy to Kubernetes cluster** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **Deploy to Kubernetes cluster**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **Download** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the "Deploy to Kubernetes cluster" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add the "Deploy to Kubernetes cluster" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Deploy to Kubernetes cluster** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **Deploy to Kubernetes cluster**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **Download** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "Deploy to Kubernetes cluster" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

Variables produced by the "Deploy to Kubernetes cluster" action

The **Deploy to Kubernetes cluster** action produces and sets the following variables at run time. These are known as *predefined variables*.

For information about referencing these variables in a workflow, see Using predefined variables.

Key	Value
cluster	The Amazon.com Resource Name (ARN) of the of the Amazon EKS cluster that was deployed to during the workflow run. Example: `arn:aws:eks:us-west-2:111122223333:cluster/codecatalyst-eks-cluster`

Key	Value
deployment-platform	The name of the deployment platform. Hardcoded to `AWS:EKS`.
metadata	Reserved. JSON-formatted metadata related to the cluster deployed during the workflow run.
namespace	The Kubernetes namespace into which the cluster was deployed. Example: `default`
resources	Reserved. JSON-formatted metadata related to the resources deployed during the workflow run.
server	The name of the API server endpoint that you can use to communicate with your cluster using management tools such as `kubectl`. For more information about the API service endpoint, see Amazon EKS cluster endpoint access control in the Amazon EKS User Guide. Example: `https://random-string .gr7.us-west-2.eks.amazonaws.com`

"Deploy to Kubernetes cluster" action YAML definition

The following is the YAML definition of the **Deploy to Kubernetes cluster** action. To learn how to use this action, see Deploying an application to Amazon Elastic Kubernetes Service with a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

Note

Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  DeployToKubernetesCluster_nn:
    Identifier: aws/kubernetes-deploy@v1
    DependsOn:
      - build-action
    Compute:
        - Type: EC2 | Lambda
        - Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: DeployToEKS
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - manifest-artifact
    Configuration:
      Namespace: namespace
      Region: us-east-1
      Cluster: eks-cluster
      Manifests: manifest-path
```

DeployToKubernetesCluster

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `DeployToKubernetesCluster_nn`.

Corresponding UI: Configuration tab/**Action display name**

Identifier

(*`DeployToKubernetesCluster`*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/kubernetes-deploy@v1`.

Corresponding UI: Workflow diagram/DeployToKubernetesCluster_nn/**aws/kubernetes-deploy@v1** label

DependsOn

(*`DeployToKubernetesCluster`*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*`DeployToKubernetesCluster`*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*DeployToKubernetesCluster*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/Advanced - optional/**Compute type**

Fleet

(*DeployToKubernetesCluster*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/Advanced - optional/**Compute fleet**

Timeout

(*DeployToKubernetesCluster*/**Timeout**)

(Optional)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Environment

(*DeployToKubernetesCluster*/**Environment**)

(Required)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*DeployToKubernetesCluster*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*DeployToKubernetesCluster*/Environment/**Connections**)

(Optional in newer versions of the action; required in older versions)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Name

(*DeployToKubernetesCluster*/Environment/Connections/**Name**)

(Optional)

Specify the name of the account connection.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Role

(*DeployToKubernetesCluster*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that the **Deploy to Kubernetes cluster** action uses to access AWS. Make sure that you have added the role to your CodeCatalyst space, and that the role includes the following policies.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

- The following permissions policy:

> **Warning**
>
> Limit the permissions to those shown in the following policy. Using a role with broader permissions might pose a security risk.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

Note

The first time the role is used, use the following wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

- The following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Make sure that this role is added to:

- Your account connection. To learn more about adding an IAM role to an account connection, see Adding IAM roles to account connections.
- Your Kubernetes ConfigMap. To learn more about adding an IAM role to a ConfigMap, see Manage IAM users and roles in the `eksctl` documentation.

Tip

See also Tutorial: Deploy an application to Amazon EKS for instructions on adding am IAM role to an account connection and ConfigMap.

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *`my-environment`*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**Role**

Inputs

(*`DeployToKubernetesCluster`*/**Inputs**)

(Required if Connections is included)

The `Inputs` section defines the data that the `DeployToKubernetesCluster` needs during a workflow run.

> **Note**
>
> Only one input (either a source or an artifact) is allowed per **Deploy to Amazon EKS** action.

Corresponding UI: **Inputs** tab

Sources

(*`DeployToKubernetesCluster`*/Inputs/**Sources**)

(Required if your manifest file is stored in a source repository)

If your Kubernetes manifest file or files are stored in a source repository, specify the label of that source repository. Currently, the only supported label is `WorkflowSource`.

If your manifest files are not contained within a source repository, they must reside in an artifact generated by another action.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*DeployToKubernetesCluster*/Inputs/**Artifacts**)

(Required if your manifest file is stored in an output artifact from a previous action)

If the Kubernetes manifest file or files are contained in an artifact generated by a previous action, specify that artifact here. If your manifest files are not contained within an artifact, they must reside in your source repository.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Configuration tab/**Artifacts - optional**

Configuration

(*DeployToKubernetesCluster*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

Namespace

(*DeployToKubernetesCluster*/Configuration/**Namespace**)

(Optional)

Specify the Kubernetes namespace into which your Kubernetes application will be deployed. Use `default` if you are not using namespaces with your cluster. For more information on namespaces, see Subdividing your cluster using Kubernetes namespaces in the Kubernetes documentation.

If you omit the namespace, a value of `default` is used.

Corresponding UI: Configuration tab/**Namespace**

Region

(*DeployToKubernetesCluster*/Configuration/**Region**)

(Required)

Specify the AWS Region where your Amazon EKS cluster and service reside. For a list of Region codes, see Regional endpoints in the *AWS General Reference*.

Corresponding UI: Configuration tab/**Region**

Cluster

(*DeployToKubernetesCluster*/Configuration/**Cluster**)

(Required)

Specify the name of an existing Amazon EKS cluster. The **Deploy to Kubernetes cluster** action will deploy your containerized application into this cluster. For more information about Amazon EKS clusters, see Clusters in the Amazon EKS User Guide.

Corresponding UI: Configuration tab/**Cluster**

Manifests

(*DeployToKubernetesCluster*/Configuration/**Manifests**)

(Required)

Specify the path to your YAML-formatted Kubernetes manifest file(s), which are called *configuration files*, *config files*, or simply, *configurations* in the Kubernetes documentation.

If you're using multiple manifest files, place them in a single folder and reference that folder. Manifest files are processed alphanumerically by Kubernetes, so make sure to prefix file names with increasing numbers or letters to control the processing order. For example:

```
00-namespace.yaml
```

```
01-deployment.yaml
```

If your manifest files reside in your source repository, the path is relative to the source repository root folder. If the files reside in an artifact from a previous workflow action, the path is relative to the artifact root folder.

Examples:

`Manifests/`

`deployment.yaml`

`my-deployment.yml`

Do not use wildcards (*).

> **Note**
>
> Helm charts and kustomization files are not supported.

For more information about manifest files, see Organizing resource configurations in the Kubernetes documentation.

Corresponding UI: Configuration tab/**Manifests**

Deploying an AWS CloudFormation stack with a workflow

This section describes how to deploy a AWS CloudFormation stack using a CodeCatalyst workflow. To accomplish this, you must add the **Deploy AWS CloudFormation stack** action to your workflow. The action deploys a CloudFormation stack of resources into AWS based on a template that you provide. The template can be a:

- AWS CloudFormation template – For more information, see Working with AWS CloudFormation templates.
- AWS SAM template – For more information, see AWS Serverless Application Model (AWS SAM) specification.

> **Note**
>
> To use a AWS SAM template, you must first package your AWS SAM application using the `sam package` operation. For a tutorial that shows you how to do this packaging automatically as part of a Amazon CodeCatalyst workflow, see Tutorial: Deploy a serverless application using AWS CloudFormation.

If the stack already exists, the action runs the CloudFormation `CreateChangeSet` operation, and then the `ExecuteChangeSet` operation. The action then waits for the changes to be deployed and marks itself as either succeeded for failed, depending on the results.

Use the **Deploy AWS CloudFormation stack** action if you already have an AWS CloudFormation or AWS SAM template that contains resources you'd like to deploy, or you plan on generating one automatically as part of a workflow build action using tools like AWS SAM and AWS Cloud Development Kit (AWS CDK).

There are no restrictions on the template you can use—whatever you can author in CloudFormation or AWS SAM you can use with the **Deploy AWS CloudFormation stack** action.

> **Tip**
>
> For a tutorial that shows you how to deploy a serverless application using the **Deploy AWS CloudFormation stack** action, see Tutorial: Deploy a serverless application using AWS CloudFormation.

Topics

- Tutorial: Deploy a serverless application using AWS CloudFormation
- Adding the "Deploy AWS CloudFormation stack" action
- Configuring rollbacks
- Variables produced by the "Deploy AWS CloudFormation stack" action
- "Deploy AWS CloudFormation stack" action YAML definition

Tutorial: Deploy a serverless application using AWS CloudFormation

In this tutorial, you learn how to build, test, and deploy a serverless application as a CloudFormation stack using a workflow.

The application in this tutorial is a simple web application that outputs a "Hello World" message. It consists of an AWS Lambda function and an Amazon API Gateway, and you build it using the AWS Serverless Application Model (AWS SAM), which is an extension of AWS CloudFormation.

Topics

- Prerequisites

- Step 1: Create a source repository
- Step 2: Create AWS roles
- Step 3: Add AWS roles to CodeCatalyst
- Step 4: Create an Amazon S3 bucket
- Step 5: Add source files
- Step 6: Create and run a workflow
- Step 7: Make a change
- Clean up

Prerequisites

Before you begin:

- You need a CodeCatalyst **space** with a connected AWS account. For more information, see Creating a space.
- In your space, you need an empty, **Start from scratch** CodeCatalyst **project** called:

  ```
  codecatalyst-cfn-project
  ```

  For more information, see Creating an empty project in Amazon CodeCatalyst.
- In your project, you need a CodeCatalyst **environment** called:

  ```
  codecatalyst-cfn-environment
  ```

  Configure this environment as follows:

  - Choose any type, such as **Non-production**.
  - Connect your AWS account to it.
  - For the **Default IAM role**, choose any role. You'll specify a different role later.

  For more information, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.

Step 1: Create a source repository

In this step, you create a source repository in CodeCatalyst. This repository is used to store the tutorial's source files, such as the Lambda function file.

For more information about source repositories, see Creating a source repository.

To create a source repository

1. In CodeCatalyst, in the navigation pane, choose **Code**, and then choose **Source repositories**.
2. Choose **Add repository**, and then choose **Create repository**.
3. In **Repository name**, enter:

   ```
   codecatalyst-cfn-source-repository
   ```

4. Choose **Create**.

You have now created a repository called `codecatalyst-cfn-source-repository`.

Step 2: Create AWS roles

In this step, you create the following AWS IAM roles:

- **Deploy role** – Grants the CodeCatalyst **Deploy AWS CloudFormation stack** action permission to access your AWS account and CloudFormation service where you'll deploy your serverless application. The **Deploy AWS CloudFormation stack** action is part of your workflow.
- **Build role** – Grants the CodeCatalyst build action permission to access your AWS account and write to Amazon S3 where your serverless application package will be stored. The build action is part of your workflow.
- **Stack role** – Grants CloudFormation permission to read and modify the resources specified in the AWS SAM template that you will provide later. Also grants permission to CloudWatch.

For more information about IAM roles, see IAM roles in the *AWS Identity and Access Management User Guide*.

> **Note**
>
> To save time, you can create a single role, called the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role, instead

of the three roles listed previously. For more information, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*spaceName* role has very broad permissions that may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern. This tutorial assumes you are creating the three roles listed previously.

Note

A Lambda execution role is also required, but you don't need to create it now because the `sam-template.yml` file creates it for you when you run the workflow in step 5.

To create a deploy role

1. Create a policy for the role, as follows:
   a. Sign in to AWS.
   b. Open the IAM console at https://console.aws.amazon.com/iam/.
   c. In the navigation pane, choose **Policies**.
   d. Choose **Create policy**.
   e. Choose the **JSON** tab.
   f. Delete the existing code.
   g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
```

```
            "cloudformation:SetStackPolicy",
            "cloudformation:ValidateTemplate",
            "cloudformation:List*",
            "iam:PassRole"
        ],
        "Resource": "*",
        "Effect": "Allow"
}]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-deploy-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the deploy role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. Choose **Next**.

f. In **Permissions policies**, search for `codecatalyst-deploy-policy` and select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-deploy-role
```

i. For **Role description**, enter:

```
CodeCatalyst deploy role
```

j. Choose **Create role**.

You have now created a deploy role with a trust policy and permissions policy.

3. Obtain the deploy role ARN, as follows:

a. In the navigation pane, choose **Roles**.

b. In the search box, enter the name of the role you just created (`codecatalyst-deploy-role`).

c. Choose the role from the list.

The role's **Summary** page appears.

d. At the top, copy the **ARN** value.

You have now created the deploy role with the appropriate permissions, and obtained its ARN.

To create a build role

1. Create a policy for the role, as follows:

   a. Sign in to AWS.

   b. Open the IAM console at https://console.aws.amazon.com/iam/.

   c. In the navigation pane, choose **Policies**.

   d. Choose **Create policy**.

   e. Choose the **JSON** tab.

   f. Delete the existing code.

   g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "s3:PutObject",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

   h. Choose **Next: Tags**.

   i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-build-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the build role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. Choose **Next**.

f. In **Permissions policies**, search for `codecatalyst-build-policy` and select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-build-role
```

i. For **Role description**, enter:

```
CodeCatalyst build role
```

j. Choose **Create role**.

You have now created a build role with a trust policy and permissions policy.

3. Obtain the build role ARN, as follows:

   a. In the navigation pane, choose **Roles**.

   b. In the search box, enter the name of the role you just created (`codecatalyst-build-role`).

   c. Choose the role from the list.

      The role's **Summary** page appears.

   d. At the top, copy the **ARN** value.

You have now created the build role with the appropriate permissions, and obtained its ARN.

To create a stack role

1. Sign in to AWS using the account where you want to deploy your stack.
2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. Create the stack role as follows:

   a. In the navigation pane, choose **Roles**.

   b. Choose **Create role**.

   c. Choose **AWS service**.

   d. In the **Use case** section, choose **CloudFormation** from the drop-down list.

   e. Select the **CloudFormation** radio button.

   f. At the bottom, choose **Next**.

   g. Using the search box, find the following permissions policies, and then select their respective check boxes.

> **Note**
>
> If you search for a policy and it doesn't appear, make sure to choose **Clear filters** and try again.

- **CloudWatchFullAccess**
- **AWSCloudFormationFullAccess**
- **IAMFullAccess**
- **AWSLambda_FullAccess**
- **AmazonAPIGatewayAdministrator**
- **AmazonS3FullAccess**
- **AmazonEC2ContainerRegistryFullAccess**

The first policy allows access to CloudWatch to enable stack rollbacks when an alarm occurs.

The remaining policies allow AWS SAM to access the services and resources in the stack that will be deployed in this tutorial. For more information, see Permissions in the *AWS Serverless Application Model Developer Guide*.

h. Choose **Next**.

i. For **Role name**, enter:

```
codecatalyst-stack-role
```

j. Choose **Create role**.

4. Obtain the stack role's ARN, as follows:

a. In the navigation pane, choose **Roles**.

b. In the search box, enter the name of the role you just created (`codecatalyst-stack-role`).

c. Choose the role from the list.

d. In the **Summary** section, copy the **ARN** value. You need it later.

You have now created the stack role with the appropriate permissions, and you have obtained its ARN.

Step 3: Add AWS roles to CodeCatalyst

In this step, you add the build role (`codecatalyst-build-role`) and deploy role (`codecatalyst-deploy-role`) to the CodeCatalyst account connection in your space.

> **Note**
>
> You don't need to add the stack role (`codecatalyst-stack-role`) to the connection. This is because the stack role is used by *CloudFormation* (not CodeCatalyst), *after* a connection is already established between CodeCatalyst and AWS using the deploy role. Since the stack role is not used by CodeCatalyst to gain access to AWS, it does not need to be associated with an account connection.

To add build and deploy roles to your account connection

1. In CodeCatalyst, navigate to your space.
2. Choose **AWS accounts**. A list of account connections appears.
3. Choose the account connection that represents the AWS account where you created your build and deploy roles.
4. Choose **Manage roles from AWS management console**.

   The **Add IAM role to Amazon CodeCatalyst space** page appears. You might need to sign in to access the page.
5. Select **Add an existing role you have created in IAM**.

   A drop-down list appears. The list displays all IAM roles with a trust policy that includes the `codecatalyst-runner.amazonaws.com` and `codecatalyst.amazonaws.com` service principals.
6. In the drop-down list, choose `codecatalyst-build-role`, and choose **Add role**.
7. Choose **Add IAM role**, choose **Add an existing role you have created in IAM**, and in the drop-down list, choose `codecatalyst-deploy-role`. Choose **Add role**.

You have now added the build and deploy roles to your space.

8. Copy the value of the **Amazon CodeCatalyst display name**. You'll need this value later, when creating your workflow.

Step 4: Create an Amazon S3 bucket

In this step, you create an Amazon S3 bucket where you store your serverless application's deployment package .zip file.

To create an Amazon S3 bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the main pane, choose **Create bucket**.
3. For **Bucket name**, enter:

```
codecatalyst-cfn-s3-bucket
```

4. For **AWS Region**, choose a Region. This tutorial assumes you chose **US West (Oregon) us-west-2**. For information about Regions supported by Amazon S3, see Amazon Simple Storage Service endpoints and quotas in the *AWS General Reference*.
5. At the bottom of the page, choose **Create bucket**.

You have now created a bucket called **`codecatalyst-cfn-s3-bucket`** in the US West (Oregon) us-west-2 Region.

Step 5: Add source files

In this step, you add several application source files to your CodeCatalyst source repository. The `hello-world` folder contains the application files that you'll deploy. The `tests` folder contains unit tests. The folder structure is as follows:

```
.
|— hello-world
|   |— tests
|      |— unit
|         |— test-handler.js
|   |— app.js
```

```
|— .npmignore
|— package.json
|— sam-template.yml
|— setup-sam.sh
```

.npmignore file

The `.npmignore` file indicates which files and folders npm should exclude from the application package. In this tutorial, npm excludes the `tests` folder because it is not part of the application.

To add the .npmignore file

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project, `codecatalyst-cfn-project`
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. From the list of source repositories, choose your repository, `codecatalyst-cfn-source-repository`.
5. In **Files**, choose **Create file**.
6. For **File name**, enter:

   ```
   .npmignore
   ```

7. In the text box, enter the following code:

   ```
   tests/*
   ```

8. Choose **Commit**, and then choose **Commit** again.

   You have now created a file called `.npmignore` in the root of your repository.

package.json file

The `package.json` file contains important metadata about your Node project such as the project name, version number, description, dependencies, and other details that describe how to interact with and run your application.

The `package.json` in this tutorial includes a list of dependencies and a `test` script. The test script does the following:

- Using mocha, the test script runs the unit tests specified in `hello-world/tests/unit/` and writes the results to a `junit.xml` file using the xunit reporter.
- Using Istanbul (nyc), the test script generates a code coverage report (`clover.xml`) using the clover reporter. For more information, see Using alternative reporters in the Istanbul documentation.

To add the package.json file

1. In your repository, in **Files**, choose **Create file**.
2. For **File name**, enter:

   ```
   package.json
   ```

3. In the text box, enter the following code:

   ```
   {
     "name": "hello_world",
     "version": "1.0.0",
     "description": "hello world sample for NodeJS",
     "main": "app.js",
     "repository": "https://github.com/awslabs/aws-sam-cli/tree/develop/samcli/local/
   init/templates/cookiecutter-aws-sam-hello-nodejs",
     "author": "SAM CLI",
     "license": "MIT",
     "dependencies": {
       "axios": "^0.21.1",
       "nyc": "^15.1.0"
     },
     "scripts": {
       "test": "nyc --reporter=clover mocha hello-world/tests/unit/ --reporter xunit
    --reporter-option output=junit.xml"
     },
     "devDependencies": {
       "aws-sdk": "^2.815.0",
       "chai": "^4.2.0",
       "mocha": "^8.2.1"
     }
   }
   ```

4. Choose **Commit**, and then choose **Commit** again.

You have now added a file called `package.json` to the root of the repository.

sam-template.yml file

The `sam-template.yml` file contains the instructions for deploying the Lambda function and API Gateway and configuring them together. It follows the AWS Serverless Application Model template specification, which extends the AWS CloudFormation template specification.

You use an AWS SAM template in this tutorial instead of a regular AWS CloudFormation template because AWS SAM offers a helpful AWS::Serverless::Function resource type. This type performs much behind-the-scenes configuration that you normally have to write out to use the basic CloudFormation syntax. For example, the `AWS::Serverless::Function` creates a Lambda function, Lambda execution role, and event source mappings that start the function. You have to code all of this if you want to write it using basic CloudFormation.

Although this tutorial uses a pre-written template, you can generate one as part of your workflow using a build action. For more information, see Deploying an AWS CloudFormation stack with a workflow.

To add the sam-template.yml file

1. In your repository, in **Files**, choose **Create file**.
2. For **File name**, enter:

```
sam-template.yml
```

3. In the text box, enter the following code:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  serverless-api

  Sample SAM Template for serverless-api

# More info about Globals: https://github.com/awslabs/serverless-application-model/
blob/master/docs/globals.rst
Globals:
  Function:
    Timeout: 3
```

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # For details on this resource type,
 see https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Properties:
      CodeUri: hello-world/
      Handler: app.lambdaHandler
      Runtime: nodejs12.x
      Events:
        HelloWorld:
          Type: Api # For details on this event source type, see
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#api
          Properties:
            Path: /hello
            Method: get

Outputs:
  # ServerlessRestApi is an implicit API created out of the events key under
 Serverless::Function
  # Find out about other implicit resources you can reference within AWS SAM at
  # https://github.com/awslabs/serverless-application-model/blob/master/docs/
internals/generated_resources.rst#api
  HelloWorldApi:
    Description: "API Gateway endpoint URL for the Hello World function"
    Value: !Sub "https://${ServerlessRestApi}.execute-api.
${AWS::Region}.amazonaws.com/Prod/hello/"
  HelloWorldFunction:
    Description: "Hello World Lambda function ARN"
    Value: !GetAtt HelloWorldFunction.Arn
  HelloWorldFunctionIamRole:
    Description: "Implicit Lambda execution role created for the Hello World
 function"
    Value: !GetAtt HelloWorldFunctionRole.Arn
```

4. Choose **Commit**, and then choose **Commit** again.

   You have now added a file called `sam-template.yml` under the root folder of your repository.

setup-sam.sh file

The `setup-sam.sh` file contains the instructions for downloading and installing the AWS SAM CLI utility. The workflow uses this utility to package the `hello-world` source.

To add the setup-sam.sh file

1. In your repository, in **Files**, choose **Create file**.
2. For **File name**, enter:

   ```
   setup-sam.sh
   ```

3. In the text box, enter the following code:

   ```
   #!/usr/bin/env bash
   echo "Setting up sam"

   yum install unzip -y

   curl -L0 https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-
   linux-x86_64.zip
   unzip -qq aws-sam-cli-linux-x86_64.zip -d sam-installation-directory

   ./sam-installation-directory/install; export AWS_DEFAULT_REGION=us-west-2
   ```

   In the preceding code, replace *us-west-2* with your AWS Region.

4. Choose **Commit**, and then choose **Commit** again.

   You have now added a file called `setup-sam.sh` to the root of the repository.

app.js file

The `app.js` contains the Lambda function code. In this tutorial, the code returns the text `hello world`.

To add the app.js file

1. In your repository, in **Files**, choose **Create file**.
2. For **File name**, enter:

hello-world/app.js

3. In the text box, enter the following code:

```
// const axios = require('axios')
// const url = 'http://checkip.amazonaws.com/';
let response;

/**
 *
 * Event doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format
 * @param {Object} event - API Gateway Lambda Proxy Input Format
 *
 * Context doc: https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-
context.html
 * @param {Object} context
 *
 * Return doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html
 * @returns {Object} object - API Gateway Lambda Proxy Output Format
 *
 */
exports.lambdaHandler = async (event, context) => {
    try {
        // const ret = await axios(url);
        response = {
            'statusCode': 200,
            'body': JSON.stringify({
                message: 'hello world',
                // location: ret.data.trim()
            })
        }
    } catch (err) {
        console.log(err);
        return err;
    }

    return response
};
```

4. Choose **Commit**, and then choose **Commit** again.

You have now created a folder called `hello-world` and a file called `app.js`.

test-handler.js file

The `test-handler.js` file contains unit tests for the Lambda function.

To add the test-handler.js file

1. In your repository, in **Files**, choose **Create file**.
2. For **File name**, enter:

```
hello-world/tests/unit/test-handler.js
```

3. In the text box, enter the following code:

```
'use strict';

const app = require('../../app.js');
const chai = require('chai');
const expect = chai.expect;
var event, context;

describe('Tests index', function () {
    it('verifies successful response', async () => {
        const result = await app.lambdaHandler(event, context)

        expect(result).to.be.an('object');
        expect(result.statusCode).to.equal(200);
        expect(result.body).to.be.an('string');

        let response = JSON.parse(result.body);

        expect(response).to.be.an('object');
        expect(response.message).to.be.equal("hello world");
        // expect(response.location).to.be.an("string");
    });
});
```

4. Choose **Commit**, and then choose **Commit** again.

You have now added a file called `test-handler.js` under the `hello-world/tests/unit` folder.

You have now added all your source files.

Take a moment to double-check your work and make sure you placed all the files in the correct folders. The folder structure is as follows:

```
.
|— hello-world
|   |— tests
|      |— unit
|         |— test-handler.js
|   |— app.js
|— .npmignore
|— README.md
|— package.json
|— sam-template.yml
|— setup-sam.sh
```

Step 6: Create and run a workflow

In this step, you create a workflow that packages your Lambda source code and deploys it. The workflow consists of the following building blocks that run sequentially:

- A trigger – This trigger starts the workflow run automatically when you push a change to your source repository. For more information about triggers, see Starting a workflow run automatically with triggers.
- A test action (`Test`) – On trigger, this action installs Node package manager (npm), and then runs the `npm run test` command. This command tells npm to run the `test` script defined in the `package.json` file. The `test` script, in turn, runs the unit tests and generates two reports: a test report (`junit.xml`) and a code coverage report (`clover.xml`). For more information, see package.json file.

  Next, the test action transforms the XML reports into CodeCatalyst reports and displays them in the CodeCatalyst console, under the **Reports** tab of the test action.

  For more information about the test action, see Testing with workflows.

- A build action (`BuildBackend`) – On completion of the test action, the build action downloads and installs the AWS SAM CLI, packages the `hello-world` source, and copies the package to your Amazon S3 bucket, where the Lambda service expects it to be. The action also outputs a new AWS SAM template file called `sam-template-packaged.yml` and places it in an output artifact called `buildArtifact`.

  For more information about the build action, see Building with workflows.
- A deploy action (`DeployCloudFormationStack`) – On completion of the build action, the deploy action looks for the output artifact generated by the build action (`buildArtifact`), finds the AWS SAM template inside of it, and then runs the template. The AWS SAM template creates a stack that deploys the serverless application.

To create a workflow

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose **Create workflow**.
3. For **Source repository**, choose `codecatalyst-cfn-source-repository`.
4. For **Branch**, choose `main`.
5. Choose **Create**.
6. Delete the YAML sample code.
7. Add the following YAML code:

> **Note**
>
> In the YAML code that follows, you can omit the `Connections:` sections if you want. If you omit these sections, you must ensure that the role specified in the **Default IAM role** field in your environment includes the permissions and trust policies of both roles described in Step 2: Create AWS roles. For more information about setting up an environment with a default IAM role, see Creating an environment.

```
Name: codecatalyst-cfn-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
```

```
    Branches:
      - main
Actions:
  Test:
    Identifier: aws/managed-test@v1
    Inputs:
      Sources:
        - WorkflowSource
    Outputs:
      Reports:
        CoverageReport:
          Format: CLOVERXML
          IncludePaths:
            - "coverage/*"
        TestReport:
          Format: JUNITXML
          IncludePaths:
            - junit.xml
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm run test
  BuildBackend:
    Identifier: aws/build@v1
    DependsOn:
      - Test
    Environment:
      Name: codecatalyst-cfn-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-build-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: . ./setup-sam.sh
        - Run: sam package --template-file sam-template.yml --s3-
bucket codecatalyst-cfn-s3-bucket --output-template-file sam-template-packaged.yml
 --region us-west-2
    Outputs:
      Artifacts:
        - Name: buildArtifact
          Files:
```

```
          - "**/*"
  DeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    DependsOn:
      - BuildBackend
    Environment:
      Name: codecatalyst-cfn-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-deploy-role
    Inputs:
      Artifacts:
        - buildArtifact
      Sources: []
    Configuration:
      name: codecatalyst-cfn-stack
      region: us-west-2
      role-arn: arn:aws:iam::111122223333:role/StackRole
      template: ./sam-template-packaged.yml
      capabilities: CAPABILITY_IAM,CAPABILITY_AUTO_EXPAND
```

In the preceding code, replace:

- Both instances of *codecatalyst-cfn-environment* with the name of your environment.
- Both instances of *codecatalyst-account-connection* with the display name of your account connection. The display name might be a number. For more information, see Step 3: Add AWS roles to CodeCatalyst.
- *codecatalyst-build-role* with the name of the build role that you created in Step 2: Create AWS roles.
- *codecatalyst-cfn-s3-bucket* with the name of the Amazon S3 bucket you created in Step 4: Create an Amazon S3 bucket.
- Both instances of *us-west-2* with the Region where your Amazon S3 bucket resides (first instance) and where your stack will be deployed (second instance). These Regions can be different. This tutorial assumes that both Regions are set to `us-west-2`. For details about Regions supported by Amazon S3 and AWS CloudFormation, see Service endpoints and quotas in the *AWS General Reference*.
- *codecatalyst-deploy-role* with the name of the deploy role that you created in Step 2: Create AWS roles.

- *codecatalyst-cfn-environment* with the name of the environment that you created in Prerequisites.
- *arn:aws:iam::111122223333:role/StackRole* with the Amazon Resource Name (ARN) of the stack role that you created in Step 2: Create AWS roles.

> **Note**
>
> If you decided not to create build, deploy, and stack roles, replace *codecatalyst-build-role*, *codecatalyst-deploy-role*, and *arn:aws:iam::111122223333:role/StackRole* with the name or ARN of the `CodeCatalystWorkflowDevelopmentRole-`*spaceName* role. For more information about this role, see Step 2: Create AWS roles.

For information about the properties in the code shown previously, see the "Deploy AWS CloudFormation stack" action YAML definition.

8. (Optional) Choose **Validate** to make sure the YAML code is valid before committing.
9. Choose **Commit**.
10. On the **Commit workflow** dialog box, enter the following:

    a. For **Workflow file name**, keep the default, `codecatalyst-cfn-workflow`.

    b. For **Commit message**, enter:

    ```
    add initial workflow file
    ```

    c. For **Repository**, choose **codecatalyst-cfn-source-repository**.

    d. For **Branch name**, choose **main**.

    e. Choose **Commit**.

You have now created a workflow. A workflow run starts automatically because of the trigger defined at the top of the workflow. Specifically, when you committed (and pushed) the `codecatalyst-cfn-workflow.yaml` file to your source repository, the trigger started the workflow run.

To view the workflow run in progress

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the workflow you just created: `codecatalyst-cfn-workflow`.
3. Choose the **Runs** tab.
4. In the **Run ID** column, choose the run ID.
5. Choose **Test** to see the tests progress.
6. Choose **BuildBackend** to see the build progress.
7. Choose **DeployCloudFormationStack** to see the deployment progress.

   For more information about viewing run details, see Viewing workflow run status and details.
8. When the **DeployCloudFormationStack** action finishes, do the following:

   - If the workflow run succeeded, go to the next procedure.
   - If the workflow run failed on the **Test** or **BuildBackend** action, choose **Logs** to troubleshoot the issue.
   - If the workflow run failed on the **DeployCloudFormationStack** action, choose the deploy action, and then choose the **Summary** tab. Scroll to the **CloudFormation events** section to view the detailed error message. If a rollback occurred, delete the `codecatalyst-cfn-stack` stack through the AWS CloudFormation console in AWS before re-running the workflow.

To verify the deployment

1. After a successful deployment, choose **Variables (7)** from the horizontal menu bar near the top. (Do not choose **Variables** in the pane on the right.)
2. Next to **HelloWorldApi**, paste the `https://` URL into a browser.

   A **hello world** JSON message from the Lambda function is displayed, indicating that the workflow deployed and configured the Lambda function and API Gateway successfully.

> **Tip**
>
> You can have CodeCatalyst display this URL in the workflow diagram with a few small configurations. For more information, see Displaying the URL of the deployed application in the workflow diagram.

To verify unit test results and code coverage

1. In the workflow diagram, choose **Test**, and then choose **Reports**.
2. Choose **TestReport** to view the unit test results, or choose **CoverageReport** to view the code coverage details of the files being tested, in this case, `app.js` and `test-handler.js`.

To verify deployed resources

1. Sign in to the AWS Management Console and open the API Gateway console at https://console.aws.amazon.com/apigateway/.
2. Observe the **codecatalyst-cfn-stack** API that the AWS SAM template created. The API name comes from the `Configuration/name` value in the workflow definition file (`codecatalyst-cfn-workflow.yaml`).
3. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
4. In the navigation pane, choose **Functions**.
5. Choose your Lambda function, `codecatalyst-cfn-stack-HelloWorldFunction-`*`string`*.
6. You can see how the API Gateway is a trigger for the function. This integration was automatically configured by the AWS SAM `AWS::Serverless::Function` resource type.

Step 7: Make a change

In this step, you make a change to your Lambda source code and commit it. This commit starts a new workflow run. This run deploys the new Lambda function in a blue-green scheme that uses the default traffic shifting configuration specified in the Lambda console.

To make a change to your Lambda source

1. In CodeCatalyst, navigate to your project.

2. In the navigation pane, choose **Code**, and then choose **Source repositories**.
3. Choose your source repository `codecatalyst-cfn-source-repository`.
4. Change the application file:

   a. Choose the `hello-world` folder.

   b. Choose the `app.js` file.

   c. Choose **Edit**.

   d. On line 23, change `hello world` to **`Tutorial complete!`**.

   e. Choose **Commit**, and then choose **Commit** again.

   The commit causes a workflow run to start. This run will fail because you haven't updated the unit tests to reflect the name change.
5. Update the unit tests:

   a. Choose `hello-world\tests\unit\test-handler.js`.

   b. Choose **Edit**.

   c. On line 19, change `hello world` to **`Tutorial complete!`**.

   d. Choose **Commit**, and then choose **Commit** again.

   The commit causes another workflow run to start. This run will succeed.
6. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
7. Choose `codecatalyst-cfn-workflow`, and then choose **Runs**.
8. Choose the run ID of the latest run. It should still be in progress.
9. Choose **Test**, **BuildBackend**, and **DeployCloudFormationStack** to see the workflow run progress.
10. When the workflow finishes, choose **Variables (7)** near the top.
11. Next to **HelloWorldApi**, paste the `https://` URL into a browser.

    A `Tutorial complete!` message appears in the browser, indicating that your new application was deployed successfully.

Clean up

Clean up the files and services used in this tutorial to avoid being charged for them.

To clean up in the CodeCatalyst console

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Delete `codecatalyst-cfn-workflow`.
3. Delete `codecatalyst-cfn-environment`.
4. Delete `codecatalyst-cfn-source-repository`.
5. Delete `codecatalyst-cfn-project`.

To clean up in the AWS Management Console

1. Clean up in CloudFormation, as follows:
   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
   b. Delete the `codecatalyst-cfn-stack`.

      Deleting the stack removes all tutorial resources from the API Gateway and Lambda services.
2. Clean up in Amazon S3, as follows:
   a. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
   b. Choose the `codecatalyst-cfn-s3-bucket`.
   c. Delete the bucket contents.
   d. Delete the bucket.
3. Clean up in IAM, as follows:
   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   b. Delete the `codecatalyst-deploy-policy`.
   c. Delete the `codecatalyst-build-policy`.
   d. Delete the `codecatalyst-stack-policy`.
   e. Delete the `codecatalyst-deploy-role`.
   f. Delete the `codecatalyst-build-role`.
   g. Delete the `codecatalyst-stack-role`.

In this tutorial, you learned how to deploy a serverless application as a CloudFormation stack using a CodeCatalyst workflow and a **Deploy AWS CloudFormation stack** action.

Adding the "Deploy AWS CloudFormation stack" action

Use the following instructions to add the **Deploy AWS CloudFormation stack** action to your workflow.

Visual

To add the "Deploy AWS CloudFormation stack" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Deploy AWS CloudFormation stack** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **Deploy AWS CloudFormation stack**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **Download** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the "Deploy AWS CloudFormation stack" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.

11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add the "Deploy AWS CloudFormation stack" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Deploy AWS CloudFormation stack** action, and do one of the following:
   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **Deploy AWS CloudFormation stack**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **Download** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "Deploy AWS CloudFormation stack" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

Configuring rollbacks

By default, if the **Deploy AWS CloudFormation stack** action fails, it will cause AWS CloudFormation to roll back the stack to the last known stable state. You can change the behavior so that rollbacks occur not only when the action fails, but also when a specified Amazon CloudWatch alarm occurs. For more information about CloudWatch alarms, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

You can also change the default behavior so that CloudFormation does not roll back the stack when the action fails.

Use the following instructions to configure rollbacks.

> **Note**
>
> You cannot start a rollback manually.

Visual

Before you begin

1. Make sure you have a workflow that includes a functioning **Deploy AWS CloudFormation stack** action. For more information, see Deploying an AWS CloudFormation stack with a workflow.
2. In the role specified in the **Stack role - optional** field of the **Deploy AWS CloudFormation stack** action, make sure to include the **CloudWatchFullAccess** permission. For information about creating this role with the appropriate permissions, see Step 2: Create AWS roles.

To configure rollback alarms for the "Deploy AWS CloudFormation stack" action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of a workflow that includes the **Deploy AWS CloudFormation stack** action. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

5. Choose **Edit**.
6. Choose **Visual**.
7. Choose your **Deploy AWS CloudFormation stack** action.
8. In the details pane, choose **Configuration**.
9. At the bottom, expand **Advanced**.
10. Under **Monitor alarm ARNs**, choose **Add alarm**.
11. Enter information into the following fields.

    - **Alarm ARN**

      Specify the Amazon Resource Name (ARN) of an Amazon CloudWatch alarm to use as a rollback trigger. For example, `arn:aws:cloudwatch::123456789012:alarm/MyAlarm`. You can have a maximum of five rollback triggers.

      > **Note**
      >
      > If you specify a CloudWatch alarm ARN, you'll also need to configure additional permissions to enable the action to access CloudWatch. For more information, see Configuring rollbacks.

    - **Monitoring time**

      Specify an amount of time, from 0 to 180 minutes, during which CloudFormation monitors the specified alarms. Monitoring begins *after* all the stack resources have been deployed. If the alarm occurs within the specified monitoring time, then the deployment fails, and CloudFormation rolls back the entire stack operation.

      Default: 0. CloudFormation only monitors alarms while the stack resources are being deployed, not after.

YAML

To configure rollback triggers for the "Deploy AWS CloudFormation stack" action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.

4. Choose the name of a workflow that includes the **Deploy AWS CloudFormation stack** action. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. Add the `monitor-alarm-arns` and `monitor-timeout-in-minutes` properties in the YAML code to add rollback triggers. For an explanation of each property, see "Deploy AWS CloudFormation stack" action YAML definition.
8. In the role specified in the `role-arn` property of the **Deploy AWS CloudFormation stack** action, make sure to include the **CloudWatchFullAccess** permission. For information about creating this role with the appropriate permissions, see Step 2: Create AWS roles.

Visual

To turn off rollbacks for the "Deploy AWS CloudFormation stack" action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of a workflow that includes the **Deploy AWS CloudFormation stack** action. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. Choose your **Deploy AWS CloudFormation stack** action.
8. In the details pane, choose **Configuration**.
9. At the bottom, expand **Advanced**.
10. Turn on **Disable rollback**.

YAML

To turn off rollbacks for the "Deploy AWS CloudFormation stack" action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of a workflow that includes the **Deploy AWS CloudFormation stack** action. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. Add the `disable-rollback: 1` property in the YAML code to stop rollbacks. For an explanation of this property, see "Deploy AWS CloudFormation stack" action YAML definition.

Variables produced by the "Deploy AWS CloudFormation stack" action

The **Deploy AWS CloudFormation stack** action produces and sets the following variables at run time. These are known as *predefined variables*.

For information about referencing these variables in a workflow, see Using predefined variables.

Key	Value
deployment-platform	The name of the deployment platform. Hardcoded to `AWS:CloudFormation` .
region	The region code of the AWS Region that was deployed to during the workflow run. Example: `us-west-2`
stack-id	The Amazon Resource Name (ARN) of the deployed stack. Example: `arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cfn-stack/6aad4380-100a-11ec-a10a-03b8a84d40df`

"Deploy AWS CloudFormation stack" action YAML definition

The following is the YAML definition of the **Deploy AWS CloudFormation stack** action. To learn how to use this action, see Deploying an AWS CloudFormation stack with a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  DeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: DeployRole
    Inputs:
      Sources:
        - source-name-1
      Artifacts:
        - CloudFormation-artifact
    Configuration:
```

```
      name: stack-name
      region: us-west-2
      template: template-path
      role-arn: arn:aws:iam::123456789012:role/StackRole
      capabilities: CAPABILITY_IAM,CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
      parameter-overrides: KeyOne=ValueOne,KeyTwo=ValueTwo | path-to-JSON-file
      no-execute-changeset: 1|0
      fail-on-empty-changeset: 1|0
      disable-rollback: 1|0
      termination-protection: 1|0
      timeout-in-minutes: minutes
      notification-arns: arn:aws:sns:us-east-1:123456789012:MyTopic,arn:aws:sns:us-
east-1:123456789012:MyOtherTopic
      monitor-alarm-arns: arn:aws:cloudwatch::123456789012:alarm/
MyAlarm,arn:aws:cloudwatch::123456789012:alarm/MyOtherAlarm
      monitor-timeout-in-minutes: minutes
      tags: '[{"Key":"MyKey1","Value":"MyValue1"},{"Key":"MyKey2","Value":"MyValue2"}]'
```

DeployCloudFormationStack

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `DeployCloudFormationStack_nn`.

Corresponding UI: Configuration tab/**Action display name**

Identifier

(*DeployCloudFormationStack*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/cfn-deploy@v1`.

Corresponding UI: Workflow diagram/DeployCloudFormationStack_nn/**aws/cfn-deploy@v1** label

DependsOn

(*DeployCloudFormationStack*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*DeployCloudFormationStack*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*DeployCloudFormationStack*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or EC2 (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or Lambda (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/Advanced - optional/**Compute type**

Fleet

(*DeployCloudFormationStack*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/Advanced - optional/**Compute fleet**

Timeout

(*DeployCloudFormationStack*/**Timeout**)

(Optional)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout in minutes - optional**

Environment

(*DeployCloudFormationStack*/**Environment**)

(Required)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the

default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*DeployCloudFormationStack*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*DeployCloudFormationStack*/Environment/**Connections**)

(Optional in newer versions of the action; required in older versions)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Name

(*DeployCloudFormationStack*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Role

(*DeployCloudFormationStack*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that the **Deploy AWS CloudFormation stack** action uses to access AWS and the AWS CloudFormation service. Make sure that you have added the role to your CodeCatalyst space, and that the role includes the following policies.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

- The following permissions policy:

Warning

Limit the permissions to those shown in the following policy. Using a role with broader permissions might pose a security risk.

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "cloudformation:List*",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

Note

The first time the role is used, use the following wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

- The following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *`my-environment`*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**Role**

Inputs

(*`DeployCloudFormationStack`*/**Inputs**)

(Optional)

The `Inputs` section defines the data that the `DeployCloudFormationStack` needs during a workflow run.

> **Note**
>
> A maximum of four inputs (one source and three artifacts) are allowed per **Deploy AWS CloudFormation stack** action.

If you need to refer to files residing in different inputs (say a source and an artifact), the source input is the primary input, and the artifact is the secondary input. References to files in secondary inputs take a special prefix to distiguish them from the primary. For details, see Example: Referencing files in multiple artifacts.

Corresponding UI: **Inputs** tab

Sources

(*DeployCloudFormationStack*/Inputs/**Sources**)

(Required if your CloudFormation or AWS SAM template is stored in a source repository)

If your CloudFormation or AWS SAM template is stored in a source repository, specify the label of that source repository. Currently, the only supported label is `WorkflowSource`.

If your CloudFormation or AWS SAM template is not contained within a source repository, it must reside in an artifact generated by another action, or in an Amazon S3 bucket.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*DeployCloudFormationStack*/Inputs/**Artifacts**)

(Required if your CloudFormation or AWS SAM template is stored in an output artifact from a previous action)

If the CloudFormation or AWS SAM template that you want to deploy is contained in an artifact generated by a previous action, specify that artifact here. If your CloudFormation template is not contained within an artifact, it must reside in your source repository or in an Amazon S3 bucket.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Configuration tab/**Artifacts - optional**

Configuration

(*DeployCloudFormationStack*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

name

(*DeployCloudFormationStack*/Configuration/**name**)

(Required)

Specify a name for the CloudFormation stack that the **Deploy AWS CloudFormation stack** action creates or updates.

Corresponding UI: Configuration tab/**Stack name**

region

(*DeployCloudFormationStack*/Configuration/**region**)

(Required)

Specify the AWS Region into which the stack will be deployed. For a list of Region codes, see Regional endpoints.

Corresponding UI: Configuration tab/**Stack region**

template

(*DeployCloudFormationStack*/Configuration/**template**)

(Required)

Specify the name and path to your CloudFormation or AWS SAM template file. The template can be in JSON or YAML format, and can reside in a source repository, an artifact from a previous action, or an Amazon S3 bucket. If the template file is in a source repository or artifact, the path is relative to the source or artifact root. If the template is in an Amazon S3 bucket, the path is the template's **Object URL** value.

Examples:

`./MyFolder/MyTemplate.json`

`MyFolder/MyTemplate.yml`

`https://MyBucket.s3.us-west-2.amazonaws.com/MyTemplate.yml`

> **Note**
>
> You may need to add a prefix to the template's file path to indicate which artifact or source to find it in. For more information, see Referencing files in a source repository and Referencing files in an artifact.

Corresponding UI: Configuration tab/**Template**

role-arn

(*`DeployCloudFormationStack`*/Configuration/**role-arn**)

(Required)

Specify the Amazon Resource Name (ARN) of the stack role. CloudFormation uses this role to access and modify resources in your stack. For example: `arn:aws:iam::123456789012:role/StackRole`.

Make sure the stack role includes:

- One or more permissions policies. The policies depend on the resources you have in your stack. For example, if your stack includes an AWS Lambda function, you need to add permissions that grant access to Lambda. If you followed the tutorial described in Tutorial: Deploy a serverless application using AWS CloudFormation, it includes a procedure titled, To create a stack role that lists the permissions that the stack role needs if you're deploying a typical serverless application stack.

  > **Warning**
  >
  > Limit the permissions to those required by the CloudFormation service to access resources in your stack. Using a role with broader permissions might pose a security risk.

- The following trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudformation.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Optionally, associate this role with your account connection. To learn more about associating an IAM role with an account connection, see Adding IAM roles to account connections. If you do not associate the stack role with the account connection, then the stack role will not appear in the **Stack role** drop-down list in the visual editor; however, the role ARN can still be specified in the `role-arn` field using the YAML editor.

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: Configuration tab/**Stack role - optional**

capabilities

(*`DeployCloudFormationStack`*/Configuration/**capabilities**)

(Required)

Specify a list of IAM capabilities that are required to allow AWS CloudFormation to create certain stacks. In most cases, you can leave `capabilities` with the default value of `CAPABILITY_IAM,CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND`.

If you see `##[error] requires capabilities: [`*`capability-name`*`]` in your **Deploy AWS CloudFormation stack** action's logs, see How do I fix IAM capabilities errors? for information about how to fix the problem.

For more information about IAM capabilities, see Acknowledging IAM resources in AWS CloudFormation templates in the *IAM User Guide*.

Corresponding UI: Configuration tab/Advanced/**Capabilities**

parameter-overrides

(*`DeployCloudFormationStack`*/Configuration/**parameter-overrides**)

(Optional)

Specify parameters in your AWS CloudFormation or AWS SAM template that don't have default values, or for which you want to specify non-default values. For more information about parameters, see Parameters in the *AWS CloudFormation User Guide*.

The `parameter-overrides` property accepts:

- A JSON file containing the parameters and values.
- A comma-separate list of parameters and values.

To specify a JSON file

1. Make sure the JSON file uses one of the following syntaxes:

```
{
  "Parameters": {
    "Param1": "Value1",
    "Param2": "Value2",
    ...
  }
}
```

   Or...

```
[
  {
     "ParameterKey": "Param1",
     "ParameterValue": "Value1"
  },
  ...
]
```

(There are other syntaxes, but they are not supported by CodeCatalyst at the time of writing.) For more information about specifying CloudFormation parameters in a JSON file, see Supported JSON syntax in the *AWS CLI Command Reference*.

2. Specify the path to the JSON file using one of the following formats:

   - If your JSON file resides in an output artifact from a previous action, use:

     `file:///artifacts/`*`current-action-name`*`/`*`output-artifact-name`*`/`*`path-to-json-file`*

     See **Example 1** for details.

   - If your JSON file resides in your source repository, use:

     `file:///sources/WorkflowSource/`*`path-to-json-file`*

     See **Example 2** for details.

     Example 1 – The JSON file resides in an output artifact

     ```
     ##My workflow YAML
     ...
     Actions:
       MyBuildAction:
         Identifier: aws/build@v1
         Outputs:
           Artifacts:
             - Name: ParamArtifact
               Files:
                 - params.json
         Configuration:
         ...
       MyDeployCFNStackAction:
     ```

```
    Identifier: aws/cfn-deploy@v1
    Configuration:
      parameter-overrides: file:///artifacts/MyDeployCFNStackAction/
ParamArtifact/params.json
```

Example 2 – The JSON file resides in your source repository, in a folder called `my/folder`

```
##My workflow YAML
...
Actions:
  MyDeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      parameter-overrides: file:///sources/WorkflowSource/my/folder/params.json
```

To use a comma-separate list of parameters

- Add parameter name-value pairs in the `parameter-overrides` property using the following format:

  param-1=*value-1*,*param-2*=*value-2*

  For example, assuming the following AWS CloudFormation template:

```
##My CloudFormation template

Description: My AWS CloudFormation template

Parameters:
  InstanceType:
    Description: Defines the Amazon EC2 compute for the production server.
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - t2.small
      - t3.medium
```

```
Resources:
...
```

...you might set the `parameter-overrides` property as follows:

```
##My workflow YAML
...
Actions:
...
  DeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    Configuration:
      parameter-overrides: InstanceType=t3.medium,UseVPC=true
```

> **Note**
>
> You can specify a parameter name without a corresponding value using `undefined` as the value. For example:
> `parameter-overrides: MyParameter=undefined`
> The effect is that during a stack update, CloudFormation uses the existing parameter value for the given parameter name.

Corresponding UI:

- Configuration tab/Advanced/**Parameter overrides**
- Configuration tab/Advanced/Parameter overrides/**Specify overrides using a file**
- Configuration tab/Advanced/Parameter overrides/**Specify overrides using a value set**

no-execute-changeset

(*`DeployCloudFormationStack`*/Configuration/**no-execute-changeset**)

(Optional)

Specify whether you want CodeCatalyst to create the CloudFormation change set and then stop before running it. This gives you the opportunity to review the change set in the CloudFormation console. If you determine that the change set looks good, disable this option and then re-run

the workflow so that CodeCatalyst can create and run the change set without stopping. The default is to create and run the change set without stopping. For more information, see the AWS CloudFormation deploy parameter in the *AWS CLI Command Reference*. For more information about viewing change sets, see Viewing a change set in the *AWS CloudFormation User Guide*.

Corresponding UI: Configuration tab/Advanced/**No execute change set**

fail-on-empty-changeset

(*DeployCloudFormationStack*/Configuration/**fail-on-empty-changeset**)

(Optional)

Specify whether you want CodeCatalyst to fail the **Deploy AWS CloudFormation stack** action if the CloudFormation change set is empty. (If a change set is empty, it means there were no changes made to the stack during the latest deployment.) The default is to allow the action to proceed if the change set is empty, and to return an `UPDATE_COMPLETE` message even though the stack was not updated.

For more information about this setting, see the AWS CloudFormation deploy parameter in the *AWS CLI Command Reference*. For more information about change sets, see Updating stacks using change sets in the *AWS CloudFormation User Guide*.

Corresponding UI: Configuration tab/Advanced/**Fail on empty changeset**

disable-rollback

(*DeployCloudFormationStack*/Configuration/**disable-rollback**)

(Optional)

Specify whether you want CodeCatalyst to roll back the stack deployment if it fails. The rollback returns the stack to the last known stable state. The default is to enable rollbacks. For more information about this setting, see the AWS CloudFormation deploy parameter in the *AWS CLI Command Reference*.

For more information about how the **Deploy AWS CloudFormation stack** action handles rollbacks, see Configuring rollbacks.

For more information about rolling back a stack, see Stack failure options in the *AWS CloudFormation User Guide*.

Corresponding UI: Configuration tab/Advanced/**Disable rollback**

termination-protection

(*DeployCloudFormationStack*/Configuration/**termination-protection**)

(Optional)

Specify whether you want the **Deploy AWS CloudFormation stack** to add termination protection to the stack that it is deploying. If a user attempts to delete a stack with termination protection enabled, the deletion fails and the stack, including its status, remains unchanged. The default is to disable termination protection. For more information, see Protecting a stack from being deleted in the *AWS CloudFormation User Guide*.

Corresponding UI: Configuration tab/Advanced/**Termination protection**

timeout-in-minutes

(*DeployCloudFormationStack*/Configuration/**timeout-in-minutes**)

(Optional)

Specify the amount of time, in minutes, that CloudFormation should allot before timing out stack creation operations and setting the stack status to `CREATE_FAILED`. If CloudFormation can't create the entire stack in the time allotted, it fails the stack creation due to timeout and rolls back the stack.

By default, there is no timeout for stack creation. However, individual resources may have their own timeouts based on the nature of the service they implement. For example, if an individual resource in your stack times out, stack creation also times out even if the timeout you specified for stack creation hasn't yet been reached.

Corresponding UI: Configuration tab/Advanced/**CloudFormation timeout**

notification-arns

(*DeployCloudFormationStack*/Configuration/**notification-arns**)

(Optional)

Specify the ARN of an Amazon SNS topic that you want CodeCatalyst to send notification messages to. For example, `arn:aws:sns:us-east-1:111222333:MyTopic`. When the **Deploy**

AWS CloudFormation stack action runs, CodeCatalyst coordinates with CloudFormation to send one notification per AWS CloudFormation event that occurs during the stack creation or update process. (The events are visible in the AWS CloudFormation console's **Events** tab for the stack.) You can specify up to five topics. For more information, see What is Amazon SNS?.

Corresponding UI: Configuration tab/Advanced/**Notification ARNs**

monitor-alarm-arns

(*DeployCloudFormationStack*/Configuration/**monitor-alarm-arns**)

(Optional)

Specify the Amazon Resource Name (ARN) of an Amazon CloudWatch alarm to use as a rollback trigger. For example, `arn:aws:cloudwatch::123456789012:alarm/MyAlarm`. You can have a maximum of five rollback triggers.

> **Note**
>
> If you specify a CloudWatch alarm ARN, you'll also need to configure additional permissions to enable the action to access CloudWatch. For more information, see Configuring rollbacks.

Corresponding UI: Configuration tab/Advanced/**Monitor alarm ARNs**

monitor-timeout-in-minutes

(*DeployCloudFormationStack*/Configuration/**monitor-timeout-in-minutes**)

(Optional)

Specify an amount of time, from 0 to 180 minutes, during which CloudFormation monitors the specified alarms. Monitoring begins *after* all the stack resources have been deployed. If the alarm occurs within the specified monitoring time, then the deployment fails, and CloudFormation rolls back the entire stack operation.

Default: 0. CloudFormation only monitors alarms while the stack resources are being deployed, not after.

Corresponding UI: Configuration tab/Advanced/**Monitoring time**

tags

(*DeployCloudFormationStack*/Configuration/**tags**)

(Optional)

Specify tags to attach to your CloudFormation stack. Tags are arbitrary key-value pairs that you can use to identify your stack for purposes such as cost allocation. For more information about what tags are and how they can be used, see Tagging your resources in the *Amazon EC2 User Guide*. For more information about tagging in CloudFormation, see Setting AWS CloudFormation stack options in the *AWS CloudFormation User Guide*.

A key can have alphanumeric characters or spaces, and can have up to 127 characters. A value can have alphanumeric characters or spaces, and can have up to 255 characters.

You can add up to 50 unique tags for each stack.

Corresponding UI: Configuration tab/Advanced/**Tags**

Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow

This section describes how to deploy an AWS CDK app into your AWS account using a workflow. To accomplish this, you must add the **AWS CDK deploy** action to your workflow. The **AWS CDK deploy** action synthesizes and deploys your AWS Cloud Development Kit (AWS CDK) app into AWS. If your app already exists in AWS, the action updates it if necessary.

For general information about writing apps using the AWS CDK, see What is the AWS CDK? in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

When to use the "AWS CDK deploy" action

Use this action if you have developed an app using the AWS CDK, and you now want to deploy it automatically as part of automated continuous integration and delivery (CI/CD) workflow. For example, you might want to deploy your AWS CDK app automatically whenever someone merges a pull request related to your AWS CDK app source.

How the "AWS CDK deploy" action works

The **AWS CDK deploy** works as follows:

1. At runtime, if you specified version 1.0.12 or earlier of the action, the action downloads the latest CDK CLI (also called the AWS CDK Tookit) to the CodeCatalyst build image.

   If you specified version 1.0.13 or later, the action comes bundled with a specific version of the CDK CLI, so no download occurs.
2. The action uses the CDK CLI to run the `cdk deploy` command. This command synthesizes and deploys your AWS CDK app into AWS. For more information about this command, see the AWS CDK Toolkit (cdk command) topic in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

CDK CLI versions used by the "AWS CDK deploy" action

The following table shows which version of the CDK CLI is used by default by different versions of the **AWS CDK deploy** action.

> **Note**
>
> You might be able to override the default. For more information, see CdkCliVersion in the "AWS CDK deploy" action YAML definition.

"AWS CDK deploy" action version	AWS CDK CLI version
1.0.0 – 1.0.12	latest
1.0.13 or later	2.99.1

How many stacks can the action deploy?

The **AWS CDK deploy** can deploy a single stack only. If your AWS CDK app consists of multiple stacks, you must create a parent stack with nested stacks, and deploy the parent using this action.

Topics

- Example workflow that deploys an AWS CDK app
- Adding the "AWS CDK deploy" action
- Variables produced by the "AWS CDK deploy" action
- "AWS CDK deploy" action YAML definition

Example workflow that deploys an AWS CDK app

The following example workflow includes the **AWS CDK deploy** action, along with the **AWS CDK bootstrap** action. The workflow consists of the following building blocks that run sequentially:

- A **trigger** – This trigger starts the workflow run automatically when you push a change to your source repository. This repository contains your AWS CDK app. For more information about triggers, see Starting a workflow run automatically with triggers.
- An **AWS CDK bootstrap** action (`CDKBootstrap`) – On trigger, the action deploys the `CDKToolkit` bootstrap stack into AWS. If the `CDKToolkit` stack already exists in the environment, it will be upgraded if necessary; otherwise, nothing happens, and the action is marked as succeeded.
- An **AWS CDK deploy** action (`AWS CDKDeploy`) – On completion of the **AWS CDK bootstrap** action, the **AWS CDK deploy** action synthesizes your AWS CDK app code into an AWS CloudFormation template and deploys the stack defined in the template into AWS.

> **Note**
>
> The following workflow example is for illustrative purposes, and will not work without additional configuration.

> **Note**
>
> In the YAML code that follows, you can omit the `Connections:` sections if you want. If you omit these sections, you must ensure that the role specified in the **Default IAM role** field in your environment includes the permissions and trust policies required by the **AWS CDK bootstrap** and **AWS CDK deploy** actions. For more information about setting up an environment with a default IAM role, see Creating an environment. For more information about the permissions and trust policies required by the **AWS CDK bootstrap** and **AWS CDK deploy** actions, see the description of the `Role` property in the "AWS CDK bootstrap" action YAML definition and "AWS CDK deploy" action YAML definition.

```
Name: codecatalyst-cdk-deploy-workflow
SchemaVersion: 1.0
```

```
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  CDKBootstrap:
    Identifier: aws/cdk-bootstrap@v1
    Inputs:
      Sources:
        - WorkflowSource
    Environment:
      Name: codecatalyst-cdk-deploy-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-cdk-bootstrap-role
    Configuration:
      Region: us-west-2

  CDKDeploy:
    Identifier: aws/cdk-deploy@v1
    DependsOn:
      - CDKBootstrap
    Environment:
      Name: codecatalyst-cdk-deploy-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-cdk-deploy-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      StackName: my-app-stack
      Region: us-west-2
```

Adding the "AWS CDK deploy" action

Use the following instructions to add the **AWS CDK deploy** action to your workflow.

Before you begin

Before you can add the **AWS CDK deploy** action to your workflow, complete the following tasks:

1. **Have an AWS CDK app ready**. You can write your AWS CDK app using AWS CDK v1 or v2, in any programming language supported by the AWS CDK. Make sure your AWS CDK app files are available in:
   - A CodeCatalyst source repository, or
   - A CodeCatalyst output artifact generated by another workflow action
2. **Bootstrap your AWS environment**. To bootstrap, you can:
   - Use one of the methods described in How to bootstrap in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.
   - Use the **AWS CDK bootstrap** action. You can add this action in the same workflow as your **AWS CDK deploy**, or in a different one. Just make sure the bootstrap action runs at least once prior to running the **AWS CDK deploy** action so that the necessary resources are in place. For more information about the**AWS CDK bootstrap** action, see Bootstrapping an AWS CDK app with a workflow.

     For more information about bootstrapping, see Bootstrapping in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

Visual

To add the "AWS CDK deploy" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **AWS CDK deploy** action, and do one of the following:
   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

Or

- Choose **AWS CDK deploy**. The action details dialog box appears. On this dialog box:
  - (Optional) Choose **Download** to view the action's source code.
  - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.

10. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the "AWS CDK deploy" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and then choose **Commit** again.

> **Note**
>
> If your **AWS CDK deploy** action fails with an `npm install` error, see How do I fix "npm install" errors? for information about how to fix the error.

YAML

To add the "AWS CDK deploy" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **AWS CDK deploy** action, and do one of the following:

- Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

  Or

- Choose **AWS CDK deploy**. The action details dialog box appears. On this dialog box:
  - (Optional) Choose **Download** to view the action's source code.
  - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.

10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "AWS CDK deploy" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and then choose **Commit** again.

> **Note**
>
> If your **AWS CDK deploy** action fails with an `npm install` error, see How do I fix "npm install" errors? for information about how to fix the error.

Variables produced by the "AWS CDK deploy" action

The **AWS CDK deploy** action produces and sets the following variables at run time. These are known as *predefined variables*.

For information about referencing these variables in a workflow, see Using predefined variables.

Key	Value
stack-id	The Amazon Resource Name (ARN) of the AWS CDK application stack that was deployed to during the workflow run. Example: `arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cdk-app-stack/6a`

Key	Value
	`ad4380-100a-11ec-a10a-03b8a84d40df`
deployment-platform	The name of the deployment platform. Hardcoded to `AWS:CloudFormation` .
region	The region code of the AWS Region that was deployed to during the workflow run. Example: `us-west-2`
SKIP-DEPLOYMENT	A value of `true` indicates that deployment of your AWS CDK application stack was skipped during the workflow run. A stack deployment will be skipped if there is no change in the stack since the last deployment. This variable is only produced if its value is `true`. Hardcoded to `true`.

Key	Value
AWS CloudFormation variables	In addition to generating the variables listed previously, the **AWS CDK deploy** action also exposes *CloudFormation* output variables as *workflow* variables for use in subsequent workflow actions. By default, the action only exposes the first four (or fewer) CloudForm ation variables that it finds. To determine which ones are exposed, run the **AWS CDK deploy** action once, and then look in the **Variables** tab of the run details page. If the variables listed on the **Variables** tab are not what you want, you can configure different ones using the `CfnOutputVariables` YAML property. For more information, see the CfnOutputVariables property description in the "AWS CDK deploy" action YAML definition.

"AWS CDK deploy" action YAML definition

The following is the YAML definition of the **AWS CDK deploy** action. To learn how to use this action, see Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.
```

```
Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  CDKDeploy_nn:
    Identifier: aws/cdk-deploy@v1
    DependsOn:
      - CDKBootstrap
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - artifact-name
    Outputs:
      Artifacts:
        - Name: cdk_artifact
          Files:
            - "cdk.out/**/*"
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      StackName: my-cdk-stack
      Region: us-west-2
      Tags: '{"key1": "value1", "key2": "value2"}'
      Context: '{"key1": "value1", "key2": "value2"}'
      CdkCliVersion: version
      CdkRootPath: directory-containing-cdk.json-file
      CfnOutputVariables: '["CnfOutputKey1","CfnOutputKey2","CfnOutputKey3"]'
      CloudAssemblyRootPath: path-to-cdk.out
```

CDKDeploy

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `CDKDeploy_nn`.

Corresponding UI: Configuration tab/**Action name**

Identifier

(*CDKDeploy*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/cdk-deploy@v1`.

Corresponding UI: Workflow diagram/CDKDeploy_nn/**aws/cdk-deploy@v1** label

DependsOn

(*CDKDeploy*/**DependsOn**)

(Optional)

Specify an action or action group that must run successfully in order for the **AWS CDK deploy** action to run. We recommend specifying the **AWS CDK bootstrap** action in the `DependsOn` property, like this:

```
CDKDeploy:
  Identifier: aws/cdk-deploy@v1
  DependsOn:
    - CDKBootstrap
```

Note

Bootstrapping is a mandatory prerequisite for deploying an AWS CDK app. If you do not include the **AWS CDK Bootstrap** action in your workflow, then you must find another way to deploy the AWS CDK bootstrap stack before running your **AWS CDK deploy** action. For

more information, see Adding the "AWS CDK deploy" action in Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*CDKDeploy*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*CDKDeploy*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/Advanced - optional/**Compute type**

Fleet

(*CDKDeploy*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/Advanced - optional/**Compute fleet**

Timeout

(*CDKDeploy*/**Timeout**)

(Required)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Inputs

(*CDKDeploy*/**Inputs**)

(Optional)

The `Inputs` section defines the data that the `CDKDeploy` needs during a workflow run.

> **Note**
>
> Only one input (either a source or an artifact) is allowed for each **AWS CDK deploy** action.

Corresponding UI: **Inputs** tab

Sources

(*CDKDeploy*/Inputs/**Sources**)

(Required if the AWS CDK app you want to deploy is stored in a source repository)

If your AWS CDK app is stored in a source repository, specify the label of that source repository. The **AWS CDK deploy** action synthesizes the app in this repository before starting the deployment process. Currently, the only supported label is `WorkflowSource`.

If your AWS CDK app is not contained within a source repository, it must reside in an artifact generated by another action.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*CDKDeploy*/Inputs/**Artifacts**)

(Required if the AWS CDK app you want to deploy is stored in an output artifact from a previous action)

If your AWS CDK app is contained in an artifact generated by a previous action, specify that artifact here. The **AWS CDK deploy** action synthesizes the app in the specified artifact into a CloudFormation template before starting the deployment process. If your AWS CDK app is not contained within an artifact, it must reside in your source repository.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Inputs tab/**Artifacts - optional**

Outputs

(*CDKDeploy*/**Outputs**)

(Optional)

Defines the data that is output by the action during a workflow run.

Corresponding UI: **Outputs** tab

Artifacts - output

(*CDKDeploy*/Outputs/**Artifacts**

(Optional)

Specify the artifacts generated by the action. You can reference these artifacts as input in other actions.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/**Artifacts**

Name

(*CDKDeploy*/Outputs/Artifacts/**Name**)

(Required if Artifacts - output is included)

Specify the name of the artifact that will contain the AWS CloudFormation template that is synthesized by the **AWS CDK deploy** action at runtime. The default value is `cdk_artifact`. If you do not specify an artifact, then the action synthesizes the template but won't save it in an artifact. Consider saving the synthesized template in an artifact to preserve a record of it for testing or troubleshooting purposes.

Corresponding UI: Outputs tab/Artifacts/Add artifact/**Build artifact name**

Files

(*CDKDeploy*/Outputs/Artifacts/**Files**)

(Required if Artifacts - output is included)

Specify the files to include in the artifact. You must specify `"cdk.out/**/*"` to include your AWS CDK app's synthesized AWS CloudFormation template.

> **Note**
>
> `cdk.out` is the default directory into which synthesized files are saved. If you specified an output directory other than `cdk.out` in your `cdk.json` file, specify that directory here instead of `cdk.out`.

Corresponding UI: Outputs tab/Artifacts/Add artifact/**Files produced by build**

Environment

(*CDKDeploy*/**Environment**)

(Required)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*CDKDeploy*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*CDKDeploy*/Environment/**Connections**)

(Optional in newer versions of the action; required in older versions)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Name

(*CDKDeploy*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Role

(*CDKDeploy*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the account connection.

Specify the name of the IAM role that the **AWS CDK deploy** action uses to access AWS and deploy the AWS CDK application stack. Make sure that you have added the role to your CodeCatalyst space, and that the role includes the following policies.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

- The following permissions policy:

> **Warning**
>
> Limit the permissions to those shown in the following policy. Using a role with broader permissions might pose a security risk.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeChangeSet",
                "cloudformation:DescribeStacks",
                "cloudformation:ListStackResources"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::aws-account:role/cdk-*"
        }
    ]
}
```

- The following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Note

You can use the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *`my-environment`*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**Role**

Configuration

(*`CDKDeploy`*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

StackName

(*CDKDeploy*/Configuration/**StackName**)

(Required)

The name of your AWS CDK app stack, as it appears in the entrypoint file in your AWS CDK app's `bin` directory. The following example shows the contents of a TypeScript entrypoint file, with the stack name highlighted in *red italics*. If your entrypoint file is in a different language, it will look similar.

```
import * as cdk from 'aws-cdk-lib';
import { CdkWorksopTypescriptStack } from '../lib/cdk_workshop_typescript-stack';

const app = new cdk.App();
new CdkWorkshopTypescriptStack(app, 'CdkWorkshopTypescriptStack');
```

You can only specify one stack.

> **Tip**
>
> If you have multiple stacks, you can create a parent stack with nested stacks. You can then specify the parent stack in this action to deploy all stacks.

Corresponding UI: Configuration tab/**Stack name**

Region

(*CDKDeploy*/Configuration/**Region**)

(Optional)

Specify the AWS Region into which the AWS CDK application stack will be deployed. For a list of Region codes, see Regional endpoints.

If you do not specify a Region, the **AWS CDK deploy** action deploys into the Region specified in your AWS CDK code. For more information, see Environments in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

Corresponding UI: Configuration tab/**Region**

Tags

(*CDKDeploy*/Configuration/**Tags**)

(Optional)

Specify tags that you want to apply to the AWS resources in the AWS CDK application stack. Tags are applied to the stack itself as well as to individual resources in the stack. For more information about tagging, see Tagging in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

Corresponding UI: Configuration tab/Advanced - optional/**Tags**

Context

(*CDKDeploy*/Configuration/**Context**)

(Optional)

Specify contexts, in the form of key-value pairs, to associate with the AWS CDK application stack. For more information about contexts, see Runtime contexts in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

Corresponding UI: Configuration tab/Advanced - optional/**Context**

CdkCliVersion

(*CDKDeploy*/Configuration/**CdkCliVersion**)

(Optional)

This property is available with version 1.0.13 or later of the **AWS CDK deploy** action, and version 1.0.8 or later of the **AWS CDK bootstrap** action.

Specify one of the following:

- The full version of the AWS Cloud Development Kit (AWS CDK) Command Line Interface (CLI) (also called the AWS CDK Toolkit) that you want this action to use. Example: `2.102.1`. Consider specifying a full version to ensure consistency and stability when building and deploying your application.

Or

- `latest`. Consider specifying `latest` to take advantage of the latest features and fixes of the CDK CLI.

The action will download the specified version (or the latest version) of the AWS CDK CLI to the CodeCatalyst build image, and then use this version to run the commands necessary to deploy your CDK application or bootstrap your AWS environment.

For a list of supported CDK CLI versions you can use, see AWS CDK Versions.

If you omit this property, the action uses a default AWS CDK CLI version described in one of the following topics:

- CDK CLI versions used by the "AWS CDK deploy" action
- CDK CLI versions used by the "AWS CDK bootstrap" action

Corresponding UI: Configuration tab/**AWS CDK CLI version**

CdkRootPath

(*CDKDeploy*/Configuration/**CdkRootPath**)

(Optional)

The path to the directory that contains your AWS CDK project's `cdk.json` file. The **AWS CDK deploy** action runs from this folder, and any outputs created by the action will be added to this directory. If unspecified, the **AWS CDK deploy** action assumes that the `cdk.json` file is in the root of your AWS CDK project.

Corresponding UI: Configuration tab/**Directory where the cdk.json resides**

CfnOutputVariables

(*CDKDeploy*/Configuration/**CfnOutputVariables**)

(Optional)

Specify which `CfnOutput` constructs in your AWS CDK application code you want to expose as workflow output variables. You can then reference the workflow output variables in subsequent

actions in your workflow. For more information about variables in CodeCatalyst, see Configuring and using variables in a workflow.

For example, if your AWS CDK application code looks like this:

```
import { Duration, Stack, StackProps, CfnOutput, RemovalPolicy} from 'aws-cdk-lib';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
export class HelloCdkStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const bucket = new s3.Bucket(this, 'my-bucket', {
      removalPolicy: RemovalPolicy.DESTROY,
    });
    new CfnOutput(this, 'bucketName', {
      value: bucket.bucketName,
      description: 'The name of the s3 bucket',
      exportName: 'myBucket',
    });
    const table = new dynamodb.Table(this, 'todos-table', {
      partitionKey: {name: 'todoId', type: dynamodb.AttributeType.NUMBER},
      billingMode: dynamodb.BillingMode.PAY_PER_REQUEST,
      removalPolicy: RemovalPolicy.DESTROY,
    })
    new CfnOutput(this, 'tableName', {
      value: table.tableName,
      description: 'The name of the dynamodb table',
      exportName: 'myDynamoDbTable',
    });
    ...
  }
}
```

...and your `CfnOutputVariables` property looks like this:

```
Configuration:
  ...
  CfnOutputVariables: '["bucketName","tableName"]'
```

...then the action generates the following workflow output variables:

Key	Value
bucketName	`bucket.bucketName`
tableName	`table.tableName`

You can then reference the `bucketName` and `tableName` variables in subsequent actions. To learn how to reference workflow output variables in subsequent actions, see Referencing a predefined variable.

If you do not specify any `CfnOutput` constructs in the `CfnOutputVariables` property, then the action exposes the first four (or fewer) CloudFormation output variables it finds as workflow output variables. For more information, see Variables produced by the "AWS CDK deploy" action .

Tip

To obtain a list of all the CloudFormation output variables the action produces, run the workflow containing the **AWS CDK deploy** action once, and then look in the action's **Logs** tab. The logs contain a list of all the CloudFormation output variables associated with your AWS CDK app. Once you know what all the CloudFormation variables are, you can specify which ones you want to convert to workflow output variables using the `CfnOutputVariables` property.

For more information about AWS CloudFormation output variables, see the documentation for the `CfnOutput` construct, available at class CfnOutput (construct) in the *AWS Cloud Development Kit (AWS CDK) API Reference*.

Corresponding UI: Configuration tab/**AWS CloudFormation output variables**

CloudAssemblyRootPath

(*CDKDeploy*/Configuration/**CloudAssemblyRootPath**)

(Optional)

If you have already synthesized your AWS CDK app's stack into a cloud assembly (using the `cdk synth` operation), specify the root path of the cloud assembly directory (`cdk.out`). The AWS

CloudFormation template located in the specified cloud assembly directory will be deployed by the **AWS CDK deploy** action into your AWS account using the `cdk deploy --app` command. When the `--app` option is present, the `cdk synth` operation does not occur.

If you do not specify a cloud assembly directory, then the **AWS CDK deploy** action will run the `cdk deploy` command without the `--app` option. Without the `--app` option, the `cdk deploy` operation will both synthesize (`cdk synth`) and deploy your AWS CDK app into your AWS account.

Why would I specify an existing, synthesized cloud assembly when the "AWS CDK deploy" action can do the synthesis at run time?

You might want to specify an existing, synthesized cloud assembly to:

- **Ensure that the exact same set of resources are deployed every time the "AWS CDK deploy" action runs**

  If you don't specify a cloud assembly, it's possible for the **AWS CDK deploy** action to synthesize and deploy different files depending on when it is run. For example, the **AWS CDK deploy** action might synthesize a cloud assembly with one set of dependencies during a testing stage, and another set of dependencies during a production stage (if those dependencies changed between stages). To guarantee exact parity between what is tested and what is deployed, we recommend synthesizing once and then using the **Path to cloud assembly directory** field (visual editor) or `CloudAssemblyRootPath` property (YAML editor) to specify the already-synthesized cloud assembly.
- **Use non-standard package managers and tooling with the AWS CDK app**

  During a `synth` operation, the **AWS CDK deploy** action tries to run your app using standard tools such as npm or pip. If the action can't successfully run your app using those tools, the synthesis will not occur and the action will fail. To work around this issue, you can specify the exact commands needed to run your app successfully in the AWS CDK app's `cdk.json` file, and then synthesize your app using a method that does not involve the **AWS CDK deploy** action. After the cloud assembly has been generated, you can specify it in the **Path to cloud assembly directory** field (visual editor) or `CloudAssemblyRootPath` property (YAML editor) of the **AWS CDK deploy** action.

For information about configuring the `cdk.json` file to include commands for installing and running your AWS CDK app, see Specifying the app command.

For information about the `cdk deploy` and `cdk synth` commands, as well as the `--app` option, see Deploying stacks, Synthesizing stacks and Skipping synthesis in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

For information about cloud assemblies, see Cloud Assembly in the *AWS Cloud Development Kit (AWS CDK) API Reference*.

Corresponding UI: Configuration tab/**Path to cloud assembly directory**

Bootstrapping an AWS CDK app with a workflow

This section describes how to bootstrap an AWS CDK application using a CodeCatalyst workflow. To accomplish this, you must add the **AWS CDK bootstrap** action to your workflow. The **AWS CDK bootstrap** action provisions a bootstrap stack in your AWS environment using the modern template. If a bootstrap stack already exists, the action updates it if necessary. Having a bootstrap stack present in AWS is a prerequisite for deploying an AWS CDK app.

For more information about bootstrapping, see Bootstrapping in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

When to use the "AWS CDK bootstrap" action

Use this action if you have a workflow that deploys an AWS CDK app, and you want to deploy (and update, if needed) the bootstrap stack at the same time. In this case, you would add the **AWS CDK bootstrap** action to the same workflow as the one that deploys your AWS CDK app.

Do not use this action if either of the following applies:

- You already deployed a bootstrap stack using another mechanism, and you want to keep it intact (no updates).
- You want to use a custom bootstrap template, which is not supported with the **AWS CDK bootstrap** action.

How the "AWS CDK bootstrap" action works

The **AWS CDK bootstrap** works as follows:

1. At runtime, if you specified version 1.0.7 or earlier of the action, the action downloads the latest CDK CLI (also called the AWS CDK Tookit) to the CodeCatalyst build image.

If you specified version 1.0.8 or later, the action comes bundled with a specific version of the CDK CLI, so no download occurs.

2. The action uses the CDK CLI to run the `cdk bootstrap` command. This command performs the bootstrapping tasks described in the Bootstrapping topic in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

CDK CLI versions used by the "AWS CDK bootstrap" action

The following table shows which version of the CDK CLI is used by default by different versions of the **AWS CDK bootstrap** action.

> **Note**
>
> You might be able to override the default. For more information, see CdkCliVersion in the "AWS CDK bootstrap" action YAML definition.

"AWS CDK bootstrap" action version	AWS CDK CLI version
1.0.0 – 1.0.7	latest
1.0.8 or later	2.99.1

Topics

- Example workflow that bootstraps an AWS CDK app
- Adding the "AWS CDK bootstrap" action
- Variables produced by the "AWS CDK bootstrap" action
- "AWS CDK bootstrap" action YAML definition

Example workflow that bootstraps an AWS CDK app

Refer to the Example workflow that deploys an AWS CDK app in the Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow for a workflow that includes the **AWS CDK bootstrap** action.

Adding the "AWS CDK bootstrap" action

Use the following instructions to add the **AWS CDK bootstrap** action to your workflow.

Before you begin

Before you can use the **AWS CDK bootstrap** action, make sure you have an AWS CDK app ready. The bootstrap action will synthesize the AWS CDK app before bootstrapping. You can write your app in any programming language supported by the AWS CDK.

Make sure your AWS CDK app files are available in:

- A CodeCatalyst source repository, or
- A CodeCatalyst output artifact generated by another workflow action

Visual

To add the "AWS CDK bootstrap" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **AWS CDK bootstrap** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **AWS CDK bootstrap**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **View source** to view the action's source code.

- Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.

10. In the **Inputs**, **Configuration**, and **Outputs** tabs, complete the fields according to your needs. For a description of each field, see the "AWS CDK bootstrap" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and then choose **Commit** again.

> **Note**
>
> If your **AWS CDK bootstrap** action fails with an `npm install` error, see How do I fix "npm install" errors? for information about how to fix the error.

YAML

To add the "AWS CDK bootstrap" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **AWS CDK bootstrap** action, and choose **+** to add it to the workflow diagram and open its configuration pane.
10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "AWS CDK bootstrap" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and then choose **Commit** again.

> **Note**
>
> If your **AWS CDK bootstrap** action fails with an `npm install` error, see How do I fix "npm install" errors? for information about how to fix the error.

Variables produced by the "AWS CDK bootstrap" action

The **AWS CDK bootstrap** action produces and sets the following variables at run time. These are known as *predefined variables*.

For information about referencing these variables in a workflow, see Using predefined variables.

Key	Value
deployment-platform	The name of the deployment platform. Hardcoded to `AWS:CloudFormation` .
region	The region code of the AWS Region that the AWS CDK bootstrap stack was deployed to during the workflow run. Example: `us-west-2`
stack-id	The Amazon Resource Name (ARN) of the deployed AWS CDK bootstrap stack. Example: `arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cdk-bootstrap-stack/6aad4380-100a-11ec-a10a-03b8a84d40df`
SKIP-DEPLOYMENT	A value of `true` indicates that deployment of your AWS CDK bootstrap stack was skipped during the workflow run. A stack deployment will be skipped if there is no change in the stack since the last deployment.

Key	Value
	This variable is only produced if its value is `true`. Hardcoded to `true`.

"AWS CDK bootstrap" action YAML definition

The following is the YAML definition of the **AWS CDK bootstrap** action. To learn how to use this action, see Bootstrapping an AWS CDK app with a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  CDKBootstrapAction_nn:
    Identifier: aws/cdk-bootstrap@v1
    DependsOn:
      - action-name
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
```

```
      Sources:
        - source-name-1
      Artifacts:
        - artifact-name
    Outputs:
      Artifacts:
        - Name: cdk_bootstrap_artifacts
          Files:
            - "cdk.out/**/*"
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      Region: us-west-2
      CdkCliVersion: version
```

CDKBootstrapAction

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `CDKBootstrapAction_nn`.

Corresponding UI: Configuration tab/**Action display name**

Identifier

(*CDKBootstrapAction*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/cdk-bootstrap@v1`.

Corresponding UI: Workflow diagram/CDKBootstrapAction_nn/**aws/cdk-bootstrap@v1** label

DependsOn

(*CDKBootstrapAction*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*CDKBootstrapAction*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*CDKBootstrapAction*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/Advanced - optional/**Compute type**

Fleet

(*CDKBootstrapAction*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/Advanced - optional/**Compute fleet**

Timeout

(*CDKBootstrapAction*/**Timeout**)

(Required)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Inputs

(*CDKBootstrapAction*/**Inputs**)

(Optional)

The `Inputs` section defines the data that the **AWS CDK bootstrap** action needs during a workflow run.

Corresponding UI: **Inputs** tab

> **Note**
>
> Only one input (either a source or an artifact) is allowed for each **AWS CDK bootstrap** action.

Sources

(*CDKBootstrapAction*/Inputs/**Sources**)

(Required if your AWS CDK app is stored in a source repository)

If your AWS CDK app is stored in a source repository, specify the label of that source repository. The **AWS CDK bootstrap** action synthesizes the app in this repository before starting the bootstrapping process. Currently, the only supported repository label is `WorkflowSource`.

If your AWS CDK app is not contained within a source repository, it must reside in an artifact generated by another action.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*CDKBootstrapAction*/Inputs/**Artifacts**)

(Required if your AWS CDK app is stored in an output artifact from a previous action)

If your AWS CDK app is contained in an artifact generated by a previous action, specify that artifact here. The **AWS CDK bootstrap** action synthesizes the app in the specified artifact into a CloudFormation template before starting the bootstrapping process. If your AWS CDK app is not contained within an artifact, it must reside in your source repository.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Inputs tab/**Artifacts - optional**

Outputs

(*CDKBootstrapAction*/**Outputs**)

(Optional)

Defines the data that is output by the action during a workflow run.

Corresponding UI: **Outputs** tab

Artifacts - output

(*CDKBootstrapAction*/Outputs/**Artifacts**)

(Optional)

Specify the artifacts generated by the action. You can reference these artifacts as input in other actions.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/**Artifacts**

Name

(*CDKBootstrapAction*/Outputs/Artifacts/**Name**)

(Required if Artifacts - output is included)

Specify the name of the artifact that will contain the AWS CloudFormation template that is synthesized by the **AWS CDK bootstrap** action at runtime. The default value is `cdk_bootstrap_artifacts`. If you do not specify an artifact, then the action synthesizes the template, but won't save it in an artifact. Consider saving the synthesized template in an artifact to preserve a record of it for testing or troubleshooting purposes.

Corresponding UI: Outputs tab/Artifacts/Add artifact/**Build artifact name**

Files

(*CDKBootstrapAction*/Outputs/Artifacts/**Files**)

(Required if Artifacts - output is included)

Specify the files to include in the artifact. You must specify `"cdk.out/**/*"` to include your AWS CDK app's synthesized AWS CloudFormation template.

> **Note**
>
> `cdk.out` is the default directory into which synthesized files are saved. If you specified an output directory other than `cdk.out` in your `cdk.json` file, specify that directory here instead of `cdk.out`.

Corresponding UI: Outputs tab/Artifacts/Add artifact/**Files produced by build**

Environment

(*CDKBootstrapAction*/**Environment**)

(Required)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*CDKBootstrapAction*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*CDKBootstrapAction*/Environment/**Connections**)

(Optional in newer versions of the action; required in older versions)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Name

(*CDKBootstrapAction*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**

- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Role

(*CDKBootstrapAction*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that the **AWS CDK bootstrap** action uses to access AWS and add the bootstrap stack. Make sure that you have added the role to your CodeCatalyst space, and that the role includes the following policies.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

> **Note**
>
> The permissions shown in the following permissions policy are those required by the `cdk bootstrap` command to perform its bootstrapping at the time of writing. These permissions may change if the AWS CDK changes its bootstrap command.

> **Warning**
>
> Only use this role with the **AWS CDK bootstrap** action. It is very permissive, and using it with other actions might pose a security risk.

- The following permissions policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "ssm:GetParameterHistory",
```

```
                "ecr:PutImageScanningConfiguration",
                "cloudformation:*",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "ssm:GetParameters",
                "iam:PutRolePolicy",
                "ssm:GetParameter",
                "ssm:DeleteParameters",
                "ecr:DeleteRepository",
                "ssm:PutParameter",
                "ssm:DeleteParameter",
                "iam:PassRole",
                "ecr:SetRepositoryPolicy",
                "ssm:GetParametersByPath",
                "ecr:DescribeRepositories",
                "ecr:GetLifecyclePolicy"
            ],
            "Resource": [
                "arn:aws:ssm:aws-region:aws-account:parameter/cdk-bootstrap/*",
                "arn:aws:cloudformation:aws-region:aws-account:stack/CDKToolkit/*",
                "arn:aws:ecr:aws-region:aws-account:repository/cdk-*",
                "arn:aws:iam::aws-account:role/cdk-*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "cloudformation:RegisterType",
                "cloudformation:CreateUploadBucket",
                "cloudformation:ListExports",
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:SetTypeDefaultVersion",
                "cloudformation:RegisterPublisher",
                "cloudformation:ActivateType",
                "cloudformation:ListTypes",
                "cloudformation:DeactivateType",
                "cloudformation:SetTypeConfiguration",
                "cloudformation:DeregisterType",
                "cloudformation:ListTypeRegistrations",
                "cloudformation:EstimateTemplateCost",
                "cloudformation:DescribeAccountLimits",
                "cloudformation:BatchDescribeTypeConfigurations",
                "cloudformation:CreateStackSet",
```

```
                "cloudformation:ListStacks",
                "cloudformation:DescribeType",
                "cloudformation:ListImports",
                "s3:*",
                "cloudformation:PublishType",
                "ecr:CreateRepository",
                "cloudformation:DescribePublisher",
                "cloudformation:DescribeTypeRegistration",
                "cloudformation:TestType",
                "cloudformation:ValidateTemplate",
                "cloudformation:ListTypeVersions"
            ],
            "Resource": "*"
        }
    ]
}
```

Note

The first time the role is used, use the following wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

- The following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-spaceName` role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-spaceName` role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**Role**

Configuration

(*CDKBootstrapAction*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

Region

(*CDKBootstrapAction*/Configuration/**Region**)

(Required)

Specify the AWS Region into which the bootstrap stack will be deployed. This Region should match the one into which your AWS CDK app is deployed. For a list of Region codes, see Regional endpoints.

Corresponding UI: Configuration tab/**Region**

CdkCliVersion

(*CDKBootstrapAction*/Configuration/**CdkCliVersion**)

(Optional)

This property is available with version 1.0.13 or later of the **AWS CDK deploy** action, and version 1.0.8 or later of the **AWS CDK bootstrap** action.

Specify one of the following:

- The full version of the AWS Cloud Development Kit (AWS CDK) Command Line Interface (CLI) (also called the AWS CDK Toolkit) that you want this action to use. Example: `2.102.1`. Consider specifying a full version to ensure consistency and stability when building and deploying your application.

  Or

- `latest`. Consider specifying `latest` to take advantage of the latest features and fixes of the CDK CLI.

The action will download the specified version (or the latest version) of the AWS CDK CLI to the CodeCatalyst build image, and then use this version to run the commands necessary to deploy your CDK application or bootstrap your AWS environment.

For a list of supported CDK CLI versions you can use, see AWS CDK Versions.

If you omit this property, the action uses a default AWS CDK CLI version described in one of the following topics:

- CDK CLI versions used by the "AWS CDK deploy" action
- CDK CLI versions used by the "AWS CDK bootstrap" action

Corresponding UI: Configuration tab/**AWS CDK CLI version**

Publishing files to Amazon S3 with a workflow

This section describes how to publish files to Amazon S3 using a CodeCatalyst workflow. To accomplish this, you must add the **Amazon S3 publish** action to your workflow. The **Amazon S3 publish** action copies files from a source directory to an Amazon S3 bucket. The source directory can reside in:

- A source repository, or
- An output artifact generated by another workflow action

When to use the "Amazon S3 publish" action

Use this action if:

- You have a workflow that generates files that you want to store in Amazon S3.

  For example, you might have a workflow that builds a static website that you want to host in Amazon S3. In this case, your workflow would include a build action to build the site's HTML and supporting files, and an **Amazon S3 publish** action to copy the files to Amazon S3.
- You have a source repository that contains files that you want to store in Amazon S3.

  For example, you might have a source repository with application source files that you want to archive on a nightly basis to Amazon S3.

Topics

- Example workflow that publishes files to Amazon S3
- Adding the "Amazon S3 publish" action
- "Amazon S3 publish" action YAML definition

Example workflow that publishes files to Amazon S3

The following example workflow includes the **Amazon S3 publish** action, along with a build action. The workflow builds a static documentation website and then publishes it to Amazon S3, where it is hosted. The workflow consists of the following building blocks that run sequentially:

- A **trigger** – This trigger starts the workflow run automatically when you push a change to your source repository. For more information about triggers, see Starting a workflow run automatically with triggers.
- A **build** action (`BuildDocs`) – On trigger, the action builds a static documentation website (`mkdocs build`) and adds the associated HTML files and supporting metadata to an artifact called `MyDocsSite`. For more information about the build action, see Building with workflows.
- An **Amazon S3 publish** action (`PublishToS3`) – On completion of the build action, this action copies the site in the `MyDocsSite` artifact to Amazon S3 for hosting.

Note

The following workflow example is for illustrative purposes, and will not work without additional configuration.

Note

In the YAML code that follows, you can omit the `Connections:` section if you want. If you omit this section, you must ensure that the role specified in the **Default IAM role** field in your environment includes the permissions and trust policies required by the **Amazon S3 publish** action. For more information about setting up an environment with a default IAM role, see Creating an environment. For more information about the permissions and trust policies required by the **Amazon S3 publish** action, see the description of the Role property in the "Amazon S3 publish" action YAML definition.

```
Name: codecatalyst-s3-publish-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildDocs:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: echo BuildDocs started on `date`
        - Run: pip install --upgrade pip
        - Run: pip install mkdocs
        - Run: mkdocs build
        - Run: echo BuildDocs completed on `date`
    Outputs:
      Artifacts:
      - Name: MyDocsSite
```

```
      Files:
        - "site/**/*"

PublishToS3:
  Identifier: aws/s3-publish@v1
  Environment:
    Name: codecatalyst-s3-publish-environment
    Connections:
      - Name: codecatalyst-account-connection
        Role: codecatalyst-s3-publish-build-role
  Inputs:
    Sources:
      - WorkflowSource
    Artifacts:
      - MyDocsSite
  Configuration:
    DestinationBucketName: my-bucket
    SourcePath: /artifacts/PublishToS3/MyDocSite/site
    TargetPath: my/docs/site
```

Adding the "Amazon S3 publish" action

Use the following instructions to add the **Amazon S3 publish** action to your workflow.

Visual

To add the "Amazon S3 publish" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Amazon S3 publish** action, and do one of the following:

- Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

  Or

- Choose **Amazon S3 publish**. The action details dialog box appears. On this dialog box:
  - (Optional) Choose **View source** to view the action's source code.
  - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.

10. In the **Inputs**, **Configuration**, and **Outputs** tabs, complete the fields according to your needs. For a description of each field, see the "Amazon S3 publish" action YAML definition. This reference provides detailed information on each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and then choose **Commit** again.

YAML

To add the "Amazon S3 publish" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Amazon S3 publish** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

- Choose **Amazon S3 publish**. The action details dialog box appears. On this dialog box:
  - (Optional) Choose **View source** to view the action's source code.
  - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.

10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "Amazon S3 publish" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and then choose **Commit** again.

"Amazon S3 publish" action YAML definition

The following is the YAML definition of the **Amazon S3 publish** action. To learn how to use this action, see Publishing files to Amazon S3 with a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  S3Publish_nn:
    Identifier: aws/s3-publish@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
```

```
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      Sources:
        - source-name-1
      Artifacts:
        - artifact-name
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      SourcePath: my/source
      DestinationBucketName: s3-bucket-name
      TargetPath: my/target
```

S3Publish

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `S3Publish_nn`.

Corresponding UI: Configuration tab/**Action name**

Identifier

(*S3Publish*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/s3-publish@v1`.

Corresponding UI: Workflow diagram/S3Publish_nn/**aws/s3-publish@v1** label

DependsOn

(*S3Publish*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*S3Publish*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*S3Publish*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/**Compute type**

Fleet

(*S3Publish*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/**Compute fleet**

Timeout

(*S3Publish*/**Timeout**)

(Required)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Inputs

(*S3Publish*/**Inputs**)

(Optional)

The `Inputs` section defines the data that the `S3Publish` needs during a workflow run.

> **Note**
>
> A maximum of four inputs (one source and three artifacts) are allowed for each **AWS CDK deploy** action. Variables do not count towards this total.

If you need to refer to files residing in different inputs (say a source and an artifact), the source input is the primary input, and the artifact is the secondary input. References to files in secondary inputs take a special prefix to distiguish them from the primary. For details, see Example: Referencing files in multiple artifacts.

Corresponding UI: **Inputs** tab

Sources

(*S3Publish*/Inputs/**Sources**)

(Required if the files you want to publish to Amazon S3 are stored in a source repository)

If the files that you want to publish to Amazon S3 are stored in a source repository, specify the label of that source repository. Currently, the only supported label is `WorkflowSource`.

If the files that you want to publish to Amazon S3 are not contained within a source repository, they must reside in an artifact generated by another action.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*S3Publish*/Inputs/**Artifacts**)

(Required if the files you want to publish to Amazon S3 are stored in an output artifact from a previous action)

If the files that you want to publish to Amazon S3 are contained in an artifact generated by a previous action, specify that artifact here. If your files are not contained within an artifact, they must reside in your source repository.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Configuration tab/**Artifacts - optional**

Variables - input

(*S3Publish*/Inputs/**Variables**)

(Optional)

Specify a sequence of name/value pairs that define the input variables that you want to make available to the action. Variable names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in variable names.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Inputs tab/**Variables - optional**

Environment

(*S3Publish*/**Environment**)

(Required)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*S3Publish*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*S3Publish*/Environment/**Connections**)

(Optional in newer versions of the action; required in older versions)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Name

(*S3Publish*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Role

(*S3Publish*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that the **Amazon S3 publish** action uses to access AWS and to copy files to Amazon S3. Make sure that you have added the role to your CodeCatalyst space, and that the role includes the following policies.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

- The following permissions policy:

> **Warning**
>
> Limit the permissions to those shown in the following policy. Using a role with broader permissions might pose a security risk.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket",
```

```
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

- The following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-spaceName` role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-spaceName` role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**Role**

Configuration

(*S3Publish*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

SourcePath

(*S3Publish*/Configuration/**SourcePath**)

(Required)

Specify the name and path of a directory or file that you want to publish to Amazon S3. The directory or file can reside in a source repository or an artifact from a previous action, and is relative to the source repository or artifact root.

Examples:

Specifying `./myFolder/` copies the contents of `/myFolder` to Amazon S3, and preserves the underlying directory structure.

Specifying `./myFolder/myfile.txt` copies *just* `myfile.txt` to Amazon S3. (The directory structure is removed.)

You cannot use wildcards.

> **Note**
>
> You may need to add a prefix to the directory or file path to indicate which artifact or source to find it in. For more information, see Referencing files in a source repository and Referencing files in an artifact.

Corresponding UI: Configuration tab/**Source path**

DestinationBucketName

(*S3Publish*/Configuration/**DestinationBucketName**)

(Required)

Specify the name of the Amazon S3 bucket where you want to publish files.

Corresponding UI: Configuration tab/**Destination bucket - optional**

TargetPath

(*S3Publish*/Configuration/**TargetPath**)

(Optional)

Specify the name and path of the directory in Amazon S3 where you want to publish your files. If the directory does not exist, it will be created. The directory path must not include the bucket name.

Examples:

`myS3Folder`

`./myS3Folder/myS3Subfolder`

Corresponding UI: Configuration tab/**Destination directory - optional**

Deploying into AWS accounts and VPCs with CodeCatalyst environments

Using CodeCatalyst workflows, you can deploy applications and other resources to target AWS accounts and Amazon VPCs in the AWS cloud. To enable these deployments, you must set up CodeCatalyst environments.

A CodeCatalyst *environment*, not to be confused with a Dev Environment, defines the target AWS account and optional Amazon VPC that a CodeCatalyst workflow connects to. An environment also defines the IAM role that a workflow needs to access the AWS services and resources within the target account.

You can set up multiple environments and give them names such as development, test, staging, and production. When you deploy into these environments, information about the deployments appears on the CodeCatalyst **Deployment activity** and **Deployment targets** tabs in the environment.

How do I get started with environments?

The high-level steps to add and use a CodeCatalyst environment are as follows:

1. In your CodeCatalyst space, **connect one or more AWS accounts**. During this process, add the IAM roles that your workflow requires to access resources in your AWS account. For more information, see Allowing access to AWS resources with connected AWS accounts.
2. In your CodeCatalyst project, **create an environment** that includes one of the AWS accounts and IAM roles from step 1. For more information, see Creating an environment.
3. In your CodeCatalyst project, in a workflow, **add an action that points to the environment** you created in step 2. For more information, see Adding an action to a CodeCatalyst workflow.

   You have now configured an environment. The action can now deploy resources into the AWS account specified in the environment.

> **Note**
>
> You can also add an Amazon VPC to the environment. For more information, see Adding VPC connections for a space in the *CodeCatalyst Administration Guide* and Associating a VPC connection with an environment.

Can multiple environments exist within a single workflow?

Yes. If a workflow includes multiple actions, each of those actions can be assigned an environment. For example, you could have a workflow that includes two deploy actions, where one is assigned a `my-staging-enviroment` environment and another is assigned a `my-production-environment` environment.

Which workflow actions support environments?

Any workflow action that deploys resources into the AWS cloud, or communicates with AWS services for other reasons (such as monitoring and reporting), supports environments.

Which actions support having their deployment information displayed in CodeCatalyst?

Of the workflow actions that support environments, only a few support having their deployment information displayed on the **Deployment activity** and **Deployment targets** pages of the CodeCatalyst console.

The following workflow actions support having their deployment information displayed:

- **Deploy AWS CloudFormation stack** – For more information, see Deploying an AWS CloudFormation stack with a workflow
- **Deploy to Amazon ECS** – For more information, see Deploying an application to Amazon Elastic Container Service (ECS) with a workflow
- **Deploy to Kubernetes cluster** – For more information, see Deploying an application to Amazon Elastic Kubernetes Service with a workflow
- **AWS CDK deploy** – For more information, see Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow

Supported Regions

The **Environments** page can display resources in any AWS Region.

Is an environment mandatory?

An environment is mandatory if the workflow action to which it is assigned deploys resources into the AWS cloud, or communicates with AWS services for other reasons (such as monitoring and reporting).

For example, if you have a build action that builds an application but doesn't need to communicate with your AWS account or Amazon VPC, then you do not need to assign an environment to the action. If, however, the build action sends logs to the Amazon CloudWatch service in your AWS account, then the action must have an environment assigned.

Topics

- Creating an environment
- Associating an environment with a workflow action
- Associating a VPC connection with an environment
- Associating an AWS account with an environment

- Assigning a different IAM role to an action

Creating an environment

Use the following instructions to create an environment that you can later associate with a workflow action.

Before you begin

You need the following:

- A CodeCatalyst space. For more information, see Set up and sign in to CodeCatalyst.
- A CodeCatalyst project. For more information, see Creating a project with a blueprint.
- An AWS account connection that includes the IAM roles your workflow action will need to access AWS. You can use a maximum of one account connection per environment. For more information, see Allowing access to AWS resources with connected AWS accounts.

> **Note**
>
> You can create an environment without an account connection; however, you will need to come back and add the connection later.

To create an environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Environments**.
4. In **Environment name**, enter a name, such as **`Production`** or **`Staging`**.
5. In **Environment type**, select one of the following:

   - **Non-production** – An environment where you can test your application to make sure it's working as intended before moving it into production.
   - **Production** – A 'live' environment that is publicly-available and hosts your finalized application.

     If you choose **Production**, a **Production** badge appears in the UI next to any actions that the environment is associated with. The badge helps you quickly see which actions are deploying

to production. Other than the appearance of the badge, there are no differences between production and non-production environments.

6. (Optional) In **Description**, enter a description such as **`Production environment for the hello-world app`**.
7. In **AWS account connection - optional**, choose the AWS account connection you want to associate with this environment. Make sure the account connection includes the IAM role that you want to associate with the environment. For more information about creating this connection, see Allowing access to AWS resources with connected AWS accounts.
8. In **Default IAM role**, choose the IAM role you want to associate with this environment. Workflow actions that are assigned this environment will inherit this IAM role, and will be able to use it to connect to services and resources in your AWS account. If you need to assign this environment to multiple actions, and those actions need an IAM role that is different from the default one specified here, then you can specify a different role on the action's **Configuration** tab, using the **Switch role** option. For more information, see Assigning a different IAM role to an action.
9. (Optional) In **VPC connection**, choose a VPC connection you want to associate with this environment. For more information about creating this VPC connection, see Managing Amazon Virtual Private Clouds in the *CodeCatalyst Administrator Guide*.
10. Choose **Create environment**. CodeCatalyst creates an empty environment.

Next steps

- Now that you have created an environment, you are ready to associate it with a workflow action. For more information, see Associating an environment with a workflow action.

Associating an environment with a workflow action

When you associate an environment with a supported workflow action, the environment's AWS account, default IAM role, and optional Amazon VPC become assigned to the action. The action can then connect and deploy to the AWS account using the IAM role, and also connect to the optional Amazon VPC.

Use the following instructions to associate an environment with an action.

Step 1: Associate the environment with a workflow action

Use the following procedure to associate an environment with a workflow action.

Visual

To associate an environment with a workflow action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose an action that is supported with environments. For more information, see Which actions support having their deployment information displayed in CodeCatalyst?.
8. Choose the **Configuration** tab, and specify information in the **Environment** field, as follows.

   Environment

   Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

   > **Note**
   >
   > If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

   For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.
9. (Optional) Change the IAM role associated with the action. You might want to change the role if it contains the wrong set of permissions for the action.

   To change the role:

1. In the **What's in *my-environment* ?** box, and choose the vertical ellipsis icon (⋮).
2. Choose one of the following:
   - **Switch role**. Choose this option to change the IAM role used by this action, and only this action. Other actions continue to use the default IAM role specified in their associated environment. For more information, see Assigning a different IAM role to an action.
   - **Edit environment**. Choose this option to change the default IAM role listed in your environment. When you choose this option, your action—and any other action associated with the same environment—begins using the new default IAM role.

     > **Important**
     >
     > Use caution when updating the default IAM role. Changing the role might lead to action failures if the permissions in the role are not sufficient for all actions that share the environment.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To associate an environment with a workflow action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In the workflow action that you want to associate with an environment, add code similar to the following:

```
action-name:
```

```
  Environment:
    Name: environment-name
```

For more information, see the Action types topic. This topic has links into the documentation for each action, including its YAML reference.

8. (Optional) If you want the action to use a different role from the default IAM role that's listed in the environment, add a `Connections:` section that includes the role you want to use. For more information, see Assigning a different IAM role to an action.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

Step 2: Add deployment information to CodeCatalyst

After associating an environment with a workflow action, you can populate the **Deployment activity** and **Deployment target** pages in the CodeCatalyst console with deployment information. Use the following instructions to populate these pages.

> **Note**
>
> Only a few actions support having their deployment information displayed in the CodeCatalyst console. For more information, see Which actions support having their deployment information displayed in CodeCatalyst?.

To add deployment information to CodeCatalyst

1. If a workflow run did not start automatically when you committed your changes in Step 1: Associate the environment with a workflow action, manually start a run as follows:

   a. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.

   b. Choose the name of the workflow where you want to start a run. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

   c. Choose **Run**.

   The workflow run starts a new deployment, which causes CodeCatalyst to add deployment information to CodeCatalyst.

2. Verify that deployment activity was added to the CodeCatalyst console:

    a. In the navigation pane, choose **CI/CD**, and then choose **Environments**.

    b. Choose your environment (for example, `Production`).

    c. Choose the **Deployment activity** tab, and verify that a deployment appears with a **Status** of **SUCCEEDED**. This indicates that a workflow run successfully deployed your application resources.

    d. Choose the **Deployment targets** tab, and verify that your application resources appear.

Associating a VPC connection with an environment

When an action is configured with an environment that has a VPC connection, the action will run connected to the VPC, adhering to the network rules and access resources specified by the associated VPC. The same VPC connection can be used by one or more environments.

Use the following instructions to associate a VPC connection with an environment.

To associate a VPC connection with an environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Environments**.
4. Choose your environment (for example, `Production`).
5. Choose the **Environment properties** tab.
6. Choose **Manage VPC connection**, choose your desired VPC connection, and choose **Confirm**. This associates your selected VPC connection with this environment.

For more information, see Managing Amazon Virtual Private Clouds in the *CodeCatalyst Administrator Guide*.

Associating an AWS account with an environment

Use the following instructions to associate an AWS account with an environment.

To associate an AWS account with an environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Environments**.
4. Choose your environment (for example, `Production`).
5. Choose the **Environment properties** tab.
6. Choose **Associate AWS account**, choose your desired AWS account, and choose **Associate**. This associates your selected AWS account with this environment.

For more information, see Allowing access to AWS resources with connected AWS accounts.

Assigning a different IAM role to an action

By default, when you associate an environment with a workflow action, the action inherits the default IAM role specified in the environment. You can change this behavior so that the action uses a different role. You might want an action to use a different role if the default IAM role is missing the permissions that the action needs to operate in the AWS cloud.

To assign a different IAM role to an action, you can use the **Switch role** option in the visual editor or the `Connections:` property in the YAML editor. The new role overrides the default IAM role specified in the environment, allowing you to keep the default IAM role as-is. You might want to keep the default IAM role as-is if there are other actions that use it.

Use the following instructions to configure an action to use a different IAM role from the one specified in its environment.

Visual

To assign a different IAM role to an action (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose the box that represents the action whose IAM role you want to update.
7. Choose the **Configuration** tab.

8. In the **What's in *my-environment* ?** box, choose the vertical ellipsis icon (⋮).
9. Choose **Switch role**.
10. In the **Switch role** dialog box, in the **IAM role** drop-down list, choose the IAM role that you want the action to use. This role will override the default IAM role in the environment. If the role you want to use is not in the list, make sure you've added it to your space. For more information, see Adding IAM roles to account connections.

    The chosen role now appears in the **What's in *my-environment*?** box along with a **Defined in workflow** badge. The role also appears in the workflow definition file, in the `Connections:` section.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To assign a different IAM role to an action (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In the workflow action where you want to use a different IAM role, add a `Connections:` section, similar to the following:

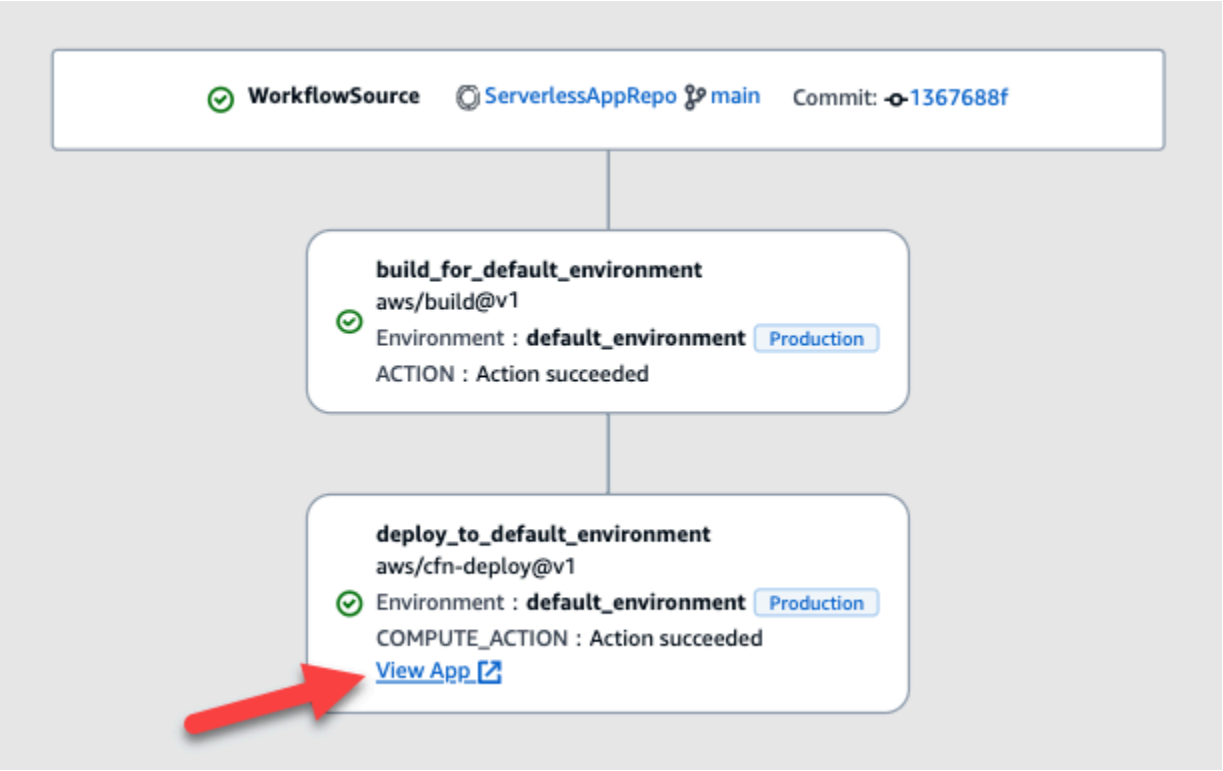
```
action-name:
  Environment:
    Name: environment-name
    Connections:
      - Name: account-connection-name
        Role: iam-role-name
```

In the preceding code, replace *account-connection-name* with the name of the account connection that contains the IAM role, and replace *iam-role-name* with the name of the IAM role that you want the action to use. This role will override the default IAM role in the environment. Make sure you've added the role to your space. For more information, see Adding IAM roles to account connections.

For more information, see the Action types topic. This topic has links into the documentation for each action, including its YAML reference.

Displaying the URL of the deployed application in the workflow diagram

If your workflow deploys an application, you can configure Amazon CodeCatalyst to display the application's URL as a clickable link. This link appears in the CodeCatalyst console, inside the action that deployed it. The following workflow diagram shows the **View App** URL appearing at the bottom of an action.



By making this URL clickable in the CodeCatalyst console, you can quickly verify your application deployment.

> **Note**
>
> The app URL is not supported with the **Deploy to Amazon ECS** action.

To enable this feature, add an output variable to your action with a name that contains `appurl`, or `endpointurl`. You can use a name with or without a joining dash (-), underscore (_), or space ( ). The string is case-insensitive. Set the variable's value to the `http` or `https` URL of your deployed application.

> **Note**
>
> If you're updating an existing output variable to include the `app url`, or `endpoint url` string, update all references to this variable to use the new variable name.

For detailed steps, see one of the following procedures:

- To display the app URL in the "AWS CDK deploy" action
- To display the app URL in the "Deploy AWS CloudFormation stack" action
- To display the app URL in all other actions

When you've finished configuring the URL, verify that it appears as expected by following these instructions:

- To verify that the application URL was added

To display the app URL in the "AWS CDK deploy" action

1. If you're using the **AWS CDK deploy** action, add a `CfnOutput` construct (which is a key-value pair) in your AWS CDK application code:

   - The key name must contain `appurl`, or `endpointurl`, with or without a joining dash (-), underscore (_), or space ( ). The string is case-insensitive.
   - The value must be the `http` or `https` URL of your deployed application.

   For example, your AWS CDK code might look like this:

```
import { Duration, Stack, StackProps, CfnOutput, RemovalPolicy} from 'aws-cdk-lib';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
export class HelloCdkStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const bucket = new s3.Bucket(this, 'my-bucket', {
      removalPolicy: RemovalPolicy.DESTROY,
    });
    new CfnOutput(this, 'APP-URL', {
      value: https://mycompany.myapp.com,
      description: 'The URL of the deployed application',
      exportName: 'myApp',
    });
    ...
  }
}
```

For more information about the `CfnOutput` construct, see interface CfnOutputProps in the *AWS Cloud Development Kit (AWS CDK) API Reference*.

2. Save and commit your code.
3. Proceed to To verify that the application URL was added.

To display the app URL in the "Deploy AWS CloudFormation stack" action

1. If you're using the **Deploy AWS CloudFormation stack** action, add an output to the `Outputs` section in your CloudFormation template or AWS SAM template with these characteristics:

   - The key (also called the logical ID) must contain `appurl`, or `endpointurl`, with or without a joining dash (-), underscore (_), or space ( ). The string is case-insensitive.
   - The value must be the `http` or `https` URL of your deployed application.

   For example, your CloudFormation template might look like this:

```
"Outputs" : {
  "APP-URL" : {
```

```
    "Description" : "The URL of the deployed app",
    "Value" : "https://mycompany.myapp.com",
    "Export" : {
      "Name" : "My App"
    }
  }
}
```

For more information about CloudFormation outputs, see Outputs in the *AWS CloudFormation User Guide*.

2. Save and commit your code.
3. Proceed to To verify that the application URL was added.

To display the app URL in all other actions

If you're using another action to deploy your application, such as the build action or **GitHub Actions**, do the following to have the app URL displayed.

1. Define an environment variable in the `Inputs` or `Steps` section of the action in the workflow definition file. The variable must have these characteristics:

   - The `name` must contain `appurl`, or `endpointurl`, with or without a joining dash (-), underscore (_), or space ( ). The string is case-insensitive.
   - The value must be the `http` or `https` URL of your deployed application.

   For example, a build action might look like this:

```
Build-action:
  Identifier: aws/build@v1
  Inputs:
    Variables:
      - Name: APP-URL
        Value: https://mycompany.myapp.com
```

   ...or this:

```
Actions:
  Build:
    Identifier: aws/build@v1
```

```
    Configuration:
      Steps:
        - Run: APP-URL=https://mycompany.myapp.com
```

For more information about defining environment variables, see Defining a variable.

2. Export the variable.

   For example, your build action might look like this:

```
Build-action:
  ...
  Outputs:
    Variables:
      - APP-URL
```

   For information about exporting variables, see Exporting a variable so that other actions can use it.

3. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
4. Choose **Commit**, enter a commit message, and choose **Commit** again.
5. Proceed to To verify that the application URL was added.

To verify that the application URL was added

- Start a workflow run, if it hasn't started automatically. The new run should have the app URL displayed as a clickable link in its workflow diagram. For more information about starting runs, see Starting a workflow run manually.

Removing a deployment target

You can remove a deployment target such as an Amazon ECS cluster or AWS CloudFormation stack from the **Deployment targets** page in the CodeCatalyst console.

> **Important**
>
> When you remove a deployment target, it is removed from the CodeCatalyst console, but remains available in the AWS service that hosts it (if it still exists).

Consider removing a deployment target if the target has become stale in CodeCatalyst. Targets might become stale if:

- You deleted the workflow that deployed to the target.
- You changed the stack or cluster that you're deploying to.
- You deleted the stack or cluster from the CloudFormation or Amazon ECS service in the AWS console.

To remove a deployment target

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Environments**.
4. Choose the name of the environment that contains the deployment target you want to remove. For information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.
5. Choose the **Deployment targets** tab.
6. Choose the radio button next to the deployment target you want to remove.
7. Choose **Remove**.

   The target is removed from the page.

Tracking deployment status by commit

At any time in the development lifecycle, it's important to know the deployment status of specific commits, such as bug fixes, new features, or other impactful changes. Consider the following scenarios in which deployment status tracking capability is helpful to development teams:

- As a developer, you've made a fix to address a bug and you want to report the status of its deployment across your team's deployment environments.
- As a release manager, you want to view a list of deployed commits to track and report their deployment status.

CodeCatalyst provides a view you can use to determine at a glance where individual commits or changes have been deployed, and to which environment. This view includes:

- A list of commits.
- The status of deployments that include the commits.
- The environments in which the commits are successfully deployed.
- The status of any tests run against the commits in your CI/CD workflow.

The following procedure details how to navigate to and use this view to track changes in your project.

> **Note**
>
> Tracking deployment status by commit is only supported with CodeCatalyst repositories. You cannot use this feature with a GitHub repository, Bitbucket repository, or GitLab project repository.

To track deployment status by commit

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Change tracking**.
4. In the two dropdown lists at the top of the main pane, choose the source repository and branch that contain the commits whose release status you want to view.
5. Choose **View changes**.

   A list of commits appears.

   For each commit, you can view the following:

   - Commit information such as ID, author, message, and when it was committed. For more information, see Store and collaborate on code with source repositories in CodeCatalyst.
   - The status of deployments to each environment. For more information, see Deploying into AWS accounts and VPCs with CodeCatalyst environments.
   - Test and code coverage results. For more information, see Testing with workflows.

> **Note**
>
> Software Composition Analysis (SCA) results are not displayed.

6. (Optional) To view more information about the changes related to a specific commit, including the latest deployment and detailed code coverage and unit test information, choose **View details** for that commit.

Viewing the deployment logs

You can view logs related to specific deploy actions to troubleshoot problems in Amazon CodeCatalyst.

You can view logs starting from a workflow, or an environment.

To view the logs of a deploy action starting from a workflow

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Runs**.
6. Choose the workflow run that deployed your application.
7. In the workflow diagram, choose the action whose logs you want to view.
8. Choose the **Logs** tab and expand the sections to reveal the log messages.
9. To view more logs, choose the **Summary** tab, and then choose **View in CloudFormation** (if it's available) to view more logs there. You may need to sign in to AWS.

To view the logs of a deploy action starting from an environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Environments**.

4. Choose the environment into which your application was deployed.
5. In **Deployment activity**, find the **Workflow Run ID** column, and choose the workflow run that deployed your stack.
6. In the workflow diagram, choose the action whose logs you want to view.
7. Choose the **Logs** tab and expand the sections to reveal the log messages.
8. To view more logs, choose the **Summary** tab, and then choose **View in CloudFormation** (if it's available) to view more logs there. You may need to sign in to AWS.

Viewing deployment status, commits, and pull requests

You can view the following information about a deployment in Amazon CodeCatalyst:

- Deployment activity, including the deployment status, start time, end time, history, and duration of events.
- Stack name, AWS Region, last update time, and associated workflows.
- Commits and pull requests.
- Action-specific information, for example, CloudFormation events and outputs.

You can view deployment information starting from a workflow, an environment, or a workflow action.

To view deployment information starting from a workflow

- Go to the workflow run that deployed your application. For instructions, see Viewing workflow run status and details.

To view deployment information starting from an environment

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Environments**.
4. Choose the environment where your stack was deployed, for example, `Production`.
5. Choose **Deployment activity** to view the deployment history of your stacks, the status of the deployments (for example, **SUCCEEDED** or **FAILED**), and other deployment-related information.

6. Choose **Deployment target** to view information about the stacks, clusters, or other targets deployed into the environment. You can view information such as the stack name, Region, provider, and identifier.

To view deployment information starting from an action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. In the workflow diagram, choose the workflow action that deployed your application. For example, you might choose **DeployCloudFormationStack**.
6. Review the contents in the right pane for action-specific deployment information.

Creating a workflow

A *workflow* is an automated procedure that describes how to build, test, and deploy your code as part of a continuous integration and continuous delivery (CI/CD) system. A workflow defines a series of steps, or *actions*, to take during a workflow run. A workflow also defines the events, or *triggers*, that cause the workflow to start. To set up a workflow, you create a *workflow definition file* using the CodeCatalyst console's visual or YAML editor.

> **Tip**
>
> For a quick look at how you might use workflows in a project, create a project with a blueprint. Each blueprint deploys a functioning workflow that you can review, run, and experiment with.

Use the following procedures to create a workflow in CodeCatalyst.

For more information about workflows, see Build, test, and deploy with workflows in CodeCatalyst.

Visual

To create a workflow using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose **Create workflow**.

   The **Create workflow** dialog box appears.
5. In the **Source repository** field, choose a source repository where the workflow definition file will reside. The file will be stored in the `~/.codecatalyst/workflows/` folder in the chosen repository. If no source repository exists, create one.
6. In the **Branch** field, choose a branch where the workflow definition file will reside.
7. Choose **Create**.

   Amazon CodeCatalyst saves the repository and branch information in memory, but the workflow is not yet committed.
8. Choose **Visual**.
9. Build the workflow:

   a. (Optional) In the workflow diagram, choose the **Source** and **Triggers** box. A **Triggers** pane appears. Choose **Add trigger** to add a trigger. For more information, see Adding a push, pull, or schedule trigger.

   b. Choose **+ Actions** (top-left). The **Actions** catalog appears.

   c. Choose the plus sign (**+**) inside an action to add it to the workflow. Use the pane on the right to configure the action. For more information, see Adding an action to a CodeCatalyst workflow.

   d. (Optional) Choose **Workflow properties** (top-right). A **Workflow properties** pane appears. Configure the workflow name run mode, and compute. For more information, see Configuring the queuing behavior of runs and Configuring the compute and runtime environment Docker images for a workflow.
10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, and on the **Commit workflow** dialog box, do the following:

    a. For **Workflow file name**, leave the default name or enter your own.

b. For **Commit message**, leave the default message or enter your own.

c. For **Repository** and **Branch**, choose the source repository and branch for the workflow definition file. These fields should be set to the repository and branch that you specified earlier in the **Create workflow** dialog box. You can change the repository and branch now, if you'd like.

> **Note**
>
> After committing your workflow definition file, it cannot be associated with another repository or branch, so make sure to choose them carefully.

d. Choose **Commit** to commit the workflow definition file.

YAML

To create a workflow using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose **Create workflow**.

   The **Create workflow** dialog box appears.
5. In the **Source repository** field, choose a source repository where the workflow definition file will reside. The file will be stored in the `~/.codecatalyst/workflows/` folder in the chosen repository. If no source repository exists, create one.
6. In the **Branch** field, choose a branch where the workflow definition file will reside.
7. Choose **Create**.

   Amazon CodeCatalyst saves the repository and branch information in memory, but the workflow is not yet committed.
8. Choose **YAML**.
9. Build the workflow:

   a. (Optional) Add a trigger to the YAML code. For more information, see Adding a push, pull, or schedule trigger.

b. Choose **+ Actions** (top-left). The **Actions** catalog appears.

c. Choose the plus sign (**+**) inside an action to add it to the workflow. Use the pane on the right to configure the action. For more information, see Adding an action to a CodeCatalyst workflow.

d. (Optional) Choose **Workflow properties** (top-right). A **Workflow properties** pane appears. Configure the workflow name, run mode, and compute. For more information, see Configuring the queuing behavior of runs and Configuring the compute and runtime environment Docker images for a workflow.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.

11. Choose **Commit**, and on the **Commit workflow** dialog box, do the following:

a. For **Workflow file name**, leave the default name or enter your own.

b. For **Commit message**, leave the default message or enter your own.

c. For **Repository** and **Branch**, choose the source repository and branch for the workflow definition file. These fields should be set to the repository and branch that you specified earlier in the **Create workflow** dialog box. You can change the repository and branch now, if you'd like.

> **Note**
>
> After committing your workflow definition file, it cannot be associated with another repository or branch, so make sure to choose them carefully.

d. Choose **Commit** to commit the workflow definition file.

Running a workflow

A *run* is a single iteration of a workflow. During a run, CodeCatalyst performs the actions defined in the workflow configuration file and outputs the associated logs, artifacts, and variables.

You can start a run manually, or you can start one automatically, through a *workflow trigger*. An example of a workflow trigger might be a software developer pushing a commit to your main branch.

You can also manually stop a workflow run midway through its processing if you started it by mistake.

If multiple workflow runs are started at around the same time, you can configure how you want these runs to be queued. You can use the default queuing behavior, where runs are queued one after the other in the order in which they were started, or you can have a later run supersede (or 'take over') from an earlier one to speed up your run throughout. Setting up your workflow runs to occur in parallel, so that no run waits for any other, is also possible.

After you've started a workflow run, either manually or automatically, you can view the status of the run and other details. For example, you can see when it was started, who it was started by, and whether it's still running.

Topics

Starting a workflow run manually

Use the following procedure to start a workflow run manually.

> **Note**
>
> You can also start a workflow run automatically by configuring a trigger.

To start a workflow run manually

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.

4. Choose the name of the workflow you want to run. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Run**.

Starting a workflow run automatically with triggers

A *workflow trigger*, or simply a *trigger*, allows you to start a workflow run automatically when certain events occur, like a code push. You might want to configure triggers to free your software developers from having to start workflow runs manually through the CodeCatalyst console.

You can use three types of trigger:

- **Push** – A code push trigger causes a workflow run to start whenever a commit is pushed.
- **Pull request** – A pull request trigger causes a workflow run to start whenever a pull request is either created, revised, or closed.
- **Schedule** – A schedule trigger causes a workflow run to start on a schedule that you define. Consider using a schedule trigger to run nightly builds of your software so that the latest build is ready for your software developers to work on the next morning.

You can use push, pull request, and schedule triggers alone or in combination in the same workflow.

Triggers are optional—if you don't configure any, you can only start a workflow manually.

> **Tip**
>
> To see a trigger in action, launch a project with a blueprint. Most blueprints contain a workflow with a trigger. Look for the `Trigger` property in the blueprint's workflow definition file. For more information about blueprints, see Creating a project with a blueprint.

Topics

- Examples of triggers

A common trigger configuration

This section describes how to set up triggers for a common software release and branching strategy.

Software release and branching strategy:

- You have application code in a source repository.
- Your `main` branch contains finalized code that is always release-ready.
- Your software developers make their changes in feature branches off the `main` branch.
- Your software developers create a pull request asking to merge their feature branch into `main` when their feature is ready.

  You want this pull request to start a workflow automatically that builds and tests—but does not deploy—the application using the files on the software developer's feature branch.
- You software developers check the build and the tests to make sure everything looks good. They then merge the pull request into the `main` branch.

  You want the merge to automatically start a workflow automatically that builds and deploys your application code.

Proposed workflow/trigger configuration:

Given the software branching strategy outlined previously, you might want to use two workflows:

- **Workflow 1** builds and tests your application when a pull request is created or revised.
- **Workflow 2** builds and deploys your application when a pull request is merged.

Workflow 1 would look like this:

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
```

```
      - REVISION
Actions:
  BuildAction:
    instructions-for-building-the-app
  TestAction:
    instructions-for-test-the-app
```

The previous trigger code automatically starts a workflow run whenever a software developer creates a pull request (or modifies one) asking to merge their feature branch to the `main` branch. CodeCatalyst starts a workflow run using the code in the source branch (that is, the developer's feature branch). The workflow builds and deploys the application.

Workflow 2 would look like this:

```
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildAction:
    instructions-for-building-the-app
  DeployAction:
    instructions-for-deploying-the-app
```

In the previous trigger code, when a merge to `main` occurs, the PUSH trigger is activated. CodeCatalyst starts a workflow run using the code in the `main` branch (which now includes the code from the pull request). The workflow builds and deploys the application.

For instructions on adding triggers to a workflow definition file, see Adding a push, pull, or schedule trigger.

For more examples of triggers and additional explanations, see Examples of triggers.

Trigger considerations when branching

This section describes some of the main considerations when setting up triggers that include branches.

- **Consideration 1:** For both push and pull request triggers, if you are going to specify a branch, you must specify the destination (or 'to') branch in the trigger configuration. Never specify the source (or 'from') branch.

In the following example, a push from any branch to `main` activates the workflow.

```
Triggers:
  - Type: PUSH
    Branches:
      - main
```

In the following example, a pull request from any branch into `main` activates the workflow.

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
      - REVISION
```

- **Consideration 2:** For push triggers, after the workflow is activated, the workflow will run using the workflow definition file and source files in the *destination* branch.
- **Consideration 3:** For pull request triggers, after the workflow is activated, the workflow will run using the workflow definition file and source files in the *source* branch (even though you specified the destination branch in the trigger configuration).
- **Consideration 4:** The exact same trigger in one branch might not run in another branch.

  Consider the following push trigger:

  ```
  Triggers:
    - Type: PUSH
      Branches:
        - main
  ```

  If the workflow definition file containing this trigger exists in `main` and gets cloned to `test`, the workflow will never start automatically using the files in `test` (although you could start the workflow *manually* to have if use the files in `test`). Review **Considerations 1** and **2** to understand why the workflow will never run automatically using the files in `test`.

  Consider also the following pull request trigger:

  ```
  Triggers:
  ```

```
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
      - REVISION
```

If the workflow definition file containing this trigger exists in `main`, the workflow will never run using the files in `main`. (However, if you create a `test` branch off of `main`, the workflow will run using the files in `test`.) Review **Considerations 1** and **3** to understand why.

Adding a push, pull, or schedule trigger

Use the following instructions to add a push, pull, or schedule trigger to your workflow.

Visual

To add a trigger (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the **Source** and **Triggers** box.
8. In the configuration pane, Choose **Add trigger**.
9. In the **Add trigger** dialog box, supply information in the fields, as follows.

   Trigger type

   Specify the type of trigger. You can use one of the following values:

   - **Push** (visual editor) or PUSH (YAML editor)

A push trigger starts a workflow run when a change is pushed to your source repository. The workflow run will use the files in the branch that you're pushing *to* (that is, the destination branch).

- **Pull request** (visual editor) or `PULLREQUEST` (YAML editor)

  A pull request trigger starts a workflow run when a pull request is opened, updated, or closed in your source repository. The workflow run will use the files in the branch that you're pulling *from* (that is, the source branch).

- **Schedule** (visual editor) or `SCHEDULE` (YAML editor)

  A schedule trigger starts workflow runs on a schedule defined by a cron expression that you specify. A separate workflow run will start for each branch in your source repository using the branch's files. (To limit the branches that the trigger activates on, use the **Branches** field (visual editor) or `Branches` property (YAML editor).)

  When configuring a schedule trigger, follow these guidelines:

  - Only use one schedule trigger per workflow.
  - If you've defined multiple workflows in your CodeCatalyst space, we recommend that you schedule no more than 10 of them to start concurrently.
  - Make sure you configure the trigger's cron expression with adequate time between runs. For more information, see Expression.

For examples, see Examples of triggers.

Events for pull request

This field only appears if you selected the **Pull request** trigger type.

Specify the type of pull request events that will start a workflow run. The following are the valid values:

- **Pull request is created** (visual editor) or `OPEN` (YAML editor)

  The workflow run is started when a pull request is created.

- **Pull request is closed** (visual editor) or `CLOSED` (YAML editor)

The workflow run is started when a pull request is closed. The `CLOSED` event's behavior is tricky, and is best understood through an example. See Example: A trigger with a pull, branches, and a 'CLOSED' event for more information.

- **New revision is made to pull request** (visual editor) or `REVISION` (YAML editor)

  The workflow run is started when a revision to a pull request is created. The first revision is created when the pull request is created. After that, a new revision is created every time someone pushes a new commit to the source branch specified in the pull request. If you include the `REVISION` event in your pull request trigger, you can omit the `OPEN` event, since `REVISION` is a superset of `OPEN`.

You can specify multiple events in the same pull request trigger.

For examples, see Examples of triggers.

Schedule

This field only appears if you selected the **Schedule** trigger type.

Specify the cron expression that describes when you want your scheduled workflow runs to occur.

Cron expressions in CodeCatalyst use the following six-field syntax, where each field is separated by a space:

```
minutes hours days-of-month month days-of-week year
```

Examples of cron expressions

Minutes	Hours	Days of month	Month	Days of week	Year	Meaning
0	0	?	*	MON-FRI	*	Runs a workflow at midnight (UTC+0) every Monday through Friday.
0	2	*	*	?	*	Runs a workflow at 2:00 am (UTC +0) every day.
15	22	*	*	?	*	Runs a workflow at 10:15 pm (UTC +0) every day.

Minutes	Hours	Days of month	Month	Days of week	Year	Meaning
0/30	22-2	?	*	SAT-SUN	*	Runs a workflow every 30 minutes Saturday through Sunday between 10:00 pm on the starting day and 2:00 am on the following day (UTC +0).
45	13	L	*	?	2023-2027	Runs a workflow at 1:45 pm (UTC +0) on the last day of the month between the years 2023 and 2027 inclusive.

When specifying cron expressions in CodeCatalyst, make sure you follow these guidelines:

- Specify a single cron expression per `SCHEDULE` trigger.
- Enclose the cron expression in double-quotes (") in the YAML editor.
- Specify the time in Coordinated Universal Time (UTC). Other time zones are not supported.
- Configure at least 30 minutes between runs. A faster cadence is not supported.
- Specify the *`days-of-month`* or *`days-of-week`* field, but not both. If you specify a value or an asterisk (`*`) in one of the fields, you must use a question mark (`?`) in the other. The asterisk means 'all' and the question mark means 'any'.

For more examples of cron expressions and information about wildcards like `?`, `*`, and `L`, see the Cron expressions reference in the *Amazon EventBridge User Guide*. Cron expressions in EventBridge and CodeCatalyst work exactly the same way.

For examples of schedule triggers, see Examples of triggers.

Branches and **Branch pattern**

(Optional)

Specify the branches in your source repository that the trigger monitors in order to know when to start a workflow run. You can use regex patterns to define your branch names. For example, use `main.*` to match all branches beginning with `main`.

The branches to specify are different depending on the trigger type:

- For a push trigger, specify the branches you're pushing *to*, that is, the *destination* branches. One workflow run will start per matched branch, using the files in the matched branch.

  Examples: `main.*`, `mainline`
- For a pull request trigger, specify the branches you're pushing *to*, that is, the *destination* branches. One workflow run will start per matched branch, using the workflow definition file and source files in the **source** branch (*not* the matched branch).

  Examples: `main.*`, `mainline`, `v1\-.*` (matches branches that start with `v1-`)

- For a schedule trigger, specify the branches that contain the files that you want your scheduled run to use. One workflow run will start per matched branch, using the the workflow definition file and source files in the matched branch.

  Examples: `main.*`, `version\-1\.0`

> **Note**
>
> If you *don't* specify branches, the trigger monitors all branches in your source repository, and will start a workflow run using the workflow definition file and source files in:
>
> - The branch you're pushing *to* (for push triggers). For more information, see Example: A simple code push trigger.
> - The branch you're pulling *from* (for pull request triggers). For more information, see Example: A simple pull request trigger.
> - All branches (for schedule triggers). One workflow run will start per branch in your source repository. For more information, see Example: A simple schedule trigger.

For more information about branches and triggers, see Trigger considerations when branching.

For more examples, see Examples of triggers.

Files changed

This field only appears if you selected the **Push** or **Pull request** trigger type.

Specify the files or folders in your source repository that the trigger monitors in order to know when to start a workflow run. You can use regular expressions to match file names or paths.

For examples, see Examples of triggers.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add a trigger (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. Add a `Triggers` section and underlying properties using the following example as a guide. For more information, see the Triggers in the Workflow YAML definition.

   A code push trigger might look like this:

   ```
   Triggers:
     - Type: PUSH
       Branches:
         - main
   ```

   A pull request trigger might look like this:

   ```
   Triggers:
     - Type: PULLREQUEST
       Branches:
         - main.*
       Events:
         - OPEN
         - REVISION
         - CLOSED
   ```

   A schedule trigger might look like this:

   ```
   Triggers:
     - Type: SCHEDULE
       Branches:
         - main.*
       # Run the workflow at 1:15 am (UTC+0) every Friday until the end of 2023
   ```

```
    Expression: "15 1 ? * FRI 2022-2023"
```

For more examples of cron expressions you can use in the `Expression` property, see Expression.

For more examples of push, pull request, and schedule triggers, see Examples of triggers.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Examples of triggers

The following examples show how to add different types of triggers in the workflow definition file.

Topics

Example: A simple code push trigger

The following example shows a trigger that starts a workflow run whenever code is pushed to *any* branch in your source repository.

When this trigger is activated, CodeCatalyst starts a workflow run using the files in the branch that you're pushing *to* (that is, the destination branch).

For example, if you push a commit to `main`, CodeCatalyst starts a workflow run using the workfow definition file and other source files on `main`.

As another example, if you push a commit to `feature-branch-123`, CodeCatalyst starts a workflow run using the workfow definition file and other source files on `feature-branch-123`.

```
Triggers:
  - Type: PUSH
```

> **Note**
>
> If you want a workflow run to start only when you push to `main`, see Example: A simple 'push to main' trigger.

Example: A simple 'push to main' trigger

The following example shows a trigger that starts a workflow run whenever code is pushed to the `main` branch—and *only* the `main` branch—in your source repository.

```
Triggers:
  - Type: PUSH
    Branches:
      - main
```

Example: A simple pull request trigger

The following example shows a trigger that starts a workflow run whenever a pull request is created or revised in your source repository.

When this trigger is activated, CodeCatalyst starts a workflow run using the workflow definition file and other source files in the branch that you're pulling *from* (that is, the source branch).

For example, if you create a pull request with a source branch called `feature-123` and a destination branch called `main`, CodeCatalyst starts a workflow run using the workfow definition file and other source files on `feature-123`.

```
Triggers:
  - Type: PULLREQUEST
    Events:
      - OPEN
      - REVISION
```

Example: A simple schedule trigger

The following example shows a trigger that starts a workflow run at midnight (UTC+0) every Monday through Friday.

When this trigger is activated, CodeCatalyst starts a single workflow run for each branch in your source repository that contains a workflow definition file with this trigger.

For example, if you have three branches in your source repository, `main`, `release-v1`, `feature-123`, and each of these branches contains a workflow definition file with the trigger that follows, CodeCatalyst starts three workflow runs: one using the files in `main`, another using the files in `release-v1`, and another using the files in `feature-123`.

```
Triggers:
  - Type: SCHEDULE
    Expression: "0 0 ? * MON-FRI *"
```

For more examples of cron expressions you can use in the `Expression` property, see Expression.

Example: A trigger with a schedule and branches

The following example shows a trigger that starts a workflow run at 6:15 pm (UTC+0) every day.

When this trigger is activated, CodeCatalyst starts a workflow run using the files in the `main` branch, and starts additional runs for each branch that begins with `release-`.

For example, if you have branches named `main`, `release-v1`, `bugfix-1`, and `bugfix-2` in your source repository, CodeCatalyst starts two workflow runs: one using the the files in `main`, and another using the files in `release-v1`. It does *not* start workflow runs for the `bugfix-1` and `bugfix-1` branches.

```
Triggers:
  - Type: SCHEDULE
    Expression: "15 18 * * ? *"
    Branches:
      - main
      - release\-.*
```

For more examples of cron expressions you can use in the `Expression` property, see Expression.

Example: A trigger with a schedule, a push, and branches

The following example shows a trigger that starts a workflow run at midnight (UTC+0) every day, and whenever code is pushed to the `main` branch.

In this example:

- A workflow run starts at midnight every day. The workflow run uses the workflow definition file and other source files in the `main` branch.
- A workflow run also starts whenever you push a commit to the `main` branch. The workflow run uses the workflow definition file and other source files in the destination branch (`main`).

```
Triggers:
  - Type: SCHEDULE
    Expression: "0 0 * * ? *"
    Branches:
      - main
  - Type: PUSH
    Branches:
      - main
```

For more examples of cron expressions you can use in the `Expression` property, see Expression.

Example: A trigger with a pull and branches

The following example shows a trigger that starts a workflow run whenever someone opens or modifies a pull request with a destination branch called `main`. Although the branch specified in the `Triggers` configuration is `main`, the workflow run will use the workflow definition file and other source files in the *source* branch (which is the branch you're pulling *from*).

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
      - REVISION
```

Example: A trigger with a pull, branches, and a 'CLOSED' event

The following example shows a trigger that starts a workflow run whenever a pull request is closed on a branch that starts with `main`.

In this example:

- When you close a pull request with a destination branch that starts with `main`, a workflow run starts automatically using the workflow definition file and other source files in the (now closed) source branch.
- If you've configured your source repository to delete branches automatically after a pull request is merged, these branches will never have the chance to enter the `CLOSED` state. This means that merged branches will not activate the pull request `CLOSED` trigger. The only way to activate the `CLOSED` trigger in this scenario is to close the pull request without merging it.

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main.*
    Events:
      - CLOSED
```

Example: A trigger with a push, branches, and files

The following example shows a trigger that starts a workflow run whenever a change is made to the `filename.txt` file, or any file in the `src` directory, on the `main` branch.

When this trigger is activated, CodeCatalyst starts a workflow run using the workflow definition file and other source files in the `main` branch.

```
Triggers:
  - Type: PUSH
    Branches:
      - main
    FilesChanged:
      - filename.txt
      - src\/.*
```

Configuring manual-only triggers

You can limit a workflow so that it can only be started manually by your team using the **Run** button in the CodeCatalyst console. To configure this functionality, you must remove the `Triggers` section in the workflow definition file. The `Triggers` section is included by default when you create a workflow, but the section is optional and can be removed.

Use the following instructions to remove the `Triggers` section in the workflow definition file so that the workflow can only be started manually.

Visual

To remove the 'Triggers' section (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. Choose the **Source** box in the workflow diagram.
8. Under **Triggers**, choose the trash can icon to remove the `Triggers` section from the workflow.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To remove the 'Triggers' section (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

5. Choose **Edit**.
6. Choose **YAML**.
7. Find the `Triggers` section and remove it.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Stopping a workflow run

Use the following procedure to stop a workflow run that's in progress. You might want to stop a run if it was started by accident.

When you stop a workflow run, CodeCatalyst waits for in-progress actions to complete before it marks the run as **Stopped** in the CodeCatalyst console. Any actions that didn't have a chance to start will not be started, and will be marked as **Abandoned**.

> **Note**
>
> If a run is queued (that is, it has no in-progress actions), then the run is stopped immediately.

To stop a workflow run

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Under **Workflows**, choose **Runs** and choose the in-progress run from the list.
5. Choose **Stop**.

Gating a workflow run

A *gate* is a workflow component that you can use to prevent a workflow run from proceeding unless certain conditions are met. An example of a gate is the **Approval** gate where users must submit an approval in the CodeCatalyst console before the workflow run is allowed to continue.

You can add gates between sequences of actions in a workflow, or before the first action (which runs immediately after the **Source** downloads). You can also add gates after the last action, if you have a need to do so.

Gate types

Currently, Amazon CodeCatalyst supports one type of gate: the **Approval** gate. For more information, see Requiring approvals on workflow runs.

Can I set up a gate to run in parallel to another action?

No. Gates can only run before or after an action. For more information, see Setting up dependencies between gates and actions.

Can I use a gate to prevent a workflow run from starting?

Yes, with qualifications.

You can prevent a workflow run from *performing tasks*, which is slightly different from preventing it from *starting*.

To prevent a workflow from performing tasks, add a gate before the very first action in a workflow. In this scenario, a workflow run *will start*—meaning it will download your source repository files—but it will be prevented from performing tasks until the gate is unlocked.

> **Note**
>
> Workflows that start and then get blocked by a gate still count against your *Maximum number of concurrent workflow runs per space* quota and other quotas. To ensure that you do not exceed workflow quotas, consider using a workflow trigger to conditionally start a workflow instead of using a gate. Also consider using a pull request approval rule instead of a gate. For more information about quotas, triggers, and pull request approval rules, see Quotas for workflows, Starting a workflow run automatically with triggers, and Managing requirements for merging a pull request with approval rules.

Limitations of gates

Gates have the following limitations:

- Gates cannot be used in conjunction with the compute sharing feature. For more information about this feature, see Sharing compute across actions.
- Gates cannot be used within action groups. For more information about action groups, see Grouping actions into action groups.

Topics

Adding a gate to a workflow

Use the following instructions to add a gate to a workflow and then configure it.

To add and configure a gate

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. On the left, choose **Gates**.
8. In the gate catalog, search for a gate, and then choose the plus sign (**+**) to add the gate to your workflow.
9. Configure the gate. Choose **Visual** to use the visual editor, or **YAML** to use the YAML editor. For detailed instructions, see:
   - Adding the "Approval" gate to a workflow
10. (Optional) Choose **Validate** to make sure the YAML code is valid.
11. Choose **Commit** to commit your changes.

Setting up dependencies between gates and actions

You can set up a gate to run before or after an action, action group, or gate. For example, you might set up an `Approval` gate to run before a `Deploy` action. In this case, the `Deploy` action is said to *depend on* the `Approval` gate.

To set up dependencies between gates and actions, configure the gate or action's **Depends on** property. For instructions, see Setting up dependencies between actions. The referenced instructions refer to workflow *actions* but apply equally to gates.

For an example of how to set up the **Depends on** property with a gate, see Example: Configuring an "Approval" gate.

Specifying the major, minor, or patch version of a gate

By default, when you add a gate to a workflow, CodeCatalyst adds the full version to the workflow definition file using the format:

v*major*.*minor*.*patch*

For example:

```
My-Gate:
  Identifier: aws/approval@v1
```

You can lengthen the version so that the workflow uses a specific major or minor version of the gate. For instructions, see Specifying the major, minor, or patch version of an action. The referenced topic refers to workflow actions but applies equally to gates.

Requiring approvals on workflow runs

You can configure a workflow run to require an approval before it can proceed. To accomplish this, you must add a **Approval** gate to the workflow. An *Approval gate* prevents a workflow from proceeding until a user or set of users submit one or more approvals in the CodeCatalyst console. Once all approvals are given, the gate is 'unlocked' and the workflow run is allowed to resume.

Use an **Approval** gate in your workflow to give your development, operations, and leadership teams a chance to review your changes before they are deployed to a wider audience.

How do I unlock an approval gate?

To unlock an **Approval** gate, *all* of the following conditions must be met:

- **Condition 1**: The required number of approvals must be submitted. The required number of approvals is configurable, and each user is allowed to submit a single approval.
- **Condition 2**: All approvals must be submitted before the gate times out. The gate times out 14 days after it is activated. This period is not configurable.
- **Condition 3**: No one must reject the workflow run. A single rejection will cause the workflow run to fail.
- **Condition 4**: (Only applies if you are using the superseded run mode.) The run must not be superseded by a later run. For more information, see How do workflow approvals work with queued, superseded, and parallel run modes?.

If any of the conditions are not met, CodeCatalyst stops the workflow and sets the run status to **Failed** (in the case of **Conditions 1** to **3**) or **Superseded** (in the case of **Condition 4**).

When to use the "Approval" gate

Typically, you would use an **Approval** gate in a workflow that deploys applications and other resources to a production server or any environment where quality standards must be validated. By placing the gate before the deployment to production, you give reviewers a chance to validate your new software revision before it becomes available to the public.

Who can provide an approval?

Any user who is a member of your project and who has the **Contributor** or **Project administrator** role can provide an approval. Users with the **Space administrator** role who belong to your project's space can also provide an approval.

> **Note**
>
> Users with the **Reviewer** role cannot provide approvals.

How do I notify users that an approval is required?

To notify users that an approval is required, you must:

- Have CodeCatalyst send them a Slack notification. For more information, see Configuring approval notifications.

- Go to the page in the CodeCatalyst console where the **Approve** and **Reject** buttons are, and paste that page's URL into an email or messaging application addressed to the approvers. For more information about how to navigate to this page, see Approving or rejecting a workflow run.

Can I use an "Approval" gate to prevent a workflow run from starting?

Yes, with qualifications. For more information, see Can I use a gate to prevent a workflow run from starting?.

How do workflow approvals work with queued, superseded, and parallel run modes?

When using the queued, superseded, or parallel run mode, the **Approval** gate works in a similar way to actions. We suggest reading the About queued run mode, About superseded run mode, About parallel run mode sections to familiarize yourself with these run modes. Once you have a basic understanding of them, return to this section to find out how these run modes work when the **Approval** gate is present.

When the **Approval** gate is present, runs are processed as follows:

- If you're using the queued run mode, runs will queue up behind the run that is currently waiting for approval at the gate. When that gate becomes unlocked (that is, all approvals have been given), the next run in the queue advances to the gate, and waits for approvals. This process continues with queued runs being processed through the gate one-by-one. Figure 1 illustrates this process.
- If you're using the superseded run mode, the behavior is the same as for the queued run mode, except that instead of having runs pile up in the queue at the gate, newer runs supersede (take over from) earlier runs. There are no queues, and any run that is currently waiting at the gate for an approval will be cancelled and superseded by a newer run. Figure 2 illustrates this process.
- If you're using the parallel run mode, runs start in parallel and no queues form. Each run gets processed by the gate immediately since there are no runs in front of it. Figure 3 illustrates this process.

Figure 1: 'Queued run mode' and an **Approval** gate

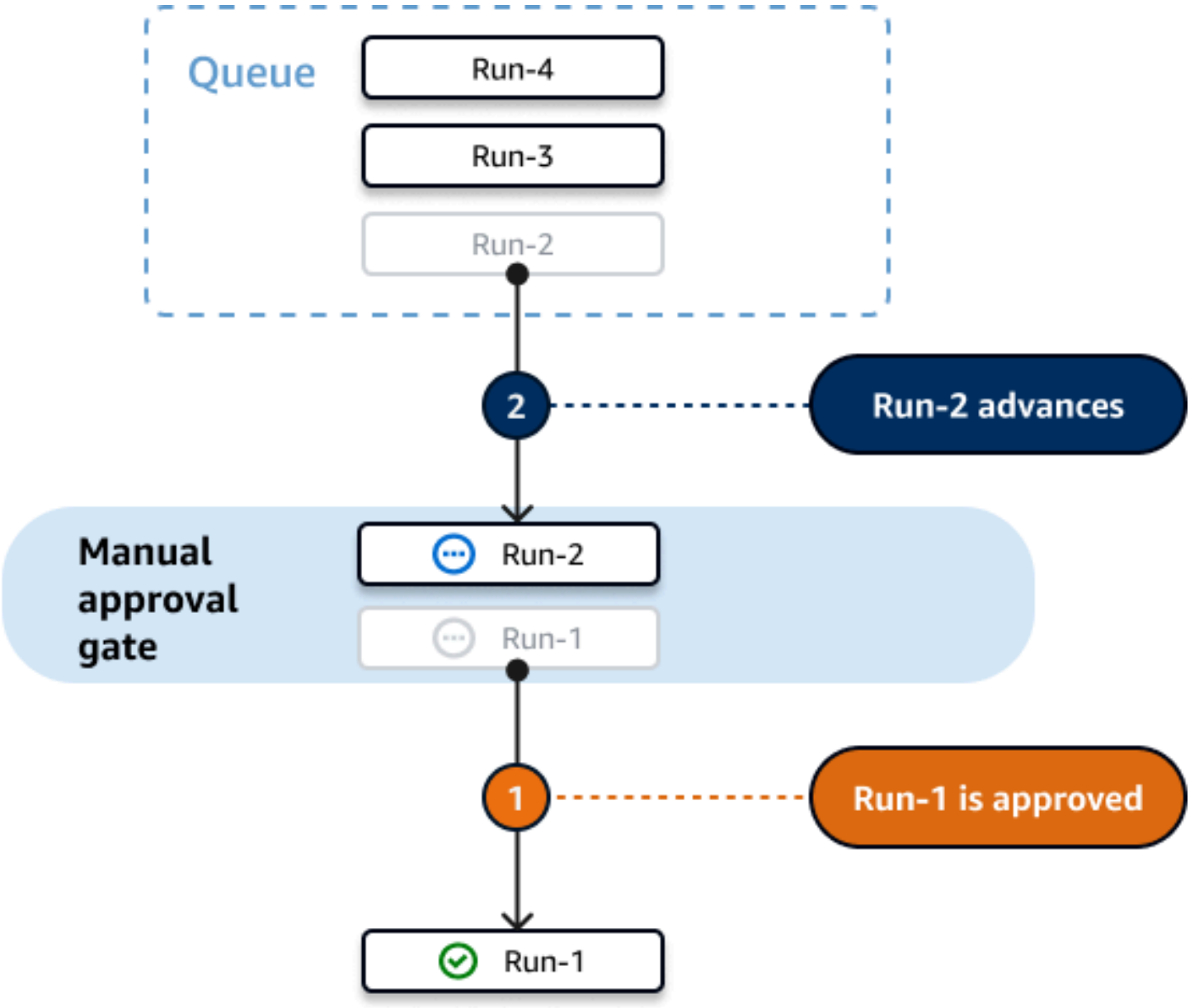


Figure 2: 'Superseded run mode' and an **Approval** gate

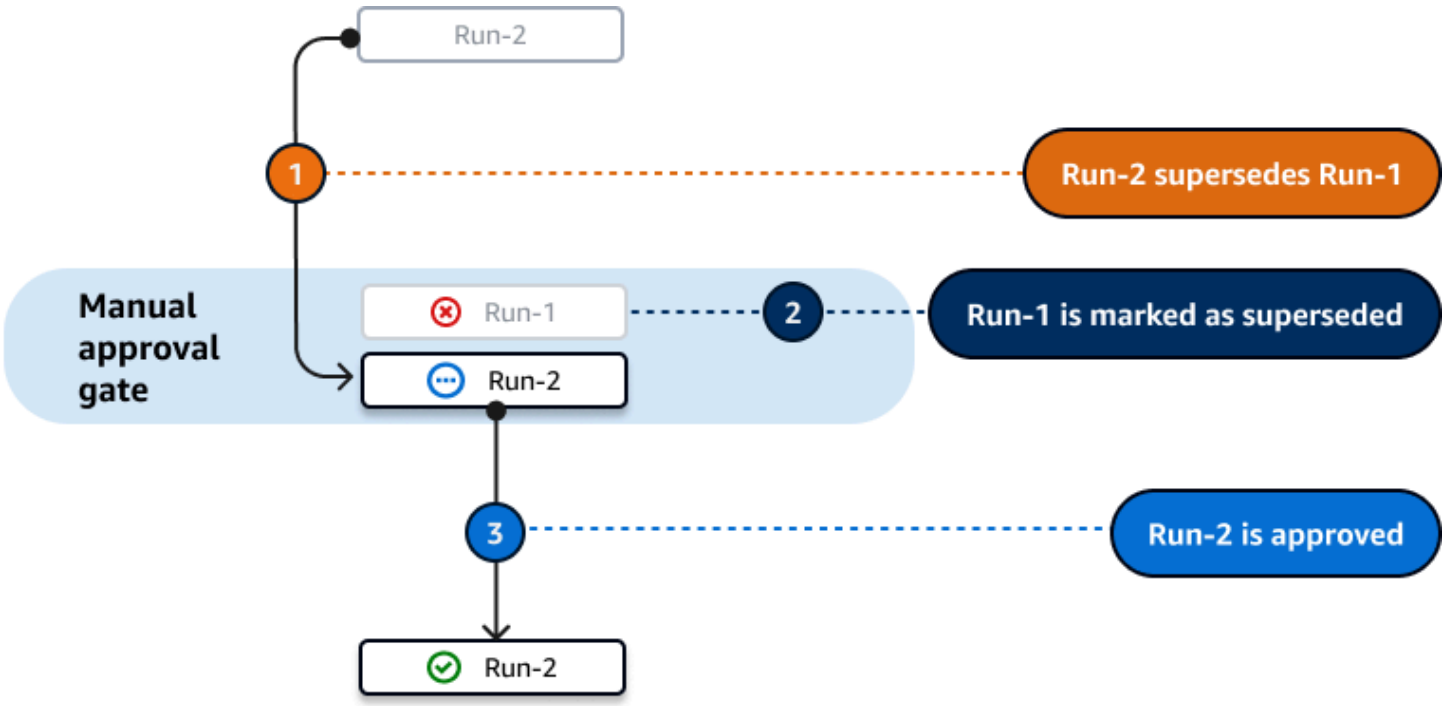
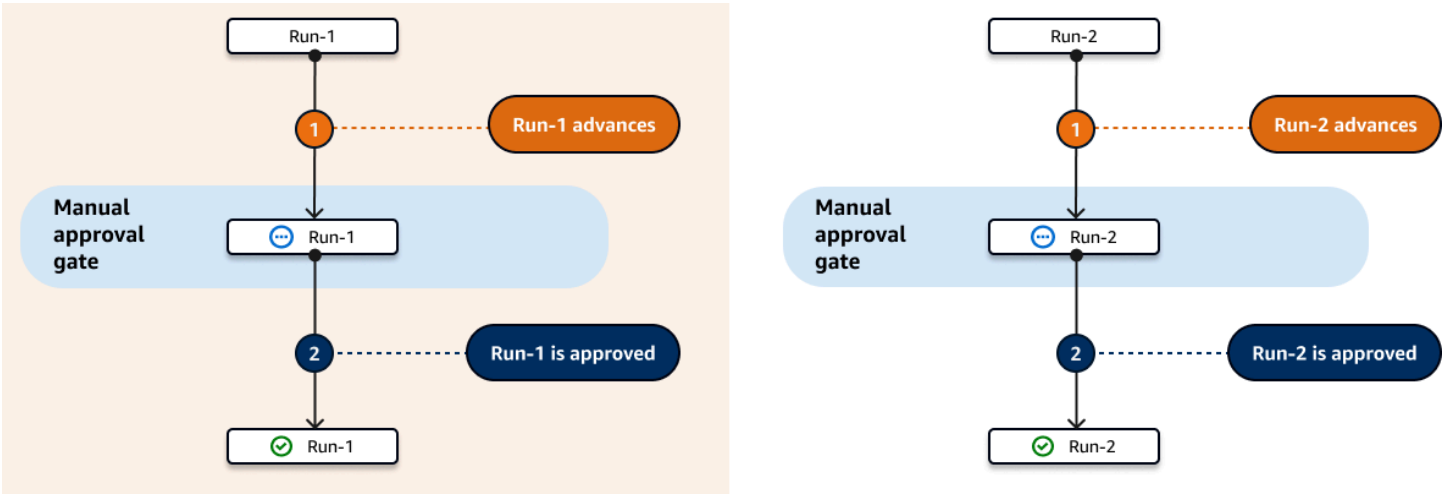


Figure 3: 'Parallel run mode' and an **Approval** gate



Topics

- Example: Configuring an "Approval" gate
- Adding the "Approval" gate to a workflow
- Configuring approval notifications
- Approving or rejecting a workflow run
- "Approval" gate YAML definition

Example: Configuring an "Approval" gate

The following example shows how to add an **Approval** gate called `Approval_01` between two actions called `Staging`, and `Production`. The `Staging` action runs first, the `Approval_01` gate second, and the `Production` action last. The `Production` action only runs if the `Approval_01` gate is unlocked. The `DependsOn` property ensures that the `Staging`, `Approval_01`, and `Production` phases run in sequential order.

For more information about the **Approval** gate, see Requiring approvals on workflow runs.

```
Actions:
  Staging: # Deploy to a staging server
    Identifier: aws/ecs-deploy@v1
    Configuration:
    ...
  Approval_01:
    Identifier: aws/approval@v1
    DependsOn:
      - Staging
    Configuration:
      ApprovalsRequired: 2
  Production: # Deploy to a production server
    Identifier: aws/ecs-deploy@v1
    DependsOn:
      - Approval_01
    Configuration:
    ...
```

Adding the "Approval" gate to a workflow

To configure your workflow to require an approval, you must add the **Approval** gate to the workflow. Use the following instructions to add an **Approval** gate to your workflow.

For more information about this gate, see Requiring approvals on workflow runs.

Visual

To add an "Approval" gate to a workflow (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.

3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. At the top-left, choose **Gates**.
7. In the **Gates** catalog, in **Approval**, choose the plus sign (**+**).
8. Choose **Inputs**, and in the **Depends on** field, do the following.

   Specify an action, action group, or gate that must run successfully in order for this gate to run. By default, when you add a gate to a workflow, the gate is set to depend on the last action in your workflow. If you remove this property, the gate will not depend on anything, and will run first, before other actions.

   > **Note**
   >
   > A gate must be configured to run before or after an action, action group, or gate. It cannot be set up to run in parallel with other actions, action groups, and gates.

   For more information about the **Depends on** functionality, see Setting up dependencies between gates and actions.
9. Choose the **Configuration** tab.
10. In the **Gate name** field, do the following.

    Specify the name you want to give the gate. All gate names must be unique within the workflow. Gate names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in gate names.
11. (Optional) In the **Number of approvals** field, do the following.

    Specify the minimum number of approvals required to unlock the **Approval** gate. The minimum is 1. The maximum is 2. If omitted, the default is 1.

> **Note**
>
> If you want to omit the `ApprovalsRequired` property, remove the gate's `Configuration` section from the workflow definition file.

12. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
13. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add an "Approval" gate to a workflow (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. Add an `Approval` section and underlying properties using the following example as a guide. For more information, see the "Approval" gate YAML definition in the Workflow YAML definition.

```
Actions:
  MyApproval_01:
    Identifier: aws/approval@v1
    DependsOn:
      - PreviousAction
    Configuration:
      ApprovalsRequired: 2
```

   For another example, see Example: Configuring an "Approval" gate.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Configuring approval notifications

You can have CodeCatalyst send a notification to a Slack channel informing users that a workflow run requires an approval. Users see the notification and click the link inside of it. The link takes them to a CodeCatalyst approvals page where they can either approve or reject the workflow.

You can also configure notifications to inform users that a workflow was approved, rejected, or that the approval request has expired.

Use the following instructions to set up Slack notifications.

Before you begin

Make sure you have added an **Approval** gate to your workflow. For more information, see Adding the "Approval" gate to a workflow.

To send workflow approval notifications to a Slack channel

1. Configure CodeCatalyst with Slack. For more information, see Getting started with Slack notifications.
2. In the CodeCatalyst project that contains the workflow that requires an approval, enable notifications, if they're not already enabled. To enable notifications:
    a. Navigate to your project and in the navigation pane, choose **Project settings**.
    b. At the top, choose **Notifications**.
    c. In **Notification events**, choose **Edit notifications**.
    d. Turn on **Workflow approval pending** and choose a Slack channel where CodeCatalyst will send the notification.
    e. (Optional) Turn on additional notifications to alert people about approved, rejected, and expired approvals. You can turn on **Workflow run approved**, **Workflow run rejected**, **Workflow approval superseded**, and **Workflow approval timed out**. Next to each notification, choose the Slack channel where CodeCatalyst will send the notification.
    f. Choose **Save**.

Approving or rejecting a workflow run

Workflow runs that include the **Approval** gate will need to be approved or rejected. Users can provide their approval or rejection starting from:

- the CodeCatalyst console
- a link provided by a team member
- an automated Slack notification

After a user provides their approval or rejection, this decision cannot be undone.

> **Note**
>
> Only certain users can approve or reject a workflow run. For more information, see Who can provide an approval?.

Before you begin

Make sure you have added an **Approval** gate to your workflow. For more information, see Adding the "Approval" gate to a workflow.

To approve or reject a workflow run starting from the CodeCatalyst console

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. In the workflow diagram, choose the box representing the **Approval** gate.

   A side panel appears.

   > **Note**
   >
   > At this point, you can send the URL of this page to other approvers if you want.

6. Under **Review decision**, choose **Approve** or **Reject**.
7. (Optional) In **Comment - optional**, enter a comment indicating why you approved or rejected the workflow run.
8. Choose **Submit**.

To approve or reject a workflow run starting from a link provided by a team member

1. Choose the link sent to you by your team member. (You can have your team member read the preceding procedure to obtain the link.)
2. Sign in to CodeCatalyst, if asked.

   You are redirected to the workflow run approval page.
3. Under **Review decision**, choose **Approve** or **Reject**.
4. (Optional) In **Comment - optional**, enter a comment indicating why you approved or rejected the workflow run.
5. Choose **Submit**.

To approve or reject a workflow run starting from an automated Slack notification

1. Make sure Slack notifications are set up. See Configuring approval notifications.
2. In Slack, in the channel to which the approval notification was sent, choose the link in the approval notification.
3. Sign in to CodeCatalyst, if asked.

   You are redirected to the workflow run page.
4. In the workflow diagram, choose the approval gate.
5. Under **Review decision**, choose **Approve** or **Reject**.
6. (Optional) In **Comment - optional**, enter a comment indicating why you approved or rejected the workflow run.
7. Choose **Submit**.

"Approval" gate YAML definition

The following is the YAML definition of the **Approval** gate. To learn how to use this gate, see Requiring approvals on workflow runs.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The "Approval" gate definition starts here.
  Approval:
    Identifier: aws/approval@v1
    DependsOn:
      - another-action
    Configuration:
      ApprovalsRequired: number
```

Approval

(Required)

Specify the name you want to give the gate. All gate names must be unique within the workflow. Gate names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in gate names.

Default: `Approval_nn`.

Corresponding UI: Configuration tab/**Gate name**

Identifier

(*Approval*/**Identifier**)

(Required)

Identifies the gate. The **Approval** gate supports version `1.0.0`. Do not change this property unless you want to shorten the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/approval@v1`.

Corresponding UI: Workflow diagram/Approval_nn/**aws/approval@v1** label

DependsOn

(*Approval*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this gate to run. By default, when you add a gate to a workflow, the gate is set to depend on the last action in your workflow. If you remove this property, the gate will not depend on anything, and will run first, before other actions.

> **Note**
>
> A gate must be configured to run before or after an action, action group, or gate. It cannot be set up to run in parallel with other actions, action groups, and gates.

For more information about the **Depends on** functionality, see Setting up dependencies between gates and actions.

Corresponding UI: Inputs tab/**Depends on**

Configuration

(*Approval*/**Configuration**)

(Optional)

A section where you can define the configuration properties of the gate.

Corresponding UI: **Configuration** tab

ApprovalsRequired

(*Approval*/Configuration/**ApprovalsRequired**)

(Optional)

Specify the minimum number of approvals required to unlock the **Approval** gate. The minimum is 1. The maximum is 2. If omitted, the default is 1.

> **Note**
>
> If you want to omit the `ApprovalsRequired` property, remove the gate's `Configuration` section from the workflow definition file.

Corresponding UI: Configuration tab/**Number of approvals**

Configuring the queuing behavior of runs

By default, when multiple workflow runs occur at the same time, CodeCatalyst queues them up, and processes them one by one, in the order that they were started. You can change this default behavior by specifying a *run mode*. There are a few run modes:

- (Default) Queued run mode – CodeCatalyst processes runs one by one
- Superseded run mode – CodeCatalyst processes runs one by one, with newer runs overtaking older ones
- Parallel run mode – CodeCatalyst processes runs in parallel

Topics

About queued run mode

In *queued run mode*, runs occur in series, with waiting runs forming a queue.

Queues form at the entry points to actions and action groups, so you can have *multiple queues within the same workflow* (see Figure 1). When a queued run enters an action, the action is locked

and no other runs can enter. When the run finishes and exits the action, the action becomes unlocked and ready for the next run.

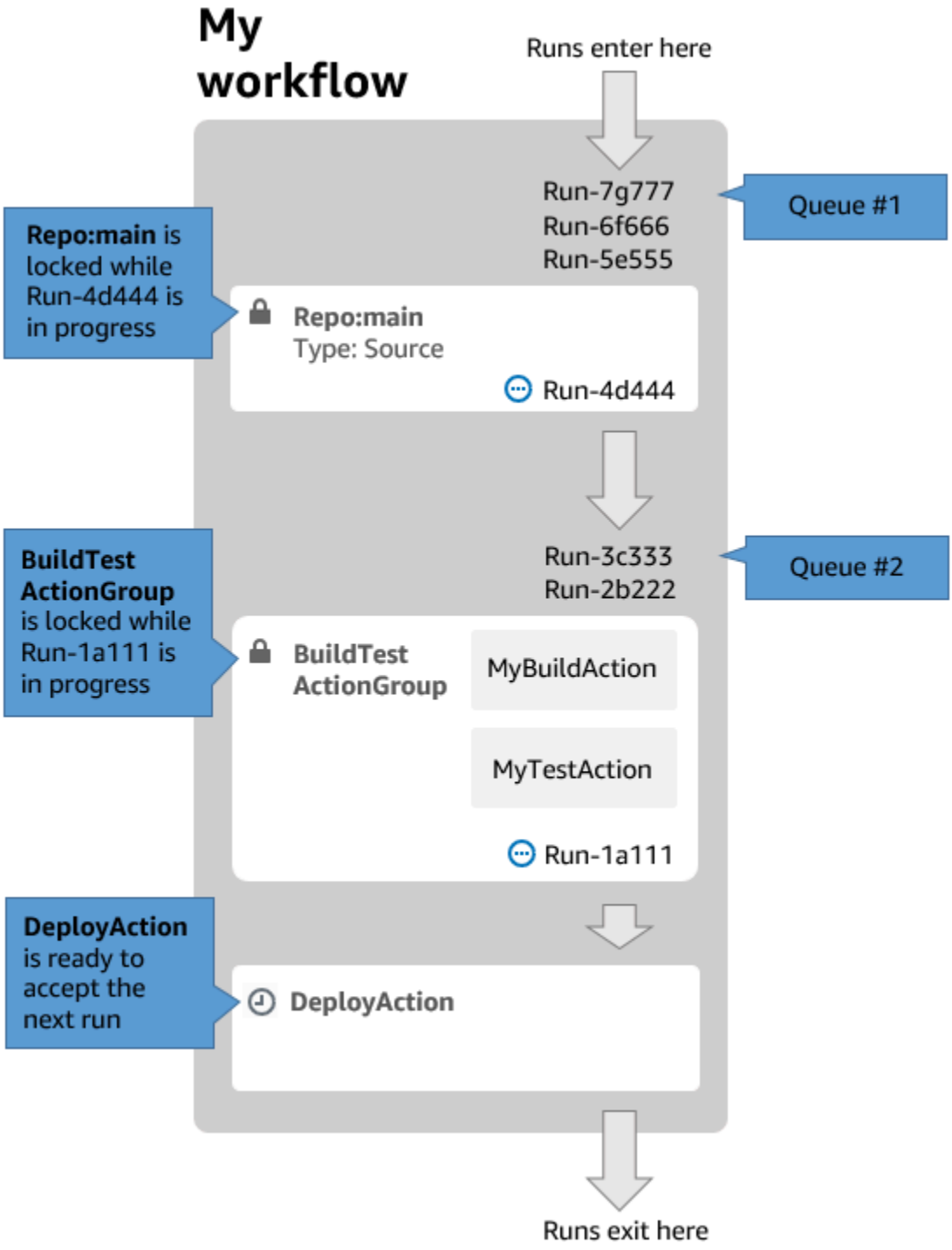
Figure 1 illustrates a workflow configured in queued run mode. It shows:

- Seven runs working their way through the workflow.
- Two queues: one outside the entry to the input source (**Repo:main**), and one outside the entry to the **BuildTestActionGroup** action.
- Two locked blocks: the input source (**Repo:main**) and the **BuildTestActionGroup**.

Here's how things will transpire as the workflow runs finish processing:

- When **Run-4d444** finishes cloning the source repository, it will exit the input source and join the queue behind **Run-3c333**. Then, **Run-5e555** will enter the input source.
- When **Run-1a111** finishes building and testing, it will exit the **BuildTestActionGroup** action and enter the **DeployAction** action. Then, **Run-2b222** will enter the **BuildTestActionGroup** action.

Figure 1: A workflow configured in 'queued run mode'



Use queued run mode if:

- **You want to keep a one-to-one relationship between features and runs – these features may be grouped when using superseded mode**. For example, when you merge feature 1 in commit 1, run 1 starts, and when you merge feature 2 in commit 2, run 2 starts, and so on. If you were

to use superseded mode instead of queued mode, your features (and commits) will be grouped together in the run that supersedes the others.

- **You want to avoid race conditions and unexpected problems that may occur when using parallel mode**. For example, if two software developers, Wang and Saanvi, start workflow runs at roughly the same time to deploy to an Amazon ECS cluster, Wang's run might begin integration tests on the cluster while Saanvi's run deploys new application code to the cluster, causing Wang's tests either to fail or to test the wrong code. As another example, you might have a target that doesn't have a locking mechanism, in which case the two runs could overwrite each other's changes in unexpected ways.
- **You want to limit the load** on the compute resources that CodeCatalyst uses to process your runs. For example, if you have three actions in your workflow, you can have a maximum of three runs occurring at the same time. Imposing a limit on the number of runs that can occur at once makes run throughput more predictable.
- **You want to constrain the number of requests made to third-party services** by the workflow. For example, your workflow might have a build action that includes instructions to pull an image from Docker Hub. Docker Hub limits the number of pull requests you can make to a certain number per hour per account, and you will be locked out if you go over the limit. Using queued run mode to slow down your run throughput will have the effect of generating fewer requests to Docker Hub per hour, thus limiting the potential for lockouts and resulting build and run failures.

Maximum queue size: 50

Notes on **Maximum queue size**:

- The maximum queue size refers to the maximum number of runs allowed across *all queues* in the workflow.
- If a queue becomes longer than 50 runs, then CodeCatalyst drops the 51st and subsequent runs.

Failure behavior:

If a run becomes unresponsive while it's being processed by an action, then the runs behind it are held up in the queue until the action times out. Actions time out after an hour.

If a run fails inside an action, then the first queued run behind it is allowed to proceed.

About superseded run mode

Superseded run mode is the same as *queued run mode* except that:

- If a queued run catches up to another run in the queue, the later run supersedes (takes over from) the earlier run, and the earlier run is canceled and marked as 'superseded'.
- As an outcome of the behavior described in the first bullet, a queue can only include one run when superseded run mode is used.

Using the workflow in Figure 1 as a guide, applying superseded run mode to this workflow would result in the following:

- **Run-7g777** would supersede the other two runs in its queue, and would be the only run remaining in **Queue #1**. **Run-6f666** and **Run-5e555** would be canceled.
- **Run-3c333** would supersede **Run-2b222** and be the only run remaining in **Queue #2**. **Run-2b222** would be canceled.

Use superseded run mode if you want:

- better throughput than with queued mode
- even fewer requests into third-party services than with queued mode; this is advantageous if the third-party service has rate limits, such as Docker Hub

About parallel run mode

In *parallel run mode*, runs are independent of one another and don't wait for other runs to complete before starting. There are no queues, and run throughput is limited only by how fast the actions inside the workflow take to complete.

Use parallel run mode in development environments where each user has their own feature branch and deploys to targets that are not shared by other users.

> **Important**
>
> If you have a shared target that multiple users can deploy to, such as a Lambda function in a production environment, do not use parallel mode, because race conditions may result. A *race condition* occurs when parallel workflow runs attempt to change a shared resource at the same time, leading to unpredictable results.

Maximum number of parallel runs: 1000 per CodeCatalyst space

Configuring the run mode

You can set the run mode to queued, superseded, or parallel. The default is queued.

When you change the run mode from queued or superseded to parallel, CodeCatalyst cancels the runs that are queued, and allows the runs that are currently being processed by an action to finish before canceling them.

When you change the run mode from parallel to queued or superseded, CodeCatalyst lets all currently-running parallel runs complete. Any runs that you start after changing the run mode to queued or superseded use the new mode.

Visual

To change the run mode using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. At the top-right, choose **Workflow properties**.
7. Expand **Advanced**, and under **Run mode**, choose one of the following:
   a. **Queued** – see About queued run mode
   b. **Superseded** – see About superseded run mode
   c. **Parallel** – see About parallel run mode
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To change the run mode using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.

3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. Add the RunMode property, like this:

```
Name: Workflow_6d39
SchemaVersion: "1.0"
RunMode: QUEUED|SUPERSEDED|PARALLEL
```

   For more information, see the description of the RunMode property in the Top-level properties section of the Workflow YAML definition.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Caching files between workflow runs

When file caching is enabled, the build and test actions save on-disk files to a cache and restore them from that cache in subsequent workflow runs. Caching reduces the latency caused by building or downloading dependencies that haven't changed between runs. CodeCatalyst also supports fallback caches, which can be used to restore partial caches containing some of the needed dependencies. This helps reduce the latency impacts of a cache miss.

> **Note**
>
> File caching is only available with the Amazon CodeCatalyst build and test actions, and only when they are configured to use the **EC2** compute type.

Topics

About file caching

File caching allows you to organize your data into multiple caches, which are each referenced under the `FileCaching` property. Each cache saves a directory specified by a given path. The specified directory will be restored in future workflow runs. The following is an example YAML snippet for caching with multiple caches named `cacheKey1` and `cacheKey2`.

```
Actions:
  BuildMyNpmApp:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm run test
    Caching:
      FileCaching:
        cacheKey1:
          Path: file1.txt
          RestoreKeys:
             - restoreKey1
        cacheKey2:
          Path: /root/repository
          RestoreKeys:
             - restoreKey2
             - restoreKey3
```

Note

CodeCatalyst uses multilayered caching, which consists of a local cache and a remote cache. When provisioned fleets or on-demand machines encounter a cache miss on a local cache, dependencies will be restored from a remote cache. As a result, some action runs may experience latency from downloading a remote cache.

CodeCatalyst applies cache access restrictions to ensure that an action in one workflow cannot modify the caches from a different workflow. This protects each workflow from others that might push incorrect data that impact builds or deployments. Restrictions are enforced with cache-scopes

which isolate caches to every workflow and branch pairing. For example, `workflow-A` in branch `feature-A` has a different file cache than `workflow-A` in sibling branch `feature-B`.

Cache misses occur when a workflow looks for a specified file cache and is unable to find it. This can occur for multiple reasons, such as when a new branch is created or when a new cache is referenced and it hasn't been created yet. It can also occur when a cache expires, which by default occurs 14 days after it was last used. To mitigate cache misses and increase the rate of cache hits, CodeCatalyst supports fallback caches. Fallback caches are alternate caches and provide an opportunity to restore partial-caches, which can be an older version of a cache. A cache is restored by first searching for a match under `FileCaching` for the property name, and if not found, evaluates `RestoreKeys`. If there is a cache miss for both the property name and all `RestoreKeys`, the workflow will continue to run, as caching is best effort and not guaranteed.

Creating a cache

You can use the following instructions to add a cache to your workflow.

Visual

To add a cache using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action where you want to add your cache.
8. Choose **Configuration**.
9. Under **File caching - optional**, choose **Add cache** and enter information into the fields, as follows:

   Key

   Specify the name of your primary cache property name. Cache property names must be unique within your workflow. Each action can have up to five entries in `FileCaching`.

Path

Specify the associated path for your cache.

Restore keys - optional

Specify the restore key to use as a fallback when the primary cache property can't be found. Restore key names must be unique within your workflow. Each cache can have up to five entries in `RestoreKeys`.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and then choose **Commit** again.

YAML

To add a cache using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In a workflow action, add code similar to the following:

```
action-name:
  Configuration:
    Steps: ...
  Caching:
    FileCaching:
      key-name:
        Path: file-path
        # # Specify any additional fallback caches
        # RestoreKeys:
        #   - restore-key
```

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.

9. Choose **Commit**, enter a commit message, and choose **Commit** again.

File caching constraints

The following are the constraints for the property name and `RestoreKeys`:

- Names must be unique within a workflow.
- Names are limited to alphanumeric characters (A-Z, a-z, 0-9), hyphens (-), and underscores (_).
- Names can have up to 180 characters.
- Each action can have up to five caches in `FileCaching`.
- Each cache can have up to five entries in `RestoreKeys`.

The following are the constraints for paths:

- Asterisks (*) are not allowed.
- Paths can have up to 255 characters.

Viewing workflow run status and details

You can view the status and details of a single workflow run, or multiple runs at the same time.

For a list of possible run states see Workflow run states.

> **Note**
>
> You can also view the workflow status, which is different from the workflow *run* status. For more information, see Viewing the workflow status.

Topics

- Viewing the status and details of a single run
- Viewing the status and details of all runs in your project
- Viewing the status and details of all runs of a specific workflow
- Viewing runs of a workflow in the workflow diagram

Viewing the status and details of a single run

You might want to view the status and details of a single workflow run to check whether it was successful, to see at what time it was completed, or to view who or what started it.

To view the status and details of a single run

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Under the workflow's name, choose **Runs**.
6. In **Run history**, in the **Run ID** column, choose a run. For example, `Run-95a4d`.
7. Under the run's name, do one of the following:

   - **Visual** to see a workflow diagram showing your workflow run's actions and their status (see Workflow run states). This view also shows the source repository and branch used during the run.

     In the workflow diagram, choose an action to see details such as logs, reports, and outputs generated by the action during the run. The information shown depends on which action type is selected. For more information about viewing build or deploy logs, see Viewing the results of a build action or Viewing the deployment logs.
   - **YAML** to see the workflow definition file that was used for the run.
   - **Artifacts** to see the artifacts produced by the workflow run. For more information about artifacts, see Sharing data between actions in a workflow using artifacts.
   - **Reports** to see the test reports and other types of reports produced by the workflow run. For more information about reports, see Quality report types.
   - **Variables** to see the output variables produced by the workflow run. For more information about variables, see Configuring and using variables in a workflow.

> **Note**
>
> If the run's parent workflow was deleted, a message indicating this fact appears at the top of the run details page.

Viewing the status and details of all runs in your project

You might want to view the status and details of all workflow runs within your project understand how much workflow activity is going on in your project, and learn about the overall health of your workflows.

To view the status and details of all runs in your project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Under **Workflows**, choose **Runs**.

   All the runs, for all workflows, in all branches, across all repositories in your project, are displayed.

   The page includes the following columns:

   - **Run ID** – The unique identifier of the run. Choose the run ID link to view detailed information about the run.
   - **Status** – The processing status of the workflow run. For more information about run states, see Workflow run states.
   - **Trigger** – The person, commit, pull request (PR), or schedule that started the workflow run. For more information, see Starting a workflow run automatically with triggers.
   - **Workflow** – The name of the workflow for which a run was started, and the source repository and branch where the workflow definition file resides. You might need to expand the column width to see this information.

> **Note**
>
> If this column is set to **Not available**, it's usually because the associated workflow was deleted or moved.

- **Start time** – The time when the workflow run started.
- **Duration** – How long the workflow run took to process. Very long or very short durations might indicate problems.
- **End time** – The time when the workflow run ended.

Viewing the status and details of all runs of a specific workflow

You might want to view the status and details of all runs associated with a specific workflow to see if any runs are creating bottlenecks within the workflow, or to see which runs are currently in progress or have completed.

To view the status and details of all runs of a specific workflow

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Under the workflow's name, choose **Runs**.

   The runs associated with the chosen workflow appear.

   The page is divided into two sections:

   - **Active runs** – Displays runs that are in progress. These runs will be in one of the following states: **In progress**.
   - **Run history** – Displays runs that have completed (that is, not in progress).

     For more information about run states, see Workflow run states.

Viewing runs of a workflow in the workflow diagram

You can view the status of all runs of a workflow as they progress together through the workflow. The runs are displayed within the workflow diagram (as opposed to in a list view). This gives you a visual representation of which runs are being processed by which actions, and which runs are waiting in a queue.

To view the status of multiple runs as they progress together through a workflow

> **Note**
>
> This procedure only applies if your workflow is using the queued or superseded run mode. For more information, see Configuring the queuing behavior of runs.

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of the workflow that contains the runs you want to view. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

   > **Note**
   >
   > Make sure you're looking at a workflow page and not a run page.

5. Choose the **Latest state** tab on the upper left.

   A workflow diagram appears.
6. Review the workflow diagram. The diagram shows all the runs that are currently in progress within the workflow, and the latest runs that have finished. More specifically:

   - Runs that appear at the top, before **Sources**, are queued and waiting to start.
   - Runs that appear between actions are queued and waiting to be processed by the next action.
   - Runs that appear within an action are 1. currently being processed by the action, 2. have finished being processed by the action, or 3. were not processed by the action (usually because a previous action failed).

Configuring the actions that a workflow performs

An *action* is the main building block of a workflow, and defines a logical unit of work, or task, to perform during a workflow run. Typically, a workflow includes multiple actions that run sequentially or in parallel depending on how you've configured them.

Topics

Action types

Within an Amazon CodeCatalyst workflow, you can use the following types of actions.

Action types

CodeCatalyst actions

A *CodeCatalyst action* is an action that is authored, maintained, and fully supported by the CodeCatalyst development team.

There are CodeCatalyst actions for building, testing, and deploying applications, as well as for performing miscellaneous tasks, such as invoking an AWS Lambda function.

The following CodeCatalyst actions are available:

- **Build**

  This action builds your artifacts and runs your unit tests in a Docker container. For more information, see Adding the build action.

- **Test**

  This action runs integration and system tests against your application or artifacts. For more information, see Adding the test action.

- **Amazon S3 publish**

  This action copies your application artifacts to an Amazon S3 bucket. For more information, see Publishing files to Amazon S3 with a workflow.

- **AWS CDK bootstrap**

  This action provisions the resources that the AWS CDK needs to deploy your CDK app. For more information, see Bootstrapping an AWS CDK app with a workflow.

- **AWS CDK deploy**

  This action synthesizes and deploys an AWS Cloud Development Kit (AWS CDK) app. For more information, see Deploying an AWS Cloud Development Kit (AWS CDK) app with a workflow.

- **AWS Lambda invoke**

  This action invokes an AWS Lambda function. For more information, see Invoking an AWS Lambda function using a workflow.

- **GitHub Actions**

  This action is a *CodeCatalyst* action that allows you to run GitHub Actions within a CodeCatalyst workflow. For more information, see Invoking an AWS Lambda function using a workflow.

- **Deploy AWS CloudFormation stack**

  This action deploys AWS CloudFormation stacks. For more information, see Deploying an AWS CloudFormation stack with a workflow.

- **Deploy to Amazon ECS**

This action registers an Amazon ECS task definition and deploys it to an Amazon ECS service. For more information, see Deploying an application to Amazon Elastic Container Service (ECS) with a workflow.

- **Deploy to Kubernetes cluster**

  This action deploys an application to a Kubernetes cluster. For more information, see Deploying an application to Amazon Elastic Kubernetes Service with a workflow.

- **Render Amazon ECS task definition**

  This action inserts a container image URI into an Amazon ECS task definition JSON file, creating a new task definition file. For more information, see Modifying an Amazon ECS task definition file using a workflow.

Documentation for CodeCatalyst actions is available in this guide, and in each action's readme.

For information about the available CodeCatalyst actions, and how to add one to a workflow, see Adding an action to a CodeCatalyst workflow.

CodeCatalyst Labs actions

A *CodeCatalyst Labs action* is an action that is part of Amazon CodeCatalyst Labs, a proving ground for experimental applications. CodeCatalyst Labs actions have been developed to showcase integrations with AWS services.

The following CodeCatalyst Labs actions are available:

- **Deploy to AWS Amplify Hosting**

  This action deploys an application to Amplify Hosting.

- **Deploy to AWS App Runner**

  This action deploys the latest image in a source image repository to App Runner.

- **Deploy to Amazon CloudFront and Amazon S3**

  This action deploys an application to CloudFront and Amazon S3.

- **Deploy with AWS SAM**

  This action deploys your serverless application with AWS Serverless Application Model (AWS SAM).

- **Invalidate Amazon CloudFront Cache**

  This action invalidates a CloudFront cache for a given set of paths.

- **Outgoing Webhook**

  This action allows users to send messages within a workflow to an arbitrary web server using an HTTPS request.

- **Publish to AWS CodeArtifact**

  This action publishes packages to a CodeArtifact repository.

- **Publish to Amazon SNS**

  This action allows users to integrate with Amazon SNS by creating a topic, publishing to a topic, or subscribing to a topic.

- **Push to Amazon ECR**

  This action builds and publishes a Docker image to an Amazon Elastic Container Registry (Amazon ECR) repository.

- **Scan with Amazon CodeGuru Security**

  This action creates a zip archive of a configured code path and uses CodeGuru Security to run a code scan.

- **Terraform Community Edition**

  This action runs Terraform Community Edition `plan` and `apply` operations.

Documentation for CodeCatalyst Labs actions is available in each action's readme.

For information about adding a CodeCatalyst Labs action to a workflow and viewing its readme, see Adding an action to a CodeCatalyst workflow.

GitHub Actions

A *GitHub Action* is a lot like a CodeCatalyst action, except that it was developed for use with GitHub workflows. For details about GitHub Actions, see the GitHub Actions documentation.

You can use GitHub Actions alongside native CodeCatalyst actions in a CodeCatalyst workflow.

For your convenience, the CodeCatalyst console provides access to several popular GitHub Actions. You can also use any GitHub Action listed in the GitHub Marketplace (subject to a few limitations).

Documentation for GitHub Actions is available in each action's readme.

For more information, see Integrating GitHub Actions into a workflow.

Third-party actions

A *third-party action* is an action that is authored by a third-party vendor, and made available in the CodeCatalyst console. Examples of third-party actions include the **Mend SCA** and **SonarCloud Scan** actions, authored by Mend and Sonar, respectively.

Documentation for third-party actions is available in each action's readme. Additional documentation might also be provided by the third-party vendor.

For information about adding a third-party action to a workflow and viewing its readme, see Adding an action to a CodeCatalyst workflow.

Adding an action to a CodeCatalyst workflow

Use the following instructions to add an action to a workflow and then configure it.

To add and configure an action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. At the top-left, choose **+ Actions**, The **Actions** catalog appears.
7. In the drop-down list, do one of the following:

   - Choose **Amazon CodeCatalyst** to view CodeCatalyst, CodeCatalyst Labs , or third-party actions.
     - CodeCatalyst actions have a **by AWS** label.
     - CodeCatalyst Labs actions have a **by CodeCatalyst Labs** label.

- Third-party actions have a **by *vendor*** label, where *vendor* is the name of the third-party vendor.

- Choose **GitHub** to view a curated list of GitHub Actions.

8. In the action catalog, search for an action, and then do one of the following:

   - Choose the plus sign (**+**) to add the action to your workflow.
   - Choose the action's name to view its readme.

9. Configure the action. Choose **Visual** to use the visual editor, or **YAML** to use the YAML editor. For detailed instructions, see the following links.

   For instructions on adding CodeCatalyst actions, see:

   - Adding the build action
   - Adding the test action
   - Adding the "Deploy to Amazon ECS" action
   - Adding the "Deploy to Kubernetes cluster" action
   - Adding the "Deploy AWS CloudFormation stack" action
   - Adding the "AWS CDK deploy" action
   - Adding the "AWS CDK bootstrap" action
   - Adding the "Amazon S3 publish" action
   - Adding the "AWS Lambda invoke" action
   - Adding the "Render Amazon ECS task definition" action

   For instructions on adding CodeCatalyst Labs actions, see:

   - The action's readme. You can find the readme by choosing the action's name in the action catalog.

   For instructions on adding GitHub Actions, see:

   - Integrating GitHub Actions into a workflow

   For instructions on adding third-party actions, see:

- The action's readme. You can find the readme by choosing the action's name in the action catalog.

10. (Optional) Choose **Validate** to make sure the YAML code is valid.
11. Choose **Commit** to commit your changes.

Removing an action from a workflow

Use the following instructions to remove an action from a workflow.

Visual

To remove an action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, in the action you want to remove, choose the vertical ellipsis icon, and choose **Remove**.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To remove an action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

5. Choose **Edit**.
6. Choose **YAML**.
7. Find the section of the YAML that contains the action you want to remove.

   Select the section and press the delete key on your keyboard.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Developing a custom action

You can develop a custom action to use in your workflows using the CodeCatalyst Action Development Kit (ADK). You can then publish the action to the CodeCatalyst actions catalog, so that other CodeCatalyst users can view and use it in their workflows.

To develop, test, and publish an action (high-level tasks)

1. Install the required tools and packages necessary to develop an action.
2. Create a CodeCatalyst repository to store your action code.
3. Initialize the action. This lays down the source files required by the action, including an action definition file (`action.yml`) that you can update with your own code.
4. Bootstrap the action code to get the necessary tools and libraries to build, test, and release the action project.
5. Build the action on your local computer, and push the changes to your CodeCatalyst repository.
6. Test the action with unit tests locally, and run the ADK-generated workflow in CodeCatalyst.
7. Publish the action to the CodeCatalyst actions catalog by choosing the **Publish** button in the CodeCatalyst console.

For detailed steps, see the Amazon CodeCatalyst Action Development Kit Developer Guide.

Grouping actions into action groups

An *action group* contains one or more actions. Grouping actions into action groups helps you keep your workflow organized, and also allows you to configure dependencies between different groups.

> **Note**
>
> You cannot nest action groups within other action groups or actions.

Topics

-
- Example: Defining two action groups

Use the following instructions to define an action group.

Visual

Not available. Choose YAML to view the YAML instructions.

YAML

To define a group

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In `Actions`, add code similar to the following:

```
Actions:
  action-group-name:
    Actions:
      action-1:
        Identifier: aws/build@v1
        Configuration:
          ...
      action-2:
        Identifier: aws/build@v1
        Configuration:
```

```
            ...
```

For another example, see Example: Defining two action groups. For more information, see the description of the `action-group-name` property in the Actions of the Workflow YAML definition.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Example: Defining two action groups

The following example shows how to define two action groups: `BuildAndTest` and `Deploy`. The `BuildAndTest` group includes two actions (`Build` and `Test`), and the `Deploy` group also includes two actions (`DeployCloudFormationStack` and `DeployToECS`).

```
Actions:
  BuildAndTest: # Action group 1
    Actions:
      Build:
        Identifier: aws/build@v1
        Configuration:
          ...
      Test:
        Identifier: aws/managed-test@v1
        Configuration:
  Deploy: #Action group 2
    Actions:
      DeployCloudFormationStack:
        Identifier: aws/cfn-deploy@v1
        Configuration:
          ...
      DeployToECS:
        Identifier: aws/ecs-deploy@v1
        Configuration:
          ...
```

Configuring actions to depend on other actions

By default, when you add actions to a workflow, they are added side by side in the visual editor. This means that the actions will run in parallel when you start a workflow run. If you want actions to run in sequential order (and appear vertically in the visual editor), you must set up dependencies

between them. For example, you might set up a `Test` action to depend on the `Build` action so that the test action runs after the build action.

You can set up dependencies between actions and action groups. You can also configure one-to-many dependencies so that one action depends on several others in order to start. Consult the Guidelines for setting up dependencies to ensure your dependency setup conforms with the workflow's YAML syntax.

Topics

Setting up dependencies between actions

Use the following instructions to set up dependencies between actions in a workflow.

Visual

To set up dependencies using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action that will depend on another action.
8. Choose the **Inputs** tab.
9. In **Depends on - optional**, do the following:

   Specify an action, action group, or gate that must run successfully in order for this action to run.

   For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To set up dependencies using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In an action that will depend on another, add code similar to the following:

```
action-name:
  DependsOn:
    - action-1
```

   For more examples, see Examples of how to configure dependencies between actions. For general guidelines, see Guidelines for setting up dependencies. For more information, see the description of the `DependsOn` property in the Workflow YAML definition for your action.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Guidelines for setting up dependencies

When configuring dependencies, follow these guidelines:

- If an action is inside a group, that action can only depend on other actions within the same group.
- Actions and action groups can depend on other actions and action groups *at the same level* in the YAML hierarchy, but *not* at a different level.

Examples of how to configure dependencies between actions

The following examples show how to configure dependencies between actions and groups in the workflow definition file.

Topics

Example: Configuring a simple dependency

The following example shows how to configure a `Test` action to depend on the `Build` action using the `DependsOn` property.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Configuration:
      ...
  Test:
    DependsOn:
      - Build
    Identifier: aws/managed-test@v1
     Configuration:
       ...
```

Example: Configuring an action group to depend on an action

The following example shows how to configure a `DeployGroup` action group to depend on the `FirstAction` action. Notice that action and action group are at the same level.

```
Actions:
  FirstAction: #An action outside an action group
    Identifier: aws/github-actions-runner@v1
    Configuration:
      ...
  DeployGroup: #An action group containing two actions
    DependsOn:
```

```
      - FirstAction
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

Example: Configuring an action group to depend on another action group

The following example shows how to configure a `DeployGroup` action group to depend on the `BuildAndTestGroup` action group. Notice that the action groups are at the same level.

```
Actions:
  BuildAndTestGroup: # Action group 1
    Actions:
      BuildAction:
      ...
      TestAction:
      ...
  DeployGroup: #Action group 2
    DependsOn:
      - BuildAndTestGroup
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

Example: Configuring an action group to depend on multiple actions

The following example shows how to configure a `DeployGroup` action group to depend on the `FirstAction` action, the `SecondAction` action, as well as the `BuildAndTestGroup` action group. Notice that `DeployGroup` is at the same level as `FirstAction`, `SecondAction`, and `BuildAndTestGroup`.

```
Actions:
  FirstAction: #An action outside an action group
    ...
  SecondAction: #Another action
    ...
  BuildAndTestGroup: #Action group 1
    Actions:
```

```
      Build:
      ...
      Test:
      ...
  DeployGroup: #Action group 2
    DependsOn:
      - FirstAction
      - SecondAction
      - BuildAndTestGroup
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

Sharing data between actions in a workflow using artifacts

An *artifact* is the output of a workflow action, and typically consists of a folder or archive of files. Artifacts are important because they allow you to share files and information between actions.

For example, you might have a build action that *generates* a `sam-template.yml` file, but you want a deploy action to *use* it. In this scenario, you would use an artifact to allow the build action to share the `sam-template.yml` file with the deploy action. The code might look something like this:

```
Actions:
  BuildAction:
    Identifier: aws/build@v1
    Steps:
      - Run: sam package --output-template-file sam-template.yml
    Outputs:
      Artifacts:
        - Name: MYARTIFACT
          Files:
            - sam-template.yml
  DeployAction:
    Identifier: aws/cfn-deploy@v1
    Inputs:
      Artifacts:
        - MYARTIFACT
    Configuration:
      template: sam-template.yml
```

In the previous code, the build action (`BuildAction`) generates a `sam-template.yml` file, and then adds it to an output artifact called `MYARTIFACT`. A subsequent deploy action (`DeployAction`) specifies `MYARTIFACT` as an input, giving it access to the `sam-template.yml` file.

Can I share artifacts without specifying them as outputs and inputs?

Yes, you can share artifacts between actions without specifying them in the `Outputs` and `Inputs` sections of your actions' YAML code. To do this, you must turn on compute sharing. For more information about compute sharing and how to specify artifacts when it is turned on, see Sharing compute across actions.

> **Note**
>
> Although the compute sharing feature allows you to simplify your workflow's YAML code by eliminating the need for the `Outputs` and `Inputs` sections, the feature has limitations that you should be aware of before you turn it on. For information about these limitations, see Considerations for compute sharing.

Can I share artifacts between workflows?

No, you cannot share artifacts between different workflows; however, you can share artifacts between actions within the same workflow.

Topics

Defining an output artifact

Use the following instructions to define an artifact that you want an action to output. This artifact then becomes available for other actions to use.

> **Note**
>
> Not all actions support output artifacts. To determine whether your action supports them, run through the visual editor instructions that follow, and see if the action includes an **Output artifacts** button on the **Outputs** tab. If yes, output artifacts are supported.

Visual

To define an output artifact using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action that will produce the artifact.
8. Choose the **Outputs** tab.
9. Under **Artifacts**, choose **Add artifact**.
10. Choose **Add artifact**, and enter information into the fields, as follows.

    Build artifact name

    Specify the name of an artifact generated by the action. Artifact names must be unique within a workflow, and are limited to alphanumeric characters (a-z, A-Z, 0-9) and underscores (_). Spaces, hyphens (-), and other special characters are not allowed. You cannot use quotation marks to enable spaces, hyphens, and other special characters in output artifact names.

    For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

    Files produced by build

Specify the files that CodeCatalyst includes in the artifact that is output by the action. These files are generated by the workflow action when it runs, and are also available in your source repository. File paths can reside in a source repository or an artifact from a previous action, and are relative to the source repository or artifact root. You can use glob patterns to specify paths. Examples:

- To specify a single file that is in the root of your build location or source repository location, use `my-file.jar`.
- To specify a single file in a subdirectory, use `directory/my-file.jar` or `directory/subdirectory/my-file.jar`.
- To specify all files, use `"**/*"`. The `**` glob pattern indicates to match any number of subdirectories.
- To specify all files and directories in a directory named `directory`, use `"directory/**/*"`. The `**` glob pattern indicates to match any number of subdirectories.
- To specify all files in a directory named `directory`, but not any of its subdirectories, use `"directory/*"`.

> **Note**
>
> If your file path includes one or more asterisks (*) or other special character, enclose the path with double quotation marks (""). For more information about special characters, see Syntax guidelines and conventions.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

> **Note**
>
> You may need to add a prefix to the file path to indicate which artifact or source to find it in. For more information, see Referencing files in a source repository and Referencing files in an artifact.

11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To define an output artifact using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In a workflow action, add code similar to the following:

```
action-name:
  Outputs:
    Artifacts:
      - Name: artifact-name
        Files:
          - file-path-1
          - file-path-2
```

   For more examples, see Examples of artifacts. For more information, see the Workflow YAML definition for your action.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Defining an input artifact

If you want to use an artifact generated by another action, you must specify it as an input to the current action. You may be able to specify multiple artifacts as input—it depends on the action. For more information, see the Workflow YAML definition for your action.

> **Note**
>
> You cannot reference artifacts from other workflows.

Use the following instructions to specify an artifact from another action as input to the current action.

Prerequisite

Before you begin, make sure you have output the artifact from the other action. For more information, see Defining an output artifact. Outputting the artifact makes it available for other actions to use.

Visual

To specify an artifact as input to an action (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action where you want to specify an artifact as input.
8. Choose **Inputs**.
9. In **Artifacts - optional**, do the following:

   Specify artifacts from previous actions that you want to provide as input to this action. These artifacts must already be defined as output artifacts in previous actions.

   If you do not specify any input artifacts, then you must specify at least one source repository under `action-name/Inputs/Sources`.

   For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

> **Note**
>
> If the **Artifacts - optional** drop-down list is unavailable (visual editor), or if you get errors in when you validate your YAML (YAML editor), it might be because the action only supports one input. In this case, try removing the source input.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To specify an artifact as input to an action (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In the action where you want to specify the artifact as input, add code similar to the following:

```
action-name:
  Inputs:
    Artifacts:
      - artifact-name
```

   For more examples, see Examples of artifacts.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Referencing files in an artifact

If you have a file that resides within an artifact, and you need to refer to this file in one of your workflow actions, complete the following procedure.

> **Note**
>
> See also Referencing files in a source repository.

To reference files in an artifact

- In the action where you want to reference a file, add code similar to the following:

```
Actions:
  My-action:
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
        - artifact-name
    Configuration:
      Steps:
        - run: cd $CATALYST_SOURCE_DIR_artifact-name/build-output && cat file.txt
```

  In the previous code, the action looks in the `build-output` directory in the *`artifact-name`* artifact to find and display the `file.txt` file.

  For more examples, see Examples of artifacts.

  > **Note**
  >
  > You may be able to omit `$CATALYST_SOURCE_DIR_`*`artifact-name`*`/` prefix depending on how you've configured your action. For more information, see the following guidance.

  Guidance on how to refer to variables:

- If your action includes only one item under `Inputs` (for example, it includes one input artifact and no source), then you can omit the prefix and specify just the file path relative to the artifact root.
- You can also omit the prefix if the file resides in the primary input. The *primary input* is either the `WorkflowSource`, or the first input artifact listed, if there is no `WorkflowSource`.
- The prefix can be different depending on the action you're using. For more information, see the following table.

Action type	File path prefix to use	Example
Build action, test action	`$CATALYST_SOURCE_DIR_` *`artifact-name`* `/`	`$CATALYST_SOURCE_DIR_MyArtifact/folder1/file.txt`

All other actions	`$CATALYST_SOURCE_DIR_` *`artifact-name`* `/`
	
	or
	
	`/artifacts/` *`current-action-name`* `/` *`artifact-name`* `/` (this path is a symbolic link to $CATALYST_SOURCE_DIR_*`artifact-name`* /)
		`$CATALYST_SOURCE_DIR_MyArtifact/folder1/file.txt`
		
		or
		
		`/artifacts/MyCurrentAction/MyArtifact/folder1/file.txt`

Downloading artifacts

You can download and inspect artifacts generated by your workflow actions for troubleshooting purposes. There are two types of artifact you can download:

- **Source artifacts** – An artifact that contains a snapshot of the source repository content as it existed when the run started.

- **Workflow artifacts** – An artifact defined in the `Outputs` property of your workflow's configuration file.

To download artifacts output by the workflow

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. AUnder the workflow's name, choose **Runs**.
6. In **Run history**, in the **Run ID** column, choose a run. For example, `Run-95a4d`.
7. Under the run's name, choose **Artifacts**.
8. Next to an artifact, choose **Download**. An archive file downloads. Its file name consists of seven random characters.
9. Extract the archive using an archive extraction utility of your choice.

Examples of artifacts

The following examples show how to output, input, and reference artifacts in the workflow definition file.

Topics

- Example: Outputting an artifact
- Example: Inputting an artifact generated by another action
- Example: Referencing files in multiple artifacts
- Example: Referencing a file in a single artifact
- Example: Referencing a file in an artifact when a WorkflowSource is present

Example: Outputting an artifact

The following example shows how to output an artifact that includes two .jar files.

```
Actions:
```

```
  Build:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ARTIFACT1
          Files:
            - build-output/file1.jar
            - build-output/file2.jar
```

Example: Inputting an artifact generated by another action

The following example shows you how to output an artifact called `ARTIFACT4` in `BuildActionA`, and input it into `BuildActionB`.

```
Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ARTIFACT4
          Files:
            - build-output/file1.jar
            - build-output/file2.jar
  BuildActionB:
    Identifier: aws/build@v1
    Inputs:
      Artifacts:
        - ARTIFACT4
    Configuration:
```

Example: Referencing files in multiple artifacts

The following example shows you how to output two artifacts named ART5 and ART6 in `BuildActionC`, and then reference two files named `file5.txt` (in artifact ART5) and `file6.txt` (in artifact ART6) in `BuildActionD` (under `Steps`).

> **Note**
>
> For more information on referencing files, see Referencing files in an artifact.

> **Note**
>
> Although the example shows the `$CATALYST_SOURCE_DIR_ART5` prefix being used, you could omit it. This is because `ART5` is the *primary input*. To learn more about the primary input, see Referencing files in an artifact.

```
Actions:
  BuildActionC:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ART5
          Files:
            - build-output/file5.txt
        - Name: ART6
          Files:
            - build-output/file6.txt
  BuildActionD:
    Identifier: aws/build@v1
    Inputs:
      Artifacts:
        - ART5
        - ART6
    Configuration:
      Steps:
        - run: cd $CATALYST_SOURCE_DIR_ART5/build-output && cat file5.txt
        - run: cd $CATALYST_SOURCE_DIR_ART6/build-output && cat file6.txt
```

Example: Referencing a file in a single artifact

The following example shows you how to output one artifact named `ART7` in `BuildActionE`, and then reference `file7.txt` (in artifact `ART7`) in `BuildActionF` (under `Steps`).

Notice how the reference does not require the `$CATALYST_SOURCE_DIR_`*`artifact-name`* prefix in front of the `build-output` directory as it did in Example: Referencing files in multiple artifacts. This is because there is only one item specified under `Inputs`.

> **Note**
>
> For more information on referencing files, see Referencing files in an artifact.

```
Actions:
  BuildActionE:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ART7
          Files:
            - build-output/file7.txt
  BuildActionF:
    Identifier: aws/build@v1
    Inputs:
      Artifacts:
        - ART7
    Configuration:
      Steps:
        - run: cd build-output && cat file7.txt
```

Example: Referencing a file in an artifact when a WorkflowSource is present

The following example shows you how to output one artifact named ART8 in BuildActionG, and then reference file8.txt (in artifact ART8) in BuildActionH (under Steps).

Notice how the reference requires the $CATALYST_SOURCE_DIR_*artifact-name* prefix, as it did in Example: Referencing files in multiple artifacts. This is because there are multiple items specified under Inputs (a source and an artifact), so you need the prefix to indicate where to look for the file.

> **Note**
>
> For more information on referencing files, see Referencing files in an artifact.

```
Actions:
  BuildActionG:
    Identifier: aws/build@v1
```

```
    Outputs:
      Artifacts:
        - Name: ART8
          Files:
            - build-output/file8.txt
  BuildActionH:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
        - ART8
    Configuration:
      Steps:
        - run: cd $CATALYST_SOURCE_DIR_ART8/build-output && cat file8.txt
```

Specifying the major, minor, or patch version of an action

By default, when you add an action to a workflow, Amazon CodeCatalyst adds the full version to the workflow definition file using the format:

v*major*.*minor*.*patch*

For example:

```
My-Build-Action:
  Identifier: aws/build@v1.0.0
```

You can shorten the full version in the `Identifier` property so that the workflow always uses the latest minor or patch version of the action.

For example, if you specify:

```
My-CloudFormation-Action:
  Identifier: aws/cfn-deploy@v1.0
```

...and the latest patch version is `1.0.4`, then the action will use `1.0.4`. If a later version is released, say `1.0.5`, then the action will use `1.0.5`. If a minor version is released, say `1.1.0`, then the action will continue to use `1.0.5`.

For detailed instructions on specifying versions, see one of the following topics.

Use the following instructions to indicate which version of an action you want your workflow to use. You can specify the latest major or minor version, or a specific patch version.

We recommend using the latest minor or patch version of an action.

Visual

Not available. Choose YAML to view the YAML instructions.

YAML

To configure a workflow to use the latest version of an action, or a specific patch version

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. Find the action whose version you want to edit.
8. Find the action's `Identifier` property, and set the version to one of the following:

   - action-identifier@v*`major`* – Use this syntax to have the workflow use a specific major version, and allow the latest minor and patch versions to be chosen automatically.
   - action-identifier@v*`major.minor`* – Use this syntax to have the workflow use a specific minor version, and allow the latest patch version to be chosen automatically.
   - action-identifier@v*`major.minor.patch`* – Use this syntax to have the workflow use a specific patch version.

> **Note**
>
> If you're not sure which versions are available, see Determining which versions of an action are available.

> **Note**
>
> You cannot omit the major version.

9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

Determining which versions of an action are available

Use the following instructions to determine which versions of an action are available for you to use in a workflow.

Visual

To determine which action versions are available

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. Find the action whose versions you want to view:
   a. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
   b. Choose the name of any workflow, or create one. For information about creating a workflow, see Creating a workflow.
   c. Choose **Edit**.
   d. At the top-left, choose **+ Actions** to open the action catalog.
   e. In the drop-down list, choose **Amazon CodeCatalyst** to view CodeCatalyst, CodeCatalyst Labs, and third-party actions, or choose **GitHub** to view curated GitHub Actions.
   f. Search for an action, and choose its name. Do not choose the plus sign (**+**).

      Details about the action appear.
4. In the action details dialog box, near the top-right, choose the **Versions** drop-down list to see a list of available versions of the action.

YAML

Not available. Choose 'visual' to view the visual editor instructions.

Viewing an action's source code

You can view an action's source code to make sure it doesn't contain risky code, security vulnerabilities, or other defects.

Use the following instructions to view the source code of a CodeCatalyst, CodeCatalyst Labs, or third-party action.

> **Note**
>
> To view the source code of a GitHub Action, go to the action's page in the GitHub Marketplace. The page includes a link to the action's repository, where you can find the action's source code.

> **Note**
>
> You cannot view the source code of the following CodeCatalyst actions: build, test, GitHub Actions.

> **Note**
>
> AWS does not support or guarantee the action code of GitHub Actions or third-party actions.

To view an action's source code

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. Find the action whose code you want to view:

    a. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.

b. Choose the name of any workflow, or create one. For information about creating a workflow, see Creating a workflow.

c. Choose **Edit**.

d. At the top-left, choose **+ Actions** to open the action catalog.

e. In the drop-down list, choose **Amazon CodeCatalyst** to view CodeCatalyst, CodeCatalyst Labs, and third-party actions.

f. Search for an action, and choose its name. Do not choose the plus sign (**+**).

Details about the action appear.

4. In the action details dialog box, near the bottom, choose **Download**.

A page appears, showing the Amazon S3 bucket where the action's source code resides. For information about Amazon S3, see What is Amazon S3? in the *Amazon Simple Storage Service User Guide*.

5. Inspect the code to ensure it meets your expectations for quality and security.

Integrating GitHub Actions into a workflow

A *GitHub Action* is a lot like a CodeCatalyst action, except that it was developed for use with GitHub workflows. For details about GitHub Actions, see the GitHub Actions documentation.

You can use GitHub Actions alongside native CodeCatalyst actions in a CodeCatalyst workflow.

There are two ways to add a GitHub Action to a CodeCatalyst workflow:

- You can select the GitHub Action from a curated list in the CodeCatalyst console. Several popular GitHub Actions are available. For more information, see Adding a curated GitHub Action.
- If the GitHub Action that you want to use is not available in the CodeCatalyst console, you can add it using a **GitHub Actions** action.

  A ***GitHub Actions*** action is a *CodeCatalyst action* that wraps a GitHub Action and makes it compatible with CodeCatalyst workflows.

  Here is an example of a **GitHub Actions** action wrapping the Super-Linter GitHub Action:

```
Actions:
  GitHubAction:
    Identifier: aws/github-actions-runner@v1
```

```
    Configuration:
      Steps:
        - name: Lint Code Base
          uses: github/super-linter@v4
          env:
            VALIDATE_ALL_CODEBASE: "true"
            DEFAULT_BRANCH: main
```

In the previous code, the CodeCatalyst **GitHub Actions** action (identified by `aws/github-actions-runner@v1`) wraps the Super-Linter action (identified by `github/super-linter@v4`), making it work in a CodeCatalyst workflow.

For more information, see Adding the "GitHub Actions" action.

All GitHub Actions—both curated and not—must be wrapped inside a **GitHub Actions** action (`aws/github-actions-runner@v1`), as shown in the previous example. The wrapper is required for the action to function properly.

How are GitHub Actions different from CodeCatalyst actions?

GitHub Actions that are used inside a CodeCatalyst workflow do not have the same level of access and integration with AWS and CodeCatalyst features (such as environments and issues) that CodeCatalyst actions do.

Can GitHub Actions interact with other CodeCatalyst actions in the workflow?

Yes. For example, GitHub Actions can use variables produced by other CodeCatalyst actions as input, and can also share output parameters and artifacts with CodeCatalyst actions. For more information, see Exporting a GitHub output parameter so that other actions can use it and Referencing a GitHub output parameter.

Which GitHub Actions can I use?

You can use any GitHub Action available through the CodeCatalyst console, and any GitHub Action available in the GitHub Marketplace. If you decide to use a GitHub Action from the Marketplace, keep in mind the following limitations.

Limitations of GitHub Actions in CodeCatalyst

- GitHub Actions cannot be used with the CodeCatalyst Lambda compute type.

- GitHub Actions run on the November 2022 runtime environment Docker image, which includes older tooling. For more information about the image and tooling, see Specifying runtime environment Docker images.

- GitHub Actions that internally rely on the `github` context or that reference GitHub-specific resources won't work in CodeCatalyst. For example, the following actions won't work in CodeCatalyst:
  - Actions that attempt to add, change, or update GitHub resources. Examples include actions that update pull requests, or create issues in GitHub.
  - Almost all actions listed in https://github.com/actions.
- GitHub Actions that are Docker container actions will work, but they must be run by the default Docker user (root). Do not run the action as user 1001. (At the time of writing, user 1001 works in GitHub, but not in CodeCatalyst.) For more information, see the USER topic in Dockerfile support for GitHub Actions.

For a list of GitHub Actions available through the CodeCatalyst console, see Adding a curated GitHub Action.

How do I add a GitHub Action (high-level steps)?

The high-level steps to add a GitHub Action to a CodeCatalyst workflow are as follows:

1. In your CodeCatalyst project, you **create a workflow**. The workflow is where you define how to build, test, and deploy your application. For more information, see Getting started with workflows.
2. In the workflow, you **add a curated GitHub Action** or you **add the GitHub Actions** action.
3. You do one of the following:
   - If you chose to add a curated action, configure it. For more information, see Adding a curated GitHub Action.
   - If you chose to add a non-curated action, within the **GitHub Actions** action, you **paste the GitHub Action's YAML code**. You can find this code on the details page of your chosen GitHub Action in the GitHub Marketplace. You will likely need to modify the code slightly to have it work in CodeCatalyst. For more information, see Adding the "GitHub Actions" action.
4. (Optional) Within the workflow, **you add other actions** like the build and test actions. For more information, see Build, test, and deploy with workflows in CodeCatalyst.

5. You **start the workflow** either manually or automatically through a trigger. The workflow runs the GitHub Action and any other actions in the workflow. For more information, see Starting a workflow run manually.

For detailed steps, see:

- Adding a curated GitHub Action.
- Adding the "GitHub Actions" action.

Does the GitHub Action run in GitHub?

No. The GitHub Action runs in CodeCatalyst, using CodeCatalyst's build machines.

Can I use GitHub workflows too?

No.

Topics

- Tutorial: Lint code using a GitHub Action in a workflow
- Adding the "GitHub Actions" action
- Adding a curated GitHub Action
- Exporting a GitHub output parameter so that other actions can use it
- Referencing a GitHub output parameter
- "GitHub Actions" action YAML definition

Tutorial: Lint code using a GitHub Action in a workflow

In this tutorial, you add the Super-Linter GitHub Action to an Amazon CodeCatalyst workflow. The Super-Linter action inspects code, finds areas where the code has errors, formatting issues, and suspicious constructs, and then outputs the results to the CodeCatalyst console). After adding the linter to your workflow, you run the workflow to lint a sample Node.js application (`app.js`). You then fix the reported problems and run the workflow again to see if the fixes worked.

> **Tip**
>
> Consider using Super-Linter to lint YAML files, such as AWS CloudFormation templates.

Topics

Prerequisites

Before you begin, you'll need:

- A CodeCatalyst **space** with a connected AWS account. For more information, see Creating a space.
- An empty **project** in your CodeCatalyst space called `codecatalyst-linter-project`. Choose the **Start from scratch** option to create this project.

  For more information, see Creating an empty project in Amazon CodeCatalyst.

Step 1: Create a source repository

In this step, you create a source repository in CodeCatalyst. You'll use this repository to store the sample application source file, `app.js`, for this tutorial.

For more information about source repositories, see Creating a source repository.

To create a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, `codecatalyst-linter-project`.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose **Add repository**, and then choose **Create repository**.
5. In **Repository name**, enter:

   ```
   codecatalyst-linter-source-repository
   ```

6. Choose **Create**.

Step 2: Add an app.js file

In this step, you add an `app.js` file to your source repository. The `app.js` contains function code that has a few mistakes that the linter will find.

To add the app.js file

1. In the CodeCatalyst console, choose your project, `codecatalyst-linter-project`.
2. In the navigation pane, choose **Code**, and then choose **Source repositories**.
3. From the list of source repositories, choose your repository, `codecatalyst-linter-source-repository`.
4. In **Files**, choose **Create file**.
5. In the text box, enter the following code:

```
// const axios = require('axios')
// const url = 'http://checkip.amazonaws.com/';
let response;
/**
 *
 * Event doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format
 * @param {Object} event - API Gateway Lambda Proxy Input Format
 *
 * Context doc: https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-
context.html
 * @param {Object} context
 *
 * Return doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html
 * @returns {Object} object - API Gateway Lambda Proxy Output Format
 *
 */
exports.lambdaHandler = async (event, context) => {
  try {
    // const ret = await axios(url);
    response = {
      statusCode: 200,
      'body': JSON.stringify({
        message: 'hello world'
```

```
        // location: ret.data.trim()
      })
    }
  } catch (err) {
    console.log(err)
    return err
  }

  return response
}
```

6. For **File name**, enter `app.js`. Keep the other default options.
7. Choose **Commit**.

   You have now created a file called `app.js`.

Step 3: Create a workflow that runs the Super-Linter action

In this step, you create a workflow that runs the Super-Linter action when you push code to your source repository. The workflow consists of the following building blocks, which you define in a YAML file:

- **A trigger** – This trigger starts the workflow run automatically when you push a change to your source repository. For more information about triggers, see Starting a workflow run automatically with triggers.
- **A "GitHub Actions" action** – On trigger, the **GitHub Actions** action runs the Super-Linter action, which in turn inspects all files in your source repository. If the linter finds an issue, the workflow action fails.

To create a workflow that runs the Super-Linter action

1. In the CodeCatalyst console, choose your project, `codecatalyst-linter-project`.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose **Create workflow**.
4. For **Source repository**, choose `codecatalyst-linter-source-repository`.
5. For **Branch**, choose `main`.
6. Choose **Create**.

7. Delete the YAML sample code.
8. Add the following YAML:

```
Name: codecatalyst-linter-workflow
SchemaVersion: "1.0"
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  SuperLinterAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        github-action-code
```

   In the preceding code, replace *github-action-code* with the Super-Linter action code, as instructed in the following steps of this procedure.
9. Go to the Super-Linter page in the GitHub Marketplace.
10. Under `steps:` (lowercase), find the code and paste it into the CodeCatalyst workflow under `Steps:` (uppercase).

    Adjust the GitHub Action code to conform to CodeCatalyst standards, as shown in the following code.

    Your CodeCatalyst workflow now looks like this:

```
Name: codecatalyst-linter-workflow
SchemaVersion: "1.0"
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  SuperLinterAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Lint Code Base
          uses: github/super-linter@v4
          env:
```

```
            VALIDATE_ALL_CODEBASE: "true"
            DEFAULT_BRANCH: main
```

11. (Optional) Choose **Validate** to make sure the YAML code is valid before committing.
12. Choose **Commit**, enter a **Commit message**, select your `codecatalyst-linter-source-repository` **Repository**, and choose **Commit** again.

    You have now created a workflow. A workflow run starts automatically because of the trigger defined at the top of the workflow.

To view the workflow run in progress

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the workflow you just created: `codecatalyst-linter-workflow`.
3. In the workflow diagram, choose **SuperLinterAction**.
4. Wait for the action to fail. This failure is expected because the linter found problems in the code.
5. Leave the CodeCatalyst console open and go to Step 4: Fix problems that the Super-Linter found.

Step 4: Fix problems that the Super-Linter found

The Super-Linter should have found problems in the `app.js` code, as well as the `README.md` file included in your source repository.

To fix the problems the linter found

1. In the CodeCatalyst console, choose the **Logs** tab, and then choose **Lint Code Base**.

   The logs that the Super-Linter action generated are displayed.
2. In the Super-Linter logs, scroll down to around line 90, where you find the start of the problems. They look similar to the following:

```
/github/workspace/hello-world/app.js:3:13: Extra semicolon.
/github/workspace/hello-world/app.js:9:92: Trailing spaces not allowed.
/github/workspace/hello-world/app.js:21:7: Unnecessarily quoted property 'body'
 found.
```

```
/github/workspace/hello-world/app.js:31:1: Expected indentation of 2 spaces but
 found 4.
/github/workspace/hello-world/app.js:32:2: Newline required at end of file but not
 found.
```

3. Fix `app.js` and `README.md` in your source repository and commit your changes.

   Tip

   To fix the `README.md`, add `markdown` to the code block, like this:

   ```
   ```markdown
   Setup examples:
   ...
   ```
   ```

   Your changes start another workflow run automatically. Wait for the workflow to finish. If you fixed all the problems, the workflow should succeed.

Clean up

Clean up in CodeCatalyst to remove traces of this tutorial from your environment.

To clean up in CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Delete `codecatalyst-linter-source-repository`.
3. Delete `codecatalyst-linter-workflow`.

In this tutorial, you learned how to add the Super-Linter GitHub Action to a CodeCatalyst workflow in order to lint some code.

Adding the "GitHub Actions" action

A ***GitHub Actions*** action is a *CodeCatalyst action* that wraps a GitHub Action and makes it compatible with CodeCatalyst workflows.

For more information, see Integrating GitHub Actions into a workflow.

To add the **GitHub Actions** action to a workflow, follow these steps.

> **Tip**
>
> For a tutorial that shows you how to use the **GitHub Actions** action, see Tutorial: Lint code using a GitHub Action in a workflow.

Visual

To add the "GitHub Actions" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **GitHub**.
9. Search for the **GitHub Actions** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **GitHub Actions**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **View source** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the "GitHub Actions" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.

12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add the "GitHub Actions" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **GitHub**.
9. Search for the **GitHub Actions** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **GitHub Actions**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **View source** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "GitHub Actions" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

"GitHub Actions" action definition

The **GitHub Actions** action is defined as a set of YAML properties inside your workflow definition file. For information about these properties, see "GitHub Actions" action YAML definition in the Workflow YAML definition.

Adding a curated GitHub Action

A *curated GitHub Action* is a GitHub Action that is made available in the CodeCatalyst console, and serves as an example of how to use a GitHub Action inside a CodeCatalyst workflow.

Curated GitHub Actions are wrapped in the CodeCatalyst-authored **GitHub Actions** action, identified by the `aws/github-actions-runner@v1` identifier. For example, here's what the curated GitHub Action, TruffleHog OSS, looks like:

```
Actions:
  TruffleHogOSS_e8:
    Identifier: aws/github-actions-runner@v1
    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this Workflow as a
 source
    Configuration:
      Steps:
        - uses: trufflesecurity/trufflehog@v3.16.0
          with:
            path: ' ' # Required; description: Repository path
            base: ' ' # Required; description: Start scanning from here (usually main
 branch).
            head: ' ' # Optional; description: Scan commits until here (usually dev
 branch).
            extra_args: ' ' # Optional; description: Extra args to be passed to the
 trufflehog cli.
```

In the previous code, the CodeCatalyst **GitHub Actions** action (identified by `aws/github-actions-runner@v1`) wraps the TruffleHog OSS action (identified by `trufflesecurity/trufflehog@v3.16.0`), making it work in a CodeCatalyst workflow.

To configure this action, you would replace the empty strings under `with:` with your own values. For example:

```
Actions:
  TruffleHogOSS_e8:
    Identifier: aws/github-actions-runner@v1
    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this Workflow as a
 source
```

```
    Configuration:
      Steps:
        - uses: trufflesecurity/trufflehog@v3.16.0
          with:
            path: ./
            base: main # Required; description: Start scanning from here (usually main
branch).
            head: HEAD # Optional; description: Scan commits until here (usually dev
branch).
            extra_args: '--debug --only-verified' # Optional; description: Extra args
to be passed to the trufflehog cli.
```

To add a curated GitHub Action to a workflow, use the following procedure. For general information about using GitHub Actions in a CodeCatalyst workflow, see Integrating GitHub Actions into a workflow.

> **Note**
>
> If you don't see your GitHub Action among the list of curated actions, you can still add it to your workflow using the **GitHub Actions** action. For more information, see Adding the "GitHub Actions" action.

Visual

To add a curated GitHub action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **GitHub**.
9. Browse or search for a GitHub Action, and do one of the following:

- Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

  Or

- Choose the name of the GitHub Action. The action details dialog box appears. On this dialog box:
  - (Optional) Choose **View source** to view the action's source code.
  - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.

10. In the **Inputs**, **Configuration**, and **Outputs** tabs, complete the fields according to your needs. For a description of each field, see the "GitHub Actions" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) available to the **GitHub Actions** action, as it appears in both the YAML and visual editors.

    For information about the configuration options available to the curated GitHub Action, see its documentation.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add a curated GitHub action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **GitHub**.
9. Browse or search for a GitHub Action, and do one of the following:

- Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

  Or

- Choose the name of the GitHub Action. The action details dialog box appears. On this dialog box:
  - (Optional) Choose **View source** to view the action's source code.
  - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.

10. Modify the properties in the YAML code according to your needs. An explanation of each property available to the **GitHub Actions** action is provided in the "GitHub Actions" action YAML definition.

    For information about the configuration options available to the curated GitHub Action, see its documentation.

11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

Exporting a GitHub output parameter so that other actions can use it

You can use GitHub output parameters in your CodeCatalyst workflows.

> **Note**
>
> Another word for *output parameter* is *variable*. Because GitHub uses the term *output parameter* in its documentation, we'll use this term too.

Use the following instructions to export a GitHub output parameter from a GitHub Action so that it is available for use by other CodeCatalyst workflow actions.

To export a GitHub output parameter

1. Open a workflow and choose **Edit**. For more information, see Creating a workflow.
2. In the **GitHub Actions** action that generates the output parameter that you want to export, add an `Outputs` section with an underlying `Variables` property that looks like this:

```
Actions:
  MyGitHubAction:
    Identifier: aws/github-actions-runner@v1
    Outputs:
      Variables:
        - 'step-id_output-name'
```

Replace:

- *step-id* with value of the `id:` property in the GitHub action's `steps` section.
- *output-name* with the name of the GitHub output parameter.

Example

The following example shows you how to export a GitHub output parameter called `SELECTEDCOLOR`.

```
Actions:
  MyGitHubAction:
    Identifier: aws/github-actions-runner@v1
    Outputs:
      Variables:
        - 'random-color-generator_SELECTEDCOLOR'
    Configuration:
      Steps:
        - name: Set selected color
          run: echo "SELECTEDCOLOR=green" >> $GITHUB_OUTPUT
          id: random-color-generator
```

Referencing a GitHub output parameter

Use the following instructions to reference a GitHub output parameter.

To reference a GitHub output parameter

1. Complete the steps in Exporting a GitHub output parameter so that other actions can use it.

   The GitHub output parameter is now available for use in other actions.
2. Note the output parameter's `Variables` value. It includes an underscore (_).

3. Refer to the output parameter using the following syntax:

```
${action-name.output-name}
```

Replace:

- *action-name* with the name of the CodeCatalyst **GitHub Action** that produces the output parameter (do not use the GitHub action's `name` or `id`).
- *output-name* with the output parameter's `Variables` value you noted earlier.

Example

```
BuildActionB:
  Identifier: aws/build@v1
  Configuration:
    Steps:
      - Run: echo ${MyGitHubAction.random-color-generator_SELECTEDCOLOR}
```

Example with context

The following example shows you how to set a `SELECTEDCOLOR` variable in `GitHubActionA`, output it, and then refer to it in `BuildActionB`.

```
Actions:
  GitHubActionA:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Set selected color
          run: echo "SELECTEDCOLOR=green" >> $GITHUB_OUTPUT
          id: random-color-generator
    Outputs:
      Variables:
      - 'random-color-generator_SELECTEDCOLOR'

   BuildActionB:
    Identifier: aws/build@v1
    Configuration:
      Steps:
```

```
      - Run: echo ${GitHubActionA.random-color-generator_SELECTEDCOLOR}
```

"GitHub Actions" action YAML definition

The following is the YAML definition of the **GitHub Actions** action.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

Choose a YAML property in the following code to see a description if it.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  action-name:
    Identifier:  aws/github-actions-runner@v1
    DependsOn:
      - dependent-action-name-1
    Compute:
      Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Inputs:
      Sources:
        - source-name-1
```

```
        - source-name-2
      Artifacts:
        - artifact-name
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Outputs:
      Artifacts:
        - Name: output-artifact-1
          Files:
            - github-output/artifact-1.jar
            - "github-output/build*"
        - Name: output-artifact-2
          Files:
            - github-output/artifact-2.1.jar
            - github-output/artifact-2.2.jar
      Variables:
        - variable-name-1
        - variable-name-2
      AutoDiscoverReports:
        Enabled: true | false
        ReportNamePrefix: AutoDiscovered
        IncludePaths:
          - "**/*"
        ExcludePaths:
          - node_modules/cdk/junit.xml
        SuccessCriteria:
          PassRate: percent
          LineCoverage: percent
          BranchCoverage: percent
          Vulnerabilities:
            Severity: CRITICAL|HIGH|MEDIUM|LOW|INFORMATIONAL
            Number: whole-number
      Reports:
        report-name-1:
          Format: format
          IncludePaths:
            - "*.xml"
          ExcludePaths:
            - report2.xml
            - report3.xml
          SuccessCriteria:
```

```
            PassRate: percent
            LineCoverage: percent
            BranchCoverage: percent
            Vulnerabilities:
              Severity: CRITICAL|HIGH|MEDIUM|LOW|INFORMATIONAL
              Number: whole-number
    Configuration
      Steps:
        - github-actions-code
```

action-name

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Corresponding UI: Configuration tab/*action-name*

Identifier

(*action-name*/**Identifier**)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Use `aws/github-actions-runner@v1` for **GitHub Actions** actions.

Corresponding UI: Workflow diagram/*action-name*/**aws/github-actions-runner@v1** label

DependsOn

(*action-name*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*action-name*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Fleet

(*action-name*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/**Compute fleet - optional**

Timeout

(*action-name*/**Timeout**)

(Optional)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Environment

(*action-name*/**Environment**)

(Optional)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*action-name*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*action-name*/Environment/**Connections**)

(Optional)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**

Name

(*action-name*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**

Role

(*action-name*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that this action uses in order to access and operate in AWS services such as Amazon S3 and Amazon ECR. Make sure this role is added to your AWS account connection in your space. To add an IAM role to an account connection, see Adding IAM roles to account connections.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-spaceName` role with this action. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-spaceName` role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

> **Warning**
>
> Limit the permissions to those required by the **GitHub Action** action. Using a role with broader permissions might pose a security risk.

Corresponding UI: Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**

Inputs

(*action-name*/**Inputs**)

(Optional)

The `Inputs` section defines the data that an action needs during a workflow run.

> **Note**
>
> A maximum of four inputs (one source and three artifacts) are allowed per **GitHub Actions** action. Variables do not count towards this total.

If you need to refer to files residing in different inputs (say a source and an artifact), the source input is the primary input, and the artifact is the secondary input. References to files in secondary inputs take a special prefix to distiguish them from the primary. For details, see Example: Referencing files in multiple artifacts.

Corresponding UI: **Inputs** tab

Sources

(*action-name*/Inputs/**Sources**)

(Optional)

Specify the labels that represent the source repositories that will be needed by the action. Currently, the only supported label is `WorkflowSource`, which represents the source repository where your workflow definition file is stored.

If you omit a source, then you must specify at least one input artifact under *action-name*/`Inputs/Artifacts`.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*action-name*/Inputs/**Artifacts**)

(Optional)

Specify artifacts from previous actions that you want to provide as input to this action. These artifacts must already be defined as output artifacts in previous actions.

If you do not specify any input artifacts, then you must specify at least one source repository under *action-name*/`Inputs/Sources`.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

> **Note**
>
> If the **Artifacts - optional** drop-down list is unavailable (visual editor), or if you get errors in when you validate your YAML (YAML editor), it might be because the action only supports one input. In this case, try removing the source input.

Corresponding UI: Inputs tab/**Artifacts - optional**

Variables - input

(*action-name*/Inputs/**Variables**)

(Optional)

Specify a sequence of name/value pairs that define the input variables that you want to make available to the action. Variable names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in variable names.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Inputs tab/**Variables - optional**

Outputs

(*action-name*/**Outputs**)

(Optional)

Defines the data that is output by the action during a workflow run.

Corresponding UI: **Outputs** tab

Artifacts - output

(*action-name*/Outputs/**Artifacts**)

(Optional)

Specify the name of an artifact generated by the action. Artifact names must be unique within a workflow, and are limited to alphanumeric characters (a-z, A-Z, 0-9) and underscores (_). Spaces, hyphens (-), and other special characters are not allowed. You cannot use quotation marks to enable spaces, hyphens, and other special characters in output artifact names.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/**Artifacts**

Name

(*action-name*/Outputs/Artifacts/**Name**)

(Required if Artifacts - output is included)

Specify the name of an artifact generated by the action. Artifact names must be unique within a workflow, and are limited to alphanumeric characters (a-z, A-Z, 0-9) and underscores (_). Spaces, hyphens (-), and other special characters are not allowed. You cannot use quotation marks to enable spaces, hyphens, and other special characters in output artifact names.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/Artifacts/Add artifact/**Build artifact name**

Files

(*action-name*/Outputs/Artifacts/**Files**)

(Required if Artifacts - output is included)

Specify the files that CodeCatalyst includes in the artifact that is output by the action. These files are generated by the workflow action when it runs, and are also available in your source repository. File paths can reside in a source repository or an artifact from a previous action, and are relative to the source repository or artifact root. You can use glob patterns to specify paths. Examples:

- To specify a single file that is in the root of your build location or source repository location, use `my-file.jar`.
- To specify a single file in a subdirectory, use `directory/my-file.jar` or `directory/subdirectory/my-file.jar`.
- To specify all files, use `"**/*"`. The `**` glob pattern indicates to match any number of subdirectories.
- To specify all files and directories in a directory named `directory`, use `"directory/**/*"`. The `**` glob pattern indicates to match any number of subdirectories.
- To specify all files in a directory named `directory`, but not any of its subdirectories, use `"directory/*"`.

> **Note**
>
> If your file path includes one or more asterisks (*) or other special character, enclose the path with double quotation marks (""). For more information about special characters, see Syntax guidelines and conventions.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

> **Note**
>
> You may need to add a prefix to the file path to indicate which artifact or source to find it in. For more information, see Referencing files in a source repository and Referencing files in an artifact.

Corresponding UI: Outputs tab/Artifacts/Add artifact/**Files produced by build**

Variables - output

(*action-name*/Outputs/**Variables**)

(Optional)

Specify the variables that you want the action to export so that they are available for use by subsequent actions.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Outputs tab/Variables/**Add variable**

variable-name-1

(*action-name*/Outputs/Variables**variable-name-1**)

(Optional)

Specify the name of a variable that you want the action to export. This variable must already be defined in the `Inputs` or `Steps` section of the same action.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Outputs tab/Variables/Add variable/**Name**

AutoDiscoverReports

(*action-name*/Outputs/**AutoDiscoverReports**)

(Optional)

Defines the configuration for the auto-discovery feature.

When you enable auto-discovery, CodeCatalyst searches all `Inputs` passed into the action as well as all files generated by the action itself, looking for test, code coverage, and software composition analysis (SCA) reports. For each report that is found, CodeCatalyst transforms it into a CodeCatalyst report. A *CodeCatalyst report* is a report that is fully integrated into the CodeCatalyst service and can be viewed and manipulated through the CodeCatalyst console.

> **Note**
>
> By default, the auto-discover feature inspects all files. You can limit which files are inspected using the IncludePaths or ExcludePaths properties.

Corresponding UI: *none*

Enabled

(*`action-name`*/Outputs/AutoDiscoverReports/**Enabled**)

(Optional)

Enable or disable the auto-discovery feature.

Valid values are `true` or `false`.

If `Enabled` is omitted, the default is `true`.

Corresponding UI: Outputs tab/Reports/**Automatically discover reports**

ReportNamePrefix

(*`action-name`*/Outputs/AutoDiscoverReports/**ReportNamePrefix**)

(Required if AutoDiscoverReports is included and enabled)

Specify a prefix that CodeCatalyst prepends to all the reports it finds in order to name their associated CodeCatalyst reports. For example, if you specify a prefix of `AutoDiscovered`, and CodeCatalyst auto-discovers two test reports, `TestSuiteOne.xml` and `TestSuiteTwo.xml`,

then the associated CodeCatalyst reports will be named `AutoDiscoveredTestSuiteOne` and `AutoDiscoveredTestSuiteTwo`.

Corresponding UI: Outputs tab/Reports/Automatically discover reports/**Report prefix**

IncludePaths

(*`action-name`*/Outputs/AutoDiscoverReports/**IncludePaths**)

Or

(*`action-name`*/Outputs/Reports/*`report-name-1`*/**IncludePaths**)

(Required if AutoDiscoverReports is included and enabled, or if Reports is included)

Specify the files and file paths that CodeCatalyst includes when searching for raw reports. For example, if you specify `"/test/report/*"`, CodeCatalyst searches the entire build image used by the action looking for the `/test/report/*` directory. When it finds that directory, CodeCatalyst then looks for reports in that directory.

> **Note**
>
> If your file path includes one or more asterisks (*) or other special characters, enclose the path with double quotation marks (""). For more information about special characters, see Syntax guidelines and conventions.

If this property is omitted, the default is `"**/*"`, meaning the search includes all files at all paths.

> **Note**
>
> For manually configured reports, `IncludePaths` must be a glob pattern that matches a single file.

Corresponding UI:

- Outputs tab/Reports/Automatically discover reports/'Include/exclude paths'/**Include paths**
- Outputs tab/Reports/Manually configure reports/*`report-name-1`*/'Include/exclude paths'/**Include paths**

ExcludePaths

(*action-name*/Outputs/AutoDiscoverReports/**ExcludePaths**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/**ExcludePaths**)

(Optional)

Specify the files and file paths that CodeCatalyst excludes when searching for raw reports. For example, if you specify `"/test/my-reports/**/*"`, CodeCatalyst will not search for files in the `/test/my-reports/` directory. To ignore all files in a directory, use the `**/*` glob pattern.

> **Note**
>
> If your file path includes one or more asterisks (*) or other special characters, enclose the path with double quotation marks (""). For more information about special characters, see Syntax guidelines and conventions.

Corresponding UI:

- Outputs tab/Reports/Automatically discover reports/'Include/exclude paths'/**Exclude paths**
- Outputs tab/Reports/Manually configure reports/*report-name-1*/'Include/exclude paths'/**Exclude paths**

SuccessCriteria

(*action-name*/Outputs/AutoDiscoverReports/**SuccessCriteria**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/**SuccessCriteria**)

(Optional)

Specify the success criteria for the test, code coverage, software composition analysis (SCA), and static analysis (SA) reports.

For more information, see Configuring success criteria for reports.

Corresponding UI:

- Outputs tab/Reports/Automatically discover reports/**Success criteria**
- Outputs tab/Reports/Manually configure reports/*report-name-1*/**Success criteria**

PassRate

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**PassRate**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**PassRate**)

(Optional)

Specify the percentage of tests in a test report that must pass for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. The pass rate criteria are applied only to test reports. For more information about test reports, see Test reports.

Corresponding UI:

- Outputs tab/Reports/Automatically discover reports/Success criteria/**Pass rate**
- Outputs tab/Reports/Manually configure reports/*report-name-1*/Success criteria/**Pass rate**

LineCoverage

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**LineCoverage**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**LineCoverage**)

(Optional)

Specify the percentage of lines in a code coverage report that must be covered for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. Line coverage criteria are applied only to code coverage reports. For more information about code coverage reports, see Code coverage reports.

Corresponding UI:

- Outputs tab/Reports/Automatically discover reports/Success criteria/**Line coverage**
- Outputs tab/Reports/Manually configure reports/*report-name-1*/Success criteria/**Line coverage**

BranchCoverage

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**BranchCoverage**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**BranchCoverage**)

(Optional)

Specify the percentage of branches in a code coverage report that must be covered for the associated CodeCatalyst report to be marked as passed. Valid values include decimal numbers. For example: `50`, `60.5`. Branch coverage criteria are applied only to code coverage reports. For more information about code coverage reports, see Code coverage reports.

Corresponding UI:

- Outputs tab/Reports/Automatically discover reports/Success criteria/**Branch coverage**
- Outputs tab/Reports/Manually configure reports/*report-name-1*/Success criteria/**Branch coverage**

Vulnerabilities

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/**Vulnerabilities**)

Or

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/**Vulnerabilities**)

(Optional)

Specify the maximum number and severity of vulnerabilities permitted in the SCA report for the associated CodeCatalyst report to be marked as passed. To specify vulnerabilities, you must specify:

- The minimum severity of the vulnerabilities you want to include in the count. Valid values, from most to least severe, are: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `INFORMATIONAL`.

  For example, if you choose `HIGH`, then `HIGH` and `CRITICAL` vulnerabilities will be tallied.
- The maximum number of vulnerabilities of the specified severity you want permit. Exceeding this number causes the CodeCatalyst report to be marked as failed. Valid values are whole numbers.

Vulnerabilities criteria are applied only to SCA reports. For more information about SCA reports, see Software composition analysis reports.

To specify the minimum severity, use the `Severity` property. To specify the maximum number of vulnerabilities, use the `Number` property.

For more information about SCA reports, see Quality report types.

Corresponding UI:

- Outputs tab/Reports/Automatically discover reports/Success criteria/**Vulnerabilities**
- Outputs tab/Reports/Manually configure reports/*`report-name-1`*/Success criteria/**Vulnerabilities**

Reports

(*`action-name`*/Outputs/**Reports**)

(Optional)

A section that specifies the configuration for test reports.

Corresponding UI: Outputs tab/**Reports**

report-name-1

(*`action-name`*/Outputs/Reports/**report-name-1**)

(Required if Reports is included)

The name you want to give to the CodeCatalyst report that will be generated from your raw reports.

Corresponding UI: Outputs tab/Reports/Manually configure reports/**Report name**

Format

(*action-name*/Outputs/Reports/*report-name-1*/**Format**)

(Required if Reports is included)

Specify the file format that you're using for your reports. Possible values are as follows.

- For test reports:
  - For Cucumber JSON, specify **Cucumber** (visual editor) or `CUCUMBERJSON` (YAML editor).
  - For JUnit XML, specify **JUnit** (visual editor) or `JUNITXML` (YAML editor).
  - For NUnit XML, specify **NUnit** (visual editor) or `NUNITXML` (YAML editor).
  - For NUnit 3 XML, specify **NUnit3** (visual editor) or `NUNIT3XML` (YAML editor).
  - For Visual Studio TRX, specify **Visual Studio TRX** (visual editor) or `VISUALSTUDIOTRX` (YAML editor).
  - For TestNG XML, specify **TestNG** (visual editor) or `TESTNGXML` (YAML editor).
- For code coverage reports:
  - For Clover XML, specify **Clover** (visual editor) or `CLOVERXML` (YAML editor).
  - For Cobertura XML, specify **Cobertura** (visual editor) or `COBERTURAXML` (YAML editor).
  - For JaCoCo XML, specify **JaCoCo** (visual editor) or `JACOCOXML` (YAML editor).
  - For SimpleCov JSON generated by simplecov, not simplecov-json, specify **Simplecov** (visual editor) or `SIMPLECOV` (YAML editor).
- For software composition analysis (SCA) reports:
  - For SARIF, specify **SARIF** (visual editor) or `SARIFSCA` (YAML editor).

Corresponding UI: Outputs tab/Reports/Manually configure reports/Add report/*report-name-1*/**Report type** and **Report format**

Configuration

(*action-name*/**Configuration**)

(Required) A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

Steps

(*action-name*/Configuration/**Steps**)

(Required)

Specify your GitHub Action code as it appears on the action's details page in GitHub Marketplace. Add the code following these guidelines:

1. Paste the code from the GitHub Action's `steps:` section into the `Steps:` section of the CodeCatalyst workflow. The code starts with a dash (-) and looks similar to the following.

   GitHub code to paste:

   ```
   - name: Lint Code Base
     uses: github/super-linter@v4
     env:
       VALIDATE_ALL_CODEBASE: false
       DEFAULT_BRANCH: master
       GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
   ```

2. Review the code you just pasted and modify it as necessary so that it conforms to CodeCatalyst standards. For example, with the preceding code block, you might remove the code in *red italics*, and add the code in **bold**.

   CodeCatalyst workflow yaml:

   ```
   Steps:
      - name: Lint Code Base
        uses: github/super-linter@v4
        env:
          VALIDATE_ALL_CODEBASE: false
          DEFAULT_BRANCH: mastermain
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
   ```

3. For additional code that's included with the GitHub Action but does not exist inside the `steps:` section, add it to the CodeCatalyst workflow using CodeCatalyst-equivalent code. You can review the Workflow YAML definition to gain insight into how you might port your GitHub code to CodeCatalyst. Detailed migration steps are outside the scope of this guide.

Here is an example of how to specify file paths in a **GitHub Actions** action:

```
Steps:
  - name: Lint Code Base
    uses: github/super-linter@v4
    ...
  - run: cd /sources/WorkflowSource/MyFolder/  && cat file.txt
  - run: cd /artifacts/MyGitHubAction/MyArtifact/MyFolder/  && cat file2.txt
```

For more information about specifying file paths, see Referencing files in a source repository and Referencing files in an artifact.

Corresponding UI: Configuration tab/**GitHub Actions YAML**

Configuring the compute and runtime environment Docker images for a workflow

In a CodeCatalyst workflow, you can specify the compute and runtime environment image that CodeCatalyst uses to run workflow actions.

Compute refers to the computing engine (the CPU, memory, and operating system) managed and maintained by CodeCatalyst to run workflow actions.

> **Note**
>
> If compute is defined as a property of the workflow, then it can't be defined as a property of any action in that workflow. Similarly, if compute is defined as a property of any action, it can't be defined in the workflow.

A *runtime environment image* is a Docker container within which CodeCatalyst runs workflow actions. The Docker container runs on top of your chosen compute platform, and includes an operating system and extra tools that a workflow action might need, such as the AWS CLI, Node.js, and .tar.

Topics

- Creating a provisioned fleet
- Editing a provisioned fleet
- Deleting a provisioned fleet
- Assigning a provisioned fleet or on-demand compute to an action
- Sharing compute across actions
- Specifying runtime environment Docker images

Compute types

CodeCatalyst offers the following compute types:

- Amazon EC2
- AWS Lambda

Amazon EC2 offers optimized flexibility during action runs and Lambda offers optimized action start-up speeds. Lambda supports faster workflow action runs due to a lower start-up latency. Lambda allows you to run basic workflows that can build, test, and deploy serverless applications with common runtimes. These runtimes include Node.js, Python, Java, .NET, and Go. However, there are some use-cases which Lambda does not support, and if they impact you, use the Amazon EC2 compute type:

- Lambda doesn't support runtime environment images from a specified registry.
- Lambda doesn't support tools that require root permissions. For tools such as `yum` or `rpm`, use the Amazon EC2 compute type or other tools that don't require root permissions.
- Lambda doesn't support Docker builds or runs. The following actions that use Docker images are not supported: Deploy AWS CloudFormation stack, Deploy to Amazon ECS, Amazon S3 publish, AWS CDK bootstrap, AWS CDK deploy, AWS Lambda invoke , and GitHub Actions. Docker-based GitHub Actions that are running within CodeCatalyst GitHub Actions action are also not supported with Lambda compute. You can use alternatives that don’t require root permissions, such as Podman.
- Lambda doesn't support writing to files outside `/tmp`. When configuring your workflow actions, you can reconfigure your tools to install or write to `/tmp`. If you have a build action that installs `npm`, make sure you configure it to install to `/tmp`.
- Lambda doesn't support runtimes longer than 15 minutes.

Compute fleets

CodeCatalyst offers the following compute fleets:

- On-demand fleets
- Provisioned fleets

With on-demand fleets, when a workflow action starts, the workflow provisions the resources it needs. The machines are destroyed when the action finishes. You only pay for the number of minutes that you're running your actions. On-demand fleets are fully managed, and includes automatic scaling capabilities to handle spikes in demand.

CodeCatalyst also offers provisioned fleets which contain machines powered by Amazon EC2 that are maintained by CodeCatalyst. With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. With provisioned fleets, your machines are always running and will incur costs as long they're provisioned.

In order to create, update, or delete a fleet, you must have the **Space administrator** role or the **Project administrator** role.

On-demand fleet properties

CodeCatalyst provides the following on-demand fleets:

Name	Operating system	Architecture	vCPUs	Memory (GiB)	Disk space	Supported compute types
`Linux.Arm64.Large`	Amazon Linu	Arm64	2	4	64 GB	Amazon EC2
					10 GB	Lambda
`Linux.Arm64.XLarge`	Amazon Linu	Arm64	4	8	128 GB	Amazon EC2
					10 GB	Lambda

Name	Operating system	Architecture	vCPUs	Memory (GiB)	Disk space	Supported compute types
`Linux.Arm64.2XLarge`	Amazon Linu	Arm64	8	16	128 GB	Amazon EC2
`Linux.x86-64.Large`	Amazon Linu	x86-64	2	4	64 GB	Amazon EC2
					10 GB	Lambda
`Linux.x86-64.XLarge`	Amazon Linu	x86-64	4	8	128 GB	Amazon EC2
					10 GB	Lambda
`Linux.x86-64.2XLarge`	Amazon Linu	x86-64	8	16	128 GB	Amazon EC2

> **Note**
>
> The specifications for on-demand fleets will vary depending on your billing tier. For more information, see Pricing.

If no fleet is selected, CodeCatalyst uses `Linux.x86-64.Large`.

Provisioned fleet properties

A provisioned fleet contains the following properties:

Operating system

The operating system. The following operating systems are available:

- Amazon Linux 2

- Windows Server 2022

> **Note**
>
> Windows fleets are only supported in the build action. Other actions do not currently support Windows.

Architecture

The processor architecture. The following architectures are available:

- x86_64
- Arm64

Machine type

The machine type for each instance. The following machine types are available:

vCPUs	Memory (GiB)	Disk space	Operating system
2	4	64 GB	Amazon Linux 2
4	8	128 GB	Amazon Linux 2
			Windows Server 2022
8	16	128 GB	Amazon Linux 2
			Windows Server 2022

Capacity

The initial number of machines allocated to the fleet, which defines the number of actions that can run in parallel.

Scaling mode

Defines the behavior when the number of actions exceeds the fleet capacity.

Provision additional capacity on demand

Additional machines are set up on demand which automatically scale up in response to new actions running, and then scale down to the base capacity as actions finish. This can incur additional costs, since you pay by the minute for each machine running.

Wait until additional fleet capacity is available

Action runs are placed in a queue until a machine is available. This limits additional costs because no additional machines are allocated.

Creating a provisioned fleet

Use the following instructions to create a provisioned fleet.

> **Note**
>
> Provisioned fleets will be deactivated after 2 weeks of inactivity. If used again, they will be re-activated automatically, but this re-activation may cause a latency to occur.

To create a provisioned fleet

1. In the navigation pane, choose **CI/CD**, and then choose **Compute**.
2. Choose **Create provisioned fleet**.
3. In the **Provisioned fleet name** text field, enter a name for your fleet.
4. From the **Operating system** drop-down menu, choose the operating system.
5. From the **Machine type** drop-down menu, choose the machine type for your machine.
6. In the **Capacity** text field, enter the maximum number of machines in the fleet.
7. From the **Scaling mode** drop-down menu, choose the desired overflow behavior. For more information about these fields, see Provisioned fleet properties.
8. Choose **Create**.

After creating the provisioned fleet, you are ready to assign it to an action. For more information, see Assigning a provisioned fleet or on-demand compute to an action.

Editing a provisioned fleet

Use the following instructions to edit a provisioned fleet.

> **Note**
>
> Provisioned fleets will be deactivated after 2 weeks of inactivity. If used again, they will be re-activated automatically, but this re-activation may cause a latency to occur.

To edit a provisioned fleet

1. In the navigation pane, choose **CI/CD**, and then choose **Compute**.
2. In the **Provisioned fleet** list, choose the fleet you want to edit.
3. Choose **Edit**.
4. In the **Capacity** text field, enter the maximum number of machines in the fleet.
5. From the **Scaling mode** drop-down menu, choose the desired overflow behavior. For more information about these fields, see Provisioned fleet properties.
6. Choose **Save**.

Deleting a provisioned fleet

Use the following instructions to delete a provisioned fleet.

To delete a provisioned fleet

> **Warning**
>
> Before deleting a provisioned fleet, remove it from all actions by deleting the `Fleet` property from the action's YAML code. Any action that continues to reference a provisioned fleet after it is deleted will fail the next time the action runs.

1. In the navigation pane, choose **CI/CD**, and then choose **Compute**.
2. In the **Provisioned fleet** list, choose the fleet you want to delete.
3. Choose **Delete**.

4. Enter **`delete`** to confirm the deletion.
5. Choose **Delete**.

Assigning a provisioned fleet or on-demand compute to an action

By default, workflow actions use the `Linux.x86-64.Large` on-demand fleet with an Amazon EC2 compute type. To use a provisioned fleet instead, or to use a different on-demand fleet, such as `Linux.x86-64.2XLarge`, use the following instructions.

Visual

Before you begin

- If you want to assign a provisioned fleet, you must first create the provisioned fleet. For more information, see Creating a provisioned fleet.

To assign a provisioned fleet or different fleet type to an action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action that you want to assign your provisioned fleet or new fleet type to.
8. Choose the **Configuration** tab.
9. In **Compute fleet**, do the following:

   Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

Before you begin

- If you want to assign a provisioned fleet, you must first create the provisioned fleet. For more information, see Creating a provisioned fleet.

To assign a provisioned fleet or different fleet type to an action

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name./
5. Choose **Edit**.
6. Choose **YAML**.
7. Find the action that you want to assign your provisioned fleet or new fleet type to.
8. In the action, add a `Compute` property and set `Fleet` to the name of your fleet or on-demand fleet type. For more information, see the description of the `Fleet` property in the Build and test action YAML definition for your action.
9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

Sharing compute across actions

By default, actions in a workflow run on separate instances in a fleet. This behavior provides actions with isolation and predictability on the state of inputs. The default behavior requires explicit configuration to share context such as files and variables between actions.

Compute sharing is a capability that allows you to run all the actions in a workflow on the same instance. Using compute sharing can provide faster workflow runtimes because less time is spent provisioning instances. You can also share files (artifacts) between actions without additional workflow configuration.

When a workflow is run using compute sharing, an instance in the default or specified fleet is reserved for the duration of all actions in that workflow. When the workflow run completes, the instance reservation is released.

Topics

Running multiple actions on shared compute

You can use the `Compute` attribute in the definition YAML at the workflow level to specify both the fleet and compute sharing properties of actions. You can also configure compute properties using the visual editor in CodeCatalyst. To specify a fleet, set the name of an existing fleet, set the compute type to **EC2**, and turn on compute sharing.

> **Note**
>
> Compute sharing is only supported if the compute type is set to **EC2**, and it's not supported for the Windows Server 2022 operating system. For more information about compute fleets, compute types, and properties, see Configuring the compute and runtime environment Docker images for a workflow.

> **Note**
>
> If you're on the Free tier and you specify the `Linux.x86-64.XLarge` or `Linux.x86-64.2XLarge` fleet manually in the workflow definition YAML, the action will still run on the default fleet (`Linux.x86-64.Large`). For more information about compute availability and pricing, see the table for the tiers options.

When compute sharing is turned on, the folder containing the workflow source is automatically copied across actions. You don't need to configure output artifacts and reference them as input artifacts throughout a workflow definition (YAML file). As a workflow author, you need to wire up environment variables using inputs and outputs, just as you would without using compute sharing. If you want to share folders between actions outside the workflow source, consider file caching. For more information, see Sharing data between actions in a workflow using artifacts and Caching files between workflow runs.

The source repository where your workflow definition file resides is identified by the label `WorkflowSource`. While using compute sharing, the workflow source is downloaded in the first action that references it and automatically made available for subsequent actions in the workflow run to use. Any changes made to the folder containing the workflow source by an action, such as adding, modifying, or removing files, are also visible in the subsequent actions in the workflow. You can reference files that reside in the workflow source folder in any of your workflow actions, just as you can without using compute sharing. For more information, see Referencing files in a source repository.

> **Note**
>
> Compute sharing workflows need to specify a strict sequence of actions, so parallel actions can't be set. While output artifacts can be configured at any action in the sequence, input artifacts aren't supported.

Considerations for compute sharing

You can run workflows with compute sharing in order to accelerate workflow runs and share context between actions in a workflow that use the same instance. Consider the following to determine whether using compute sharing is appropriate for your scenario:

	Compute sharing	Without compute sharing
Compute type	Amazon EC2	Amazon EC2, AWS Lambda
Instance provisioning	Actions run on same instance	Actions run on separate instances
Operating system	Amazon Linux 2	Amazon Linux 2, Windows Server 2022 (build action only)
Referencing files	`$CATALYST_SOURCE_DIR_WorkflowSource` , `/sources/WorkflowSource/`	`$CATALYST_SOURCE_DIR_WorkflowSource` , `/sources/WorkflowSource/`
Workflow structure	Actions can only run sequentially	Actions can run parallel
Accessing data across workflow actions	Access cached workflow source (`WorkflowSource`)	Access outputs of shared artifacts (requires additional configuration)

Turning on compute sharing

Use the following instruction to turn on compute sharing for a workflow.

Visual

To turn on compute sharing using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow.
5. Choose **Edit**.
6. Choose **Visual**.

7. Choose **Workflow properties**.
8. From the **Compute type** dropdown menu, choose **EC2**.
9. (Optional) From the **Compute fleet - optional** dropdown menu, choose a fleet you want to use to run workflow actions. You can choose an on-demand fleet or create and choose a provisioned fleet. For more information, see Creating a provisioned fleet and Assigning a provisioned fleet or on-demand compute to an action
10. Switch the toggle to turn on compute sharing and have actions in the workflow run on the same fleet.
11. (Optional) Choose the run mode for the workflow. For more information, see Configuring the queuing behavior of runs.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To turn on compute sharing using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow.
5. Choose **Edit**.
6. Choose **YAML**.
7. Turn on compute sharing setting the `SharedInstance` field to `TRUE` and `Type` to `EC2`. Set `Fleet` to a compute fleet you want to use to run workflow actions. You can choose an on-demand fleet or create and choose a provisioned fleet. For more information, see Creating a provisioned fleet and Assigning a provisioned fleet or on-demand compute to an action

   In a workflow YAML, add code similar to the following:

```
Name: MyWorkflow
SchemaVersion: "1.0"
Compute: # Define compute configuration.
  Type: EC2
  Fleet: MyFleet # Optionally, choose an on-demand or provisioned fleet.
  SharedInstance: true # Turn on compute sharing. Default is False.
```

```
  Actions:
    BuildFirst:
      Identifier: aws/build@v1
      Inputs:
        Sources:
          - WorkflowSource
      Configuration:
        Steps:
          - Run: ...
          ...
```

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Examples

Topics

- Example: Amazon S3 Publish

Example: Amazon S3 Publish

The following workflow examples show how to perform the Amazon Amazon S3 Publish action in two ways: first using input artifacts and then using compute sharing. With compute sharing, the input artifacts aren't needed since you can access the cached `WorkflowSource`. Additionally, the output artifact in the Build action is no longer needed. The S3 Publish action is configured to use the explicit `DependsOn` property to maintain sequential actions; the Build action must run successfully in order for the S3 Publish action to run.

- Without compute sharing, you need to use input artifacts and share the outputs with subsequent actions:

```
Name: S3PublishUsingInputArtifact
SchemaVersion: "1.0"
Actions:
  Build:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ArtifactToPublish
```

```
          Files: [output.zip]
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: ./build.sh # Build script that generates output.zip
  PublishToS3:
    Identifier: aws/s3-publish@v1
    Inputs:
      Artifacts:
      - ArtifactToPublish
    Environment:
      Connections:
        - Role: codecatalyst-deployment-role
          Name: dev-deployment-role
      Name: dev-connection
    Configuration:
      SourcePath: output.zip
      DestinationBucketName: dev-bucket
```

- When using compute sharing by setting `SharedInstance` to TRUE, you can run multiple actions on the same instance and share artifacts by specifying a single workflow source. Input artifacts aren't required and can't be specified:

```
Name: S3PublishUsingComputeSharing
SchemaVersion: "1.0"
Compute:
  Type: EC2
  Fleet: dev-fleet
  SharedInstance: TRUE
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: ./build.sh # Build script that generates output.zip
  PublishToS3:
```

```
    Identifier: aws/s3-publish@v1
    DependsOn:
      - Build
    Environment:
      Connections:
        - Role: codecatalyst-deployment-role
          Name: dev-deployment-role
      Name: dev-connection
    Configuration:
      SourcePath: output.zip
      DestinationBucketName: dev-bucket
```

Specifying runtime environment Docker images

A *runtime environment image* is a Docker container within which CodeCatalyst runs workflow actions. The Docker container runs on top of your chosen compute platform, and includes an operating system and extra tools that a workflow action might need, such as the AWS CLI, Node.js, and .tar.

By default, workflow actions will run on one of the active images that are supplied and maintained by CodeCatalyst. Only build and test actions support custom images. For more information, see Assigning a custom runtime environment Docker image to an action.

Topics

Active images

Active images are runtime environment images that are fully supported by CodeCatalyst and include preinstalled tooling. There are currently two sets of active images: one released in March 2024, and another released in November 2022.

Whether an action uses a March 2024 or November 2022 image depends on the action:

- Build and test actions that are added to a workflow on or after March 26, 2024 will include a `Container` section in their YAML definition that explicitly specifies a March 2024 image. You can optionally remove the `Container` section to revert back to the November 2022 image.
- Build and test actions that were added to a workflow prior to March 26, 2024 will *not* include a `Container` section in their YAML definition, and consequently will use a November 2022 image. You can keep the November 2022 image, or you can upgrade it. To upgrade the image, open the action in the visual editor, choose the **Configuration** tab, and then select the March 2024 image from the **Runtime environment docker image** drop-down list. This selection will add a `Container` section to the action's YAML definition that is populated with the appropriate March 2024 image.
- All other actions will use a November 2022 image regardless of when they were added to the workflow. Upgrading these actions to use a March 2024 image is currently not possible.

Topics

March 2024 images

The March 2024 images are the latest images provided by CodeCatalyst. There is one March 2024 image per compute type/fleet combination.

The following table shows the tools installed on each March 2024 image.

March 2024 image tools

Tool	CodeCatalyst Amazon EC2 for Linux x86_64 - `CodeCatalystLinux_x86_64:2024_03`	CodeCatalyst Lambda for Linux x86_64 - `CodeCatalystLinuxLambda_x86_64:2024_03`	CodeCatalyst Amazon EC2 for Linux Arm64 - `CodeCatalystLinux_Arm64:2024_03`	CodeCatal Lambda fo Linux Arm - `CodeCat ystLinux ambda_Ar 64:2024_`
AWS CLI	2.15.17	2.15.17	2.15.17	2.15.17
AWS Copilot CLI	1.32.1	1.32.1	1.32.1	1.32.1

Tool	CodeCatalyst Amazon EC2 for Linux x86_64 - CodeCatalystLinux_x86_64:2024_03	CodeCatalyst Lambda for Linux x86_64 - CodeCatalystLinuxLambda_x86_64:2024_03	CodeCatalyst Amazon EC2 for Linux Arm64 - CodeCatalystLinux_Arm64:2024_03	CodeCatal Lambda fo Linux Arm - CodeCat ystLinux ambda_Ar 64:2024_
Docker	24.0.9	N/A	24.0.9	N/A
Docker Compose	2.23.3	N/A	2.23.3	N/A
Git	2.43.0	2.43.0	2.43.0	2.43.0
Go	1.21.5	1.21.5	1.21.5	1.21.5
Gradle	8.5	8.5	8.5	8.5
Java	Corretto17	Corretto17	Corretto17	Corretto17
Maven	3.9.6	3.9.6	3.9.6	3.9.6
Node.js	18.19.0	18.19.0	18.19.0	18.19.0
npm	10.2.3	10.2.3	10.2.3	10.2.3
Python	3.9.18	3.9.18	3.9.18	3.9.18
Python3	3.11.6	3.11.6	3.11.6	3.11.6
pip	22.3.1	22.3.1	22.3.1	22.3.1
.NET	8.0.100	8.0.100	8.0.100	8.0.100

November 2022 images

There is one November 2022 image per compute type/fleet combination. There is also a November 2022 Windows image available with the build action if you've configured a provisioned fleet.

The following table shows the tools installed on each November 2022 image.

November 2022 image tools

Tool	CodeCatalyst Amazon EC2 for Linux x86_64 - CodeCatalystLinux_x86_64:2022_11	CodeCatalyst Lambda for Linux x86_64 - CodeCatalystLinuxLambda_x86_64:2022_11	CodeCatalyst Amazon EC2 for Linux Arm64 - CodeCatalystLinux_Arm64:2022_11	CodeCatal Lambda fo Linux Arm - CodeCat ystLinux ambda_Ar 64:2022_
AWS CLI	2.15.17	2.15.17	2.15.17	2.15.17
AWS Copilot CLI	0.6.0	0.6.0	N/A	N/A
Docker	23.01	N/A	23.0.1	N/A
Docker Compose	2.16.0	N/A	2.16.0	N/A
Git	2.40.0	2.40.0	2.39.2	2.39.2
Go	1.20.2	1.20.2	1.20.1	1.20.1
Gradle	8.0.2	8.0.2	8.0.1	8.0.1
Java	Corretto17	Corretto17	Corretto17	Corretto17
Maven	3.9.4	3.9.4	3.9.0	3.9.0
Node.js	16.20.2	16.20.2	16.19.1	16.14.2
npm	8.19.4	8.19.4	8.19.3	8.5.0
Python	3.9.15	2.7.18	3.11.2	2.7.18
Python3	N/A	3.9.15	N/A	3.11.2
pip	22.2.2	22.2.2	23.0.1	23.0.1
.NET	6.0.407	6.0.407	6.0.406	6.0.406

What if an active image doesn't include the tools I need?

If none of the active images supplied by CodeCatalyst include the tools you need, you have a couple of options:

- You can provide a custom runtime environment Docker image that includes the necessary tools. For more information, see Assigning a custom runtime environment Docker image to an action.

  > **Note**
  >
  > If you want to provide a custom runtime environment Docker image, make sure that your custom image has Git installed in it.

- You can have your workflow's build or test action install the tools you need.

  For example, you could include the following instructions in the `Steps` section of the build or test action's YAML code:

  ```
  Configuration:
    Steps:
      - Run: ./setup-script
  ```

  The *`setup-script`* instruction would then run the following script to install the Node package manager (npm):

  ```
  #!/usr/bin/env bash
  echo "Setting up environment"

  touch ~/.bashrc
  curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
  source ~/.bashrc
  nvm install v16.1.0
  source ~/.bashrc
  ```

  For more information about the build action YAML, see Build and test action YAML definition.

Assigning a custom runtime environment Docker image to an action

If you don't want to use an Active image supplied by CodeCatalyst, you can provide a custom runtime environment Docker image. If you want to provide a custom image, make sure it has Git installed in it. The image can reside in Docker Hub, Amazon Elastic Container Registry, or any public repository.

To learn how to create a custom Docker image, see Containerize an application in the Docker documentation.

Use the following instructions to assign your custom runtime environment Docker image to an action. After specifying an image, CodeCatalyst deploys it to your compute platform when the action starts.

> **Note**
>
> The following actions do not support custom runtime environment Docker images: **Deploy AWS CloudFormation stack**, **Deploy to ECS**, and **GitHub Actions**. custom runtime environment Docker images also do not support the **Lambda** compute type.

Visual

To assign a custom runtime environment Docker image using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
4. Choose **Edit**.
5. Choose **Visual**.
6. In the workflow diagram, choose the action that will use your custom runtime environment Docker image.
7. Choose the **Configuration** tab.
8. Near the bottom, fill out the following fields.

   Runtime environment Docker image - optional

Specify the registry where your image is stored. Valid values include:

- `CODECATALYST` (YAML editor)

  The image is stored in the CodeCatalyst registry.

- **Docker Hub** (visual editor) or `DockerHub` (YAML editor)

  The image is stored in the Docker Hub image registry.

- **Other registry** (visual editor) or `Other` (YAML editor)

  The image is stored in a custom image registry. Any publicly available registry can be used.

- **Amazon Elastic Container Registry** (visual editor) or `ECR` (YAML editor)

  The image is stored in an Amazon Elastic Container Registry image repository. To use an image in an Amazon ECR repository, this action needs access to Amazon ECR. To enable this access, you must create an IAM role that includes the following permissions and custom trust policy. (You can modify an existing role to include the permissions and policy, if you want.)

  The IAM role must include the following permissions in its role policy:

  - `ecr:BatchCheckLayerAvailability`
  - `ecr:BatchGetImage`
  - `ecr:GetAuthorizationToken`
  - `ecr:GetDownloadUrlForLayer`

  The IAM role must include the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
```

```
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

For more information about creating IAM roles, see Creating a role using custom trust policies (console) in the *IAM User Guide*.

Once you have created the role, you must assign it to the action through an environment. For more information, see Associating an environment with a workflow action.

ECR image URL, **Docker Hub image** or **Image URL**

Specify one of the following:

- If you are using a `CODECATALYST` registry, set the image to one of the of the following active images:
  - `CodeCatalystLinux_x86_64:2024_03`
  - `CodeCatalystLinux_x86_64:2022_11`
  - `CodeCatalystLinux_Arm64:2024_03`
  - `CodeCatalystLinux_Arm64:2022_11`
  - `CodeCatalystLinuxLambda_x86_64:2024_03`
  - `CodeCatalystLinuxLambda_x86_64:2022_11`
  - `CodeCatalystLinuxLambda_Arm64:2024_03`
  - `CodeCatalystLinuxLambda_Arm64:2022_11`
  - `CodeCatalystWindows_x86_64:2022_11`
- If you are using a Docker Hub registry, set the image to the Docker Hub image name and optional tag.

  Example: `postgres:latest`
- If you are using an Amazon ECR registry, set the image to the Amazon ECR registry URI.

  Example: `111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo`

- If you are using a custom registry, set the image to the value expected by the custom registry.

9. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
10. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To assign a custom runtime environment Docker image using the YAML editor

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
3. Choose **Edit**.
4. Choose **YAML**.
5. Find the action that you want to assign a runtime environment Docker image to.
6. In the action, add a `Container` section and underlying `Registry` and `Image` properties. For more information, see the description of the `Container`, `Registry` and `Image` properties in the Actions for your action.
7. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
8. Choose **Commit**, enter a commit message, and choose **Commit** again.

Examples

The following examples show how to assign a custom runtime environment Docker image to an action in the workflow definition file.

Topics

- Example: Using a custom runtime environment Docker image to add support for Node.js 18 with Amazon ECR
- Example: Using a custom runtime environment Docker image to add support for Node.js 18 with Docker Hub

Example: Using a custom runtime environment Docker image to add support for Node.js 18 with Amazon ECR

The following example shows how to use a custom runtime environment Docker image to add support for Node.js 18 with Amazon ECR.

```
Configuration:
  Container:
    Registry: ECR
    Image: public.ecr.aws/amazonlinux/amazonlinux:2023
```

Example: Using a custom runtime environment Docker image to add support for Node.js 18 with Docker Hub

The following example shows how to use a custom runtime environment Docker image to add support for Node.js 18 with Docker Hub.

```
Configuration:
  Container:
    Registry: DockerHub
    Image: node:18.18.2
```

Connecting a workflow to a source repository

A *source*, also called an *input source*, is a source repository to which a workflow action connects in order to obtain the files it needs to carry out its operations. For example, a workflow action might connect to a source repository to obtain application source files in order to build an application.

CodeCatalyst workflows support the following sources:

- CodeCatalyst source repositories – For more information, see Store and collaborate on code with source repositories in CodeCatalyst.
- GitHub repositories, Bitbucket repositories, and GitLab project repositories – For more information, see Add functionality to projects with extensions in CodeCatalyst.

Topics

- Referencing files in a source repository
- Variables produced by the source ("BranchName" and "CommidId")

Specifying the source that will store the workflow definition file

Use the following instructions to specify the CodeCatalyst source repository where you want to store your workflow definition file. If you'd rather specify a GitHub repositoriy, Bitbucket repository, or GitLab project repository, see instead Add functionality to projects with extensions in CodeCatalyst.

The source repository where your workflow definition file resides is identified by the label, `WorkflowSource`.

> **Note**
>
> You specify the source repository where your workflow definition file resides when you first commit your workflow definition file. After this commit, the repository and workflow definition file are linked together permanently. The only way to change the repository after the initial commit is to re-create the workflow in a different repository.

To specify the source repository that will store the workflow definition file

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose **Create workflow** and create the workflow. For more information, see To create a workflow using the visual editor.

   During the workflow creation process, you are asked to specify the CodeCatalyst repository where you want to store your workflow definition file.

Specifying the source that a workflow action will use

Use the following instructions to specify a source repository to use with a workflow action. On startup, the action bundles the files at the configured source repository into an artifact, downloads

the artifact to the runtime environment Docker image where the action is running, and then completes its processing using the downloaded files.

> **Note**
>
> Currently, within a workflow action, you can only specify one source repository, which is the source repository where the workflow definition file resides (in the `.codecatalyst/workflows/` directory). This source repository is represented by the label `WorkflowSource`.

Visual

To specify the source repository that an action will use (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action where you want to specify the source.
8. Choose **Inputs**.
9. In **Sources - optional** do the following:

   Specify the labels that represent the source repositories that will be needed by the action. Currently, the only supported label is `WorkflowSource`, which represents the source repository where your workflow definition file is stored.

   If you omit a source, then you must specify at least one input artifact under *`action-name`*`/Inputs/Artifacts`.

   For more information about sources, see Connecting a workflow to a source repository.
10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To specify the source repository that an action will use (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In an action, add code similar to the following:

```
action-name:
 Inputs:
   Sources:
     - WorkflowSource
```

   For more information, see the description of the `Sources` property in Workflow YAML definition for your action.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Referencing files in a source repository

If you have files that reside in a source repository, and you need to refer to these files in one of your workflow actions, complete the following procedure.

> **Note**
>
> See also Referencing files in an artifact.

To reference a file in a source repository

- In the action where you want to reference a file, add code similar to the following:

```
Actions:
  My-action:
    Inputs:
      Sources:
        - WorkflowSource
      Configuration:
        Steps:
        - run: cd my-app && cat file1.jar
```

In the previous code, the action looks in the `my-app` directory in the root of the `WorkflowSource` source repository to find and display the `file1.jar` file.

Variables produced by the source ("BranchName" and "CommidId")

The CodeCatalyst source produces and sets "BranchName" and "CommitId" variables when your workflow runs. These are known as *predefined variables*. See the following table for information about these variables.

For information about referencing these variables in a workflow, see Using predefined variables.

Key	Value
CommitId	The commit ID representing the state of the repository at the time the workflow run started. Example: `example3819261db00a3ab59468c8b` See also: Example: Referencing the "CommitId " predefined variable
BranchName	The name of the branch against which the workflow run started. Examples: `main`, `feature/branch` , `test-LiJuan`

Key	Value
	See also: Example: Referencing the "BranchName" predefined variable

Publishing and importing packages using a workflow

A *package* is a bundle that includes both software and the metadata that is required to install the software and resolve any dependencies. CodeCatalyst supports the npm package format.

A package consists of:

- A name (for example, `webpack` is the name of a popular npm package)
- An optional namespace (for example, `@types` in `@types/node`)
- A set of versions (for example, `1.0.0`, `1.0.1`, `1.0.2`)
- Package-level metadata (for example, npm dist tags)

In CodeCatalyst, you can publish packages to and consume packages from CodeCatalyst package repositories in your workflows. You can configure a build or test action with a CodeCatalyst package repository to automatically configure an action's npm client to push and pull packages from the specified repository.

For more information about packages, see Publish and share software packages in CodeCatalyst.

> **Note**
>
> Currently, build and test actions support CodeCatalyst package repositories.

Topics

- Tutorial: Pull dependencies from a CodeCatalyst package repository using a workflow
- Specifying CodeCatalyst package repositories in workflows
- Using authorization tokens in workflow actions
- Examples of specifying package repositories in workflows

Tutorial: Pull dependencies from a CodeCatalyst package repository using a workflow

In this tutorial, you learn how to create a workflow that runs an application whose dependencies are pulled from a CodeCatalyst package repository. The application is a simple Node.js app that prints a 'Hello World' message to the CodeCatalyst logs. The application has a single dependency: the lodash npm package. The `lodash` package is used to transform a `hello-world` string to `Hello World`. You will use version 4.17.20 of this package.

After setting up your application and workflow, you configure CodeCatalyst to block additional versions of `lodash` from being imported into the CodeCatalyst package repository from the public external registry (npmjs.com). You then test that additional versions of `lodash` are blocked successfully.

By the end of this tutorial, you should have a good understanding of how a workflow interacts with package repositories, both inside and outside CodeCatalyst, in order to retrieve packages. You should also understand the behind-the-scenes interactions that occur between npm, your package repository, your workflow, and your application's `package.json` file.

Topics

Prerequisites

Before you begin:

- You need a CodeCatalyst **Space**. For more information, see Creating a space.

- In your CodeCatalyst space, you need an empty, **Start from scratch** CodeCatalyst **project** called:

```
codecatalyst-package-project
```

  For more information, see Creating an empty project in Amazon CodeCatalyst.

Step 1: Create a source repository

In this step, you create a source repository in CodeCatalyst. This repository stores the tutorial's source files, such as the `index.js` and `package.json` files.

For more information about source repositories, see Creating a source repository.

To create a source repository

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your project, `codecatalyst-package-project`.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose **Add repository**, and then choose **Create repository**.
5. In **Repository name**, enter:

```
hello-world-app
```

6. Choose **Create**.

Step 2: Create the CodeCatalyst and gateway package repositories

In this step, you create a package repository in your CodeCatalyst project, and connect it to a gateway repository, also in your CodeCatalyst project. You later import the tutorial's dependency, `lodash`, from npmjs.com into both repositories.

The gateway repository is the 'glue' that connects your package repository in CodeCatalyst to the public npmjs.com.

For more information about package repositories, see Publish and share software packages in CodeCatalyst.

> **Note**
>
> This tutorial uses the terms *CodeCatalyst package repository* and *gateway repository* to refer to the two repositories that you create in CodeCatalyst in the following procedure.

To create CodeCatalyst package and gateway repositories

1. In the navigation pane, choose **Packages**.
2. Choose **Create package repository**.
3. In **Repository name**, enter:

   ```
   codecatalyst-package-repository
   ```

4. Choose **+ Select upstream repositories**.
5. Choose **Gateway repositories**.
6. In the **npm-public-registry-gateway** box, choose **Create**.
7. Choose **Select**.
8. Choose **Create**.

   CodeCatalyst creates a package repository called `codecatalyst-package-repository` which is connected to a gateway repository. The gateway repository is connected to the npmjs.com registry.

Step 3: Create the 'Hello World' application

In this step, you create a 'Hello World' Node.js application and import its dependency (`lodash`) into your gateway and CodeCatalyst package repositories.

To create the application, you need a development machine with Node.js and the associated `npm` client installed.

This tutorial assumes you'll be using a CodeCatalyst Dev Environment as your development machine. Although you don't have to use a CodeCatalyst Dev Environment, it is recommended because it provides a clean working environment, has Node.js and `npm` preinstalled, and is easy to delete when you have finished the tutorial. For more information about CodeCatalyst Dev Environments, see Creating a Dev Environment.

Use the following instructions to launch a CodeCatalyst Dev Environment and use it to create the 'Hello World' application.

To launch a CodeCatalyst Dev Environment

1. In the navigation pane, choose **Code**, and then choose **Dev Environments**.
2. Near the top choose **Create Dev Environment**, and then choose **AWS Cloud9 (in browser)**.
3. Make sure that **Repository** is set to `hello-world-app` and **Existing branch** is set to `main`. Choose **Create**.

   Your Dev Environment launches in a new browser tab, and your repository (`hello-world-app`) is cloned into it.
4. Leave both CodeCatalyst browser tabs open, and go to the next procedure.

To create the 'Hello World' Node.js application

1. Go to your Dev Environment.
2. At the terminal prompt, change to the `hello-world-app` source repository root directory:

   ```
   cd hello-world-app
   ```
3. Initialize a Node.js project:

   ```
   npm init -y
   ```

   The initialization creates a `package.json` file in the root directory of `hello-world-app`.
4. Connect the npm client in your Dev Environment to your CodeCatalyst package repository:

   1. Switch to the CodeCatalyst console.
   2. In the navigation pane, choose **Packages**.
   3. Choose `codecatalyst-package-repository`.
   4. Choose **Connect to repository**.
   5. Choose **Create token**. A personal access token (PAT) is created for you.
   6. Choose **Copy** to copy the commands.
   7. Switch to your Dev Environment.
   8. Make sure you're in the `hello-world-app` directory.

9. Paste the commands. They look similar to the following:

```
npm set registry=https://packages.us-west-2.codecatalyst.aws/npm/ExampleCompany/
codecatalyst-package-project/codecatalyst-package-repository/ --location project
npm set //packages.us-west-2.codecatalyst.aws/npm/ExampleCompany/codecatalyst-
package-project/hello-world-app/:_authToken=username:token-secret
```

5. Import `lodash` version 4.17.20:

```
npm install lodash@v4.17.20 --save --save-exact
```

npm looks for `lodash` version 4.17.20 in the following locations, in the following order:

- In the Dev Environment. It can't find it here.
- In the CodeCatalyst package repository. It can't find it here.
- In the gateway repository. It can't find it here.
- In npmjs.com. It finds it here.

npm imports `lodash` into the gateway repository, the CodeCatalyst package repository, and the Dev Environment.

> **Note**
>
> If you had not connected the npm client to your CodeCatalyst package repository in step 4, then npm would have pulled `lodash` directly from npmjs.com and would not have imported the package to either repository.

npm also updates your `package.json` file with the `lodash` dependency, and creates a `node_modules` directory containing `lodash` and all its dependencies.

6. Test that `lodash` was successfully imported to your Dev Environment. Enter:

```
npm list
```

The following message appears, indicating a successful import:

```
`-- lodash@4.17.20
```

7. (Optional) Open `hello-world-app/package.json` and verify that the lines in ***red bold*** were added:

```
{
  "name": "hello-world-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  dependencies": {
    "lodash": "4.17.20"
  }
}
```

8. In `/hello-world-app`, create a file called `index.js` with the following contents:

> **Tip**
>
> You can use the side navigation in your Dev Environment to create this file.

```
// Importing lodash library
const _ = require('lodash');

// Input string
const inputString = 'hello-world';

// Transforming the string using lodash
const transformedString = _.startCase(inputString.replace('-', ' '));

// Outputting the transformed string to the console
console.log(transformedString);
```

To test that 'lodash' was imported to your gateway and CodeCatalyst package repositories

1. Switch to the CodeCatalyst console.
2. In the navigation pane, choose **Packages**.
3. Choose **npm-public-registry-gateway**.
4. Make sure `lodash` is displayed. The **Latest version** column indicates `4.17.20`.
5. Repeat this procedure for the `codecatalyst-package-repository`. You might need to refresh the browser window to see the imported package.

To test 'Hello World' in your Dev Environment

1. Switch to your Dev Environment.
2. Make sure you're still in the `hello-world-app` directory, and then run the application:

   ```
   node index.js
   ```

   A `Hello World` message appears. Node.js ran the application using the `lodash` package that you downloaded to your Dev Environment in a previous step.

To ignore the 'node_modules' directory and commit 'Hello World'

1. Ignore the `node_modules` directory. Enter:

   ```
   echo "node_modules/" >> .gitignore
   ```

   It is a best practice to avoid committing this directory. Also, committing this directory will interfere with later steps in this tutorial.
2. Add, commit, and push:

   ```
   git add .
   git commit -m "add the Hello World application"
   git push
   ```

   The 'Hello World' application and project files are added to your source repository.

Step 4: Create a workflow that runs 'Hello World'

In this step, you create a workflow that runs the 'Hello World' application using the `lodash` dependency. The workflow includes a single *action*, or task, called `RunHelloWorldApp`. The `RunHelloWorldApp` action includes the following noteworthy commands and sections:

- **`Packages`**

  This section indicates the name of the CodeCatalyst package repository that the action must connect to when running `npm install`.

- **`- Run: npm install`**

  This command tells npm to install the dependencies specified in the `package.json` file. The only dependency specified in the `package.json` file is `lodash`. npm looks for `lodash` in the following locations:

  - In the Docker image running the action. It can't find it here.
  - In the CodeCatalyst package repository. It finds it here.

  After npm finds `lodash`, it imports it to the Docker image running the action.

- **`- Run: npm list`**

  This command prints out which version of `lodash` was downloaded to the Docker image running the action.

- **`- Run: node index.js`**

  This command runs the 'Hello World' application using the dependency specified in the `package.json` file.

Notice that the `RunHelloWorldApp` action is a build action, as indicated by the `aws/build@v1` identifier near the top of the workflow. For more information about the build action, see Building with workflows.

Use the following instructions to create a workflow that pulls the `lodash` dependency from your CodeCatalyst package repository and then runs your 'Hello World' application.

To create a workflow

1. Switch to the CodeCatalyst console.

2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose **Create workflow**.
4. For **Source repository**, choose `hello-world-app`.
5. For **Branch**, choose `main`.

   The workflow definition file is will be created in the chosen source repository and branch.
6. Choose **Create**.
7. Choose **YAML** near the top.
8. Delete the YAML sample code.
9. Add the following YAML code:

```
Name: codecatalyst-package-workflow
SchemaVersion: "1.0"

# Required - Define action configurations.
Actions:
  RunHelloWorldApp:
    # Identifies the action. Do not modify this value.
    Identifier: aws/build@v1
    Compute:
      Type: Lambda
    Inputs:
      Sources:
        - WorkflowSource # This specifies your source repository.
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm list
        - Run: node index.js
      Container: # This specifies the Docker image that runs the action.
        Registry: CODECATALYST
        Image: CodeCatalystLinuxLambda_x86_64:2024_03
    Packages:
      NpmConfiguration:
        PackageRegistries:
          - PackagesRepository: codecatalyst-package-repository
```

In the preceding code, replace *codecatalyst-package-repository* with the name of the CodeCatalyst package repository that you created in Step 2: Create the CodeCatalyst and gateway package repositories.

For information about the properties in this file, see the Build and test action YAML definition.

10. (Optional) Choose **Validate** to make sure the YAML code is valid before committing.
11. Choose **Commit**.
12. On the **Commit workflow** dialog box, enter the following:

    a. For **Workflow file name**, keep the default, `codecatalyst-package-workflow`.

    b. For **Commit message**, enter:

    ```
    add initial workflow file
    ```

    c. For **Repository**, choose **hello-world-app**.

    d. For **Branch name**, choose **main**.

    e. Choose **Commit**.

    You have now created a workflow.

To run the workflow

1. Next to the workflow you just created (`codecatalyst-package-workflow`), choose **Actions** and then choose **Run**.

   A workflow run starts.
2. In the green notification at the top, on the right, choose the link to the run. The link looks similar to `View Run-1234`.

   A workflow diagram appears, showing who started the run and the **RunHelloWorldApp** action.
3. Choose the **RunHelloWorldApp** action box to watch the action's progress.
4. When the run finishes, go to Step 5: Verify the workflow.

Step 5: Verify the workflow

In this step, you verify that the workflow successfully ran the 'Hello World' application with its `lodash` dependency.

To verify that the 'Hello World' application ran using its dependency

1. In the workflow diagram, choose the **RunHelloWorldApp** box.

   A list of log messages appear.

2. Expand the `node index.js` log message.

   The following message appears:

   ```
   [Container] 2024/04/24 21:15:41.545650 Running command node index.js
   Hello World
   ```

   The appearance of `Hello Word` (instead of `hello-world`) indicates that the `lodash` dependency was successfully used.

3. Expand the `npm list` log.

   A message similar to the following appears:

   ```
   ### lodash@4.17.20
   ```

   This message indicates that `lodash` version 4.17.20 was downloaded to the Docker image running the workflow action.

Step 6: Block imports from npmjs.com

Now that `lodash` version 4.17.20 is present in your gateway and CodeCatalyst package repositories, you can block imports of other versions. Blocking prevents you from accidentally importing later (or earlier) versions of `lodash`, which might contain malicious code. For more information, see Editing package origin controls and Dependency substitution attacks.

Use the following instructions to block imports of `lodash` into your gateway repository. When you block packages at the gateway, they are also blocked at downstream locations.

To block imports to your gateway repository

1. In the navigation pane, choose **Packages**.
2. Choose **npm-publish-registry-gateway**.
3. Choose `lodash`.
4. Near the top, choose **Origin controls**.
5. Under **Upstream**, choose **Block**.
6. Choose **Save**.

   You have now blocked imports into your gateway repository (and downstream repositories and computers) from npmjs.com.

Step 7: Test the blocking feature

In this section, you verify that the blocking you set up in Step 6: Block imports from npmjs.com is working. You start by configuring 'Hello World' to request version 4.17.2**1** of `lodash` instead of the one available in your gateway repository, which is 4.17.2**0**. You then check that the application cannot pull version 4.17.21 from nmpjs.com, indicating a successful blockage. As a final test, you unblock imports to your gateway repository, and check that the application can successfully pull version 4.17.21 of `lodash`.

Use the following set of procedures to test the blocking feature.

Before you begin

1. Switch to your Dev Environment.
2. Pull the `codecatalyst-package-workflow.yaml` file that you created using the CodeCatalyst console earlier:

```
git pull
```

To configure 'Hello World' to request version 4.17.21 of 'lodash'

1. Open `/hello-world-app/package.json`.
2. Change the `lodash` version to 4.17.21 as shown in ***red bold***:

```
{
```

```
  "name": "hello-world-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "lodash": "4.17.21"
  }
}
```

There is now a mismatch between the version in the `package.json` file (4.17.21) and the version in the gateway and CodeCatalyst package repositories (4.17.20).

3. Add, commit, and push:

```
git add .
git commit -m "update package.json to use lodash 4.17.21"
git push
```

To test that 'Hello World' cannot pull version 4.17.21 of 'lodash'

1. Run the workflow with the version mismatch:

   1. Switch to the CodeCatalyst console.
   2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
   3. Next to `codecatalyst-package-workflow`, choose **Actions**, and then choose **Run.**

   npm looks in `package.json` for dependencies and sees that version 4.17.21 of `lodash` is required by 'Hello World'. npm looks for the dependency in the following locations, in the following order:

   - In the Docker image running the action. It can't find it here.
   - In the CodeCatalyst package repository. It can't find it here.
   - In the gateway repository. It can't find it here.
   - In npmjs.com. It finds it here.

After npm finds version 4.17.21 in npmjs.com, it tries to import it into the gateway repository, but because you set up the gateway to block imports of `lodash`, the import does not occur.

Because the import does not occur, the workflow fails.

2. Verify that the workflow failed:

   1. In the green notification at the top, on the right, choose the link to the run. The link looks similar to `View Run-2345`.
   2. In the workflow diagram, choose the **RunHelloWorldApp** box.
   3. Expand the `npm install` log message.

      The following message appears:

```
[Container] 2024/04/25 17:20:34.995591 Running command npm install
npm ERR! code ETARGET
npm ERR! notarget No matching version found for lodash@4.17.21.
npm ERR! notarget In most cases you or one of your dependencies are requesting
npm ERR! notarget a package version that doesn't exist.

npm ERR! A complete log of this run can be found in: /tmp/.npm/
_logs/2024-05-08T22_03_26_493Z-debug-0.log
```

      The error indicates that version 4.17.21 couldn't be found. This is expected because you blocked it.

To unblock imports from npmjs.com

1. In the navigation pane, choose **Packages**.
2. Choose **npm-publish-registry-gateway**.
3. Choose `lodash`.
4. Near the top, choose **Origin controls**.
5. Under **Upstream**, choose **Allow**.
6. Choose **Save**.

   You have now unblocked imports of `lodash`.

Your workflow can now import version 4.17.21 of `lodash`.

To test that imports from npmjs.com are unblocked

1. Run your workflow again. This time the workflow should succeed because the import of 4.17.21 should now work. To run the workflow again:

   1. Choose **CI/CD** and then choose **Workflows**.
   2. Next to `codecatalyst-package-workflow`, choose **Actions** and choose **Run**.
   3. In the green notification at the top, on the right, choose the link to the run. The link looks similar to `View Run-3456`.

      A workflow diagram appears, showing who started the run and the **RunHelloWorldApp** action.
   4. Choose the **RunHelloWorldApp** action box to watch the action's progress.
   5. Expand the `npm list` log message and verify that a message similar to the following appears:

      ```
      ### lodash@4.17.21
      ```

      This message indicates that `lodash` version 4.17.21 was downloaded.
2. Verify that version 4.17.21 was imported to your CodeCatalyst and gateway repositories:

   1. In the navigation pane, choose **Packages**.
   2. Choose **npm-public-registry-gateway**.
   3. Find `lodash` and make sure the version is `4.17.21`.

      > **Note**
      >
      > Although version 4.17.20 is not listed on this page, you can find it by choosing `lodash` and then choosing **Versions** near the top.
   4. Repeat these steps to check that version 4.17.21 was imported to `codecatalyst-package-repository`.

Clean up

Clean up the files and services used in this tutorial to avoid being charged for them.

To clean up the packages tutorial

1. Delete the `codecatalyst-package-project`:
   a. In the CodeCatalyst console, nagivate to the `codecatalyst-package-project` project if you're not there already.
   b. In the navigation pane, choose **Project settings**.
   c. Choose **Delete project**, enter **`delete`**, and choose **Delete project**.

      CodeCatalyst deletes all project resources, including the source, gateway, and CodeCatalyst package repositories. The Dev Environment is also deleted.
2. Delete the PAT token:
   a. Choose your username on the right, and then choose **My settings**.
   b. Under **Personal access tokens**, choose the token you created in this tutorial and choose **Delete**.

In this tutorial, you learned how to create a workflow that runs an application that pulls its dependencies from a CodeCatalyst package repository. You also learned how to block and unblock packages from entering your gateway and CodeCatalyst package repositories.

Specifying CodeCatalyst package repositories in workflows

In CodeCatalyst, you can add a CodeCatalyst package repository to your build and test actions in your workflow. Your package repository must be configured with a package format, such as npm. You can also choose to include a sequence of scopes for your selected package repository.

Use the following instructions to specify a package configuration to use with a workflow action.

Visual

To specify the package configuration that an action will use (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.

3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the **Build** or **Test** action with which you want to configure with a package repository.
8. Choose **Packages**.
9. From the **Add configuration** dropdown menu, choose the package configuration you want to use with your workflow actions.
10. Choose **Add package repository**.
11. In the **Package repository** dropdown menu, specify the name of your CodeCatalyst *package repository* that you want the action to use.

    For more information about package repositories, see Package repositories.
12. (Optional) In **Scopes - optional**, specify a sequence of *scopes* that you want to define in your package registry.

    When defining scopes, the specified package repository is configured as the registry for all listed scopes. If a package with the scope is requested through the npm client, it will use that repository instead of the default. Each scope name must be prefixed with "@".

    If `Scopes` is omitted, then the specified package repository is configured as the default registry for all packages used by the action.

    For more information about scopes, see Package namespaces and Scoped packages.
13. Choose **Add**.
14. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
15. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To specify the package configuration that an action will use (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.

3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In a **Build** or **Test** action, add code similar to the following:

```
action-name:
 Configuration:
    Packages:
        NpmConfiguration:
          PackageRegistries:
            - PackagesRepository: package-repository
              Scopes:
                - "@scope"
```

   For more information, see the description of the `Packages` property in Build and test action YAML definition for your action.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Using authorization tokens in workflow actions

You can use a token provided by the workflow action to manually configure a package manager to authenticate with CodeCatalyst package repositories. CodeCatalyst makes this token available as an environment variable for you to reference in your actions.

Environment variable	Value
CATALYST_MACHINE_RESOURCE_NAME	The user identity of the authorization token.
CATALYST_PACKAGES_AUTHORIZATION_TOKEN	The value of the authorization token.

> **Note**
>
> Note that these environment variables will only be populated if you have configured your action to export the authorization token.

Use the following instructions to use an authorization token with a workflow action.

Visual

To use an exported authorization token with an action (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the **Build** or **Test** action with which you want to configure with a package repository.
8. Choose **Packages**.
9. Turn on **Export authorization token**.

YAML

To use an exported authorization token with an action (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.

7. In a **Build** or **Test** action, add code similar to the following:

```
Actions:
  action-name:
    Packages:
      ExportAuthorizationToken: true
```

   You can reference the `$CATALYST_MACHINE_RESOURCE_NAME` and `$CATALYST_PACKAGES_AUTHORIZATION_TOKEN` environment variables in the `Steps` section of your YAML. For more information, refer to Example: Manually configuring `pip` to authenticate with CodeCatalyst.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Examples of specifying package repositories in workflows

The following examples show how to reference packages in the workflow definition file.

Topics

- Example: Defining packages with NpmConfiguration
- Example: Overriding the default registry
- Example: Overriding scopes in your package registry
- Example: Manually configuring pip to authenticate with CodeCatalyst

Example: Defining packages with `NpmConfiguration`

The following example shows how to define a package with `NpmConfiguration` in your workflow definition file.

```
Actions:
  Build:
  Identifier: aws/build-beta@v1
  Configuration:
    Packages:
        NpmConfiguration:
          PackageRegistries:
            - PackagesRepository: main-repo
```

```
            - PackagesRepository: scoped-repo
              Scopes:
                - "@scope1"
```

This example configures the npm client as such:

```
default: main-repo
@scope1: scoped-repo
```

In this example, there are two repositories defined. The default registry is set as `main-repo` as it is defined without a scope. Scope `@scope1` is configured in `PackageRegistries` for `scoped-repo`.

Example: Overriding the default registry

The following example shows you how to override the default registry.

```
NpmConfiguration:
  PackageRegistries:
    - PackagesRepository: my-repo-1
    - PackagesRepository: my-repo-2
    - PackagesRepository: my-repo-3
```

This example configures the npm client as such:

```
default: my-repo-3
```

If you specify multiple default repositories, the last repository will take priority. In this example, the last repository listed is `my-repo-3`, meaning that npm will connect to `my-repo-3`. This overrides the repositories `my-repo-1` and `my-repo-2`.

Example: Overriding scopes in your package registry

The following example shows you how to override a scope in your package registry.

```
NpmConfiguration:
  PackageRegistries:
    - PackagesRepository: my-default-repo
    - PackagesRepository: my-repo-1
      Scopes:
        - "@scope1"
```

```
        - "@scope2"
    - PackagesRepository: my-repo-2
      Scopes:
        - "@scope2"
```

This example configures the npm client as such:

```
default: my-default-repo
@scope1: my-repo-1
@scope2: my-repo-2
```

If you include overriding scopes, the last repository will take priority. In this example, the last time that scope `@scope2` is configured in `PackageRegistries` is for `my-repo-2`. This overrides the scope `@scope2` configured for `my-repo-1`.

Example: Manually configuring `pip` to authenticate with CodeCatalyst

The following example shows you how to reference CodeCatalyst authorization environment variables in a build action.

```
Actions:
  Build:
    Identifier: aws/build@v1.0.0
    Configuration:
      Steps:
        - Run: pip config set global.index-url https://$CATALYST_MACHINE_RESOURCE_NAME:
$CATALYST_PACKAGES_AUTHORIZATION_TOKEN@codecatalyst.aws/pypi/my-space/my-project/my-
repo/simple/
    Packages:
      ExportAuthorizationToken: true
```

Invoking an AWS Lambda function using a workflow

This section describes how to invoke a AWS Lambda function using a CodeCatalyst workflow. To accomplish this, you must add the **AWS Lambda invoke** action to your workflow. The **AWS Lambda invoke** action invokes the Lambda function that you specify.

In addition to invoking your function, the **AWS Lambda invoke** action also converts each top-level key in the response payload received from the Lambda function into a workflow output variable. These variables can then be referenced in subsequent workflow actions. If you don't want all top-

level keys to be converted to variables, you can use filters to specify the exact ones. For more information, see ResponseFilters property description in the "AWS Lambda invoke" action YAML definition.

When to use this action

Use this action if you want to add functionality to your workflow that is encapsulated in, and performed by, a Lambda function.

For example, you might want your workflow to send a `Build started` notification to a Slack channel before starting a build of your application. In this case, your workflow would include an **AWS Lambda invoke** action to invoke a Lambda to send out the Slack notification, and a build action to build your application.

As another example, you might want your workflow to conduct a vulnerability scan on your application before it is deployed. In this case, you would use a build action to build your application, an **AWS Lambda invoke** action to invoke a Lambda to scan for vulnerabilities, and a deploy action to deploy the scanned application.

Topics

Example workflow that invokes a Lambda function

The following workflow includes the **AWS Lambda invoke** action, along with a deploy action. The workflow sends out a Slack notification indicating that a deployment has started, and then deploys an application into AWS using an AWS CloudFormation template. The workflow consists of the following building blocks that run sequentially:

- A **trigger** – This trigger starts the workflow run automatically when you push a change to your source repository. For more information about triggers, see Starting a workflow run automatically with triggers.
- An **AWS Lambda invoke** action (`LambdaNotify`) – On trigger, this action invokes the `Notify-Start` Lambda function in the specified AWS account and Region (`my-aws-account`, and `us-`

`west-2`). On invocation, the Lambda function sends a Slack notification indicating a deployment has started.

- A **Deploy AWS CloudFormation stack** action (`Deploy`) – On completion of the **AWS Lambda invoke** action, the **Deploy AWS CloudFormation stack** action runs the template (`cfn-template.yml`) to deploy your application stack. For more information about the **Deploy AWS CloudFormation stack** action, see Deploying an AWS CloudFormation stack with a workflow.

> **Note**
>
> The following workflow example is for illustrative purposes, and will not work without additional configuration.

> **Note**
>
> In the YAML code that follows, you can omit the `Connections:` sections if you want. If you omit these sections, you must ensure that the role specified in the **Default IAM role** field in your environment includes the permissions and trust policies required by the **AWS Lambda invoke** and **Deploy AWS CloudFormation stack** actions. For more information about setting up an environment with a default IAM role, see Creating an environment. For more information about the permissions and trust policies required by the **AWS Lambda invoke** and **Deploy AWS CloudFormation stack** actions, see the description of the `Role` property in the "AWS Lambda invoke" action YAML definition and "Deploy AWS CloudFormation stack" action YAML definition.

```
Name: codecatalyst-lamda-invoke-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  LambdaNotify:
    Identifier: aws/lambda-invoke@v1
    Environment:
      Name: my-production-environment
```

```
      Connections:
        - Name: my-aws-account
          Role: codecatalyst-lambda-invoke-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Function: Notify-Start
      AWSRegion: us-west-2

  Deploy:
    Identifier: aws/cfn-deploy@v1
    Environment:
      Name: my-production-environment
      Connections:
        - Name: my-aws-account
          Role: codecatalyst-deploy-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      name: my-application-stack
      region: us-west-2
      role-arn: arn:aws:iam::111122223333:role/StackRole
      template: ./cfn-template.yml
      capabilities: CAPABILITY_IAM,CAPABILITY_AUTO_EXPAND
```

Adding the "AWS Lambda invoke" action

Use the following instructions to add the **AWS Lambda invoke** action to your workflow.

Prerequisite

Before you begin, make sure your AWS Lambda function and associated Lambda execution role are ready and available in AWS. For more information, see the Lambda execution role topic in the *AWS Lambda Developer Guide*.

Visual

To add the "AWS Lambda invoke" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.

3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **AWS Lambda invoke** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **AWS Lambda invoke**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **View source** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. In the **Inputs**, **Configuration**, and **Outputs** tabs, complete the fields according to your needs. For a description of each field, see the "AWS Lambda invoke" action YAML definition. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add the "AWS Lambda invoke" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.

6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **AWS Lambda invoke** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **AWS Lambda invoke**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **View source** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "AWS Lambda invoke" action YAML definition.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

Variables produced by the "AWS Lambda invoke" action

By default, the **AWS Lambda invoke** action produces one variable per top-level key in the Lambda response payload.

For example, if the response payload looks like this:

```
responsePayload = {
  "name": "Saanvi",
  "location": "Seattle",
  "department": {
    "company": "Amazon",
    "team": "AWS"
  }
}
```

...then the action would generate the following variables.

Key	Value
name	Saanvi
location	Seattle
department	{"company": "Amazon", "team": "AWS"}

> **Note**
>
> You can change which variables are generated using the `ResponseFilters` YAML property. For more information, see the ResponseFilters in the "AWS Lambda invoke" action YAML definition.

The variables produced and set by the "AWS Lambda invoke" action at run time are known as *predefined variables*.

For information about referencing these variables in a workflow, see Using predefined variables.

"AWS Lambda invoke" action YAML definition

The following is the YAML definition of the **AWS Lambda invoke** action. To learn how to use this action, see Invoking an AWS Lambda function using a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
```

```
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  LambdaInvoke_nn:
    Identifier: aws/lambda-invoke@v1
    DependsOn:
      - dependent-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - request-payload
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      Function: my-function|function-arn
      AWSRegion: us-west-2
      # Specify RequestPayload or RequestPayloadFile, but not both.
      RequestPayload: '{"firstname": "Li", lastname: "Jean", "company": "ExampleCo",
 "team": "Development"}'
      RequestPayloadFile: my/request-payload.json
      ContinueOnError: true|false
      LogType: Tail|None
      ResponseFilters: '{"name": ".name", "company": ".department.company"}'
    Outputs:
      Artifacts:
        - Name: lambda_artifacts
          Files:
            - "lambda-response.json"
```

LambdaInvoke

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `Lambda_Invoke_Action_Workflow_nn`.

Corresponding UI: Configuration tab/**Action name**

Identifier

(*LambdaInvoke*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/lambda-invoke@v1`.

Corresponding UI: Workflow diagram/LambdaInvoke_nn/**aws/lambda-invoke@v1** label

DependsOn

(*LambdaInvoke*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*LambdaInvoke*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*LambdaInvoke*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/**Compute type**

Fleet

(*LambdaInvoke*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/**Compute fleet**

Timeout

(*`LambdaInvoke`*/**Timeout**)

(Required)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Inputs

(*`LambdaInvoke`*/**Inputs**)

(Required)

The `Inputs` section defines the data that the **AWS Lambda invoke** action needs during a workflow run.

> **Note**
>
> Only one input (either a source or an artifact) is allowed per **AWS Lambda invoke** action. Variables do not count towards this total.

Corresponding UI: **Inputs** tab

Sources

(*`LambdaInvoke`*/Inputs/**Sources**)

(Required if RequestPayloadFile is provided)

If you want to pass a request payload JSON file to the **AWS Lambda invoke** action, and this payload file is stored in a source repository, specify the label of that source repository. Currently, the only supported label is `WorkflowSource`.

If your request payload file is not contained within a source repository, it must reside in an artifact generated by another action.

For more information about the payload file, see RequestPayloadFile.

> **Note**
>
> Instead of specifying a payload file, you can add the payload's JSON code directly to the action using the `RequestPayload` property. For more information, see RequestPayload.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*`LambdaInvoke`*/Inputs/**Artifacts**)

(Required if RequestPayloadFile is provided)

If you want to pass a request payload JSON file to the **AWS Lambda invoke** action, and this payload file is contained in an output artifact from a previous action, specify that artifact here.

For more information about the payload file, see RequestPayloadFile.

> **Note**
>
> Instead of specifying a payload file, you can add the payload's JSON code directly to the action using the `RequestPayload` property. For more information, see RequestPayload.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Configuration tab/**Artifacts - optional**

Variables - input

(*LambdaInvoke*/Inputs/**Variables**)

(Optional)

Specify a sequence of name/value pairs that define the input variables that you want to make available to the action. Variable names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in variable names.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Inputs tab/**Variables - optional**

Environment

(*LambdaInvoke*/**Environment**)

(Required)

Specify the CodeCatalyst environment to use with the action. The action connects to the AWS account and optional Amazon VPC specified in the chosen environment. The action uses the default IAM role specified in the environment to connect to the AWS account, and uses the IAM role specified in the Amazon VPC connection to connect to the Amazon VPC.

> **Note**
>
> If the default IAM role does not have the permissions required by the action, you can configure the action to use a different role. For more information, see Assigning a different IAM role to an action.

For more information about environments, see Deploying into AWS accounts and VPCs with CodeCatalyst environments and Creating an environment.

Corresponding UI: Configuration tab/**Environment**

Name

(*LambdaInvoke*/Environment/**Name**)

(Required if Environment is included)

Specify the name of an existing environment that you want to associate with the action.

Corresponding UI: Configuration tab/**Environment**

Connections

(*LambdaInvoke*/Environment/**Connections**)

(Optional in newer versions of the action; required in older versions)

Specify the account connection to associate with the action. You can specify a maximum of one account connection under `Environment`.

If you do not specify an account connection:

- The action uses the AWS account connection and default IAM role specified in the environment in the CodeCatalyst console. For information about adding an account connection and default IAM role to environment, see Creating an environment.
- The default IAM role must include the policies and permissions required by the action. To determine what those policies and permissions are, see the description of the **Role** property in the action's YAML definition documentation.

For more information about account connections, see Allowing access to AWS resources with connected AWS accounts. For information about adding an account connection to an environment, see Creating an environment.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Name

(*LambdaInvoke*/Environment/Connections/**Name**)

(Required if Connections is included)

Specify the name of the account connection.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *my-environment*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**AWS account connection**

Role

(*LambdaInvoke*/Environment/Connections/**Role**)

(Required if Connections is included)

Specify the name of the IAM role that the **AWS Lambda invoke** action uses to access AWS and invoke your Lambda function. Make sure that you have added the role to your CodeCatalyst space, and that the role includes the following policies.

If you do not specify an IAM role, then the action uses the default IAM role listed in the environment in the CodeCatalyst console. If you use the default role in the environment, make sure it has the following policies.

- The following permissions policy:

> **Warning**
>
> Limit the permissions to those shown in the following policy. Using a role with broader permissions might pose a security risk.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:aws-region:aws-account:function:function-
name"
        }
    ]
}
```

- The following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> You can use the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role with this action, if you'd like. For more information about this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. Understand that the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role has full access permissions which may pose a security risk. We recommend that you only use this role in tutorials and scenarios where security is less of a concern.

Corresponding UI: One of the following depending on the action version:

- (Newer versions) Configuration tab/Environment/What's in *`my-environment`*?/three dot menu/**Switch role**
- (Older versions) Configuration tab/'Environment/account/role'/**Role**

Configuration

(*`LambdaInvoke`*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

Function

(*LambdaInvoke*/Configuration/**Function**)

(Required)

Specify the AWS Lambda function that this action will invoke. You can specify the name of the function, or its Amazon Resource Name (ARN). You can find the name or ARN in the Lambda console.

> **Note**
>
> The AWS account where the Lambda function resides can be different from the account specified under `Connections:`.

Corresponding UI: Configuration tab/**Function**

AWSRegion

(*LambdaInvoke*/Configuration/**AWSRegion**)

(Required)

Specify the AWS Region where your AWS Lambda function resides. For a list of Region codes, see Regional endpoints in the *AWS General Reference*.

Corresponding UI: Configuration tab/**Destination bucket - optional**

RequestPayload

(*LambdaInvoke*/Configuration/**RequestPayload**)

(Optional)

If you want to pass a request payload to the **AWS Lambda invoke** action, specify the request payload here, in JSON format.

Example request payload:

```
'{ "key": "value" }'
```

If you do not want to pass a request payload to your Lambda function, then omit this property.

> **Note**
>
> You can specify either `RequestPayload` or `RequestPayloadFile`, but not both.

For more information about the request payload, see the Invoke topic in the *AWS Lambda API Reference*.

Corresponding UI: Configuration tab/**Request payload - optional**

RequestPayloadFile

(*LambdaInvoke*/Configuration/**RequestPayloadFile**)

(Optional)

If you want to pass a request payload to the **AWS Lambda invoke** action, specify the path to this request payload file here. The file must be in JSON format.

The request payload file can reside in a source repository or an artifact from a previous action. The file path is relative to the source repository or artifact root.

If you do not want to pass a request payload to your Lambda function, then omit this property.

> **Note**
>
> You can specify either `RequestPayload` or `RequestPayloadFile`, but not both.

For more information about the request payload file, see the Invoke topic in the *AWS Lambda API Reference*.

Corresponding UI: Configuration tab/**Request payload file - optional**

ContinueOnError

(*LambdaInvoke*/Configuration/**RequestPayloadFile**)

(Optional)

Specify whether you want to mark the **AWS Lambda invoke** action as succeeded even if the invoked AWS Lambda function fails. Consider setting this property to `true` to allow subsequent actions in your workflow to start despite the Lambda failure.

The default is to fail the action if the Lambda function fails ("off" in the visual editor or `false` in the YAML editor).

Corresponding UI: Configuration tab/**Continue on error**

LogType

(*LambdaInvoke*/Configuration/**LogType**)

(Optional)

Specify whether you want to include error logs in the response from the Lambda function after it is invoked. You can view these logs in the **Lambda invoke** action's **Logs** tab in the CodeCatalyst console. Possible values are:

- `Tail` – return logs
- None – do not return logs

The default is **Tail**.

For more information about the log type, see the Invoke topic in the *AWS Lambda API Reference*.

For more information about viewing logs, see Viewing workflow run status and details.

Corresponding UI: Configuration tab/**Log type**

ResponseFilters

(*LambdaInvoke*/Configuration/**ResponseFilters**)

(Optional)

Specify which keys in the Lambda response payload you want to convert to output variables. You can then reference the output variables in subsequent actions in your workflow. For more information about variables in CodeCatalyst, see Configuring and using variables in a workflow.

For example, if your response payload looks like this:

```
responsePayload = {
  "name": "Saanvi",
  "location": "Seattle",
  "department": {
    "company": "Amazon",
    "team": "AWS"
  }
}
```

...and your response filters look like this:

```
Configuration:
  ...
  ResponseFilters: '{"name": ".name", "company": ".department.company"}'
```

...then the action generates the following output variables:

Key	Value
name	Saanvi
company	Amazon

You can then reference the `name` and `company` variables in subsequent actions.

If you do not specify any keys in `ResponseFilters`, then the action converts each top-level key in the Lambda response into an output variable. For more information, see Variables produced by the "AWS Lambda invoke" action .

Consider using response filters to limit the generated output variables to only those you actually want to use.

Corresponding UI: Configuration tab/**Response filters - optional**

Outputs

(*LambdaInvoke*/**Outputs**)

(Optional)

Defines the data that is output by the action during a workflow run.

Corresponding UI: **Outputs** tab

Artifacts

(*LambdaInvoke*/Outputs/**Artifacts**)

(Optional)

Specify the artifacts generated by the action. You can reference these artifacts as input in other actions.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/Artifacts/**Build artifact name**

Name

(*LambdaInvoke*/Outputs/Artifacts/**Name**)

(Optional)

Specify the name of the artifact that will contain the Lambda response payload that is returned by the Lambda function. The default value is `lambda_artifacts`. If you do not specify an artifact, then the Lambda response payload can be viewed in the action's logs, which are available on the **Logs** tab for the action in the CodeCatalyst console. For more information about viewing logs, see Viewing workflow run status and details.

Corresponding UI: Outputs tab/Artifacts/**Build artifact name**

Files

(*LambdaInvoke*/Outputs/Artifacts/**Files**)

(Optional)

Specify the files to include in the artifact. You must specify `lambda-response.json` so that the Lambda response payload file will be included.

Corresponding UI: Outputs tab/Artifacts/**Files produced by build**

Modifying an Amazon ECS task definition file using a workflow

This section describes how to update the `image` field in an Amazon Elastic Container Service (Amazon ECS) task definition file using a CodeCatalyst workflow. To accomplish this, you must add the **Render Amazon ECS task definition** action to your workflow. This action updates the image field in the task definition file with a Docker image name that is supplied by your workflow at runtime.

> **Note**
>
> You can also use this action to update the task definition's `environment` field with environment variables.

When to use this action

Use this if you have a workflow that builds and tags a Docker image with dynamic content, such as a commit ID or timestamp.

Do not use this action if your task definition file contains an image value that always stays the same. In this case, you can manually enter the name of your image into the task definition file.

How the "Render Amazon ECS task definition" action works

You must use the **Render Amazon ECS task definition** action with the **build** and **Deploy to Amazon ECS** actions in your workflow. Together, these actions work as follows:

1. The **build** action builds your Docker image and tags it with a name, a commit ID, timestamp, or other dynamic content. For example, your build action might look like this:

```
MyECSWorkflow
  Actions:
```

```
    BuildAction:
      Identifier: aws/build@v1
      ...
      Configuration:
        Steps:
        # Build, tag, and push the Docker image...
          - Run: docker build -t MyDockerImage:${WorkflowSource.CommitId} .
          ...
```

In the preceding code, the `docker build -t` directive indicates to build the Docker image and tag it with the commit ID at action runtime. The generated image name might look like this:

`MyDockerImage:a37bd7e`

2. The **Render Amazon ECS task definition** action adds the dynamically generated image name, `MyDockerImage:a37bd7e`, to your task definition file, like this:

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-
execution-role",
    "containerDefinitions": [
        {
            "name": "codecatalyst-ecs-container",
            "image":  MyDockerImage:a37bd7e,
            "essential": true,
            ...
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
...
}
```

Optionally, you can also have the **Render Amazon ECS task definition** action add environment variables to the task definition, like this:

```
{
```

```
  "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-execution-
role",
  "containerDefinitions": [
    {
      "name": "codecatalyst-ecs-container",
      "image":  MyDockerImage:a37bd7e,
      ...
      "environment": [
        {
          name": "ECS_LOGLEVEL",
          value": "info"
        }
      ]
    }
  ],
...
}
```

For more information about environment variables, see Specifying environment variables in the *Amazon Elastic Container Service Developer Guide*.

3. The **Deploy to Amazon ECS** action registers the updated task definition file with Amazon ECS. Registering the updated task definition file deploys the new image, `MyDockerImage:a37bd7e` into Amazon ECS.

Topics

Example workflow that modifies an Amazon ECS task definition file

The following is an example of a full workflow that includes the **Render Amazon ECS task definition** action, along with build and deploy actions. The workflow's purpose is to build and deploy a Docker image into your Amazon ECS cluster. The workflow consists of the following building blocks that run sequentially:

- A **trigger** – This trigger starts the workflow run automatically when you push a change to your source repository. For more information about triggers, see Starting a workflow run automatically with triggers.
- A **build** action (`BuildDocker`) – On trigger, the action builds the Docker image using the Dockerfile, tags it with a commit ID, and pushes the image to Amazon ECR. For more information about the build action, see Building with workflows.
- A **Render Amazon ECS task definition** action (`RenderTaskDef`) – On completion of the build action, this action updates an existing `taskdef.json` located in the root of your source repository with an `image` field value that includes the correct commit ID. It saves the updated file with a new file name (`task-definition-`*`random-string`*`.json`) and then creates an output artifact that contains this file. The render action also generates a variable called `task-definition` and sets it to the name of the new task definition file. The artifact and variable will be used the deploy action, which is next.
- A **Deploy to Amazon ECS** action (`DeployToECS`) – On completion of the **Render Amazon ECS task definition** action, the **Deploy to Amazon ECS** action looks for the output artifact generated by the render action (`TaskDefArtifact`), finds the `task-definition-`*`random-string`*`.json` file inside of it, and registers it with your Amazon ECS service. The Amazon ECS service then follows the instructions in the `task-definition-`*`random-string`*`.json` file to run Amazon ECS tasks—and associated Docker image containers—inside your Amazon ECS cluster.

```
Name: codecatalyst-ecs-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildDocker:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-build-role
    Inputs:
      Variables:
```

```
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-
repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      Steps:
        #pre_build:
        - Run: echo Logging in to Amazon ECR...
        - Run: aws --version
        - Run: aws ecr get-login-password --region us-east-2 | docker login --username
 AWS --password-stdin 111122223333.dkr.ecr.us-east-2.amazonaws.com
        #build:
        - Run: echo Build started on `date`
        - Run: echo Building the Docker image...
        - Run: docker build -t $REPOSITORY_URI:latest .
        - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
        #post_build:
        - Run: echo Build completed on `date`
        - Run: echo Pushing the Docker images...
        - Run: docker push $REPOSITORY_URI:latest
        - Run: docker push $REPOSITORY_URI:$IMAGE_TAG

  RenderTaskDef:
    DependsOn:
      - BuildDocker
    Identifier: aws/ecs-render-task-definition@v1
    Inputs:
      Variables:
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-
repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      task-definition: taskdef.json
      container-definition-name: codecatalyst-ecs-container
      image: $REPOSITORY_URI:$IMAGE_TAG
    # The output artifact contains the updated task definition file.
    # The new file is prefixed with 'task-definition'.
    # The output variable is set to the name of the updated task definition file.
    Outputs:
      Artifacts:
        - Name: TaskDefArtifact
```

```
        Files:
          - "task-definition*"
    Variables:
      - task-definition

DeployToECS:
  Identifier: aws/ecs-deploy@v1
  Environment:
    Name: codecatalyst-ecs-environment
    Connections:
      - Name: codecatalyst-account-connection
        Role: codecatalyst-ecs-deploy-role
  #Input artifact contains the updated task definition file.
  Inputs:
    Sources: []
    Artifacts:
      - TaskDefArtifact
  Configuration:
    region: us-east-2
    cluster: codecatalyst-ecs-cluster
    service: codecatalyst-ecs-service
    task-definition: ${RenderTaskDef.task-definition}
```

Adding the "Render Amazon ECS task definition" action

Use the following instructions to add the **Render Amazon ECS task definition** action to your workflow.

Prerequisite

Before you begin, make sure you have a workflow that includes a build action that dynamically generates a Docker image. See the preceding example workflow for details.

Visual

To add the "Render Amazon ECS task definition" action using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

5. Choose **Edit**.
6. Choose **Visual**.
7. At the top-left, choose **+ Actions** to open the action catalog.
8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Render Amazon ECS task definition** action, and do one of the following:

   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or

   - Choose **Render Amazon ECS task definition**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **View source** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. In the **Inputs** and **Configuration** tabs, complete the fields according to your needs. For a description of each field, see the "Render Amazon ECS task definition" action YAML definition reference. This reference provides detailed information about each field (and corresponding YAML property value) as it appears in both the YAML and visual editors.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To add the "Render Amazon ECS task definition" action using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. At the top-left, choose **+ Actions** to open the action catalog.

8. From the drop-down list, choose **Amazon CodeCatalyst**.
9. Search for the **Render Amazon ECS task definition** action, and do one of the following:
   - Choose the plus sign (**+**) to add the action to the workflow diagram and open its configuration pane.

     Or
   - Choose **Render Amazon ECS task definition**. The action details dialog box appears. On this dialog box:
     - (Optional) Choose **View source** to view the action's source code.
     - Choose **Add to workflow** to add the action to the workflow diagram and open its configuration pane.
10. Modify the properties in the YAML code according to your needs. An explanation of each available property is provided in the "Render Amazon ECS task definition" action YAML definition reference.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.

Next steps

After adding the render action, add the **Deploy to Amazon ECS** action to your workflow following the instructions in Deploying an application to Amazon Elastic Container Service (ECS) with a workflow. While adding the deploy action, do the following:

1. In the **Inputs** tab of the deploy action, in **Artifacts - optional**, select the artifact that was generated by the render action. It contains the updated task definition file.

   For more information about artifacts, see Sharing data between actions in a workflow using artifacts.
2. In the **Configuration** tab of the deploy action, in the **Task definition** field, specify the following action variable: `${action-name.task-definition}` where *`action-name`* is the name of your render action, for example, `RenderTaskDef`. The render action sets this variable to the new name of the task definition file.

   For more information about variables, see Configuring and using variables in a workflow.

For more information about how to configure the deploy action, see the preceding example workflow.

Viewing the updated task definition file

You can view the name and contents of the updated task definition file.

To view the name of the updated task definition file, after the Render Amazon ECS task definition action has processed it.

1. Find the run that includes a completed render action:
    a. Open the CodeCatalyst console at https://codecatalyst.aws/.
    b. Choose your project.
    c. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
    d. Choose the name of the workflow that contains the render action. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
    e. Choose a run that includes the completed render action.
2. In the workflow diagram, choose the render action.
3. Choose **Outputs**.
4. Choose **Variables**.
5. The task definition file name is displayed. It looks similar to `task-definition--259-0a2r7gxlTF5X-.json`.

To view the contents of the updated task definition file

1. Find the run that includes a completed render action:
    a. Open the CodeCatalyst console at https://codecatalyst.aws/.
    b. Choose your project.
    c. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
    d. Choose the name of the workflow that contains the render action. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

e. Choose a run that includes the completed render action.

2. In the workflow run, at the top, next to **Visual** and **YAML**, choose **Workflow outputs**.
3. In the **Artifacts** section, choose **Download** next to the artifact that contains the updated task definition file. This artifact will have a **Produced by** column set to the name of your render action.
4. Open the .zip file to view the task definition .json file.

Variables produced by the "Render Amazon ECS task definition" action

The **Render Amazon ECS task definition** action produces and sets the following variables at run time. These are known as *predefined variables*.

For information about referencing these variables in a workflow, see Using predefined variables.

Key	Value
task-definition	The name given to the task definition file that was updated by the **Render Amazon ECS task definition** action. The name follows the format `task-definition- random-string .json`. Example: `task-definition--259-0a2r7gxlTF5Xr.json`

"Render Amazon ECS task definition" action YAML definition reference

The following is the YAML definition of the **Render Amazon ECS task definition** action. To learn how to use this action, see Modifying an Amazon ECS task definition file using a workflow.

This action definition exists as a section within a broader workflow definition file. For more information about this file, see Workflow YAML definition.

Note

Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

```
# The workflow definition starts here.
# See Top-level properties for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  ECSRenderTaskDefinition_nn:
    Identifier: aws/ecs-render-task-definition@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - task-definition-artifact
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Configuration
      task-definition: task-definition-path
      container-definition-name: container-definition-name
      image: docker-image-name
      environment-variables:
        - variable-name-1=variable-value-1
        - variable-name-2=variable-value-2
    Outputs:
```

```
      Artifacts:
        - Name: TaskDefArtifact
          Files: "task-definition*"
      Variables:
        - task-definition
```

ECSRenderTaskDefinition

(Required)

Specify the name of the action. All action names must be unique within the workflow. Action names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in action names.

Default: `ECSRenderTaskDefinition_nn`.

Corresponding UI: Configuration tab/**Action name**

Identifier

(*ECSRenderTaskDefinition*/**Identifier**)

(Required)

Identifies the action. Do not change this property unless you want to change the version. For more information, see Specifying the major, minor, or patch version of an action.

Default: `aws/ecs-render-task-definition@v1`.

Corresponding UI: Workflow diagram/ECSRenderTaskDefinition_nn/**aws/ecs-render-task-definition@v1** label

DependsOn

(*ECSRenderTaskDefinition*/**DependsOn**)

(Optional)

Specify an action, action group, or gate that must run successfully in order for this action to run.

For more information about the 'depends on' functionality, see Configuring actions to depend on other actions.

Corresponding UI: Inputs tab/**Depends on - optional**

Compute

(*ECSRenderTaskDefinition*/**Compute**)

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

Corresponding UI: *none*

Type

(*ECSRenderTaskDefinition*/Compute/**Type**)

(Required if Compute is included)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or `EC2` (YAML editor)

  Optimized for flexibility during action runs.
- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: Configuration tab/**Compute type**

Fleet

(*ECSRenderTaskDefinition*/Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

Corresponding UI: Configuration tab/**Compute fleet**

Timeout

(*`ECSRenderTaskDefinition`*/**Timeout**)

(Optional)

Specify the amount of time in minutes (YAML editor), or hours and minutes (visual editor), that the action can run before CodeCatalyst ends the action. The minimum is 5 minutes and the maximum is described in Quotas for workflows. The default timeout is the same as the maximum timeout.

Corresponding UI: Configuration tab/**Timeout - optional**

Inputs

(*`ECSRenderTaskDefinition`*/**Inputs**)

(Optional)

The `Inputs` section defines the data that the `ECSRenderTaskDefinition` needs during a workflow run.

> **Note**
>
> Only one input (either a source or an artifact) is allowed per **Render Amazon ECS task definition** action. Variables do not count towards this total.

Corresponding UI: **Inputs** tab

Sources

(*ECSRenderTaskDefinition*/Inputs/**Sources**)

(Required if your task definition file is stored in a source repository)

If your task definition file is stored in a source repository, specify the label of that source repository. Currently, the only supported label is `WorkflowSource`.

If your task definition file is not contained within a source repository, it must reside in an artifact generated by another action.

For more information about sources, see Connecting a workflow to a source repository.

Corresponding UI: Inputs tab/**Sources - optional**

Artifacts - input

(*ECSRenderTaskDefinition*/Inputs/**Artifacts**)

(Required if your task definition file is stored in an output artifact from a previous action)

If the task definition file that you want to deploy is contained in an artifact generated by a previous action, specify that artifact here. If your task definition file is not contained within an artifact, it must reside in your source repository.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Configuration tab/**Artifacts - optional**

Variables - input

(*ECSRenderTaskDefinition*/Inputs/**Variables**)

(Required)

Specify a sequence of name/value pairs that define the input variables that you want to make available to the action. Variable names are limited to alphanumeric characters (a-z, A-Z, 0-9),

hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in variable names.

For more information about variables, including examples, see Configuring and using variables in a workflow.

Corresponding UI: Inputs tab/**Variables - optional**

Configuration

(*ECSRenderTaskDefinition*/**Configuration**)

(Required)

A section where you can define the configuration properties of the action.

Corresponding UI: **Configuration** tab

task-definition

(*ECSRenderTaskDefinition*/Configuration/**task-definition**)

(Required)

Specify the path to an existing task definition file. If the file resides in your source repository, the path is relative to the source repository root folder. If your file resides in an artifact from a previous workflow action, the path is relative to the artifact root folder. For more information about task definition files, see Task definitions in the *Amazon Elastic Container Service Developer Guide*.

Corresponding UI: Configuration tab/**Task definition**

container-definition-name

(*ECSRenderTaskDefinition*/Configuration/**container-definition-name**)

(Required)

Specify the name of the container where your Docker image will run. You can find this name in the `containerDefinitions`, `name` field in your task definition file. For more information, see Name in the *Amazon Elastic Container Service Developer Guide*.

Corresponding UI: Configuration tab/**Container name**

image

(*ECSRenderTaskDefinition*/Configuration/**image**)

(Required)

Specify the name of the Docker image that you want the **Render Amazon ECS task definition** action to add to your task definition file. The action adds this name to the `containerDefinitions`, `image` field in your task definition file. If a value already exists in the `image` field, then the action overwrites it. You can include variables in the image name.

Examples:

If you specify `MyDockerImage:${WorkflowSource.CommitId}`, the action adds `MyDockerImage:`*`commit-id`* to the task definition file, where *`commit-id`* is a commit ID generated at runtime by the workflow.

If you specify `my-ecr-repo/image-repo:$(date +%m-%d-%y-%H-%m-%s)`, the action adds *`my-ecr-repo`*/image-repo:*`date +%m-%d-%y-%H-%m-%s`* to the task definition file, where *`my-ecr-repo`* is the URI of an Amazon Elastic Container Registry (ECR) and *`date +%m-%d-%y-%H-%m-%s`* is a timestamp in the format `month-day-year-hour-minute-second` generated at runtime by the workflow.

For more information about the `image` field, see Image in the *Amazon Elastic Container Service Developer Guide*. For more information about variables, see Configuring and using variables in a workflow.

Corresponding UI: Configuration tab/**Image name**

environment-variables

(*ECSRenderTaskDefinition*/Configuration/**environment-variables**)

(Required)

Specify environment variables that you want the **Render Amazon ECS task definition** action to add to your task definition file. The action adds the variables to the `containerDefinitions`, `environment` field in your task definition file. If variables already exist in the file, the action

overwrites the values of existing variables and adds any new variables. For more information about Amazon ECS environment variables, see Specifying environment variables in the *Amazon Elastic Container Service Developer Guide*.

Corresponding UI: Configuration tab/**Environment variables - optional**

Outputs

(*ECSRenderTaskDefinition*/**Outputs**)

(Required)

Defines the data that is output by the action during a workflow run.

Corresponding UI: **Outputs** tab

Artifacts

(*ECSRenderTaskDefinition*/Outputs/**Artifacts**)

(Required)

Specify the artifacts generated by the action. You can reference these artifacts as input in other actions.

For more information about artifacts, including examples, see Sharing data between actions in a workflow using artifacts.

Corresponding UI: Outputs tab/**Artifacts**

Name

(*ECSRenderTaskDefinition*/Outputs/Artifacts/**Name**)

(Required)

Specify the name of the artifact that will contain the updated task definition file. The default value is `MyTaskDefinitionArtifact`. You must then specify this artifact as input into the **Deploy to Amazon ECS** action. To understand how to add this artifact as input to the **Deploy to Amazon ECS** action, see Example workflow that modifies an Amazon ECS task definition file.

Corresponding UI: Outputs tab/Artifacts/**Name**

Files

(*ECSRenderTaskDefinition*/Outputs/Artifacts/**Files**)

(Required)

Specify the files to include in the artifact. You must specify `task-definition-*` so that the updated task definition file, which starts with `task-definition-`, will be included.

Corresponding UI: Outputs tab/Artifacts/**Files**

Variables

(*ECSRenderTaskDefinition*/Outputs/**Variables**)

(Required)

Specify the name of a variable to be set by the render action. The render action will set this variable's value to the name of the updated task definition file (for example, `task-definition-`*`random-string`*`.json`). You must then specify this variable in the **Deploy to Amazon ECS** action's **Task definition** (visual editor) or `task-definition` (yaml editor) property. To understand how to add this variable to the **Deploy to Amazon ECS** action, see Example workflow that modifies an Amazon ECS task definition file .

Default: `task-definition`

Corresponding UI: Outputs tab/Variables/**Name** field

Configuring and using variables in a workflow

A *variable* is a key-value pair that contains information that you can reference in your CodeCatalyst workflow.

There are two types of variable that you can use in a workflow:

- **User-defined variables** – These are key-value pairs that you define.
- **Predefined variables** – These are key-value pairs that are emitted by a workflow automatically. There is no need for you to define them.

> **Note**
>
> CodeCatalyst also supports GitHub output parameters, which behave like variables and can be referenced in other actions. For more information, see Exporting a GitHub output parameter so that other actions can use it and Referencing a GitHub output parameter

Topics

Using user-defined variables

User-defined variables are key-value pairs that you define. There are two types:

- **Plaintext variables**, or simply **variables** – These are key-value pairs that you define in plaintext within the workflow definition file.
- **Secrets** – These are key-value pairs that you define on a separate **Secrets** page of the Amazon CodeCatalyst console. The *key* (name) is a public label, and the *value* contains the information you want to keep private. You only specify the key in the workflow definition file. Use secrets in place of passwords and other sensitive information in the workflow definition file.

> **Note**
>
> For brevity, this guide uses the term *variable* to mean *plaintext variable*.

Topics

- Examples of user-defined variables

Defining a variable

You can define variables in two ways:

- In the `Inputs` section of a workflow action – see To define a variable in the "Inputs" section
- In the `Steps` section of a workflow action – see To define a variable in the "Steps" section

> **Note**
>
> The `Steps` method only works with the CodeCatalyst build, test, and **GitHub Actions** actions, because these are the only actions that include a `Steps` section.

For examples, see Examples of user-defined variables.

Visual

To define a variable in the "Inputs" section (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action where you want to set the variable.
8. Choose **Inputs**.
9. In **Variables - optional**, choose **Add variable**, and then do the following:

   Specify a sequence of name/value pairs that define the input variables that you want to make available to the action. Variable names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in variable names.

For more information about variables, including examples, see Configuring and using variables in a workflow.

10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To define a variable in the "Inputs" section (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In a workflow action, add code similar to the following:

```
action-name:
  Inputs:
    Variables:
      - Name: variable-name
        Value: variable-value
```

For more examples, see Examples of user-defined variables. For more information, see the Workflow YAML definition for your action.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Visual

To define a variable in the "Steps" section (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.

3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action where you want to set the variable.
8. Choose **Configuration**.
9. In **Shell commands** or **GitHub Actions YAML**, whichever is available, define a variable in the action's `Steps`, either explicitly or implicitly.

   - To define the variable explicitly, include it in a bash command directly to the `Steps` section.
   - To define a variable implicitly, specify it in a file that's referenced in the action's `Steps` section.

   For examples, see Examples of user-defined variables. For more information, see the Workflow YAML definition for the action.
10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To define a variable in the "Steps" section (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In a workflow action, define a variable in the action's `Steps` section, either explicitly or implicitly.

- To define the variable explicitly, include it in a bash command directly to the `Steps` section.
- To define a variable implicitly, specify it in a file that's referenced in the action's `Steps` section.

  For examples, see Examples of user-defined variables. For more information, see the Workflow YAML definition for the action.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Defining a secret

You define a secret on the **Secrets** page of the CodeCatalyst console. For more information, see Configuring and using secrets in a workflow.

For example, you might define a secret that looks like this:

- Name (key): **`my-password`**
- Value: **`^*H3#!b9`**

After the secret is defined, you can specify the secret's key (**`my-password`**) in the workflow definition file. For an example of how to do this, see Example: Referencing a secret.

Exporting a variable so that other actions can use it

Use the following instructions to export a variable from an action so that you can reference it in other actions.

Before you export a variable, note the following:

- If you only need to reference the variable within the action where it's defined, then you don't need to export it.
- Not all actions support exporting variables. To determine whether your action supports this feature, run through the visual editor instructions that follow, and see if the action includes a **Variables** button on the **Outputs** tab. If yes, exporting variables is supported.
- To export a variable from a GitHub Action, see Exporting a GitHub output parameter so that other actions can use it.

Prerequisite

Make sure you have defined the variable you want to export. For more information, see Defining a variable.

Visual

To export a variable (visual editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose the action that you want to export the variable from.
8. Choose **Outputs**.
9. In **Variables - optional**, choose **Add variable**, and then do the following:

   Specify the name of a variable that you want the action to export. This variable must already be defined in the `Inputs` or `Steps` section of the same action.
10. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
11. Choose **Commit**, enter a commit message, and choose **Commit** again.

YAML

To export a variable (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.

7. In the action that you want to export the variable from, add code similar to the following:

```
action-name:
  Outputs:
    Variables:
      - Name: variable-name
```

   For more examples, see Examples of user-defined variables.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Referencing a variable in the action that defines it

Use the following instructions to reference a variable in the action that defines it.

> **Note**
>
> To reference a variable generated by a GitHub Action, see Referencing a GitHub output parameter.

Prerequisite

Make sure you have defined the variable you want to reference. For more information, see Defining a variable.

Visual

Not available. Choose YAML to view the YAML instructions.

YAML

To reference a variable in the action that defines it

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

5. Choose **Edit**.
6. Choose **YAML**.
7. In the CodeCatalyst action that defines the variable that you want to refer to, add the variable using the following bash syntax:

   ```
   $variable-name
   ```

   For example:

   ```
   MyAction:
       Configuration:
         Steps:
           - Run: $variable-name
   ```

   For more examples, see Examples of user-defined variables. For more information, see the reference information for your action in the Workflow YAML definition.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Referencing a variable output by another action

Use the following instructions to reference variables output by other actions.

> **Note**
>
> To reference a variable output from a GitHub Action, see Referencing a GitHub output parameter.

Prerequisite

Make sure you have exported the variable you want to reference. For more information, see Exporting a variable so that other actions can use it.

Visual

Not available. Choose YAML to view the YAML instructions.

YAML

To reference a variable output by another action (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In the CodeCatalyst action, add a reference to the variable using the following syntax:

```
${action-group-name.action-name.variable-name}
```

Replace:

- *action-group-name* with the name of the action group that contains the action that outputs variable.

> **Note**
>
> You can omit *action-group-name* if there is no action group, or if the variable is produced by an action in the same action group.

- *action-name* with the name of the action that outputs the variable.
- *variable-name* with the name of the variable.

For example:

```
MySecondAction:
    Configuration:
      Steps:
        - Run: ${MyFirstAction.TIMESTAMP}
```

For more examples, see Examples of user-defined variables. For more information, see the Workflow YAML definition for your action.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Referencing a secret

For instructions on referencing a secret in the workflow definition file, see Using a secret.

For an example, see Example: Referencing a secret.

Examples of user-defined variables

The following examples show how to define and reference variables in the workflow definition file.

Examples

- Example: Defining a variable using the Inputs property
- Example: Defining a variable using the Steps property
- Example: Exporting a variable using the Outputs property
- Example: Referencing a variable defined in the same action
- Example: Referencing a variable defined in another action
- Example: Referencing a secret

Example: Defining a variable using the Inputs property

The following example shows you how to define two variables, `VAR1` and `VAR2`, in an `Inputs` section.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Variables:
      - Name: VAR1
        Value: "My variable 1"
      - Name: VAR2
        Value: "My variable 2"
```

Example: Defining a variable using the Steps property

The following example shows you how to define a `DATE` variable in the `Steps` section explicitly.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: DATE=$(date +%m-%d-%y)
```

Example: Exporting a variable using the Outputs property

The following example shows you how to define two variables, `REPOSITORY-URI` and `TIMESTAMP`, and export them using the `Outputs` section.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Variables:
        - Name: REPOSITORY-URI
          Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-
repo
    Configuration:
      Steps:
        - Run: TIMESTAMP=$(date +%m-%d-%y-%H-%m-%s)
    Outputs:
      Variables:
        - REPOSITORY-URI
        - TIMESTAMP
```

Example: Referencing a variable defined in the same action

The following example shows you how to specify a `VAR1` variable in `MyBuildAction`, and then reference it in the same action using `$VAR1`.

```
Actions:
  MyBuildAction:
    Identifier: aws/build@v1
    Inputs:
      Variables:
        - Name: VAR1
          Value: my-value
    Configuration:
```

```
      Steps:
        - Run: $VAR1
```

Example: Referencing a variable defined in another action

The following example shows you how to specify a `TIMESTAMP` variable in `BuildActionA`, export it using the `Outputs` property, and then reference it in `BuildActionB` using `${BuildActionA.TIMESTAMP}`.

```
Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: TIMESTAMP=$(date +%m-%d-%y-%H-%m-%s)
    Outputs:
      Variables:
        - TIMESTAMP
  BuildActionB:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: docker build -t my-ecr-repo/image-repo:latest .
        - Run: docker tag my-ecr-repo/image-repo:${BuildActionA.TIMESTAMP}

        # Specifying just '$TIMESTAMP' here will not work
        # because TIMESTAMP is not a variable
        # in the BuildActionB action.
```

Example: Referencing a secret

The following example shows you how to reference a `my-password` secret. The `my-password` is the secret's key. This secret's key and corresponding password value must be specified on the **Secrets** page of the CodeCatalyst console prior to being used in the workflow definition file. For more information, see Configuring and using secrets in a workflow.

```
Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Configuration:
      Steps:
```

```
        - Run: curl -u LiJuan:${Secrets.my-password} https://example.com
```

Using predefined variables

Predefined variables are key-value pairs that are emitted by a workflow automatically, and made available for you to use in workflow actions.

You can use predefined variables in any workflow action.

Topics

Referencing a predefined variable

Use the following instructions to reference a predefined variable.

Prerequisite

Determine the name of the predefined variable you want to reference, such as `CommitId`. For more information, see Determining which predefined variables your workflow emits.

Visual

Not available. Choose YAML to view the YAML instructions.

YAML

To reference a predefined variable (YAML editor)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.

6. Choose **YAML**.
7. In a CodeCatalyst action, add the predefined variable reference using the following syntax:

```
${action-group-name.action-name-or-WorkflowSource.variable-name}
```

Replace:

- *action-group-name* with the name of the action group.

  > **Note**
  >
  > You can omit *action-group-name* if there is no action group, or if the variable is produced by an action in the same action group.

- *action-name-or-WorkflowSource* with:

  The name of the action that outputs the variable.

  or

  `WorkflowSource`, if the variable is the `BranchName` or `CommitId` variable.

- *variable-name* with the name of the variable.

For example:

```
MySecondAction:
    Configuration:
      Steps:
        - Run: echo ${MyFirstECSAction.cluster}
```

Another example:

```
MySecondAction:
    Configuration:
      Steps:
        - Run: echo ${WorkflowSource.CommitId}
```

For more examples, see Examples of predefined variables. For more information, see the Workflow YAML definition for your action.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

Determining which predefined variables your workflow emits

You can determine which predefined variables your workflow emits in two ways:

- **Run the workflow once**. After the run finishes, the variables emitted by the workflow are displayed on the **Variables** tab of the run details page. For more information, see Viewing workflow run status and details.
- **Consult the List of predefined variables**. This reference lists the variable name (key) and value for each predefined variable.

> **Note**
>
> The maximum total size of a workflow's variables is listed in Quotas for workflows. If the total size exceeds the maximum, the action that occurs after the maximum is reached may fail.

Examples of predefined variables

The following examples show how to reference predefined variables in the workflow definition file.

Examples

- Example: Referencing the "CommitId" predefined variable
- Example: Referencing the "BranchName" predefined variable

Example: Referencing the "CommitId" predefined variable

The following example shows you how to refer to the `CommitId` predefined variable in the `MyBuildAction` action. The `CommitId` variable is output automatically by CodeCatalyst.

Although the example shows the variable being used in the build action, you can use `CommitId` in any action.

```
MyBuildAction:
```

```
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
      #Build Docker image and tag it with a commit ID
        - Run: docker build -t image-repo/my-docker-image:latest .
        - Run: docker tag image-repo/my-docker-image:${WorkflowSource.CommitId}
```

Example: Referencing the "BranchName" predefined variable

The following example shows you how to refer to the `BranchName` predefined variable in the `CDKDeploy` action. The `BranchName` variable is output automatically by CodeCatalyst.

Although the example shows the variable being used in the **AWS CDK deploy** action, you can use `BranchName` in any action.

```
CDKDeploy:
    Identifier: aws/cdk-deploy@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      StackName: app-stack-${WorkflowSource.BranchName}
```

List of predefined variables

Consult the following sections to view the predefined variables produced automatically by CodeCatalyst actions.

> **Note**
>
> This list only includes predefined variables emitted by the CodeCatalyst source and CodeCatalyst actions. If you're using other types of actions, such as GitHub Actions or CodeCatalyst Labs actions, see instead Determining which predefined variables your workflow emits.

List

> **Note**
>
> Not all CodeCatalyst actions produce predefined variables. If the action is not in the list, then it does not produce variables.

- Variables produced by the source ("BranchName" and "CommidId")
- Variables produced by the "Deploy AWS CloudFormation stack" action
- Variables produced by the "Deploy to Amazon ECS" action
- Variables produced by the "Deploy to Kubernetes cluster" action
- Variables produced by the "AWS CDK deploy" action
- Variables produced by the "AWS CDK bootstrap" action
- Variables produced by the "AWS Lambda invoke" action
- Variables produced by the "Render Amazon ECS task definition" action

Configuring and using secrets in a workflow

There may be times when you need to use sensitive data, such as authentication credentials, in your workflows. Storing these values in plaintext anywhere in your repository should be avoided because anyone with access to the repository which contains the secret can see them. Similarly, these values shouldn't be used directly in any workflow definitions because they will be visible as files in your repository. With CodeCatalyst, you can protect these values by adding a secret to your project, and then referencing the secret in your workflow definition file. Note that you can have a maximum of five secrets per action.

> **Note**
>
> Secrets can only be used to replace passwords and sensitive information in the workflow definition file.

Topics

- Creating a secret
- Editing a secret
- Using a secret

- Deleting a secret

Creating a secret

Use the following procedure to create a secret. The secret contains the sensitive information that you want to hide from view.

> **Note**
>
> Secrets are visible to actions and are not masked when written to a file.

To create a secret

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Secrets**.
3. Choose **Create secret**.
4. Enter the following information:

   Name

   Enter a name for your secret.

   Value

   Enter the value for the secret. This is the sensitive information that you want to hide from view. By default, the value is not displayed. To display the value, choose **Show value**.

   Description

   (Optional) Enter a description for your secret.
5. Choose **Create**.

Editing a secret

Use the following procedure to edit a secret.

To edit a secret

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. In the navigation pane, choose **CI/CD**, and then choose **Secrets**.
3. In the secrets list, choose the secret that you want to edit.
4. Choose **Edit**.
5. Edit the following properties:

   Value

   Enter the value for the secret. This is the value that you want to hide from view. By default, the value is not displayed.

   Description

   (Optional) Enter a description for your secret.
6. Choose **Save**.

Using a secret

To use a secret in a workflow action, you must obtain the reference identifier of the secret and use that identifier in the workflow action.

Topics

- Obtaining the identifier of a secret
- Referencing a secret in a workflow

Obtaining the identifier of a secret

Use the following procedure to obtain the reference identifier of the secret. You'll add this identifier to your workflow.

To obtain the reference identifier of the secret

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Secrets**.
3. In the list of secrets, find the secret that you want to use.
4. In the **Reference ID** column, copy the identifier of the secret. The following is the syntax for the **Reference ID**:

```
${Secrets.<name>}
```

Referencing a secret in a workflow

Use the following procedure to reference a secret in a workflow.

To reference a secret

1. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
2. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
3. Choose **Edit**.
4. Choose **YAML**.
5. Modify the YAML to use the identifier of the secret. For example, to use a user name and password that are stored as secrets with the `curl` command, you would use a Run command similar to the following:

```
- Run: curl -u <username-secret-identifier>:<password-secret-identifier> https://
example.com
```

6. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
7. Choose **Commit**, enter a commit message, and choose **Commit** again.

Deleting a secret

Use the following procedure to delete a secret and the secret reference identifier.

> **Note**
>
> Before deleting a secret, we recommend that you remove the secret's reference identifier from all workflow actions. If you delete the secret without deleting the reference identifier, the action will fail the next time it runs.

To delete a secret's reference identifier from a workflow

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
3. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
4. Choose **Edit**.
5. Choose **YAML**.
6. Search the workflow for the following string:

   ```
   ${Secrets.
   ```

   This finds all reference identifiers of all secrets.
7. Delete the reference identifier of the chosen secret, or replace it with a plaintext value.
8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.

To delete a secret

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **CI/CD**, and then choose **Secrets**.
3. In the secrets list, choose the secret you want to delete.
4. Choose **Delete**.
5. Enter **`delete`** to confirm the deletion.
6. Choose **Delete**.

Viewing the workflow status

You might want to view the status of a workflow to see if there are any workflow configuration issues you need to address, or to troubleshoot runs that fail to start. CodeCatalyst evaluates the workflow status every time you create or update the workflow's underlying workflow definition file.

> **Note**
>
> You can also view the workflow's *run* status, which is different from the workflow status. For more information, see Viewing workflow run status and details.

For a list of possible workflow states, see Workflow states.

To view the status of a workflow

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Find the workflow whose status you want to view. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

   The status is displayed with the workflow in the list.
5. (Optional) Choose the name of the workflow, and find the **Workflow definition** field. It shows the workflow status.

Quotas for workflows

The following table describes quotas and limits for workflows in Amazon CodeCatalyst.

For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Maximum number of workflows per space	800
Maximum workflow definition file size	256 KB
Maximum number of workflow files processed in a single source event	50
Maximum number of files processed in a single source event	4,000
Maximum number of active fleets per space	10

Maximum number of active compute instances per fleet	20
Maximum number of input artifacts per action	10
Maximum number of output artifacts per action	10
Maximum total size of a single action's output variables	120 KB
Maximum length of an output variable value	500 characters or more, depending on the action that emits the value. **Note** Values may be truncated if they exceed the action's limit.
Maximum number of days to keep artifacts generated during a workflow run	30
Maximum number of reports per action	50
Maximum number of test cases per test report	20,000
Maximum number of files per code coverage report	20,000
Maximum number of software composition analysis findings per report	20,000
Maximum number of files per static analysis report	20,000
Maximum number of concurrent workflow runs per space	100
Maximum number of actions per workflow	50

Maximum number of actions running concurrently per workflow	50
Maximum number of actions running concurrently per space	200
Maximum amount of time an action can run	For the build and test actions, the timeout is 8 hours. For all other actions, the timeout is 1 hour.
Maximum number of environments associated with an AWS account per space	5,000
Maximum number of secrets per action	5
Maximum number of secrets per space	500,000

Workflow run states

A workflow run can be in one of the following states:

- **Succeeded** – The workflow run was processed successfully.
- **Failed** – One or more actions in the workflow run failed.
- **In progress** – The workflow run is currently being processed.
- **Stopped** – A person stopped the workflow run while it was in progress.
- **Stopping** – The workflow run is currently being stopped.
- **Cancelled** – The workflow run was canceled by CodeCatalyst because the associated workflow was deleted or updated while the run was in progress.
- **Superseded** – Only occurs if you have configured superseded run mode. The workflow run was canceled by CodeCatalyst because a later workflow run superseded it.

Workflow states

A workflow can have one of the following states:

- **Valid** – The workflow is runnable and can be activated by triggers.

  For a workflow to be marked as valid, both of the following conditions must be true:

  - The workflow definition file must be valid.
  - The workflow must have no triggers, no push triggers, or a push trigger that runs using the files on the current branch. For more information, see Trigger considerations when branching.

- **Not valid** – The workflow's definition file is not valid. The workflow cannot be run manually, or automatically through triggers. Workflows that are not valid appear with a **Workflow definition has *n* errors** message (or similar) in the CodeCatalyst console.

  For a workflow to be marked as not valid, the following condition must be true:

  - The workflow definition file must be misconfigured.

    To fix a misconfigured workflow definition file, see How do I fix "Workflow definition has *n* errors" errors?.

- **Inactive** – The workflow definition is valid but cannot be run manually, or automatically through triggers.

  For a workflow to be marked as inactive, both of the following conditions must be true:

  - The workflow definition file must be valid.
  - The workflow definition file must include a push trigger that specifies a branch that is different from the one that the workflow definition file is currently on. For more information, see Trigger considerations when branching.

    To switch a workflow from **Inactive** to **Active**, see How do I fix "Workflow is inactive" messages?.

> **Note**
>
> If the workflow specifies a resource that you later remove (for example, a package repository), CodeCatalyst won't detect this change and will continue to mark the workflow as valid. These types of issues will be caught when the workflow runs.

Workflow YAML definition

The following is the reference documentation for the workflow definition file.

A *workflow definition file* is a YAML file that describes your workflow. The file is stored in a `~/.codecatalyst/workflows/` folder in the root of your source repository. The file can have a .yml or .yaml extension.

To create and edit the workflow definition file, you can use an editor such as vim, or you can use the CodeCatalyst console's visual editor or YAML editor. For more information, see Using the CodeCatalyst console's visual and YAML editors.

> **Note**
>
> Most of the YAML properties that follow have corresponding UI elements in the visual editor. To look up a UI element, use **Ctrl+F**. The element will be listed with its associated YAML property.

Topics

Example of a workflow definition file

The following is an example of a simple workflow definition file. It includes a few top-level properties, a `Triggers` section, and an `Actions` section with two actions: `Build` and `Test`. For more information, see About the workflow definition file.

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  Build:
```

```
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: docker build -t MyApp:latest .
  Test:
    Identifier: aws/managed-test@v1
    DependsOn:
      - Build
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm run test
```

Syntax guidelines and conventions

This section describes the syntax rules for the workflow definition file, as well as the naming conventions used in this reference documentation.

YAML syntax guidelines

The workflow definition file is written in YAML and follows the YAML 1.1 specification, so whatever is allowed in that specification is also allowed in the workflow YAML. If you're new to YAML, here are some quick guidelines to ensure you're supplying valid YAML code.

- **Case-sensitivity**: The workflow definition file is case-sensitive, so make sure you use the casing shown in this documentation.
- **Special characters**: We recommend using quotes or double-quotes around property values that include any of the following special characters: {, } , [,] , &, * , # , ? , | , - , < , >, = , ! , % , @ , : , ` and ,

  If you don't include the quotes, the special characters listed previously may be interpreted in an unexpected way.
- **Property names**: Property *names* (as opposed to property *values*) are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotes or double-quotes to enable special characters and spaces in property names.

Not permitted:

`'My#Build@action'`

`My#Build@action`

`My Build Action`

Permitted:

`My-Build-Action_1`

- **Escape codes**: If your property value includes escape codes (for example, `\n` or `\t`), follow these guidelines:
  - Use single quotes to return the escape code as a string. For example, `'my string \n my string'`, returns the string `my string \n my string`.
  - Use double quotes to parse the escape code. For example, `"my string \n my new line"`, returns:

    ```
    my string
    my new line
    ```
- **Comments**: Preface comments with #.

  Example:

  ```
  Name: MyWorkflow
  # This is a comment.
  SchemaVersion: 1.0
  ```
- **Triple dash (`---`)**: Do not use `---` in your YAML code. CodeCatalyst ignores everything after the `---`.

Naming conventions

In this guide, we use the terms *property* and *section* to refer to the main items in a workflow definition file.

- A *property* is any item that includes a colon (`:`). For example, in the following code snippet, all of the following are properties: `Name`, `SchemaVersion`, `RunMode`, `Triggers`, `Type`, and `Branches`.
- A *section* is any property that has sub-properties. In the following code snippet, there is one `Triggers` section.

> **Note**
>
> In this guide, 'sections' are sometimes referred to as 'properties', and vise versa, depending on the context.

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
```

Top-level properties

The following is the reference documentation for the top-level properties in the workflow definition file.

```
# Name
Name: workflow-name

# Schema version
SchemaVersion: 1.0

# Run mode
RunMode: QUEUED|SUPERSEDED|PARALLEL

# Compute
Compute:
...

# Triggers
```

```
Triggers:
...

# Actions
Actions:
...
```

Name

(Required)

The name of the workflow. The workflow name is shown in the workflows list and mentioned in notifications and logs. The workflow name and workflow definition file name can match, or you can name them differently. Workflow names do not need to be unique. Workflow names are limited to alphanumeric characters (a-z, A-Z, 0-9), hyphens (-), and underscores (_). Spaces are not allowed. You cannot use quotation marks to enable special characters and spaces in workflow names.

Corresponding UI: visual editor/Workflow properties/**Workflow name**

SchemaVersion

(Required)

The schema version of the workflow definition. Currently, the only valid value is `1.0`.

Corresponding UI: *none*

RunMode

(Optional)

How CodeCatalyst handles multiple runs. You can use one of the following values:

- QUEUED – Multiple runs are queued and run one after the other. You can have up to 50 runs in a queue.
- SUPERSEDED – Multiple runs are queued and run one after the other. A queue can only have one run, so if two runs end up together in the same queue, the later run supersedes (takes over from) the earlier run, and the earlier run is canceled.

- PARALLEL – Multiple runs occur simultaneously.

If this property is omitted, the default is QUEUED.

For more information, see Configuring the queuing behavior of runs.

Corresponding UI: visual editor/Workflow properties/Advanced/**Run mode**

Compute

(Optional)

The computing engine used to run your workflow actions. You can specify compute either at the workflow level or at the action level, but not both. When specified at the workflow level, the compute configuration applies to all actions defined in the workflow. At the workflow level, you can also run multiple actions on the same instance. For more information, see Sharing compute across actions.

For more information about compute, see Configuring the compute and runtime environment Docker images for a workflow.

Corresponding UI: *none*

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Compute:
  Type: EC2 | Lambda
  Fleet: fleet-name
  SharedInstance: true | false
```

Type

(Compute/**Type**)

(Required if Compute is set)

The type of compute engine. You can use one of the following values:

- **EC2** (visual editor) or EC2 (YAML editor)

  Optimized for flexibility during action runs.

- **Lambda** (visual editor) or `Lambda` (YAML editor)

  Optimized action start-up speeds.

For more information about compute types, see Compute types.

Corresponding UI: visual editor/Workflow properties/Advanced/**Compute type**

Fleet

(Compute/**Fleet**)

(Optional)

Specify the machine or fleet that will run your workflow or workflow actions. With on-demand fleets, when an action starts, the workflow provisions the resources it needs, and the machines are destroyed when the action finishes. Examples of on-demand fleets: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. For more information about on-demand fleets, see On-demand fleet properties.

With provisioned fleets, you configure a set of dedicated machines to run your workflow actions. These machines remain idle, ready to process actions immediately. For more information about provisioned fleets, see Provisioned fleet properties.

If `Fleet` is omitted, the default is `Linux.x86-64.Large`.

For more information about compute fleets, see Compute fleets.

Corresponding UI: visual editor/Workflow properties/Advanced/**Compute fleet**

SharedInstance

(Compute/**SharedInstance**)

(Optional)

Specify the compute sharing capability for your actions. With compute sharing, actions in a workflow run on the same instance (runtime environment image). You can use one of the following values:

- TRUE means that the runtime environment image is shared between workflow actions.

- FALSE means that a separate runtime environment image is started and used for each action in a workflow, so you can't share resources such as artifacts and variables without extra configuration.

For more information about compute sharing, see Sharing compute across actions.

Corresponding UI: *none*

Triggers

(Optional)

A sequence of one or more triggers for this workflow. If a trigger is not specified, then you must manually start your workflow.

For more information about triggers, see Starting a workflow run automatically with triggers.

Corresponding UI: visual editor/workflow diagram/**Triggers**

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Triggers:
  - Type: PUSH
    Branches:
      - branch-name
    FilesChanged:
      - folder1/file
      - folder2/

  - Type: PULLREQUEST
    Events:
      - OPEN
      - CLOSED
      - REVISION
    Branches:
      - branch-name
    FilesChanged:
      - file1.txt

  - Type: SCHEDULE
    # Run the workflow at 10:15 am (UTC+0) every Saturday
```

```
    Expression: "15 10 ? * 7 *"
    Branches:
      - branch-name
```

Type

(Triggers/**Type**)

(Required if `Triggers` is set)

Specify the type of trigger. You can use one of the following values:

- **Push** (visual editor) or PUSH (YAML editor)

  A push trigger starts a workflow run when a change is pushed to your source repository. The workflow run will use the files in the branch that you're pushing *to* (that is, the destination branch).
- **Pull request** (visual editor) or PULLREQUEST (YAML editor)

  A pull request trigger starts a workflow run when a pull request is opened, updated, or closed in your source repository. The workflow run will use the files in the branch that you're pulling *from* (that is, the source branch).
- **Schedule** (visual editor) or SCHEDULE (YAML editor)

  A schedule trigger starts workflow runs on a schedule defined by a cron expression that you specify. A separate workflow run will start for each branch in your source repository using the branch's files. (To limit the branches that the trigger activates on, use the **Branches** field (visual editor) or `Branches` property (YAML editor).)

  When configuring a schedule trigger, follow these guidelines:
  - Only use one schedule trigger per workflow.
  - If you've defined multiple workflows in your CodeCatalyst space, we recommend that you schedule no more than 10 of them to start concurrently.
  - Make sure you configure the trigger's cron expression with adequate time between runs. For more information, see Expression.

For examples, see Examples of triggers.

Corresponding UI: visual editor/workflow diagram/Triggers/**Trigger type**

Events

(Triggers/**Events**)

(Required if the trigger `Type` is set to `PULLREQUEST`)

Specify the type of pull request events that will start a workflow run. The following are the valid values:

- **Pull request is created** (visual editor) or `OPEN` (YAML editor)

  The workflow run is started when a pull request is created.
- **Pull request is closed** (visual editor) or `CLOSED` (YAML editor)

  The workflow run is started when a pull request is closed. The `CLOSED` event's behavior is tricky, and is best understood through an example. See Example: A trigger with a pull, branches, and a 'CLOSED' event for more information.
- **New revision is made to pull request** (visual editor) or `REVISION` (YAML editor)

  The workflow run is started when a revision to a pull request is created. The first revision is created when the pull request is created. After that, a new revision is created every time someone pushes a new commit to the source branch specified in the pull request. If you include the `REVISION` event in your pull request trigger, you can omit the `OPEN` event, since `REVISION` is a superset of `OPEN`.

You can specify multiple events in the same pull request trigger.

For examples, see Examples of triggers.

Corresponding UI: visual editor/workflow diagram/Triggers/**Events for pull request**

Branches

(Triggers/**Branches**)

(Optional)

Specify the branches in your source repository that the trigger monitors in order to know when to start a workflow run. You can use regex patterns to define your branch names. For example, use `main.*` to match all branches beginning with `main`.

The branches to specify are different depending on the trigger type:

- For a push trigger, specify the branches you're pushing *to*, that is, the *destination* branches. One workflow run will start per matched branch, using the files in the matched branch.

  Examples: `main.*`, `mainline`
- For a pull request trigger, specify the branches you're pushing *to*, that is, the *destination* branches. One workflow run will start per matched branch, using the workflow definition file and source files in the **source** branch (*not* the matched branch).

  Examples: `main.*`, `mainline`, `v1\-.*` (matches branches that start with `v1-`)
- For a schedule trigger, specify the branches that contain the files that you want your scheduled run to use. One workflow run will start per matched branch, using the the workflow definition file and source files in the matched branch.

  Examples: `main.*`, `version\-1\.0`

> **Note**
>
> If you *don't* specify branches, the trigger monitors all branches in your source repository, and will start a workflow run using the workflow definition file and source files in:
>
> - The branch you're pushing *to* (for push triggers). For more information, see Example: A simple code push trigger.
> - The branch you're pulling *from* (for pull request triggers). For more information, see Example: A simple pull request trigger.
> - All branches (for schedule triggers). One workflow run will start per branch in your source repository. For more information, see Example: A simple schedule trigger.

For more information about branches and triggers, see Trigger considerations when branching.

For more examples, see Examples of triggers.

Corresponding UI: visual editor/workflow diagram/Triggers/**Branches**

FilesChanged

(Triggers/**FilesChanged**)

(Optional if the trigger Type is set to PUSH, or PULLREQUEST. Not supported if the trigger Type is set to SCHEDULE.)

Specify the files or folders in your source repository that the trigger monitors in order to know when to start a workflow run. You can use regular expressions to match file names or paths.

For examples, see Examples of triggers.

Corresponding UI: visual editor/workflow diagram/Triggers/**Files changed**

Expression

(Triggers/**Expression**)

(Required if the trigger Type is set to SCHEDULE)

Specify the cron expression that describes when you want your scheduled workflow runs to occur.

Cron expressions in CodeCatalyst use the following six-field syntax, where each field is separated by a space:

```
minutes hours days-of-month month days-of-week year
```

Examples of cron expressions

Minutes	Hours	Days of month	Month	Days of week	Year	Meaning
0	0	?	*	MON-FRI	*	Runs a workflow at midnight (UTC+0) every Monday through Friday.
0	2	*	*	?	*	Runs a workflow at 2:00 am

Minutes	Hours	Days of month	Month	Days of week	Year	Meaning
						(UTC+0) every day.
15	22	*	*	?	*	Runs a workflow at 10:15 pm (UTC +0) every day.
0/30	22-2	?	*	SAT-SUN	*	Runs a workflow every 30 minutes Saturday through Sunday between 10:00 pm on the starting day and 2:00 am on the following day (UTC +0).

Minutes	Hours	Days of month	Month	Days of week	Year	Meaning
45	13	L	*	?	2023-2027	Runs a workflow at 1:45 pm (UTC +0) on the last day of the month between the years 2023 and 2027 inclusive.

When specifying cron expressions in CodeCatalyst, make sure you follow these guidelines:

- Specify a single cron expression per `SCHEDULE` trigger.
- Enclose the cron expression in double-quotes (") in the YAML editor.
- Specify the time in Coordinated Universal Time (UTC). Other time zones are not supported.
- Configure at least 30 minutes between runs. A faster cadence is not supported.
- Specify the *`days-of-month`* or *`days-of-week`* field, but not both. If you specify a value or an asterisk (*) in one of the fields, you must use a question mark (?) in the other. The asterisk means 'all' and the question mark means 'any'.

For more examples of cron expressions and information about wildcards like ?, *, and L, see the Cron expressions reference in the *Amazon EventBridge User Guide*. Cron expressions in EventBridge and CodeCatalyst work exactly the same way.

For examples of schedule triggers, see Examples of triggers.

Corresponding UI: visual editor/workflow diagram/Triggers/**Schedule**

Actions

A sequence of one or more actions for this workflow. CodeCatalyst supports several action types, such as build and test actions, which offer different types of functionality. Each action type has:

- an `Identifier` property that indicates the action's unique, hard-coded ID. For example, `aws/build@v1` identifies the build action.
- a `Configuration` section that contains properties that are specific to the action.

For more information about each action type, see Action types. The Action types topic has links into the documentation for each action.

The following is the YAML reference for actions and action groups in the workflow definition file.

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Actions:
  action-or-gate-name:
    Identifier: identifier
    Configuration:
    ...
  #Action groups
  action-group-name:
    Actions:
      ...
```

action-or-gate-name

(Actions/*action-or-gate-name*)

(Required)

Replace *action-name* with a name you want to give the action. Action names must be unique within the workflow, and must only include alphanumeric characters, hyphens, and underscores. For more information about syntax rules, see YAML syntax guidelines.

For more information about naming practices for actions, including restrictions, see the action-or-gate-name.

Corresponding UI: visual editor/*action-name*/Configuration tab/**Action name** or **Action display name**

action-group-name

(Actions/*action-group-name*)

(Optional)

An *action group* contains one or more actions. Grouping actions into action groups helps you keep your workflow organized, and also allows you to configure dependencies between different groups.

Replace *action-group-name* with a name you want to give the action group. Action group names must be unique within the workflow, and must only include alphanumeric characters, hyphens, and underscores. For more information about syntax rules, see YAML syntax guidelines.

For more information about action groups, see Grouping actions into action groups.

Corresponding UI: *none*

Track and organize work with issues in CodeCatalyst

In CodeCatalyst, you can monitor features, bugs, and any other work involved in your project. Each piece of work is kept in a distinct record called an *issue*. You can break up an issue into smaller objectives by adding a checklist of tasks to it. Each issue can have a description, assignee, status, and other properties, which you can search for, group, and filter on. You can view your issues using the default views, or you can create your own views with custom filtering, sorting, or grouping. For more information about concepts related to issues, see Issues concepts. To learn how to create your first issue, see Creating an issue in CodeCatalyst.

Here is one possible workflow for a team using issues:

Jorge Souza is a developer working in a project. He and his fellow project members Li Juan, Mateo Jackson, and Wang Xiulan collaborate to determine what work needs to be done. Every day, he and his fellow developers hold a sync-up meeting, led by Wang Xiulan. They pull up the board by navigating to one of their teams views of the board. By creating views, users and teams can save filters, groupings, and sorting of issues to easily view issues that meet their specified criteria. Their view contains issues grouped by **Assignee** and sorted by **Priority** to show the most important issues and status of the issues for each developer. As Jorge is assigned tasks to complete, he plans his work by creating an issue for each task. When creating issues, Jorge can choose the appropriate **Status**, **Priority**, and work **Estimation** effort. For larger issues, Jorge adds tasks to the issue, to break the work into smaller objectives. Jorge creates his issues with a draft status, such as **backlog**, as he doesn't plan to start on them immediately. Issues in a draft status appear on the **Drafts** view where they are to be planned and prioritized. Once Jorge is ready to start the work, he moves the corresponding issue to the board by updating its status to a status in another category (**Not Started**, **Started**, or **Completed**). As each task is being worked on, the team can filter by the title, status, assignee, label, priority, and estimation to find a specific issue or similar issues that match the specified parameter. Using the board, Jorge and his team can see the number of tasks completed for each issue, and track the day-to-day progress by dragging each issue from one status to the next until the task is complete. As the project progresses, finished issues accumulate in the **Completed** status. Wang Xiulan decides to remove them from view by archiving them using the quick archive button, so that the developers can focus on the issues that are related to current and upcoming work.

When planning their work, the developers working on the project choose **Sort by** and **Group by** to find the issues they want to move from the backlog to the board. They might choose to add issues to the board based on the highest priority customer requests, so they group the board by

a **Customer request** label and sort by **Priority**. They might also sort by estimate to ensure that they're taking on a volume of work they can achieve. The project manager, Saanvi Sarkar, regularly reviews and grooms the backlog to help ensure that the priority accurately reflects the importance of each issue to the success of the project.

Topics

Issues concepts

Creating an issue is a quick and efficient way to track work being done within a project. You can use issues to help you discuss work in daily sync-up meetings, prioritize work, and more.

This page includes a list of concepts that will help you effectively use issues in CodeCatalyst.

Active issues

Active issues are issues that are any issues that are not in a **Draft** status or archived. In other words, active issues are issues with a status in any of the following status categories: **Not started**, **Started**, and **Completed**. For more information about statuses and status categories, see Status and status categories.

You can view all of the active issues in your project from the default **Active issues** view.

Archived issues

An *archived issue* is an issue that is no longer relevant to your project. For example, you can archive an issue if it is completed and you no longer need to see it in the **Done** column, or if it was created in error. Archived issues can be unarchived if needed.

Assignee

The *assignee* is the person the issue is assigned to. If the person doesn't appear in the list when you search for them, they have not been added to your project. To add them, see Inviting a user to a project. To enable multiple assignees to an issue, see Enabling or disabling multiple assignees.

Issues with multiple assignees will appear on your board with different colored avatars, each representing one of the assignees.

Custom fields

Custom fields allow you to customize different attributes of an issue according to your needs for tracking and maintaining issues within a project. For example, you can add a field for roadmapping, a specific due date, or a requester field.

Estimate

In agile development, the *estimate* is known as story points. You can use the estimate for an issue to represent the amount of work required, in addition to the ambiguity and complexity of the issue. Consider using higher estimates for issues with greater risk, difficulty, and unknowns.

For more information about estimation types and how to configure them, see Configuring issue effort estimation.

Issue

An *issue* is a record that tracks the work related to your project. You can create an issue for a feature, a task, a bug, or any other body of work related to your project. If you're using agile development, an issue can also describe an epic or user story.

Label

The *label* is used to group, sort and filter issues. You can enter a new label name or choose one of the labels from the populated list. This list consists of recently used labels in the project. An issue can have multiple labels, and a label can be removed from an issue. To customize labels, see Categorizing work with labels.

Priority

Priority refers to the level of importance of the issue. There are four options: **Low**, **Medium**, **High**, and **No priority**.

Status and status categories

The *status* is the current state of the issue and is used to quickly check the progress of an issue through its lifecycle, from inception to completion. All issues must have a status, and each status

belongs to a *status category*. Status categories are used to help organize your statuses and populate the default issue views.

There are five default statuses and four status categories in CodeCatalyst. You can create other statuses, but you cannot create other status categories. The following list contains the default statuses and their status categories in parenthesis: **Backlog (Draft)**, **To do (Not started)**, **In progress (Started)**, **In review (Started)**, and **Done (Completed)**.

For more information on working with statuses, see Tracking work with custom statuses.

Tasks

Tasks can be added to issues to further break down and organize the work of that issue. You can add tasks to an issue on creation, or add tasks to an existing issue. When viewing an issue, you can reorder, remove, or mark its tasks as completed.

Views

Issues in your CodeCatalyst project are displayed in *views*. Views can either be grid views that show issues in list format or board views that show issues as tiles in columns organized by issue status. There are four default views, and you can create your own views with custom grouping, filtering, and sorting. The following list contains details about the four default views.

- The **Drafts** view is a grid view that shows issues not currently being worked on. Any issue created with a status in the **Draft** status category shows up in this view. This view can be used by teams to see which issues are still being defined or are waiting to be assigned and worked on.
- The **Active issues** view is a board view of all issues that are currently being worked on. Any issue with a status in the **Not started**, **Started**, or **Completed** status categories will show up in this view.
- The **All issues** view is a grid view that shows all the issues in the project, both *drafts* and *active issues*.
- The **Archived** view shows all archived issues.

Tracking work with issues

You can plan and track your work on a project by using issues. Each issue is a piece of work kept in a distinct record. Issues can be broken down into tasks to further organize and track the work

of that issue. You can also create links between issues to help you keep track of related work, add labels to help you organize and categorize work, group issues, assign priorities to work, and indicate whether work is blocked.

When you're ready to work on an issue or set of issues, you can estimate the work, assign them to users, and add comments to help others understand the work and its progress. You can also export issues to help bring the information they contain into other formats.

Creating an issue in CodeCatalyst

Development teams create issues to help track and manage their work. You can create issues within a project based on your needs. For example, you could create an issue to track updating a variable in your code. You can assign issues to other users in the project, use labels to help you track your work, and more.

Follow these instructions to create an issue in CodeCatalyst.

To create an issue

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the project where you want to create an issue.
3. On the project home page, choose **Create issue**. Alternatively, in the navigation pane, choose **Issues**.
4. Choose **Create issue**.

> **Note**
>
> You can also add issues inline when using a grid view.

5. Enter a title for the issue.
6. (Optional) Enter a **Description**. You can use Markdown to add formatting.
7. (Optional) Choose a **Status**, **Priority**, and **Estimation** for the issue.

> **Note**
>
> If the project's estimation setting is set to **Hide estimates**, there will not be an **Estimation** field.

8. (Optional) Add tasks to the issue. Tasks can be used to break down the work of an issue into smaller objectives. To add a task, choose **+ Add tasks**. Then, input the task name in the text field and press enter. After adding tasks, you can mark them as complete by choosing the checkbox, or reorder them by choosing and dragging the task from the left side of the checkbox.
9. (Optional) Add an existing label or create a new label and add it by choosing **+ Add label**.
    a. To add an existing label, choose the label from the list. You can enter a search term in the field to search all labels containing that term in the project.
    b. To create a new label and add it, enter the name of the label you want to create in the search field and press enter.
10. (Optional) Add an assignee by choosing **+ Add an assignee**. You can quickly add yourself as the assignee by choosing **+ Add me**.

> **Tip**
>
> You can choose to assign an issue to **Amazon Q** to have Amazon Q try to solve the issue. For more information, see Tutorial: Using CodeCatalyst generative AI features to speed up your development work. This feature is only available in the US West (Oregon) Region.
> This functionality requires that generative AI features are enabled for the space. For more information, see Managing generative AI features.

11. (Optional) Add an existing **custom field** or create a new custom field. Issues can have multiple custom fields.
    a. To add an existing custom field, choose the custom field from the list. You can enter a search term in the field to search all custom fields containing that term in the project.
    b. To create a new custom field and add it, enter the name of the custom field you want to create in the search field and press enter. Then choose the type of custom field you want to create and set a value.
12. Choose **Create issue**. A notification appears in the lower right corner: If the issue was created successfully, a confirmation message appears saying the issue was successfully created. If the issue was not created successfully, an error message with the reason for the failure appears. You can then choose **Retry** to edit and retry creating the issue, or choose **Discard** to discard the issue. Both options will dismiss the notification.

> **Note**
>
> You cannot link a pull request to an issue when you create it. However, you can edit it after you create it to add links to pull requests.

Best practices when creating and working with issues assigned to Amazon Q

When you create issues, sometimes some of them linger. The causes for this can be complex and variable. Sometimes it's because it's not clear who should work on it. Other times the issue requires research into or expertise with a particular part of the code base and the best candidates for the work are busy with other issues. Often there is other urgent work must be attended to first. Any or all of these causes can result in issues that aren't worked on. CodeCatalyst includes integration with a generative AI assistant called Amazon Q that can analyze an issue based on its title and its description. If you assign the issue to Amazon Q, it will attempt to create a draft solution for you to evaluate. This can help you and your team to focus and optimize work on issues that require your attention, while Amazon Q works on a solution for problems you don't have resources to address immediately.

> **Note**
>
> **Powered by Amazon Bedrock**: AWS implements automated abuse detection. Because the **Assign issues to Amazon Q** feature with Amazon Q Developer Agent for software development is built on Amazon Bedrock, users can take full advantage of the controls implemented in Amazon Bedrock to enforce safety, security, and the responsible use of artificial intelligence (AI).

Amazon Q performs best on simple issues and straightforward problems. For best results, use plain language to clearly explain what you want done. The following are some best practices to help you create issues optimized for Amazon Q to work on.

> **Important**
>
> Generative AI features are only available in the US West (Oregon) Region.

- **Keep it simple**. Amazon Q does best with simple code changes and fixes that can be explained in the title and description of the issue. Don't assign issues with vague titles or overly flowery or contradictory descriptions.
- **Be specific**. The more information you can provide about the exact changes needed to resolve the issue, the more likely Amazon Q will be able to create a solution that solves the issue. If possible, include specific details such as the name of APIs you want changed, methods you want updated, tests that need changes, and any other details you can think of.
- **Make sure you have all the details included in the title and description of the issue before assigning it to Amazon Q**. You can't change the title or description of an issue after you assign it to Amazon Q, so make sure you have all the information required in an issue before you assign it to Amazon Q.
- **Only assign issues that require code changes in a single source repository**. Amazon Q can only work on code in a single source repository in CodeCatalyst. Linked repositories are not supported. Make sure that the issue only requires changes in a single source repository before you assign that issue to Amazon Q.
- **Use the default suggested by Amazon Q for approving each step**. By default, Amazon Q will require your approval for each step it takes. This allows you to interact with Amazon Q in comments not only on the issue, but also on any pull request it creates. This provides a more interactive experience with Amazon Q that helps you adjust its approach and refine the code it creates to solve the issue.

  > **Note**
  >
  > Amazon Q does not respond to individual comments in issues or pull requests, but it will review them when asked to reconsider its approach or create a revision.

- **Always carefully review the approach suggested by Amazon Q**. Once you approve its approach, Amazon Q will start work on generating code based on that approach. Make sure that the approach seems correct and includes all the details you expect before you tell Amazon Q to proceed.
- **Make sure to only allow Amazon Q to work on workflows if you don't have existing workflows that might deploy them before they're reviewed**. Your project might have workflows configured to start runs on pull request events. If so, any pull request that Amazon Q creates that includes creating or updating workflow YAML might start a run of those workflows included in the pull request. As a best practice, don't choose to allow Amazon Q to work on

workflow files unless you are sure there are no workflows in your project that will automatically run these workflows before you review and approve the pull request it creates.

For more information, see Tutorial: Using CodeCatalyst generative AI features to speed up your development work and Managing generative AI features.

Estimating an issue

In agile development, the *estimate* is known as story points. You can use the estimate for an issue to represent the amount of work required, in addition to the ambiguity and complexity of the issue. Consider using higher estimates for issues with greater risk, difficulty, and unknowns.

Before you can begin estimating your issues, you must first choose what type of estimates you want to use for your project. There are two types to choose from by default. To use either **T-shirt sizing** or **Fibonacci sequencing** effectively, your team must align on what each size represents. Decide together what each estimate represents to you, and then start applying those estimates to each issue. Consider periodically reviewing

Follow these steps to configure the setting for effort estimations for issues in CodeCatalyst.

To configure effort estimation for issues

1. In the navigation pane, choose **Issues**.
2. Choose **Active issues** to open the **issues view switcher** dropdown menu and choose **Settings**.
3. In **Estimation** in the **Basic settings** section, choose how the estimation values will be displayed. The types of estimates available are **T-shirt sizing**, **Fibonacci sequencing**, or **Hide estimates**. If the project's estimation setting is set to **Hide estimates**, there will not be an **Estimation** field in the issues for the project.

   When the estimation type is updated, no data will be lost and the estimation value of all issues will be converted automatically. The conversion mapping is shown in the following table.

T-shirt size	Fibonacci sequence
XS	1
XS	2

T-shirt size	Fibonacci sequence
S	3
M	5
L	8
XL	13

To add or change an estimate for an issue, you can edit the issue.

Editing and collaborating on issues in CodeCatalyst

Contents

Editing an issue

Follow these steps to edit the title, description, status, assignee, priority, estimate, or labels of an issue.

To edit an issue

1. Choose the issue that you want to edit to view the issue details. For help on finding your issue, see Finding and viewing issues.
2. To edit the issue title, choose the title, enter a new title, and press enter.
3. To edit the description, choose the description, enter a new description, and press enter. You can use Markdown to add formatting.

4. In **Tasks**, you can view and manage the tasks for the issue. If there are no tasks, you can have Amazon Q analyze the issue and recommend tasks that can break down the work in the issue into separate items that can each be assigned to a user. For more information, see Managing tasks on issues.
5. To edit the **Status**, **Estimate**, or **Priority**, choose an option from the respective dropdown menus.
6. In **Labels**, you can add an existing label, create a new label, or remove a label.

    a. To add an existing label, choose **+ Add label** and choose the label from the list. You can enter a search term in the field to search all labels containing that term in the project.

    b. To create a new label and add it, choose **+ Add label** enter the name of the label you want to create in the search field and press enter.

    c. To remove a label, choose the **X** icon next to the label you want to remove. If you remove a label from all issues, the label will appear in the **Unused labels** section in the **Labels** section of issue **settings**. Unused labels appear at the end of the list of labels when using filters or adding labels to an issue. You can find an overview of all labels (used and unused) and issues that have them in the issue **settings**.
7. To assign an issue, choose **+ Add an assignee** in the **Assignee** section, then search and choose the assignee from the list. You can choose **+ Add me** to quickly add yourself as the assignee.
8. In **Attachments**, you can add, download, or remove attachments. For more information, see Working with attachments.
9. To link a pull request, choose **Link pull request**, and then either choose a pull request from the list or enter its URL or ID. To unlink a pull request, choose the unlink icon.

    > **Tip**
    >
    > After you add a link to a pull request to an issue, you can quickly navigate to it by choosing its ID in the list of linked pull requests. You can use the URL of a pull request to link pull requests that are in different projects than the issue board, but only users that are members of that project will be able to view or navigate to that pull request.

10. (Optional) Add and set an existing custom field, create a new custom field, or remove a custom field. Issues can have multiple custom fields.

    a. To add an existing custom field, choose the custom field from the list. You can enter a search term in the field to search all custom fields containing that term in the project.

b. To create a new custom field and add it, enter the name of the custom field you want to create in the search field and press enter. Then choose the type of custom field you want to create and set a value.

c. To remove a custom field, choose the **X** icon next to the custom field you want to remove. If you remove a custom field from all issues, the custom field will be deleted and you will no longer see it when filtering.

Working with attachments

You can add attachments to issues in CodeCatalyst to make related files easily accessible. Use the following procedure to manage attachments for an issue.

The size of attachments added to issues counts towards your space's storage quotas. For information about viewing and managing attachments for your project, see Viewing and managing attachments.

> **Important**
>
> Attachments to issues are not scanned or analyzed by Amazon CodeCatalyst. Any user could add an attachment to an issue that might potentially contain malicious code or content. Make sure that users are aware of best practices when it comes to managing attachments and guarding against malicious code, content, or viruses.

To add, download, or remove attachments

1. Choose the issue for which you want to manage attachments. For help on finding your issue, see Finding and viewing issues.
2. To add an attachment, choose **Upload file**. Navigate to the file in your operating system's file explorer and select it. Choose **Open** to add it as an attachment. For quota information, such as maximum attachment size, see Quotas for issues in CodeCatalyst.

   Note the following restrictions to attachment file names and content types:

   - The following characters are not permitted in file names:
     - Control characters: `0x00-0x1f` and `0x80-0x9f`
     - Reserved characters: /, ?, <, >, \, :, *, |, and "

- Unix reserved filenames: `.` and `..`
- Trailing periods and spaces
- Windows reserved filenames: `CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9`

- The content type of the attachment must adhere to the following media-type pattern:

```
media-type = type "/" [tree "."] subtype ["+" suffix]* [";" parameter];
```

For example, `text/html; charset=UTF-8`.

3. To download an attachment, choose the ellipses menu next to the attachment you want to download and choose **Download**.
4. To copy an attachment's URL, choose the ellipses menu next to the attachment of which you want to copy the URL and choose **Copy URL**.
5. To remove an attachment, choose the ellipses menu next to the attachment you want to remove and choose **Delete**.

Viewing and managing attachments

You can view a table with every attachment added to issues in your project in issue settings. This table includes details of each attachment, including information such as content type, when it was added, the issue it's added to and its status, and the file size.

This table can be used to easily identify large attachments on issues that are completed or archived to remove them to free up space storage.

> **Important**
>
> Attachments to issues are not scanned or analyzed by Amazon CodeCatalyst. Any user could add an attachment to an issue that might potentially contain malicious code or content. Make sure that users are aware of best practices when it comes to managing attachments and guarding against malicious code, content, or viruses.

To view and manage all issue attachments in a project

1. In the navigation pane, choose **Issues**.

2. Choose the ellipsis icon and choose **Settings**.
3. Choose the **Attachments** tab.

Managing tasks on issues

Tasks can be added to issues to further break down, organize, and track the work of that issue. You can create tasks yourself, or you can use Amazon Q to recommend tasks based on its analysis of the issue and its complexity.

Amazon Q Developer is a generative AI-powered conversational assistant that can help you to understand, build, extend, and operate AWS applications. To accelerate your building on AWS, the model that powers Amazon Q is augmented with high-quality AWS content to produce more complete, actionable, and referenced answers. For more information, see What is Amazon Q Developer? in the *Amazon Q Developer User Guide*.

To manage tasks on an issue

1. Choose the issue for which you want to manage tasks. For help on finding your issue, see Finding and viewing issues.
2. In **Tasks**, you can view and manage tasks for the issue.

    1. To add a task, input the task name in the text field and press enter.
    2. If there are no tasks for the issue, you can choose to have Amazon Q analyze the issue and create tasks based on the issue title, description, and its analysis of the complexity of the issue and the repository code, choose **Recommend tasks**. You will need to specify the source repository that contains the code for the issue. Choose **Start recomending tasks** to begin the task recommendation analysis. That dialog will close. Once the recommendation is complete, choose **View recommended tasks** to review the tasks and take any needed action, such as deleting or adding tasks to the list or reordering the recommended tasks, before choosing **Create tasks**.

        After tasks are created for you, you can assign them to users and work with them the same way you work with manually created tasks.
    3. To mark a task as completed, choose the checkbox of the task.
    4. To view or update the details of a task, choose it from the list.
    5. To reorder the tasks, choose and drag the task from the left side of the checkbox.
    6. To remove a task, choose the ellipses menu of the task and choose **Remove**.

Link an issue to another issue

If an issue is related to work in another issue, you can link them. This is a quick way to help you understand and track dependencies between work items.

When linking an issue, there are four states to choose from:

- Related to: Use this state to indicate work that is related to the linked issue.
- Blocked by: Use this state to indicate that the issue is blocked by the linked issue.
- Blocking to: Use this state to indicate that the issue is blocking progress on the linked issue.
- Duplicate: Use this state to indicate that the issue duplicates work captured in another issue.

You can change the state of a link after you create it. Links and their link state appear in **Linked issues** in both the issue where you create the link, and the issue that is linked. You can also remove links after they are created.

To link an issue to another issue or task

1. Open the issue you want to link to another issue. For help with finding your issue, see Finding and viewing issues.
2. Choose **Link issues**.
3. In **Mark as**, choose the state for the link.

   In **Issue #**, enter the number for the issue, or choose it from the drop-down list.
4. To add another link, choose **Add linked issue**.
5. When you are done, choose **Update** to update the issue and all linked issues with the linked relationship.

Marking an issue as blocked or unblocked

If something is preventing you from working on an issue, you might want to mark it as *blocked*. For example, your issue might be blocked if it relies on a change to another part of your code base that hasn't been merged yet.

When you mark an issue as blocked, CodeCatalyst adds a red **Blocked** label to the issue, making it highly visible in your backlog or archive, or on your board.

You can unblock the issue when outside circumstances are resolved.

To mark an issue as blocked

1. Open the issue you want to mark as blocked. For help with finding your issue, see Finding and viewing issues.
2. Choose **Actions**, and then choose **Mark as blocked**.

To unblock an issue

1. Open the issue that you want to unblock. For help with finding your issue, see Finding and viewing issues.
2. Choose **Actions**, and then choose **Mark as unblocked**.

Adding, editing, or deleting comments

You can leave a comment on an issue. In the comments you can tag other space members, other projects in the space, related issues, and code.

To add a comment to an issue

1. Navigate to your project.
2. In the navigation bar choose **Issues**.
3. Choose the issue where you want to add the comment. For help on finding your issue, see Finding and viewing issues.
4. Enter the comment in the **Comments** field. You can use Markdown to add formatting.
5. Choose **Send**.

To edit a comment

You can edit comments you make on issues. You can only edit comment that you authored.

1. Navigate to your project.
2. In the navigation bar choose **Issues**.
3. Choose the issue where you want to edit a comment. For help on finding your issue, see Finding and viewing issues.

4. To edit a comment, find the comment you want to edit.

> **Tip**
>
> You can sort comments by oldest or newest first. Comments are loaded 10 at a time.

5. Choose the ellipsis icon, then choose **Edit**.
6. Edit the comment. You can use Markdown to add formatting.
7. Choose **Save**. The comment is now updated.

To delete a comment

You can delete comments you make on issues. You can only delete comments that you authored. When a comment is deleted, your user name will show, but with the words *This comment has been deleted* in place of the original comment text.

1. Navigate to your project.
2. In the navigation bar choose **Issues**.
3. Choose the issue where you want to delete a comment. For help on finding your issue, see Finding and viewing issues.
4. Choose the ellipsis icon, choose **Delete**, and then choose **Confirm**.

Using mentions in a comment

You can mention space members, other projects in the space, related issues, and code in comments. Doing so creates a quick link to the user or resource you mention.

To @mention in a comment

1. Navigate to your project.
2. In the navigation bar choose **Issues**.
3. Choose the issue that you want to edit to view the issue details. For help on finding your issue, see Finding and viewing issues.
4. Choose the **Add a comment** textbox.
5. Type @*`user_name`* to mention another user.
6. Type @*`project_name`* to mention a project.

7. Type `@issue_name` or `@issue_number` to mention another issue.
8. Type `@file_name` to mention specific files or code in a source repository.

> **Note**
>
> A list of the top 5 items (users, source repositories, projects, etc) containing the terms that match your `@mention` will populate as you type.

9. Choose the desired item you would like to mention. The pathway showing where the item is located will populate in the comment textbox.
10. Finish your comment and choose **Send**.

Finding and viewing issues

The following sections describe how to effectively search for and view issues within a CodeCatalyst project.

Searching for an issue

You can find an issue by searching for specific parameters. For more information about refining your search, see Search for code, issues, projects, and users in CodeCatalyst.

To search for an issue

1. Navigate to your project.
2. Use the search bar to search for issues or information related to issues. You can use query parameters to refine your search. For more information, see Search for code, issues, projects, and users in CodeCatalyst.

Sorting issues

By default, issues in CodeCatalyst are sorted by **Manual order**. Manual order displays issues in the order they are moved to by users. You can drag and drop issues when sorted in Manual order to change their order. This sorting option is helpful when grooming the issues backlog and prioritizing issues.

The following table shows how issues can be sorted in both grid and board views.

Grid view sorting options	Board view sorting options
Manual order	Manual order
Last updated	Last updated
Priority	Priority
Estimate	Estimate
Title	Title
ID	
Status	
Blocked	
Custom fields	

Use the following procedure to change how issues are sorted.

To sort issues

1. Navigate to your project.
2. In the navigation pane, choose **Issues**. The default view is the **Board**.
3. (Optional) Choose **Active issues** to open the **issues view switcher** dropdown menu to navigate to a different issues view.
4. To sort a grid view, there are two options:
    a. Choose the **header** of the field you want to sort by. Choosing the **header** will cycle between ascending and descending order.
    b. Choose the **Sort by** dropdown menu and choose a parameter to sort by. Issues will be sorted in ascending order.
5. To sort a board view, choose the **Sort by** dropdown menu and choose a parameter to sort by. Issues will be sorted in ascending order.

Grouping issues

Grouping is used to organize issues on the board by multiple parameters, such as assignee, labels, and priority.

To group issues

1. Navigate to your project.
2. In the navigation pane, choose **Issues**. The default view is the **Board**.
3. (Optional) Choose **Active issues** to open the **issues view switcher** dropdown menu to navigate to a different issues view.
4. Choose **Group**.
5. In **Group by**, choose a parameter to group by:
   - If you choose **Assignee** or **Priority**, choose the **Group order**.
   - If you choose **Label**, choose the labels and then choose **Group order**.
6. (Optional) Choose the **Show empty groups** toggle to show or hide groups that have no issues currently assigned to them.
7. The view updates as you make your choices. An issue only appears in the group that matches the configured parameters.

Filtering issues

Use filtering to find issues that contain a specified name, priority, label, custom fields, or assignee.

To filter issues

1. Navigate to your project.
2. In the navigation pane, choose **Issues**.
3. (Optional) Choose **Active issues** to open the **issues view switcher** dropdown menu to navigate to a different issues view.

> **Note**
>
> To filter based on a string in the issue name or description, enter the string into the issues search bar.

4. Choose **Filter**, then choose **+ Add filter**.
5. Choose the parameters to filter for. You can choose multiple filters and parameters. You can configure filters to show issues that match every filter or any individual filter by selecting **and** or **or**. The view will update to show the issues that match the filter.

Progressing an issue

Every issue has a lifecycle. In CodeCatalyst, issues typically start as a draft in the backlog. When work for that issue is to be started, it is moved into another status category and moves through various statuses until it is complete, and then it is archived. You can move or progress an issue through its lifecycle in the following ways:

- You can move an issue between the backlog and the board.
- You can move in-progress issue through various completion stages.
- You can archive an issue that is completed.

Moving an issue beetween the backlog and board

You can move an issue from the backlog to the board once you begin to work on the issue. You can also move an issue back to the backlog if the work is postponed.

To move an issue between the backlog and the board

1. Navigate to your project.
2. In the navigation pane, choose **Issues**. The default view is the **Board**.
3. To move an issue from the board to the backlog:
    a. Choose the issue that you want to move. For help with finding your issue, see Finding and viewing issues.
    b. Choose **Backlog** from the dropdown **Status** menu.
4. To move an issue from the backlog to the board:
    a. To navigate to the backlog, choose **Board** and choose **Backlog**.
    b. Choose the issue that you want to move. For help with finding your issue, see Finding and viewing issues.
    c. Choose **Add to board**, or choose a **Status** other than **Backlog**.

Progress an issue through lifecycle stages on the board

You can move an issue within a board through different statuses until completion.

To move an issue within the board

1. In the navigation pane, choose **Issues**. The default view is the **Board**.
2. Do one of the following:
   - Drag and drop an issue to another status.
   - Choose an issue, and then choose a status from the **Status** dropdown menu.
   - Choose an issue, and then choose **Move to: *`next-status`***.

For information on archiving an issue, see Archiving an issue.

Moving issues between groups

You can group issues in the **All issues** and **Board** views by various parameters. If the issues are grouped, you can move issues from one group to another. Moving an issue from one group to another will automatically edit the field that the issues are grouped on to match the target group.

As an example scenario, assume there is a company using CodeCatalyst that has issues assigned to two people, Wang Xiulan and Saanvi Sarkar. The board is grouped by `Assignee`, and there are two groups, one for each assignee. Moving an issue from the Wang Xiulan group to the Saanvi Sarkar group will update the issue's assignee to Saanvi Sarkar.

Archiving an issue

> **Note**
>
> Issues are not deleted within a project, they are archived. To delete issues, you must delete the project.

You can archive an issue when it is no longer needed in your project. When you archive an issue, CodeCatalyst removes it from all views that filter out archived issues. Archived issues can be viewed in the **Archived issues** default view, where they can be unarchived if needed.

You archive an issue if:

- You have completed the issue and no longer need it in the **Done** column.
- You have no plans to work on it.
- You created it in error.
- You have reached the maximum number of active issues.

To archive an issue

1. Open the issue you want to archive. For help with finding your issue, see Finding and viewing issues.
2. Choose **Actions**, and then choose **Move to archive**.
3. (Optional) To quickly archive multiple issues with a **Completed** status, choose the vertical ellipsis at the top of any **Completed** status on the board and choose **Archive issues**.

To unarchive an issue

1. Open the issue that you want to unarchive. You can view a list of archived issues by opening the **Archived issues** view from the **issues view switcher** dropdown menu. For help with finding your issue, see Finding and viewing issues.
2. Choose **Unarchive**.

Exporting issues

You can export issues in your current view into a .xlsx file. To export issues, perform the following steps.

To export issues

1. Navigate to your project.
2. In the navigation bar choose **Issues**.
3. Choose **Active issues** to open the **issues view switcher** dropdown menu and navigate to the view containing the issues you want to export. Only issues shown in the view will be exported.
4. Choose the ellipses menu and choose **Export to Excel**.

5. The .xlsx file downloads. By default, it is titled the name of the project and the date the export was completed.

Organizing work with backlogs, labels, and boards

Not all teams work in the same way. You can configure the way issues appear and can be assigned in Amazon CodeCatalyst to help you precisely understand what's being worked on and the status of that work. You can choose what estimation method to allow for issues so that your users all use the same estimation method. You can create cusotm labels and statuses which can also be used to filter the view of the work. Depending on how your team works, you can configure whether to allow multiple assignees for an issue, or only allow an issue to be assigned to a single user. You can also create custom views of issues so that work is displayed in a way that shows the most relevant information to you or your team.

Categorizing work with labels

You can customize labels for issues. This includes editing the label and changing the color. Labels can help you categorize and organize your work. For example, you can create labels for specific aspects of your software, or for different groups or teams.

Topics

Creating a label

In CodeCatalyst, you create labels by either adding them when you create a new issue or when you edit an existing issue. For more information, see Creating an issue in CodeCatalyst and Editing and collaborating on issues in CodeCatalyst.

Editing a label

Use the following procedure to change the name or color of an existing label.

To edit a label

1. In the navigation pane, choose **Issues**.

2. Choose **Active issues** to open the **issues view switcher** dropdown menu and choose **Settings**.
3. On the **Labels** tile is a list of the labels used in the project. Choose the edit icon next to the label that you want to edit. Do one or more of the following:
   a. Edit the name of the label.
   b. To change the color, choose the color wheel. Use the picker to choose a new color.
4. To save the changes you made to the label, choose the check mark icon.
5. The changed label is now visible in your list of available labels. You can also see how many issues are using that label.

> **Note**
>
> You can choose the number displayed next to each label to navigate to the **All issues** page and see all issues that contain that label.

Deleting a label

You cannot currently delete an issues label in CodeCatalyst. If you remove a label from all issues, the label will appear in the **Unused labels** section in the **Labels** section of issue **settings**. Unused labels appear at the end of the list of labels when using filters or adding labels to an issue. You can find an overview of all labels (used and unused) and issues that have them in the issue **settings**.

Organizing work with custom fields

You can create custom fields to help organize and view the work for your project. Custom fields are added to the list of available filters in **Filter** so you can filter issues by custom fields. Custom fields are name and value pairs. You filter by the name of the custom field, and then the value of that custom field.

An issue can have multiple custom fields.

Creating a custom field

In CodeCatalyst, you create custom fields by either adding them when you create an issue or when you edit an existing issue. For more information, see Creating an issue in CodeCatalyst and Editing and collaborating on issues in CodeCatalyst.

Deleting a custom field

To delete a custom field, you must remove the custom field from each issue it is added to. When a custom field is deleted, you will no longer see the custom field in **Filter**. You can use filters to view all issues with a custom field, and remove them by editing the issues. For more information, see Finding and viewing issues and Editing an issue

Tracking work with custom statuses

You can add custom statuses on your board. Each custom status must belong to one of the following categories: **Draft**, **Not started**, **Started**, or **Completed**. Status categories are used to help organize statuses and populate default views. For more information about statuses and status categories, see Status and status categories and for more information about views, see Finding and viewing issues.

To create a status

1. In the navigation pane, choose **Issues**.
2. Choose **Active issues** to open the **issues view switcher** dropdown menu and choose **Settings**.
3. In **Statuses**, choose the plus icon next to the category you want the status to be in.
4. Name the status, then choose check mark icon.

> **Note**
>
> Choose the X icon to cancel adding a status.

The custom status is now visible on your board and shows as an option when creating an issue.

To edit a status

1. In the navigation pane, choose **Issues**.
2. Choose **Active issues** to open the **issues view switcher** dropdown menu and choose **Settings**.
3. In **Statuses**, choose the edit icon next to the status you want to edit or change.
4. Edit the status, then choose the check mark icon.

The edited status is now visible on your board.

To move a status

1. In the navigation pane, choose **Issues**.
2. Choose the ellipsis icon and choose **Settings**.
3. In **Statuses**, choose a status you want to move.
4. Drag and drop the status where you want it to be.

> **Note**
>
> You can only move a status within its designated category.

The statuses are now reordered on your board.

To deactivate a status

1. In the navigation pane, choose **Issues**.
2. Choose **Active issues** to open the **issues view switcher** dropdown menu and choose **Settings**.
3. In **Statuses**, choose a status you want to deactivate.
4. On the status you want to deactivate, choose the toggle on the status. The status is now grayed out.

> **Note**
>
> The deactivated status appears on the board until all issues are moved out of it. Issues cannot be added to a deactivated status.

5. To reactivate a deactivated status, choose the toggle on the status. The status is no longer grayed out.

> **Note**
>
> There must be at least one active status in each category. If there is only one status in the category, you cannot deactivate it.

Configuring issue effort estimation

Follow these steps to configure the setting for effort estimations for issues in CodeCatalyst.

To configure effort estimation for issues

1. In the navigation pane, choose **Issues**.
2. Choose **Active issues** to open the **issues view switcher** dropdown menu and choose **Settings**.
3. In **Estimation** in the **Basic settings** section, choose how the estimation values will be displayed. The types of estimates available are **T-shirt sizing**, **Fibonacci sequencing**, or **Hide estimates**. When the estimation type is updated, no data will be lost and the estimation value of all issues will be converted automatically. The conversion mapping is shown in the following table.

T-shirt size	Fibonacci sequence
XS	1
XS	2
S	3
M	5
L	8
XL	13

Enabling or disabling multiple assignees

Follow these steps to configure the setting for multiple assignees for issues in CodeCatalyst.

To enable or disable multiple assignees

1. In the navigation pane, choose **Issues**.
2. Choose **Active issues** to open the **issues view switcher** dropdown menu and choose **Settings**.

3. In **Assignee** in the **Basic settings** section, toggle the indicator to enable multiple assignees to be assigned to the same issue. An issue can have up to 10 assignees. If you do not enable this option, you will only be able to assign one assignee to an issue.

Creating an issues view

You can create views to quickly view issues that match a particular set of filters. This can help you save time and quickly view issues you have previously filtered, grouped, or sorted by.

To create an issues view

1. In the navigation pane, choose **Issues**.
2. (Optional) Depending on your use case, you may want to create a view from an existing view. To navigate to a different view, choose **Active issues** to open the **issues view switcher** dropdown menu and choose the view.
3. (Optional) Configure filters, grouping, and sorting before you create your view. You can add these while creating a view, but if you do it before, you can preview what is shown in the view before creating it.
4. Open the **issues view switcher** dropdown menu from the header bar. To create a board view where issues are viewed in columns based on status, choose the **+** in the **Board** column. To create a grid view where issues are viewed in a list, choose the **+** in the **Grid** column. You can change the type of view before it is created if you change you mind.
5. In the **Create view** dialog box, enter a **Name** for the view.
6. The **Filters**, **Group issues by**, and **Sort issues by** fields are filled based on the settings of the current view. Update them if necessary.
7. Choose **Create view** to create the view and be switched to it.

Quotas for issues in CodeCatalyst

The following table describes quotas and limits for issues in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst..

Resource	Default quota
Active issues	Maximum of 1,000 per project.

Resource	Default quota
Attachment size	Maximum of 500MB per attachment. Maximum total attachment storage is impacted by the overall storage limits for your space. For more information, see Pricing.
Total number of issues (active and archived)	Maximum of 100,000 per project.
Saved views	Maximum of 50 saved issue views per project.
Number of pull requests you can link to an issue	Maximum of 50 pull requests per issue.
Statuses (per project)	Maximum of 50 per project.
Statuses (per issue)	Maximum of 50 per issue.
Labels (per project)	Maximum of 200 per project.
Labels (per issue)	Maximum of 50 per issue.
Custom fields (per issue)	Maximum of 50 per issue.
Assignees	Maximum of 10 per issue.
Comments	Maximum of 1,000 per issue.
Tasks	Maximum of 100 per issue.

Configure identity, permissions, and access in CodeCatalyst

When you sign in to Amazon CodeCatalyst for the first time, you create an AWS Builder ID. AWS Builder IDs do not exist in AWS Identity and Access Management. The user name that you choose during your first sign-in becomes your unique user ID for your identity.

In CodeCatalyst, you can sign in for the first time in one of two ways:

- As part of creating a space.
- As part of accepting an invitation to a project or space in CodeCatalyst.

The *role* or roles associated with your identity determine the actions you can perform in CodeCatalyst. Project roles, such as **Project administrator** and **Contributor**, are specific to a project, so you can have one role in one project and a different role in another project. If you create a space, CodeCatalyst automatically assigns you the **Space administrator** role. When users accept invitations to a project, CodeCatalyst adds those identities to the space and assigns them the **Limited access** role. When you invite users to projects, you choose the role you want them to have in the project, which determines what actions they can and cannot take within the project. Most users working on a project only need the **Contributor** role to perform their tasks. For more information, see Granting access with user roles.

In addition to a project role, users in a project need a personal access token (PAT) to access source repositories for a project when using Git clients or integrated development environments (IDEs). Project members can use this PAT with third-party applications as an application-specific password associated with their CodeCatalyst identity. For example, when you clone a source repository to a local computer, you must provide a PAT as well as your CodeCatalyst user name.

You can configure access between CodeCatalyst and AWS resources by using a service role to perform actions such as accessing AWS CloudFormation stacks and resources when you deploy actions in workflows. You must configure access between CodeCatalyst and AWS resources for the workflow actions that are included with the project templates to run.

Topics

- Accessing GitHub resources with personal connections
- Configure your AWS Builder ID to sign in with multi-factor authentication (MFA)
- Security in Amazon CodeCatalyst
- Monitoring events and API calls using logging
- Quotas for identity, permission, and access in CodeCatalyst
- Troubleshooting

Granting access with user roles

In Amazon CodeCatalyst, you can assign roles to users at both the project level and the space level. In a project, a role specifies what a user is allowed to do in a project with the resources for that project. Users gain membership in a space when they join a project. You can add or remove users as administrators of a space. The **Space administrator** role has the broadest permissions of any role in CodeCatalyst. As a best practice, assign users the narrowest permissions necessary to perform their jobs.

You can assign roles to users in the space. You can also assign roles to users in the projects where they are members. Each user can only have one role in a project or space, but users can have different roles in each project and space. For example, a user might have the Project administrator role in one project and the Contributor role in another project.

Topics

- Understanding user roles for spaces and projects
- Viewing the permissions available for each role
- Viewing and changing user roles

Understanding user roles for spaces and projects

There are three roles available for a space:

- **Space administrator**
- **Power user**
- **Limited access**

Users who accept an invitation to a project have the **Limited access** role automatically assigned to them in the space that contains the project.

There are four roles available for members in a project:

- **Project administrator**
- **Contributor**
- **Reviewer**
- **Read only**

When you add a user to a project, CodeCatalyst automatically gives them the **Limited access** role. If you remove a user from all projects, CodeCatalyst automatically removes the Limited access role from that user.

Space administrator role

The **Space administrator** role is the most powerful role in CodeCatalyst. Only assign the **Space administrator** role to users who need to administer every aspect of a space, because this role has all permissions in CodeCatalyst. Users with the **Space administrator** role are the only users who can add or remove other users from the **Space administrator** role and delete the space.

When you create a space, CodeCatalyst automatically assigns you the **Space administrator** role. As a best practice, we recommend that you add this role to at least one other user who can act in this role in case the original space creator is unavailable.

Power user role

The **Power user** role is the second-most powerful role in CodeCatalyst spaces, but it has no access to projects in a space. It is designed for users who need to be able to create projects in a space and help manage the users and resources for the space. Assign the **Power user** role to users who are team leaders or managers who need the ability to create projects and manage users in the space as part of their work.

Limited access role

The **Limited access** access role is the role most users will have in CodeCatalyst spaces. It is the role automatically assigned to users when they accept an invitation to a project in a space. It provides the limited permissions they need to work within the space that contains that project. Assign the

Limited access role to users you invite directly to the space unless their work requires that they manage some aspect of the space.

Project administrator role

The **Project administrator** role is the most powerful role in a CodeCatalyst project. Only assign this role to users who need to administer every aspect of a project, including editing project settings, managing project permissions, and deleting projects.

Project roles do not have any permissions at the space level. Therefore, users with the **Project administrator** role cannot create additional projects. Only users with the **Space administrator** or **Power user** role can create projects.

> **Note**
>
> The **Space administrator** role has all permissions in CodeCatalyst.

Contributor role

The **Contributor** role is intended for the majority of members in a CodeCatalyst project. Assign this role to users who need to be able to work with code, workflows, issues, and actions in a project.

Reviewer role

The **Reviewer** role is intended for users who need to be able to interact with resources in a project, such as pull requests and issues, but not create and merge code, create workflows, or start or stop workflow runs in a CodeCatalyst project. Assign the **Reviewer** role to users who need to be able to approve and comment on pull requests, create, update, resolve, and comment on issues, and view code and workflows in a project.

Read only role

The **Read only** role is intended for users who need to view the resources and status of resources but not interact with them or contribute directly to the project. Users with this role cannot create resources in CodeCatalyst, but they can view them and copy them, such as cloning repositories and downloading attachments to issues to a local computer. Assign the **Read only** role to users who need to view resources and the state of the project, but not interact directly with it.

Viewing the permissions available for each role

The following table shows the permissions available for each CodeCatalyst role. Use the links to jump to the appropriate set of permissions.

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Space permissions							
Create space	✓	✗	✗	✗	✗	✗	✗
Edit space billing details	✓	✗	✗	✗	✗	✗	✗
Set up and enable	✓	✗	✗	✗	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
single sign-on							
Remove single sign-on	✓	✗	✗	✗	✗	✗	✗
Enable generative AI features for a space	✓	✗	✗	✗	✗	✗	✗
Disable generative AI features for a space	✓	✗	✗	✗	✗	✗	✗
Delete space	✓	✗	✗	✗	✗	✗	✗
Add other users to the Space administrator role	✓	✗	✗	✗	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Remove other users from the Space administrator role	✓	✗	✗	✗	✗	✗	✗
Create team	✓	✗	✗	✗	✗	✗	✗
Delete team	✓	✗	✗	✗	✗	✗	✗
Update team	✓	✗	✗	✗	✗	✗	✗
Disable machine resources for the space	✓	✗	✗	✗	✗	✗	✗
Enable machine resources for the space	✓	✗	✗	✗	✗	✗	✗
Create project	✓	✓	✗	✗	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Associate an AWS account connection with the space	✓	✓	✗	✗	✗	✗	✗
Update an AWS account connection	✓	✓	✗	✗	✗	✗	✗
Disassociate an AWS account connection from the space	✓	✓	✗	✗	✗	✗	✗
Delete an AWS account connection and remove it from the space	✓	✓	✗	✗	✗	✗	✗
Invite others to the space	✓	✓	✗	✗	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create VPC connection	✓	✓	✗	✗	✗	✗	✗
Edit VPC connection	✓	✓	✗	✗	✗	✗	✗
Delete VPC connection	✓	✓	✗	✗	✗	✗	✗
View logs of activity in the space	✓	✓	✗	✗	✗	✗	✗
View AWS account connections	✓	✓	✓	✓	✓	✓	✓
View incidents for CodeCatalyst	✓	✓	✓	✓	✓	✓	✓
View space	✓	✓	✓	✓	✓	✓	✓

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
View teams							
View VPC connections							
Extensions permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Install extensions							
Update extensions							
Delete extensions							
Connect a GitHub account							
Disconnect a GitHub account							
Connect a Jira site							

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Disconnect a Jira site							
View configuration details for installed extensions							
View extensions							
Project permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Edit project settings							
Disable machine resources for the project							

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Enable machine resources for the project	✓	✗	✗	✓	✗	✗	✗
Delete project	✓	✗	✗	✓	✗	✗	✗
Invite users to a project	✓	✗	✗	✓	✗	✗	✗
Change roles of users in a project	✓	✗	✗	✓	✗	✗	✗
Remove users from a project	✓	✗	✗	✓	✗	✗	✗
Add team to a project	✓	✗	✗	✓	✗	✗	✗
Remove team from a project	✓	✗	✗	✓	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Change project role of team							
View project							
View project activity							
View teams in project							
View blueprints							
Source repository permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create repositories							
Link repositories							

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Unlink repositories	✓	✗	✗	✓	✗	✗	✗
Delete repositories	✓	✗	✗	✓	✗	✗	✗
Edit repository settings	✓	✗	✗	✓	✓	✗	✗
View repositories	✓	✗	✗	✓	✓	✓	✓
View repository settings	✓	✗	✗	✓	✓	✓	✓
Clone repositories	✓	✗	✗	✓	✓	✓	✓
Create branches	✓	✗	✗	✓	✓	✗	✗
Create branch rules	✓	✗	✗	✓	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Change default branch	✓	✗	✗	✓	✗	✗	✗
Delete branches	✓	✗	✗	✓	✓	✗	✗
Merge branches	✓	✗	✗	✓	✓	✗	✗
Update branch rules	✓	✗	✗	✓	✗	✗	✗
View branches	✓	✗	✗	✓	✓	✓	✓
View branch rules	✓	✗	✗	✓	✓	✓	✓
Create folders	✓	✗	✗	✓	✓	✗	✗
Delete folders	✓	✗	✗	✓	✓	✗	✗
Edit folders	✓	✗	✗	✓	✓	✗	✗
View folders	✓	✗	✗	✓	✓	✓	✓
Create files	✓	✗	✗	✓	✓	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Delete files	✓	✗	✗	✓	✓	✗	✗
Edit files	✓	✗	✗	✓	✓	✗	✗
View files	✓	✗	✗	✓	✓	✓	✓
Create and push commits	✓	✗	✗	✓	✓	✗	✗
View commits	✓	✗	✗	✓	✓	✓	✓
Create pull requests	✓	✗	✗	✓	✓	✗	✗
Create approval rules for pull requests	✓	✗	✗	✓	✗	✗	✗
Override merge requirements for pull requests	✓	✗	✗	✓	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Update pull requests	✓	✗	✗	✓	✓	✗	✗
Update approval rules for pull requests	✓	✗	✗	✓	✗	✗	✗
View pull requests	✓	✗	✗	✓	✓	✓	✓
View approval rules for pull requests	✓	✗	✗	✓	✓	✓	✓
Close pull requests	✓	✗	✗	✓	✓	✗	✗
Approve pull requests	✓	✗	✗	✓	✓	✓	✗
Comment on pull requests	✓	✗	✗	✓	✓	✓	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Interact with Amazon Q in comments on pull requests							
Create a revision for a pull request created by Amazon Q							
Link an issue to a pull request							
Unlink an issue from a pull request							
Dev Environment permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create your own Dev Environment	✓	✗	✗	✓	✓	✓	✗
Stop your own Dev Environment	✓	✗	✗	✓	✓	✓	✗
Stop Dev Environments created by other users	✓	✓	✗	✓	✗	✗	✗
Resume your own Dev Environment	✓	✗	✗	✓	✓	✓	✗
View your own Dev Environments	✓	✗	✗	✓	✓	✓	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
View Dev Environments created by other users	✓	✓	✗	✓	✗	✗	✗
Edit your own Dev Environment	✓	✗	✗	✓	✓	✓	✗
Edit Dev Environments created by other users	✓	✗	✗	✗	✗	✗	✗
Delete your own Dev Environment	✓	✗	✗	✓	✓	✓	✗
Delete Dev Environments created by other users	✓	✓	✗	✗	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create a devfile for Dev Environments							
Edit a devfile for Dev Environments							
Delete a devfile for Dev Environments							
View a devfile for Dev Environments							
Package repository and package permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create package repository	✓	✗	✗	✓	✗	✗	✗
View package repositories	✓	✗	✗	✓	✓	✓	✓
Edit package repository	✓	✗	✗	✓	✗	✗	✗
Delete package repository	✓	✗	✗	✓	✗	✗	✗
Create gateway package repository	✓	✗	✗	✓	✗	✗	✗
View gateway package repositories	✓	✗	✗	✓	✓	✓	✓

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Delete gateway package repository	✓	✗	✗	✓	✗	✗	✗
Add upstream package repository	✓	✗	✗	✓	✗	✗	✗
Edit search order of upstream repositories	✓	✗	✗	✓	✗	✗	✗
Remove upstream package repository	✓	✗	✗	✓	✗	✗	✗
Connect to a package repository	✓	✗	✗	✓	✓	✓	✓

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Read packages from a package repository	✓	✗	✗	✓	✓	✓	✓
Publish packages to a package repository	✓	✗	✗	✓	✓	✗	✗
Read and retain packages from an upstream repository	✓	✗	✗	✓	✓	✓	✓
View packages	✓	✗	✗	✓	✓	✓	✓
View package versions	✓	✗	✗	✓	✓	✓	✓
View package version assets	✓	✗	✗	✓	✓	✓	✓

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
List package version dependencies							
Update package version status							
Update package origin configuration							
Delete package version							
Workflow permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create workflow							
Update workflow							
Delete workflow							

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Start workflow	✓	✗	✗	✓	✓	✗	✗
Stop workflow	✓	✗	✗	✓	✓	✗	✗
Create workflow secrets	✓	✗	✗	✓	✓	✗	✗
Update workflow secrets	✓	✗	✗	✓	✓	✗	✗
Delete workflow secrets	✓	✗	✗	✓	✓	✗	✗
Create environments	✓	✗	✗	✓	✓	✗	✗
Delete environments	✓	✗	✗	✓	✗	✗	✗
Create fleet	✓	✗	✗	✓	✗	✗	✗
Update fleet	✓	✗	✗	✓	✗	✗	✗
Delete fleet	✓	✗	✗	✓	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Manage workflow resources in other accounts	✓	✗	✗	✓	✗	✗	✗
Associate a default IAM role with a environment	✓	✗	✗	✓	✗	✗	✗
Associate a VPC connection with a environment	✓	✗	✗	✓	✗	✗	✗
Disassociate a VPC connection with a environment	✓	✗	✗	✓	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Associate a VPC-connected environment with a workflow							
Disassociate a VPC-connected environment with a workflow							
Approve workflow runs							
Track a commit in a workflow							
View environments							
View build action logs							

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
View fleets							
View test action logs							
View workflow							
View workflow runs							
View workflow run results							
View workflow secrets							

Issues permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create issue							
Update issue							
View issues							

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create task	✓	✗	✗	✓	✓	✓	✗
Update task	✓	✗	✗	✓	✓	✓	✗
View tasks	✓	✗	✗	✓	✓	✓	✓
Archive an issue	✓	✗	✗	✓	✓	✓	✗
Assign an issue to Amazon Q	✓	✗	✗	✓	✓	✗	✗
Interact with Amazon Q in comments on an issue	✓	✗	✗	✓	✓	✗	✗
Unassign Amazon Q from an issue	✓	✗	✗	✓	✓	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Recommend tasks for an issue with Amazon Q	✓	✗	✗	✓	✓	✗	✗
Create tasks recommended by Amazon Q	✓	✗	✗	✓	✓	✗	✗
Update issues created by other users	✓	✗	✗	✓	✓	✓	✗
View comments on an issue	✓	✗	✗	✓	✓	✓	✓
Create a comment on an issue	✓	✗	✗	✓	✓	✗	✗
Update a comment on an issue	✓	✗	✗	✓	✓	✓	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create a label	✓	✗	✗	✓	✓	✗	✗
Update a label	✓	✗	✗	✓	✓	✗	✗
View labels	✓	✗	✗	✓	✓	✓	✓
Add a label to an issue	✓	✗	✗	✓	✓	✓	✗
Remove a label from an issue	✓	✗	✗	✓	✓	✓	✗
Create a custom status for issues	✓	✗	✗	✓	✓	✗	✗
Update a custom status	✓	✗	✗	✓	✓	✗	✗
View a custom status	✓	✗	✗	✓	✓	✓	✓
Move a custom status	✓	✗	✗	✓	✓	✓	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Deactivate a custom status	✓	✗	✗	✓	✓	✓	✗
Add an attachment to an issue	✓	✗	✗	✓	✓	✓	✗
View an issue attachment	✓	✗	✗	✓	✓	✓	✓
Remove an attachment from an issue	✓	✗	✗	✓	✓	✓	✗
Link an issue to another issue	✓	✗	✗	✓	✓	✓	✗
Unlink an issue from another issue	✓	✗	✗	✓	✓	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Update an issue link							
View links for an issue							
Link a pull request to an issue							
Unlink a pull request from an issue							
Link a Jira project							
Unlink a Jira project							
Blueprint permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Create custom blueprint project	✓	✓	✗	✗	✗	✗	✗
Publish a preview custom blueprint	✓	✓	✗	✗	✗	✗	✗
Publish a custom blueprint	✓	✓	✗	✗	✗	✗	✗
Add a custom blueprint to a space blueprints catalog	✓	✓	✗	✗	✗	✗	✗
Remove a custom blueprint from a space blueprints catalog	✓	✓	✗	✗	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Manage the publishing permissions for a custom blueprint	✓	✓	✗	✗	✗	✗	✗
Manage the catalog version for a custom blueprint	✓	✓	✗	✗	✗	✗	✗
Update a custom blueprint	✓	✓	✗	✗	✗	✗	✗
Delete a custom blueprint version	✓	✓	✗	✗	✗	✗	✗
Delete a custom blueprint	✓	✓	✗	✗	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Convert a source repository to a custom blueprint	✓	✓	✗	✓	✗	✗	✗
Add a custom blueprint to a project	✓	✓	✗	✓	✗	✗	✗
Disassociate a custom blueprint from a project	✓	✓	✗	✓	✗	✗	✗
Update the version of an applied custom blueprint	✓	✓	✗	✓	✗	✗	✗
Edit the settings of a custom blueprint	✓	✓	✗	✓	✗	✗	✗

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
View published custom blueprints							
Notifications permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Configure notification channel							
Remove notification channel							
Edit notification settings							
View notification settings							

Permission	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Automatically receive notifications about CodeCatalyst incidents							
Configure email notifications for your associated email account							

Search permissions	Space administrator role	Power user role	Limited access role	Project administrator role	Contributor role	Reviewer role	Read only role
Search inside a project							
Search across the space							

Viewing and changing user roles

You can view the role assigned to a user. This helps you understand what actions they can take in a project. You can also change their role if they need additional permissions.

To view the role of a user in a project

1. Navigate to the project where you want to view the roles associated with each project member.

   > **Tip**
   >
   > You can choose which project to view in the top navigation bar.

2. In the navigation pane, choose **Project settings**.
3. On the **Members** tab, the role for each project member is displayed in **Role**.

To change users' roles in a project

1. Navigate to the project where you want to change the roles associated with project members.

   > **Tip**
   >
   > You can choose which project to view in the top navigation bar.

2. In the navigation pane, choose **Project settings**.
3. On the **Members** tab, in **Project members**, choose the users whose roles you want to change. Choose **Action**, and then choose **Edit role**.
4. In **Role**, choose the project role, and then choose **Confirm**.

Viewing and changing roles in the space

All users who accept invitations to a project in CodeCatalyst become members of the project's space. You can view the list of space members. You can change users' roles from **Limited access** to **Space administrator** to better manage your space and its resources. The **Space administrator** role is the only role that allows users to create projects in CodeCatalyst.

Warning

The **Space administrator** role is the most powerful role in CodeCatalyst. Users with this role can perform any action in CodeCatalyst, including deleting the space. Only assign this role to users who require this level of access to your space. For more information, see Space administrator role.

To change a user's role in the space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to the space.

   Tip

   If you belong to more than one space, you can choose which space to view in the top navigation bar.

3. Choose the **Members** tab.
4. Choose the user whose role you want to change, and then choose **Change role**.
5. In **Change role**, choose the role you want to assign, and then choose **Confirm**.

Grant users repository access with personal access tokens

To access some CodeCatalyst resources, such as source repositories, on a local computer with a Git client or integrated development environment (IDE), you must enter an application-specific password. You can create a personal access token (PAT) to use for this purpose. PATs you create are associated with your user identity across all spaces and projects in CodeCatalyst. You can create more than one PAT for your CodeCatalyst identity.

You can view the names and expiration dates of the PATs you have created, and you can delete those you no longer need. You can only copy the PAT secret at the time you create it.

Note

By default, PATs expire in 1 year.

Creating PATs

PATs are associated with your user identity in CodeCatalyst. You can only copy a PAT secret at the time you create it.

Creating PATs (console)

You can use the console to create PATs in CodeCatalyst.

To create a personal access token (console)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the top menu bar, choose your profile badge, and then choose **My settings**. The CodeCatalyst **My settings** page opens.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

3. Under **Personal access tokens**, choose **Create**.

   The **Create PAT** page displays.
4. In **PAT name**, enter a descriptive name for your PAT.
5. In **Expiration date**, keep the default date or choose the calendar icon to select a custom date. The expiration date defaults to 1 year from the current date.
6. Choose **Create**.

   > **Tip**
   >
   > You can also create this token when you choose **Clone repository** for a source repository.

7. To copy the PAT secret, choose **Copy**. Store the PAT secret where you will be able to retrieve it.

> **Important**
>
> The PAT secret only displays once. You cannot retrieve it after you close the window. If you did not save the PAT secret in a secure location, you can create another one.

.

Creating PATs (CLI)

You can use the CLI to create PATs in CodeCatalyst.

To create a personal access token (AWS CLI)

1. At the terminal or command line, run the **create-access-token** command as follows.

   ```
   aws codecatalyst create-access-token
   ```

   If successful, the command returns information about the created PAT like the following example.

   ```
   {
       "secret": "value",
       "name": "marymajor-22222EXAMPLE",
       "expiresTime": "2024-02-04T01:56:04.402000+00:00"
   }
   ```

2.

You can only view the PAT secret once—when you create the PAT. If you've misplaced a PAT secret or you're concerned that it's not stored securely, you can create another one.

You can view the PATs associated with your user account by using the AWS CLI. You can only view information about the PAT, and not the value of the PAT secret itself.

> **Note**
>
> Make sure that you're using a recent version of the AWS CLI to work with CodeCatalyst. Earlier versions might not contain the CodeCatalyst commands. You must configure your

AWS CLI profile before you can use it with CodeCatalyst. For more information, see Setting up to use the AWS CLI with CodeCatalyst.

Viewing PATs

You can view PATs in CodeCatalyst. The list shows all of the PATs that you have associated with your user identity. Your PAT is associated with your user profile across all spaces and projects in CodeCatalyst. Expired PATs do not display because they're deleted after they expire.

Viewing PATs (console)

You can use the console to view PATs associated with your user identity in CodeCatalyst.

To view your personal access tokens (console)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the top menu bar, choose your profile badge, and then choose **My settings**. The CodeCatalyst **My settings** page opens.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

3. Under **Personal access tokens**, view the names and expiration dates of your current PATs.

Viewing PATs (CLI)

You can use the CLI to view PATs associated with your user identity in CodeCatalyst.

To view your personal access tokens (AWS CLI)

- At the terminal or command line, run the **list-access-tokens** command as follows.

  ```
  aws codecatalyst list-access-tokens
  ```

  If successful, the command returns information about the PATs associated with your user account like the following example.

```
{
    "items": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "name": "marymajor-22222EXAMPLE",
            "expiresTime": "2024-02-04T01:56:04.402000+00:00"
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
            "name": "marymajor-11111EXAMPLE",
            "expiresTime": "2023-03-12T01:58:40.694000+00:00"
        }
    ]
}
```

Deleting PATs

You can delete PATs associated with your user identity in CodeCatalyst.

Deleting PATs (console)

You can use the console to delete PATs in CodeCatalyst.

To delete a personal access token (console)

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the top menu bar, choose your profile badge, and then choose **My settings**. The CodeCatalyst **My settings** page opens.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

3. Under **Personal access tokens**, choose the selector next to the PAT you want to delete, and then choose **Delete**.

   On the **Delete PAT: <name>?** page, to confirm deletion, type *delete* in the text field. Choose **Delete**.

Deleting PATs (CLI)

You can delete a PAT associated with your user identity by using the AWS CLI. To do this, you must supply the ID for the PAT, which you can view by using the **delete-access-token** command.

> **Note**
>
> Make sure that you're using a recent version of the AWS CLI to work with CodeCatalyst. Earlier versions might not contain the CodeCatalyst commands. For more information about using the AWS CLI with CodeCatalyst, see Setting up to use the AWS CLI with CodeCatalyst.

To delete a personal access token (AWS CLI)

- At the terminal or command line, run the **delete-access-token** command, providing the ID for the PAT you want to delete. For example, run the following command to delete a PAT with an ID of *123EXAMPLE*.

  ```
  aws codecatalyst delete-access-token --id a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb
  ```

  If successful, this command returns no response.

Accessing GitHub resources with personal connections

You can use personal connections to authorize and connect your third-party GitHub resources with CodeCatalyst. For example, use a personal connection to authorize CodeCatalyst to access your GitHub account and create a repository as the source for your project or blueprint. The connection is mapped to your CodeCatalyst identity and can be used to connect to one or more source repositories. Connections you create are associated with your user identity across all spaces and projects in CodeCatalyst.

> **Note**
>
> You can manage personal connections with blueprints in GitHub organizations where you have access to do so.

You can create one personal connection for one user identity (CodeCatalyst alias) across all spaces, per provider type.

You can use your personal connections in CodeCatalyst to create a GitHub repository for a project, choose a GitHub source repository for a blueprint, and manage pull requests in CodeCatalyst for your GitHub repository.

> **Note**
>
> The use of personal connections for associating blueprints with a GitHub repository is not the same as use of extensions in CodeCatalyst to link a GitHub repository. For more information about extensions, see Add functionality to projects with extensions in CodeCatalyst.

Creating personal connections

You can use the console to create a personal connection associated with your user identity in CodeCatalyst.

To create a personal connection

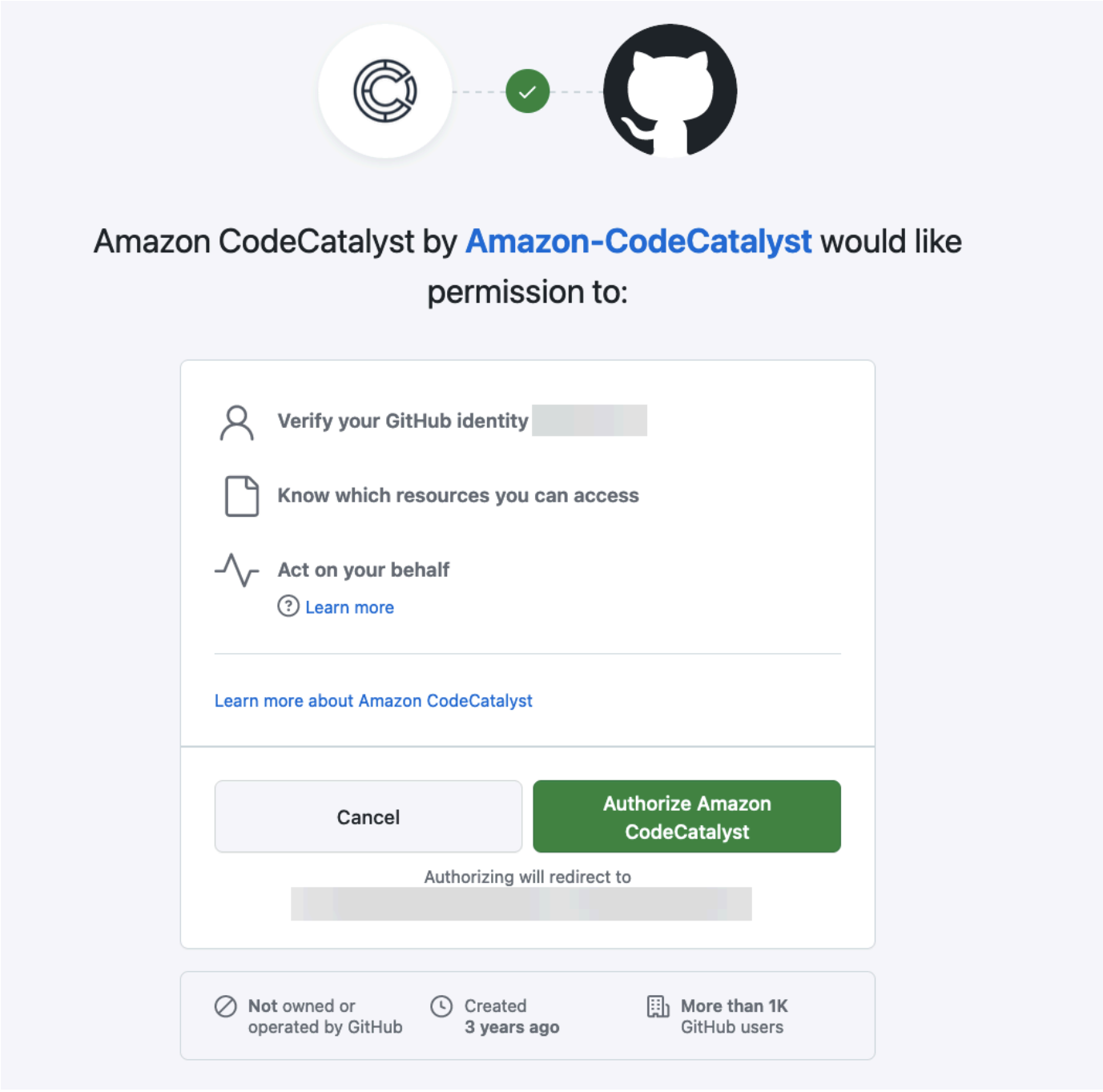
1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the top menu bar, choose your profile badge, and then choose **My settings**. The CodeCatalyst **My settings** page opens.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

3. Under **Personal connections**, choose **Create**.

   The **Create connection** page displays.
4. Choose **Create**. A **Create connection** page displays.
5. On the **Create connection** page, in **Provider**, choose **GitHub**. In **Connection name**, type a name for your connection. Choose **Create**.
6. If prompted, sign in to your GitHub account.

7. On the connection confirmation page, choose **Accept**.
8. On the installation confirmation page, choose the authorization button to confirm that you want to install the connector application.



Deleting personal connections

You can delete a personal connection associated with your user identity in CodeCatalyst.

> **Note**
>
> Deleting the personal connection in CodeCatalyst does not uninstall the application in your GitHub account. If you create a new personal connection, the app installation can be used. To uninstall the application in GitHub, you can revoke the application and reinstall at a later time.

To delete a personal connection in CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the top menu bar, choose your profile badge, and then choose **My settings**. The CodeCatalyst **My settings** page opens.

   > **Tip**
   >
   > You can also find your user profile by going to the members page for a project or space and choosing your name from the members list.

3. Under **Personal connections**, choose the selector next to the connection you want to delete, and then choose **Delete**.

   On the **Delete connection: <name>?** page, to confirm deletion, type *delete* in the text field. Choose **Delete**.

1. Sign in to GitHub and navigate to your account settings for installed apps. Choose your profile icon, choose **Settings**, and then choose **Applications**.
2. On the **Authorized GitHub Apps** tab, in the list of authorized applications, view the app installed for CodeCatalyst. To revoke the installation, choose **Revoke**.

Configure your AWS Builder ID to sign in with multi-factor authentication (MFA)

Whether you created your AWS Builder ID profile for personal use or professional use, we encourage configuring multi-factor authentication (MFA) as another layer of security. We especially recommend configuring MFA if you're a member of a space and collaborate with others on projects.

Because more than one person can have access to a project, more opportunities exist for security breaches.

When you enable MFA, you must sign in to Amazon CodeCatalyst with your email and password. This portion of signing in is the first factor, where you use something that you know. You then sign in with either a code or security key. This is the second factor, which is something that you have. The second factor could be an authentication code that is generated either by your mobile device or by tapping or pressing a security key connected to your computer. Taken together, these multiple factors provide increased security by preventing unauthorized access.

How to register a device for use with multi-factor authentication

Use the following procedure on **My profile** > **Multi-factor authentication** to register your new device for multi-factor authentication (MFA).

> **Note**
>
> We recommend that you first download the appropriate authenticator app onto your device before starting the steps in this procedure. For a list of apps that you can use for MFA devices, see Authenticator applications.

To register your device for use with MFA

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. At the top right, choose the arrow next to the icon with your first initial, and then choose **User profile**. The CodeCatalyst **Profile** page opens.
3. On the profile page, choose **Manage profile and security**. The **AWS Builder ID** profile page opens.
4. On the left side of the page, choose **Security**.
5. On the **Multi-factor authentication** page, choose **Register device**.
6. On the **Register MFA device** page, choose one of the following MFA device types, and follow the instructions:

   - **Security key** or **Built-in authenticator**
     1. On the **Register your user's security key** page, follow the instructions given to you by your browser or platform.

> **Note**
>
> This experience varies based on your operating system and browser, so follow the instructions displayed by your browser or platform. After your device has been successfully registered, you will be given the option to associate a friendly display name to your newly enrolled device. If you want to change this, choose **Rename**, enter the new name, and then choose **Save**.

- **Authenticator app**

  1. On the **Set up the authenticator app** page, you might notice configuration information for the new MFA device, including a QR code graphic. The graphic is a representation of the secret key that is available for manual entry on devices that do not support QR codes.
  2. Using the physical MFA device, do the following:

     a. Open a compatible MFA authenticator app. For a list of tested apps that you can use with MFA devices, see Tested authenticator apps. If the MFA app supports multiple devices, choose the option to create a new MFA device.

     b. Determine whether the MFA app supports QR codes, and then do one of the following on the **Set up the authenticator app** page:

        i. Choose **Show QR code**, and then use the app to scan the QR code. For example, you might choose the camera icon or choose an option similar to **Scan code**. Then use the device's camera to scan the code.

        ii. Choose **show secret key**, and then enter that secret key into your MFA app.

        > **Important**
        >
        > When you configure an MFA device for AWS Builder ID, save a copy of the QR code or secret key in a secure place. This can help if you lose the phone or have to reinstall the MFA authenticator app. If either of those things happen, you can quickly reconfigure the app to use the same MFA configuration.

  3. On the **Set up the authenticator app** page, under **Authenticator code**, enter the one-time password that currently appears on the physical MFA device.

> **Important**
>
> Submit your request immediately after generating the code. If you generate the code and then wait too long to submit the request, the MFA device is successfully associated with your AWS Builder ID profile, but the MFA device is out of sync. This happens because time-based one-time passwords (TOTP) expire after a short period of time. If this happens, you can resync the device.

4. Choose **Assign MFA**. The MFA device can now start generating one-time passwords and is now ready for use.

Authenticator applications

Authenticator apps are one-time password (OTP)–based third party-authenticators. Users can use an authenticator application installed on their mobile device or tablet as an authorized MFA device. The third-party authenticator application must be compliant with RFC 6238, which is a standards-based TOTP (time-based one-time password) algorithm capable of generating six-digit authentication codes.

When prompted for MFA, users must enter a valid code from their authenticator app within the input box presented. Each MFA device assigned to a user must be unique. Two authenticator apps can be registered for any given user.

Tested authenticator apps

Although any TOTP-compliant application will work with IAM Identity Center MFA, the following table lists well-known third-party authenticator apps to choose from.

Operating system	Tested authenticator app
Android	Authy, Duo Mobile, LastPass Authenticator, Microsoft Authenticator, Google Authenticator
iOS	Authy, Duo Mobile, LastPass Authenticator, Microsoft Authenticator, Google Authenticator

Changing your MFA devices

After you register an MFA device, you can change its name or delete it. We recommend always having at least one MFA device enabled for an extra layer of security. You can have up to five devices registered. To find out how to add more, see How to register a device for use with multi-factor authentication.

Renaming an MFA device

To rename your MFA device

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. At the top right, choose the arrow next to the icon with your first initial, and then choose **User profile**. The CodeCatalyst **Profile** page opens.
3. On the profile page, choose **Manage profile and security**. The **AWS Builder ID** profile page opens.
4. Choose **Multi-factor authentication** on the left side of the page. You'll see that **Rename** is grayed out when you arrive at the page.
5. Select the MFA device that you want to change. Choose **Rename**. Then a modal pops up.
6. In the prompt that opens, enter the new name in **MFA device name**, and then choose **Rename**. The renamed device appears under **Multi-factor authentication devices (MFA)**.

Deleting an MFA device

To delete an MFA device

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. At the top right, choose the arrow next to the icon with your first initial, and then choose **User profile**. The CodeCatalyst **Profile** page opens.
3. On the profile page, choose **Manage profile and security**. The **AWS Builder ID** profile page opens.
4. Choose **Multi-factor authentication** on the left side of the page. You'll see that **Delete** is grayed out when you arrive at the page.
5. Select the MFA device that you want to change. Choose **Delete**. A modal appears that says **Delete MFA device?**. Follow the instructions to delete your device.

6. Choose **Delete**. The deleted device no longer appears under **Multi-factor authentication devices (MFA)**.

Security in Amazon CodeCatalyst

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive spaces.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to CodeCatalyst, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS services that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon CodeCatalyst. It shows you how to configure CodeCatalyst to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CodeCatalyst resources.

Contents

- Understanding the CodeCatalyst trust model

Data protection in Amazon CodeCatalyst

Security and Compliance is a shared responsibility between Amazon CodeCatalyst and the customer, just as the AWS shared responsibility model applies to your use of AWS resources used in a workflow. As described in this model, CodeCatalyst is responsible for protecting the global infrastructure for the service. You are responsible for maintaining control over your content that is hosted on this infrastructure. This shared responsibility model applies to data protection in CodeCatalyst.

For data protection purposes, we recommend that you protect your account credentials, and that you set up multi-factor authentication when signing in. For more information, see Configure your AWS Builder ID to sign in with multi-factor authentication (MFA).

Do not enter confidential or sensitive information, such as your customers' email addresses, in tags or free-form fields such as a **Name** field. This includes resource names and any other identifiers you enter in CodeCatalyst in addition to any connected AWS accounts. For example, do not enter confidential or sensistive information as part of space, project, or deployment fleet names. Any data that you enter in tags, names, or free-form fields used for names might be used for billing or diagnostic logs or could be included in URL paths. This applies to using the console, API, AWS CLI, the CodeCatalyst Action Development Kit, or any AWS SDKs.

If you provide a URL to an external server, we strongly recommend that you do not include any security credentials information in the URL to validate your request to that server.

CodeCatalyst source repositories are automatically encrypted at rest. No customer action is required. CodeCatalyst also encrypts repository data in transit using the HTTPS protocol.

CodeCatalyst supports MFA. For more information, see Configure your AWS Builder ID to sign in with multi-factor authentication (MFA).

Data encryption

CodeCatalyst securely stores and transfers data within the service. All data is encrypted in transit and at rest. Any data created or stored by the service, including any metadata for the service, is stored natively in the service and encrypted.

> **Note**
>
> While information about issues is stored securely within the service, information about open issues is also stored in the local cache of the browser where you viewed issue boards, backlogs, and individual issues. For optimal security, be sure to clear your browser cache to remove this information.

If you use resources linked to CodeCatalyst, such as an account connection to an AWS account or a linked repository in GitHub, data in transit from CodeCatalyst to that linked resource is encrypted, but the data handling in that linked resource is managed by that linked service. For more information, see the documentation for the linked service and Best practices for workflow actions in Amazon CodeCatalyst.

Key management

CodeCatalyst does not support key management.

Inter-network traffic privacy

When you create a space in CodeCatalyst, you choose the AWS Region where the data and resources will be stored for that space. Project data and metadata never leaves that AWS Region. However, to support navigation within CodeCatalyst, a limited set of space, project, and user metadata is replicated across all AWS Regions in the partition. It will not be replicated to AWS Regions outside of that partition. For example, if you choose **US West (Oregon)** as the AWS Region when you create your space, your data will not be replicated to Regions in China Regions or AWS GovCloud (US). For more information, see Managing AWS Regions, AWS Global Infrastructure, and AWS service endpoints.

Data replicated across AWS Regions inside a partition includes:

- An encrypted hash value that represents the name of the space in order to ensure the uniqueness of space names. This value is not human-readable and does not expose the actual names of spaces
- The unique ID of the space
- Metadata for the space that assists in the navigation across spaces
- The AWS Region where the space is located
- The unique IDs of all projects in the space

- The role ID that indicates a user's role in a space or project
- When signing up for CodeCatalyst, data and metadata about the signup process, including:
  - The unique ID of the AWS Builder ID
  - The display name for the user in their AWS Builder ID
  - The alias of the user in their AWS Builder ID
  - The email address used when the user signed up for their AWS Builder ID
  - The progress of the sign up process
  - If creating a space as part of the sign up process, the AWS account ID that is used as the billing account for the space

Space names are unique across CodeCatalyst. Be sure not to include sensitive data in the name of the space.

When working with linked resources and connected accounts such as a connection to an AWS account or a GitHub repository, we recommend configuring your source and destination locations with the highest level of security that each one supports. CodeCatalyst secures the connection between AWS accounts, AWS Regions, and Availability Zones by using Transport Layer Security (TLS) 1.2.

Identity and Access Management and Amazon CodeCatalyst

In Amazon CodeCatalyst, you create and use an AWS Builder ID in order to sign in and access your spaces and projects. An AWS Builder ID is not an identity in AWS Identity and Access Management (IAM) and does not exist in an AWS account. However, CodeCatalyst does integrate with IAM when verifying a space for billing purposes, and when connected to an AWS account to create and use resources in that AWS account.

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use resources. IAM is an AWS service that you can use with no additional charge.

When you create a space in Amazon CodeCatalyst, you must connect an AWS account as the billing account for your space. You must have administrator permissions in the AWS account to verify the CodeCatalyst space, or have the permission. You also have the option to add an IAM role for your space that CodeCatalyst can use to create and access resources in that connected AWS account.

This is called a service role. You can choose to create connections to more than one AWS account and create service roles for CodeCatalyst in each of those accounts.

> **Note**
>
> Billing for CodeCatalyst takes place in the AWS account designated as the billing account. However, if you create a CodeCatalyst service role in that AWS account or in any other connected AWS account, resources created and used by the CodeCatalyst service role will be billed in that connected AWS account. For more information, see Managing billing in the Amazon CodeCatalyst Administrator Guide.

Topics

Identity-based policies in IAM

Identity-based policies are JSON permissions policy documents that you can attach to an identity. That identity could be a user, a group of users, or a role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all

of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for CodeCatalyst

To view examples of CodeCatalyst identity-based policies, see Identity-based policy examples for CodeCatalyst connections.

Policy actions in IAM

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform which **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "prefix:action1",
      "prefix:action2"
         ]
```

Policy resources in IAM

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform which **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Policy condition keys in IAM

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform which **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Identity-based policy examples for CodeCatalyst connections

In CodeCatalyst, AWS accounts are required to manage billing for a space and to access resources in project workflows. An account connection is used to authorize adding AWS accounts to a space. Identity-based polices are used in the connected AWS accounts.

By default, users and roles don't have permission to create or modify CodeCatalyst resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies for users that require them.

The following example IAM policies grant permissions for actions related to account connections. Use them to limit access for connecting accounts to CodeCatalyst.

Example 1: Allow a user to accept connection requests in a single AWS Region

The following permissions policy only allows users to view and accept requests for connections between CodeCatalyst and AWS accounts. In addition, the policy uses a condition to only allow the actions in the us-west-2 Region and not from other AWS Regions. To view and approve the request,

the user signs in to the AWS Management Console with the same account as that specified in the request.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:AcceptConnection",
        "codecatalyst:GetPendingConnection"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": "us-west-2"
        }
      }
    }
  ]
}
```

Example 2: Allow managing connections in the console for a single AWS Region

The following permissions policy allows users to manage connections between CodeCatalyst and AWS accounts in a single Region. The policy uses a condition to only allow the actions in the us-west-2 Region and not from other AWS Regions. After you create a connection, you can create the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role by choosing the option in the AWS Management Console. In the example policy, the condition for the `iam:PassRole` action includes the service principals for CodeCatalyst. Only roles with that access will be created in the AWS Management Console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecatalyst:*"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:RequestedRegion": "us-west-2"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "codecatalyst.amazonaws.com",
                        "codecatalyst-runner.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Example 3: Deny managing connections

The following permissions policy denies users any ability to manage connections between CodeCatalyst and AWS accounts.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
```

```
            "Action": [
                "codecatalyst:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Using tags to control access to account connection resources

Tags can be attached to the resource or passed in the request to services that support tagging. Resources in policies can have tags, and some actions in policies can include tags. Tagging condition keys include the `aws:RequestTag` and `aws:ResourceTag` condition keys. When you create an IAM policy, you can use tag condition keys to control the following:

- Which users can perform actions on a connection resource, based on tags that it already has.
- Which tags can be passed in an action's request.
- Whether specific tag keys can be used in a request.

The following examples demonstrate how to specify tag conditions in policies for CodeCatalyst account connections users. For more information about condition keys, see Policy condition keys in IAM.

Example 1: Allow actions based on tags in the request

The following policy grants users permission to approve account connections.

To do that, it allows the `AcceptConnection` and `TagResource` actions if the request specifies a tag named `Project` with the value `ProjectA`. (The `aws:RequestTag` condition key is used to control which tags can be passed in an IAM request.) The `aws:TagKeys` condition ensures tag key case sensitivity.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:AcceptConnection",
        "codecatalyst:TagResource"
```

```
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 2: Allow actions based on resource tags

The following policy grants users permission to perform actions on, and get information about, account connection resources.

To do that, it allows specific actions if the connection has a tag named `Project` with the value `ProjectA`. (The `aws:ResourceTag` condition key is used to control which tags can be passed in an IAM request.)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:GetConnection",
        "codecatalyst:DeleteConnection",
        "codecatalyst:AssociateIamRoleToConnection",
        "codecatalyst:DisassociateIamRoleFromConnection",
        "codecatalyst:ListIamRolesForConnection",
        "codecatalyst:PutBillingAuthorization"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "ProjectA"
        }
      }
    }
```

```
  ]
}
```

CodeCatalyst permissions reference

This section provides a permissions reference for actions used with the account connection resource for AWS accounts that are connected to CodeCatalyst. The following section describes permissions-only actions that are related to connecting accounts.

Required permissions for account connections

The following permissions are required for working with account connections.

CodeCatalyst permissions for account connections	Required permissions	Resources
AcceptConnection	Required to accept a request to connect this account to a CodeCatalyst space. This is an IAM policy permission only, not an API action.	Supports only a wildcard (*) in the policy `Resource` element.
AssociateIamRoleToConnection	Required to associate an IAM role to an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*
DeleteConnection	Required to delete an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*
DisassociateIamRoleFromConnection	Required to disassociate an IAM role from an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*

CodeCatalyst permissions for account connections	Required permissions	Resources
GetBillingAuthorization	Required to describe the billing authorization for an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*
GetConnection	Required to get an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*
GetPendingConnection	Required to get a pending request to connect this account to a CodeCatalyst space. This is an IAM policy permission only, not an API action.	Supports only a wildcard (*) in the policy `Resource` element.
ListConnections	Required to list account connections that are not pending. This is an IAM policy permission only, not an API action.	Supports only a wildcard (*) in the policy `Resource` element.
ListIamRolesForConnection	Required to list IAM roles associated with an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*
ListTagsForResource	Required to list tags associated with an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*

CodeCatalyst permissions for account connections	Required permissions	Resources
PutBillingAuthorization	Required to create or update the billing authorization for an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*
RejectConnection	Required to reject a request to connect this account to a CodeCatalyst space. This is an IAM policy permission only, not an API action.	Supports only a wildcard (*) in the policy `Resource` element.
TagResource	Required to create or edit tags associated with an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*
UntagResource	Required to remove tags associated with an account connection. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *account_ID* `:/connections/` *connection_ID*

Required permissions for IAM Identity Center applications

The following permissions are required for working with IAM Identity Center applications.

CodeCatalyst permissions for IAM Identity Center applications	Required permissions	Resources
AssociateIdentityCenterApplicationToSpace	Required to associate an IAM Identity Center applicati	`arn:aws:codecatalyst:region:` *account_I*

CodeCatalyst permissions for IAM Identity Center applications	Required permissions	Resources
	on with a CodeCatalyst space. This is an IAM policy permission only, not an API action.	*D* :/identity-center-applicati ons/ *identity-center-application_ID*
AssociateIdentityToIdentity CenterApplication	Required to associate an identity with an IAM Identity Center application for a CodeCatalyst space. This is an IAM policy permission only, not an API action.	arn:aws:codecataly st:region: *account_I D* :/identity-center-applicati ons/ *identity-center-application_ID*
BatchAssociateIdentitiesToI dentityCenterApplication	Required to associate multiple identities with an IAM Identity Center application for a CodeCatalyst space. This is an IAM policy permission only, not an API action.	arn:aws:codecataly st:region: *account_I D* :/identity-center-applicati ons/ *identity-center-application_ID*
BatchDisassociateIdentities FromIdentityCenterApplicati on	Required to disassociate multiple identities from an IAM Identity Center application for a CodeCatalyst space. This is an IAM policy permission only, not an API action.	arn:aws:codecataly st:region: *account_I D* :/identity-center-applicati ons/ *identity-center-application_ID*
CreateIdentityCenterApplica tion	Required to create an IAM Identity Center applicati on. This is an IAM policy permission only, not an API action.	arn:aws:codecataly st:region: *account_I D* :/identity-center-applicati ons/ *identity-center-application_ID*

CodeCatalyst permissions for IAM Identity Center applications	Required permissions	Resources
CreateSpaceAdminRoleAssignment	Required to create an administrator role assignment for a given CodeCatalyst space and IAM Identity Center application. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*
DeleteIdentityCenterApplication	Required to delete an IAM Identity Center application. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*
DisassociateIdentityCenterApplicationFromSpace	Required to disassociate an IAM Identity Center application from a CodeCatalyst space. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*
DisassociateIdentityFromIdentityCenterApplication	Required to disassociate an identity from an IAM Identity Center application for a CodeCatalyst space. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*

CodeCatalyst permissions for IAM Identity Center applications	Required permissions	Resources
GetIdentityCenterApplication	Required to get information about an IAM Identity Center application. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*
ListIdentityCenterApplications	Required to view a list of all IAM Identity Center applications in the account. This is an IAM policy permission only, not an API action.	Supports only a wildcard (*) in the policy `Resource` element.
ListIdentityCenterApplicationsForSpace	Required to view a list of IAM Identity Center applications by CodeCatalyst space. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*
ListSpacesForIdentityCenterApplication	Required to view a list of CodeCatalyst spaces by IAM Identity Center application. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*
SynchronizeIdentityCenterApplication	Required to synchronize an IAM Identity Center application with the backing identity store. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*

CodeCatalyst permissions for IAM Identity Center applications	Required permissions	Resources
UpdateIdentityCenterApplication	Required to update an IAM Identity Center application. This is an IAM policy permission only, not an API action.	`arn:aws:codecatalyst:region:` *`account_ID`* `:/identity-center-applications/` *`identity-center-application_ID`*

Using service-linked roles for CodeCatalyst

Amazon CodeCatalyst uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to CodeCatalyst. Service-linked roles are predefined by CodeCatalyst and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up CodeCatalyst easier because you don't have to manually add the necessary permissions. CodeCatalyst defines the permissions of its service-linked roles, and unless defined otherwise, only CodeCatalyst can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your CodeCatalyst resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for CodeCatalyst

CodeCatalyst uses the service-linked role named **AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization** – Allows Amazon CodeCatalyst read-only access to application instance profiles and associated directory users and groups on your behalf.

The AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization service-linked role trusts the following services to assume the role:

- `codecatalyst.amazonaws.com`

The role permissions policy named AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy allows CodeCatalyst to complete the following actions on the specified resources:

- Action: `View application instance profiles and associated directory users and groups` for `CodeCatalyst spaces that support identity federation and SSO users and groups`

You must configure permissions to allow your users, groups, or roles to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for CodeCatalyst

You don't need to manually create a service-linked role. When you create a space in the AWS Management Console, the AWS CLI, or the AWS API, CodeCatalyst creates the service-linked role for you.

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. Also, if you were using the CodeCatalyst service before November 17, 2023, when it began supporting service-linked roles, then CodeCatalyst created the AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization role in your account. To learn more, see A new role appeared in my AWS account.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a space, CodeCatalyst creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **View application instance profiles and associated directory users and groups** use case. In the AWS CLI or the AWS API,

create a service-linked role with the `codecatalyst.amazonaws.com` service name. For more information, see Creating a service-linked role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for CodeCatalyst

CodeCatalyst does not allow you to edit the AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for CodeCatalyst

You don't need to manually delete the AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization role. When you delete a space in the AWS Management Console, the AWS CLI, or the AWS API, CodeCatalyst cleans up the resources and deletes the service-linked role for you.

You can also use the IAM console, the AWS CLI or the AWS API to manually delete the service-linked role. To do this, you must first manually clean up the resources for your service-linked role and then you can manually delete it.

> **Note**
>
> If the CodeCatalyst service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete CodeCatalyst resources used by the AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization

- Delete the space.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for CodeCatalyst service-linked roles

CodeCatalyst supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS Regions and endpoints.

CodeCatalyst does not support using service-linked roles in every Region where the service is available. You can use the AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization role in the following Regions.

Region name	Region identity	Support in CodeCatalyst
US East (N. Virginia)	us-east-1	No
US East (Ohio)	us-east-2	No
US West (N. California)	us-west-1	No
US West (Oregon)	us-west-2	Yes
Africa (Cape Town)	af-south-1	No
Asia Pacific (Hong Kong)	ap-east-1	No
Asia Pacific (Jakarta)	ap-southeast-3	No
Asia Pacific (Mumbai)	ap-south-1	No
Asia Pacific (Osaka)	ap-northeast-3	No
Asia Pacific (Seoul)	ap-northeast-2	No
Asia Pacific (Singapore)	ap-southeast-1	No
Asia Pacific (Sydney)	ap-southeast-2	No
Asia Pacific (Tokyo)	ap-northeast-1	No
Canada (Central)	ca-central-1	No
Europe (Frankfurt)	eu-central-1	No

Region name	Region identity	Support in CodeCatalyst
Europe (Ireland)	eu-west-1	Yes
Europe (London)	eu-west-2	No
Europe (Milan)	eu-south-1	No
Europe (Paris)	eu-west-3	No
Europe (Stockholm)	eu-north-1	No
Middle East (Bahrain)	me-south-1	No
Middle East (UAE)	me-central-1	No
South America (São Paulo)	sa-east-1	No
AWS GovCloud (US-East)	us-gov-east-1	No
AWS GovCloud (US-West)	us-gov-west-1	No

AWS managed policies for Amazon CodeCatalyst

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AmazonCodeCatalystSupportAccess

This is a policy that grants permissions for all space administrators and space members to utilize the Business or Enterprise premium support plan associated with the space billing account. These permissions allow space administrators and members to utilitze the premium support plan for the resources they have permissions to within CodeCatalyst permissions policies.

Permissions details

This policy includes the following permissions.

- `support` – Grants permissions to allow users to search for, create, and resolve AWS Support cases. Also grants permissions to describe communications, severity levels, attachments, and related support case details.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "support:DescribeAttachment",
        "support:DescribeCaseAttributes",
        "support:DescribeCases",
        "support:DescribeCommunications",
        "support:DescribeIssueTypes",
        "support:DescribeServices",
```

```
        "support:DescribeSeverityLevels",
        "support:DescribeSupportLevel",
        "support:SearchForCases",
        "support:AddAttachmentsToSet",
        "support:AddCommunicationToCase",
        "support:CreateCase",
        "support:InitiateCallForCase",
        "support:InitiateChatForCase",
        "support:PutCaseAttributes",
        "support:RateCaseCommunication",
        "support:ResolveCase"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy: AmazonCodeCatalystFullAccess

This is a policy that grants permissions to manage your CodeCatalyst space and connected accounts in the Amazon CodeCatalyst Spaces page in the AWS Management Console. This application is used to configure AWS accounts that are connected to your space in CodeCatalyst.

Permissions details

This policy includes the following permissions.

- `codecatalyst` – Grants full permissions to the Amazon CodeCatalyst Spaces page in the AWS Management Console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CodeCatalystResourceAccess"
            "Effect": "Allow",
```

```
            "Action": [
                "codecatalyst:*",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CodeCatalystAssociateIAMRole"
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "codecatalyst.amazonaws.com",
                        "codecatalyst-runner.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

AWS managed policy: AmazonCodeCatalystReadOnlyAccess

This is a policy that grants permissions to view and list information for spaces and connected accounts in the Amazon CodeCatalyst Spaces page in the AWS Management Console. This application is used to configure AWS accounts that are connected to your space in CodeCatalyst.

Permissions details

This policy includes the following permissions.

- `codecatalyst` – Grants read-only permissions to the Amazon CodeCatalyst Spaces page in the AWS Management Console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecatalyst:Get*",
                "codecatalyst:List*",
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy

You can't attach AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy; to your IAM entities. This policy is attached to a service-linked role that allows CodeCatalyst to perform actions on your behalf. For more information, see Using service-linked roles for CodeCatalyst.

This policy allows customers to view application instance profiles and associated directory users and groups when managing spaces in CodeCatalyst. Customers will view these resources when managing spaces that support identity federation and SSO users and groups.

Permissions details

This policy includes the following permissions.

- `sso` – Grants permissions to allow users to view application instance profiles that are managed in IAM Identity Center for associated spaces in CodeCatalyst.

```
{
 "Version": "2012-10-17",
```

```
 "Statement": [
  {
   "Sid":
 "AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy",
   "Effect": "Allow",
   "Action": [
    "sso:ListInstances",
    "sso:ListApplications",
    "sso:ListApplicationAssignments",
    "sso:DescribeInstance",
    "sso:DescribeApplication"
   ],
   "Resource": "*"
  }
 ]
}
```

CodeCatalyst updates to AWS managed policies

View details about updates to AWS managed policies for CodeCatalyst since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CodeCatalyst Document history page.

Change	Description	Date
AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy – New policy	CodeCatalyst added the policy. Grants permissions to allow CodeCatalyst users to view application instance profiles and associated directory users and groups.	November 17, 2023
AmazonCodeCatalystSupportAccess – New policy	CodeCatalyst added the policy. Grants permissions to allow CodeCatalyst users to search	April 20, 2023

Change	Description	Date
	for, create, and resolve support cases, as well as viewing related communica tions and details.	
AmazonCodeCatalyst FullAccess – New policy	CodeCatalyst added the policy. Grants full access to CodeCatalyst.	April 20, 2023
AmazonCodeCatalyst ReadOnlyAccess – New policy	CodeCatalyst added the policy. Grants read-only access to CodeCatalyst.	April 20, 2023
CodeCatalyst started tracking changes	CodeCatalyst started tracking changes for its AWS managed policies.	April 20, 2023

Grant access to project AWS resources with IAM roles

CodeCatalyst can access AWS resources by connecting your AWS account to a CodeCatalyst space. You can then create the following service roles and associate them when you connect your account.

For more information about the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

- To access resources in an AWS account for your CodeCatalyst projects and workflows, you must first grant permission for CodeCatalyst to access those resources on your behalf. To do so, you must create a service role in a connected AWS account that CodeCatalyst can assume on behalf of users and projects in the space. You can either choose to create and use the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can create customized service roles and configure these IAM policies and roles manually. As a best practice, assign these roles the least amount of permissions necessary.

> **Note**
>
> For customized service roles, the CodeCatalyst service principal is required. For more information about the CodeCatalyst service principal and trust model, see Understanding the CodeCatalyst trust model.

- To manage support for a space through the connected AWS account, you can choose to create and use the **AWSRoleForCodeCatalystSupport** service role that allows CodeCatalyst users to access support. For more information about support for a CodeCatalyst space, see AWS Support for Amazon CodeCatalyst.

Understanding the CodeCatalystWorkflowDevelopmentRole-*spaceName* service role

You can add an IAM role for your space that CodeCatalyst can use to create and access resources in a connected AWS account. This is called a service role. The simplest way to create a service role is to add one when you create the space and to choose the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** option for that role. This not only creates the service role with the `AdministratorAccess` attached, but it also creates the trust policy that allows CodeCatalyst to assume the role on behalf of users in projects in the space. The service role is scoped to the space, not to individual projects. To create this role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. You can only create one role for each space in each account.

> **Note**
>
> This role is only recommended for use with development accounts and uses the `AdministratorAccess` AWS managed policy, giving it full access to create new policies and resources in this AWS account.

The policy attached to the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role is designed to work with projects created with blueprints in the space. It allows users in those projects to develop, build, test, and deploy code using resources in the connected AWS account. For more information, see Creating a role for an AWS service.

The policy attached to the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role is the `AdministratorAccess` managed policy in AWS. This is a policy that grants full access to

all AWS actions and resources. To view the JSON policy document in the IAM console, see AdministratorAccess.

The following trust policy allows CodeCatalyst to assume the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role. For more information about the CodeCatalyst trust model, see Understanding the CodeCatalyst trust model.

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
             "Principal": {
                "Service": [
                    "codecatalyst-runner.amazonaws.com",
                    "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/*"
                }
            }
        }
    ]
```

Creating the CodeCatalystWorkflowDevelopmentRole-*spaceName* role for your account and space

Follow these steps to create the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role that will be used for workflows in your space. For each account that you want to have IAM roles for use in projects, to your space, you must add a role such as the developer role.

Before you begin, you must have administrative privileges for your AWS account or be able to work with your administrator. For more information about how AWS accounts and IAM roles are used in CodeCatalyst, see Allowing access to AWS resources with connected AWS accounts.

To create and add the CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*

1. Before you start in the CodeCatalyst console, open the AWS Management Console, and then make sure you are logged in with the same AWS account for your space.
2. Open the CodeCatalyst console at https://codecatalyst.aws/.
3. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
4. Choose the link for the AWS account where you want to create the role. The **AWS account details** page displays.
5. Choose **Manage roles from AWS Management Console**.

   The **Add IAM role to Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst spaces** page. You might need to log in to access the page.
6. Choose **Create CodeCatalyst development administrator role in IAM**. This option creates a service role that contains the permissions policy and trust policy for the development role. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role.

   > **Note**
   >
   > This role is only recommended for use with developer accounts and uses the `AdministratorAccess` AWS managed policy, giving it full access to create new policies and resources in this AWS account.
7. Choose **Create development role**.
8. On the connections page, under **IAM roles available to CodeCatalyst**, view the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role in the list of IAM roles added to your account.
9. To return to your space, choose **Go to Amazon CodeCatalyst**.

Understanding the AWSRoleForCodeCatalystSupport service role

You can add an IAM role for your space that CodeCatalyst users in a space can use to create and access support cases. This is called a service role for support.The simplest way to create a service role for support is to add one when you create the space and to choose the

AWSRoleForCodeCatalystSupport option for that role. This not only creates the policy and the role, but it also creates the trust policy that allows CodeCatalyst to assume the role on behalf of users in projects in the space. The service role is scoped to the space, not to individual projects. To create this role, see Creating the **AWSRoleForCodeCatalystSupport** role for your account and space.

The policy attached to the AWSRoleForCodeCatalystSupport role is managed policy that provides access to support permissions. For more information, see AWS managed policy: AmazonCodeCatalystSupportAccess.

The trust role for the policy allows CodeCatalyst to assume the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "codecatalyst.amazonaws.com",
                    "codecatalyst-runner.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Creating the AWSRoleForCodeCatalystSupport role for your account and space

Follow these steps to create the AWSRoleForCodeCatalystSupport role that will be used for support cases in your space. The role must be added to the designated billing account for the space.

Before you begin, you must have administrative privileges for your AWS account or be able to work with your administrator. For more information about how AWS accounts and IAM roles are used in CodeCatalyst, see Allowing access to AWS resources with connected AWS accounts.

To create and add the CodeCatalyst AWSRoleForCodeCatalystSupport

1. Before you start in the CodeCatalyst console, open the AWS Management Console, and then make sure you are logged in with the same AWS account for your space.

2. Navigate to your CodeCatalyst space. Choose **Settings**, and then choose **AWS accounts**.
3. Choose the link for the AWS account where you want to create the role. The **AWS account details** page displays.
4. Choose **Manage roles from AWS Management Console**.

   The **Add IAM role to Amazon CodeCatalyst space** page opens in the AWS Management Console. This is the **Amazon CodeCatalyst Spaces** page. You might need to sign in to access the page.
5. Under **CodeCatalyst space details**, choose **Add CodeCatalyst Support role**. This option creates a service role that contains the permissions policy and trust policy for the preview development role. The role will have a name **AWSRoleForCodeCatalystSupport** with a unique identifier appended. For more information about the role and role policy, see Understanding the **AWSRoleForCodeCatalystSupport** service role.
6. On the **Add role for CodeCatalyst Support** page, leave the default selected, and then choose **Create role**.
7. Under **IAM roles available to CodeCatalyst**, view the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role in the list of IAM roles added to your account.
8. To return to your space, choose **Go to Amazon CodeCatalyst**.

Configuring IAM roles for workflow actions in CodeCatalyst

This section details IAM roles and policies that you can create to use with your CodeCatalyst account. For instructions to create example roles, see Creating roles manually for workflow actions. After you create your IAM role, copy the role ARN to add the IAM role to your account connection and associate it with your project environment. To learn more, see Adding IAM roles to account connections.

CodeCatalyst build role for Amazon S3 access

For CodeCatalyst workflow build actions, you can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can create an IAM role named **CodeCatalystBuildRoleforS3Access**. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on AWS CloudFormation resources in your AWS account.

This role gives permissions to do the following:

- Write to Amazon S3 buckets.
- Support building of resources with AWS CloudFormation. This requires Amazon S3 access.

This role uses the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "s3:PutObject",
        "iam:PassRole"
    ],
    "Resource": "resource_ARN",
    "Effect": "Allow"
}]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

CodeCatalyst build role for AWS CloudFormation

For CodeCatalyst workflow build actions, you can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can create an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on AWS CloudFormation resources in your AWS account.

This role gives permissions to do the following:

- Support building of resources with AWS CloudFormation. This is required along with the CodeCatalyst build role for Amazon S3 access and the CodeCatalyst deploy role for AWS CloudFormation.

The following AWS managed policies should be attached to this role:

- **AWSCloudFormationFullAccess**
- **IAMFullAccess**
- **AmazonS3FullAccess**
- **AmazonAPIGatewayAdministrator**
- **AWSLambdaFullAccess**

CodeCatalyst build role for CDK

For CodeCatalyst workflows that run CDK build actions, such as Modern three-tier web application, you can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can create an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to bootstrap and run CDK build commands for AWS CloudFormation resources in your AWS account.

This role gives permissions to do the following:

- Write to Amazon S3 buckets.
- Support building of CDK constructs and AWS CloudFormation resource stacks. This requires access to Amazon S3 for artifact storage, Amazon ECR for image repository support, and SSM for system governance and monitoring for virtual instances.

This role uses the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:*",
                "ecr:*",
                "ssm:*",
                "s3:*",
                "iam:PassRole",
                "iam:GetRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
```

```
                "iam:PutRolePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

CodeCatalyst deploy role for AWS CloudFormation

For CodeCatalyst workflow deploy actions that use AWS CloudFormation, you can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can use a policy with scoped permissions that CodeCatalyst needs to run tasks on AWS CloudFormation resources in your AWS account.

This role gives permissions to do the following:

- Allow CodeCatalyst to invoke a Λ function to perform blue/green deployment through AWS CloudFormation.
- Allow CodeCatalyst to create and update stacks and changesets in AWS CloudFormation.

This role uses the following policy:

```
{"Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
```

```
        "cloudformation:ValidateTemplate",
        "cloudformation:List*",
        "iam:PassRole"
    ],
    "Resource": "resource_ARN",
    "Effect": "Allow"
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

CodeCatalyst deploy role for Amazon EC2

CodeCatalyst workflow deploy actions use an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on Amazon EC2 resources in your AWS account. The default policy for the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role does not include permissions for Amazon EC2 or Amazon EC2 Auto Scaling.

This role gives permissions to do the following:

- Create Amazon EC2 deployments.
- Read the tags on an instance or identify an Amazon EC2 instance by Auto Scaling group names.
- Read, create, update, and delete Amazon EC2 Auto Scaling groups, lifecycle hooks, and scaling policies.
- Publish information to Amazon SNS topics.
- Retrieve information about CloudWatch alarms.
- Read and update Elastic Load Balancing.

This role uses the following policy:

```
{
```

```
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
 "autoscaling:CompleteLifecycleAction",
"autoscaling:DeleteLifecycleHook",
"autoscaling:DescribeAutoScalingGroups",
"autoscaling:DescribeLifecycleHooks",
"autoscaling:PutLifecycleHook",
"autoscaling:RecordLifecycleActionHeartbeat",
"autoscaling:CreateAutoScalingGroup",
"autoscaling:UpdateAutoScalingGroup",
"autoscaling:EnableMetricsCollection",
"autoscaling:DescribePolicies",
"autoscaling:DescribeScheduledActions",
"autoscaling:DescribeNotificationConfigurations",
"autoscaling:SuspendProcesses",
"autoscaling:ResumeProcesses",
"autoscaling:AttachLoadBalancers",
"autoscaling:AttachLoadBalancerTargetGroups",
"autoscaling:PutScalingPolicy",
"autoscaling:PutScheduledUpdateGroupAction",
"autoscaling:PutNotificationConfiguration",
"autoscaling:PutWarmPool",
"autoscaling:DescribeScalingActivities",
"autoscaling:DeleteAutoScalingGroup",
"ec2:DescribeInstances",
"ec2:DescribeInstanceStatus",
"ec2:TerminateInstances",
"tag:GetResources",
"sns:Publish",
"cloudwatch:DescribeAlarms",
"cloudwatch:PutMetricAlarm",
"elasticloadbalancing:DescribeLoadBalancers",
"elasticloadbalancing:DescribeInstanceHealth",
"elasticloadbalancing:RegisterInstancesWithLoadBalancer",
"elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
"elasticloadbalancing:DescribeTargetGroups",
"elasticloadbalancing:DescribeTargetHealth",
"elasticloadbalancing:RegisterTargets",
"elasticloadbalancing:DeregisterTargets"
],
"Resource": "resource_ARN"
```

```
      }
   ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

CodeCatalyst deploy role for Amazon ECS

For CodeCatalyst workflow actions, you can create an IAM role with the necessary permissions. You can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can create an IAM role for CodeCatalyst deploy actions to use for Lambda deployments. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on Amazon ECS resources in your AWS account.

This role gives permissions to do the following:

- Initiate rolling Amazon ECS deployment on behalf of a CodeCatalyst user, in an account specified in the CodeCatalyst connection.
- Read, update, and delete Amazon ECS task sets.
- Update Elastic Load Balancing target groups, listeners, and rules.
- Invoke Lambda functions.
- Access revision files in Amazon S3 buckets.
- Retrieve information about CloudWatch alarms.
- Publish information to Amazon SNS topics.

This role uses the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [{
```

```
    "Action":[
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs:DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
    ],
    "Resource":"*",
```

```
   "Effect":"Allow"
},{"Action":[
      "iam:PassRole"
   ],
   "Effect":"Allow",
   "Resource":"*",
   "Condition":{"StringLike":{"iam:PassedToService":[
            "ecs-tasks.amazonaws.com",
            "codedeploy.amazonaws.com"
         ]
      }
   }
}]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

CodeCatalyst deploy role for Lambda

For CodeCatalyst workflow actions, you can create an IAM role with the necessary permissions. You can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or or you create an IAM role for CodeCatalyst deploy actions to use for Lambda deployments. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on Lambda resources in your AWS account.

This role gives permissions to do the following:

- Read, update, and invoke Lambda functions and aliases.
- Access revision files in Amazon S3 buckets.
- Retrieve information about CloudWatch Events alarms.
- Publish information to Amazon SNS topics.

This role uses the following policy:

```
*{*
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:DescribeAlarms",
                "lambda:UpdateAlias",
                "lambda:GetAlias",
                "lambda:GetProvisionedConcurrencyConfig",
                "sns:Publish"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::/CodeDeploy/",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:::function:CodeDeployHook_*",
            "Effect": "Allow"
        }
```

```
    ]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

CodeCatalyst deploy role for Lambda

For CodeCatalyst workflow actions, you can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can create an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on Lambda resources in your AWS account.

This role gives permissions to do the following:

- Read, update, and invoke Lambda functions and aliases.
- Access revision files in Amazon S3 buckets.
- Retrieve information about CloudWatch alarms.
- Publish information to Amazon SNS topics.

This role uses the following policy:

```
*{*
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:DescribeAlarms",
                "lambda:UpdateAlias",
                "lambda:GetAlias",
                "lambda:GetProvisionedConcurrencyConfig",
                "sns:Publish"
            ],
```

```
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::/CodeDeploy/",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:::function:CodeDeployHook_*",
            "Effect": "Allow"
        }
    ]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

CodeCatalyst deploy role for AWS SAM

For CodeCatalyst workflow actions, you can use the default **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role, or you can create an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on AWS SAM and AWS CloudFormation resources in your AWS account.

This role gives permissions to do the following:

- Allow CodeCatalyst to invoke a Lambda function to perform deployment of serverless and AWS SAM CLI applications.
- Allow CodeCatalyst to create and update stacks and changesets in AWS CloudFormation.

This role uses the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "iam:PassRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:TagRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "cloudformation:*",
                "lambda:*",
                "apigateway:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

CodeCatalyst read only role for Amazon EC2

For CodeCatalyst workflow actions, you can create an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on Amazon EC2 resources in your AWS account. The **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role does not include permissions for Amazon EC2 or the described actions for Amazon CloudWatch.

This role gives permissions to do the following:

- Get status of Amazon EC2 instances.
- Get CloudWatch metrics for Amazon EC2 instances.

This role uses the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:Describe",
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": "elasticloadbalancing:Describe",
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:Describe"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:Describe",
            "Resource": "resource_ARN"
        }
    ]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

CodeCatalyst read only role for Amazon ECS

For CodeCatalyst workflow actions, you can create an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on Amazon ECS resources in your AWS account.

This role gives permissions to do the following:

- Read Amazon ECS task sets.
- Retrieve information about CloudWatch alarms.

This role uses the following policy:

```
*{*
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Action": [
                "ecs:DescribeServices",
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
        {
            "Action": [
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeRules"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam:::role/ecsTaskExecutionRole",
                "arn:aws:iam:::role/ECSTaskExecution"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "ecs-tasks.amazonaws.com"
                    ]
                }
```

```
            }
        }
    ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

CodeCatalyst read only role for Lambda

For CodeCatalyst workflow actions, you can create an IAM role with the necessary permissions. This role uses a policy with scoped permissions that CodeCatalyst needs to run tasks on Lambda resources in your AWS account.

This role gives permissions for the following:

- Read Lambda functions and aliases.
- Access revision files in Amazon S3 buckets.
- Retrieve information about CloudWatch alarms.

This role uses the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:DescribeAlarms",
                "lambda:GetAlias",
                "lambda:GetProvisionedConcurrencyConfig"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
```

```
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::/CodeDeploy/",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        }
    ]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

Creating roles manually for workflow actions

CodeCatalyst workflow actions use IAM roles that you create called the **build role**, the **deploy role**, and the **stack role**.

Follow these steps to create these roles in IAM.

To create a deploy role

1. Create a policy for the role, as follows:

   a. Sign in to AWS.

   b. Open the IAM console at https://console.aws.amazon.com/iam/.

   c. In the navigation pane, choose **Policies**.

   d. Choose **Create policy**.

   e. Choose the **JSON** tab.

   f. Delete the existing code.

   g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "cloudformation:List*",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
        "Resource": "*"
```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-deploy-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the deploy role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. Choose **Next**.

f. In **Permissions policies**, search for `codecatalyst-deploy-policy` and select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-deploy-role
```

i. For **Role description**, enter:

```
CodeCatalyst deploy role
```

j. Choose **Create role**.

You have now created a deploy role with a trust policy and permissions policy.

3. Obtain the deploy role ARN, as follows:

a. In the navigation pane, choose **Roles**.

b. In the search box, enter the name of the role you just created (`codecatalyst-deploy-role`).

c. Choose the role from the list.

The role's **Summary** page appears.

d. At the top, copy the **ARN** value.

You have now created the deploy role with the appropriate permissions, and obtained its ARN.

To create a build role

1. Create a policy for the role, as follows:

a. Sign in to AWS.

b. Open the IAM console at https://console.aws.amazon.com/iam/.

c. In the navigation pane, choose **Policies**.

d. Choose **Create policy**.

e. Choose the **JSON** tab.

f. Delete the existing code.

g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "s3:PutObject",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-build-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the build role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. Choose **Next**.

f. In **Permissions policies**, search for `codecatalyst-build-policy` and select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-build-role
```

i. For **Role description**, enter:

```
CodeCatalyst build role
```

j. Choose **Create role**.

You have now created a build role with a trust policy and permissions policy.

3. Obtain the build role ARN, as follows:

a. In the navigation pane, choose **Roles**.

b. In the search box, enter the name of the role you just created (`codecatalyst-build-role`).

c. Choose the role from the list.

The role's **Summary** page appears.

d. At the top, copy the **ARN** value.

You have now created the build role with the appropriate permissions, and obtained its ARN.

To create a stack role

> **Note**
>
> You don't have to create a stack role, although doing so is recommended for security reasons. If you don't create the stack role, you'll need to add the permissions policies described further on in this procedure to the deploy role.

1. Sign in to AWS using the account where you want to deploy your stack.
2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, choose **Roles**. and then choose **Create role**.
4. At the top, choose **AWS service**.
5. From the list of services, choose **CloudFormation**.
6. Choose **Next: Permissions**.
7. In the search box, add any policies that are required to access the resources in your stack. For example, if your stack includes an AWS Lambda function, you need to add a policy that grants access to Lambda.

   > **Tip**
   >
   > If you're unsure which policies to add, you can omit them for now. When you test the action, if you don't have the right permissions, AWS CloudFormation generates errors that show which permissions you need to add.

8. Choose **Next: Tags**.
9. Choose **Next: Review**.
10. For **Role name**, enter:

```
codecatalyst-stack-role
```

11. Choose **Create role**.
12. To obtain the stack role's ARN, do the following:

    a. In the navigation pane, choose **Roles**.
    b. In the search box, enter the name of the role you just created (`codecatalyst-stack-role`).
    c. Choose the role from the list.
    d. On the **Summary** page, copy the **Role ARN** value.

Using AWS CloudFormation to create policies and roles in IAM

You can choose to create and use AWS CloudFormation templates to create the policies and roles you need to access resources in an AWS account for your CodeCatalyst projects and workflows. AWS CloudFormation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications that run on AWS. If you intend to create roles in multiple AWS accounts, creating a template can help you perform this task more quickly.

The following example template creates a deploy action role and policy.

```
Parameters:
  CodeCatalystAccountId:
    Type: String
    Description: Account ID from the connections page
  ExternalId:
    Type: String
    Description: External ID from the connections page
Resources:
  CrossAccountRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              AWS:
                - !Ref CodeCatalystAccountId
            Action:
              - 'sts:AssumeRole'
```

```
            Condition:
              StringEquals:
                sts:ExternalId: !Ref ExternalId
      Path: /
      Policies:
        - PolicyName: CodeCatalyst-CloudFormation-action-policy
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - 'cloudformation:CreateStack'
                  - 'cloudformation:DeleteStack'
                  - 'cloudformation:Describe*'
                  - 'cloudformation:UpdateStack'
                  - 'cloudformation:CreateChangeSet'
                  - 'cloudformation:DeleteChangeSet'
                  - 'cloudformation:ExecuteChangeSet'
                  - 'cloudformation:SetStackPolicy'
                  - 'cloudformation:ValidateTemplate'
                  - 'cloudformation:List*'
                  - 'iam:PassRole'
                Resource: '*'
```

Creating the role manually for the web application blueprint

The CodeCatalyst web application blueprint uses IAM roles that you create called the **build role for CDK**, the **deploy role**, and the **stack role**.

Follow these steps to create the role in IAM.

To create a build role

1. Create a policy for the role, as follows:

    a. Sign in to AWS.

    b. Open the IAM console at https://console.aws.amazon.com/iam/.

    c. In the navigation pane, choose **Policies**.

    d. Choose **Create Policy**.

    e. Choose the **JSON** tab.

    f. Delete the existing code.

g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:*",
                "ecr:*",
                "ssm:*",
                "s3:*",
                "iam:PassRole",
                "iam:GetRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-webapp-build-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the build role, as follows:

   a. In the navigation pane, choose **Roles**, and then choose **Create role**.

   b. Choose **Custom trust policy**.

   c. Delete the existing custom trust policy.

   d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

   e. Choose **Next**.

   f. Attach the permissions policy to the build role. On the **Add permissions** page, in the **Permissions policies** section, search for `codecatalyst-webapp-build-policy` and select its check box.

   g. Choose **Next**.

   h. For **Role name**, enter:

```
codecatalyst-webapp-build-role
```

   i. For **Role description**, enter:

```
CodeCatalyst Web app build role
```

   j. Choose **Create role**.

You have now created a build role with a trust policy and permissions policy.

3. Attach the permissions policy to the build role, as follows:

   a. In the navigation pane, choose **Roles**, and then search for `codecatalyst-webapp-build-role`.

   b. Choose `codecatalyst-webapp-build-role` to display its details.

   c. In the **Permissions** tab, choose **Add permissions**, and then choose **Attach policies**.

   d. Search for `codecatalyst-webapp-build-policy`, select its check box, and then choose **Attach policies**.

   You have now attached the permissions policy to the build role. The build role now has two policies: a permissions policy and a trust policy.

4. Obtain the build role ARN, as follows:

   a. In the navigation pane, choose **Roles**.

   b. In the search box, enter the name of the role you just created (`codecatalyst-webapp-build-role`).

   c. Choose the role from the list.

   The role's **Summary** page appears.

   d. At the top, copy the **ARN** value.

   You have now created the build role with the appropriate permissions, and obtained its ARN.

Creating roles manually for the SAM blueprint

The CodeCatalyst SAM blueprint uses IAM roles that you create called the **build role for CloudFormation** and the **deploy role for SAM**.

Follow these steps to create the roles in IAM.

To create a build role for CloudFormation

1. Create a policy for the role, as follows:

   a. Sign in to AWS.

b. Open the IAM console at https://console.aws.amazon.com/iam/.

c. In the navigation pane, choose **Policies**.

d. Choose **Create Policy**.

e. Choose the **JSON** tab.

f. Delete the existing code.

g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudformation:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.
>
> ```
> "Resource": "*"
> ```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-SAM-build-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the build role, as follows:

   a. In the navigation pane, choose **Roles**, and then choose **Create role**.

   b. Choose **Custom trust policy**.

   c. Delete the existing custom trust policy.

   d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

   e. Choose **Next**.

   f. Attach the permissions policy to the build role. On the **Add permissions** page, in the **Permissions policies** section, search for `codecatalyst-SAM-build-policy` and select its check box.

   g. Choose **Next**.

   h. For **Role name**, enter:

```
codecatalyst-SAM-build-role
```

   i. For **Role description**, enter:

```
CodeCatalyst SAM build role
```

   j. Choose **Create role**.

You have now created a build role with a trust policy and permissions policy.

3. Attach the permissions policy to the build role, as follows:

   a. In the navigation pane, choose **Roles**, and then search for `codecatalyst-SAM-build-role`.

   b. Choose `codecatalyst-SAM-build-role` to display its details.

   c. In the **Permissions** tab, choose **Add permissions**, and then choose **Attach policies**.

   d. Search for `codecatalyst-SAM-build-policy`, select its check box, and then choose **Attach policies**.

   You have now attached the permissions policy to the build role. The build role now has two policies: a permissions policy and a trust policy.

4. Obtain the build role ARN, as follows:

   a. In the navigation pane, choose **Roles**.

   b. In the search box, enter the name of the role you just created (`codecatalyst-SAM-build-role`).

   c. Choose the role from the list.

   The role's **Summary** page appears.

   d. At the top, copy the **ARN** value.

You have now created the build role with the appropriate permissions, and obtained its ARN.

To create a deploy role for SAM

1. Create a policy for the role, as follows:

   a. Sign in to AWS.

   b. Open the IAM console at https://console.aws.amazon.com/iam/.

   c. In the navigation pane, choose **Policies**.

   d. Choose **Create Policy**.

   e. Choose the **JSON** tab.

   f. Delete the existing code.

g. Paste the following code:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "iam:PassRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:TagRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "cloudformation:*",
                "lambda:*",
                "apigateway:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Note

The first time the role is used to run workflow actions, use the wildcard in the resource policy statement and then scope down the policy with the resource name after it is available.

```
"Resource": "*"
```

h. Choose **Next: Tags**.

i. Choose **Next: Review**.

j. In **Name**, enter:

```
codecatalyst-SAM-deploy-policy
```

k. Choose **Create policy**.

You have now created a permissions policy.

2. Create the build role, as follows:

a. In the navigation pane, choose **Roles**, and then choose **Create role**.

b. Choose **Custom trust policy**.

c. Delete the existing custom trust policy.

d. Add the following custom trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. Choose **Next**.

f. Attach the permissions policy to the build role. On the **Add permissions** page, in the **Permissions policies** section, search for `codecatalyst-SAM-deploy-policy` and select its check box.

g. Choose **Next**.

h. For **Role name**, enter:

```
codecatalyst-SAM-deploy-role
```

i. For **Role description**, enter:

```
CodeCatalyst SAM deploy role
```

j. Choose **Create role**.

You have now created a build role with a trust policy and permissions policy.

3. Attach the permissions policy to the build role, as follows:

   a. In the navigation pane, choose **Roles**, and then search for `codecatalyst-SAM-deploy-role`.

   b. Choose `codecatalyst-SAM-deploy-role` to display its details.

   c. In the **Permissions** tab, choose **Add permissions**, and then choose **Attach policies**.

   d. Search for `codecatalyst-SAM-deploy-policy`, select its check box, and then choose **Attach policies**.

   You have now attached the permissions policy to the build role. The build role now has two policies: a permissions policy and a trust policy.

4. Obtain the build role ARN, as follows:

   a. In the navigation pane, choose **Roles**.

   b. In the search box, enter the name of the role you just created (`codecatalyst-SAM-deploy-role`).

   c. Choose the role from the list.

   The role's **Summary** page appears.

   d. At the top, copy the **ARN** value.

You have now created the build role with the appropriate permissions, and obtained its ARN.

Compliance validation for Amazon CodeCatalyst

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon CodeCatalyst

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure. To learn more about what CodeCatalyst data is replicated across AWS Regions, see Data protection in Amazon CodeCatalyst.

Infrastructure Security in Amazon CodeCatalyst

As a managed service, Amazon CodeCatalyst is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access CodeCatalyst through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Configuration and vulnerability analysis in Amazon CodeCatalyst

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Your data and privacy in Amazon CodeCatalyst

Amazon CodeCatalyst takes your privacy seriously, and the security of your information is our top priority. You can review more about how we handle your information in the AWS Privacy Notice.

To request and view your data, see Requesting your data in the AWS General Reference.

Deleting your AWS Builder ID profile

Deleting your profile is a permanent action that cannot be reversed. The deletion process begins immediately after you choose **Delete**. Amazon CodeCatalyst starts deleting your profile and all associated personal information. This process may take up to 90 days to complete.

When your profile is deleted, you cannot access or recover your data in Amazon CodeCatalyst. This includes Personal Access Tokens, Roles, User Memberships, and any Amazon CodeCatalyst spaces of which you are the only member. You can no longer sign in to Amazon CodeCatalyst.

For information on how to delete your AWS Builder ID profile, see Deleting your AWS Builder ID in the AWS General Reference.

Best practices for workflow actions in Amazon CodeCatalyst

There are a number of security best practices to consider as you develop your workflows in CodeCatalyst. The following are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Topics

Sensitive information

Do not embed sensitive information in your YAML. Rather than embedding credentials, keys, or tokens in your YAML, we recommend you use CodeCatalyst secrets. Secrets provide an easy way to store and reference sensitive information from within your YAML.

Licensing terms

Make sure to pay attention to the licensing terms of the action you choose to use.

Untrusted code

Actions are generally self-contained, single purpose modules that can be shared across a project, space, or the broader community. Using code from others can be a great convenience and efficiency gain, but also introduces a new threat vector. Review the following sections to ensure you're following best practices to keep your CI/CD workflows secure.

GitHub Actions

GitHub Actions are open source, built and maintained by the community. We follow the shared responsibility model and consider GitHub Actions source code as customer data for which you are responsible. GitHub Actions can be granted access to secrets, repository tokens, source code, account links, and your compute time. Make sure you are confident in the trustworthiness and security of the GitHub Actions you plan to run.

More specific guidance and security best practices for GitHub Actions:

- Security hardening
- Preventing pwn requests
- Untrusted input
- How to trust your building blocks

Understanding the CodeCatalyst trust model

The Amazon CodeCatalyst trust model allows CodeCatalyst to assume the service role in the connected AWS account. The model connects the IAM role, the CodeCatalyst service principals, and the CodeCatalyst space. The trust policy uses the `aws:SourceArn` condition key to grant permissions to the CodeCatalyst space specified in the condition key. For more information about this condition key, see aws:SourceArn in the *IAM User Guide*.

A trust policy is a JSON policy document in which you define the principals that you trust to assume the role. A role trust policy is a required resource-based policy that is attached to a role in IAM. For more information, see Terms and concepts in the *IAM User Guide*. For details about the service principals for CodeCatalyst, see Service principals for CodeCatalyst.

In the following trust policy, the service principals listed in the `Principal` element are granted permissions from the resource-based policy, and the `Condition` block is used to limit access to the scoped-down resource.

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
             "Principal": {
                "Service": [
                    "codecatalyst-runner.amazonaws.com",
                    "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/*"
                }
            }
        }
    ]
```

In the trust policy, the CodeCatalyst service principals are given access through the `aws:SourceArn` condition key, which contains the Amazon Resource Name (ARN) for the CodeCatalyst space ID. The ARN uses the following format:

```
arn:aws:codecatalyst:::space/spaceId/project/*
```

> **Important**
>
> Use the space ID only in condition keys, such as `aws:SourceArn`. Do not use the space ID in IAM policy statements as a resource ARN.

As a best practice, scope down permissions as much as possible in the policy.

- You can use the wildcard (*) in the `aws:SourceArn` condition key for specifying all projects in the space with `project/*`.

- You can specify resource-level permissions in the `aws:SourceArn` condition key for a specific project in the space with `project/projectId`.

Service principals for CodeCatalyst

You use the `Principal` element in a resource-based JSON policy to specify the principal that is allowed or denied access to a resource. The principals that you can specify in the trust policy include users, roles, accounts, and services. You cannot use the `Principal` element in an identity-based policy; similarly, you cannot identify a user group as a principal in a policy (such as a resource-based policy) because groups relate to permissions, not authentication, and principals are authenticated IAM entities.

In the trust policy, you can specify AWS services in the `Principal` element of a resource-based policy or in condition keys that support principals. Service principals are defined by the service. The following are the service principals defined for CodeCatalyst:

- **codecatalyst.amazonaws.com** - This service principal is used for a role that will grant CodeCatalyst access to AWS.
- **codecatalyst-runner.amazonaws.com** - This service principal is used for a role that will grant CodeCatalyst access to AWS resources in deployments for CodeCatalyst workflows.

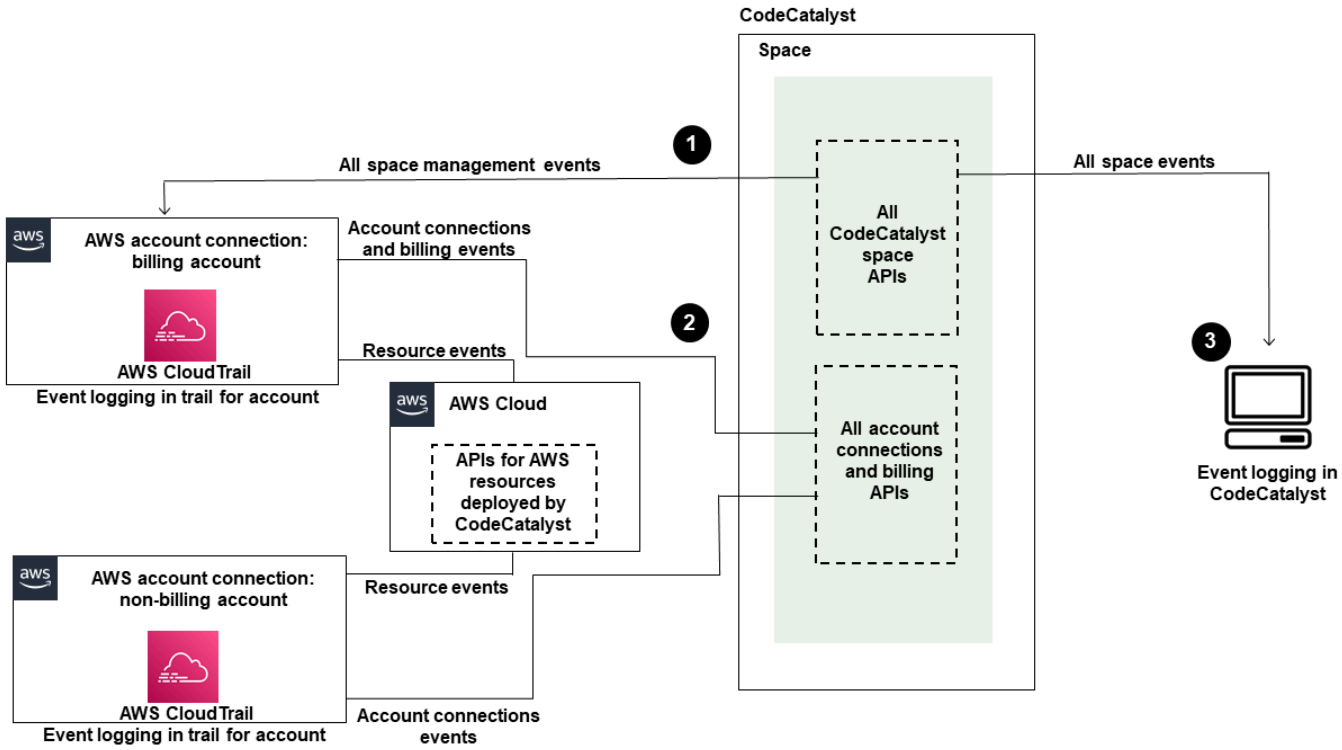
For more information, see AWS JSON policy elements: Principal in the *IAM User Guide*.

Monitoring events and API calls using logging

In Amazon CodeCatalyst, management events for the space are collected by AWS CloudTrail and are logged in the trail for the billing account for the space. CloudTrail logging is the primary method to manage logging for CodeCatalyst events, and a secondary method is viewing event logging in CodeCatalyst.

Events in the account are logged with the trail and designated bucket that is set up for the AWS account.

The following diagram shows how all management events for the space are logged in CloudTrail for the billing account, while account connections/billing events and AWS resource events are logged in CloudTrail for the respective account.



The diagram illustrates the following steps:

1. When a space is created, an AWS account is connected to the space and is designated as the billing account. The trail used is the trail that was created in CloudTrail for the billing account, where space events are logged. CloudTrail captures API calls and related events made by or on behalf of a CodeCatalyst space and delivers the log files to an S3 bucket that you specify. If the billing account changes to another AWS account, then space events are logged in the trail and bucket for that account. For more information about CodeCatalyst management events that are logged by CloudTrail, see CodeCatalyst information in CloudTrail.
2. Other accounts connected to the space, including the billing account, log a subset of events for account connections and billing events. CodeCatalyst workflows that generate account events for AWS resources deployed for that account are also logged in the trail and bucket for the AWS account. CloudTrail captures API calls and related events made by or on behalf of a CodeCatalyst space and delivers the log files to an S3 bucket that you specify. For more information about CodeCatalyst management events that are logged by CloudTrail, see Accessing logged events using event logging.
3. You can also monitor CodeCatalyst actions in your space within a specific time in the space with the **list-event-logs** command using the AWS CLI. For more information, see the Amazon

CodeCatalyst API Reference Guide. You must have the **Space administrator** role to call the list of events for CodeCatalyst actions in your space. For more information, see Accessing logged events using event logging.

> **Note**
>
> `ListEventLogs` guarantees events for the last 30 days in a given space. You can also view and retrieve a list of management events over the last 90 days for CodeCatalyst in the AWS CloudTrail console by viewing **Event history**, or by creating a trail to create and maintain a record of events that extends past 90 days. For more information, see Working with CloudTrail Event history and Working with CloudTrail trails.

> **Note**
>
> AWS resources that are deployed into connected accounts for CodeCatalyst workflows, are not logged as part of CloudTrail logging for the CodeCatalyst space. For example, CodeCatalyst resources include a space or project. AWS resources include an Amazon ECS service or Lambda function. You must configure CloudTrail logging separately for each AWS account where resources are deployed into.

Here is one possible flow for event monitoring in CodeCatalyst.

Mary Major is a **Space administrator** for a CodeCatalyst space and views all management events in CodeCatalyst for space-level and project-level resources in the space that are logged in CloudTrail. See CodeCatalyst information in CloudTrail for example events that are logged in CloudTrail.

For resources that are created in CodeCatalyst, such as Dev Environments, Mary views the **Event history** in the billing account for the space and investigates events where Dev Environments were created by project members in CodeCatalyst. The event provides the identity store IAM identity type and credentials for the AWS Builder ID for the user who created the Dev Environment. For resources that are created in AWS when deployed by workflows in CodeCatalyst, such as a Lambda function for a serverless deployment, the AWS account owner can view the event history for the trail associated with the separate AWS account (which is also a connected account to CodeCatalyst) for the workflow deploy action.

To investigate further, Mary can also view events for all CodeCatalyst APIs in the space by using the **list-event-logs** command in the AWS CLI.

Topics

Monitoring API calls by AWS accounts using AWS CloudTrail logging

Amazon CodeCatalyst is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures API calls made on behalf of CodeCatalyst in connected AWS accounts as events. If you create a trail, you can enable continuous delivery of CloudTrail events to an S3 bucket, including events for CodeCatalyst. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**.

CodeCatalyst supports logging the following actions as events in CloudTrail log files:

- Management events for CodeCatalyst spaces will be logged in the AWS account that is the designated billing account for the space. For more information, see CodeCatalyst space events.

  > **Note**
  >
  > Data events for CodeCatalyst spaces are accessible by using the CLI as detailed in Accessing logged events using event logging.

- Events for resources that are used in CodeCatalyst workflow actions that occur in a connected AWS account will be logged as events in that AWS account. For more information, see CodeCatalyst account connections and billing events.

> **Important**
>
> While multiple accounts can be associated with a space, CloudTrail logging for events in CodeCatalyst spaces and projects apply only for the billing account.

The space billing account is your AWS account that is charged for CodeCatalyst resources beyond the AWS Free tier. Multiple accounts can be connected to a space, while only one account can be the designated billing account. The billing account or additional connected accounts for the space can have IAM roles that are used for deploying AWS resources and infrastructure, such as an Amazon ECS cluster or S3 bucket, from CodeCatalyst workflows. You can use the workflow YAML to identify the AWS account that you deployed to.

> **Note**
>
> AWS resources that are deployed into connected accounts for CodeCatalyst workflows, are not logged as part of CloudTrail logging for the CodeCatalyst space. For example, CodeCatalyst resources include a space or project. AWS resources include an Amazon ECS service or Lambda function. CloudTrail logging must be configured separately for each AWS account where resources are deployed into.

CodeCatalyst logging in connected accounts includes the following considerations:

- Access to CloudTrail events is managed with IAM in the connected account and not in CodeCatalyst.
- Third-party connections, such as linking to a GitHub repository, will result in third-party resource names being recorded in CloudTrail logs.

> **Note**
>
> CloudTrail logging for CodeCatalyst events is at the space level and does not isolate events by project boundaries.

For more information about CloudTrail, see the AWS CloudTrail User Guide.

> **Note**
>
> This section describes CloudTrail logging for all events logged in a logged in a CodeCatalyst space and the AWS accounts that are connected to CodeCatalyst. Additionally, to review all events logged in a CodeCatalyst space, you can also use the AWS CLI and the **aws**

codecatalyst list-event-logs command. For more information, see Accessing logged events using event logging.

CodeCatalyst space events

Actions in CodeCatalyst for managing space-level and project-level resources are logged in the billing account for the space. For CloudTrail logging for a CodeCatalyst space, events are logged with the following considerations.

- CloudTrail events apply across the entire space and are not scoped to any single project.
- When you connect an AWS account to a CodeCatalyst space, loggable events for account connections will be logged in that AWS account. After you enable this connection, you cannot disable it.
- When you connect an AWS account to a CodeCatalyst space and designate it as the billing account for the space, events will be logged in that AWS account. After you enable this connection, you cannot disable it.

  Events for space-level and project-level resources are logged only in the billing account. To change the CloudTrail destination account, update the billing account in CodeCatalyst. At the beginning of the next monthly billing cycle, the change takes effect for the new billing account in CodeCatalyst. After that, the CloudTrail destination account is updated.

The following are examples of events in AWS that are related to actions in CodeCatalyst for managing space-level and project-level resources. The following APIs are released through the SDK and CLI. Events will be logged in the AWS account specified as the billing account for the CodeCatalyst space.

- `CreateDevEnvironment`
- `CreateProject`
- `DeleteDevEnvironment`
- `GetDevEnvironment`
- `GetProject`
- `GetSpace`
- `GetSubscription`

- ListDevEnvironments
- ListDevEnvironmentSessions
- ListEventLogs
- ListProjects
- ListSourceRepositories
- StartDevEnvironment
- StartDevEnvironmentSession
- StopDevEnvironment
- StopDevEnvironmentSession
- UpdateDevEnvironment

CodeCatalyst account connections and billing events

The following are examples of events in AWS that are related to actions in CodeCatalyst for account connections or billing:

- AcceptConnection
- AssociateIAMRoletoConnection
- DeleteConnection
- DissassociateIAMRolefromConnection
- GetBillingAuthorization
- GetConnection
- GetPendingConnection
- ListConnections
- ListIAMRolesforConnection
- PutBillingAuthorization
- RejectConnection

CodeCatalyst information in CloudTrail

CloudTrail is enabled on an AWS account when you create that account. When you connect that AWS account to a CodeCatalyst space, events for that space that occur in that AWS account are

logged in CloudTrail logs in that AWS account. Loggable events in CodeCatalyst are recorded as CloudTrail events in CloudTrail logs in the connected account and in **Event history** in the CloudTrail console, along with other loggable AWS events in that account.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made by a user with their AWS Builder ID.
- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Accessing CloudTrail events

For an ongoing record of events in your AWS account, including events for CodeCatalyst activity in the AWS account, create a trail. A *trail* enables CloudTrail to deliver log files to an S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

Example CodeCatalyst account connections event in AWS

The following example shows a CloudTrail log entry that demonstrates the `ListConnections` action. For an AWS account that is connected to the space, `ListConnections` is used to view all account connections to CodeCatalyst for this AWS account. The event will be logged in the AWS account specified in `accountId`, and the value of the `arn` will be the Amazon Resource Name (ARN) of the role used for the action.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "role-ARN",
        "accountId": "account-ID",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAI44QH8DHBEXAMPLE",
                "arn": "role-ARN",
                "accountId": "account-ID",
                "userName": "user-name"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-09-06T15:04:31Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-09-06T15:08:43Z",
    "eventSource": "account-ID",
    "eventName": "ListConnections",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.168.0.1",
    "userAgent": "aws-cli/1.18.147 Python/2.7.18 Linux/5.4.207-126.363.amzn2int.x86_64
 botocore/1.18.6",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "readOnly": true,
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-ID",
    "eventCategory": "Management"
}
```

Example CodeCatalyst project resource event in AWS

The following example shows a CloudTrail log entry that demonstrates the `CreateDevEnvironment` action. An AWS account that is connected to the space and is the designated billing account for the space is used for project-level events in the space, such as creating a Dev Environment.

Under `userIdentity`, in the `accountId` field, this is the IAM Identity Center account ID (432677196278) that hosts the identity pool for all AWS Builder ID identities. This account ID contains the following information about the CodeCatalyst user for the event.

- The `type` field indicates the type of IAM entity for the request. For CodeCatalyst events for space and project resources, this value is `IdentityCenterUser`. The `accountId` field specifies the account that owns the entity that was used to get credentials.
- The `userId` field contains the AWS Builder ID identifier for the user.
- The `identityStoreArn` field contains the role ARN for the identity store account and user.

The `recipientAccountId` field contains the account ID for the billing account for the space, with an example value here of 111122223333.

For more information, see the CloudTrail userIdentity element.

```
{
 "eventVersion": "1.09",
 "userIdentity": {
  "type": "IdentityCenterUser",
  "accountId": "432677196278",
  "onBehalfOf": {
   "userId": "user-ID",
   "identityStoreArn": "arn:aws:identitystore::432677196278:identitystore/d-9067642ac7"
  },
  "credentialId": "ABCDefGhiJKLMn11Lmn_1AbCDEFgHijk-AaBCdEFGHIjKLmnOPqrs11abEXAMPLE"
```

```
 },
 "eventTime": "2023-05-18T17:10:50Z",
 "eventSource": "codecatalyst.amazonaws.com",
 "eventName": "CreateDevEnvironment",
 "awsRegion": "us-west-2",
 "sourceIPAddress": "192.168.0.1",
 "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101
 Firefox/102.0",
 "requestParameters": {
  "spaceName": "MySpace",
  "projectName": "MyProject",
  "ides": [{
   "runtime": "public.ecr.aws/q6e8p2q0/cloud9-ide-runtime:2.5.1",
   "name": "Cloud9"
  }],
  "instanceType": "dev.standard1.small",
  "inactivityTimeoutMinutes": 15,
  "persistentStorage": {
   "sizeInGiB": 16
  }
 },
 "responseElements": {
  "spaceName": "MySpace",
  "projectName": "MyProject",
  "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 "
 },
 "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
 "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
 "readOnly": false,
 "eventType": "AwsApiCall",
 "managementEvent": true,
 "recipientAccountId": "111122223333",
 "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
 "eventCategory": "Management"
}
```


Note

In certain events, the user agent may not be known. In this case, CodeCatalyst will provide a value of Unknown in the `userAgent` field in the CloudTrail event.

Querying your CodeCatalyst event trails

You can create and manage queries for your CloudTrail logs using a query table in Amazon Athena. For more information about creating a query, see Querying AWS CloudTrail logs in the *Amazon Athena User Guide*.

Accessing logged events using event logging

When users perform actions in Amazon CodeCatalyst, these actions are recorded as events. You can use the AWS CLI to view logs of events in a space in a specified timeframe. You can view these events to review actions taken in the space, including the date and time of the actions, the name of the user who performed the action, and the IP address where the user made the request.

> **Note**
>
> Management events for a CodeCatalyst space are logged in CloudTrail for the connected billing account. For more information about CodeCatalyst management events that are logged by CloudTrail, see CodeCatalyst information in CloudTrail.

In order to view a log of events for a space, you must have installed and configured the AWS CLI with a profile for CodeCatalyst, and you must have the **Space administrator** role for the space. For more information, see Setting up to use the AWS CLI with CodeCatalyst and Space administrator role.

> **Note**
>
> To view logging for events that occur on behalf of CodeCatalyst in connected AWS accounts, or to view logging for events for space or project resources in the connected billing account, you can use AWS CloudTrail. For more information, see Monitoring API calls by AWS accounts using AWS CloudTrail logging.

1. Open a terminal or command line and run the **aws codecatalyst list-event-logs** command, specifying:

   - The name of the space with the **`--space-name`** option.
   - The date and time when you want to start reviewing events, in coordinated universal time (UTC) timestamp format as specified in RFC 3339, with the **`--start-time`** option.

- The date and time when you want to stop reviewing events, in coordinated universal time (UTC) timestamp format as specified in RFC 3339, with the **`--end-time`** option.
- (Optional) The maximum number of results to return in a single response, with the **`--max-results`** option. If the number of results is larger than the number you specify, the response will include a `nextToken` element which you can use to return the next results.
- (Optional) Limit the results to a specific event type you want returned, with the **`--event-name`** option.

This example returns logged events in the space named *ExampleCorp* from the time period *2022-11-30* to *2022-12-01*, and that a maximum of *2* events be returned in the response.

```
aws codecatalyst list-event-logs --space-name ExampleCorp --start-time 2022-11-30
 --end-time 2022-12-01 --event-name list-event-logs --max-results 2
```

2. If events occurred in this time frame, the command returns results similar to the following:

```
{
    "nextToken": "EXAMPLE",
    "items": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "eventName": "listEventLogs",
            "eventType": "AwsApiCall",
            "eventCategory": "MANAGEMENT",
            "eventSource": "manage",
            "eventTime": "2022-12-01T22:47:24.605000+00:00",
            "operationType": "READONLY",
            "userIdentity": {
                "userType": "USER",
                "principalId": "a1b2c3d4e5-678fgh90-1a2b-3c4d-e5f6-EXAMPLE11111"
                "userName": "MaryMajor"
            },
            "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "requestPayload": {
                "contentType": "application/json",
                "data": "{\"spaceName\":\"ExampleCorp\",\"startTime\":
\"2022-12-01T00:00:00Z\",\"endTime\":\"2022-12-10T00:00:00Z\",\"maxResults\":
\"2\"}"
            },
            "sourceIpAddress": "127.0.0.1",
```

```
            "userAgent": "aws-cli/2.9.0 Python/3.9.11 Darwin/21.3.0 exe/x86_64
 prompt/off command/codecatalyst.list-event-logs"
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "eventName": "createProject",
            "eventType": "AwsApiCall",
            "eventCategory": "MANAGEMENT",
            "eventSource": "manage",
            "eventTime": "2022-12-01T09:15:32.068000+00:00",
            "operationType": "MUTATION",
            "userIdentity": {
                "userType": "USER",
                "principalId": "a1b2c3d4e5-678fgh90-1a2b-3c4d-e5f6-EXAMPLE11111",
                "userName": "MaryMajor"
            },
            "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "requestPayload": {
                "contentType": "application/json",
                "data": "{\"spaceName\":\"ExampleCorp\",\"name\":\"MyFirstProject
\",\"displayName\":\"MyFirstProject\"}"
            },
            "responsePayload": {
                "contentType": "application/json",
                "data": "{\"spaceName\":\"ExampleCorp\",\"name\":\"MyFirstProject
\",\"displayName\":\"MyFirstProject\",\"id\":\"a1b2c3d4-5678-90ab-cdef-
EXAMPLE4444\"}"
            },
            "sourceIpAddress": "192.0.2.23",
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:102.0)
 Gecko/20100101 Firefox/102.0"
        }
    ]
}
```

3. Run the **list-event-logs** command again with the **--next-token** option and the value of the returned token to retrieve the next set of logged events that match the request.

Quotas for identity, permission, and access in CodeCatalyst

The following table describes quotas and limits for identity, permission, and access in Amazon CodeCatalyst. For more information about quotas in Amazon CodeCatalyst, see Quotas for CodeCatalyst.

Resource	Information
Aliases in CodeCatalyst	Any combination of allowed characters between 3 and 100 characters in length and must start with a letter. Valid characters: A-Z, a-z, and 0-9. Aliases cannot: • contain fewer than 3 characters • contain spaces or any of the following characters: `? ^ * [ \ ~ :`
Maximum number of invitations sent by a user per day	500
Maximum number of invitations sent to an email address per day	25
Maximum number of Personal Access Tokens (PAT) per user	100
Maximum number of personal connections for each user identity (CodeCatalyst alias) across all spaces, per provider type	1
Passwords in CodeCatalyst	Any combination of allowed characters between 8 and 64 characters in length. Valid characters: A-Z, a-z, and 0-9. Your password can include the following nonalphanumeric characters: `( ~ ! @ # $ % ^ & * _ - + = ` | \ { } [ ] : ; " ' < > , . ? /)`

Resource	Information
PAT names in CodeCatalyst	Any combination of allowed characters between 1 and 100 characters
Time until a project member invitation expires	Expires after 24 hours
Time until a space member invitation expires	Expires after 24 hours
Time until an email address verification expires	Expires 10 minutes after sending

Troubleshooting

This section can help you troubleshoot some common issues you might encounter while accessing your Amazon CodeCatalyst profile.

Problems signing up

You might encounter some issues while signing up. We've got some solutions.

My email address is already in use

If the email that you entered is already in use and you recognize it as your own, then you may already have a profile with us. Sign in with this existing identity. If you do not own the existing email, then sign up with a different, unused one.

I can't complete email verification

If you have not received your verification email

1. Check your spam, junk, and deleted items folder.

> **Note**
>
> This verification email comes from either the address `no-reply@signin.aws` or `no-reply@login.awsapps.com`. We recommend that you configure your mail system so that it accepts emails from these sender email addresses and does not handle them as junk or spam.

2. Wait 5 minutes and refresh your inbox. Check your spam, junk, and deleted items folder again.
3. If you still don't see your verification email, choose **Resend code**. If you exited that page already, then restart your workflow for signing up with Amazon CodeCatalyst.

My password doesn't meet minimum requirements

For your security, your password must include 8-20 characters, both uppercase and lowercase letters, and numbers.

Problems signing in

I forgot my password

Follow the steps in I forgot my password.

My password isn’t working

You must follow these requirements whenever you set or change your password:

- Passwords are case-sensitive.
- Passwords must be between 8 and 64 characters in length with both uppercase and lowercase letters, numbers, and at least one non-alphanumeric character.
- You can't reuse the last three passwords
  .

I can't enable MFA

To enable MFA, add one or more MFA devices to your profile by following the steps in Configure your AWS Builder ID to sign in with multi-factor authentication (MFA).

I can't add an MFA device

If you find that you can't add another MFA device, it may have reached the limit of MFA devices that you can register. You may need to remove an existing MFA device before adding a new one.

I can't remove an MFA device

If you intend to disable MFA, then proceed with removing your MFA device by following the steps in Deleting an MFA device. However, if you want to keep MFA enabled, you should add another MFA

device before attempting to delete an existing MFA device. For more information about adding another MFA device, see How to register a device for use with multi-factor authentication.

Problems signing out

I can't find where to sign out

In the top right corner of the page, choose **Sign out**.

Sign out doesn't sign me out completely

The system is designed to sign out immediately, but full sign out may take up to an hour.

I get a role does not exist error for a failed workflow

Issue: After creating a project from the web application or serverless blueprint, the workflow fails with the following error:

CLIENT_ERROR: Role does not exist

Possible solution: After you configure an IAM role with the permissions to run your workflow, and you have added the IAM role to your workflow YAML, the workflow still fails because the IAM role might need to be added to your account connection. Add the IAM role to the account connection for your space as detailed in Adding IAM roles to account connections.

I get a role error for a failed workflow

Issue: After creating a project from the web application or serverless blueprint, the workflow fails with the following error:

CLIENT_ERROR: Role not set up properly or does not exist

Possible solution: The space where the project was created might need to set up an AWS account connection or might need to complete an account connection request. If your space already has an active AWS account connection, create and add an IAM role with permissions to run workflow actions. Add the IAM role to your account connection as detailed in Adding IAM roles to account connections.

Possible solution: If the project was created without specifying a connection, then the account connection needs to be associated with the deployment environment. If your space already has an

active AWS account connection and IAM role added, you must add the account connection with the IAM role to your deployment environment as detailed in Adding the account connection and IAM roles to your deploy environment.

I need to update the IAM role in a project workflow

If the AWS account connection is set up completely, and the IAM role is created and added to the account connection, you can update the IAM role in your project workflow.

1. Choose the **CI/CD** option and choose your workflow. Choose the YAML button.
2. Choose **Edit**.
3. In the `ActionRoleArn:` field, replace the IAM role ARN with the updated IAM role ARN. Choose **Validate**.
4. Choose **Commit**.

   The workflow starts automatically if on the mainline branch. Otherwise, to rerun the workflow, choose **Run**.

I have a review request for my GitHub account after creating a personal connection

After you create a personal connection to GitHub, the CodeCatalyst app is installed for your GitHub account as a GitHub app. If there are certain resources in CodeCatalyst that require updated read or write permissions, you might need to access your GitHub account to update the permissions on the installed app.

1. Sign in to GitHub and navigate to your account settings for installed apps. Choose your profile icon, choose **Settings**, and then choose **Applications**.
2. On the **Installed GitHub Apps** tab, in the list of installed applications, view the app installed for CodeCatalyst. A **Review request** link will display if there are permissions to review.
3. Choose the link, and then confirm your credentials when prompted. Enter your credentials and choose **Verify**.
4. Accept the new permissions, indicate the repositories where you want to apply the permissions, and then choose **Save**.

How do I fill out a support form?

You can go to Amazon CodeCatalyst or fill out a Support Feedback form. In the **Request information** section, under **How can we help you**, include that you are an Amazon CodeCatalyst customer. Provide as much detail as possible so that we can most efficiently address your issue.

Add functionality to projects with extensions in CodeCatalyst

Amazon CodeCatalyst includes extensions that help you add functionality and integrate with products outside of CodeCatalyst. With extensions from the CodeCatalyst catalog, teams can customize their experiences in CodeCatalyst.

Topics

Available third-party extensions

You can add particular functionality to your CodeCatalyst project depending on the extension you choose to integrate resources with.

Integrating GitHub repositories in CodeCatalyst

GitHub is a cloud-based service that helps developers store and manage their code. The **GitHub repositories** extension lets you use linked GitHub repositories in Amazon CodeCatalyst projects. You can also link GitHub repositories when creating a new CodeCatalyst project. For more information, see Creating a project with a linked third-party repository.

Note

- You can't use empty or archived GitHub repositories with CodeCatalyst projects.
- The **GitHub repositories** extension isn't compatible with GitHub Enterprise Server repositories.

Once you install and configure the **GitHub repositories** extension, you will be able to:

- View your GitHub repositories in the list of source repositories in CodeCatalyst
- Store and manage workflow definition files in your GitHub repositories
- Create, read, update, and delete files stored in linked GitHub repositories from CodeCatalyst Dev Environments
- Store and index files from the linked GitHub repositories in CodeCatalyst
- Create CodeCatalyst projects with existing repositories of connected GitHub accounts
- Create a GitHub repository with code generated by a blueprint when creating a project with a blueprint or adding a blueprint
- Start CodeCatalyst workflow runs automatically when code is pushed to linked GitHub repositories, or when pull requests are created, modified, or closed in linked GitHub repositories
- Use linked GitHub repository source files in CodeCatalyst workflows
- Read and execute GitHub actions in CodeCatalyst workflows
- Send CodeCatalyst workflow run statuses to linked GitHub repositories, and block GitHub pull request merges based on commit statuses

Integrating Bitbucket repositories in CodeCatalyst

Bitbucket is a cloud-based service that helps developers store and manage their code. The **Bitbucket repositories** extension lets you use linked Bitbucket repositories in Amazon CodeCatalyst

projects. You can also link Bitbucket repositories when creating a new CodeCatalyst project. For more information, see Creating a project with a linked third-party repository.

Note

- You can't use empty or archived Bitbucket repositories with CodeCatalyst projects.
- The **Bitbucket repositories** extension isn't compatible with Bitbucket Data Center repositories.

Once you install and configure the **Bitbucket repositories** extension, you will be able to:

- View your Bitbucket repositories in the list of source repositories in CodeCatalyst
- Store and manage workflow definition files in your Bitbucket repositories.
- Create, read, update, and delete files stored in linked Bitbucket repositories from CodeCatalyst Dev Environments
- Create CodeCatalyst projects with existing repositories of connected Bitbucket accounts
- Store and index files from the linked Bitbucket repositories in CodeCatalyst
- Create a Bitbucket repository with code generated by a blueprint when creating a project with a blueprint or adding a blueprint
- Start CodeCatalyst workflow runs automatically when code is pushed to linked Bitbucket repositories, or when pull requests are created, modified, or closed in linked Bitbucket repositories
- Use your linked Bitbucket repository source files in CodeCatalyst workflows
- Send CodeCatalyst workflow run statuses to linked Bitbucket repositories, and block Bitbucket pull request merges based on commit statuses

Integrating GitLab repositories in CodeCatalyst

GitLab is a cloud-based service that helps developers store and manage their code. The **GitLab repositories** extension lets you use linked GitLab project repositories in Amazon CodeCatalyst projects. You can also link GitLab project repositories when creating a new CodeCatalyst project. For more information, see Creating a project with a linked third-party repository.

Note

- You can't use empty or archived GitLab project repositories with CodeCatalyst projects.
- The **GitLab repositories** extension isn't compatible with GitLab self-managed repositories.

Once you install and configure the **GitLab repositories** extension, you will be able to:

- View your GitLab project repositories in the list of source repositories in CodeCatalyst
- Store and manage workflow definition files in your GitLab project repositories.
- Create, read, update, and delete files stored in linked GitLab project repositories from CodeCatalyst Dev Environments
- Create CodeCatalyst projects with existing repositories of connected GitLab users
- Store and index files from the linked GitLab project repositories in CodeCatalyst
- Create a GitLab project repository with code generated by a blueprint when creating a project with a blueprint or adding a blueprint
- Start CodeCatalyst workflow runs automatically when code is pushed to linked GitLab project repositories, or when pull requests are created, modified, or closed in linked GitLab project repositories
- Use your linked GitLab project repository source files in CodeCatalyst workflows
- Send CodeCatalyst workflow run statuses to linked GitLab project repositories, and block GitLab merge requests based on commit statuses

Integrating Jira issues in CodeCatalyst

Jira is a software application that helps agile development teams plan, assign, track, report, and manage work. The **Jira Software** extension lets you use Jira projects in Amazon CodeCatalyst projects.

Note

CodeCatalyst is only compatible with **Jira Software Cloud**.

Once you install and configure the **Jira Software** extension for an Amazon CodeCatalyst project, you will be able to:

- Access Jira projects from CodeCatalyst by linking them to CodeCatalyst projects
- Update Jira issues with CodeCatalyst pull requests
- View status and workflow runs of linked CodeCatalyst pull requests in Jira issues

Extensions concepts

Here are some concepts and terms to know when working with extensions in CodeCatalyst.

Extensions

An *extension* is an add-on that you can install into your CodeCatalyst space to add new functionality to your projects and integrate with services outside of CodeCatalyst. Extensions can be browsed and installed from the CodeCatalyst catalog.

CodeCatalyst catalog

The CodeCatalyst catalog is a centralized listing of all the extensions available in CodeCatalyst. You can browse the CodeCatalyst catalog to find extensions that can improve your team's experiences in areas of CodeCatalyst such as source, workflows, and more.

Connecting and linking

Depending on the third-party resources you want to use or manage, you need to connect your GitHub account, Bitbucket workspace, or Jira project. Then, you need to link your GitHub repository, Bitbucket repository, or Jira project to your CodeCatalyst project.

- **GitHub repositories**: Connect GitHub account and then link GitHub repositories.
- **Bitbucket repositories**: Connect Bitbucket workspace and then link Bitbucket repositories.
- **GitLab repositories**: Connect GitLab user and then link GitLab project repositories.
- **Jira Software**: Connect Jira site and then link Jira projects.

Quickstart: Installing extensions, connecting providers, and linking resources in CodeCatalyst

This tutorial provides a walkthrough of the following three tasks:

1. Install the **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software** extension. You're prompted in an external site to connect and provide CodeCatalyst with access to your third-party resources, which is done as part of the next step.

   Important

   To install the **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software** extension to your CodeCatalyst space, you must be signed in with an account that has the **Space administrator** role in the space.

2. Connect your GitHub account, Bitbucket workspace, GitLab user, or Jira site to CodeCatalyst.

   Important

   To connect your GitHub account, Bitbucket workspace, GitLab user, or Jira site to your CodeCatalyst space, you must be both the third-party source's administrator and the CodeCatalyst **Space administrator**.

   Important

   After you install a repository extension, any repositories you link to CodeCatalyst will have their code indexed and stored in CodeCatalyst. This will make the code searchable in CodeCatalyst. To better understand the data protection for your code when using linked repositories in CodeCatalyst, see Data protection in the *Amazon CodeCatalyst User Guide*.

   Note

   If you're using a connection to a GitHub account, you must create a personal connection to establish identity mapping between your CodeCatalyst identity and your GitHub

identity. For more information, see Personal connections and Accessing GitHub resources with personal connections.

3. Link your GitHub repository, Bitbucket repository, GitLab project repository, or Jira project to your CodeCatalyst project.

Important

- While you can link a GitHub repository, Bitbucket repository, or GitLab project repository as a **Contributor**, you can only unlink a third-party repository as the **Space administrator** or the **Project administrator**. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
- To link your Jira project to your CodeCatalyst project, you must be the CodeCatalyst **Space administrator** or CodeCatalyst **Project administrator**.

Important

CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
As a best practice, always make sure you have the latest version of the extension before you link a repository.

Note

- A GitHub repository, Bitbucket repository, or GitLab project repository can only be linked to one CodeCatalyst project in a space.
- You can't use empty or archived GitHub repositories, Bitbucket repositories, or GitLab project repositories with CodeCatalyst projects.
- You can't link a GitHub repository, Bitbucket repository, or GitLab project repository that has the same name as a repository in a CodeCatalyst project.

- The **GitHub repositories** extension isn't compatible with GitHub Enterprise Server repositories.
- The **Bitbucket repositories** extension isn't compatible with Bitbucket Data Center repositories.
- The **GitLab repositories** extension isn't compatible with GitLab self-managed project repositories.
- You can't use the **Write description for me** or **Summarize comments** features with linked repositories. These features are only available in pull requests in CodeCatalyst.
- A CodeCatalyst project can only be linked to one Jira project. A Jira project can be linked to multiple CodeCatalyst projects.

You can also install the **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories** extension, connect to your GitHub account, Bitbucket workspace, or GitLab user, and link third-party repositories when creating a new CodeCatalyst project. For more information, see Creating a project with a linked third-party repository.

Topics

Step 1: Install a third-party extension from the CodeCatalyst catalog

The first step to using thid-party resources in CodeCatalyst is to install the **GitHub repositories** extension from the CodeCatalyst catalog. To install the extension, perform the following steps, choosing the extension for the third-party resources you want to use. **GitHub repositories**, **Bitbucket repositories**, and **GitLab repositories** allow you to use GitHub repositories, Bitbucket repositories, or GitLab project repositories in CodeCatalyst. **Jira Software** allows you to manage Jira issues in CodeCatalyst.

To install an extension from the CodeCatalyst catalog

1. Open the CodeCatalyst console at https://codecatalyst.aws/.

2. Navigate to your CodeCatalyst space.
3. Navigate to the CodeCatalyst CodeCatalyst catalog by choosing the **Catalog** icon

   in the top menu. You can search for **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software**. You can also filter extensions based on categories.
4. (Optional) To see more details about the extension, such as the permissions the extension will have, choose the extension name.
5. Choose **Install**. Review the permissions required by the extension, and if you want to continue, choose **Install** again.

After installing the extension, you are taken to the extension details page. Depending on the extension you installed, you can view and manage connected providers and linked resources.

Step 2: Connect your third-party provider to your CodeCatalyst space

After you install the **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software** extension, the next step is to connect your GitHub account, Bitbucket workspace, GitLab project repository, or Jira site to your CodeCatalyst space.

To connect your GitHub account, Bitbucket workspace, or Jira site to CodeCatalyst

- Do one of the following depending on the third-party extension you installed:
  - **GitHub repositories**: Connect to a GitHub account.
    1. In the **Connected GitHub accounts** tab, choose **Connect GitHub account** to go to the external site for GitHub.
    2. Sign in to your GitHub account using your GitHub credentials, and then choose the account where you want to install Amazon CodeCatalyst.

       Tip

       If you have previously connected a GitHub account to the space, you will not be prompted to reauthorize. You will instead see a dialog box asking you where you would like to install the extension if you are a member or collaborator in more than one GitHub organization, or the configuration page for the Amazon CodeCatalyst application if you only belong to one GitHub organization.

Configure the application for the repository access that you want to allow, and then choose **Save**. If the **Save** button is not active, make a change to the configuration, and then try again.

3. Choose whether you want to allow CodeCatalyst to access all current and future repositories, or choose the specific GitHub repositories you want to use in CodeCatalyst. The default option is to include all GitHub repositories in the GitHub account, including future repositories that will be accessed by CodeCatalyst.
4. Review the permissions given to CodeCatalyst, and then choose **Install**.

After connecting your GitHub account to CodeCatalyst, you're taken to the **GitHub repositories** extension details page, where you can view and manage connected GitHub accounts and linked GitHub repositories.

- **Bitbucket repositories**: Connect to a Bitbucket workspace.

1. In the **Connected Bitbucket workspaces** tab, choose **Connect Bitbucket workspace** to go to the external site for Bitbucket.
2. Sign into your Bitbucket workspace using your Bitbucket credentials and review the permissions given to CodeCatalyst.
3. From the **Authorize for workspace** dropdown menu, choose the Bitbucket workspace you want to provide CodeCatalyst access to, and then choose **Grant access**.

> **Tip**
>
> If you have previously connected a Bitbucket workspace to the space, you will not be prompted to reauthorize. You will instead see a dialog asking you where you would like to install the extension if you're a member or collaborator in more than one Bitbucket workspace, or the configuration page for the Amazon CodeCatalyst application if you only belong to one Bitbucket workspace. Configure the application for the workspace access you want to allow, and then choose **Grant access**. If the **Grant access** button is not active, make a change to the configuration, and then try again.

After connecting your Bitbucket workspace to CodeCatalyst, you're taken to the **Bitbucket repositories** extension details page, where you can view and manage connected Bitbucket workspaces and linked Bitbucket repositories.

- **GitLab repositories**: Connect to a GitLab user.

  1. Choose **Connect GitLab user** to go to the external site for GitLab.
  2. Sign in to your GitLab user using your GitLab credentials and review the permissions given to CodeCatalyst.

     > **Tip**
     >
     > If you have previously connected a GitLab user to the space, you will not be prompted to reauthorize. You will instead be navigated back to the CodeCatalyst console.

  3. Choose **Authorize AWS Connector for GitLab**.

  After connecting your GitLab user to CodeCatalyst, you're taken to the **GitLab repositories** extension details page, where you can view and manage connected GitLab user and linked GitLab project repositories.

- **Jira Software**: Connect a Jira site.

  1. In the **Connected Jira sites** tab, choose **Connect Jira site** to go to the external site for Atlassian Marketplace.
  2. Choose **Get it now** to get started with installing CodeCatalyst on your Jira site.

     > **Note**
     >
     > If you previously installed CodeCatalyst to your Jira site, you will be notified. Choose **Get started** to be taken to the final step.

  3. Depending on your role, do one of the following:

     1. If you are a Jira site administrator, from the site dropdown menu, choose the Jira site to install the CodeCatalyst application, and then choose **Install app**.

> **Note**
>
> If you have one Jira site, this step won't appear, and you'll automatically be directed to the next step.

2. a. If you aren't a Jira administrator, from the site dropdown menu, choose the Jira site to install the CodeCatalyst application, and then choose **Request app**. For more information on installing Jira apps, see Who can install apps?.

   b. Enter the reason you need to install CodeCatalyst into the input text field or keep the default text, and then choose **Submit request**.

4. Review the actions performed by CodeCatalyst when the application is installed, and then choose **Get it now**.
5. After the application is installed, choose **Return to CodeCatalyst** to return to CodeCatalyst.

After connecting your Jira site to CodeCatalyst, you can view the connected site in the **Connected Jira sites** tab of the **Jira Software** extension details page.

Step 3: Link your third-party resources to your CodeCatalyst project

The third and final step to using your GitHub repositories, Bitbucket repositories, or GitLab project repositories or manage Jira issues in CodeCatalyst is to link them to the CodeCatalyst project in which you want to use it.

To link a GitHub repository, Bitbucket repository, GitLab project repository, or Jira project to a CodeCatalyst project from the extension details page

- Do one of the following depending on the third-party extension you installed and provider you connected:
  - **GitHub repositories**: Link a GitHub repository.
    1. In the **Linked GitHub repositories** tab, choose **Link GitHub repository**.
    2. From the **GitHub account** dropdown, choose the GitHub account that contains the repository that you want to link.

3. From the **GitHub repository** dropdown, choose the repository you want to link to a CodeCatalyst project.

   > **Tip**
   >
   > If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the space.

4. (Optional) If you don't see a GitHub repository in the list of repositories, it might not have been configured for repository access in the Amazon CodeCatalyst application in GitHub. You can configure which GitHub repositories can be used in CodeCatalyst in the connected account.

   a. Navigate to your GitHub account, choose **Settings**, and then choose **Applications**.

   b. In the **Installed GitHub Apps** tab, choose **Configure** for the Amazon CodeCatalyst application.

   c. Do one of the following to configure access of GitHub repositories you want to link in CodeCatalyst:

      - To provide access to all current and future repositories, choose **All repositories**.
      - To provide access to specific repositories, choose **Only select repositories**, choose the **Select repositories** dropdown, and then choose a repository you want to allow to link in CodeCatalyst.

5. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link the GitHub repository to.

6. Choose **Link**.

If you no longer want to use a GitHub repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a repository is unlinked, events in that repository will not start workflow runs, and you will not be able to use that repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

- **Bitbucket repositories**: Link a Bitbucket repository.

  1. In the **Linked Bitbucket repositories** tab, choose **Link Bitbucket repository**.

2. From the **Bitbucket workspace** dropdown, choose the Bitbucket workspace that contains the repository that you want to link.
3. From the **Bitbucket repository** dropdown, choose the repository you want to link to a CodeCatalyst project.

   > **Tip**
   >
   > If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the space.

4. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link the Bitbucket repository to.
5. Choose **Link**.

If you no longer want to use a Bitbucket repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a repository is unlinked, events in that repository will not start workflow runs, and you will not be able to use that repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

- **GitLab repositories**: Link a GitLab project repository.

  1. In the **Linked GitLab project repositories** tab, choose **Link GitLab project repository**.
  2. From the **GitLab user** dropdown, choose the GitLab user that contains the repository that you want to link.
  3. From the **GitLab project repository** dropdown, choose the repository you want to link to a CodeCatalyst project.

     > **Tip**
     >
     > If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the space.

  4. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link the GitLab project repository to.
  5. Choose **Link**.

If you no longer want to use a GitLab project repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a project repository is unlinked, events in that project repository will not start workflow runs, and you will not be able to use that project repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

- **Jira Software**: Link a Jira project.

  1. In the **Linked Jira projects** tab, choose **Link Jira project**.
  2. From the **Jira site** dropdown menu, choose the Jira site that contains the project that you want to link.
  3. From the **Jira project** dropdown menu, choose the project you want to link to a CodeCatalyst project.
  4. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link to a Jira project.
  5. Choose **Link**.

  Once a Jira project is linked to a CodeCatalyst project, access to CodeCatalyst issues is disabled entirely, and **Issues** in the CodeCatalyst navigation pane will be replaced with a **Jira issues** item that links to the Jira project.

  If you no longer want to use a Jira project in CodeCatalyst, you can unlink it from your CodeCatalyst project. When a Jira project is unlinked, Jira issues will not be available in the CodeCatalyst project, and CodeCatalyst **Issues** will be the issue provider again. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

You can also link your GitHub repository, Bitbucket repository, or GitLab project repository to a project from **Source repositories** in **Code**. For more information, see Linking resources from connected third-party providers.

Next steps

After installing the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension, connecting your resource provider, and linking your third-party repositories to your CodeCatalyst

projects, you can use it in CodeCatalyst workflows and Dev Environments. You can also create third-party repositories in the connected GitHub account, Bitbucket workspace, or GitLab user with code generated from a blueprint. For more information, see Automatically starting a workflow run after third-party repository events and Creating a Dev Environment.

After installing the **Jira Software** extension, connecting your Jira site, linking your Jira projects to your CodeCatalyst project, and linking a pull request, updates from CodeCatalyst are reflected in your Jira project. For more information about linking pull requests to Jira issues, see Linking Jira issues to CodeCatalyst pull requests. For more information on viewing CodeCatalyst events in Jira, see Viewing CodeCatalyst events in Jira issues.

Installing an extension in a space

You can install extensions for your CodeCatalyst space that add functionality to projects in that space. You can view the CodeCatalyst catalog by choosing the **Catalog** icon

.

To learn more about the extensions and their functionalities, see Available third-party extensions.

> **Important**
>
> To install an extension, you must be signed in with an account that has the **Space administrator** role in the space.

> **Important**
>
> After you install a repository extension, any repositories you link to CodeCatalyst will have their code indexed and stored in CodeCatalyst. This will make the code searchable in CodeCatalyst. To better understand the data protection for your code when using linked repositories in CodeCatalyst, see Data protection in the *Amazon CodeCatalyst User Guide*.

To install an extension from the CodeCatalyst catalog

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

3. Navigate to the CodeCatalyst catalog by choosing the **Catalog** icon

   in the top menu. You can search for **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software**. You can also filter extensions based on categories.
4. (Optional) Choose the name of the extension to see more details about the extension, such as the permissions the extension will have.
5. Choose **Install**. Review the permissions required by the extension, and if you want to continue, choose **Install** again.

After installing an extension, you will see the details page for the installed extension. Browse the tabs for more information about the extension. The details page is also where you will perform further configuration of the extension if needed.

Uninstalling an extension in a space

You can uninstall extensions that were previously installed in your CodeCatalyst space. Uninstalling an extension may remove resources related to that extension from your CodeCatalyst space or projects.

> **Important**
>
> To uninstall an extension, you must be signed in with an account that has the **Space administrator** role in the space.

To uninstall an extension from your CodeCatalyst space

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.
3. Do one of the following to view a list of the installed extensions for your space:

   a. Choose **Settings**, and then choose **Installed extensions**.

   b. Choose the **Catalog** icon

      in the top menu.

4. Choose **Configure** on the extension you want to uninstall.
5. Choose **Uninstall** on the extension details page.
6. Review the information in the **Uninstall extension** dialog box. Follow the instructions, and then choose **Uninstall** to uninstall the extension.

Connecting GitHub accounts, Bitbucket workspaces, GitLab users, and Jira sites CodeCatalyst

To use a GitHub repository, Bitbucket repository, or GitLab project repository or manage a Jira project in CodeCatalyst, you must first connect your third-party source to your CodeCatalyst space. To learn more about the extensions and their functionalities, see Available third-party extensions.

Important

To connect your GitHub account, Bitbucket workspace, GitLab user, or Jira site to your CodeCatalyst space, you must be both the third-party source's administrator and the CodeCatalyst **Space administrator**.

Note

If you're using a connection to a GitHub account, you must create a personal connection to establish identity mapping between your CodeCatalyst identity and your GitHub identity. For more information, see Personal connections and Accessing GitHub resources with personal connections.

To connect your GitHub account, Bitbucket workspace, GitLab user, or Jira site to CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.
3. Do one of the following to view a list of the installed extensions for your space:

    a. Choose **Settings**, and then choose **Installed extensions**.

b. Choose the **Catalog** icon

in the top menu.

4. Choose **Configure** for one of the following extensions you want to configure: **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software**.
5. Do one of the following depending on the third-party extension you chose to configure:

- **GitHub repositories**: Connect to a GitHub account.

  1. In the **Connected GitHub accounts** tab, choose **Connect GitHub account** to go to the external site for GitHub.
  2. Sign in to your GitHub account using your GitHub credentials, and then choose the account where you want to install Amazon CodeCatalyst.

  > **Tip**
  >
  > If you have previously connected a GitHub account to the space, you will not be prompted to reauthorize. You will instead see a dialog box asking you where you would like to install the extension if you are a member or collaborator in more than one GitHub space, or the configuration page for the Amazon CodeCatalyst application if you only belong to one GitHub space. Configure the application for the repository access that you want to allow, and then choose **Save**. If the **Save** button is not active, make a change to the configuration, and then try again.

  3. Choose whether you want to allow CodeCatalyst to access all current and future repositories, or choose the specific GitHub repositories you want to use in CodeCatalyst. The default option is to include all GitHub repositories in the GitHub account, including future repositories that will be accessed by CodeCatalyst.
  4. Review the permissions given to CodeCatalyst, and then choose **Install**.

  After connecting your GitHub account to CodeCatalyst, you're taken to the **GitHub repositories** extension details page, where you can view and manage connected GitHub accounts and linked GitHub repositories.

- **Bitbucket repositories**: Connect to a Bitbucket workspace.

1. In the **Connected Bitbucket workspaces** tab, choose **Connect Bitbucket workspace** to go to the external site for Bitbucket.
2. Sign into your Bitbucket workspace using your Bitbucket credentials and review the permissions given to CodeCatalyst.
3. From the **Authorize for workspace** dropdown menu, choose the Bitbucket workspace you want to provide CodeCatalyst access to, and then choose **Grant access**.

> **Tip**
>
> If you have previously connected a Bitbucket workspace to the space, you will not be prompted to reauthorize. You will instead see a dialog asking you where you would like to install the extension if you're a member or collaborator in more than one Bitbucket workspace, or the configuration page for the Amazon CodeCatalyst application if you only belong to one Bitbucket workspace. Configure the application for the workspace access you want to allow, and then choose **Grant access**. If the **Grant access** button is not active, make a change to the configuration, and then try again.

After connecting your Bitbucket workspace to CodeCatalyst, you're taken to the **Bitbucket repositories** extension details page, where you can view and manage connected Bitbucket workspaces and linked Bitbucket repositories.

- **GitLab repositories**: Connect to a GitLab user.

1. Choose **Connect GitLab user** to go to the external site for GitLab.
2. Sign in to your GitLab user using your GitLab credentials and review the permissions given to CodeCatalyst.

> **Tip**
>
> If you have previously connected a GitLab user to the space, you will not be prompted to reauthorize. You will instead be navigated back to the CodeCatalyst console.

3. Choose **Authorize AWS Connector for GitLab**.

After connecting your GitLab user to CodeCatalyst, you're taken to the **GitLab repositories** extension details page, where you can view and manage connected GitLab user and linked GitLab project repositories.

- **Jira Software**: Connect a Jira site.

  1. In the **Connected Jira sites** tab, choose **Connect Jira site** to go to the external site for Atlassian Marketplace.
  2. Choose **Get it now** to get started with installing CodeCatalyst on your Jira site.

     > **Note**
     >
     > If you previously installed CodeCatalyst to your Jira site, you will be notified. Choose **Get started** to be taken to the final step.

  3. Depending on your role, do one of the following:

     1. If you are a Jira site administrator, from the site dropdown menu, choose the Jira site to install the CodeCatalyst application, and then choose **Install app**.

        > **Note**
        >
        > If you have one Jira site, this step won't appear, and you'll automatically be directed to the next step.

     2. a. If you aren't a Jira administrator, from the site dropdown menu, choose the Jira site to install the CodeCatalyst application, and then choose **Request app**. For more information on installing Jira apps, see Who can install apps?.

        b. Enter the reason you need to install CodeCatalyst into the input text field or keep the default text, and then choose **Submit request**.

  4. Review the actions performed by CodeCatalyst when the application is installed, and then choose **Get it now**.
  5. After the application is installed, choose **Return to CodeCatalyst** to return to CodeCatalyst.

  After connecting your Jira site to CodeCatalyst, you can view the connected site in the **Connected Jira sites** tab of the **Jira Software** extension details page.

If you no longer want to use GitHub repositories, Bitbucket repositories, or GitLab project repositories, or manage Jira issues in CodeCatalyst, you can disconnect your third-party source. When a GitHub account, Bitbucket workspace, or GitLab user is disconnected, events in the third-party repositories will not start workflow runs, and you will not be able to use those repositories with CodeCatalyst Dev Environments. When a Jira site is disconnected, Jira issues from the site's projects will not be available in the CodeCatalyst projects, and CodeCatalyst **Issues** will be the issue provider again. For more information, see Disconnecting GitHub accounts, Bitbucket workspaces, GitLab users, and Jira sites CodeCatalyst.

Disconnecting GitHub accounts, Bitbucket workspaces, GitLab users, and Jira sites CodeCatalyst

If you no longer want to use GitHub repositories, Bitbucket repositories, or GitLab project repositories, or manage Jira issues in CodeCatalyst, you can disconnect your third-party source. Once a GitHub account, Bitbucket workspace, or GitLab user is disconnected, events in the repositories will not start workflow runs, and you will not be able to use those repositories with CodeCatalyst Dev Environments. When a Jira site is disconnected, Jira issues from the site's projects will not be available in the CodeCatalyst projects, and CodeCatalyst **Issues** will be the issue provider again.

Note

- To disconnect a GitHub account, you must first unlink all linked GitHub repositories from that account.
- To disconnect a Bitbucket workspace, you must first unlink all linked Bitbucket repositories from that workspace.
- To disconnect a GitLab user, you must first unlink all linked GitLab project repositories from that workspace.
- To disconnect a Jira site, you must first unlink all linked Jira projects from that account.

For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

To disconnect a GitHub project, Bitbucket workspace, GitLab user, or Jira site

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.
3. Do one of the following to view a list of the installed extensions for your space:

   a. Choose **Settings**, and then choose **Installed extensions**.

   b. Choose the **Catalog** icon

   in the top menu.
4. Choose **Configure** for one of the following extensions you want to configure: **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software**.
5. Do one of the following depending on the third-party extension you chose to configure:

   - **GitHub repositories**: Disonnect to a GitHub account.

     In the **Connected GitHub accounts** tab, choose the GitHub account you want to disconnect, and then choose **Disconnect GitHub account**.
   - **Bitbucket repositories**: Disonnect to a Bitbucket workspace.

     In the **Connected Bitbucket workspaces** tab, choose the Bitbucket workspace you want to disconnect, and then choose **Disconnect Bitbucket workspace**.
   - **GitLab repositories**: Disonnect to a GitLab user.

     In the **Connected GitLab users** tab, choose the GitLab user you want to disconnect, and then choose **Disconnect GitLab user**.
   - **Jira Software**: Disonnect to a Jira site.

     In the **Connected Jira sites** tab, choose the Jira site you want to disconnect, and then choose **Disconnect Jira site**.
6. In the **Disconnect** dialog box, review the effects of disconnecting the account.
7. Enter **disconnect** into the text input field, and then choose **Disconnect**.

Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst

Before you can use a GitHub repository, Bitbucket repository, or GitLab project repository, or manage a Jira project, you must connect the third-party source that the repository or project belongs to with your CodeCatalyst space. For more information, see Connecting GitHub accounts, Bitbucket workspaces, GitLab users, and Jira sites CodeCatalyst.

You can use linked GitHub repositories, Bitbucket repositories, or GitLab project repositories in workflows, where events in the linked repositories start workflows that might build, test, or deploy code, depending on the workflow configuration. Workflow configuration files for workflows that use linked GitHub or Bitbucket repositories are stored in the linked repositories. Linked repositories can also be used with Dev Environments to create, update, and delete files in the linked repositories. You can link a GitHub repository, Bitbucket repository, or GitLab project repository to a CodeCatalyst project from either the details page of the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension, or from the **Source repositories** view in **Code** in the project itself.

Important

While you can link a GitHub or Bitbucket repository as a **Contributor**, you can only unlink a third-party repository as the **Space administrator** or the **Project administrator**. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

Important

After you install a repository extension, any repositories you link to CodeCatalyst will have their code indexed and stored in CodeCatalyst. This will make the code searchable in CodeCatalyst. To better understand the data protection for your code when using linked repositories in CodeCatalyst, see Data protection in the *Amazon CodeCatalyst User Guide*.

Important

CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again.
As a best practice, always make sure you have the latest version of the extension before you link a repository.

You can use linked Jira projects to manage issues and to link CodeCatalyst pull requests to a Jira issue. Summary status of a pull request and the status of associated CodeCatalyst workflow events are reflected in your Jira issue.

Important

To link your Jira project to your CodeCatalyst project, you must be the CodeCatalyst **Space administrator** or CodeCatalyst **Project administrator**.

Note

- A GitHub repository, Bitbucket repository, or GitLab project repository can only be linked to one CodeCatalyst project in a space.
- You can't use empty or archived GitHub repositories, Bitbucket repositories, or GitLab project repositories with CodeCatalyst projects.
- You can't link a GitHub repository, Bitbucket repository, or GitLab repository that has the same name as a repository in a CodeCatalyst project.
- The **GitHub repositories** extension isn't compatible with GitHub Enterprise Server repositories.
- The **Bitbucket repositories** extension isn't compatible with Bitbucket Data Center repositories.
- The **GitLab repositories** extension isn't compatible with GitLab self-managed project repositories.
- You can't use the **Write description for me** or **Summarize comments** features with linked repositories. These features are only available in pull requests in CodeCatalyst.

- A CodeCatalyst project can only be linked to one Jira project. A Jira project can be linked to multiple CodeCatalyst projects.

Topics

Linking resources from connected third-party providers

To link a GitHub repository, Bitbucket repository, GitLab project repository, or Jira project to a CodeCatalyst project from the extension details page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.
3. Do one of the following to view a list of the installed extensions for your space space:

   a. Choose **Settings**, and then choose **Installed extensions**.

   b. Choose the **Catalog** icon

   in the top menu.
4. Choose **Configure** for one of the following extensions: **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software**.
5. Do one of the following depending on the third-party extension you chose to configure:

   - **GitHub repositories**: Link a GitHub repository.

     1. In the **Linked GitHub repositories** tab, choose **Link GitHub repository**.
     2. From the **GitHub account** dropdown, choose the GitHub account that contains the repository that you want to link.
     3. From the **GitHub repository** dropdown, choose the repository you want to link to a CodeCatalyst project.

> **Tip**
>
> If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the space.

4. (Optional) If you don't see a GitHub repository in the list of repositories, it might not have been configured for repository access in the Amazon CodeCatalyst application in GitHub. You can configure which GitHub repositories can be used in CodeCatalyst in the connected account.

   a. Navigate to your GitHub account, choose **Settings**, and then choose **Applications**.

   b. In the **Installed GitHub Apps** tab, choose **Configure** for the Amazon CodeCatalyst application.

   c. Do one of the following to configure access of GitHub repositories you want to link in CodeCatalyst:

      - To provide access to all current and future repositories, choose **All repositories**.
      - To provide access to specific repositories, choose **Only select repositories**, choose the **Select repositories** dropdown, and then choose a repository you want to allow to link in CodeCatalyst.

5. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link the GitHub repository to.

6. Choose **Link**.

If you no longer want to use a GitHub repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a repository is unlinked, events in that repository will not start workflow runs, and you will not be able to use that repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

- **Bitbucket repositories**: Link a Bitbucket repository.

  1. In the **Linked Bitbucket repositories** tab, choose **Link Bitbucket repository**.
  2. From the **Bitbucket workspace** dropdown, choose the Bitbucket workspace that contains the repository that you want to link.

3. From the **Bitbucket repository** dropdown, choose the repository you want to link to a CodeCatalyst project.

   > **Tip**
   >
   > If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the space.

4. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link the Bitbucket repository to.
5. Choose **Link**.

If you no longer want to use a Bitbucket repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a repository is unlinked, events in that repository will not start workflow runs, and you will not be able to use that repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

- **GitLab repositories**: Link a GitLab project repository.

  1. In the **Linked GitLab project repositories** tab, choose **Link GitLab project repository**.
  2. From the **GitLab user** dropdown, choose the GitLab user that contains the project repository that you want to link.
  3. From the **GitLab project repository** dropdown, choose the repository you want to link to a CodeCatalyst project.

     > **Tip**
     >
     > If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the space.

  4. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link the GitLab project repository to.
  5. Choose **Link**.

  If you no longer want to use a GitLab project repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a project repository is unlinked, events in that

project repository will not start workflow runs, and you will not be able to use that project repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

- **Jira Software**: Link a Jira project.

  1. In the **Linked Jira projects** tab, choose **Link Jira project**.
  2. From the **Jira site** dropdown menu, choose the Jira site that contains the project that you want to link.
  3. From the **Jira project** dropdown menu, choose the project you want to link to a CodeCatalyst project.
  4. From the **CodeCatalyst project** dropdown menu, choose the CodeCatalyst project you want to link to a Jira project.
  5. Choose **Link**.

  Once a Jira project is linked to a CodeCatalyst project, access to CodeCatalyst issues is disabled entirely, and **Issues** in the CodeCatalyst navigation pane will be replaced with a **Jira issues** item that links to the Jira project.

  If you no longer want to use a Jira project in CodeCatalyst, you can unlink it from your CodeCatalyst project. When a Jira project is unlinked, Jira issues will not be available in the CodeCatalyst project, and CodeCatalyst **Issues** will be the issue provider again. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

To link a GitHub repository, Bitbucket repository, or GitLab project repository to a CodeCatalyst project from the source repositories page in a project

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst project.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose **Add repository**, and then choose **Link repository**.
5. From the **Repository provider** dropdown menu, choose one of the following third-party repository providers: **GitHub**, **Bitbucket**, **GitLab**.
6. Do one of the following depending on the third-party repository provider you chose to link:

- **GitHub repositories**: Link a GitHub repository.

  1. From the **GitHub account** dropdown menu, choose the GitHub account that contains the repository that you want to link.
  2. From the **GitHub repository** dropdown menu, choose the GitHub repository you want to link your CodeCatalyst project.

     > **Tip**
     >
     > If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the Amazon CodeCatalyst.

  3. (Optional) If you don't see a GitHub repository in the list of repositories, it might not have been configured for repository access in the Amazon CodeCatalyst application in GitHub. You can configure which GitHub repositories can be used in CodeCatalyst in the connected account.

     a. Navigate to your GitHub account, choose **Settings**, and then choose **Applications**.

     b. In the **Installed GitHub Apps** tab, choose **Configure** for the Amazon CodeCatalyst application.

     c. Do one of the following to configure access of GitHub repositories you want to link in CodeCatalyst:

        - To provide access to all current and future repositories, choose **All repositories**.
        - To provide access to specific repositories, choose **Only select repositories**, choose the **Select repositories** dropdown, and then choose a repository you want to allow to link in CodeCatalyst.

- **Bitbucket repositories**: Link a Bitbucket repository.

  1. From the **Bitbucket workspace** dropdown menu, choose the Bitbucket workspace that contains the repository that you want to link.
  2. From the **Bitbucket repository** dropdown menu, choose the Bitbucket repository you want to link your CodeCatalyst project.

> **Tip**
>
> If the name of the repository is greyed out, you can't link that repository because it has already been linked to another project in the Amazon CodeCatalyst.

- **GitLab repositories**: Link a GitLab project repository.

  1. From the **GitLab user** dropdown menu, choose the GitLab user that contains the project repository that you want to link.
  2. From the **GitLab project repository** dropdown menu, choose the GitLab project repository you want to link your CodeCatalyst project.

  > **Tip**
  >
  > If the name of the project repository is greyed out, you can't link that project repository because it has already been linked to another project in the Amazon CodeCatalyst.

7. Choose **Link**.

If you no longer want to use a GitHub repository, Bitbucket repository, or GitLab project repository in CodeCatalyst, you can unlink it from a CodeCatalyst project. When a repository is unlinked, events in that repository will not start workflow runs, and you will not be able to use that repository with CodeCatalyst Dev Environments. For more information, see Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

After linking your GitHub repository, Bitbucket repository, or GitLab project repository to your CodeCatalyst project, you can use it in CodeCatalyst workflows and Dev Environments. You can also use the linked repositories with Amazon Q Developer, blueprints, and more. For more information, see Automatically starting a workflow run after third-party repository events and Creating a Dev Environment.

After linking your Jira project to your CodeCatalyst project, and linking a pull request, updates from CodeCatalyst are reflected in your Jira project. For more information about linking pull requests to Jira issues, see Linking Jira issues to CodeCatalyst pull requests. For more information on viewing CodeCatalyst events in Jira, see Viewing CodeCatalyst events in Jira issues.

Linking a third-party repository to during CodeCatalyst project creation

You can link a GitHub repository, Bitbucket repository, or GitLab project respository to a new CodeCatalyst project when creating the new CodeCatalyst project. For more information, see Creating a project with a linked third-party repository.

Unlinking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst

If you no longer want to use a GitHub repository, Bitbucket repository, or GitLab project repository, or manage a Jira project in CodeCatalyst, you can unlink the repository or project from your CodeCatalyst project.

Unlinking a GitHub repository, Bitbucket repository, or GitLab project repository doesn't delete the repository or make any changes to it. It doesn't delete any workflow configuration files stored in that linked repository. However, once you unlink a GitHub repository, Bitbucket repository, or GitLab project repository, events in that repository will no longer start workflow runs, and you can't use the repository with Dev Environments. You can unlink a GitHub repository, Bitbucket repository, or GitLab project repository from a CodeCatalyst project from either the details page of the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension, or from the **Source repositories** view in **Code** in the project itself.

Unlinking a Jira project doesn't delete the project, including planning items or development information, or make any changes to it. However, once you unlink a Jira project, the project's Jira issues will no longer be available to link to the CodeCatalyst project, and CodeCatalyst **Issues** will be the issue provider again.

> **Important**
>
> To unlink your GitHub repository, Bitbucket repository, or Gitlab project repository from your CodeCatalyst project, you must be the **Space administrator** or the **Project administrator**.

To unlink a GitHub repository, Bitbucket repository, GitLab project repository, or Jira project in a CodeCatalyst project from the extension details page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.
3. Do one of the following to view a list of the installed extensions for your space:

   a. Choose **Settings**, and then choose **Installed extensions**.

   b. Choose the **Catalog** icon

   in the top menu.
4. Choose **Configure** for one of the following extensions you want to configure: **GitHub repositories**, **Bitbucket repositories**, **GitLab repositories**, or **Jira Software**.
5. Do one of the following depending on the third-party extension you chose to configure:

   - **GitHub repositories**: Unlink a GitHub repository.

     In the **GitHub repositories** tab, choose the GitHub repository you want to unlink, and then choose **Unlink GitHub repository**.
   - **Bitbucket repositories**: Unlink a Bitbucket repository.

     In the **Bitbucket repositories** tab, choose the Bitbucket repository you want to unlink, and then choose **Unlink Bitbucket repository**.
   - **GitLab repositories**: Unlink a GitLab project repository.

     In the **GitLab project repositories** tab, choose the GitLab project repository you want to unlink, and then choose **Unlink GitLab project repository**.
   - **Jira Software**: Unlink a Jira project.

     In the **Jira projects** tab, choose the Jira project you want to unlink, and then choose **Unlink Jira project**.
6. In the **Unlink** dialog box, review the effects of unlinking the repository.
7. Enter **unlink** into the text input field and choose **Unlink**.

To unlink a GitHub repository, Bitbucket repository, or GitLab project repository in a CodeCatalyst project from the source repositories page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst project.
3. In the navigation pane, choose **Code**, and then choose **Source repositories**.
4. Choose the radio button of the repository you want to unlink, and then choose **Unlink repository**.
5. Review the information in the dialog box. Follow the instructions, and then choose **Unlink** to unlink the repository.

Viewing third-party repositories and searching Jira issues in CodeCatalyst

After linking GitHub repositories, Bitbucket repositories, or GitLab project repositories, you can view them in CodeCatalyst to confirm and configure the resources. You can also search for linked Jira issues in CodeCatalyst.

Topics

Viewing third-party repositories in CodeCatalyst

You can view the linked GitHub repositories, Bitbucket repositories, or GitLab project repositories in the list of source repositories for your project or from the **GitHub repositories**, **Bitbucket repositories**, or **GitLab repositories** extension details page. Choosing them from the list of repositories doesn't open them in CodeCatalyst. Instead, they open in the third-party repository provider, where you can view and work on the code in the linked repository.

To view linked GitHub repositories, Bitbucket repositories, or GitLab project repositories in CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst project.

3. In the navigation pane, choose **Code**, and then choose **Source repositories**.

To view linked GitHub repositories, Bitbucket repositories, or GitLab project repositories from the extension details page

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space, and then choose the **Installed extensions** tab.
3. Depending on the third-party repositories you want to view, do one of the following:

   - In **GitHub repositories**, choose **Configure**, and then choose **Linked GitHub repositories** to view all GitHub repositories connected to CodeCatalyst projects in your CodeCatalyst space.
   - In **Bitbucket repositories**, choose **Configure**, and then choose **Linked Bitbucket repositories** to view all Bitbucket repositories connected to CodeCatalyst projects in your CodeCatalyst space.
   - In **GitLab repositories**, choose **Configure**, and then choose **Linked GitLab project repositories** to view all GitLab project repositories connected to CodeCatalyst projects in your CodeCatalyst space.

The GitHub repositories, Bitbucket repositories, or GitLab project repositories that are linked to your CodeCatalyst project are shown in the list. Choose the GitHub repository, Bitbucket repository, or GitLab project repository to view and edit files in the third-party repository provider.

> **Note**
>
> If a workflow uses a GitHub repository, Bitbucket repository, or GitLab project repository in a source action, changes you make to the workflow YAML in the visual editor or the YAML editor in CodeCatalyst will be automatically committed and pushed to the third-party repository.

Searching Jira issues in CodeCatalyst

After linking a Jira project, you can search the linked Jira project for issues using the CodeCatalyst global search bar. You can also search for Jira issues in CodeCatalyst while linking to the issues from a pull request. For more information about linking Jira issues to a CodeCatalyst pull request, see Linking Jira issues to CodeCatalyst pull requests.

To search for Jira issues in linked Jira projects

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst project.
3. In the global search bar, search a linked Jira project for issues or Jira issues you want to link to a pull request.

Automatically starting a workflow run after third-party repository events

You can use a linked GitHub repository, Bitbucket repository, or GitLab project repository as the source for a workflow, where changes to a specified branch in a linked GitHub repository, Bitbucket repository, or GitLab project repository automatically start a workflow run.

A *workflow* is an automated procedure that describes how to build, test, and deploy your code as part of a continuous integration and continuous delivery (CI/CD) system. A workflow defines a series of steps, or *actions*, to take during a workflow run. A workflow also defines the events, or *triggers*, that cause the workflow to start. To set up a workflow, you create a *workflow definition file* using the CodeCatalyst console's visual or YAML editor.

> **Tip**
>
> For a quick look at how you might use workflows in a project, create a project with a blueprint. Each blueprint deploys a functioning workflow that you can review, run, and experiment with.

When you configure a workflow to use a linked GitHub repository, Bitbucket repository, or GitLab project repository, the workflow configuration file is stored in that GitHub repository, Bitbucket repository, or GitLab project repository. The workflow configuration is a YAML file that defines the workflow name, triggers, resources, artifacts, and actions. For more information about the workflow configuration file, see Workflow YAML definition.

The workflow configuration file must be in the `./codecatalyst/workflows/` directory in your GitHub repository, Bitbucket repository, or GitLab project repository.

You can use the workflow editor to create and configure workflows. For more information see Getting started with workflows and Connecting a workflow to a source repository.

Adding triggers to start workflow runs

You can configure a CodeCatalyst workflow to automatically start a run when code is pushed to the specified branch of your GitHub or Bitbucket repository. To start a workflow run automatically, add a trigger to the `Triggers` section of the workflow configuration file.

Example: A simple code push trigger

The following example shows a trigger that starts a workflow run whenever code is pushed to any branch in your source repository.

```
Triggers:
  - Type: PUSH
```

Example: A simple pull request trigger

The following example shows a trigger that starts a workflow run whenever a pull request is created against any branch in your source repository.

```
Triggers:
  - Type: PULLREQUEST
    Events:
      - OPEN
```

For more information, see Starting a workflow run automatically with triggers.

Restricting IP access with third-party repository providers

You can restrict access to your GitHub repositories, Bitbucket repositories, or GitLab project repositories based on IP addresses by setting up rules or configurations. You can do this through the third-party provider's settings or access control features.

Depending on which third-party repository provider you're using, see one of the following:

- The Amazon CodeCatalyst **GitHub repositories** extension is compatible with GitHub Enterprise Cloud IP access restrictions. When configuring a GitHub Enterprise Cloud organization to restrict access to specific IP addresses, you can also enable GitHub apps to configure the allow list, which will let CodeCatalyst register its IP addresses automatically with GitHub. Alternatively, you can manually add the CodeCatalyst IP addresses.

- The Amazon CodeCatalyst **Bitbucket repositories** extension is compatible with Bitbucket Cloud Premium access restrictions. When configuring a Bitbucket Cloud Premium workspace to restrict access to specific IP addresses, you can also add IP addresses or network blocks for a set of IP addresses to an allowlist.
- The Amazon CodeCatalyst **GitLab repositories** extension is compatible with GitLab IP address restrictions. When configuring a GitLab Premium or Ultimate group to restrict access to specific IP addresses, you can also add IP addresses or network blocks for a set of IP addresses to an allowlist.

If the CodeCatalyst IP addresses aren't in a third-party repository's allowlist, the Amazon CodeCatalyst app won't be able to access your third-party repositories. For more information, see IP addresses used by third-party repositories extension.

IP addresses used by third-party repositories extension

The following IP addresses are used by the third-party extensions to access your third-party resources:

- **GitHub repositories**:

```
us-west-2
  52.32.242.246
  54.148.176.49
  35.164.118.94
eu-west-1
  34.241.64.10
  34.246.255.80
  3.248.38.7
```

- **Bitbucket repositories** and **GitLab repositories**:

```
us-west-2
  35.160.210.199
  54.71.206.108
  54.71.36.205
eu-west-1
  34.242.64.82
  52.18.37.201
  54.77.75.62
```

Blocking third-party merges when workflows fail

After linking a GitHub or Bitbucket repository to CodeCatalyst, you can add CodeCatalyst workflows for pull requests. Similarly, after linking a GitLab project repository to CodeCatalyst you can add CodeCatalyst workflows for merge requests. One or more workflow runs can occur on a specific commit, and the run status of each workflow in CodeCatalyst is also reflected as part of the commit status in GitHub, Bitbucket, or GitLab. When a new commit is pushed, new workflow run statuses are reflected in GitHub, Bitbucket, or GitLab for that new commit. If you run a workflow again for a commit, the new workflow run status overrides the previous status for that commit and workflow.

You can set branch protection rules in GitHub or Bitbucket to block a pull request merge, or in GitLab to block a merge request, when the latest commit has a failed workflow run status. With branch protection rules, the status of the latest commit affects the ability to merge a pull request in GitHub, Bitbucket, or GitLab. To learn more about workflows, see Running a workflow and Starting a workflow run automatically with triggers.

Depending on which third-party repository provider you're using, see the following:

- **GitHub repositories**: GitHub's documentation About status checks and About protected branches.
- **Bitbucket repositories**: Bitbucket's documentation for Using branch permissions and Take control with branch permissions in Bitbucket Cloud.
- **GitLab repositories**: GitLab's documentation for Auto merge and Protected branches.

Linking Jira issues to CodeCatalyst pull requests

You can link pull requests that are created in a CodeCatalyst source repository to Jira issues. After linking a Jira issue, the issue is displayed as a property of the pull request. As a result, pull request events, workflow events, and deployment events are sent to Jira and added to the Jira issue. Pull requests can be linked to one or more Jira issues. You can only link pull requests that are in a CodeCatalyst source repository, not those in a third-party repository like GitHub. Before you can link Jira issues to a pull request, your Jira project must be linked to the CodeCatalyst project. For more information about linking a Jira project to a CodeCatalyst project, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

> **Note**
>
> You can't create a pull request without a source repository with two branches in your CodeCatalyst project. For more information on pull requests, see Working with pull requests in CodeCatalyst.

To link Jira issues to a CodeCatalyst pull request

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst project.
3. In the navigation pane, choose **Code**, and then choose **Pull requests**.
4. Choose **Create pull request** to enter the pull request details.
5. From the **Source repository** drop-down menu, choose the source repository in which you want to link a pull request.
6. From the **Source branch** drop-down menu, choose the branch that contains the changes you want reviewed.
7. From the **Destination branch** drop-down menu, choose the branch where you want to merge reviewed changes.
8. In the **Pull request title** text input field, enter the title of your pull requests.
9. Choose **Link issues** for the **Jira issues - *optional*** field, choose the drop-down, and search the Jira issues you want to add from the linked Jira project.
10. Select the Jira issues you want to add to the pull request.
11. Choose **Create** to create the pull request.

Once you link Jira issues to a CodeCatalyst pull request, a summary of the pull request is available. The summary includes workflow runs, linked issues, required reviewers, optional reviewers, and the author.

> **Note**
>
> **Assignee** and **Created by** information associated with the Jira issue is not available in CodeCatalyst.

After linking a pull request, the synced CodeCatalyst project and Jira project allow updates from CodeCatalyst to be reflected in your Jira project. The status of the linked pull request and any workflow events related to the pull request will show up in the Jira issue when viewing it in Jira. For more information on viewing CodeCatalyst events in Jira, see Viewing CodeCatalyst events in Jira issues.

Viewing CodeCatalyst events in Jira issues

If your CodeCatalyst projects and Jira projects are linked, the summary status of the pull request and the status of associated CodeCatalyst workflow events are reflected in your Jira issue. For example, if you close or merge a pull request in CodeCatalyst, the status update is reflected in the Jira issue. CodeCatalyst workflow CI/CD events related to a CodeCatalyst pull request are synchronized, so a successful workflow run would be sent to the Jira issue as well.

To view CodeCatalyst events in a Jira issue

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst project.
3. In the CodeCatalyst navigation pane, choose **Code**, choose **Pull requests**, and then choose the pull request with the Jira issue that you want to view in your Jira project.
4. In the **Additional info** pane, choose the Jira issue that you want to view in your Jira project.
5. From the **Details** pane in the Jira project, choose **pull request** listed for **Development** to see details of the pull request.
6. (Optional) To see the latest builds, choose the **Builds** tab.
7. (Optional) To see the development status, choose the **Deployments** tab.

Search for code, issues, projects, and users in CodeCatalyst

Use the search bar or a dedicated search results window in CodeCatalyst to search through code, issues, projects, and users CodeCatalyst.

You can find resources across your space and projects by entering queries such as name, description, and status into the search bar. You can also refine your search queries using the search query language.

Topics

To search

1. In the search bar in the top navigation bar, enter a search query.
2. (Optional) Refine your search query using CodeCatalyst's search query language. For more information, see Refining your search query.
3. Do one of the following:
   - To search for resources within the project that you are currently in, choose **This project**.
   - To search for resources within all of the projects in the space you are currently in, choose **This space**.
4. View search results in a dedicated search results window by doing one of the following:
   - In the bottom of the quick search results window, choose **View all results in project-name | space-name** to view all search results.
   - Press **Enter** to view all search results.

> **Tip**
>
> Mention other project users in a pull request comment or description, or in an issue comment or description, by using the @ sign followed by their display name or user name.

You can also link to resources like issues or code files by using the @ sign followed by the name of the issue or code file.

Refining your search query

If you can't find what you're looking for after searching, you can refine your search with CodeCatalyst's specialized query language. Individual fields have no character limit, but the overall query has a limit of 1,024 characters.

Topics

Refining by type

To refine the scope of your search to a specific type of information, include *`type:result-type`* in your search, where *`result-type`* is `code`, `issue`, `project`, or `user` .

Examples:

- `type:code AND java` – Show code results in code-related fields that contain “java”.

  For more information, see Code fields.
- `type:issue AND Bug` – Show issue results in issue-related fields that contain “Bug”.

  For more information, see Issue fields.
- `type:user AND MaryMajor` – Show user results in user-related fields that contain “MaryMajor”.

  For more information, see User fields.
- `type:project AND Datafeeder` – Show project results that contain "Datafeeder".

  For more information, see Project fields.

Refining by field

To refine the scope of your search to a specific field, include *`field-name:query`* in your search, where *`field-name`* is `title`, `username`, `project`, `description`, and so on, and *`query`* is the text for which you are searching. For a list of fields, see Searchable fields reference. You can search for multiple queries using parentheses.

Examples:

- `title:bug` – Show results where the title contains “bug”.
- `username:John` – Show results where the user name contains “John”.
- `project:DataFeeder` – Show results in the project “DataFeeder”. Query isn't case sensitive.
- `description:overview` – Show results where the description contains “overview”.

Refining with Boolean operators

To specify constraints on search phrases, you can use the Boolean operators `AND`, `OR`, and `NOT`. If you list multiple phrases, CodeCatalyst joins them with `OR` by default. You can group search phrases using parentheses.

- `exception AND type:code` – Show only code results for “exception”.
- `path:README.md AND repo:ServerlessAPI` – Show results for paths with “README.md” where the repository is named “ServerlessAPI”.
- `buildspec.yml AND (repo:ServerlessAPI OR ServerlessWebApp)` – Show results for “buildspec.yml” where the repository is “ServerlessAPI” or “ServerlessWebApp”.
- `path:java NOT (path:py OR path:ts)` – Show results where the path contains “java” but not “py” or “ts”.

Refining by project

To refine the scope of your search to a specific project, include *`project:name AND query`* in your search, where *`name`* is the project within which you are searching and *`query`* is the content for which you are searching.

- `project:name AND query` – Show results where the path contains the query and the project name.

Considerations when working with search

Delayed content updates – It can take several minutes for content updates, such as name changes or issue reassignments, to be reflected in the search results. Large updates, such as a code base migration, can take longer to appear in search results.

Escaping special characters – The following special characters require special consideration in your search queries: `+ - & & || ! ( ) { } [ ] ^ " ~ * ? : \`. Special characters will not influence the query, and you must either remove them or escape them. To escape a character, add a backslash (\) in front of it. For example, the search query [Feature] should either be Feature or \[Feature\].

Narrowing search – Search isn't case sensitive. Searching in all lowercase prevents your queries from splitting up words on case change. For example, to query for `MyService` and only `MyService`, consider querying `myservice` to avoid results that contain only `my` or `service`.

Search joins words and parts of words with OR-wise conjunction by default. For example, `new function` could return results containing both `new` and `function` and also results with only `new` or `function`. To avoid the latter, combine multiple words with `AND`. For example, you can search `new AND function`.

Default branches – Search will only return code results from the latest commit on a source repository's default branch. To find code on other branches or commits, consider cloning the repository locally, opening the branch in a Dev Environment, or viewing the branches and details in the CodeCatalyst UI. Changing the default branch results in updates to the files discoverable by search. For more information, see Managing the default branch for a repository.

> **Important**
>
> CodeCatalyst doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.
>
> As a best practice, always make sure you have the latest version of the extension before you link a repository.

Searchable fields reference

CodeCatalyst searches the following fields when you enter search queries. Aliases are another name that you can use to reference the field in the advanced query language.

Code fields

Field	Alias	Description
branchName	branch	Name of branch the code file is on.
code	N/A	Information about the code contents in the form of code snippets indicating parts of the source code that matched the search.
commitId	N/A	Commit ID of the commit in which the returned code file was last updated. May or may not be the commit ID at the tip of the branch name specified in `branchName` .
commitMessage	N/A	Commit message of the commit in which the code file was last updated. May or may not be the commit message at the tip of the branch name specified in `branchName` . If no commit message was provided, this value will be an empty string.
filePath	path	File path of this code file.

Field	Alias	Description
lastUpdatedBy	N/A	CodeCatalyst user who last updated the code file. If the user name isn't available , this value will be the email address of the user as configured in the Git configuration file.
lastUpdatedById	N/A	System-generated unique ID of user that last updated the code file. If the user ID isn't available, this value might be the email address of the user.
lastUpdatedTime	N/A	Time when the search data was last updated with the commit that contained the code file (in coordinat ed universal time (UTC) timestamp).
projectId	N/A	System-generated unique ID of the project.
projectName	projectNames, project	Display name of the project that contains the source repository where the code file has been committed.
repositoryId	repoId	System-generated unique ID of the source repository.
repositoryName	repository, repo	Display name of the source repository where the code file has been committed.

Issue fields

Field	Alias	Description
assigneeIds	assigneeId	System-generated unique IDs of the users assigned to the issue.
assignees	assignee	User names of the users assigned to the issue.
createdBy	N/A	Display name of the user who created the issue.
createdById	N/A	System-generated unique ID of the user who created the issue.
createdTime	N/A	Time the issue was created (in coordinated universal time (UTC) timestamp).
description	N/A	Description of the issue.
isArchived	archived	Boolean value that indicates whether to create the issue in an archived state.
isBlocked	blocked	Boolean value that indicates whether the issue is marked as blocked.
labelIds	labelId	System-generated unique IDs of the labels for an issue.
lastUpdatedBy	N/A	Display name of use who last updated the issue.

Field	Alias	Description
lastUpdatedById	N/A	System-generated unique ID of the user who last updated the issue.
lastUpdatedTime	N/A	Time the issue was last updated (in coordinat ed universal time (UTC) timestamp).
priority	N/A	Priority of the issue, if one has been assigned.
projectId	N/A	System-generated unique ID of the project.
projectName	projectNames, project	Project in which this issue can be found.
shortId	N/A	Shortened, auto-incr ementing identifier for the issue.
status	N/A	Status of the issue that indicates if issue is in backlog or column on board.
statusId	N/A	System identifier of the status.
title	N/A	Title of the issue.

Project fields

Field	Alias	Description
description	N/A	Description of the project.

Field	Alias	Description
lastUpdatedTime	N/A	Time when the project metadata was last updated (in coordinated universal time (UTC) timestamp).
projectName	project	Name of the project in the space.
projectPath	N/A	URL-routable name of the project, defined during project creation. Used in URLs that require the project name.

User fields

Field	Alias	Description
displayName	N/A	Name used for the user in CodeCatalyst. Display names are not unique.
email	N/A	Email address of the user.
lastUpdatedTime	N/A	Time when the user metadata was last updated (in coordinated universal time (UTC) timestamp).
userName	username	User name chosen by the user when they signed up for CodeCatalyst. Unlike display names, user names can't be changed.

Troubleshooting Amazon CodeCatalyst

The following information can help you troubleshoot common issues in CodeCatalyst. You can also use the Amazon CodeCatalyst health report to determine if there are service issues that might be impacting your experience.

Topics

Troubleshooting general access issues

I forgot my password

Problem: I forgot the password I use for my AWS Builder ID and Amazon CodeCatalyst.

Possible fixes: The easiest way to fix this problem is to reset your password.

1. Open Amazon CodeCatalyst and enter your **Email address**. Then, choose **Continue**.
2. Choose **Forgot password?**

3. We'll send you an email with a link for you to change your password. If you don't see the email in your inbox, check your spam folder.

Some or all of Amazon CodeCatalyst isn't available

Problem: I navigated to or followed a link to the CodeCatalyst console, but I see an error.

Possible fixes: The most common reasons for this problem are that you either followed a link to a project or a space you haven't been invited to, or there is a general availability issue with the service. Check the Health report to see if there are any known issues with the service. If not, contact the person who invited you to the project or space and ask for another invitation. If you haven't been invited to any projects or spaces, you can sign up and create your own space and projects.

I can't create a project in CodeCatalyst

Problem: I want to create a project, but the **Create project** button shows as unavailable, or I receive an error message.

Possible fixes: The most common reasons for this problem are that you are signed in to the console with an AWS Builder ID that doesn't have the **Space administrator** role. You must have this role to create projects in a space.

If you do have this role and the button does not appear as available, there might be a transitory issue with the service. Refresh your browser and try again.

Troubleshooting support issues

I get an error when I access AWS Support for Amazon CodeCatalyst

Problem: When I choose the AWS Support for Amazon CodeCatalyst option, I receive the following error message:

```
Unable to assume role

To access support cases, you must add the role
AWSRoleForCodeCatalystSupport to the AWS account that is the billing
account for the space.
```

Possible fixes: Add the required role to the AWS account that is the billing account for the space. The account designated as the billing account for the space uses the `AWSRoleForCodeCatalystSupport` role and `AmazonCodeCatalystSupportAccess` managed policy. For more information, see Creating the **AWSRoleForCodeCatalystSupport** role for your account and space.

> **Note**
>
> An AWS Builder ID can only get support for the alias they are authenticated with and only for resources based on permissions in CodeCatalyst. Account and Billing support is available for all users in the space. However, builders can only get support for resources and information they have permissions for in CodeCatalyst.

I cannot create technical support cases for my space

Problem: I cannot create technical support cases for my space.

Fixes: A Business Support or Enterprise Support plan needs to be added to the space billing account in order for users in the space to create technical support cases. Ask your space administrator to add an AWS Support plan to your space billing account or visit https://repost.aws/ to ask the AWS community.

My account for support cases is no longer connected to my space in CodeCatalyst

Problem: My account for support cases is no longer connected to my space in CodeCatalyst.

Fixes: If a user with the **Space administrator** role switches the space billing account, this will disconnect the AWS Support plan and all associated cases from the space. The AWS Support cases associated with the old space billing account will no longer be visible in AWS Support for Amazon CodeCatalyst. The root user for that billing account can view and resolve old cases from the AWS Management Console and can set up IAM permissions for AWS Support for other users to view and resolve old cases. You will not be able to continue to get technical support for CodeCatalyst from the old space billing account through the AWS Management Console, but you can receive technical support for other services until your AWS Support plan is canceled.

For more information, see Updating, resolving, and reopening your case in the *AWS Support User Guide*.

I can't open a support case for another AWS service inAWS Support for Amazon CodeCatalyst

Problem: I can’t open a support case for another AWS service in AWS Support for CodeCatalyst.

Possible fixes: You can only open CodeCatalyst support cases from AWS Support for CodeCatalyst. If you need support for services or resources deployed from CodeCatalyst to another AWS, Amazon, or other third-party service, you will need to create a case through the AWS Management Console or the third-party service support channel. For more information, see Creating support cases and case management in the *AWS Support User Guide*.

Some or all of Amazon CodeCatalyst isn't available

Problem: I navigated to or followed a link to the CodeCatalyst console, but I see an error.

Possible fixes: The most common reasons for this problem are that you either followed a link to a project or a space you haven't been invited to, or there is a general availability issue with the service. Check the Health report to see if there are any known issues with the service. If not, contact the person who invited you to the project or space and ask for another invitation. If you haven't been invited to any projects or spaces, you can sign up and create your own space and projects.

I can't create a project in CodeCatalyst

Problem: I want to create a project, but the **Create project** button shows as unavailable, or I receive an error message.

Possible fixes: The most common reasons for this problem are that you are signed in to the console with an AWS Builder ID that doesn't have the **Space administrator** role. You must have this role to create projects in a space.

If you do have this role and the button does not appear as available, there might be a transitory issue with the service. Refresh your browser and try again.

I want to submit feedback in CodeCatalyst

Problem: I found a bug in CodeCatalyst and I want to submit feedback.

Possible fixes: You can submit feedback directly in CodeCatalyst.

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. In the navigation pane, choose **Give feedback**.
3. Choose the type of feedback from the drop-down menu and enter your feedback.

Troubleshooting problems with source repositories

The following information can help you troubleshoot common issues with source repositories in CodeCatalyst.

Topics

I have reached the maximum storage for my space and see warnings or errors

Problem: I want to commit code to one or more source repositories in CodeCatalyst, but I see an error. In the console, I see a message on the source repository page that I have reached the storage limit for the space.

Possible fixes: Depending on your role in the project or space, you can either reduce the size of one or more of your source repositories, delete unused source repositories, or change your billing tier to one that has more storage.

- To reduce the size of a source repository in a project, you can delete unused branches. For more information, see Deleting a branch and Contributor role.
- To reduce the overall storage for a space, you can delete unused source repositories. For more information, see Deleting a source repository and Project administrator role.
- To increase the amount of storage available for your space, you can change your billing tier to one with more storage. For more information, see Changing your CodeCatalyst billing tier in the Amazon CodeCatalyst Administrator Guide.

I receive an error when trying to clone or push to an Amazon CodeCatalyst source repository

Problem: When I try to clone a source repository to a local computer or into an integrated development environment (IDE), I receive a permissions error.

Possible fixes: You might not have a personal access token (PAT) for your AWS Builder ID, you might not have configured your credential management system with your PAT, or your PAT might have expired. Try one or more of the following solutions:

- Create a personal access token (PAT). For more information, see Grant users repository access with personal access tokens.
- Make sure you have accepted an invitation to the project that contains the source repository and that you are still a member of that project. You cannot clone a source repository if you aren't an active member of that project. Sign in to the console and attempt to navigate to the space and the project where you're trying to clone a source repository. If you cannot see the project in the list of projects for the space, you either aren't a member of that project, or you haven't accepted an invitation to that project. For more information, see Accepting an invitation and creating your AWS Builder ID.
- Make sure your clone command is formatted correctly and includes your AWS Builder ID. For example:

```
https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

- Use the AWS CLI to make sure that you have a PAT associated with your AWS Builder ID, and that it is not expired. If you don't have one or the PAT is expired, create one. For more information, see Grant users repository access with personal access tokens.

- Try creating a Dev Environment to work with the code in the source repository instead of cloning it to a local repo or IDE. For more information, see Creating a Dev Environment.

I receive an error when trying to commit or push to an Amazon CodeCatalyst source repository

Problem: When I try to push to a source repository, I receive a permissions error.

Possible fixes: You might not have a role in the project that allows you to commit and push code changes to the project. View your role in the project where you are trying to push changes to a source repository. For more information, see Getting a list of members and their project roles and Granting access with user roles.

If you have a role that allows committing and pushing changes, the branch where you are trying to commit changes might have a branch rule configured for it that prevents you from pushing code changes to that branch. Try creating a branch and pushing your code to that branch instead. For more information, see Manage allowed actions for a branch with branch rules.

I need a source repository for my project

Problem: My project either doesn't have a source repository, or I need another source repository for my project.

Possible fixes: Some projects are created without any resources. If you are a member of the project, you can create source repositories for that project in CodeCatalyst. If someone with the **Space administrator** role installs the **GitHub Repositories** and connects it to a GitHub account, you can link to available GitHub repositories to add them to your project if you have the **Project administrator** role. For more information, see Creating a source repository and Linking a source repository.

My source repository is brand-new but contains a commit

Problem: I just created a source repository. It should be empty, but it has a commit, a branch, and a `README.md` file in it.

Possible fixes: This is expected behavior. All source repositories in CodeCatalyst include an initial commit that sets the default branch to `main` and includes either sample code (if the repository was created for a project using a blueprint that included sample code) or a template markdown file for

a repository README file. You can create additional branches in the console and in Git clients. You can create and edit files in the console, and delete files in Dev Environments and Git clients.

I want a different branch as my default branch

Problem: My source repository came with a default branch named `main`, but I want a different branch as my default branch.

Possible fixes: You cannot change or delete the default branch in source repositories in CodeCatalyst. You can create additional branches and use those branches in source actions in workflows. You can also choose to link GitHub repositories and use them as repositories for your project.

I am receiving emails about activity in pull requests

Problem: I didn't sign up or configure email notifications about pull request activity, but I'm receiving them anyway.

Possible fixes: Email notifications are sent automatically about pull request activity. For more information, see Reviewing code with pull requests in Amazon CodeCatalyst.

I forgot my personal access token (PAT)

Problem: I've been using a PAT for cloning, pushing, and pulling code for source repositories, but I've lost the value for my token, and I can't find it in the CodeCatalyst console.

Possible fixes: The quickest way to solve this problem is to create another PAT and configure your credential manager or IDE to use this new PAT. We only display the value of a PAT when you create it. If you lose this value, it cannot be retrieved. For more information, see Grant users repository access with personal access tokens.

A pull request doesn't display the changes I expect

Problem: I created a pull request, but I don't see the changes I expect to see between the source and destination branches.

Possible fixes: This might be caused by a number of issues. Try one or more of the following solutions:

- You might be reviewing the changes between older revisions, or you might not be viewing the latest changes. Refresh your browser and make sure that you've chosen the comparison between revisions you want to view.
- Not all changes in a pull request can be displayed in the console. For example, you cannot view Git submodules in the console, so you cannot view differences in a submodule in a pull request. Some differences might be too large to display. For more information, see Quotas for source repositories in CodeCatalyst and Viewing a file.
- Pull requests display the differences between the merge base and whatever revision you choose. When you create a pull request, the difference displayed for you is the difference between the tip of the source branch and the tip of the destination branch. Once the pull request has been created, the displayed difference is between the revision and its merge base. The merge base is the commit that was the tip of the destination branch when the revision was created. The merge base can change between revisions. For more information about differences and merge bases in Git, see git-merge-base in the Git documentation.

A pull request shows a status of Not mergeable

Problem: I want to merge a pull request, but its status shows as **Not mergeable**.

Possible fixes: This can be caused by one or more problems:

- All required reviewers for your pull request must approve a pull request before it can be merged. Review the list of required reviewers for any reviewers with a clock icon next to the name. A clock icon indicates that the reviewer hasn't approved the pull request.

  > **Note**
  >
  > If a required reviewer has been removed from your project before approving the pull request, you cannot merge the pull request. Close the pull request and create a new pull request.

- There might be a merge conflict between the source branch and the destination branch. CodeCatalyst does not support all possible Git merge strategies and options. You can evaluate the branches for merge conflicts in a Dev Environment or clone the repository and use an IDE or Git tools to find and resolve merge conflicts. For more information, see Merging a pull request.

Troubleshooting projects and blueprints

This section can help you troubleshoot some common issues you might encounter while working with projects and blueprints in Amazon CodeCatalyst.

Java API with AWS Fargate blueprint missing dependencies for apache-maven-3.8.6

Issue: For a project created from the Java API with AWS Fargate blueprint, the workflow fails with an error for missing `apache-maven-3.8.6` dependencies. The workflow fails with output similar to the following example:

```
Step 8/25 : RUN wget https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/apache-
maven-3.8.6-bin.tar.gz -P /tmp
---> Running in 1851ce6f4d1b
[91m--2023-03-10 01:24:55--  https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/
apache-maven-3.8.6-bin.tar.gz
[0m[91mResolving dlcdn.apache.org (dlcdn.apache.org)...
[0m[91m151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443...
[0m[91mconnected.
[0m[91mHTTP request sent, awaiting response... [0m[91m404 Not Found
2023-03-10 01:24:55 ERROR 404: Not Found.
[0mThe command '/bin/sh -c wget https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/
apache-maven-3.8.6-bin.tar.gz -P /tmp' returned a non-zero code: 8
[Container] 2023/03/10 01:24:55 Command failed with exit status 8
```

Solution: Update the blueprint Dockerfile using the following steps.

1. In the search bar, enter `apache-maven-3.8.6` to locate the dockerfile inside the project created with the Java API with AWS Fargate blueprint.
2. Update the Dockerfile (`/static-assets/app/Dockerfile`) to use `maven:3.9.0-amazoncorretto-11` as a base image and remove the dependency on the `apache-maven-3.8.6` package.
3. (Recommended) We also recommend updating the Maven heap size to 6 GB.

Below is an example Dockerfile.

```
FROM maven:3.9.0-amazoncorretto-11 AS builder

COPY ./pom.xml ./pom.xml
COPY src ./src/

ENV MAVEN_OPTS='-Xmx6g'

RUN mvn -Dmaven.test.skip=true clean package

FROM amazoncorretto:11-alpine

COPY –from=builder target/CustomerService-0.0.1.jar CustomerService-0.0.1.jar
EXPOSE 80
CMD ["java","-jar","-Dspring.profiles.active=prod","/CustomerService-0.0.1.jar", "–
server.port=80"]
```

Modern three-tier web application blueprint workflow OnPullRequest fails with permissions error for Amazon CodeGuru

Issue: When I try to run a workflow for my project, the workflow fails to run with the following message:

```
Failed at codeguru_codereview: The action failed during runtime. View the action's logs
 for more details.
```

Solution: One possible cause of this action failure might be due to missing permissions in the IAM role policy, where your version of the service role used by CodeCatalyst in the connected AWS account is missing required permissions for the **codeguru_codereview** action to run successfully. To fix this problem, either the service role must be updated with the required permissions, or you must change the service role used for the workflow to one that has the required permissions for Amazon CodeGuru and Amazon CodeGuru Reviewer. Using the following steps, find your role and update the role policy permissions in order to allow the workflow to run successfully.

> **Note**
>
> These steps apply for the following workflows in CodeCatalyst:
>
> - The **OnPullRequest** workflow provided for projects created with the Modern three-tier web application blueprint in CodeCatalyst.

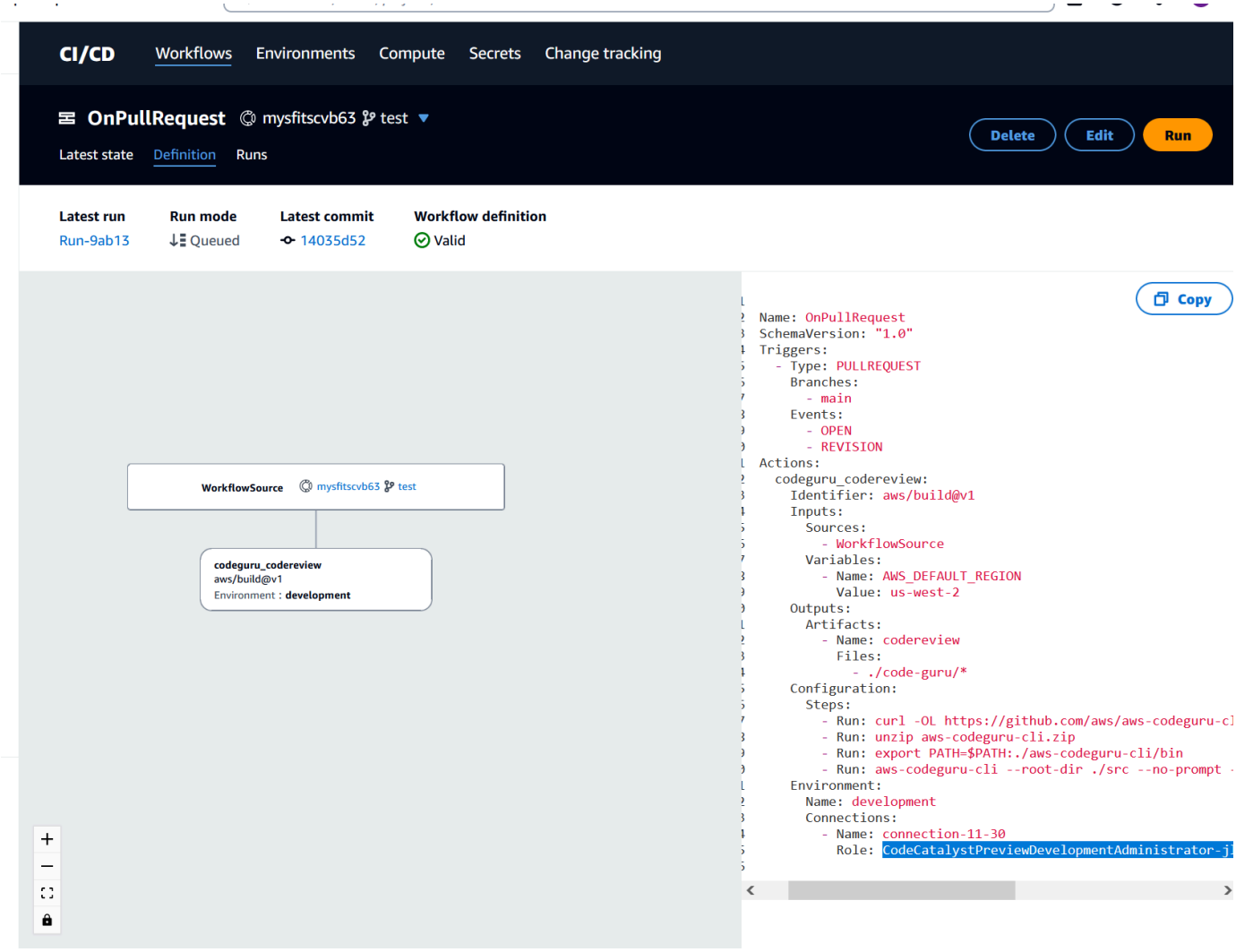
- Workflows added to projects in CodeCatalyst with actions that access Amazon CodeGuru or Amazon CodeGuru Reviewer.

Each project contains workflows with actions that use a role and environment provided by the AWS account connected to your project in CodeCatalyst. The workflow with the actions and their designated policy is stored in your source repository in the directory /.codecatalyst/workflows. Modifying the workflow YAML is not required unless you are adding a new role ID to the existing workflow. For information about YAML template elements and formatting, see Workflow YAML definition.

These are the high-level steps to follow to edit your role policy and verify the workflow YAML.

To reference your role name in the workflow YAML and update the policy

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space. Navigate to your project.
3. Choose **CI/CD**, and then choose **Workflows**.
4. Choose the workflow titled **OnPullRequest**. Choose the **Definition** tab.
5. In the workflow YAML, in the `Role:` field under the **codeguru_codereview** action, make a note of the role name. This is the role with the policy that you will modify in IAM. The following example shows the role name.



6. Do one of the following:

- (Recommended) Update the service role connected to your project with the required permissions for Amazon CodeGuru and Amazon CodeGuru Reviewer. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* with a unique identifier appended. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role. Proceed to the next steps to update the policy in IAM.

> **Note**
>
> You must have AWS administrator access to the AWS account with the role and policy.

- Change the service role used for the workflow to one that has the required permissions for Amazon CodeGuru and Amazon CodeGuru Reviewer or create a new role with the required permissions.

7. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

   In the IAM console, find the role from step 5, such as `CodeCatalystPreviewDevelopmentRole`.

8. In the role from step 5, change the permission policy to include the `codeguru-reviewer:*` and `codeguru:*` permissions. After adding these permissions, the permission policy should look similar to the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudformation:*",
                "lambda:*",
                "apigateway:*",
                "ecr:*",
                "ecs:*",
                "ssm:*",
                "codedeploy:*",
                "s3:*",
                "iam:DeleteRole",
                "iam:UpdateRole",
                "iam:Get*",
                "iam:TagRole",
                "iam:PassRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "iam:PutRolePolicy",
                "iam:CreatePolicy",
                "iam:DeletePolicy",
                "iam:CreatePolicyVersion",
                "iam:DeletePolicyVersion",
                "iam:PutRolePermissionsBoundary",
                "iam:DeleteRolePermissionsBoundary",
                "sts:AssumeRole",
```

```
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:ModifyRule",
                "cloudwatch:DescribeAlarms",
                "sns:Publish",
                "sns:ListTopics",
                "codeguru-reviewer:*",
                "codeguru:*"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

9. After you make the policy corrections, return to CodeCatalyst and start the workflow run again.

Still looking to solve your problem?

You can go to Amazon CodeCatalyst or fill out a Support Feedback form. In the **Request information** section, under **How can we help you**, include that you are an Amazon CodeCatalyst customer. Provide as much detail as possible so that we can most efficiently address your issue.

Troubleshooting problems with Dev Environments

Consult the following sections to troubleshoot problems related to Dev Environments. For more information about Dev Environments, see Write and modify code with Dev Environments in CodeCatalyst.

Topics

- I'm unable to connect to my Dev Environment via SSH
- I'm unable to connect to my Dev Environment via SSH because my local SSH config is missing
- I'm unable to connect to my Dev Environment via SSH because I'm having problems with my AWS Config for the codecatalyst profile
- Troubleshooting problems with IDEs
- Troubleshooting problems with devfiles

My Dev Environment creation didn't succeed due to a problem with quotas

Problem: I want to create a Dev Environment in CodeCatalyst, but I see an error. In the console, I see a message on the Dev Environments page that I have reached the storage limit for the space.

Possible fixes: Depending on your role in the project or space, you can either delete one or more of your own Dev Environments, or if you have the Space administrator role, you can delete unused Dev Environments created by other users. You can also decide to change the billing tier to one that includes more storage.

- To view the storage limits, view the **Billing** tab of the Amazon CodeCatalyst space to see if the **Usage** quotas have reached the maximum allowed. If the quotas have reached the maximum, contact someone with the Space administrator role to remove unneeded Dev Environments or to consider changing the billing tier.
- To remove any Dev Environments you created that you no longer need, see Deleting a Dev Environment.

If the issue continues and you get an error in your IDE, check that you have a CodeCatalyst role that allows you to create a Dev Environment. The **Space administrator** role, **Project administrator** role, and **Contributor** role all have permission to create Dev Environments. For more information, see Granting access with user roles.

I can't push changes from my Dev Environment to a specific branch in a repository

Problem: I want to commit and push code changes in my Dev Environment to a branch in a source repository, but I see an error.

Possible fixes: Depending on your role in the project or space, you might not have permissions to push code to source repositories in the project. The **Space administrator** role, **Project administrator** role, and **Contributor** role all have permission to push code to repositories in the project.

If you have the **Contributor** role but cannot push code to a specific branch, there might be a branch rule configured for the specific branch that prevents users with that role from pushing code to that particular branch. Try pushing your changes to a different branch, or create a branch and then push your code to that branch. For more information, see Manage allowed actions for a branch with branch rules.

My Dev Environment didn't resume

Problem: My Dev Environment didn't resume after I stopped it.

Possible fixes: To fix the problem, view the **Billing** tab of the Amazon CodeCatalyst space to see if the **Usage** quotas have reached the maximum limits. If the quotas have reached the maximum limit, contact your Space administrator to raise the billing tier.

My Dev Environment disconnected

Problem: My Dev Environment disconnected while I was using it.

Possible fixes: To fix the problem, check your internet connection. If you are not connected to the internet, connect and resume working in your Dev Environment.

My VPC-connected Dev Environment failed

Problem: I associated a VPC connection to my Dev Environment and it's running into errors.

Possible fixes: Docker uses a link layer device called a bridge network that enables containers that are connected to the same bridge network to communicate. The default bridge typically uses the `172.17.0.0/16` subnet for container networking. If the VPC subnet for your environment's instance uses the same address range that's already used by Docker, an IP address conflict might occur. To resolve an IP address conflict that's caused by Amazon VPC and Docker using the same IPv4 CIDR address block, configure a CIDR block that's different from `172.17.0.0/16`.

> **Note**
>
> You can't change the IP address range of an existing VPC or subnet.

I can't find which directory my project is in

Problem: I can't find which directory my project is in.

Possible fixes: To locate your project, change directory to `/projects`. This is the directory where you can find your projects.

I'm unable to connect to my Dev Environment via SSH

To troubleshoot your connection to your Dev Environment via SSH, you can execute the `ssh` command with `-vvv` option to show more information on how to resolve your issue:

```
ssh -vvv codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

I'm unable to connect to my Dev Environment via SSH because my local SSH config is missing

If your local SSH config (`~/.ssh/config`) is missing or the contents of `Host codecatalyst-dev-env*` section is out of date, you won't be able to connect to your Dev Environment via SSH. To troubleshoot this, delete the `Host codecatalyst-dev-env*` section and execute the first command from the **SSH Access** modal again. For more information, see Connecting to a Dev Environment using SSH.

I'm unable to connect to my Dev Environment via SSH because I'm having problems with my AWS Config for the `codecatalyst` profile

Make sure your AWS Config (`~/.aws/config`) for the `codecatalyst` profile matches the one described in Setting up to use the AWS CLI with CodeCatalyst. If not, delete the profile for `codecatalyst` and execute the first command from the **SSH Access** modal again. For more information, see Connecting to a Dev Environment using SSH.

Troubleshooting problems with IDEs

Consult the following sections to troubleshoot problems related to IDEs in CodeCatalyst. For more information on IDEs, see Creating a Dev Environment in an IDE.

Topics

I have mismatched runtime image versions in AWS Cloud9

AWS Cloud9 is using different versions of the frontend asset and the backend runtime image. Using different versions might cause the Git extension and AWS Toolkit to work incorrectly. To fix the problem, navigate to the Dev Environment dashboard, stop your Dev Environment, and then start it again. To fix the problem using APIs, use the `UpdateDevEnvironment` API to update the runtime. For more information, see UpdateDevEnvironment in the *Amazon CodeCatalyst API reference*.

I can't access my files in `/projects/projects` in AWS Cloud9

The AWS Cloud9 editor is unable to access files in the directory `/projects/projects`. To fix the problem, use the AWS Cloud9 terminal to access your files or move them to a different directory.

I can't launch my Dev Environment in AWS Cloud9 using a custom devfile

Your devfile image might not be compatible with AWS Cloud9. To fix the problem, review the devfile from your repository and corresponding Dev Environment and create a new one to continue.

I'm having issues in AWS Cloud9

For other issues, check the troubleshooting section in the AWS Cloud9 User Guide.

In JetBrains, I can't connect to my Dev Environments through CodeCatalyst

To fix the problem, check that you have only latest version of JetBrains installed. If you have multiple versions, uninstall the older versions and register your protocol handler again by closing the IDE and the browser. Then open JetBrains and register the protocol handler again.

I can't install AWS Toolkit for my IDE

To fix this problem for VS Code, manually install AWS Toolkit for Visual Studio Code from GitHub.

To fix this problem for JetBrains, manually install AWS Toolkit for JetBrains from GitHub.

In my IDE, I can't launch my Dev Environments

To fix this problem for VS Code, check that you have latest version of VS Code and AWS Toolkit for Visual Studio Code installed. If you don't have the latest version, update and launch your Dev Environment. For more information, see Amazon CodeCatalyst for VS Code.

To fix this problem for JetBrains, check that you have latest version of JetBrains and AWS Toolkit for JetBrains installed. If you don't have the latest version, update and launch your Dev Environment. For more information, see Amazon CodeCatalyst for JetBrains.

Troubleshooting problems with devfiles

Consult the following sections to troubleshoot problems related to devfiles in CodeCatalyst. For more information on devfiles, see Configuring a devfile for a Dev Environment.

Topics

My Dev Environment is using the default universal devfile even though I have implemented a custom image in a custom devfile

If CodeCatalyst encounter errors while starting a Dev Environment that is using a custom devfile, the Dev Environment defaults to the default universal devfile. To fix the problem, you can check the exact error in the logs under `/aws/mde/logs/devfile.log`. You can also check if `postStart` execution was successful in your logs: `/aws/mde/logs/devfileCommand.log`.

My project is not building in my Dev Environment with the default universal devfile

To fix the problem check that you are not using a custom devfile. If you are not using a custom devfile, view the `devfile.yaml` file in the source repository of the project to locate and fix any errors.

I want to move a repository devfile for a Dev Environment

You can move the default devfile in `/projects/devfile.yaml` to your source code repository. To update the location of the devfile, use following command: `/aws/mde/mde start --location` *`repository-name`*`/devfile.yaml`.

I'm having a problem starting my devfile

If there's a problem starting your devfile, it will enter recovery mode so that you can still connect to your environment and fix your devfile. While in recovery mode, running `/aws/mde/mde status` won't contain the location of your devfile.

```
{
    "status": "STABLE"
}
```

You can check the error in the logs under `/aws/mde/logs`, fix the devfile, and try running `/aws/mde/mde start` again.

I'm not sure how to check my devfile status

You can check your devfile status by running `/aws/mde/mde status`. After running this command, you may see one of the following:

- `{"status": "STABLE", "location": "devfile.yaml" }`

  This indicates that your devfile is correct.
- `{"status": "STABLE" }`

  This indicates that your devfile is could not start and has entered recovery mode.

You can check the exact error in the logs under `/aws/mde/logs/devfile.log`.

You can also check if `postStart` execution was successful in your logs: `/aws/mde/logs/devfileCommand.log`.

For more information, see Specifying universal devfile images for a Dev Environment.

My devfile is not compatible with the tooling provided in the latest image

In your Dev Environment, `devfile` or `devfile postStart` may fail if the `latest` tooling does not have the tooling required for a specific project. To fix the problem, do the following:

1. Navigate to your devfile.
2. In your devfile, update to a granular image version instead of `latest`. It may look similar to the following:

```
components:
  - container:
      image: public.ecr.aws/amazonlinux/universal-image:1.0
```

3. Create a new Dev Environment using the updated devfile.

Troubleshooting problems with workflows

Consult the following sections to troubleshoot problems related to workflows in Amazon CodeCatalyst. For more information about workflows, see Build, test, and deploy with workflows in CodeCatalyst.

Topics

How do I fix "Workflow is inactive" messages?

Problem: In the CodeCatalyst console, under **CI/CD**, **Workflows**, your workflow appears with the following message:

`Workflow is inactive.`

This message indicates that the workflow definition file contains a trigger that doesn't apply to the branch that you're currently on. For example, your workflow definition file might contain a PUSH trigger that references your `main` branch, but you're on a feature branch. Since the changes you're making in your feature branch don't apply to `main`, and will not start workflow runs in `main`, CodeCatalyst decommissions the workflow on the branch and marks it as `Inactive`.

Possible fixes:

If you want to start a workflow on your feature branch, you can do the following:

- In your feature branch, in the workflow definition file, remove the `Branches` property from the `Triggers` section so that it looks like this:

  ```
  Triggers:
    - Type: PUSH
  ```

  This configuration causes the trigger to activate on a push to any branch, including your feature branch. If the trigger is activated, CodeCatalyst will start a workflow run using the workflow definition file and source files in whatever branch you're pushing to.
- In your feature branch, in the workflow definition file, remove the `Triggers` section and run the workflow manually.

- In your feature branch, in the workflow definition file, change the PUSH section so that it references your feature branch rather than another branch (like `main`, for example).

> **Important**
>
> Be careful not to commit these changes if you don't intend to merge them to back to your `main` branch.

For more information about editing the workflow definition file, see Creating a workflow.

For more information about triggers, see Starting a workflow run automatically with triggers.

How do I fix "Workflow definition has *n* errors" errors?

Problem: You see any of the following error messages:

Error 1:

In the **CI/CD**, **Workflows** page, under your workflow's name, you see:

`Workflow definition has n errors`

Error 2:

While editing a workflow, you choose the **Validate** button and the following message appears at the top of the CodeCatalyst console:

`The workflow definition has errors. Fix the errors and choose Validate to verify your changes.`

Error 3:

After navigating to your workflow's details page, you see the following error in the **Workflow definition** field:

`n errors`

Possible fixes:

- Choose **CI/CD**, choose **Workflows**, and choose the name of the workflow that has the error. In the **Workflow definition** field near the top, choose the link to the error. Details about the error appear at the bottom of the page. Follow the troubleshooting tips in the error to fix the issue.
- Make sure that the workflow definition file is a YAML file.
- Make sure that the YAML properties in the workflow definition file are nested at the right level. To see how properties should be nested in the workflow definition file, refer to the Workflow YAML definition, or consult your action's documentation, which is linked to from Adding an action to a CodeCatalyst workflow.
- Make sure that asterisks (*) and other special characters are escaped properly. To escape them, add single or double quotes. For example:

```
Outputs:
  Artifacts:
    - Name: myartifact
      Files:
        - "**/*"
```

  For more information about special characters in the workflow definition file, see Syntax guidelines and conventions.
- Make sure that the YAML properties in the workflow definition file use the right capitalization. For more information about casing rules, see Syntax guidelines and conventions. To determine the correct casing of each property, refer to the Workflow YAML definition, or consult your action's documentation, which is linked to from Adding an action to a CodeCatalyst workflow.
- Make sure that the `SchemaVersion` property is present and set to the correct version in the workflow definition file. For more information, see SchemaVersion.
- Make sure that the `Triggers` section in the workflow definition file includes all required properties. To determine the required properties, choose the trigger in the visual editor and look for fields that are missing information, or consult the trigger reference documentation at Triggers.
- Make sure that the `DependsOn` property in the workflow definition file is properly configured and does not introduce circular dependencies. For more information, see Configuring actions to depend on other actions.
- Make sure that the `Actions` section in the workflow definition file includes at least one action. For more information, see Actions.

- Make sure that each action includes all required properties. To determine the required properties, choose the action in the visual editor and look for fields that are missing information, or consult your action's documentation, which is linked to from Adding an action to a CodeCatalyst workflow.
- Make sure that all input artifacts have corresponding output artifacts. For more information, see Defining an output artifact.
- Make sure that variables defined in one action are exported so that they can be used in other actions. For more information, see Exporting a variable so that other actions can use it.

How do I fix "Unable to locate credentials" and "ExpiredToken" errors?

Problem: While working through Tutorial: Deploy an application to Amazon EKS, you see one or both of the following error messages in your development machine's terminal window:

```
Unable to locate credentials. You can configure credentials by running "aws configure".
```

```
ExpiredToken: The security token included in the request is expired
```

Possible fixes:

These errors indicate that the credentials that you're using to access AWS services have expired. In this case, do not run the `aws configure` command. Instead, use the following instructions to refresh your AWS access key and session token.

To refresh your AWS access key and session token

1. Make sure you have the AWS access portal URL, username, and password for the user that you're using the complete the Amazon EKS tutorial (`codecatalyst-eks-user`). You should have configured these items when you completed Step 1: Set up your development machine of the tutorial.

   > **Note**
   >
   > If you do not have this information, go to the `codecatalyst-eks-user` details page in IAM Identity Center, choose **Reset password**, **Generate a one-time password [...]**, and **Reset password** again to display the information on the screen.

2. Do one of the following:

   - Paste the AWS access portal URL into your browser's address bar.

     Or

   - Refresh the AWS access portal page if it's already loaded.

3. Sign in with the `codecatalyst-eks-user`'s username and password, if you're not already signed in.
4. Choose **AWS account**, and then choose the name of the AWS account to which you assigned the `codecatalyst-eks-user` user and permission set.
5. Next to the permission set name (`codecatalyst-eks-permission-set`), choose **Command line or programmatic access**.
6. Copy the commands in the middle of the page. They look similar to the following:

   ```
   export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
   export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
   export AWS_SESSION_TOKEN="session-token"
   ```

   ...where *`session-token`* is a long random string.

7. Paste the commands into your terminal prompt on your development machine and press Enter.

   The new keys and session token are loaded.

   You have now refreshed your credentials. The AWS CLI, `eksctl`, and `kubectl` commands should now work.

How do I fix "Unable to connect to the server" errors?

Problem: While working through the tutorial described in Tutorial: Deploy an application to Amazon EKS, you see an error message similar to the following in your development machine's terminal window:

`Unable to connect to the server: dial tcp: lookup long-string.gr7.us-west-2.eks.amazonaws.com on 1.2.3.4:5: no such host`

Possible fixes:

This error usually indicates that the credentials that the `kubectl` utility is using to connect to your Amazon EKS cluster have expired. To solve the issue, refresh the credentials by entering the following command at the terminal prompt:

```
aws eks update-kubeconfig --name codecatalyst-eks-cluster --region us-west-2
```

Where:

- *`codecatalyst-eks-cluster`* is replaced with the name of your Amazon EKS cluster.
- *`us-west-2`* is replaced with the AWS Region where your cluster is deployed.

Why are CodeDeploy fields missing from the visual editor?

Problem: You are using a Deploy to Amazon ECS action, and you are not seeing the CodeDeploy fields such as **CodeDeploy AppSpec** in the workflow's visual editor. This problem may occur because the Amazon ECS service that you specified in the **Service** field is not configured to perform blue/green deployments.

Possible fixes:

- Choose a different Amazon ECS service on the **Deploy to Amazon ECS** action's **Configuration** tab. For more information, see Deploying an application to Amazon Elastic Container Service (ECS) with a workflow.
- Configure the selected Amazon ECS service to perform blue/green deployments. For more information about configuring blue/green deployments, see Blue/Green deployment with CodeDeploy in the *Amazon Elastic Container Service Developer Guide*.

How do I fix IAM capabilities errors?

Problem: You are using a Deploy AWS CloudFormation stack action, and you see `##[error] requires capabilities: [`*`capability-name`*`]` in your **Deploy AWS CloudFormation stack** action's logs.

Possible fixes: Complete the following procedure to add the capability to the workflow definition file. For more information about IAM capabilities, see Acknowledging IAM resources in AWS CloudFormation templates in the *IAM User Guide*.

Visual

To add an IAM capability using the visual editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **Visual**.
7. In the workflow diagram, choose your **Deploy AWS CloudFormation stack** action.
8. Choose the **Configuration** tab.
9. At the bottom, choose **Advanced - optional**.
10. In the **Capabilities** drop-down list, select the check box next to the capability mentioned in the error message. If the capability is not available in the list, use the YAML editor to add it.
11. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
12. Choose **Commit**, enter a commit message, and choose **Commit** again.
13. If a new workflow run doesn't start automatically, run the workflow manually to see if the changes fix the error. For more information about running a workflow manually, see Starting a workflow run manually.

YAML

To add an IAM capability using the YAML editor

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Choose your project.
3. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.
4. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.
5. Choose **Edit**.
6. Choose **YAML**.
7. In the **Deploy AWS CloudFormation stack** action, add a `capabilities` property, like this:

```
DeployCloudFormationStack:
  Configuration:
    capabilities: capability-name
```

Replace *capability-name* with the name of the IAM capability shown in the error message. Use commas and no spaces to list multiple capabilities. For more information, see the description of the `capabilities` property in the "Deploy AWS CloudFormation stack" action YAML definition.

8. (Optional) Choose **Validate** to validate the workflow's YAML code before committing.
9. Choose **Commit**, enter a commit message, and choose **Commit** again.
10. If a new workflow run doesn't start automatically, run the workflow manually to see if the changes fix the error. For more information about running a workflow manually, see Starting a workflow run manually.

How do I fix "npm install" errors?

Problem: Your AWS CDK deploy action or AWS CDK bootstrap action fails with an `npm install` error. This error may occur because you are storing your AWS CDK app dependencies in a private node package manager (npm) registry that cannot be accessed by the action.

Possible fixes: Use the following instructions to update your AWS CDK app's `cdk.json` file with additional registry and authentication information.

Before you begin

1. Create secrets for your authentication information. You'll reference these secrets in the `cdk.json` file instead of providing the cleartext equivalents. To create the secrets:
   a. Open the CodeCatalyst console at https://codecatalyst.aws/.
   b. Choose your project.
   c. In the navigation pane, choose **CI/CD**, and then choose **Secrets**.
   d. Create two secrets with the following properties:

First secret	Second secret
Name: `npmUsername`	**Name**: `npmAuthToken`

First secret	Second secret
Value: *npm-username* , where *npm-username* is the username used to authenticate to your private npm registry. (Optional) **Description**: `The username used to authenticate to the private npm registry.`	**Value**: *npm-auth-token* , where *npm-auth-token* is the access token used to authenticate to your private npm registry. For more information about npm access tokens, see About access tokens in the npm documentation. (Optional) **Description**: `The access token used to authenticate to the private npm registry.`

For more information about secrets, see Configuring and using secrets in a workflow.

2. Add the secrets as environment variables to your AWS CDK action. The action will replace the variables with real values when it runs. To add the secrets:

   a. In the navigation pane, choose **CI/CD**, and then choose **Workflows**.

   b. Choose the name of your workflow. You can filter by the source repository or branch name where the workflow is defined, or filter by workflow name.

   c. Choose **Edit**.

   d. Choose **Visual**.

   e. In the workflow diagram, choose your AWS CDK action.

   f. Choose the **Inputs** tab.

   g. Add two variables with the following properties:

First variable	Second variable
Name: `NPMUSER` **Value**: `${Secrets.npmUsername}`	**Name**: `NPMTOKEN` **Value**: `${Secrets.npmAuthToken}`

You now have two variables containing references to secrets.

Your workflow definition file YAML code should look similar to the following:

> **Note**
>
> The following code sample is from an **AWS CDK bootstrap** action; a **AWS CDK deploy** action will look similar.

```
Name: CDK_Bootstrap_Action
SchemaVersion: 1.0
Actions:
  CDKBootstrapAction:
    Identifier: aws/cdk-bootstrap@v1
    Inputs:
      Variables:
        - Name: NPMUSER
          Value: ${Secrets.npmUsername}
        - Name: NPMTOKEN
          Value: ${Secrets.npmAuthToken}
      Sources:
        - WorkflowSource
    Environment:
      Name: Dev2
      Connections:
        - Name: account-connection
          Role: codecatalystAdmin
    Configuration:
      Parameters:
        Region: "us-east-2"
```

You are now ready to use the `NPMUSER` and `NPMTOKEN` variables in your `cdk.json` file. Go to the next procedure.

To update your cdk.json file

1. Change to the root directory of your AWS CDK project, and open the `cdk.json` file.
2. Find the `"app":` property, and change it to include the code shown in *`red italics`*:

> **Note**
>
> The following sample code is from a TypeScript project. If you're using a JavaScript project, the code will look similar though not identical.

```
{
  "app": "npm set registry=https://your-registry/folder/CDK-package/ --
userconfig .npmrc && npm set //your-registry/folder/CDK-package/:always-auth=true
 --userconfig .npmrc && npm set //your-registry/folder/CDK-package/:_authToken=
\"${NPMUSER}\":\"${NPMTOKEN}\" && npm install  && npx ts-node --prefer-ts-exts bin/
hello-cdk.ts|js",
  "watch": {
    "include": [
      "**"
    ],
    "exclude": [
      "README.md",
      "cdk*.json",
      "**/*.d.ts",
      "**/*.js",
      "tsconfig.json",
      "package*.json",
...
```

3. In the code highlighted in *red italics*, replace:

   - *your-registry/folder/CDK-package/* with the path to your AWS CDK project dependencies in your private registry.
   - *hello-cdk.ts|.js* with the name of your entrypoint file. This may be a `.ts` (TypeScript) or `.js` (JavaScript) file depending on the language you're using.

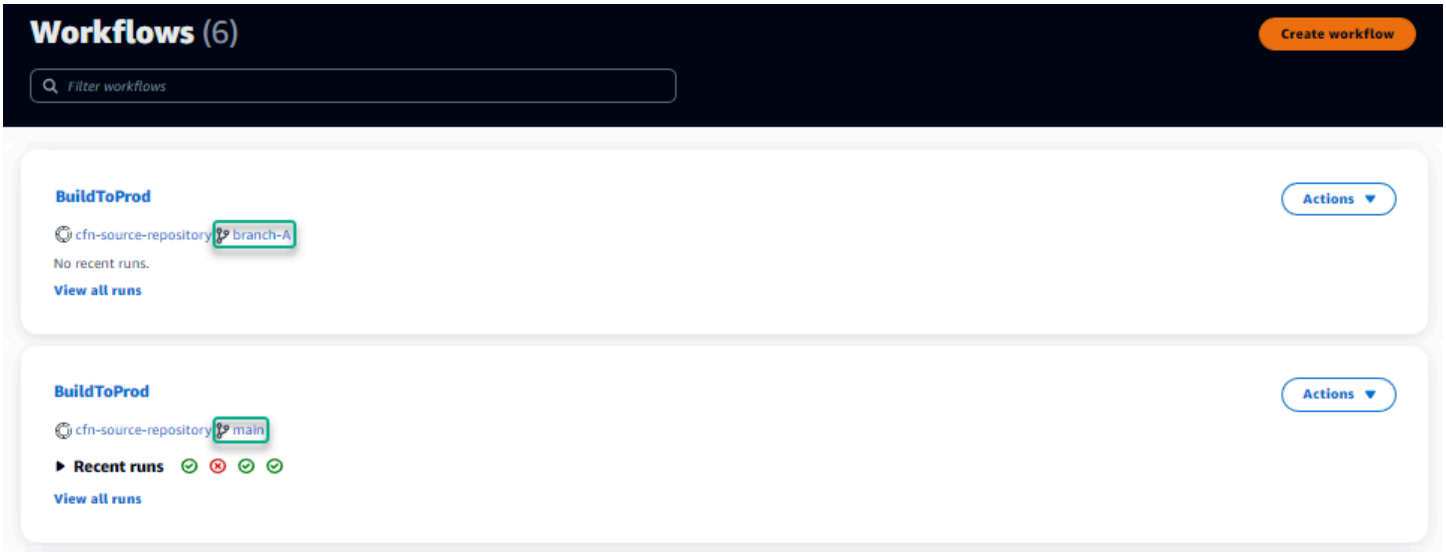
   > **Note**
   >
   > The action will replace the *NPMUSER* and *NPMTOKEN* variables with the npm username and access token that you specified in **Secrets**.

4. Save your `cdk.json` file.

5. Re-run the action manually to see if the changes fix the error. For more information about running actions manually, see Starting a workflow run manually.

Why do multiple workflows have the same name?

Workflows are stored per branch per repository. Two different workflows can have the same name if they exist in different branches. In the Workflows page, you can differentiate workflows of the same name by looking at the branch name. For more information, see Organizing your source code work with branches in Amazon CodeCatalyst.



Can I store my workflow definition files in another folder?

No, you must store all workflow definition files in the `.codecatalyst/workflows` folder. If you are using a mono repo with multiple logical projects, place all your workflow definition files in the `.codecatalyst/workflows` folder, and then use the `FilesChanged` property inside a Trigger to trigger the workflow at a specified project path. For more information, see Starting a workflow run automatically with triggers.

How do I add actions in sequence to my workflow?

By default, when you add an action to your workflow, it will have no dependencies and will run in parallel with other actions.

If you want to arrange actions in sequence, you can set a dependency on another action by setting the `DependsOn` field. You can also configure an action to consume artifacts or variables which

are outputs of other actions. For more information, see Configuring actions to depend on other actions.

Why does my workflow successfully validate but fail at runtime?

If you validated your workflow using the `Validate` button, but your workflow failed anyway, it might because a limitation in the validator.

Any errors in reference to a CodeCatalyst resource like secrets, environments, or fleets in the workflow configuration will not register during a commit. If any references that aren't valid are used, the error will only be identified when a workflow is run. Similarly, if there are any errors in your action configuration like missing a required field or typos in action attributes, they will be identified only when the workflow is run. For more information, see Creating a workflow.

Auto-discovery doesn't discover any reports for my action

Problem: I configured auto-discovery for an action that runs tests, but no reports are discovered by CodeCatalyst.

Possible fixes: This might be caused by a number of issues. Try one or more of the following solutions:

- Make sure that the tool used to run tests produces outputs in one of the formats that CodeCatalyst understands. For example, if you would like `pytest` to allow CodeCatalyst to discover test and code coverage reports, include the following arguments:

  ```
  --junitxml=test_results.xml --cov-report xml:test_coverage.xml
  ```

  For more information, see Quality report types.
- Make sure that the file extension for the outputs are consistent with the chosen format. For example, when configuring `pytest` to produce results in `JUnitXML` format, check that the file extension is `.xml`. For more information, see Quality report types.
- Make sure that `IncludePaths` is configured to include the entire file system (`**/*`) unless you are excluding certain folders on purpose. Similarly, make sure that `ExcludePaths` don't exclude directories where you expect your reports to be located.
- If you manually configured a report to use a specific output file, it will be excluded from auto-discovery. For more information, see Quality reports YAML example.

- Auto-discovery may not find reports because the action failed before any outputs were generated. For example, the build may have failed before any unit tests have been run.

My action fails on auto-discovered reports after I configure success criteria

Problem: When I enable auto-discovery and configure success criteria, some of the reports don't meet the success criteria and the action fails as a result.

Possible fixes: To resolve this, try one or more of the following solutions:

- Modify `IncludePaths` or `ExcludePaths` to exclude reports that you are not interested in.
- Update success criteria to allow all reports to pass. For example, if two reports were discovered with one having line coverage of 50% and another one of 70%, adjust the minimum line coverage to 50%. For more information, see Success criteria
- Turn the failing report into a manually configured report. This allows you to configure different success criteria for that specific report. For more information, see Configuring success criteria for reports.

Auto-discovery generates reports that I don't want

Problem: When I enable auto-discovery, it generates reports that I don't want. For example, CodeCatalyst generates code coverage reports for files included in my application's dependencies stored in `node_modules`.

Possible fixes: You can adjust the `ExcludePaths` configuration to exclude unwanted files. For example, to exclude `node_modules`, add `node_modules/**/*`. For more information, see Include/exclude paths.

Auto-discovery generates many small reports for a single test framework

Problem: When I use certain test and code coverage reporting frameworks, I noticed that auto-discovery generates a large number of reports. For example, when using the Maven Surefire Plugin, auto-discovery produces a different report for each test class.

Possible fixes: Your framework may be able to aggregate outputs into a single file. For example, if you are using Maven Surefire Plugin, you can use `npx junit-merge` to aggregate the files manually. The full expression may look like this:

```
mvn test; cd test-package-path/surefire-reports && npx junit-merge -d ./ && rm
 *Test.xml
```

Workflows listed under CI/CD don't match those in the source repository

Problem: The workflows displayed on the **CI/CD**, **Workflows** page do not match those in the `~/.codecatalyst/workflows/` folder in your source repository. You may see the following mismatches:

- A workflow appears on the **Workflows** page, but a corresponding workflow definition file does not exist in your source repository.
- A workflow definition file exists in your source repository, but a corresponding workflow does not appear on the **Workflows** page.
- A workflow exists in both the source repository and **Workflows** page, but the two are different.

This problem may occur if the **Workflows** page hasn't had time to refresh, or if a workflow quota was exceeded.

Possible fixes:

- Wait. You usually have to wait two or three seconds after a commit to source before you see the change on the **Workflows** page.
- If you've exceeded a workflow quota, do one of the following:

> **Note**
>
> To determine whether a workflow quota was exceeded, review Quotas for workflows, and cross-check the documented quotas against the workflows in your source repository or on the **Workflows** page. There is no error message to indicate that a quota was exceeded, so you'll have to investigate on your own.

- If you've exceeded the **Maximum number of workflows per space** quota, delete some workflows and then perform a test commit against the workflow definition file. An example of a test commit might be to add a space to the file.
- If you've exceeded the **Maximum workflow definition file size** quota, change the workflow definition file to reduce its length.
- If you've exceeded the **Maximum number of workflow files processed in a single source event** quota, perform several test commits. Modify fewer than the maximum number of workflows in each commit.
- Increase the workflow quotas by turning on the paid tiers billing. For more information, see Managing billing in the Amazon CodeCatalyst Administrator Guide.

I can't create or update workflows

Problem: I want create or update a workflow, but I see an error when I try to commit the change.

Possible fixes: Depending on your role in the project or space, you might not have permissions to push code to source repositories in the project. The YAML files for workflows are stored in repositories. For more information, see Workflow definition files. The **Space administrator** role, **Project administrator** role, and **Contributor** role all have permission to commit and push code to repositories in a project.

If you have the **Contributor** role but cannot create or commit changes to workflow YAML in a specific branch, there might be a branch rule configured for that branch that prevents users with that role from pushing code to that particular branch. Try creating a workflow in a different branch, or commiting your changes to a different branch. For more information, see Manage allowed actions for a branch with branch rules.

Troubleshooting problems with issues

The following information can help you troubleshoot common problems with issues in CodeCatalyst.

Topics

I can't choose an assignee for my issue

Problem: When creating an issue, the list of assignees is empty.

Possible fixes: The list of assignees is directly linked to the CodeCatalyst users listed as members for the project. To verify that user profile access is functioning properly, choose the profile icon and then choose **User profile**. If the user profile information does not populate, check the health report for any incidents. If it does populate, file a service ticket.

Troubleshooting problems with search in CodeCatalyst

Consult the following sections to troubleshoot problems related to searching in CodeCatalyst. For more information about workflows, see Search for code, issues, projects, and users in CodeCatalyst.

Topics

I can't find a user in my project

Problem: When I try to view a user's details, I don't see their information in the project.

Possible fixes: Search doesn't currently support searching for users within a project. To search for users with access to your space, switch to **This space** in QuickSearch, or remove any project filters you might have specified using the advanced query language.

I don't see what I'm looking for in my project or space

Problem: Results don't appear when I try to search for particular information.

Possible fixes: Content updates are likely to take a few seconds to update in search results. Large updates can take several minutes.

For resources that haven't been recently updated, you might need to refine your search. You can refine by adding more keywords or using the advanced query language. For more information about refining your queries, see Refining your search query.

Number of search results keep changing when I navigate through the pages

Problem: The number of search results appear to change when I go to the next page, so it's not clear how many total results there are.

Possible fixes: When navigating through pages of search results, you might see a change in the number of search results that match your query. The number of results might update to reflect a more accurate number of matches discovered as you navigate through the pages.

As you navigate through the results, you might see the following message: **No results for "test"**. You will receive the message if you don't have access to the remaining results.

My search query isn't being completed

Problem: The results of my search query aren't showing up, and it appears to be taking too long.

Possible fixes: Your search may not be completed when there are many searches being made at the same time in the space, either programmatically or because of high team activity. If you're running programmatic searches, pause or decrease them. Otherwise, try again in a few seconds.

Troubleshooting problems with extensions

Consult the following sections to troubleshoot problems related to extensions in CodeCatalyst. For more information about extensions, see Add functionality to projects with extensions in CodeCatalyst.

Topics

I can't see the changes to a linked third-party repositories or search for results of those changes

Problem: The changes in my third-party reposiory aren't showing up in CodeCatalyst.

Possible fixes: CodeCatalyst currently doesn't support detecting changes in the default branch for linked repositories. To change the default branch for a linked repository, you must first unlink it

from CodeCatalyst, change the default branch, and then link it again. For more information, see Linking GitHub repositories, Bitbucket repositories, GitLab project repositories, and Jira projects in CodeCatalyst.

Troubleshooting problems with accounts associated with your space

In CodeCatalyst, you can add an AWS account to your space to grant permissions to resources and for billing purposes. The following information can help you troubleshoot common issues with associated accounts in CodeCatalyst.

Topics

My AWS account connection request receives an invalid token error

Problem: When creating a connection request with a connection token, the page does not accept the token and shows an error stating that the token is not valid.

Possible fixes: Make sure you provide the account ID that you want to add to your space. You must have administrative permissions for your AWS account or be able to work with your administrator to add the account.

When you choose to verify the account, a new browser window will open in the AWS Management Console. The same account is required to be logged in on the console side. Try again after verifying the following:

- You are logged in to the AWS Management Console with the same AWS account that you want to add to your space.
- You are logged in to the AWS Management Console with the Region set to the correct Region for your space.

- If you have arrived from the billing page and you want to add the AWS account as a specified billing account for your space, make sure the account has not reached the quota as a billing account for another space or spaces.

My Amazon CodeCatalyst project workflow fails with an error for the configured account, environment, or IAM role

Problem: When the workflow runs and does not find a configured account or IAM roles associated with your space, you must fill in the role, connection, and environment fields manually in the workflow YAML. View the failed workflow action, and note whether the error messages are as follows:

- The role is not available for use with the connection associated with the environment.
- Action did not succeed. Status: FAILED; The provided value for account connection or environment is not valid. Verify the connection is associated with your space and the environment is associated with your project.
- Action did not succeed. Status: FAILED; The provided value for IAM role is not valid. Verify the name exists, the IAM role is added to your account connection, and the connection is already associated with your Amazon CodeCatalyst space

Possible fixes: Make sure that the workflow YAML fields have accurate values for Environment, Connections, and Role. The CodeCatalyst workflow actions that require an environment are build or deploy actions that run AWS resources or that generate AWS resource stacks.

Choose the failed workflow action block and then choose **Visual**. Choose the **Configuration** tab. If the **Environment**,**Connection name**, and **Role name** fields are not populated, then you will need to manually update the workflow. Use the following steps to edit your workflow YAML:

- Expand the `/.codecatalyst` directory, and then expand the `/workflows` directory. Open the workflow YAML file. Make sure that the IAM roles and account information are specified in the YAML that you have configured for your workflow. Example:

```
Actions:
  cdk_bootstrap:
    Identifier: action-@v1
    Inputs:
      Sources:
```

```
      - WorkflowSource
  Environment:
    Name: Staging
    Connections:
      - Name: account-connection
        Role: build-role
```

The **Environment, Connection, and Role** properties are required to run CodeCatalyst workflow build and deploy actions with AWS resources. For an example, see the CodeCatalyst build action reference YAML parameters for Environment, Connections, and Role.

- Make sure your space has an account added to it, and make sure that the account has the appropriate IAM role or roles added to the account. You can adjust or add accounts if you have the **Space administrator** role. For more information, see Allowing access to AWS resources with connected AWS accounts.

I need an associated account, role, and environment to create a project

Problem: In the project creation options, my project either doesn't have an added account available in my space, or I need another account added to my space for my project to use.

Possible fixes: For your space, you can add authorized AWS accounts to add them to your project if you have the **Space administrator** role. You must also have an AWS account where you have administrative permissions or can work with your AWS administrator.

To make sure an account and role will be available in the project creation screen, you must first add the account and roles. For more information, see Allowing access to AWS resources with connected AWS accounts.

You have the option to choose to create a service role with a role policy called the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role policy. The role will have a name `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* with a unique identifier appended. For more information about the role and role policy, see Understanding the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** service role. For the steps to create the role, see Creating the **CodeCatalystWorkflowDevelopmentRole-*spaceName*** role for your account and space. The role is added to your account and available in project creation pages in CodeCatalyst.

I cannot access the Amazon CodeCatalyst Spaces page in the AWS Management Console

Problem: When I try to access the Amazon CodeCatalyst page in the AWS Management Console to add an account to my CodeCatalyst space or add roles to an account in AWS, I receive a permissions error.

Possible fixes:

For your space, you can add authorized AWS accounts to add them to your project if you have the **Space administrator** role. You must also have an AWS account where you have administrative permissions or can work with your AWS administrator. You must first make sure you are signed in to the AWS Management Console with the same account that you want to manage. After you are signed in to the AWS Management Console, you can open the console and try again.

Open the Amazon CodeCatalyst page in the AWS Management Console at https://us-west-2.console.aws.amazon.com/codecatalyst/home?region=us-west-2#/.

I want a different account as my billing account

Problem: When I set up my CodeCatalyst login, I completed several steps to set up my space and associate an authorized AWS account. Now, I want to authorize a different account for billing.

Possible fixes: For your space, you can authorize billing accounts if you have the **Space administrator** role. You must also have an AWS account where you have administrative permissions or can work with your AWS administrator.

For more information, see Managing billing in the Amazon CodeCatalyst Administrator Guide.

Troubleshooting problems between Amazon CodeCatalyst and the AWS SDKs or the AWS CLI

The following information can help you troubleshoot common issues when working with CodeCatalyst and the AWS CLI or the AWS SDKs.

Topics

I receive an error when I enter aws codecatalyst at a command line or terminal saying it's an invalid choice

Problem: When I try to use the AWS CLI with CodeCatalyst, one or more of the **aws codecatalyst** commands are not recognized as valid.

Solution: The most common cause for this problem is that you are using a version of the AWS CLI that does not contain the most recent updates for the latest services and commands. Update your installation of the AWS CLI and then try again. For more information see Setting up to use the AWS CLI with CodeCatalyst.

I receive a credentials error when I run aws codecatalyst commands

Problem: When I try to use the AWS CLI with CodeCatalyst, I receive a message stating `You can configure credentials by running "aws configure".` or `Unable to locate authorization token`.

Solution: You must configure an AWS CLI profile to work with CodeCatalyst commands. For more information see Setting up to use the AWS CLI with CodeCatalyst.

Understanding current service status with the CodeCatalyst health report

The Amazon CodeCatalyst health report is a public dashboard that provides users with an aggregated list of up-to-the-minute notifications regarding resource performance and availability of services in CodeCatalyst that have widespread impact. You can see which resources are having problems and may affect applications in CodeCatalyst. This allows you to track outages and other resource downtime system wide. When an incident occurs, a blue indicator appears on the health report icon. Additionally, CodeCatalyst automatically sends an alert and email notification to all users with the **Space administrator** role in the project, providing details and a history of the incident in near real time.

The dashboard provides a list of all active events and a record of up to 100 previous incidents occurring in the last 30 days. You can organize the list of incidents based on the date the incident was updated. You can also refresh the list of incidents for up to the minute updates.

Here is a possible workflow for using the CodeCatalyst health report:

Mateo Jackson is a developer in the Budding Space with Space administrator permissions. While attempting to create a pull request, he keeps getting an error message. He checks his email and discovers he received an auto-generated system incident email from CodeCatalyst providing a detailed history about the system issue affecting his space. He chooses **View update** and is taken to the CodeCatalyst health report where he can view all system-reported incidents. He chooses the incident from the list to find out more information. A split screen opens that provides a timestamp of the last update, history, impacted capabilities, start time, and current status of the incident. He can also see that the issue is ongoing, but the service team has begun working on it. Each time there is an update to the history or status of the incident, he receives an email. In the event that he does not have access to his email, he can choose the bell icon in the top panel to get to the CodeCatalyst health report.

CodeCatalyst health report concepts

Learning the following concepts will help you understand the CodeCatalyst health report and how they enable you to track the health of your applications, services, and resources.

Incident

The *incident* is the system event that is affecting applications and resources within CodeCatalyst. You can choose the incident to view a detailed history of the event, including the time it began and if the service team is working on resolving it.

Status

The *status* is the real-time status of the incident. It will show as **Ongoing** or **Resolved**.

Impacted capabilities

The *impacted capabilities* are the resources or applications affected by the incident. A single incident can affect multiple areas in the system, including **pull requests**, **issues**, **workflows**, **test**, **deploy**, and **source**.

Updated on

Updated on provides a timestamp of the last update for the incident.

AWS Support for Amazon CodeCatalyst

When you create a space, you must connect an AWS account and designate it as the billing account for your space. The AWS account you designate as your billing account is also where you access your AWS Support plan for Amazon CodeCatalyst. If you need support, you can create support cases from this designated AWS account.

CodeCatalyst users in a space use the AWS Support for Amazon CodeCatalyst page in CodeCatalyst to manage support cases. You can upgrade to an AWS Support plan such as Business Support or Enterprise Support to create and manage CodeCatalyst technical support cases in CodeCatalyst. Support is available through phone, web, or chat for support cases.

Only cases specific to the CodeCatalyst service and resources can be supported through AWS Support for Amazon CodeCatalyst. CodeCatalyst resources include resources deployed within CodeCatalyst and by users in CodeCatalyst, but these do not include resources deployed for other AWS or third-party services. If you need support for any other AWS service, you must open it through the AWS Management Console.

To change your support plan, see Changing support plans.

> **Note**
>
> Developer Support plans are not designed for production environments. If a space billing account has a Developer Support plan, this plan does not **cascade** to all space administrators and space members within AWS Support in CodeCatalyst.

Billing for AWS Support for Amazon CodeCatalyst

When you create a space in CodeCatalyst, users in the space can create and manage support cases from AWS Support for Amazon CodeCatalyst. You can create two types of customer cases:

- **Account and billing** support cases are available to all CodeCatalyst users in the space. You can get help with billing and account questions based on your permissions in CodeCatalyst.
- **Technical** support cases connect you to a technical support engineer for help with service-related technical issues and extensions to third-party applications. If you have Basic Support, you can't create a technical support case.

The AWS account designated as the billing account for the space must have a Business Support or Enterprise Support plan for the space to use AWS Support for CodeCatalyst for technical cases.

> **Note**
>
> If your space uses AWS Support for Amazon CodeCatalyst from an account that doesn't have a Business Support or Enterprise Support plan, you can still use AWS Support for Amazon CodeCatalyst for account and billing cases.

For technical support, you must open all cases through the CodeCatalyst console. You cannot create technical support cases for CodeCatalyst from AWS Support in the AWS Management Console.

> **Note**
>
> **Service limit increase** requests are not available from AWS Support for Amazon CodeCatalyst. These requests can only be submitted by the root user for the space billing account in the AWS Support Center Console.

AWS Support for Amazon CodeCatalyst has the same support agreements as AWS Support, with the following considerations:

- Severity lists, response times, and SLAs in AWS Support apply for support cases in AWS Support for CodeCatalyst, as detailed in Choosing a severity.
- Space administrators and space members cannot use the AWS Support APIs or AWS SDK or AWS Support app in Slack to create cases for CodeCatalyst. CodeCatalyst support cases can only be submitted from CodeCatalyst.

> **Note**
>
> CodeCatalyst is not fully integrated with AWS Trusted Advisor or AWS Incident Detection and Response. Validate how CodeCatalyst is integrated to ensure your business practices are aligned with the current integration.

You must be a user in the space where you want to request support.

> **Note**
>
> If you have more than one builder in your space, we recommend that you purchase a Business Support or Enterprise Support plan. These plans provide technical support for the space for up to 5,000 builders.

The AWS account designated as the billing account for the space uses the `AWSRoleForCodeCatalystSupport` role and AmazonCodeCatalystSupportAccess managed policy. This allows CodeCatalyst users in a space to access the AWS Support for Amazon CodeCatalyst page. For more information about this role and policy, see AmazonCodeCatalystSupportAccess. For other considerations about billing, see Managing billing in the Amazon CodeCatalyst Administrator Guide.

Here is a possible flow for a builder creating a support case in CodeCatalyst:

Mateo Jackson is a developer on a project in CodeCatalyst. After signing up the AWS account that manages billing with AWS Support for Amazon CodeCatalyst and upgrading to a Business Support plan, all builders in the space can create technical support cases. Mateo submits a technical support case for a failed workflow in their project. Mateo uses the AWS Support for Amazon CodeCatalyst page to fill out the form and create a case, providing the workflow ID and other details in the request. The case is created with a case ID and includes the account ID of the AWS account designated as the billing account and associated with support plan for the space.

While all builders can create support cases in AWS Support for CodeCatalyst, you are not charged for each case created. You can open virtually unlimited cases and contacts based on the AWS Support Premium plan you purchase on your space billing account.

> **Note**
>
> The space billing account is the AWS account that you are charged for CodeCatalyst users and resources. If you have deployed to additional AWS accounts, contact AWS Support through the AWS Management Console for assistance with resources deployed to other services.
> You can identify the AWS account you deployed to from the workflow.

Setting up your space for AWS Support for Amazon CodeCatalyst

AWS Support for Amazon CodeCatalyst manages support cases as part of AWS Support API integration with CodeCatalyst.

The `AWSRoleForCodeCatalystSupport` role is a service role that is used for support cases in your space. The role must be added to the designated billing account for the space. For more information or to create the role, see Creating the **AWSRoleForCodeCatalystSupport** role for your account and space.

> **Note**
>
> For a space that was created before April 20, 2023, you must create the role in order for support for CodeCatalyst to work for your space. If creating a space after April 20, 2023, you can create the role during space creation, on the Billing details page in CodeCatalyst, or by clicking the support banner link in CodeCatalyst.

To set up support for your space

1. When you create a CodeCatalyst space, you are instructed to connect a billing account. The designated billing account for the space will be billed by AWS. For more information about creating a space, see Creating your first space and development role (starting without an invitation).
2. When you create a CodeCatalyst space, the option is available to create the `AWSRoleForCodeCatalystSupport` service role that allows CodeCatalyst users to access support. The role uses the managed policy `AmazonCodeCatalystSupportAccess`. The role must be added to the AWS account designated as the billing account for the space. For more information about creating this role, see Creating the **AWSRoleForCodeCatalystSupport** role for your account and space.
3. For the designated billing account for the space, the space administrator is recommended to purchase a Business Support or Enterprise Support plan for the AWS account. All members in the space will be able to manage support cases from AWS Support for Amazon CodeCatalyst, and channels of support will be aligned to the AWS Support plan you have purchased where integrations are completed.

4. To create and manage support cases in CodeCatalyst, see Creating a CodeCatalyst support case in CodeCatalyst.

Accessing support for CodeCatalyst in the AWS Management Console

If the support enabled billing account for a space is disconnected, AWS Support cases associated with the previous space billing account and associated support plan will no longer be visible in AWS Support for Amazon CodeCatalyst. The root user for that billing account can view and resolve old cases from the AWS Management Console and can set up IAM permissions for AWS Support for other users to view and resolve old cases. You will still be able to partake in the benefits of your support plan from the AWS Management Console for all other AWS services and complete any CodeCatalyst support cases that were not previously resolved.

For more information, see Updating, resolving, and reopening your case in the *AWS Support User Guide*.

Support cases for general how-to information about CodeCatalyst can also be opened in the AWS Management Console, but no technical support can be received through this channel for CodeCatalyst. For more information, see Creating support cases and case management in the *AWS Support User Guide*.

Here is a possible flow for a user resolving a support case for CodeCatalyst in the AWS Management Console:

While all builders can create support cases with AWS Support for Amazon CodeCatalyst, the support requests are billed from the account that is designated as the billing account for the space. Mateo Jackson is a developer on a project in CodeCatalyst who opened a technical support case for a failed workflow in their project. However, the billing account for the space that was signed up with AWS Support for Amazon CodeCatalyst and had purchased a Business Support plan has been disconnected from the space. The only way for Mateo to view the latest communication and resolve cases opened for CodeCatalyst is to manage the case ID from the AWS Support Center in the AWS Management Console. To do this, Mateo is given IAM permissions from the root user of the previous space billing account attached to their support case and resolves the case through AWS Support in the console.

Important

If you change the designated billing account for your space, your AWS Support plan will still be accessible until the end of the month through the AWS Management Console only. You will need to repurchase AWS Support on the updated billing account to continue accessing your previously created support cases in CodeCatalyst. We recommend waiting until you have resolved all of your support cases to change space billing accounts to avoid any impact to accessing your support cases through AWS Support for Amazon CodeCatalyst.

Creating a CodeCatalyst support case in CodeCatalyst

You can create a support case in the AWS Support for Amazon CodeCatalyst page.

An AWS Builder ID can only get support for the alias they are authenticated with and only for resources based on their permissions. Account and billing options are available for all space administrators and space members. However, users can only get support for resources they have access to in CodeCatalyst and not in relation to managing billing for the account.

You can create an Account and Billing case or technical support case for your CodeCatalyst resources using the AWS Support for CodeCatalyst page for your space.

Note

Only cases specific to the CodeCatalyst service and resources can be supported through AWS Support for Amazon CodeCatalyst. CodeCatalyst resources include resources deployed within CodeCatalyst and by users in CodeCatalyst, but these do not include resources deployed for other AWS or third-party services. If you need support for any other AWS service, you must open it through the AWS Management Console.

To create a support case in CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. At the top of the page, choose the **?** icon, and then choose **Support**.
4. Choose **Create case**.
5. Choose one of the following options:

   - **Account and billing**
   - **Technical**

   > **Note**
   >
   > In AWS Support for Amazon CodeCatalyst, if a Business Support or Enterprise Support plan is added to the space billing account, CodeCatalyst technical case support will be available to all space administrators and space members. For troubleshooting information, see I cannot create technical support cases for my space.
   > AWS Support plans do not span across spaces. If you are a member of multiple spaces, your space administrator will need to purchase an AWS Support Premium plan for every space in order to receive technical support across all spaces.

6. Choose the **Service**, **Category**, and **Severity**. For information about choosing a severity, see Choosing a severity.

   - **General guidance**
   - **System impaired**
   - **Production system impaired**
   - **Production system down**
   - **Business-critical system down**

7. Choose **Next step: Additional information**.
8. On the **Additional information** page, for **Subject**, enter a title about your issue.
9. For **Description**, follow the prompts to describe your case, such as the following:

- Troubleshooting information that is specific to CodeCatalyst, such as workflow ID, logs, or screenshots
- Error messages that you received
- Troubleshooting steps that you followed

> **Note**
>
> Don't share any sensitive information in case correspondences, such as credentials, credit cards, signed URLs, or personally identifiable information.

10. (Optional) Choose **Attach files** to add any relevant files to your case, such as error logs or screenshots. You can attach up to three files. Each file can be up to 5 MB.
11. In **Space name**, the name of your space displays.
12. In **Builder name**, the full name associated with your AWS Builder ID autopopulates.
13. (Optional) Choose the project in **Project name (if applicable)**.

> **Note**
>
> You will only be shown projects you have permissions to. If you need access to another project, ask your project administrator to provide you with access before creating a support case.

14. Choose **Next step: Contact us**.
15. In **Preferred contact language**, choose the default. Only **English** is available at this time.
16. Choose the **Web**, **Phone**, or **Chat** option for contact method.
17. Review your case details, and then choose **Submit**. Your case ID number and summary appear.

    The support case is created at the space level and is viewable by all members with access to the space and project (if selected) that are defined in your support case. There is no way to omit a support case from individual users at this time.

Resolving a support case in CodeCatalyst

You can resolve open support cases from the AWS Support for Amazon CodeCatalyst page.

You must have a **Space administrator** or **Space member** role in the space where you want to resolve a support case. If you do not have the **Space administrator** role, or if a project was selected when the case was created, you will also need to have membership to the project to view and resolve the case.

To resolve an open support case in CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

   > **Tip**
   >
   > If you belong to more than one space, choose a space in the top navigation bar.

3. At the top of the page, choose the **?** icon, and then choose **AWS Support for Amazon CodeCatalyst**.
4. Choose the link for the support case that you want to manage. Choose **Resolve case**.

Reopening a support case in CodeCatalyst

You can use the reopen a resolved support case from the AWS Support for Amazon CodeCatalyst page.

> **Note**
>
> You can reopen your support case up to 14 days from when your issue was resolved. However, you can't reopen a case that has been inactive for more than 14 days. If you're unable to reopen your case, open a new case and include the previous case ID as a reference.
> If you reopen an existing case that has different information than your current issue, the support agent might ask you to create a new case.

To reopen a support case in CodeCatalyst

1. Open the CodeCatalyst console at https://codecatalyst.aws/.
2. Navigate to your CodeCatalyst space.

> **Tip**
>
> If you belong to more than one space, choose a space in the top navigation bar.

3. At the top of the page, choose the **?** icon, and then choose **AWS Support for CodeCatalyst**.
4. Choose the link for the support case that you want to manage. Choose **Reopen**. Choose **OK** on the confirmation screen, and then choose **Submit**.
5. Fill in the **Description** with the latest information about the same issue. Don't share any sensitive information in case correspondences, such as credentials, credit cards, signed URLs, or personally identifiable information.

Quotas for CodeCatalyst

The following table describes quotas and limits for Amazon CodeCatalyst. You can find additional information for specific aspects of CodeCatalyst in the following topics:

- Quotas for source repositories in CodeCatalyst
- Quotas for identity, permission, and access in CodeCatalyst
- Quotas for workflows
- Quotas for Dev Environments in CodeCatalyst
- Quotas for projects
- Quotas for blueprints in CodeCatalyst
- Quotas for spaces
- Quotas for issues in CodeCatalyst

Maximum number of spaces in an account	5
Maximum number of spaces a user can create in a calendar month	5
Minimum number of AWS accounts for a space	1
Maximum number of account connections for a space	5,000
Maximum number of AWS accounts as the billing account for a space	1
Maximum number of spaces in the Free tier for which a single AWS account can be specified as the billing account	3
Maximum number of spaces in a paid tier for which a single AWS account can be specified as the billing account	15

Maximum number of VPC connections for a space	100
Maximum number of projects in a space	100
Maximum number of projects to which a user can belong	1,000
Space descriptions	Space descriptions are optional. If specified, they must be between 0 and 200 characters in length. They can contain any combination of letters, numbers, spaces, periods, underscores, commas, dashes, and the following special characters: `? & $ % + = / \ ; : \n \t \r`
Space names	Space names must be unique across CodeCatalyst. You cannot reuse names of deleted spaces. Space names must be between 3 and 63 characters in length. They must also begin with an alphanumeric character. Space names can contain any combination of letters, numbers, periods, underscores, and dashes. They cannot contain any of the following characters: `! ? @ # $ % ^ & * ( ) + = { } [ ] | /\ > < ~ ` ' " ; :`

Document history for Amazon CodeCatalyst

The following table describes the documentation history and updates for the overall documentation for CodeCatalyst.

Change	Description	Date
New content: Configuring manual-only triggers	Added a Configuring manual-only triggers topic.	June 26, 2024
Updated content: Creating an environment	Updated the Creating an environment section and several others to reflect a new feature that lets you specify a default IAM role when creating a CI/CD environme nt. This role will be assigned to all workflow actions that are associated with the environment. You no longer have to assign IAM roles directly to actions.	June 21, 2024
New content: Tutorial: Pull dependencies from a CodeCatalyst package repository using a workflow	Added a tutorial that describes how to configure a workflow to pull dependenc ies from a CodeCatalyst package repository.	June 20, 2024
Updated content: Use GitLab project repositories with blueprints	Updated documentation for ability to create a GitLab project repository when Creating a project with a blueprint, Adding a blueprint in a project to integrate resources, or creating a custom blueprint.	June 19, 2024

Updated content: Tutorial: Using generative AI features	Updated the tutorial to reflect ability to use Amazon Q to choose a blueprint when creating a project or adding a blueprint to an existing project.	June 18, 2024
Updated content: Creating a source repository	Updated the documentation to include information about creating empty repositories.	June 18, 2024
New content: Creating a project in CodeCatalyst with Amazon Q	Added new sections to Creating a project titled Best practices when using Amazon Q to create projects or add functionality with blueprints and Best practices for using blueprints with projects with instructions to create projects and add blueprints with Amazon Q.	June 18, 2024
New content: Clone an existing Git repository into a source repository	Added a new topic about how to use Git to push a mirror clone or a local repo to an empty source repository in CodeCatalyst.	June 18, 2024
New content: Support for Maven, NuGet, and Python package types	Added documentation for using Maven, NuGet, and Python packages in CodeCatalyst.	June 17, 2024

Updates to spaces and account connections	You can now connect the AWS account specified as the billing account to multiple CodeCatalyst spaces. For spaces set up for identity federation, one Identity Center instance can be connected to multiple spaces. See Allowing access to AWS resources with connected AWS accounts, Troubleshooting problems with accounts associated with your space, and Quotas for CodeCatalyst.	June 13, 2024
Updated content: Working with roles	Updated the documentation for roles to include permissions for using Amazon Q to recommend and create tasks for an issue.	June 13, 2024
Updated content: Working with roles	Updated the documentation for roles to include permissions for creating, updating, and removing links between issues.	June 13, 2024
Updated content: Tutorial: Using generative AI features	Updated the tutorial to include information about using Amazon Q to recommend tasks for an issue.	June 13, 2024
Updated content: Managing tasks on issues	Added information about using Amazon Q to recommend tasks for an issue in CodeCatalyst.	June 13, 2024

New content: Link an issue to another issue	Added a new topic about linking issues to other issues in CodeCatalyst.	June 13, 2024
Updated content: "AWS CDK deploy" action YAML definition	Fixed the description of the `Region` property.	June 12, 2024
Updated content: Use third-party repositories with blueprints	Updated documentation for ability to create a GitHub repository when Creating a project with a blueprint, Adding a blueprint in a project to integrate resources, or creating a custom blueprint.	June 6, 2024
New content: Added steps for working with personal connections	Added steps for creating and deleting personal connections. Personal connections allow you to manage GitHub resources for projects and blueprints in Amazon CodeCatalyst.	June 6, 2024
New content: **Bitbucket repositories** extension	Added new content for using **Bitbucket repositories** extension in CodeCatalyst.	June 5, 2024
New content: Action types	Updated the Third-party actions topic to mention the **SonarCloud Scan** action.	May 29, 2024
Updated content: "AWS CDK deploy" action YAML definition	Fixed the `CdkRootRootPath` example.	May 28, 2024

Updated content	Updated topic titles and reorganized content to improve readability and discovery. If you'd like to provide feedback on these changes, use this Provide feedback link.	May 17, 2024
New content: Viewing the history of changes to a file	Updated the documentation to reflect new functionality for viewing the history of changes to a file in a source repository.	May 1, 2024
Updated content: Tutorial: Using generative AI features	Updated the tutorial to reflect integration with Amazon Q Developer.	April 29, 2024
Updated content: Tutorial: Using generative AI features	Updated the tutorial to reflect allowing Amazon Q to analyze issues for complexity, suggest and create tasks, and work in tasks in an issue.	April 22, 2024
New content: Gating a workflow run	Added Gating a workflow run, Requiring approvals on workflow runs, and several other topics related to workflow approvals.	April 22, 2024
New content: Tutorial: Creating a full-stack application with composable PDK blueprints	Added a new tutorial for using AWS Project Development Kit (AWS PDK) blueprints in a Amazon CodeCatalyst project.	April 9, 2024

New content: Using tasks to break down issues into smaller objectives	Added content to support the launch of tasks in issues. Tasks can be added to issues to further break down, organize, and track the work of that issue.	April 4, 2024
Updated content: Sharing data between actions in a workflow using artifacts	Updated the Sharing data between actions in a workflow using artifacts topic to include two new subtopics : Can I share artifacts without specifying them as outputs and inputs? and Can I share artifacts between workflows?	April 2, 2024
Updated content: Limitations of GitHub Actions in CodeCatalyst	Updated the Limitations of GitHub Actions in CodeCatalyst topic to indicate that GitHub Actions run on an older runtime environment Docker image.	April 2, 2024
New content: "AWS CDK deploy" action YAML definition	Added a new `CloudAssemblyRootPath` property to the "AWS CDK deploy" action YAML definition.	April 1, 2024
Updated content: Specifying runtime environment Docker images	Updated the Specifying runtime environment Docker images topic to include information about the new March 2024 runtime environment image.	March 26, 2024

Updated content: Working with roles	Consolidated role permissio n information into a single table. The table is in a new Viewing the permissions available for each role topic.	March 18, 2024
New content: View all spaces and projects for a user	Added information about viewing a listing on the user home page that shows each CodeCatalyst space or project for the signed-in user in CodeCatalyst. See View all spaces and projects for a user.	March 18, 2024
New content: Example: A trigger with a pull and branches	Added an example of a pull request trigger. Made small corrections throughout the Starting a workflow run automatically with triggers topic.	March 11, 2024
Updated content: Working with roles	Updated the documenta tion for the roles to include permissions for creating, deleting, and viewing environments.	March 4, 2024
Updated content: Tutorial: Using generative AI features	Updated the tutorial to reflect changes when creating and assigning issues to Amazon Q.	March 4, 2024
New content: Issues components	Added new content on how to work with issues component s as a custom blueprints developer.	February 27, 2024

Updated content: Action types	Updated the CodeCatalyst Labs actions topic to include a list of CodeCatalyst Labs actions.	February 21, 2024
Updated content: Working with pull requests	Updated the documentation to reflect new functionality with approval rules and overriding requirements to merge a pull request.	February 15, 2024
Updated content: Merging pull requests	Added documentation for pull requests to include information about overriding merge requirements to merge a pull request that has not yet received approvals from required reviewers or met approval rules.	February 15, 2024
New content: Manage approval rules	Added documentation for pull requests to include information about creating and managing approval rules.	February 15, 2024
Updated content: Working with roles	Updated the documentation for the roles to include permissions for working with approval rules and pull requests.	February 14, 2024
Updated content: How do I fix "Workflow definition has *n* errors" errors?	Updated the How do I fix "Workflow definition has *n* errors" errors? section to include more troubleshooting tips.	February 9, 2024

New content: Viewing the workflow status	Added a section that describes workflow states.	February 9, 2024
New content: Viewing the workflow status	Added a section that describes workflow states.	February 9, 2024
Update content: Quotas for workflows	Updated the Quotas for workflows topic with the *Maximum number of actions per workflow* and *Maximum number of environments associated with an AWS account per space* quotas.	February 7, 2024
Updated content: Creating an environment	Updated the Creating an environment section to indicate that you can use a maximum of one account connection per environment.	January 31, 2024
New content: Custom blueprints GitHub repository	Added new content for GitHub repository that is made publicly available.	January 10, 2024
Updated content: Configuring npm with CodeCatalyst	Updated general configuration instructions for using npm with CodeCatalyst, and added clarity around `always-auth=true` option.	January 5, 2024
Updated content: Working with pull requests	Updated the documentation to reflect new functionality with the generative AI features in CodeCatalyst.	November 28, 2023

Updated content: Creating an issue	Updated the documentation to reflect new functionality with the generative AI features in CodeCatalyst.	November 28, 2023
New content: Tutorial: Using generative AI features	Added a tutorial for using the generative AI features in Amazon CodeCatalyst.	November 28, 2023
New content: Custom blueprints and lifecycle management	Added new content for using custom blueprint and lifecycle management features in Amazon CodeCatalyst.	November 27, 2023
Updated content: Tutorial: Creating a project with the Modern three-tier web application blueprint	Updated the tutorial with fixes and troubleshooting information.	November 22, 2023
Updated content: Starting a workflow run automatically with triggers	Fixed a few examples and descriptions related to pull request triggers. Added a Trigger considerations when branching section.	November 22, 2023
New content: Sign in with SSO	Added information about signing in with Single Sign-On (SSO) and links to information about setting up and managing a CodeCatalyst space that supports identity federation. See Set up and sign in to CodeCatalyst and Sign in with SSO.	November 17, 2023

Updated content: Working with roles	Updated the documentation for the roles to include permissions for working with teams, VPC connections, single sign-on, and machine resources.	November 16, 2023
Updated content: Working with pull requests	Updated the documentation to reflect changes in how changes for a pull request are displayed.	November 16, 2023
Updated content: Quotas for CodeCatalyst	Updated the Quotas for CodeCatalyst topic with the *Maximum number of VPC connections for a space* quota.	November 16, 2023
New content: Managing teams for a space and for CodeCatalyst projects	Added information about using teams with spaces. See Allowing space access using teams and Allowing project access using teams.	November 16, 2023
New content: Managing machine resources for blueprints and workflows in a space	Added information about using machine resources with spaces. See Allowing space access for machine resources .	November 16, 2023
New content: Managing machine resources for blueprints and workflows in a CodeCatalyst project	Added information about using machine resources with CodeCatalyst projects. See Allowing project access for machine resources.	November 16, 2023
New content: Associating a VPC connection with an environment	Added documentation for associating a VPC connection with an environment, which can be used in a workflow.	November 16, 2023

New content: Associating a VPC connection to a Dev Environments	Added documentation for using Dev Environments with a VPC connection.	November 16, 2023
New content	Initial publication of the Amazon CodeCatalyst Administrator Guide.	November 16, 2023
New content: "AWS CDK deploy" action YAML definition	Added a new `CdkCliVersion` property to the "AWS CDK deploy" action YAML definition and the "AWS CDK bootstrap" action YAML definition.	November 14, 2023
Updated content: Working with roles	Updated the documentation for the roles to include permissions for working with branch rules.	November 13, 2023
Updated content: Troubleshooting problems with source repositories, workflows, and Dev Environments	Updated the troubleshooting topics to include information about working with branch rules.	November 13, 2023
Updated content: Build and test action YAML definition	Updated the documentation for the `Environment` property. It is now an optional field for build and test actions.	November 13, 2023
New content: Manage branch rules	Added documentation for branches to include information about viewing any rules for branches in a source repository, and creating and managing branch rules.	November 13, 2023

Updated content: Working with pull requests	Updated the documentation to reflect changes in how information about a pull request is displayed.	November 10, 2023
Updated content: Caching files between workflow runs	Updated the documentation to include file caching limitations.	November 10, 2023
Updated content: Tutorial: Deploy an application to Amazon EKS	Updated the documentation to mention the **EKS App Deployment** blueprint.	November 9, 2023
New content: Packages in CodeCatalyst	Added documentation for using packages in CodeCatalyst.	November 1, 2023
New and updated content: Working with roles	Updated the documentation for four new roles in CodeCatalyst: Power user, Limited access, Reviewer, and Read only.	November 1, 2023
Updated content: Exporting a GitHub output parameter so that other actions can use it	Updated the examples to use the `GITHUB_OUTPUT` environment file instead of the `set-output` command. Using environment files is GitHub's recommended method for setting output parameters.	October 24, 2023
New content: Starting a workflow run automatically with triggers	Added documentation for schedule triggers.	October 16, 2023

Updated content: "Deploy to Kubernetes cluster" action YAML definition	Added information about using the `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* role to the "Deploy to Kubernetes cluster" action YAML definition and Tutorial: Deploy an application to Amazon EKS topics.	September 22, 2023
Updated content: New role name and policy for the **CodeCatalystWorkflowDevelopmentRole-***`spaceName`* role	Updated the steps and role descriptions for the developer role name change to **CodeCatalystWorkflowDevelopmentRole-***`spaceName`* . The developer role now uses the `AdministratorAccess` AWS managed policy. See Understanding the **CodeCatalystWorkflowDevelopmentRole-***`spaceName`* service role and Creating your first space and development role (starting without an invitation).	September 20, 2023
Updated content: Configuring and using variables in a workflow	Introduced two new concepts: *user-defined variables* and *predefined variables*. These concepts should make the Configuring and using variables in a workflow section easier to read and understand.	September 19, 2023

Updated content: Working with commits	Updated the documentation to reflect the change in displayed information and provide details about viewing commits with multiple parents.	September 7, 2023
New content: Starting a workflow run automatically with triggers	Added the following example to the Starting a workflow run automatically with triggers topic: Example: A simple 'push to main' trigger	September 6, 2023
Updated content: Working with pull requests	Updated the documentation to reflect the change in display order for source branch and destination branch when creating a pull request.	August 30, 2023
New content: View and change the default branch	Added documentation for branches to include information about viewing and changing the default branch for a source repository.	August 30, 2023
Updated content: "Deploy to Kubernetes cluster" action YAML definition	Added a note about Helm and Kustomize to the Manifests property description in the "Deploy to Kubernetes cluster" action YAML definition.	August 15, 2023
New content: Managing issue attachments	Added documentation for working with and managing attachments on issues.	August 15, 2023

Updated content: Starting a workflow run automatically with triggers	Improved and expanded the documentation related to workflow triggers.	August 11, 2023
New content: Troubleshooting role permissions	Added information about updating the role permissions to run a workflow that requires access to Amazon CodeGuru. See Modern three-tier web application blueprint workflow **OnPullRequest** fails with permissions error for Amazon CodeGuru.	August 11, 2023
New content: How do I fix "Workflow is inactive" messages?	Added the following troubleshooting topic: How do I fix "Workflow is inactive" messages?	August 11, 2023
New content: Deploying an application to Amazon Elastic Kubernetes Service with a workflow	Added documentation for the **Deploy to Kubernetes cluster** action. For more information, see Deploying an application to Amazon Elastic Kubernetes Service with a workflow and Tutorial: Deploy an application to Amazon EKS.	July 27, 2023

Updates for how management events are logged for a CodeCatalyst space	Added information about how management events are logged for specific actions in a CodeCatalyst space with AWS CloudTrail. Added information about how all events in a space can be viewed with the list-event-logs command. See Monitoring events and API calls using logging.	July 20, 2023
Updated content: Starting a workflow run automatically with triggers	Updated the documentation to indicate that pull request triggers are now supported with GitHub source repositories. Previously, pull request triggers were only supported with CodeCatalyst source repositories.	July 14, 2023
Updated content: Quotas for workflows	Updated the Quotas for workflows topic with the *Maximum amount of time an action can run* quota.	June 27, 2023
Updated content: Workflow YAML definition	Fixed a formatting error in the `Compute` code block.	June 27, 2023
Updated content: Data protection	Updated the documentation to include additional information about data replication.	June 26, 2023
New content: Specifying the major, minor, or patch version of an action	Added a Specifying the major, minor, or patch version of an action topic.	June 21, 2023

Updated content: Deploying into AWS accounts and VPCs with CodeCatalyst environments	Clarified the Which actions support having their deployment information displayed in CodeCatalyst? section.	June 14, 2023
Updated content: Reorganized issues documentation	Reorganized most of the issues documentation to better align with the overall documentation set and user flows.	May 31, 2023
Updated content: Issues view switcher	Updated various user flows to align with the updated issue view switcher.	May 31, 2023
Updated content: Managing notifications	Updated the documentation for notifications to include information about configuring personal Slack notifications.	May 30, 2023
Updated content: Managing notifications	Updated the documentation for notifications to include information about configuring personal Slack notifications.	May 30, 2023

New content: CodeCatalyst trust model	Added a new topic with information about the trust model, which allows CodeCatalyst to assume the service role in the connected AWS account. Added a new section about the defined service principals for CodeCatalyst. See Understanding the CodeCatalyst trust model.	May 20, 2023
Updated content: Connecting a workflow to a source repository	Simplified the instructions in Referencing files in a source repository.	May 10, 2023
Updated content: Quotas for workflows	Updated the Quotas for workflows topic with the *Maximum length of an output variable value* quota.	May 10, 2023
New content: Sharing data between actions in a workflow using artifacts	Added two examples: Example: Referencing a file in a single artifact and Example: Referencing a file in an artifact when a WorkflowSource is present.	May 10, 2023
Updated content: Working with pull requests	Updated the documentation for pull requests to include information about configuring email preferences for pull request events.	April 21, 2023

Updated content: Managing notifications	Updated the documentation for notifications to include information about configuri ng email preferences for pull request events.	April 21, 2023
Updates to managed policies	Added the AWS managed policy: AmazonCodeCatalyst FullAccess, AWS managed policy: AmazonCodeCatalyst ReadOnlyAccess, and AWS managed policy: AmazonCod eCatalystSupportAccess managed policies. See CodeCatalyst updates to AWS managed policies.	April 20, 2023
New content: Removing a deployment target	Added a Removing a deployment target topic.	April 20, 2023
New content: Action types	Added a CodeCatalyst actions topic.	April 20, 2023
Updates for managing a user with the Space administrator role in a space	Added information about removing or changing the role for a user with the Space administrator role in a space. See Removing or changing the role for a user with the **Space administrator** role.	April 19, 2023
Updates for administering Dev Environments	Added information about administering Dev Environme nts as a Space administr ator. See Administering Dev Environments for a space.	April 19, 2023

Updated content: Finding and viewing issues	Reorganized the Finding and viewing issues topic and subtopics.	April 19, 2023
Updated content: Creating a project in CodeCatalyst with a linked GitLab project repository	Updated Creating a project to include GitLab integration inCreating a project with a linked third-party repository section.	April 19, 2023
Updated content: Configuring the compute and runtime environment Docker images for a workflow	Added support for Arm64 architecture on Amazon Linux 2.	April 19, 2023
New content: Moving issues within groups	Added documentation for moving issues within groups on the **Board** and **All issues** views.	April 19, 2023
Updated content: Quotas for workflows	Updated the Quotas for workflows topic with missing quotas, and updated the *Maximum total size of a single action's output variables* quota to 120 KB (from 2 KB).	April 18, 2023
New content: Viewing an action's source code	Added a Viewing an action's source code topic.	April 18, 2023
New content: Retrying test cases of a report	Added a Retrying test cases of a report topic.	April 11, 2023
New content: Stopping a workflow run	Added a Stopping a workflow run topic.	April 10, 2023

New content: Added sections for tagging resources for account connections between AWS and Amazon CodeCatalyst	Added information for tagging account connection resources and managing IAM policies for connection resources. See Using tags to control access to account connection resources and CodeCatalyst permissions reference.	April 6, 2023
New content: Action types	Added an Action types topic.	April 6, 2023
Updated content: "Deploy AWS CloudFormation stack" action YAML definition	Updated the `parameter-overrides` property description. It now supports JSON files.	April 5, 2023
New content: Creating a project in CodeCatalyst with a linked GitHub repository	Added a new section to Creating a project titled Creating a project with a linked third-party repository with instructions to create a project that links to your GitHub repository.	April 5, 2023
Updated content: Working with notifications	Updated the documentation for notifications to include information about configuring emails about project events.	March 31, 2023
New content	Initial publication of the Amazon CodeCatalyst Action Development Kit guide.	March 31, 2023
Updated content: Restructured the Spaces section in Amazon CodeCatalyst	Updated the Spaces section by removing landing pages and consolidating topics.	March 29, 2023

Updated content: Tutorial: Deploy an application to Amazon ECS	Changed Step 1: Set up an AWS user and AWS CloudShell to describe how to create a user in AWS IAM Identity Center instead of AWS Identity and Access Management. Creating IAM users is no longer recommend ed.	March 23, 2023
Updated content: Working with roles	Updated the documentation for the Space administrator, Project administrator, and Contributor roles to include permissions for linking issues to pull requests.	March 13, 2023
Updated content: Working with pull requests	Updated the documentation for pull requests to include information about linking issues to pull requests.	March 13, 2023
Updated content: Working with issues	Updated the documentation for issues to include informati on about linking issues to pull requests.	March 13, 2023
New content: Viewing the status and details of all runs in your project	Added a section that describes the new aggregated workflow run page.	March 8, 2023
New content: How do I fix "Workflow definition has *n* errors" errors?	Added a section on how to troubleshoot "The workflow definition has errors" errors.	March 7, 2023
Updated content: Creating a workflow	Updated the instructions to reflect new UI.	March 3, 2023

New content: Integrating universal-test-runner into a test action	Added a Integrating universal-test-runner into a test action topic.	March 3, 2023
Updated content: Build, test, and deploy with workflows in CodeCatalyst	Updated various sections to reflect new source repository, branch, and workflow name filters on the **Workflows** summary page.	March 2, 2023
New content: Tracking deployment status by commit	Added a section on viewing code quality and deployment status by commit.	February 27, 2023
New content: Variables produced by the source ("BranchName" and "CommidId")	Added a new `BranchName` predefined variable.	February 16, 2023
Updated content: Managing space members in Amazon CodeCatalyst	Updated information about changing member roles, inviting members, and removing members in two new tables based on the user's assigned role in CodeCatalyst.	February 15, 2023
Updated content: Added steps for PAT management in the Amazon CodeCatalyst console	Added steps for viewing, creating, and deleting PATs in the console.	February 15, 2023
Updated content: Specifying runtime environment Docker images	Added more tools to the **Default image tool versions** table.	January 10, 2023

Updated content: Sharing data between actions in a workflow using artifacts	Fixed an artifact path.	January 3, 2023
Updated content: "GitHub Actions" action YAML definition	Fixed the code snippet in the `Steps` section.	January 3, 2023
Updated content: Connecting a workflow to a source repository	Fixed a source path.	January 3, 2023
Updated content: Updating a pull request	Updated the documentation to include information about updating required or optional reviewers for a pull request.	December 23, 2022
New content: Caching files between workflow runs	Added a page for file caching in a workflow.	December 20, 2022
Updated content: Working with pull requests	Updated the documentation for pull requests to include information about notifications.	December 16, 2022
New content: "AWS CDK deploy" action YAML definition	Added a new `CdkRootPath` property.	December 16, 2022
New content: Sharing compute across actions	Added a Sharing compute across actions topic.	December 14, 2022
Updated content: Sharing data between actions in a workflow using artifacts	Fixed examples showing how to specify input artifacts.	December 13, 2022
New content: "GitHub Actions" action YAML definition	Added a dedicated reference page for the **GitHub Actions** action.	December 13, 2022

Updated content: Quotas for projects in CodeCatalyst	Updated the documentation with a maximum of 100 projects in a space.	December 2, 2022
New content	Initial publication of the Amazon CodeCatalyst User Guide.	December 1, 2022
New content: ???	Added a troubleshoot topic on possible issues users might come across when using the third-party extensions feature.	June 18, 2022

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.