Implementation Guide

Application Pattern Orchestrator on AWS

Application Pattern Orchestrator on AWS: Implementation Guide

Table of Contents

Solution to establish and manage an internal catalog of reusable, repeatable, well-architected, secure-by-design, and production-ready cloud infrastructure patterns

Publication date: *November 2022* (last update: August 2023)

Application Pattern Orchestrator (APO) is an AWS solution that helps customers in regulated industries such as Financial Services Industry (FSI), manufacturing, and healthcare to establish and manage an internal catalog of reusable, repeatable, well-architected, secure-by-design, and production-ready cloud infrastructure patterns for use by application development and engineering teams throughout their organizations.

Note

A pattern may be described by one or more attributes on its initial definition or as part of a subsequent update. Although such attributes can describe any characteristic of a pattern, such as hosting construct or technology stack, in the context of this solution, they are intended to inform governance, risk, and compliance characteristics.

This solution offers a set of integrated components that provide an end-to-end orchestration framework to allow decentralized contribution, implement automated compliance validation, centralize approval and publishing, and lifecycle notifications of an enterprise's internal application-driven cloud infrastructure patterns.

Using this solution, application and technology teams can use a self-service web user interface (UI) to submit their application patterns as CloudFormation or CDK for automatic validation, manual review, approval and publishing to Service Catalog as Service Catalog products (for CloudFormation-based patterns) and to AWS CodeArtifact as software packages (for CDK-based patterns).

- Automatic validation provides feedback within minutes, while the solution's manual review and approval workflows provide asynchronous collaboration between application teams and centralized architecture and security teams, via familiar tools such as Git, where rework or further iterations are needed. For more information, refer to the

- The web UI provides a notification capability to subscribers to alert to the availability of newly published patterns or versions per category, portfolio, etc.
- You can use a browsable and searchable catalogue of published patterns for consumption, with metadata and supporting assets, for example, architecture diagrams and Frequently Asked Questions (FAQs) to locate useful patterns.

This implementation guide describes architectural considerations and configuration steps for deploying the Application Pattern Orchestrator in the AWS cloud.

Use this navigation table to quickly find answers to these questions:

If you want to . . .	Read . . .
Know the cost for running this solution. The estimated cost for running this solution in the US-East (N.Virginia) Region is USD $194.32 per month.	Cost
Understand the security considerations for this solution.	Security
Know which AWS Regions are supported for this solution.	Supported AWS Regions
View or download the AWS CloudForm ation template included in this solution to automatically deploy the infrastructure resources (the “stack”) for this solution.	AWS CloudFormation template

This guide is intended for deployment in an enterprise by **IT infrastructure and security architects, security administrators, developers, and DevSecOps professionals** who have practical experience with the AWS Cloud.

Features and benefits

The Application Pattern Orchestrator on AWS solution provides the following features:

Self-service, and low-touch developer-friendly experience

- Create decentralized processes for distributed engineering teams to iterate on pattern feedback in an asynchronous manner.

Drive consistency and standardization of controls across your organizations

- Embed attributes to be automatically inherited by new applications using underlying patterns that incorporate repeatable guardrails.
- Automatically validate pattern security, architecture, and compliance against organization-specific policy-as-code.

Central discovery of approved application patterns

- Allow engineering teams to browse and search for patterns through a centrally accessible user interface built for application developers.

For more information, refer to the Application Pattern Orchestrator on AWS solution page.

Use cases

The Application Pattern Orchestrator on AWS solution allows engineering teams in an organization to establish and manage an internal catalog of reusable, repeatable, well-architected, secure-by-design, and production-ready cloud infrastructure patterns.

This enables organizations in shifting governance to the left through the use of patterns and incorporate guardrails for new applications at scale. The solution makes an engineer's job easier because it automatically validates a pattern's security, architecture, and compliance against industry best practices.

Concepts and definitions

This section describes key concepts and defines terminology specific to this solution:

Attribute

Attributes help describe any characteristic of a pattern, such as a hosting construct or technology stack. In the context of this solution, they are intended to inform governance, risk, and compliance characteristics.

Pattern

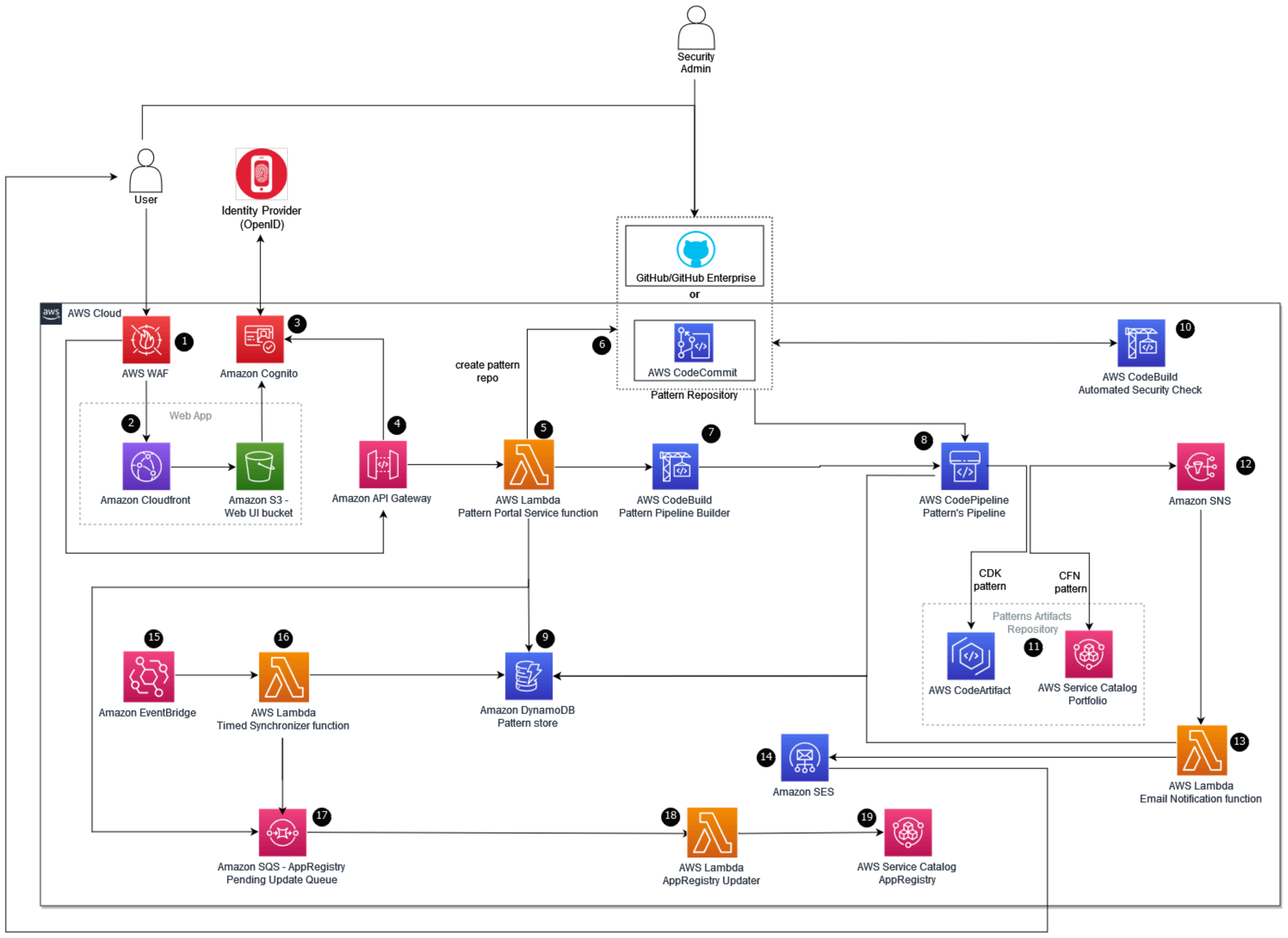
A *pattern* may be described by one or more attributes on its initial definition or as part of a subsequent update.

For a general reference of AWS terms, see the AWS glossary in the AWS General Reference.

Architecture overview

Deploying this solution with the default parameters deploys the following components in your AWS account.

Architecture diagram



Security Admin
User
Identity Provider (OpenID)
GitHub/GitHub Enterprise
or
AWS CodeCommit
Pattern Repository
AWS Cloud
AWS WAF
1
Amazon Cognito
3
Web App
2
Amazon Cloudfront
Amazon S3 - Web UI bucket
4
Amazon API Gateway
create pattern repo
6
5
AWS Lambda Pattern Portal Service function
7
AWS CodeBuild Pattern Pipeline Builder
10
AWS CodeBuild Automated Security Check
8
AWS CodePipeline Pattern's Pipeline
12
Amazon SNS
CDK pattern
CFN pattern
Patterns Artifacts Repository
AWS CodeArtifact
11
AWS Service Catalog Portfolio
15
Amazon EventBridge
16
AWS Lambda Timed Synchronizer function
9
Amazon DynamoDB Pattern store
14
Amazon SES
13
AWS Lambda Email Notification function
17
Amazon SQS - AppRegistry Pending Update Queue
18
AWS Lambda AppRegistry Updater
19
AWS Service Catalog AppRegistry

Application Pattern Orchestrator on AWS architecture diagram

Note

AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs

The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows. The numbers below match the number designated in the architecture diagram.

1. AWS WAF to protect the web UI and Amazon API Gateway endpoints against common web exploits and bots that may affect availability, compromise security, or consume excessive resources.
2. An Amazon CloudFront distribution to serve the optional UI. Amazon CloudFront delivers low latency, high performance, and secure static web hosting. An Amazon Simple Storage Service (Amazon S3) web UI bucket hosts the static web application artifacts.
3. Amazon Cognito to provide authentication mechanism for both the static content hosted in S3 bucket for the web UI and API Gateway endpoints. Amazon Cognito also manages federating and storing users from external identity providers (IDPs).
4. Amazon API Gateway to expose a set of RESTful APIs. API Gateway processes HTTP requests issued by the users to manage the lifecycle of application patterns and their attributes.
5. A `Pattern Portal` AWS Lambda function to process the validated requests from the API Gateway. This Lambda function encapsulates the solution's business logic, receiving REST requests from the user via the API Gateway, validating them and storing these requests, and retrieving data to and from the database.
6. AWS CodeCommit to store the pattern's source code.

Note

To configure GitHub or GitHub Enterprise as your pattern's source code repository instead, deploy the solution using AWS CDK by following the instructions in the solution README.

7. A pattern pipeline builder AWS CodeBuild to provision the CI/CD pipeline for the patterns.
8. AWS CodePipeline to provide the CI/CD pipeline to publish a pattern to its target pattern store.
9. Amazon DynamoDB to store and retrieve pattern's metadata, publish data and attributes.
10. Automated security check AWS CodeBuild to perform security scan on the pattern's CloudFormation template which gets initiated automatically when the pattern's developer raises a pull request. On completion of the security check, the results are published on the pull request page for the security admin to review. Once approved and the pattern's code changes merged into the main branch of the pattern's code repository, the CI/CD pipeline is automatically initiated to publish the pattern.

11. A Pattern's artifacts store to store the published artifacts to Service Catalog (for CloudFormation-based patterns) and to AWS CodeArtifact (for CDK-based patterns).
12. Amazon SNS topic to receive the published pattern data from the pattern's publishing pipeline to start the email notification mechanism.
13. An `email notification` AWS Lambda function to receive the pattern's published data from Amazon SNS topic, get the list of subscribers from AWS DynamoDB and invoke Amazon SES to send email notification about the pattern's publishing to the subscriber list.
14. Amazon SES to send email notification to the pattern's subscriber list whenever a new pattern's version is published.
15. Amazon EventBridge rule to periodically initiate the pattern attribute sync process.
16. Amazon EventBridge initiates a `Timed Synchronizer` AWS Lambda function to pull the pattern attributes from Amazon DynamoDB and push them to the Amazon SQS queue for performing the sync attribute operation.
17. Amazon SQS queue to receive the attributes data and send it to the `AppRegistry Updater` AWS Lambda function to update the attribute groups in Service Catalog AppRegistry.
18. An `AppRegistry Updater` AWS Lambda function to sync the pattern attributes with Service Catalog AppRegistry.
19. Service Catalog AppRegistry to store the attributes data in the form of attribute groups synced from Amazon DynamoDB.

AWS Well-Architected design considerations

This solution was designed with best practices from the AWS Well-Architected Framework which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework were applied when building this solution.

Operational excellence

This section describes how we architected this solution using the principles and best practices of the operational excellence pillar.

- All Lambda functions send logging output to Amazon CloudWatch.

- Access to S3 buckets and CloudFront distribution is logged into an access log bucket.
- A comprehensive CloudWatch dashboard is provided to monitor the operational status of underlying services.

Security

This section describes how we architected this solution using the principles and best practices of the security pillar.

- All inter-service communications use AWS Identity and Access Management (IAM) roles.
- All IAM roles used by the solution follow the least-privilege access principle. They only contain the minimum permissions required so that the service can function properly.
- All Lambda functions are created inside a VPC.
- The solution uses Amazon Cognito for user authentication and authorization. Additionally, it also supports federated user authentication and authorization from a different identity provider.
- All data stored in Amazon DynamoDB, AWS Service Catalog AppRegistry, AWS CodeArtifact, and Amazon S3 buckets have encryption at REST.
- AWS WAF is applied on both CloudFront Distribution as well as APIGateway APIs to mitigate any potential attacks.

Reliability

This section describes how we architected this solution using the principles and best practices of the reliability pillar.

- The solution uses AWS serverless services wherever possible to ensure high availability and recovery from service failure.
- All compute processing uses Lambda functions.
- Data is stored in Amazon S3 and DynamoDB tables, so it persists in multiple Availability Zones by default.

Performance efficiency

This section describes how we architected this solution using the principles and best practices of the performance efficiency pillar.

- The solution uses serverless compute and data resources throughout the architecture.
- You can launch the solution in any Region that supports AWS services used in this solution.
- The solution is developed with AWS CDK and managed with AWS CloudFormation stacks. By implementing a complete Infrastructure-as-Code (IAC) approach, it allows easy upgrading and resource management.
- The solution leverages as many AWS managed services as possible. For more information, refer to the AWS services used in this solution section.

Cost optimization

This section describes how we architected this solution using the principles and best practices of the cost optimization pillar.

- The solution only uses Lambda functions for compute needs and only charges for what is used.
- Full serverless architecture and automatic scalability to scale out when demand is high and scale in when demand is low.

Read the Cost Optimization Pillar whitepaper

Sustainability

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

- Serverless resources are used for compute and data storage.
- Most data storage is maintained within Amazon DynamoDB, Amazon S3 buckets, AWS Service Catalog AppRegistry, and AWS CodeArtifact that you can remove easily.

Read the Sustainability Pillar whitepaper

Architecture details

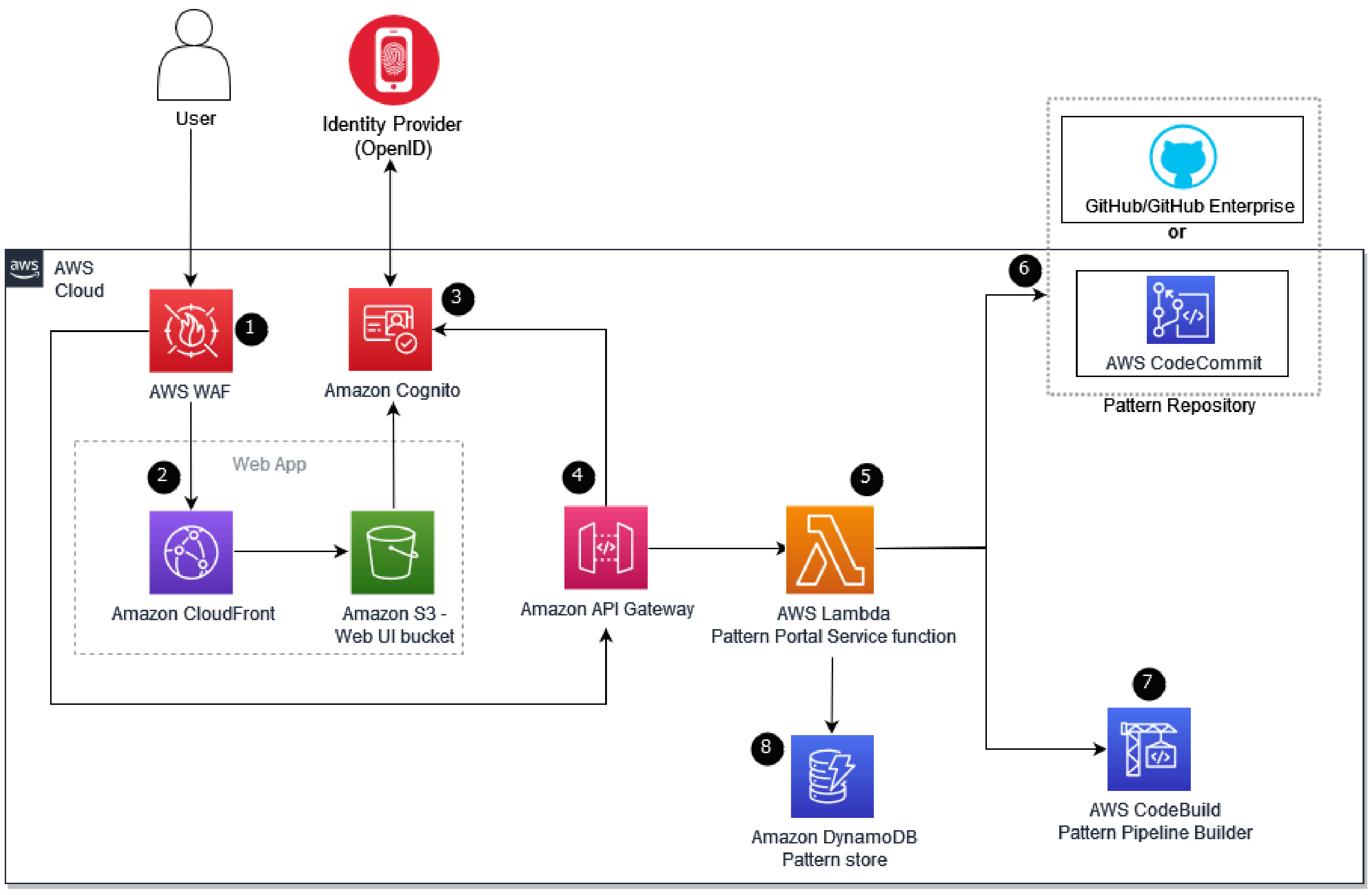
The solution deploys the following components that work together to provide pattern governance and catalog functionality:

- **Application pattern portal:** A web UI that allows the user to create new patterns, its attributes and provides a storefront to browse and search for patterns.
- **Application patterns pipeline builder:** Provisions governed pipelines for evaluating and publishing application patterns upon a pattern's submission request. Pipelines are provisioned per application pattern.
- **Policy evaluation and security review microservice:** Provides an interface to plug in a policy-as-code implementation such as CFN Nag.
- **Provisioned publishing pipeline(s):** AWS CodePipeline based pipeline responsible for publishing a pattern to Service Catalog (for CloudFormation-based patterns) and to AWS CodeArtifact (for CDK-based patterns).
- **Application pattern subscriber email notification:** An email notification mechanism that allows users to subscribe to the patterns they are interested in and receive notification whenever a new version of the patterns is published.

The solution provides a self-service mechanism for application teams to submit their application foundation constructs for automatic validation and receive feedback within minutes. It provides capabilities to create new Git repositories for distributed maintenance of patterns. Patterns are first validated against a third-party linting tool called cfn_nag. The security report generated by `cfn_nag` is reviewed by the Security team. Once approved and code merged, the pattern is automatically published via CodePipeline to Service Catalog (for CloudFormation-based patterns) and to AWS CodeArtifact (for CDK-based patterns).

Application pattern portal

The following diagram provides an overview of the Application pattern portal.


User
Identity Provider (OpenID)
GitHub/GitHub Enterprise
or
AWS CodeCommit
Pattern Repository
AWS Cloud
1
AWS WAF
3
Amazon Cognito
6
Web App
2
Amazon CloudFront
Amazon S3 - Web UI bucket
4
Amazon API Gateway
5
AWS Lambda Pattern Portal Service function
7
AWS CodeBuild Pattern Pipeline Builder
8
Amazon DynamoDB Pattern store

Application Pattern Portal overview

1. AWS WAF to protect the web UI and API Gateway endpoints against common web exploits and bots that may affect availability, compromise security, or consume excessive resources.
2. An Amazon CloudFront distribution to serve the optional UI. Amazon CloudFront delivers low latency, high performance, and secure static web hosting. An Amazon Simple Storage Service (Amazon S3) web UI bucket hosts the static web application artifacts.
3. Amazon Cognito to provide authentication mechanism for both the static content hosted in S3 bucket for the web UI and API Gateway endpoints. Amazon Cognito also manages federating and storing users from external identity providers (IDPs).
4. Amazon API Gateway to expose a set of RESTful APIs. API Gateway processes HTTP requests issued by the users to manage the lifecycle of application patterns and their attributes.
5. A `Pattern Portal` AWS Lambda function to process the validated requests from API Gateway. This Lambda function encapsulates the solution's business logic, receiving REST requests from

the user via API Gateway, validating them and storing, and retrieving data to and from the database.

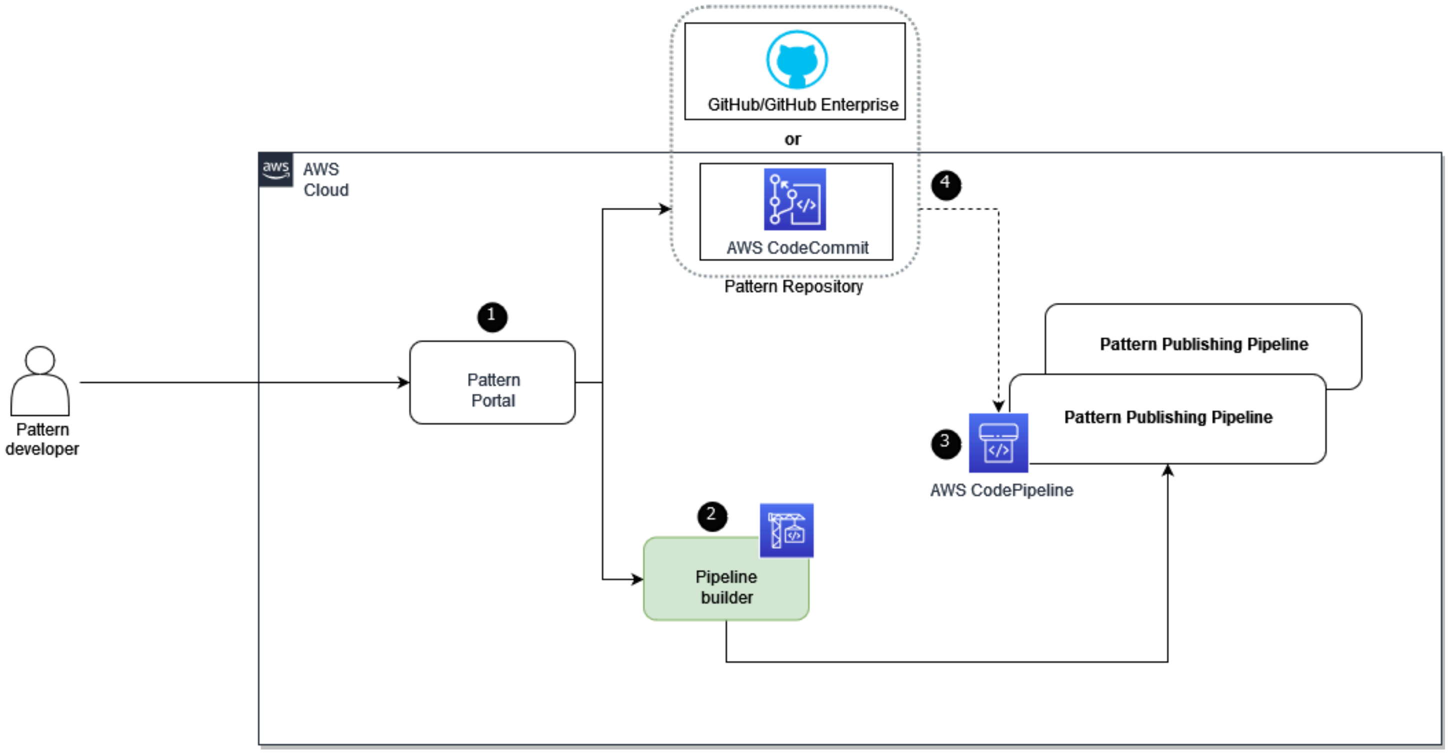
6. AWS CodeCommit to store the pattern's source code.

> **Note**
>
> To configure GitHub or GitHub Enterprise as your pattern's source code repository instead, deploy the solution using AWS CDK by following the instructions in the solution README.

7. A pattern pipeline builder AWS CodeBuild to provision the CI/CD pipeline for the patterns.
8. Amazon DynamoDB to store and retrieve the pattern's metadata, publish data and attributes.

Patterns pipeline builder

The following diagram represents the component overview of Patterns pipeline builder.



Patterns pipeline builder overview

1. After the patterns developer submits a create pattern request, a new code repository is created in AWS CodeCommit for the pattern.

> **Note**
>
> To configure GitHub or GitHub Enterprise as your pattern’s source code repository instead, deploy the solution using AWS CDK by following the instructions in the solution README.

2. The Pipeline builder initiates the provisioning of CodePipeline for publishing the pattern.
3. The publishing pipeline uses AWS CodeCommit to store the pattern’s source code.

> **Note**
>
> To configure GitHub or GitHub Enterprise as your pattern’s source code repository instead, deploy the solution using AWS CDK by following the instructions in the solution README.

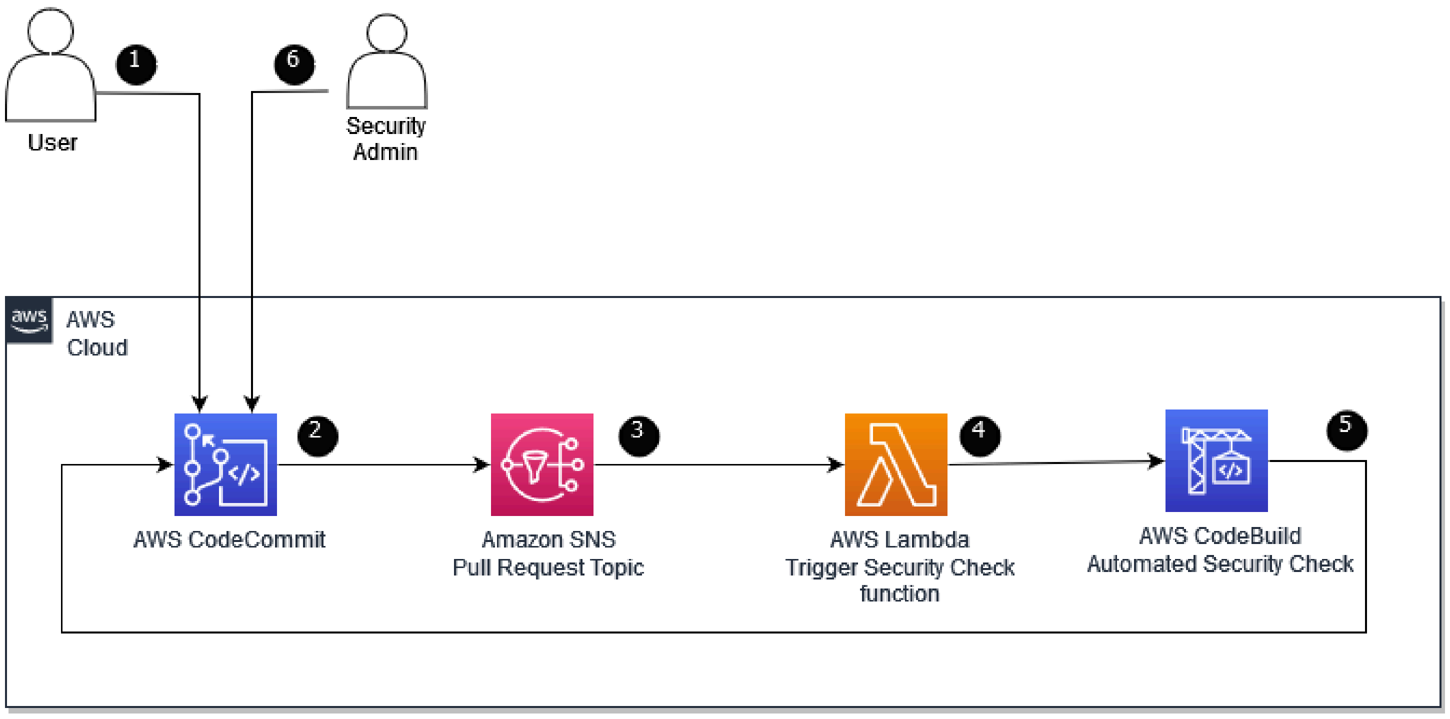
4. On every new commit in the main branch of the pattern’s source code repository, the pattern’s publishing pipeline is launched automatically and publishes the pattern’s new version.

Policy evaluation and security review microservice

By default, the solution uses AWS CodeCommit to store the pattern’s source code.

> **Note**
>
> To configure GitHub or GitHub Enterprise as your pattern’s source code repository instead, deploy the solution using AWS CDK by following the instructions in the solution README.

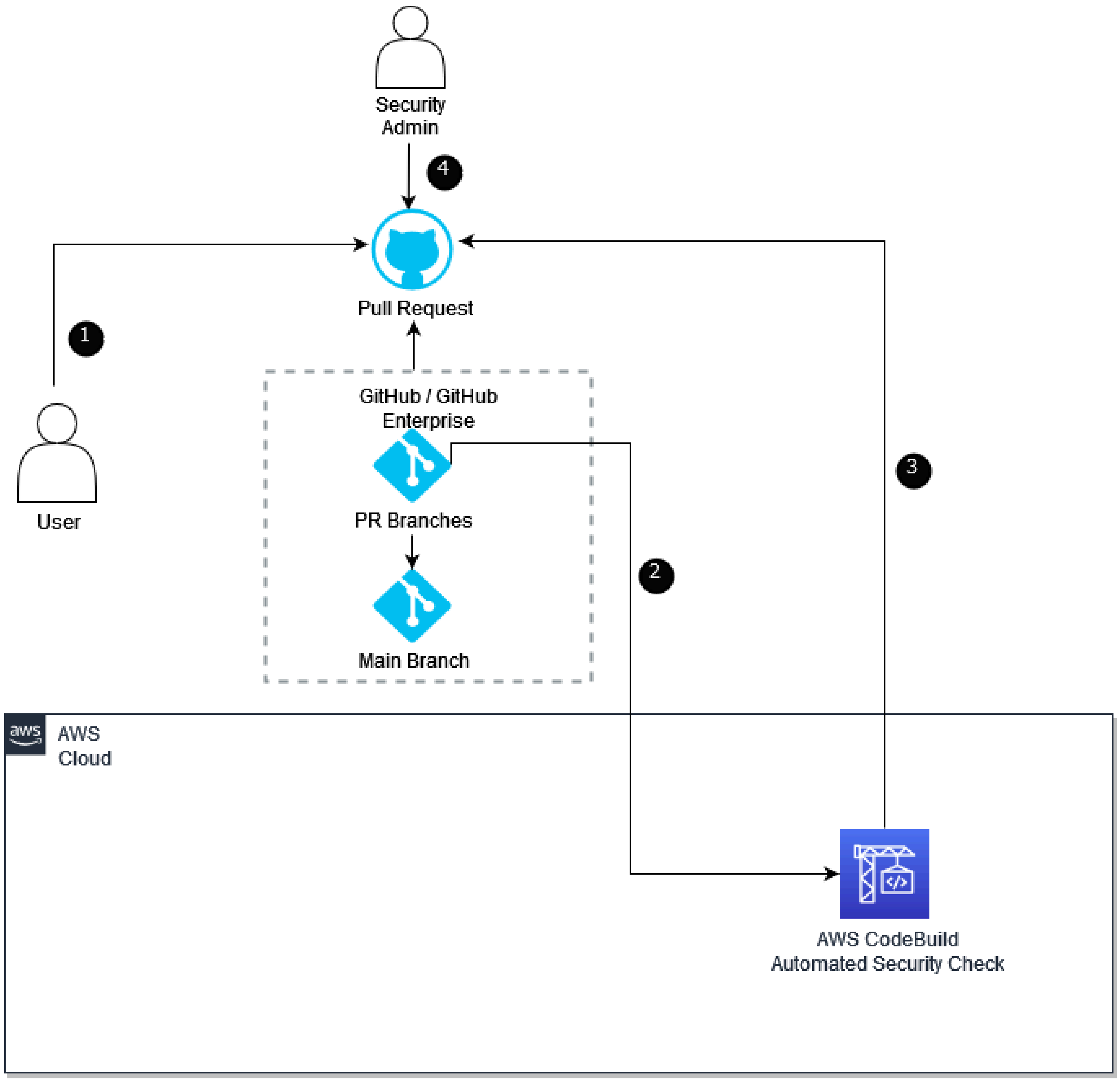
The following diagram describes the policy evaluation and security review microservice based on AWS CodeCommit as the pattern's source code repository.


User
Security Admin
1
6
AWS Cloud
2
3
4
5
AWS CodeCommit
Amazon SNS Pull Request Topic
AWS Lambda Trigger Security Check function
AWS CodeBuild Automated Security Check

Policy evaluation and security review microservice using AWS CodeCommit

1. The Pattern developer creates a pattern and raises a `pull request` to the master branch.
2. The `pull request` event is sent to an Amazon SNS topic.
3. On receiving the `pull request` event, the Amazon SNS topic initiates an AWS Lambda function with the event data page.
4. The AWS Lambda function parses the `pull request` event data and invokes the AWS CodeBuild Automated Security Check project which runs the automated security check (`cfn_nag`) on the Pull Request (PR) branch where the pattern was updated.
5. The AWS CodeBuild Automated security check project sends the security findings back to the `pull request` page as comments.
6. Security Admin reviews the results of the evaluated pattern reported by security review microservice in the `pull request` and approves the `pull request`.

The following diagram describes the policy evaluation and security review microservice based on GitHub/GitHub Enterprise as the pattern's source code repository.



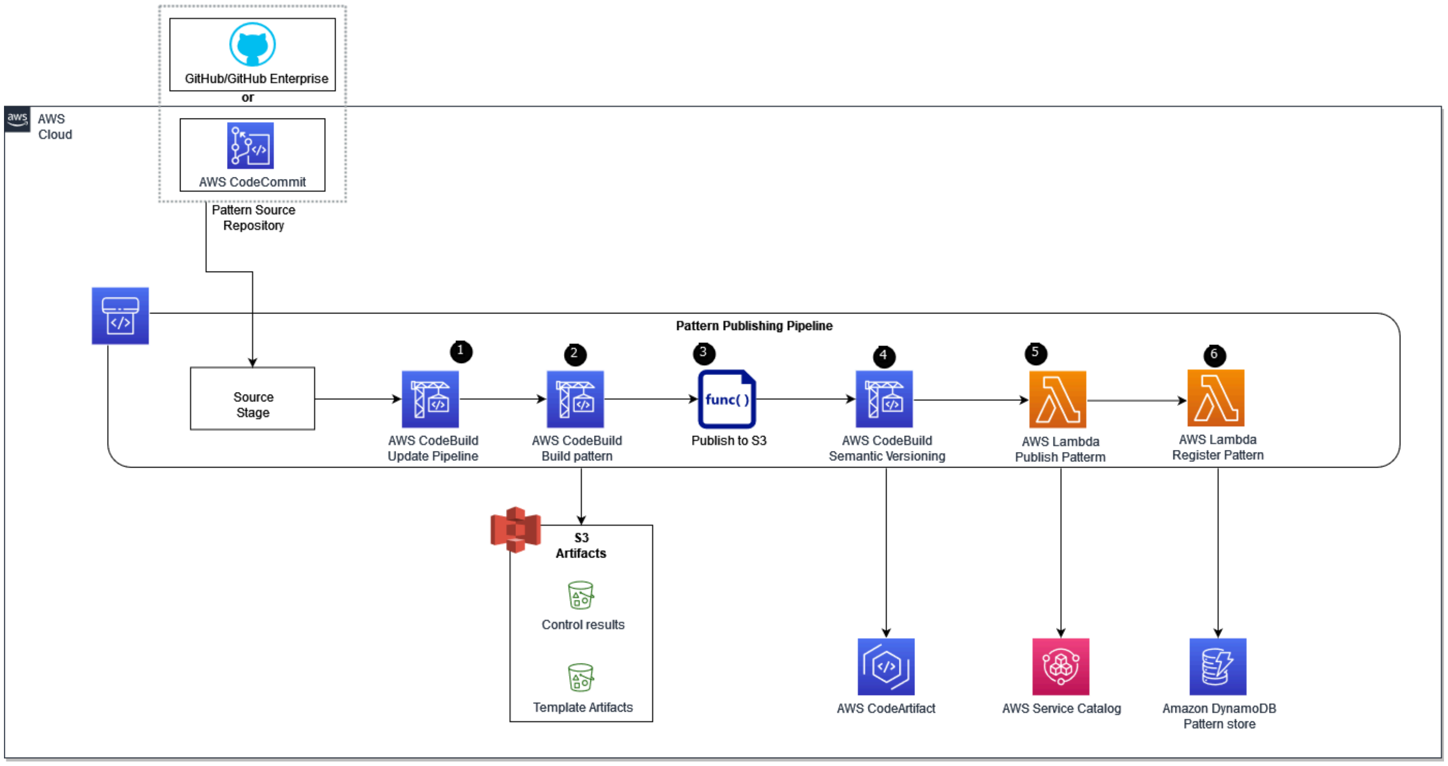
Policy evaluation and security review microservice using GitHub/GitHub Enterprise as pattern's source repository

1. The Pattern developer creates a pattern and raises a `pull request` to the main branch.
2. The `pull request` initiates a webhook request to AWS CodeBuild project which runs the automated security check (`cfn_nag`) on the Pull Request (PR) branch where the pattern was updated.

3. The automated security check sends the security findings back to the `pull request` page.
4. Security Admin reviews the results of the evaluated pattern reported by security review microservice in the `pull request` and approves the `pull request`.

Provisioned publishing pipeline

The following diagram describes the provisioned publishing pipeline and catalog manager microservice.


GitHub/GitHub Enterprise
or
AWS CodeCommit
Pattern Source Repository
AWS Cloud
Pattern Publishing Pipeline
Source Stage
1
AWS CodeBuild Update Pipeline
2
AWS CodeBuild Build pattern
3
func( )
Publish to S3
4
AWS CodeBuild Semantic Versioning
5
AWS Lambda Publish Patterm
6
AWS Lambda Register Pattern
S3 Artifacts
Control results
Template Artifacts
AWS CodeArtifact
AWS Service Catalog
Amazon DynamoDB Pattern store

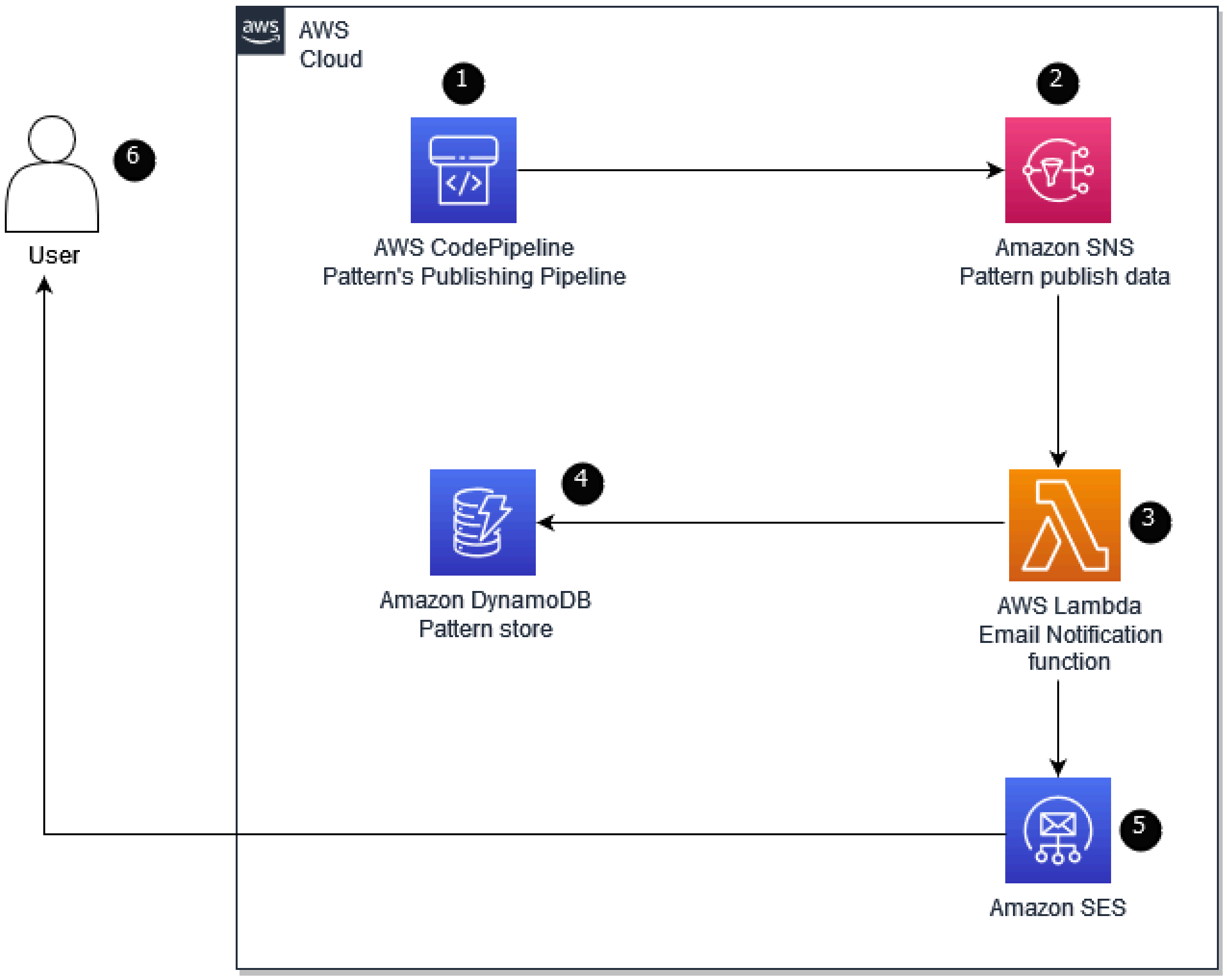
Provisioned publishing pipeline overview

1. When the pattern is merged into the main branch, it initiates the provisioned publishing pipeline. The `Update Pipeline` stage updates the pipeline itself.
2. The build stage builds the pattern code and runs the security evaluation (`cfn_nag`) on the pattern code.
3. Security evaluation results are stored in an S3 bucket along with the template artifacts.
4. Semantic versioning is leveraged to increment/generate version of application pattern based on git commit history.
5. The pattern gets published as Service Catalog product (for CloudFormation-based patterns) and to AWS CodeArtifact (for CDK-based patterns).

6. The published data is inserted in a DynamoDB table.

Application pattern subscriber email notification

The following diagram describes the pattern's subscriber email notification component.



Application pattern subscriber email notification

1. After the application pattern's publishing pipeline publishes the pattern, it sends the publishing details to Amazon SNS topic.
2. The Amazon SNS topic forwards the pattern's publishing data to AWS Lambda email notification function for further processing.
3. The AWS Lambda email notification function pulls the subscribers list for the published pattern from Amazon DynamoDB.
4. Amazon DynamoDB stores the pattern to subscriber email mapping data.

5. Once the AWS Lambda email notification service has the subscriber email list, it invokes Amazon SES to send emails to the pattern subscribers.
6. Users who subscribed to the pattern receive the email notification about the published pattern with details of the new version.

AWS services in this solution

AWS service	Description
Amazon CloudFront	**Core**. Distribution to serve the optional UI.
Amazon S3	**Core**. Web UI bucket to host the static web application artifacts.
AWS Lambda	**Core**. • Pattern Portal: Processes the validated requests from the API Gateway. • Email notification: Receives the pattern's published data from Amazon SNS topic, gets the list of subscribers from AWS DynamoDB, and invokes Amazon SES. • Timed Synchronizer: Pulls the pattern attributes from Amazon DynamoDB and pushes them to Amazon SQS. • AppRegistry Updater: Syncs the pattern attributes with Service Catalog AppRegistry.
AWS CodeCommit	**Core**. Stores the pattern's source code, by default.
AWS CodeBuild	**Core**. • Provisions the CI/CD pipeline for the patterns.

AWS service	Description
	• Performs security scan on the pattern's CloudFormation template.
Amazon DynamoDB	**Core**. Stores and retrieves the pattern's metadata, publishes data and attributes.
AWS CodePipeline	**Core**. Provides the CI/CD pipeline to publish a pattern to its target pattern store.
Service Catalog	**Core**. Stores the published artifacts for CloudFormation-based patterns.
AWS CodeArtifact	**Core**. Stores the published artifacts for CDK-based patterns.
Service Catalog AppRegistry	**Core**. Stores the attributes data in the form of attribute groups synced from Amazon DynamoDB.
AWS WAF	**Supporting**. Protects the web UI against common web exploits and bots that may affect availability, compromise security, or consume excessive resources.
Amazon Cognito	**Supporting**. Provides the authentication mechanism for both the static content hosted in an S3 bucket for the web UI and API Gateway endpoints.
API Gateway	**Supporting**. Protects the API endpoints against common web exploits and bots that may affect availability, compromise security, or consume excessive resources.

AWS service	Description
Amazon SNS	**Supporting**. Topic to receive the published pattern data from the pattern's publishin g pipeline to start the email notification mechanism.
Amazon SES	**Supporting**. Sends email notification about the pattern's publishing to the subscriber list whenever a new pattern's version is published.
Amazon EventBridge	**Supporting**. Rule to periodically initiate the pattern attribute sync process.
Amazon SQS	**Supporting**. Queue for performing the sync attribute operation.

Plan your deployment

This section describes the cost, security, Region, and design considerations for planning your deployment.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of the most recent revision, the cost for running this solution with the default settings, considering a deployment of 50 application patterns, in the US East (N.Virginia) Region is approximately USD **$194.32 a month**.

Cost table

Service	Monthly usage estimate	Total monthly cost (USD)
Amazon CloudFront	Data transfer out to internet: first 1,024 GB/month are free.	$0
Amazon Cognito	The Cognito User Pool feature has a free tier of 50,000 MAUs for users who sign in directly to Cognito User Pools and 50 MAUs for users federated through SAML 2.0 based identity providers. Percent of monthly users who sign in through SAML or OIDC federation: 10%	$0
AWS CodeCommit	First 5 active users are free. For 50 patterns there will be 50 IAM roles correspon ding to Security check AWS CodeBuild projects.	$45.00

Service	Monthly usage estimate	Total monthly cost (USD)
Amazon EventBridge	Event rules are free.	$0
AWS Secrets Manager	Number of secrets (1), Average duration of each secret (30 days), Number of API calls (100 per month)	$0.40
Amazon API Gateway	HTTP API requests units (millions), Average size of each request (34 KB), REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), Average message size (32 KB), Requests (1 per month)	$1.29
AWS Key Management Service	Number of customer managed Customer Master Keys (CMK) (59), Number of symmetric requests (2000000)	$65.00
AWS Lambda	Architecture (x86), Amount of ephemeral storage allocated (512 MB), Number of requests (4000 per month)	$0
AWS CodeBuild	Number of builds in a month (1000), Average build duration (minutes) (5), Operating system (Linux), Compute instance type (general1.small)	$25.00

Service	Monthly usage estimate	Total monthly cost (USD)
AWS CodePipeline	Number of active pipelines used per account per month (50)	$49.00
Amazon DynamoDB	Free 25 GB of storage and up to 200 million read/writ e requests per month, Table class (Standard), Average item size (all attributes) (1 KB)	$0
Service Catalog	Create one portfolio containin g 50 products published 10 times a day	$0
Amazon S3	S3 Standard storage (2 GB per month)	$0.11
AWS CodeArtifact	First 2GB of storage and first 100,000 requests of CodeArtifact usage for free every month.	$0
Amazon CloudWatch	Number of Metrics (includes detailed and custom metrics) (10), GetMetricData: Number of metrics requested (10), GetMetricWidgetImage: Number of metrics requested (10), Number of other API requests (10), Number of Dashboards (1), Standard Logs: Data Ingested (1 GB), Logs Delivered to CloudWatch Logs: Data Ingested (1 GB)	$4.35

Service	Monthly usage estimate	Total monthly cost (USD)
Amazon Simple Notification Service	Requests (1 million per month)	$0
Amazon Simple Queue Service	Data transfer cost (0), Standard queue requests (1 million per month), FIFO queue requests (1 million per month)	$0
Amazon Simple Email Service	For 50 application patterns published 2 times a day and sending email notifications to 100 pattern subscribers, the total emails sent will be approximately 10,000 a month.	$3.80
Total monthly cost		**USD $194.32**

You can use the AWS Pricing Calculator for estimating costs based on your usage.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, refer to the AWS Cloud Security page.

IAM roles

AWS Identity and Access Management (IAM) roles allow customers to assign granular access policies and permissions to services and users in the AWS Cloud. This solution creates IAM roles that grant the solution's automated functions access to perform remediation actions within a narrow scope set of permissions specific to each remediation.

Amazon S3 bucket configuration and policy

By default, all Amazon S3 buckets for the solution have the following configuration:

- Blocked all public access
- Versioning enabled
- Access log enabled
- Encryption at rest by an AWS KMS customer managed key

Additionally, the Amazon S3 buckets are also configured with a default buckets policy that deny all non-HTTPS requests to ensure data in transit encryption.

AWS Key Management Service (AWS KMS) keys

The Application Pattern Orchestrator on AWS solution allows you to provide your own AWS KMS keys to encrypt stored data. We recommend referring to the security best practices for AWS Key Management Service to enhance the protection of your encryption keys.

Amazon CloudFront

This solution deploys a web application hosted in an Amazon S3 bucket. To help reduce latency and improve security, this solution includes an Amazon CloudFront distribution with an Origin Access Identity (OAI), which is a CloudFront user that provides public access to the solution's website bucket contents. For more information, refer to Restricting access to an Amazon S3 origin section in the *Amazon CloudFront Developer Guide*.

Network configuration

The Application Pattern Orchestrator on AWS solution is deployed in Amazon VPC, with the Lambda functions in a private subnet. All traffic in and out of the isolated subnet is controlled by security groups.

User authorization

By default, the solution creates two user groups in the Amazon Cognito user pool for user authorization:

- `SYSTEM_ADMIN`: This user group has permissions to access all pages in the web UI. By default, any user created by the solution is automatically added to this group when the solution is deployed.
- `PATTERN_PUBLISHER`: This group has permissions to create, update, and view patterns. The group also allows you to view pattern attributes.

Note

To update or delete pattern attributes, you must be in the `SYSTEM_ADMIN` group.

Federating solution user groups through an Identity provider (IdP)

You can federate the solution user groups using a third-party identity provider via OpenID Connect (OIDC). To configure this:

1. Deploy the solution using AWS CDK by following the instructions in the solution README.
2. In your IdP settings, add a claim type group and map the roles that will relate to the `SYSTEM_ADMIN` and `PATTERN_PUBLISHER` roles in Amazon Cognito user pool. In absence of this mapping, a federated user would only have read-only access to the solution web UI.

Data protection

All data committed to Application Pattern Orchestrator on AWS is encrypted at rest; this includes data stored in:

- Amazon S3
- Amazon DynamoDB
- AWS CodeArtifact
- Service Catalog
- Amazon SQS

Communication between the solution’s different components is over HTTPS to ensure data encryption in transit.

Design considerations

Regional deployment

This solution uses the Amazon API Gateway, Amazon Cognito, AWS Secrets Manager, AWS Lambda, Amazon EventBridge, AWS Key Management Service, AWS CodeBuild, AWS CodePipeline, Amazon DynamoDB, Service Catalog, AWS CodeArtifact, Amazon CloudWatch, Amazon SQS, and Amazon SNS, which are currently available in specific AWS Regions only. We recommend you launch this solution in an AWS Region where these AWS services are available. For the most current service availability by Region, refer to the AWS-Regional Services List**.**

Solution pattern limit

When you deploy this solution with a newly provisioned AWS account, it will allow you to only create a maximum of 60 patterns based on the default IAM roles limit in your AWS account.

Note

If you need to increase the solution pattern limit, you will need to request and get approved for an IAM roles increase. For a maximum hard limit of 5000 IAM roles, you can create up to approximately 300 patterns.

For more information, refer to the IAM and AWS STS quotas, name requirements, and character limits.

Supported AWS Regions

The Application Pattern Orchestrator on AWS solution can be deployed in the following AWS Regions in accordance with the regional availability of its constituent services:

Region ID	Region Name
us-east-1	US East (N. Virginia)
us-east-2	US East (Ohio)
us-west-2	US West (Oregon)

Region ID	Region Name
ap-south-1	Asia Pacific (Mumbai)
ap-northeast-1	Asia Pacific (Tokyo)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)
eu-west-1	Europe (Ireland)
eu-west-2	Europe (London)
eu-west-3	Europe (Paris)
eu-central-1	Europe (Frankfurt)
eu-north-1	Europe (Stockholm)

Deploy the solution

Before you launch the Application Pattern Orchestrator on AWS solution, review the cost, architecture, network security, and other considerations discussed earlier in this guide. Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Time to deploy: Approximately 15 mins

This solution uses AWS CloudFormation templates and stacks to automate its deployment. The CloudFormation templates describe the AWS resources included in this solution and their properties. The CloudFormation stacks provisions the resources that are described in the templates.

Note

The automated deployment deploys the solution with the default configuration settings, and can be used for evaluation and production purposes. However, to fine tune the advanced settings for better performance or customize the solution to your specific environment, we recommended downloading the source code from the GitHub repository and building and deploying the solution with AWS CDK.

AWS CloudFormation template

You can download the CloudFormation template for this solution before deploying it.

View template

ApoStack.template: Use this template to launch the solution and all associated components. The default configuration deploys Application Pattern Orchestrator on AWS in the AWS Cloud, but you can customize the template to meet your specific needs.

Note

AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

This AWS CloudFormation template deploys Application Pattern Orchestrator on AWS in the AWS Cloud. You must meet the following prerequisites before launching the stack:

Note

If you have previously deployed this solution, see Update the solution for update instructions.

Prerequisites

AWS account

- **A CDK bootstrapped AWS account**: You must bootstrap your AWS CDK environment in the target region you want to deploy, using the AWS CDK toolkit's `cdk bootstrap` command. From the command line, authenticate into your AWS account, and run `cdk bootstrap 'aws://<YOUR ACCOUNT NUMBER>/<REGION>'`. For more information, refer to the AWS CDK's How to bootstrap page.
- **Production access for Amazon SES**: This solution uses Amazon SES for sending email notifications to application pattern's subscribers. In order to use this feature, ensure that Amazon SES (in your account) is in a production environment, and not in the sandbox environment. For more information, refer to the Moving out of the Amazon SES sandbox page.
- **Your AWS account should be part of an AWS Organization**: This prerequisite is only applicable for application patterns that are of the CloudFormation type, and needs to be shared across accounts using AWS Service Catalog, as currently, the AWS Service Catalog AppRegistry attribute groups can only be shared to AWS accounts within an organization. This prerequisite does not apply to CDK-based application patterns.

GitHub and GitHub Enterprise account (required only if you use these for your pattern's source code repository)

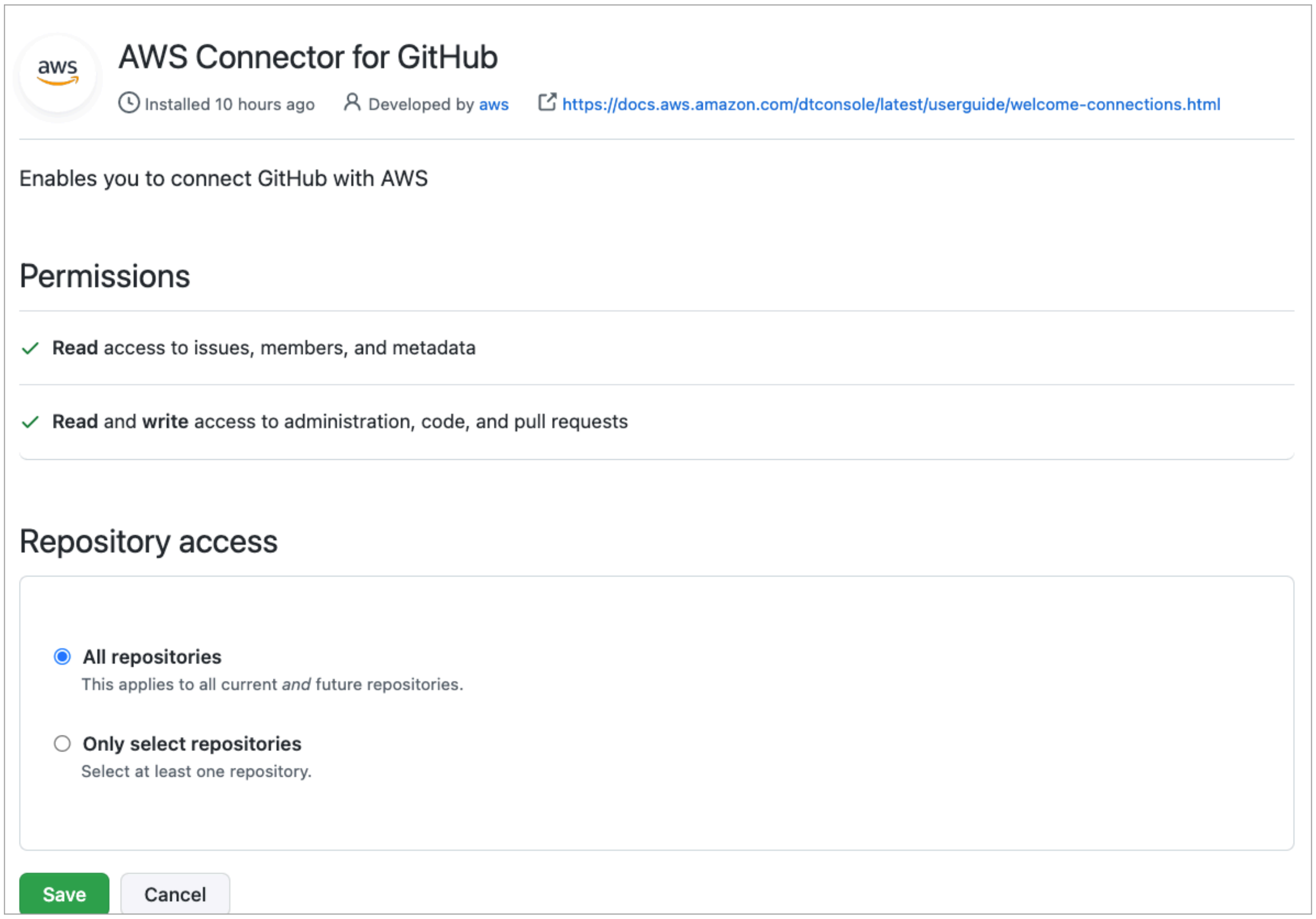
By default, the solution uses AWS CodeCommit to create pattern repositories.

Note

To configure GitHub or GitHub Enterprise as your pattern's source code repository instead, deploy the solution using AWS CDK by following the instructions in the solution README.

The solution supports both GitHub Teams and GitHub Enterprise (Enterprise Cloud and Enterprise Server) plans. A complete list of prerequisites related to GitHub/GitHub Enterprise is listed below:

- **GitHub Organization:** The solution assumes that an organization exists in the GitHub account. The pattern repositories will be created in this organization.
- **GitHub Organization Owner Account**: The organization owner is the only account that is allowed to create a GitHub App which is required to create an AWS CodeStar connection to GitHub, GitHub Enterprise Cloud or GitHub Enterprise server.
- **AWS CodeStar connection to GitHub, GitHub Enterprise Cloud, or GitHub Enterprise server:**
  - The solution integrates with GitHub, GitHub Enterprise Cloud or GitHub Enterprise server using AWS CodeStar connection. To create a AWS CodeStar connection to GitHub or GitHub Enterprise Cloud, refer to the Create a connection to GitHub guide. To create a AWS CodeStar connection to GitHub Enterprise Server, refer to the Create a connection to GitHub Enterprise Server guide.
  - As part of creating a AWS CodeStar connection, a GitHub app is installed to establish the connection between AWS and GitHub. Install the GitHub app in the Organization.
  - GitHub app permissions:
    - The GitHub app must have admin permissions granted as read and write. The admin permission for the GitHub app is required because when a new pattern is created by the solution, its code repository is created with master/main branch as protected. When the pattern's publishing pipeline runs, it upgrades the package versions and tries to push the change directly to the master/main branch. As the master/main branch is protected, only admins have the required permissions to directly push to the protected branches.
    - For GitHub and GitHub Enterprise in the cloud, the GitHub app has read and write admin permissions by default.


AWS Connector for GitHub

Installed 10 hours ago · Developed by aws · https://docs.aws.amazon.com/dtconsole/latest/userguide/welcome-connections.html

Enables you to connect GitHub with AWS

Permissions

✓ **Read** access to issues, members, and metadata

✓ **Read** and **write** access to administration, code, and pull requests

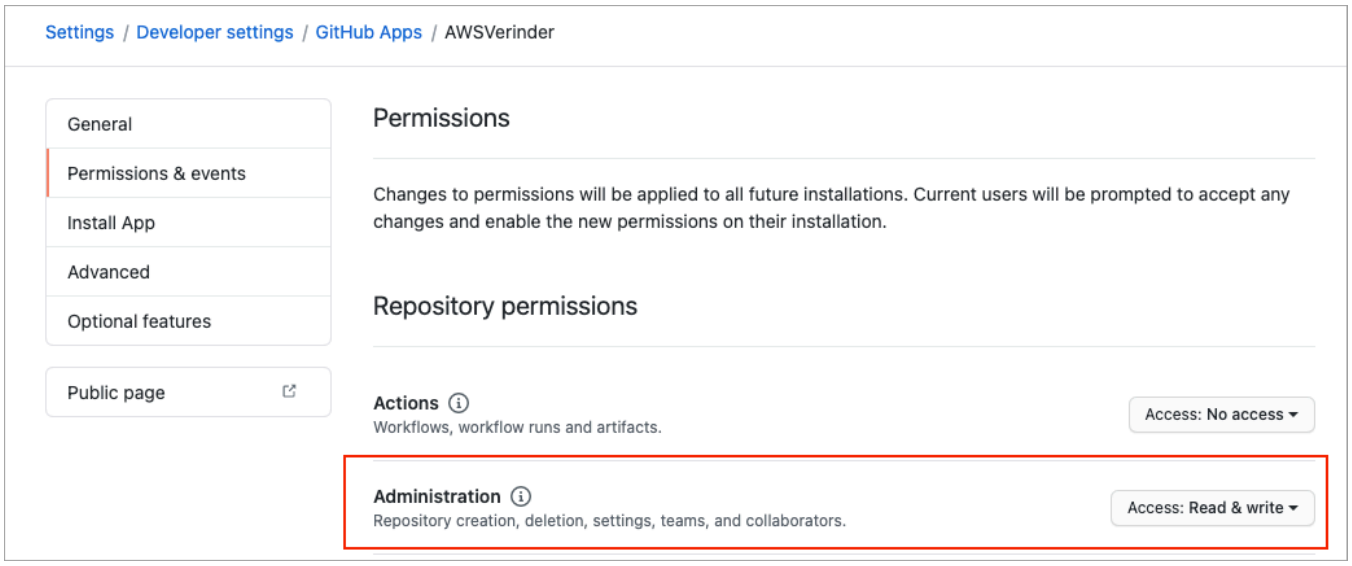
Repository access

● **All repositories**
This applies to all current *and* future repositories.

○ **Only select repositories**
Select at least one repository.

AWS Connector for GitHub

- For GitHub Enterprise Server, you must manually grant the admin permissions to the GitHub app. For more information about how to grant permissions to the GitHub app, refer to the Editing a GitHub App's permissions guide. Ensure that the permissions changes are accepted by the Organization account before you deploy the solution.


Settings / Developer settings / GitHub Apps / AWSVerinder

Permissions

Changes to permissions will be applied to all future installations. Current users will be prompted to accept any changes and enable the new permissions on their installation.

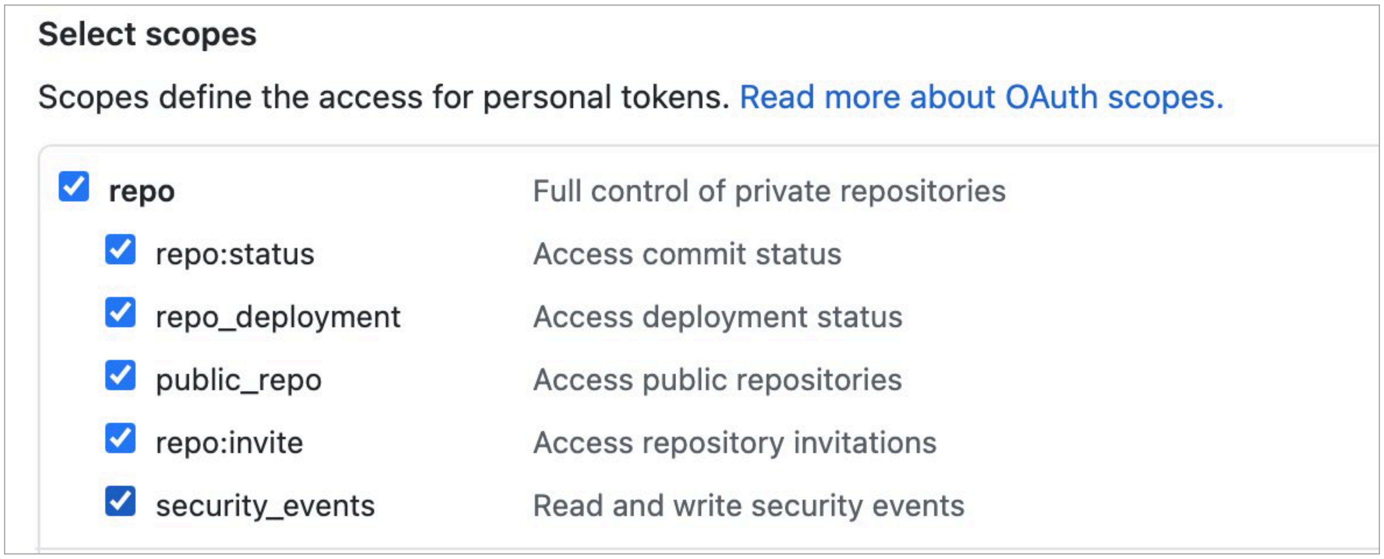
Repository permissions

Actions — Workflows, workflow runs and artifacts. — Access: No access

Administration — Repository creation, deletion, settings, teams, and collaborators. — Access: Read & write

GitHub App permissions

- Once the AWS CodeStar connection has been created successfully using the previous step, create an AWS SSM parameter with the name as `githubConnectionArn` and value as `AWS CodeStar connection ARN`.
- GitHub personal access token:
  - Create a personal access token from a GitHub account that is a member of the organization. This token is required by the solution to create the pattern's code repository in the organization and also to initialize it with an initial commit.
  - Token permissions should have `repo` and `delete repo` scopes.


Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ repo	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events

OAuth scopes

- GitHub personal access token to be stored as a secret in a text form in AWS Secrets Manager with the name **githubTokenSecretId**. It must be encrypted using the AWS managed key for Secrets Manager (`aws/secretsmanager`).

Launch the stack

> **Note**
>
> This solution includes an option to send anonymous operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy. To opt out of this feature, during the deployment, on the **Specify stack** details, from template parameter `sendAnonymousData` dropdown, select **No**.

This automated AWS CloudFormation template deploys the Application Pattern Orchestrator on AWS solution in the AWS Cloud.

> **Note**
>
> You are responsible for the cost of the AWS services used while running this solution. For more details, visit the Cost section in this guide, and refer to the pricing webpage for each AWS service used in this solution.

1. Sign in to the AWS Management Console and select the button to launch the **ApoStack.template** AWS CloudFormation template.

   Launch solution

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.
3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
patternType	`CloudFormation`	The type of application patterns that the solution would enable. Valid values are: 1. **CloudFormation**: CloudFormation based patterns are automatically published to Service Catalog as products. 2. **CDK**: CDK based patterns are automatically published to AWS CodeArtifact as npm packages. 3. **All**: Enables both CloudFormation and CDK based patterns.
adminEmail	*`<Requires input>`*	The solution creates a default user with this email address to login to the solution's UI. This has to be a valid email address as you will receive the temporary password on this email address.

Parameter	Default	Description
sendAnonymousData	`Yes`	Send anonymous operation al metrics to AWS. We use this data to better understan d how customers use this solution and related services and products.

6. Choose **Next**.
7. On the **Configure stack options** page, select **Next**.
8. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack. You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a `CREATE_COMPLETE` status in approximately 15 minutes.

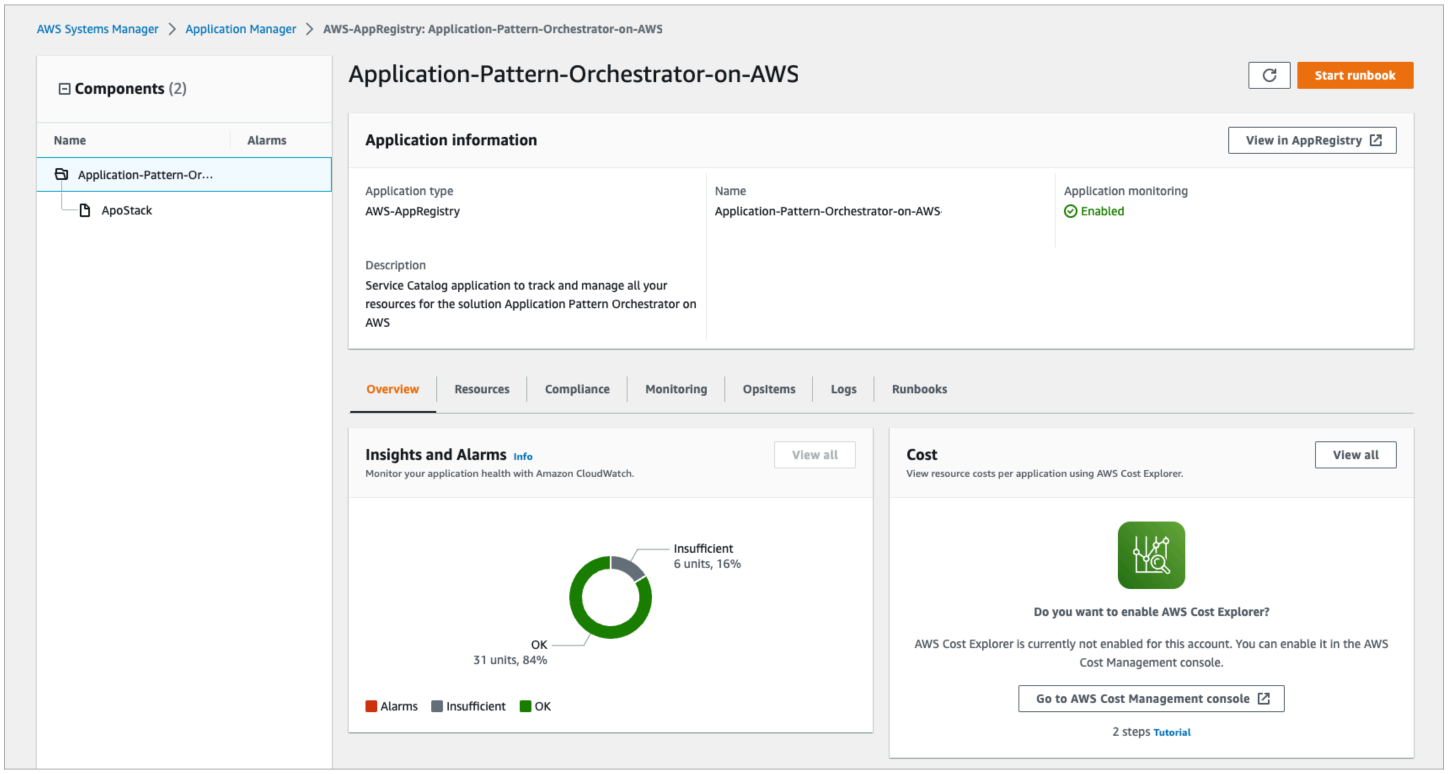
Monitoring the solution with AppRegistry

The Application Pattern Orchestrator on AWS solution includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both AWS Service Catalog AppRegistry and AWS Systems Manager Application Manager.

AWS Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:

- Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.
- View operations data for the resources of this solution in the context of an application. For example, deployment status, CloudWatch alarms, resource configurations, and operational issues.

The following figure depicts an example of the application view for the Application-Pattern-Orchestrator-on-AWS stack in Application Manager



Application-Pattern-Orchestrator-on-AWS

Application information

Application type: AWS-AppRegistry

Name: Application-Pattern-Orchestrator-on-AWS

Application monitoring: Enabled

Description: Service Catalog application to track and manage all your resources for the solution Application Pattern Orchestrator on AWS

Insights and Alarms Info
Monitor your application health with Amazon CloudWatch.

Cost
View resource costs per application using AWS Cost Explorer.

Do you want to enable AWS Cost Explorer?

AWS Cost Explorer is currently not enabled for this account. You can enable it in the AWS Cost Management console.

Application Pattern Orchestrator on AWS stack in Application Manager

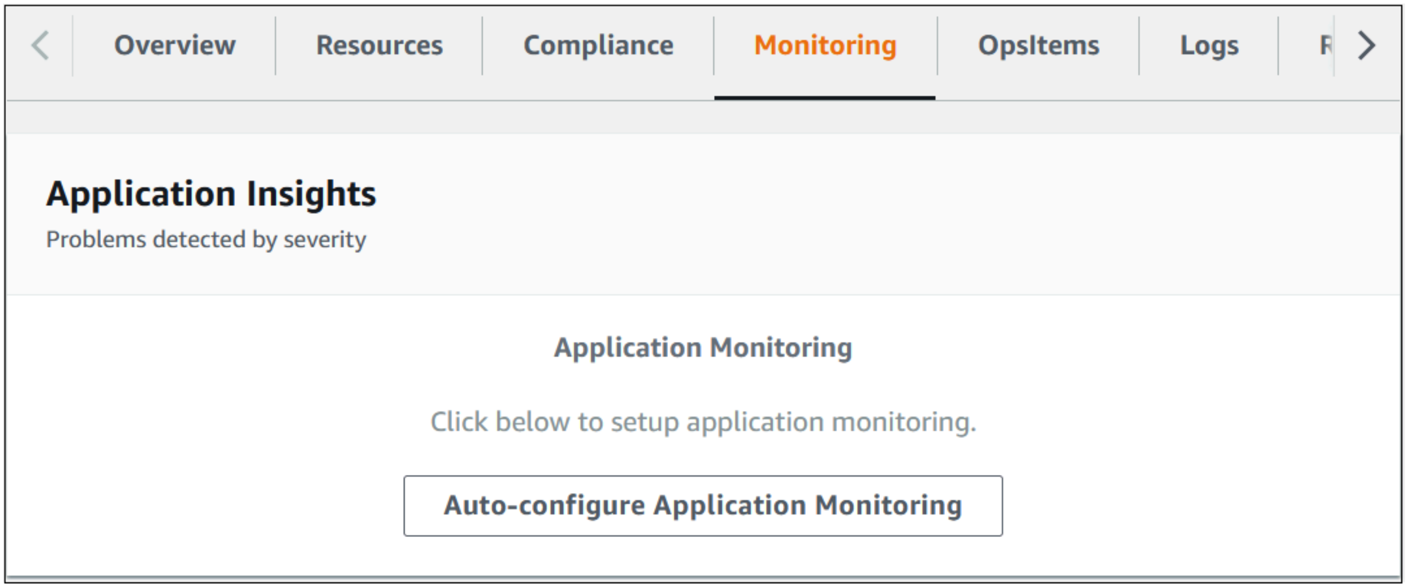
Note

You must activate CloudWatch Application Insights, AWS Cost Explorer, and cost allocation tags associated with this solution. They are not activated by default.

Activate CloudWatch Application Insights

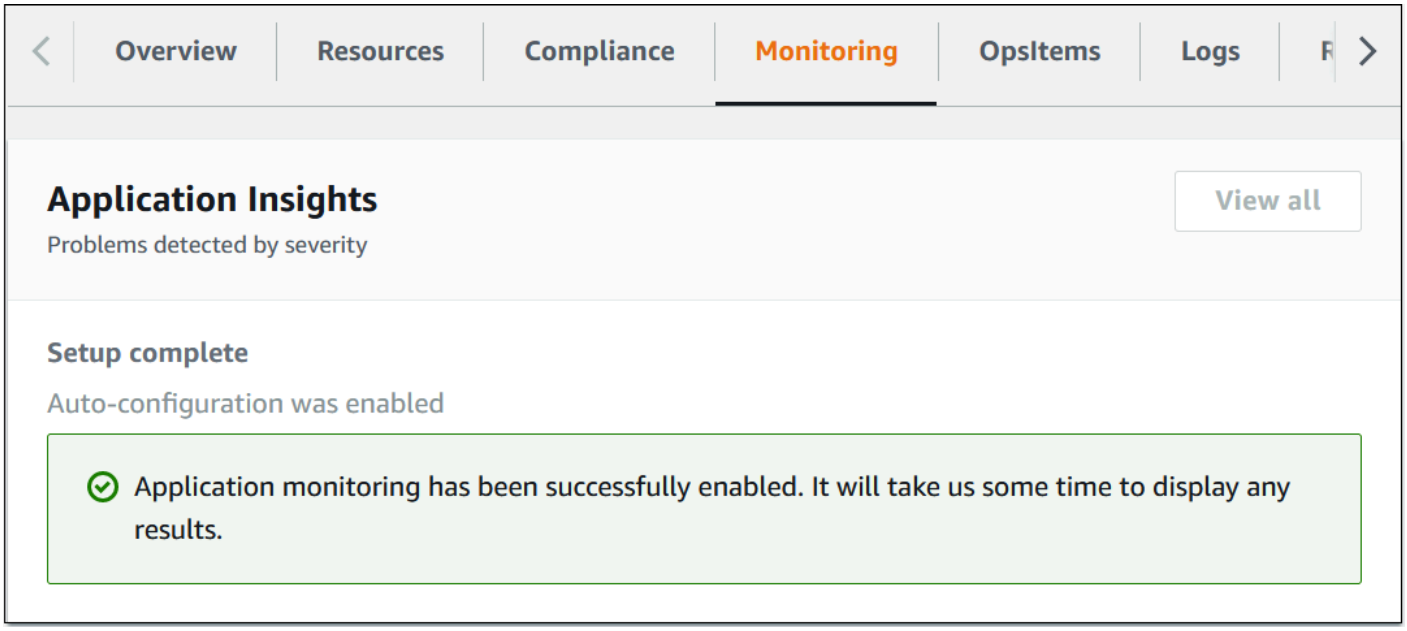
After the solution CloudFormation stack has been deployed and launched, you can sign in to the web interface.

1. Sign in to the Systems Manager console. In the navigation pane, choose **Application Manager**.
2. In **Applications**, choose **AppRegistry applications**.
3. In **AppRegistry applications**, search for the application name for this solution and select it. The next time you open Application Manager, you can find the new application for your solution in the **AppRegistry application** category.
4. In the **Components** tree, choose the application stack you want to activate.
5. In the **Monitoring** tab, in **Application Insights**, select **Auto-configure Application Monitoring**.


Overview
Resources
Compliance
Monitoring
OpsItems
Logs
R
Application Insights
Problems detected by severity
Application Monitoring
Click below to setup application monitoring.
Auto-configure Application Monitoring

Auto configure Application Monitoring

6. Monitoring for your applications is now activated and the following status box appears.



Application Insights

Problems detected by severity

Setup complete

Auto-configuration was enabled

Application monitoring has been successfully enabled. It will take us some time to display any results.

Activated Application Insights

Activate AWS Cost Explorer

You can see the overview of the costs associated with the application and application components within the Application Manager console through integration with AWS Cost Explorer which must be first activated. Cost Explorer helps you manage costs by providing a view of your AWS resource costs and usage over time. To activate Cost Explorer for the solution:

1. Sign in to the AWS Cost Management console.
2. In the navigation pane, select **Cost Explorer**.
3. On the **Welcome to Cost Explorer** page, choose **Launch Cost Explorer**.

Activate cost allocation tags associated with the solution

After you activate Cost Explorer, you must activate the cost allocation tags associated with this solution to see the costs for this solution. The cost allocation tags can only be activated from the management account for the organization.

To activate cost allocation tags:

1. Sign in to the AWS Billing and Cost Management console , and select **Cost Allocation Tags** in the left navigation menu.
2. On the **Cost allocation tags** page, filter for the `AppManagerCFNStackKey` tag, then select the tag from the results shown.
3. Choose **Activate**.

The activation process can take up to 24 hours to complete and the tag data to appear.

Update the solution

If you have previously deployed the solution, follow the following steps to update the current deployment with the latest released version.

1. Select the **View Template** button to download the **ApoStack.template** to a location on your computer.

   View template

2. Sign in to the AWS CloudFormation console, and select the correct account and Region. From the list of stacks, select the solution's stack name (this is the name that you specified when deploying the solution for the first time), and choose the **Update** button.
3. On the **Update stack** page, select **Replace current template**, and under **Specify template**, select **Upload a template file**.
4. Under **Upload a template file**, select **Choose file**, and select the **ApoStack.template** file that you downloaded on your computer.
5. Select **Next** to continue.
6. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values:

Parameter	Default	Description
patternType	`CloudFormation`	The type of application patterns that the solution would enable. Valid values are: 1. **CloudFormation**: CloudFormation based patterns are automatically published to Service Catalog as products. 2. **CDK**: CDK based patterns are automatically published

Parameter	Default	Description
		to AWS CodeArtifact as npm packages. 3. **All**: Enables both CloudFormation and CDK based patterns.
adminEmail	*`<Requires input>`*	The solution creates a default user with this email address to login to the solution's UI. This has to be a valid email address as you will receive the temporary password on this email address.
sendAnonymousData	`Yes`	Send anonymous operation al metrics to AWS. We use this data to better understan d how customers use this solution and related services and products.

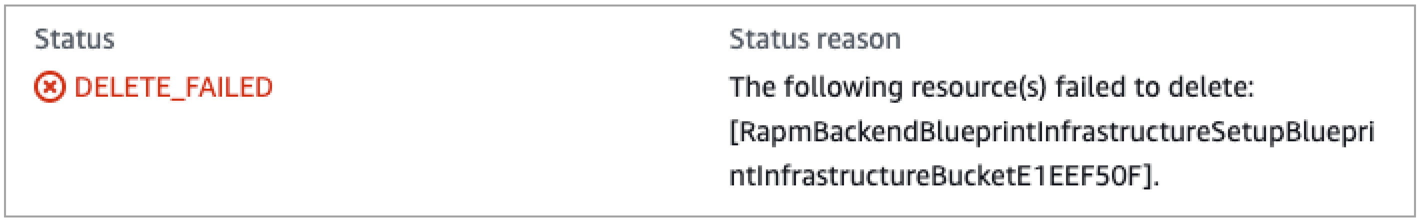
7. Select **Next**.
8. On the Configure stack options page, select **Next.**
9. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.

10Select **Submit** to update the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column.

Troubleshooting

Deletion of the solution stack fails



Status
DELETE_FAILED

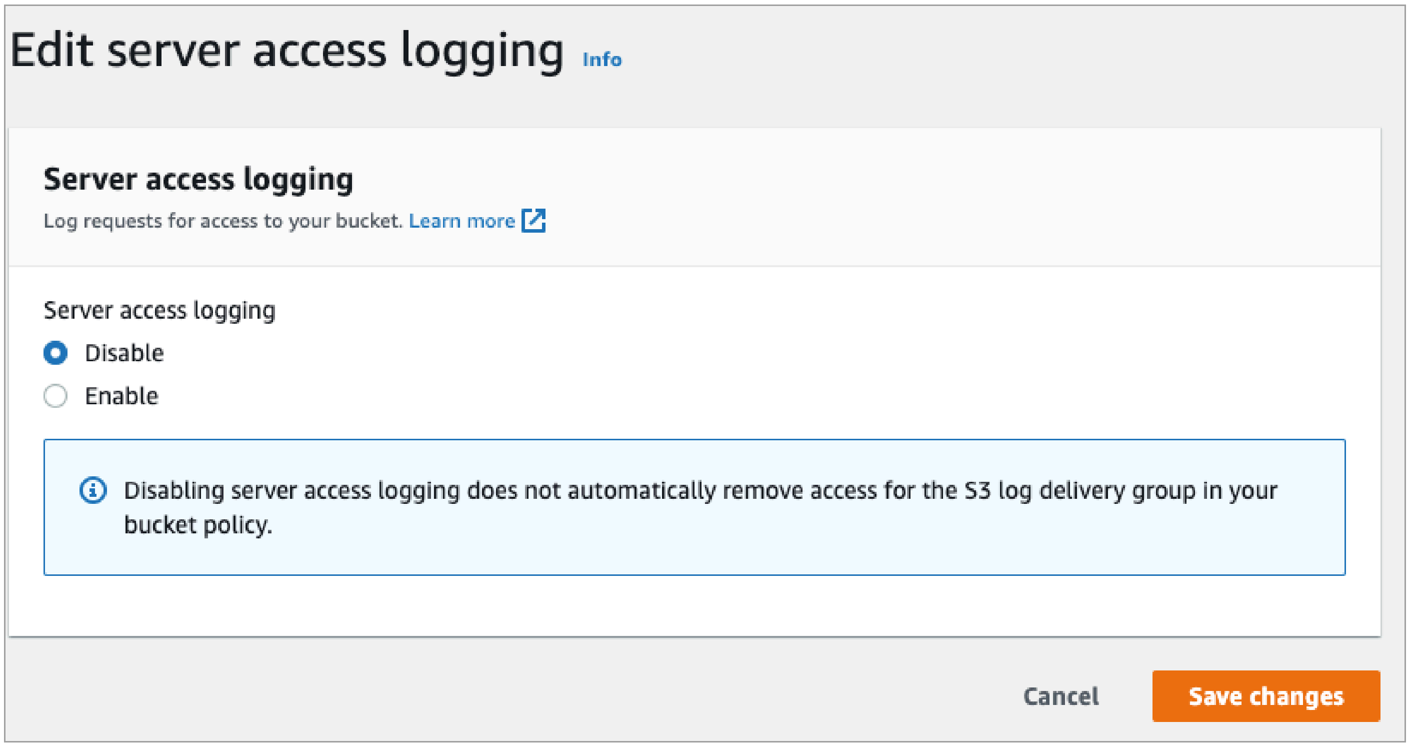
Status reason
The following resource(s) failed to delete: [RapmBackendBlueprintInfrastructureSetupBlueprintInfrastructureBucketE1EEF50F].

Delete solution stack error

Issue: The issue occurs because all the S3 buckets created in the solution have access logging enabled by default, which may prevent deleting the buckets while the access logs are still being written to the access log bucket.

Resolution: To resolve this:

1. Open the AWS Management Console and select your account and region.
2. Navigate to the **S3 console**, and select the bucket that caused this issue.
3. Select **Edit**, and **Disable**.



Edit server access logging Info

Server access logging
Log requests for access to your bucket. Learn more

Server access logging
[✓ Disable]
Enable

Disabling server access logging does not automatically remove access for the S3 log delivery group in your bucket policy.

Edit server access logging

4. Select **Save changes** to disable the access logging on that bucket.
5. Repeat steps 2 to 4 for all the buckets that failed during stack deletion.
6. After all the failed buckets have been removed, delete the solution stack.

Uninstall the solution

You can uninstall the solution by deleting the stacks from the AWS CloudFormation console.

Sign in to the AWS CloudFormation console, and find and delete the following stacks (in the specified order):

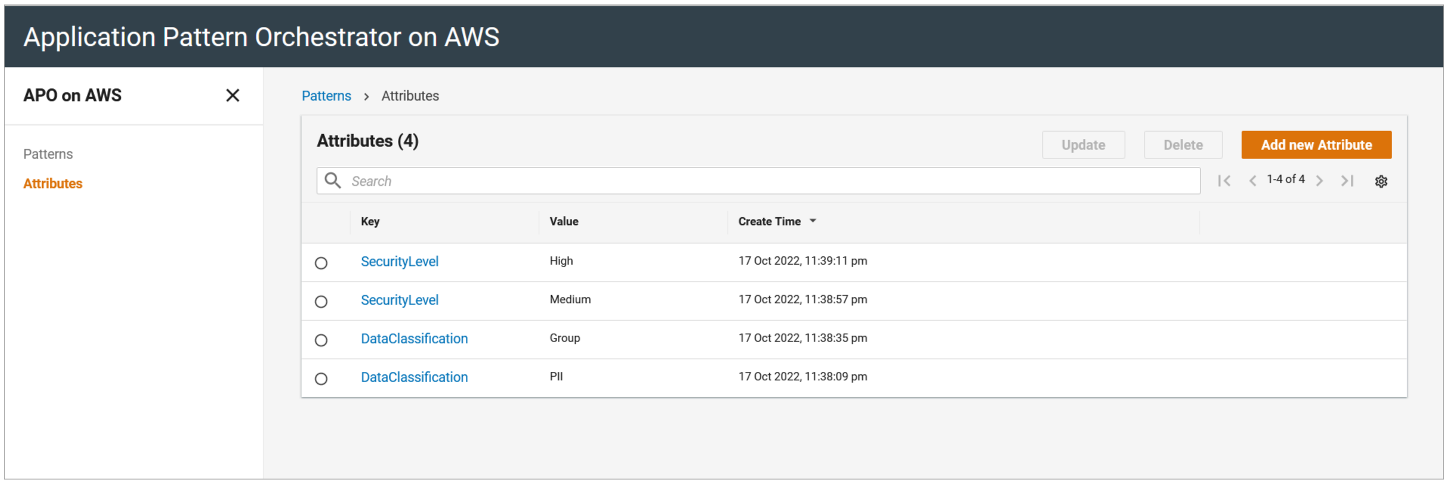
- All the stacks with the prefix `BlueprintInfrastructureStack`
- The stack name you used to deploy the solution.
- Delete the pattern repositories (if not needed anymore).

Using the solution's web UI

You can use the Application Pattern Orchestrator on AWS web user interface to add, delete, and manage application patterns and attributes. This section provides details about the features of the Application Pattern Orchestrator on AWS web UI and how to use them for patterns and attributes.

The side navigation pane displays the following options:

Option	Description
Patterns	Displays a list of all patterns currently set up and validated for use.
Attributes	Displays a list of currently available attributes.



Key	Value	Create Time
SecurityLevel	High	17 Oct 2022, 11:39:11 pm
SecurityLevel	Medium	17 Oct 2022, 11:38:57 pm
DataClassification	Group	17 Oct 2022, 11:38:35 pm
DataClassification	PII	17 Oct 2022, 11:38:09 pm

Solution web UI

Sign in to the web interface

After the solution CloudFormation stack has been deployed and launched, you can sign in to the web interface.

1. Sign in to the AWS CloudFormation console and select the stack name you used when deploying the solution.
2. Select the **Outputs** tab.

3. Under the **Key** column, search for the key starting with `RapmFrontendCloudFrontURL`, and select to open the link.
4. The initial username is the `adminEmail` specified during the deployment. You will receive a temporary password from `no-reply@verificationemail.com`.
5. On the Application Pattern Orchestrator on AWS login screen, enter the **Username** and **Password** and select **Sign In**.

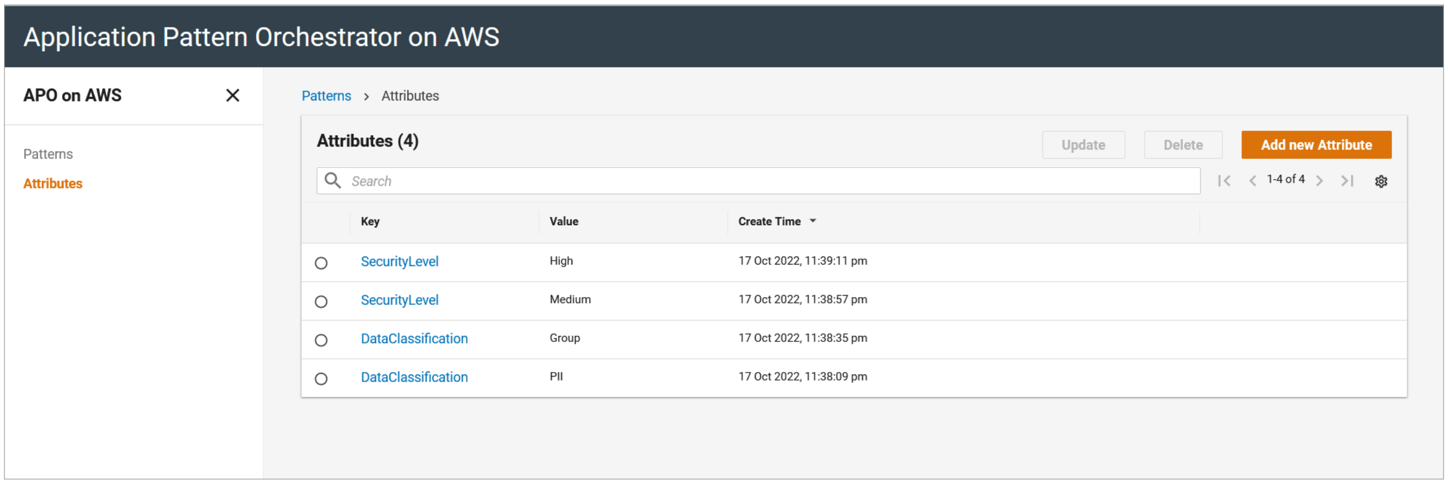
The Application Pattern Orchestrator on AWS redirects to the login page. By default, the solution's web frontend uses Amazon Cognito to authenticate and authorize users.

Create an attribute

You can create attributes to describe any characteristic of a pattern, such as hosting construct or technology stack. In the context of this solution, they are intended to inform governance, risk, and compliance characteristics.

Note

Attributes are optional and you do not need them for creating patterns.



Key	Value	Create Time
SecurityLevel	High	17 Oct 2022, 11:39:11 pm
SecurityLevel	Medium	17 Oct 2022, 11:38:57 pm
DataClassification	Group	17 Oct 2022, 11:38:35 pm
DataClassification	PII	17 Oct 2022, 11:38:09 pm

Attributes overview

1. Sign in to the web interface, and from the left navigation menu, select **Attributes**.
2. Select **Add new Attribute**.

3. On the **Create Attribute** screen, enter an attribute key, value and description, and select **Next**.
4. On the **Attribute Metadata** page, enter metadata information about the attribute. This is optional.
5. On the **Review** page, verify the data entered and select **Submit** to create the attribute.

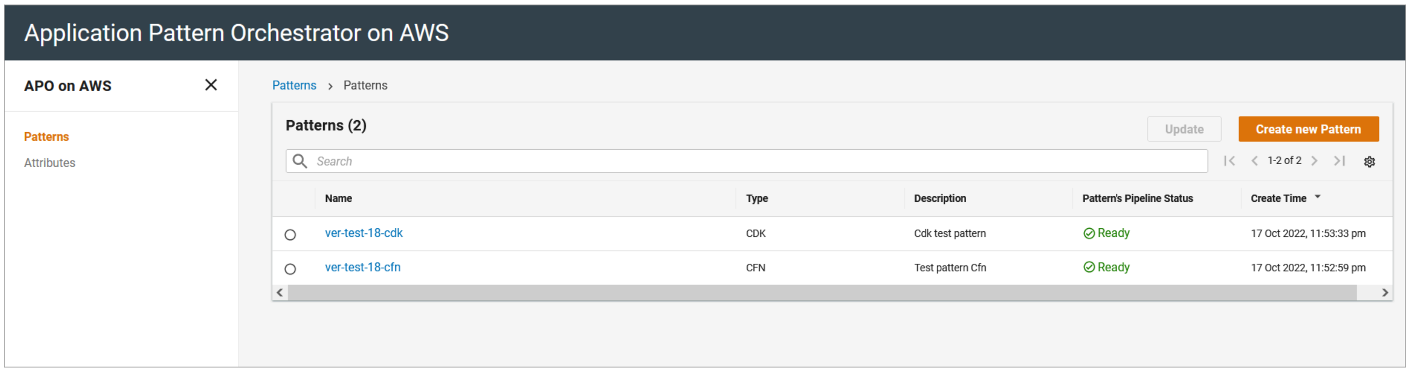
Create a pattern

A pattern may be described by one or more attributes on its initial definition or as part of a subsequent update. For example, if a pattern is associated with an attribute called dataClassification with the value as Confidential, an application that uses the pattern may be deemed a higher risk compared to an application that uses a pattern tagged with an attribute of `{dataClassification:Public}`.

Based on such attributes, you can exercise appropriate control behaviors, such as limiting access to the patterns themselves to privileged parties within an organization, or running specific preventative governance checks in a CI/CD pipeline.

To create a pattern:

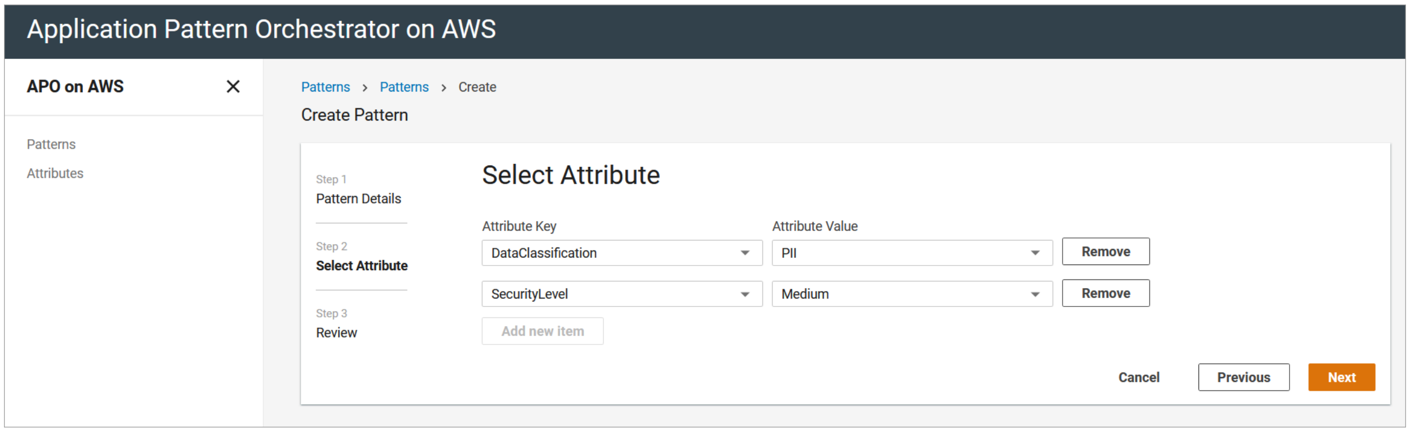
1. Sign in to the web interface, and from the left navigation menu, select **Patterns.**
2. Choose **Create new Pattern**.



Name	Type	Description	Pattern's Pipeline Status	Create Time
ver-test-18-cdk	CDK	Cdk test pattern	Ready	17 Oct 2022, 11:53:33 pm
ver-test-18-cfn	CFN	Test pattern Cfn	Ready	17 Oct 2022, 11:52:59 pm

Patterns overview

3. On the **Pattern details** page, enter the pattern name and description, select the pattern type, and select **Next**.
4. On the **Select Attributes** page, associate attributes with the pattern, and select **Next**.


Application Pattern Orchestrator on AWS

APO on AWS

Patterns
Attributes

Patterns > Patterns > Create

Create Pattern

Step 1 Pattern Details
Step 2 Select Attribute
Step 3 Review

Select Attribute

Attribute Key	Attribute Value	
DataClassification	PII	Remove
SecurityLevel	Medium	Remove

Add new item

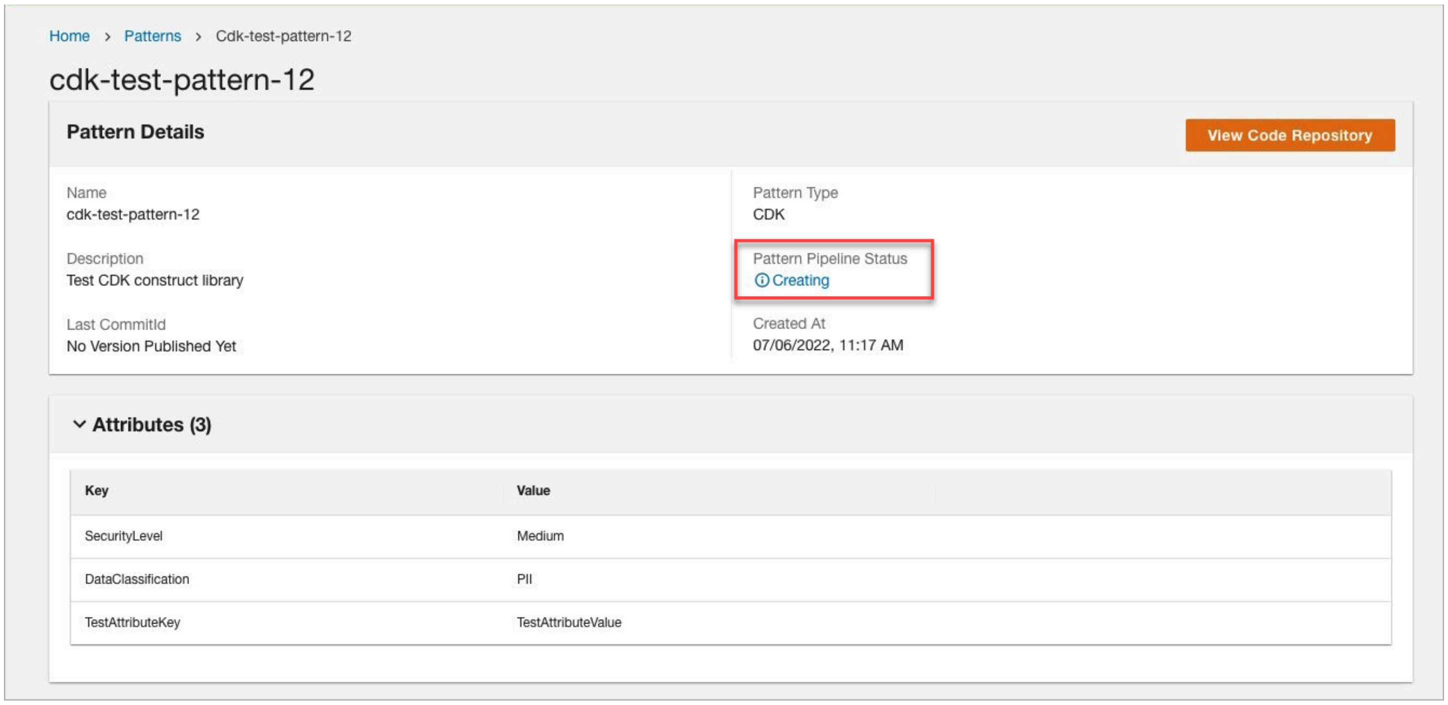
Cancel Previous Next

Select attributes

5. On the **Review** page, verify the data you entered, and select **Submit**.

This will create a pattern with the status as **Creating**. The process creates a new code repository for this pattern and initiates the provisioning of the pattern's publishing pipeline (takes approximately 10 minutes to provision).

After the pattern's publishing pipeline is ready, the pattern pipeline status on the pattern's details page changes from **Creating** to **Ready.**


Home > Patterns > Cdk-test-pattern-12

cdk-test-pattern-12

Pattern Details

View Code Repository

Name
cdk-test-pattern-12

Description
Test CDK construct library

Last CommitId
No Version Published Yet

Pattern Type
CDK

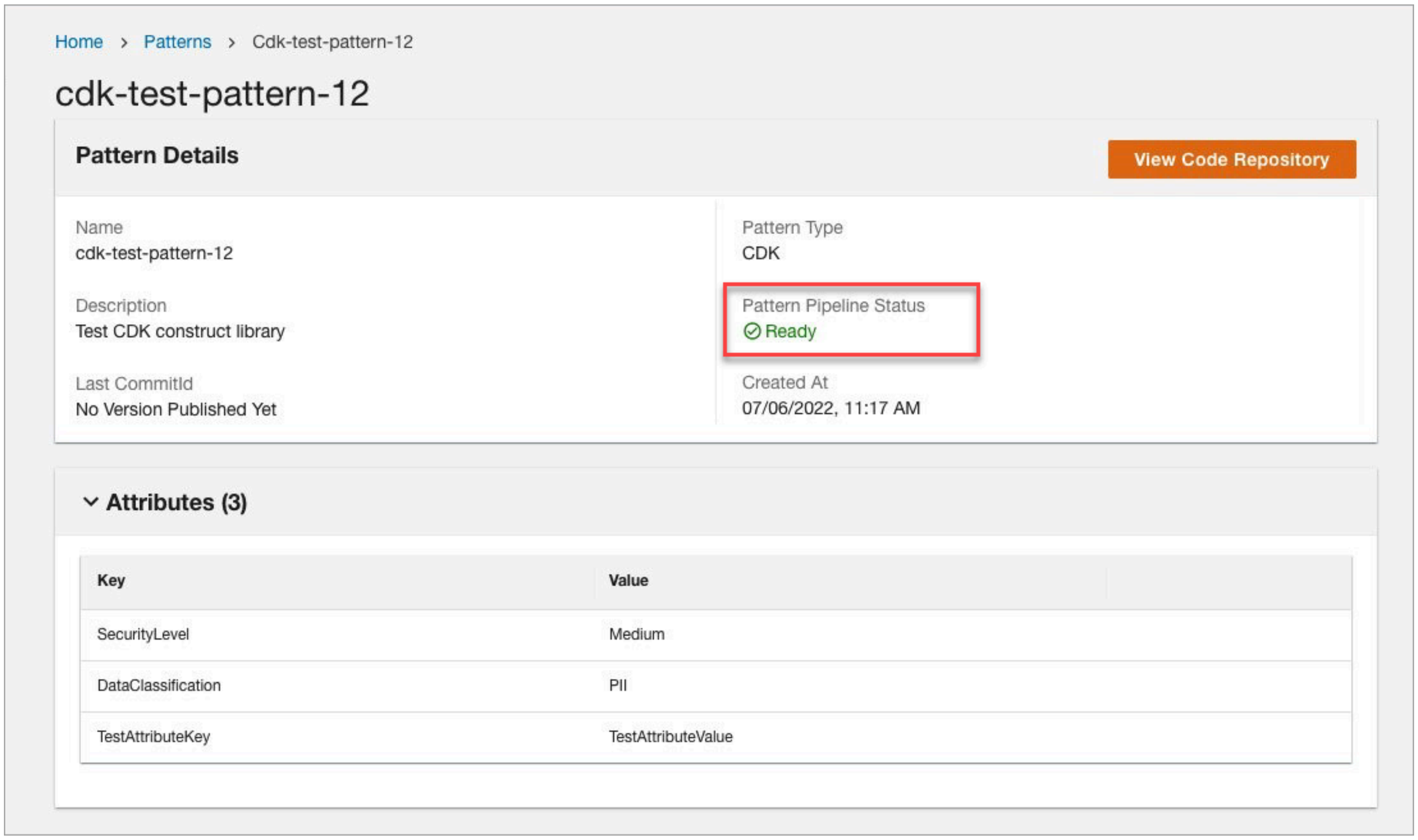
Pattern Pipeline Status
Creating

Created At
07/06/2022, 11:17 AM

Attributes (3)

Key	Value
SecurityLevel	Medium
DataClassification	PII
TestAttributeKey	TestAttributeValue

Pattern publishing pipeline in Creating state



Home > Patterns > Cdk-test-pattern-12

cdk-test-pattern-12

Pattern Details

View Code Repository

Name	cdk-test-pattern-12
Description	Test CDK construct library
Last CommitId	No Version Published Yet
Pattern Type	CDK
Pattern Pipeline Status	✓ Ready
Created At	07/06/2022, 11:17 AM

Attributes (3)

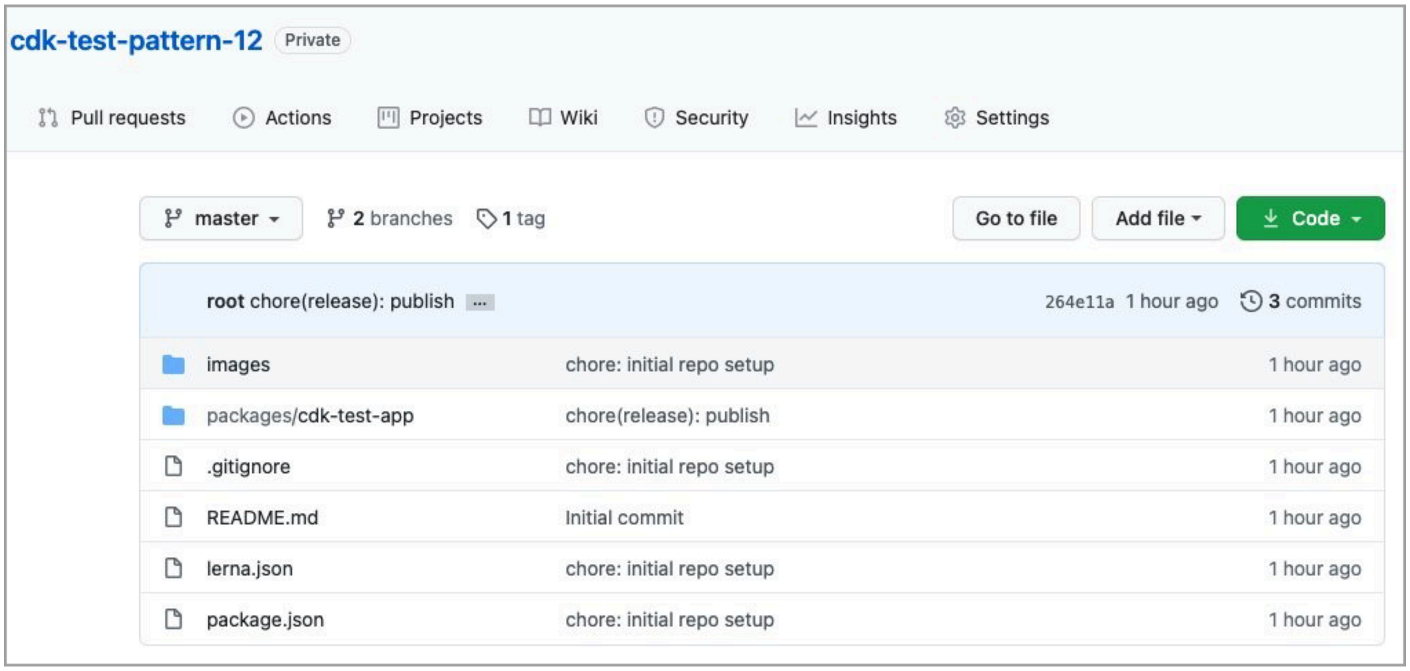
Key	Value
SecurityLevel	Medium
DataClassification	PII
TestAttributeKey	TestAttributeValue

Publishing pipeline in Ready state

After the pattern's publishing pipeline is in **Ready** state, the pattern developer is ready to clone the pattern's repository.

Publish a pattern

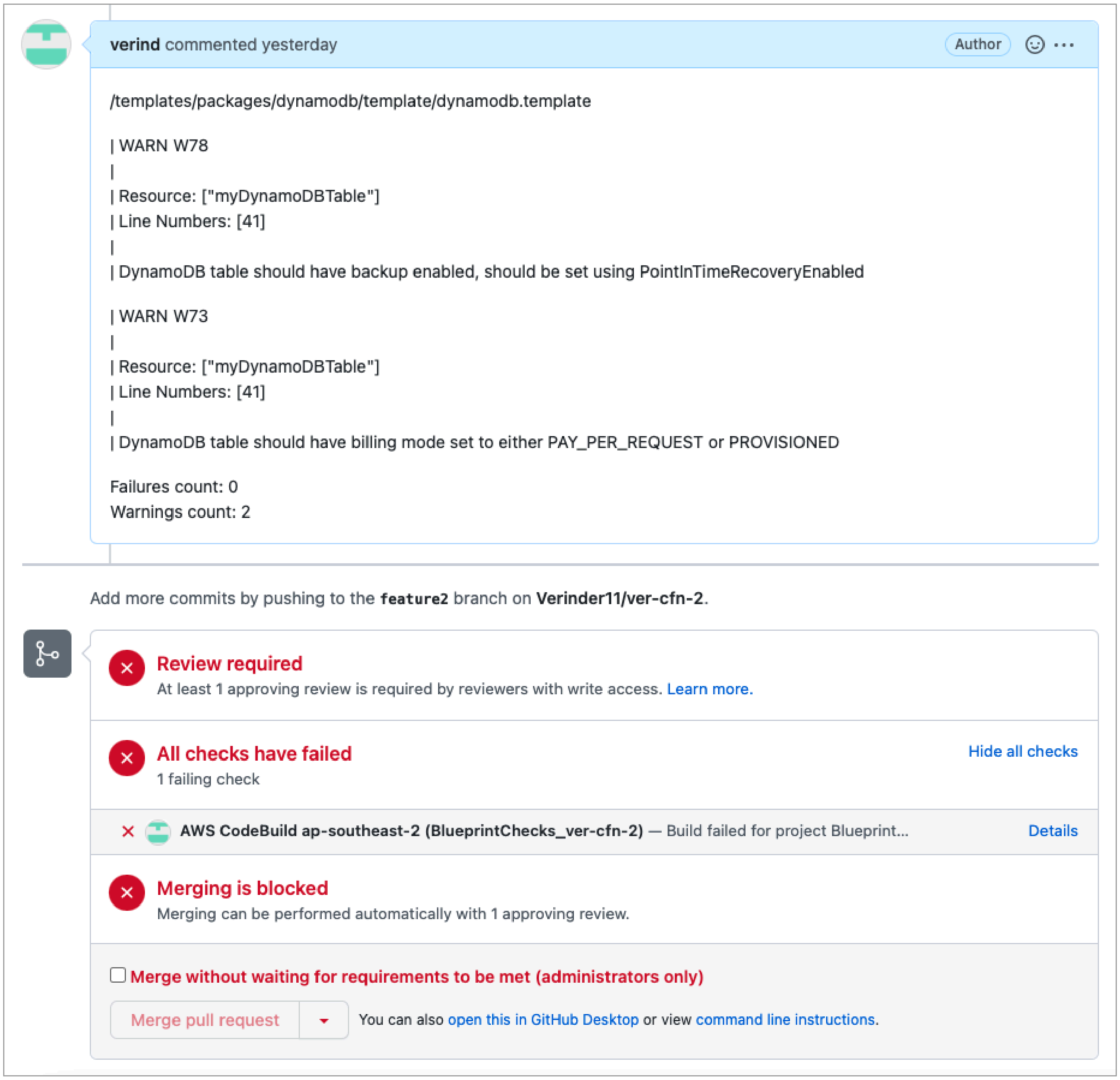
1. Click the **View Code Repository** button on the pattern details page to open your repository.



root chore(release): publish	264e11a 1 hour ago	3 commits
images	chore: initial repo setup	1 hour ago
packages/cdk-test-app	chore(release): publish	1 hour ago
.gitignore	chore: initial repo setup	1 hour ago
README.md	Initial commit	1 hour ago
lerna.json	chore: initial repo setup	1 hour ago
package.json	chore: initial repo setup	1 hour ago

View code repo

2. Clone the Git repo, create a feature branch and add the code changes to add new CDK constructs or CloudFormation templates for CDK or CloudFormation pattern types respectively. For more information on the repo structure, refer to the Code repo recommendations section.
3. For a CDK type pattern, ensure that the new CDK constructs you add their demonstrated usage in `packages # cdk-test-app # bin # cdk-test-app.ts`.
4. `Raise a pull request` (PR) with the main/master branch as the base.
5. The `pull request` launches an automated security check using AWS CodeBuild which runs a `cfn_nag` check against the CDK constructs in the repo and reports either a **Success** or **Failure** status back along with the scan results in the comments section.



verind commented yesterday

Author

```
/templates/packages/dynamodb/template/dynamodb.template

| WARN W78
|
| Resource: ["myDynamoDBTable"]
| Line Numbers: [41]
|
| DynamoDB table should have backup enabled, should be set using PointInTimeRecoveryEnabled

| WARN W73
|
| Resource: ["myDynamoDBTable"]
| Line Numbers: [41]
|
| DynamoDB table should have billing mode set to either PAY_PER_REQUEST or PROVISIONED

Failures count: 0
Warnings count: 2
```

Add more commits by pushing to the **feature2** branch on **Verinder11/ver-cfn-2**.

Review required
At least 1 approving review is required by reviewers with write access. Learn more.

All checks have failed
1 failing check

Hide all checks

AWS CodeBuild ap-southeast-2 (BlueprintChecks_ver-cfn-2) — Build failed for project Blueprint... Details

Merging is blocked
Merging can be performed automatically with 1 approving review.

Merge without waiting for requirements to be met (administrators only)

Merge pull request

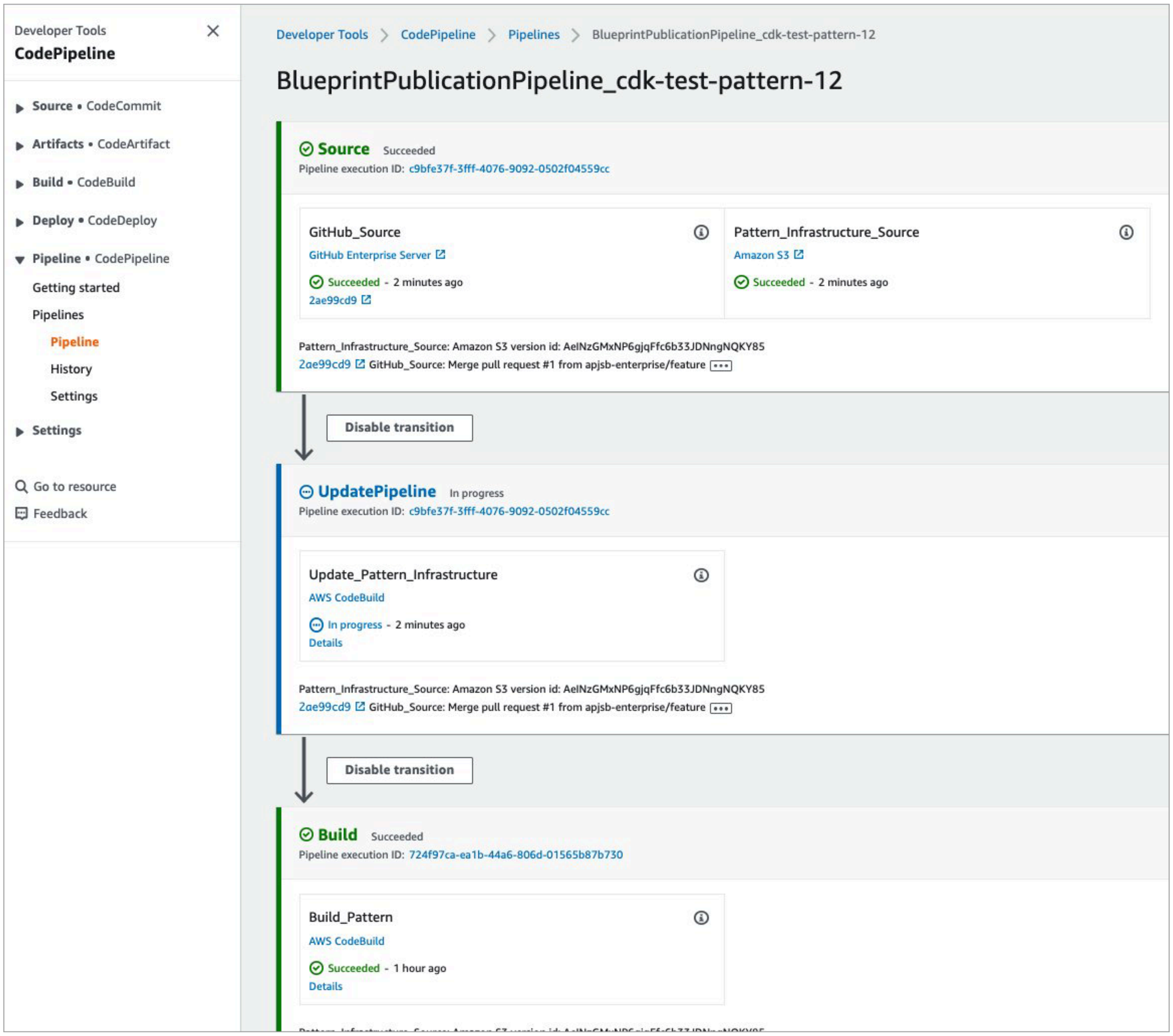
You can also open this in GitHub Desktop or view command line instructions.

PR comments

We recommend a security person review the PR, who should be able to review the `cfn_nag` results. Based on the `cfn_nag` results, the security reviewer can approve or reject the PR.

6. Once the `pull request` is approved and merged, the pattern's publishing pipeline automatically initiates and publishes the pattern to either Service Catalog (for a

CloudFormation-based pattern) or to AWS CodeArtifact (for a CDK-based pattern). The pattern publishing process takes approximately 10-15 minutes.



BlueprintPublicationPipeline_cdk-test-pattern-12

Source Succeeded
Pipeline execution ID: c9bfe37f-3fff-4076-9092-0502f04559cc

GitHub_Source
GitHub Enterprise Server
Succeeded - 2 minutes ago
2ae99cd9

Pattern_Infrastructure_Source
Amazon S3
Succeeded - 2 minutes ago

Pattern_Infrastructure_Source: Amazon S3 version id: AelNzGMxNP6gjqFfc6b33JDNngNQKY85
2ae99cd9 GitHub_Source: Merge pull request #1 from apjsb-enterprise/feature

UpdatePipeline In progress
Pipeline execution ID: c9bfe37f-3fff-4076-9092-0502f04559cc

Update_Pattern_Infrastructure
AWS CodeBuild
In progress - 2 minutes ago
Details

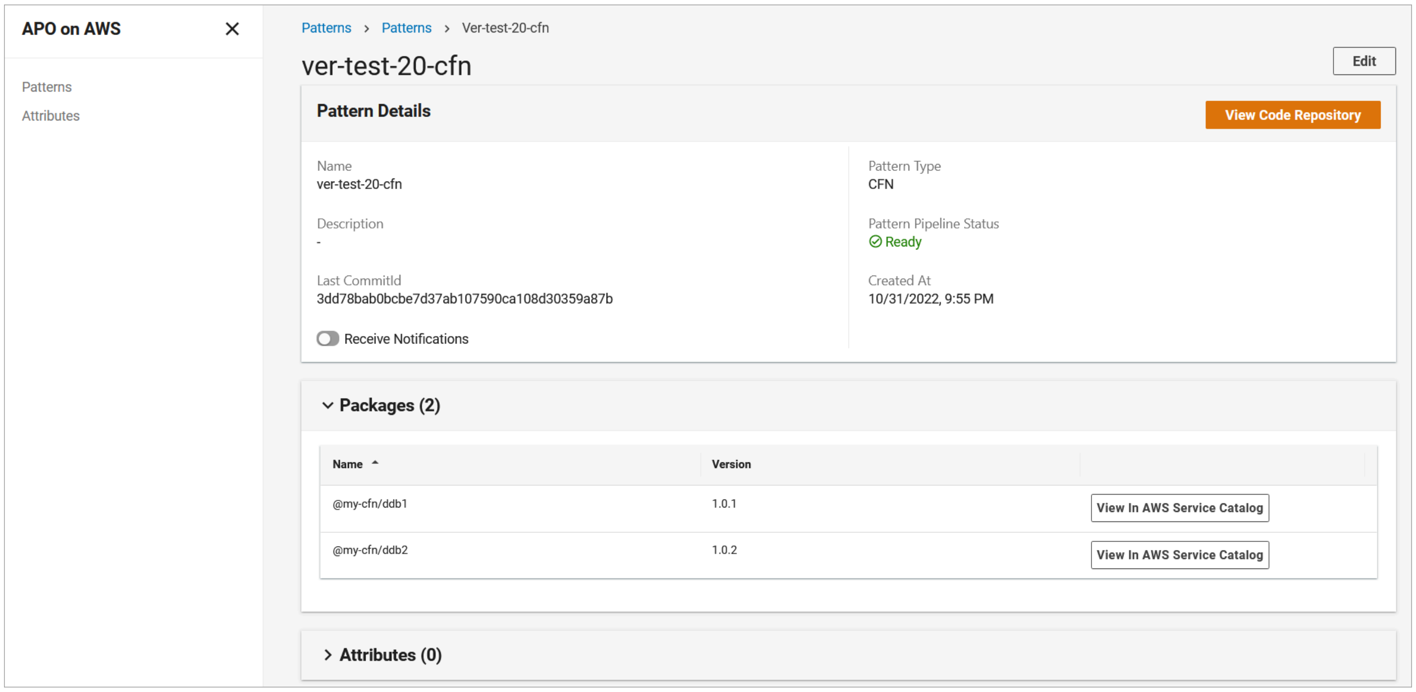
Pattern_Infrastructure_Source: Amazon S3 version id: AelNzGMxNP6gjqFfc6b33JDNngNQKY85
2ae99cd9 GitHub_Source: Merge pull request #1 from apjsb-enterprise/feature

Build Succeeded
Pipeline execution ID: 724f97ca-ea1b-44a6-806d-01565b87b730

Build_Pattern
AWS CodeBuild
Succeeded - 1 hour ago
Details

Pattern publishing process

- Once the pattern is published the pattern details page on the solution's UI is updated with the publish details.



APO on AWS

Patterns
Attributes

Patterns > Patterns > Ver-test-20-cfn

ver-test-20-cfn

Edit

Pattern Details

View Code Repository

Name	Pattern Type
ver-test-20-cfn	CFN
Description	Pattern Pipeline Status
-	Ready
Last CommitId	Created At
3dd78bab0bcbe7d37ab107590ca108d30359a87b	10/31/2022, 9:55 PM

Receive Notifications

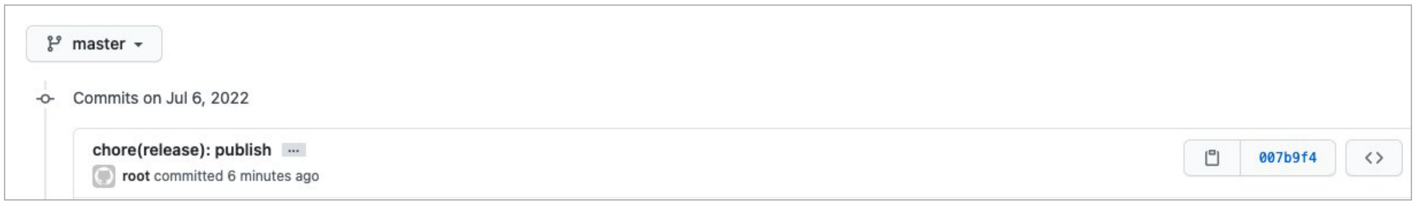
Packages (2)

Name	Version	
@my-cfn/ddb1	1.0.1	View In AWS Service Catalog
@my-cfn/ddb2	1.0.2	View In AWS Service Catalog

Attributes (0)

Published pattern details

- As part of publishing the pattern, the solution creates a tag in the code repo and increments the package version numbers.



master

Commits on Jul 6, 2022

chore(release): publish

root committed 6 minutes ago

007b9f4

Package version numbers

- To view the CDK npm packages:
  - Click the **View in CodeArtifact** button to open the AWS CodeArtifact page in the AWS Management Console.
  - For CloudFormation-based pattern, click the **View in Service Catalog** button to open the associated Service Catalog page in the AWS management console.

Developer guide

Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others.

The Application Pattern Orchestrator on AWS templates are generated using the AWS Cloud Development Kit (AWS CDK) (AWS CDK). Refer to the README.md file for additional information.

Use the solution APIs

You can use the Application Pattern Orchestrator on AWS solution via Application Programming Interfaces (APIs). Using the APIs, you can perform all the same actions as you would with using the web UI, such as:

- Creating, updating, deleting and viewing attributes,
- Creating, updating, and viewing patterns

For more information, refer to the source code API documentation.

Reference

This section includes information about recommended best practices for code repositories, an optional feature for collecting unique metrics for this solution, pointers to related resources, and a list of builders who contributed to this solution.

Best practice recommendations

Code repo recommendations

Mono-repo

There are two strategies for managing the codebase in a repository: mono-repo and multi-repo. From this solution's perspective we recommend using the mono-repo strategy for patterns.

The main reason for recommending mono-repo for patterns code repository is because of the AWS resource limits in an AWS account. When a new pattern is created by the solution, a dedicated publishing pipeline is also created for that pattern to publish a new version of the pattern each time there is a code commit in the pattern's main branch of the repo. Each publishing pipeline creates a number of AWS resources in the AWS account where the solution is deployed. As the number of patterns grow in number, the number of publishing pipelines grow as well and so do the number of AWS resources in each of the publishing pipeline. Below are some figures on number of resources each pattern's publishing pipeline creates.

Note

This list is not comprehensive and covers only the key AWS resources which should be considered for calculating the numbers of patterns in an AWS account.

AWS Resource Name	Count per pattern publishing pipeline	Default Quota	Maximum Quota	Reference
IAM Roles	17	1,000	5,000	IAM and AWS STS quotas,

AWS Resource Name	Count per pattern publishing pipeline	Default Quota	Maximum Quota	Reference
				name requirements, and character limits
CodePipeline	1	1,000		Quotas in AWS CodePipeline
CodeBuild	3	5,000		Quotas for AWS CodeBuild
KMS Keys	1	10,000		Resource quotas

For example, using the IAM roles maximum quota (5000), the maximum number of patterns that can be created are roughly ~290. This is an estimated figure as there are a few IAM roles created when deploying the solution as well.

Using mono-repo for a pattern allows you to have a single pattern publishing pipeline to publish multiple packages which saves resources and scale much more than a multi-repo strategy. For example, AWS CDK uses the mono-repo strategy for managing multiple packages in the same repository.

Code repository structure

When you create a new application pattern, the solution creates a new code repository for the pattern with a pre-initialized code repo structure based on a mono-repo. The files and directory structure of a new pattern's repo are as shown:

CloudFormation based pattern's code repo structure

```
images
packages
package.json
lerna.json
README.md
```

Resource	Description
images	• This directory provides a placeholder location to store the architecture image of the pattern. We recommend using the name **architecture.png** to provide an architecture diagram for the pattern (Storing an image is optional).

packages	• This is the root directory for storing multiple related CloudFormation templates.
	• For example, if there are two CloudForm ation templates we want to store in this repo, follow the below structure as a convention used by the solution.
	
	`images`
	`packages`
	`   -- <package-name-dir>`
	`         -- package.json`
	`         -- template`
	`             -- <cfn-stack-name>.t`
	`emplate`
	`package.json`
	`lerna.json`
	`README.md`
	
	• Make sure every package’s package.j son(packages → <package-name-dir> → package.json) has the name and version attributes defined at a minimum. This name and version are used later in the publishing pipeline to create Service Catalog products.
	
	`{`
	` "name": "<@test-cfn/pattern1>",`
	` "version": "1.0.1",`

Resource	Description
	`}`
package.json	• This is the root package.json file
lerna.json	• Lerna is a fast modern build system for managing and publishing JavaScript/ TypeScript mono-repos. • lerna.json defines the configuration for managing multiple packages in a repo.
README.md	• You can add a README file to your repositor y to share why your project is useful, what they can do with your project, and how they can use the project.
USAGE.md (optional)	• This is an optional file that can be added to show the usage of the pattern.

CDK based pattern's code repo structure

```
images
packages
   -- cdk-test-app
      -- bin
          -- cdk-test-app.ts
      -- cdk.json
      -- package.json
      -- tsconfig.json
package.json
lerna.json
README.md
```

Resource	Description
images	• This directory provides a placeholder location to store the architecture image of

Resource	Description
	the pattern. We recommend using the name **architecture.png** to provide an architecture diagram for the pattern (Storing an image is optional).

Resource	Description

packages	• This is the root directory for storing multiple npm typescript packages
	• For instance if you need to create two L3 CDK constructs called secure-s3-bucket and secure-dynamodb in this repo, follow this structure as a convention used by the solution.
	```
	images
	packages
	  -- cdk-test-app
	       -- bin
	           -- cdk-test-app.ts
	       -- cdk.json
	       -- package.json
	       -- tsconfig.json
	  -- secure-s3-bucket
	       -- lib
	           -- index.ts
	       -- test
	       -- package.json
	       -- tsconfig.json
	  -- secure-dynamodb
	       -- lib
	           -- index.ts
	       -- test
	       -- package.json
	       -- tsconfig.json
	package.json
	lerna.json
	README.md
	```
	• In order to run the automated security check on the CDK based pattern packages (which is automatically initiated on raising a Pull Request), it is important to demonstra te all the possible uses cases of the L3 CDK

Resource	Description
	constructs. For example, in the example above, use both the secure-s3-bucket and secure-dynamodb CDK constructs with all combinations of the supported parameters in packages → cdk-test-app → bin → cdk-test-app.ts. The automated security check job automatically synthesizes the AWS CDK code to CloudFormation template and runs a cfn_nag scan to generate a security report.
package.json	• This is the root package.json file.
lerna.json	• Lerna is a fast modern build system for managing and publishing JavaScript/TypeScript mono-repos. • lerna.json defined the configuration for managing multiple packages in a repo.
USAGE.md (optional)	• This is an optional file that can be added to show how the L3 CDK constructs can be used.

Security recommendations

In addition to the existing security configuration in the solution, we also recommend additional security settings as follows.

Enable Amazon Macie for Amazon S3 buckets

Amazon Macie is a fully managed data security and data privacy service that uses machine learning and pattern matching to discover and protect your sensitive data in AWS. Amazon Macie automatically provides an inventory of Amazon S3 buckets including a list of unencrypted buckets, publicly accessible buckets, and buckets shared with AWS accounts outside those you have defined in AWS Organizations. For more information, refer to the Amazon Macie page.

Enable MFA to Amazon Cognito User Pool

Multi-factor authentication (MFA) increases security for your app. It adds a *something you have* authentication factor to the *something you know* factor of user name and password. You can choose SMS text messages or time-based one-time passwords (TOTP) as second factors to sign in your users.

For more information, refer to Adding MFA to a user pool.

Enable Advanced Security mode for Amazon Cognito User Pool

You can turn on the Amazon Cognito user pool advanced security features, and customize the actions that are taken in response to different risks. Alternatively, you can use audit mode to gather metrics on detected risks without applying any security mitigations. In the audit mode, the advanced security features publish metrics to Amazon CloudWatch. For more information, refer to Adding advanced security to a user pool.

Anonymized data collection

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When invoked, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Unique ID (UUID)** - Randomly generated, unique identifier for each Application Pattern Orchestrator on AWS deployment
- **Timestamp** - Data-collection timestamp

AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy.

Opt out of operational metrics collection

To opt out of this feature, for the solution **parameters**, set sendAnonymousData parameter to **No.**

Contributors

Product

- Hafiz Saadullah
- Marc Teichtahl

Engineering

- Verinder Singh
- Frank Cao
- Deenadayaalan Thirugnanasambandam
- Van Vo Thanh
- Yang Yang
- APJ Solutions Engineering team

Technical Writing and Documentation

- Swapnil Ogale
- Suyog Sainkar
- Daniil Millwood

Security Guardian review

- Andrew Hodges

Revisions

Date	Change
November 2022	Initial release
May 2023	v1.1.0 • Added information on AWS CodeCommit as default source code repository. • Updated architecture diagrams and solution component diagrams with AWS CodeCommi t. • Updated costs table with change in solution costs to include AWS CodeCommit. • Mitigated impact caused by new default settings for S3 Object Ownership (ACLs disabled) for all new S3 buckets. For more information, refer to the CHANGELOG.md file in the GitHub repositor y.
August 2023	v1.2.0 • Added information about user group authorizations under the Security section. • Added instructions on how to update the solution. For more information, refer to the CHANGELOG.md file in the GitHub repositor y.

Notices