aws

Distributed Load Testing on AWS

Distributed Load Testing on AWS: Implementation Guide

Table of Contents

Automate the testing of your software applications at scale

Publication date: *November 2019 ([last update](): June 2024)*

Distributed Load Testing on AWS helps you automate the testing of your software applications at scale and at load to identify bottlenecks before you release your application. This solution creates and simulates thousands of connected users generating transactional records at a constant pace without the need to provision servers.

This solution leverages [Amazon Elastic Container Service (Amazon ECS) on AWS Fargate]() to deploy containers that can run all of your simulations and offers the following features:

- Deploy Amazon ECS on AWS Fargate containers that can run independently to test the load capabilities of the software being tested.
- Simulate tens of thousands of connected users, across multiple AWS Regions, generating transactional records at a continuous pace.
- Customize your application tests by creating custom [JMeter scripts]().
- Schedule load tests to either automatically begin at a future date or on recurring dates.
- Run your application load tests concurrently or run multiple tests simultaneously.

This implementation guide discusses architectural considerations and configuration steps for deploying Distributed Load Testing on AWS in the Amazon Web Services (AWS) Cloud. It includes links to an [AWS CloudFormation]() template that launches and configures the AWS services required to deploy this solution using AWS best practices for security and availability.

The guide is intended for IT infrastructure architects, administrators, and DevOps professionals who have practical experience architecting in the AWS Cloud.

Cost

You are responsible for the cost of the AWS services used while running this solution. The total cost for running this solution depends on the number of load tests run, the duration of those load tests, and the amount of data used as a part of the tests. As of this revision, the cost for running this solution with default settings in the US East (N. Virginia) Region is approximately **$30.90 per month**. The cost estimate assumes the following factors:

AWS service	Dimensions	Cost [USD]
AWS Fargate	10 on-demand tasks (using two vCPUs and 4 GB memory) running for 30 hours	$29.62
Amazon DynamoDB	1,000 on-demand write capacity units 1,000 on-demand read capacity units	$0.0015
AWS Lambda	1,000 requests 10 minutes total duration	$1.25
AWS Step Functions	1,000 state transitions	$0.025
	Total:	**$30.90 per month**

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, refer to the pricing webpage for each AWS service used in this solution.

> ⚠️ **Important**
>
> Starting in version 1.3.0, the CPU is increased to 2 vCPU and the memory is increased to 4 GB. These changes increase the estimated cost compared to previous versions of this solution. If your load tests do not require these increases to your AWS resources, you can

reduce them. For additional information, refer to the [Increase the container resources](#) section in this guide.

> **ⓘ Note**
>
> This solution provides the option to include live data when running a test. This feature requires an additional AWS Lambda function and AWS IoT Core topic that incur extra costs.

Prices are subject to change. For full details, see the pricing webpage for each AWS service you will be using in this solution.

Architecture overview

Deploying this solution with the default parameters builds the following environment in the AWS Cloud.
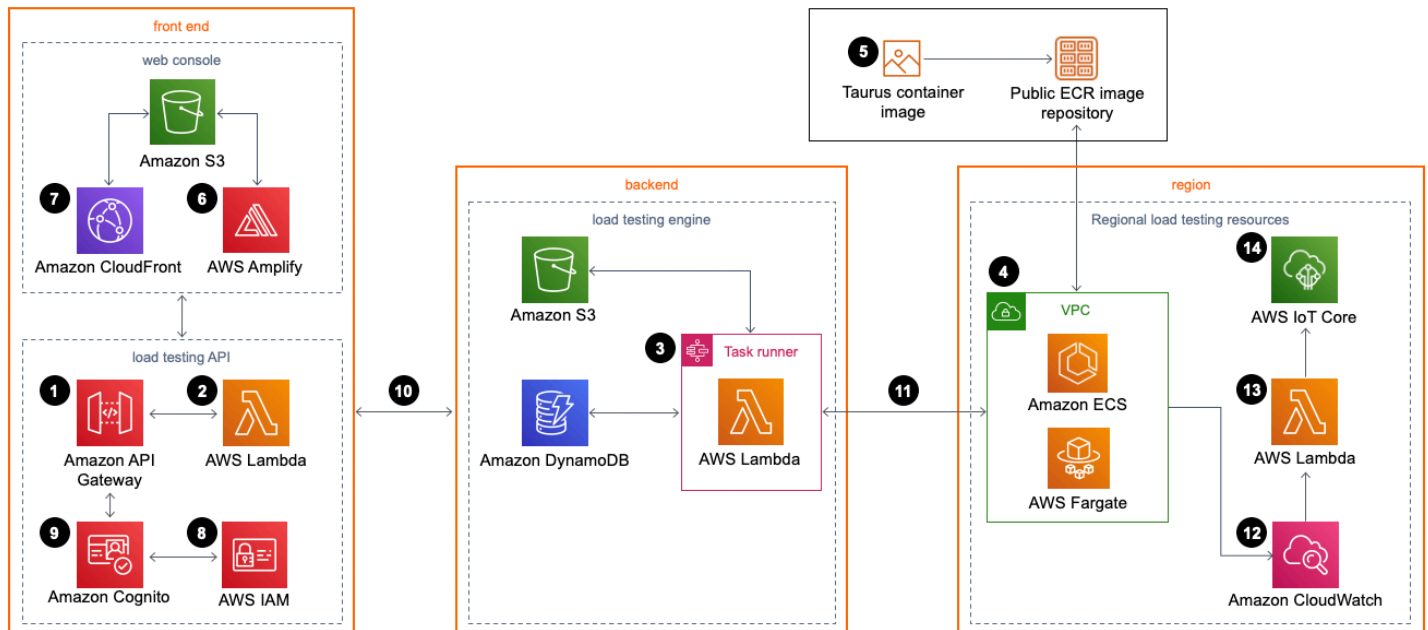
Distributed Load Testing on AWS architecture

The AWS CloudFormation template deploys the following resources:

1. A `distributed load tester` API, which leverages Amazon API Gateway to invoke the solution's microservices (AWS Lambda functions).

2. The microservices provide the business logic to manage test data and run the tests.

3. These microservices interact with Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, and AWS Step Functions to provide storage for the test scenario details and results and run test scenarios.

4. An Amazon Virtual Private Cloud (Amazon VPC) network topology is deployed containing the solution's Amazon Elastic Container Service (Amazon ECS) containers running on AWS Fargate.

5. The containers include the Taurus load testing Open Container Initiative (OCI) compliant container image which is used to generate load for testing your application's performance. Taurus is an open-source test automation framework. The container image is hosted by AWS in an Amazon Elastic Container Registry (Amazon ECR) public repository. For more information about the ECR image repository, refer to Container image customization.

6. A web console powered by [AWS Amplify](#) is deployed it into an Amazon S3 bucket configured for static web hosting.

7. [Amazon CloudFront](#) provides secure, public access to the solution's website bucket contents.

8. During initial configuration, this solution also creates a default solution administrator role (IAM role) and sends an access invite to a customer-specified user email address.

9. An [Amazon Cognito](#) user pool manages user access to the console and the `distributed load tester` API.

10. After you deploy this solution, you can use the web console to create a test scenario that defines a series of tasks.

11. The microservices use this test scenario to run Amazon ECS on AWS Fargate tasks in the Regions specified.

12. In addition to storing the results in Amazon S3 and DynamoDB, once the test is complete the output is logged in [Amazon CloudWatch](#).

13. If you select the live data option, the solution sends the Amazon CloudWatch logs for the AWS Fargate tasks to a Lambda function during the test, for each Region in which the test was run.

14. The Lambda function then publishes the data to the corresponding topic in [AWS IoT Core](#) in the Region where the main stack was deployed. The web console subscribes to the topic, and you can see the data while the test runs in the web console.

Solution components

The Distributed Load Testing on AWS solution consists of two high-level components, a front end and a backend.

Front end

The front end consists of a load testing API and web console you use to interact with the solution's backend.

Load testing API

Distributed Load Testing on AWS configures Amazon API Gateway to host the solution's RESTful API. Users can interact with testing data securely through the included web console and RESTful API. The API acts as a "front door" for access to testing data stored in Amazon DynamoDB. You can also use the APIs to access any extended functionality you build into the solution.

This solution takes advantage of the user authentication features of Amazon Cognito user pools. After successfully authenticating a user, Amazon Cognito issues a JSON web token that is used to allow the console to submit requests to the solution's APIs (Amazon API Gateway endpoints). HTTPS requests are sent by the console to the APIs with the authorization header that includes the token.

Based on the request, API Gateway invokes the appropriate AWS Lambda function to perform the necessary tasks on the data stored in the DynamoDB tables, store test scenarios as JSON objects in Amazon S3, retrieve Amazon CloudWatch metrics images, and submit test scenarios to the AWS Step Functions state machine.

For more information on the solution's API, refer to the [Distributed load testing API](#) section of this guide.

Web console

This solution includes a web console that you can use to configure and run tests, monitor running tests, and view detailed test results. The console is a ReactJS application hosted in Amazon S3 and accessed through Amazon CloudFront. The application leverages AWS Amplify to integrate with Amazon Cognito to authenticate users. The web console also contains an option to view live data for a running test, in which it subscribes to the corresponding topic in AWS IoT Core.

The web console is designed to demonstrate how you can interact with this load testing solution. In a production environment, we recommend customizing the web console to meet your specific needs or building your own console.

The web console URL is the CloudFront distribution domain name which can be found in the CloudFormation outputs as **Console**. After you launch the CloudFormation template, you will also receive an email that contains the web console URL and the one-time password to log into it.

Backend

The backend consists of a container image pipeline and load testing engine you use to generate load for the tests. You interact with the backend through the front end. Additionally, Amazon ECS on AWS Fargate tasks launched for each test are tagged with a unique test identifier (ID). These test ID tags can be used to help you monitor costs for this solution. For additional information, refer to [User-Defined Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*.

Container image pipeline

This solution leverages a container image of the [Taurus](#) load testing framework. This image is hosted in an Amazon Elastic Container Registry (Amazon ECR) public repository. The image is used to run tasks in the Amazon ECS on AWS Fargate cluster.

For more information, refer to the [Container image customization](#) section of this guide.

Testing infrastructure

In addition to the main template, the solution creates a secondary template to launch the required resources to run tests in multiple Regions. The template is stored in Amazon S3, and a link to the template is provided in the web console. The secondary templates create a VPC, an AWS Fargate cluster, and a Lambda function for processing live data.

For more information on how to launch a secondary Region, refer to the [Multi-Region deployment](#) section of this guide.

Load testing engine

The Distributed Load Testing solution uses Amazon Elastic Container Service (Amazon ECS) and AWS Fargate to simulate thousands of connected users, across multiple Regions, generating a select number of transactions per second.

You define the parameters for the tasks that will be run as part of the test using the included web console. The solution uses these parameters to generate a JSON test scenario and stores it in Amazon S3.

An AWS Step Functions state machine runs and monitors Amazon ECS tasks in an AWS Fargate cluster. The AWS Step Functions state machine includes an `ecr-checker` AWS Lambda function, a `task-status-checker` AWS Lambda function, a `task-runner` AWS Lambda function, a `task-canceler` AWS Lambda function, and a `results-parser` AWS Lambda function. For more information on the workflow, refer to the [Test workflow](#) section of this guide. For more information on test results, refer to the [Test results](#) section of this guide. For more information on the test cancellation workflow, refer to the [Test cancellation workflow](#) section of this guide.

If you select live data, the solution initiates a `real-time-data-publisher` Lambda function in each Region by the CloudWatch logs that correspond to the Fargate tasks in that Region. The solution then processes and publishes the data to a topic in AWS IoT Core within the Region where you launched the main stack. For more information, refer to the *[Live data](#)* section of this guide.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared model can reduce your operational burden as AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the services operate. For more information about security on AWS, visit AWS Cloud Security.

IAM roles

AWS Identity and Access Management (IAM) roles allow customers to assign granular access policies and permissions to services and users on AWS. This solution creates several IAM roles, including roles that grant the solution's AWS Lambda function access to the other AWS services used in this solution.

Amazon CloudFront

This solution deploys a static website hosted in an Amazon Simple Storage Service (Amazon S3) bucket. To help reduce latency and improve security, this solution includes an Amazon CloudFront distribution with an origin access identity, which is a special CloudFront user that helps provide secure, public access to the solution's website bucket contents. For more information, see Restricting Access to Amazon S3 Content by Using an Origin Access Identity.

AWS Fargate security group

By default, this solution opens the outbound rule of the AWS Fargate security group to the public. If you want to block AWS Fargate from sending traffic to everywhere, then change the outbound rule to a specific Classless Inter-Domain Routing (CIDR).

This security group also includes an inbound rule that allows local traffic on port 50,000 to any source that belongs to the same security group. This is used to allow the containers to communicate with one another.

Network stress test

You are responsible for using this solution under the Network Stress Test policy. This policy covers situations such as if you are planning on running high volume network tests directly

from your Amazon EC2 instances to other locations such as other Amazon EC2 instances, AWS properties/services, or external endpoints. These tests are sometimes called stress tests, load tests, or gameday tests. Most customer testing will not fall under this policy, however, refer to this policy if you believe you will be generating traffic that sustains, in aggregate, for more than 1 minute, over 1 Gbps (1 billion bits per second) or over 1 Gpps (1 billion packets per second).

Restricting access to the public user interface

To restrict access to the public-facing user interface beyond the authentication and authorization mechanisms provided by IAM and Amazon Cognito, use the AWS WAF (web application firewall) Security Automations solution.

This solution automatically deploys a set of AWS WAF rules that filter common web-based attacks. Users can select from preconfigured protective features that define the rules included in an AWS WAF web access control list (web ACL).

Design considerations

Supported applications

This solution supports cloud-based applications, and on-premises applications as long as you have a network connection from your AWS account to your application. The solution supports APIs that use either HTTP or HTTPS. You also have control over the HTTP request headers, so you can add authorization or custom headers to pass tokens or API keys.

JMeter script support

When creating a test scenario using this solution's user interface (UI), you can use a JMeter test script. After selecting the JMeter script file, it is uploaded to the *<stack-name>*-`scenariosbucket` Amazon Simple Storage Service (Amazon S3) bucket. When Amazon Elastic Container Service (Amazon ECS) tasks are running, the JMeter script downloads from the *<stack-name>*-`scenariosbucket` Amazon S3 bucket and the test runs.

If you have JMeter input files, you can zip the input files together with the JMeter script. You can choose the zip file when you create a test scenario.

If you would like to include plugins, any `.jar` files that are included in a `/plugins` subdirectory in the bundled zip file will be copied to the JMeter extensions directory and be available for load testing.

> **ⓘ Note**
>
> If you include JMeter input files with your JMeter script file, you must include the relative path of the input files in your JMeter script file. In addition, the input files must be at the relative path. For example, when your JMeter input files and script file are in the `/home/user` directory and you refer to the input files in the JMeter script file, the path of input files must be `./INPUT_FILES`. If you use `/home/user/INPUT_FILES` instead, the test will fail because it will not be able to find the input files.
>
> If you include JMeter plugins, the `.jar` files must be bundled in a subdirectory named `/plugins` within the root of the zip file. Relative to the root of the zip file, the path to the jar files must be `./plugins/BUNDLED_PLUGIN.jar`.

For more information about how to use JMeter scripts, refer to [JMeter User's Manual](#).

Scheduling tests

You can schedule tests to run at a future date or use the **Run Now** option. You can schedule a test as a one-time run in the future or set up a recurring test in which you specify a first run date, and planned recurrence. The options for recurrence include: daily, weekly, bi-weekly, and monthly. For more information on how scheduling works, refer to the [Test scheduling workflow](#) section of this guide.

Load testing quotas

The maximum number of tasks that can be running in Amazon ECS using the AWS Fargate launch type is based on the vCPU size of the tasks. The default task size in Distributed Load Testing on AWS is 2 vCPU. To see the current default quotas, refer to [Amazon ECS service quotas](#). Current account quotas may differ from the listed quotas. To check quotas specific to an account, check the service quota for Fargate on-demand vCPU resource count in the AWS Management Console. For instructions on how to request an increase, refer to [AWS service quotas](#) in the *AWS General Reference Guide*.

The Taurus load testing container image does not limit concurrent connections per task, but that does not mean that it can support an unlimited number of users. To determine the number of concurrent users the containers can generate for a test, refer to [Determine the number of users](#) section of this guide.

> ⓘ **Note**
>
> The recommended limit for concurrent users based on default settings is 200 users.

Concurrent tests

This solution includes an Amazon CloudWatch dashboard for each test and displays the combined output of all tasks running for that test in the Amazon ECS cluster in real-time. The CloudWatch dashboard displays the average response time, the number of concurrent users, the number of successful requests, and the number of failed requests. Each metric is aggregated by the second, and the dashboard is updated every minute.

Amazon EC2 testing policy

You do not need approval from AWS to run load tests using this solution as long as your network traffic stays below 1 Gbps. If your test will generate more than 1 Gbps, contact AWS. For more information, refer to the [Amazon EC2 Testing Policy](#).

Amazon CloudFront load testing policy

If you plan on load testing a CloudFront endpoint, refer to the [load testing guidelines](#) in the *Amazon CloudFront Developer Guide*. We also recommended spreading the traffic across multiple tasks and Regions. Provide at least 30 minutes of ramp-up time for the load test. For load tests sending more than 500,000 requests per second or demanding more than 300 Gbps data, we recommend first obtaining a pre-approval for sending the traffic. CloudFront may throttle unapproved load test traffic that impacts CloudFront service availability.

User management

During initial configuration, you provide a username and email address that Amazon Cognito uses to grant you access to the solution's web console. The console does not provide user administration. To add additional users, you must use the Amazon Cognito console. For more information, refer to [Managing Users in User Pools](#) in the *Amazon Cognito Developer Guide*.

Regional deployment

This solution uses Amazon Cognito which is available in specific AWS Regions only. Therefore, you must deploy this solution in a region where Amazon Cognito is available. For the most current service availability by Region, refer to the [AWS Regional Services List](#).

AWS CloudFormation template

This solution uses AWS CloudFormation to automate the deployment of Distributed Load Testing on AWS. It includes the following AWS CloudFormation template, which you can download before deployment:

View template

distrib

load-testing-on-aws.template - Use this template to launch the solution and all associated components. The default configuration deploys Amazon Elastic Container Service (Amazon ECS), AWS Fargate, Amazon Virtual Private Cloud (Amazon VPC), AWS Lambda, Amazon Simple Storage Service (Amazon S3), AWS Step Functions, Amazon DynamoDB, Amazon CloudWatch Logs, Amazon API Gateway, Amazon Cognito, AWS Identity and Access Management (IAM), and Amazon CloudFront, but you can also customize the template based on your specific network needs.

Automated deployment

> ⚠️ **Important**
>
> If you are updating the stack from a version prior to v3.2.6 to the latest version, read this section before updating the stack.

Before you launch the automated deployment, review the architecture and other considerations discussed in this guide. Follow the step-by-step instructions in this section to configure and deploy Distributed Load Testing on AWS into your account.

Time to deploy: Approximately 15 minutes

Launch the stack

> ⚠️ **Important**
>
> 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. AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy.
>
> To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your template and deploy the solution. For more information, refer to the Anonymized data collection section of this guide.

This automated AWS CloudFormation template deploys Distributed Load Testing on AWS.

> ℹ️ **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 click the button below to launch the
   `distributed-load-testing-on-aws` AWS CloudFormation template.

   Launch solution

   Alternatively, you can also download the template as a starting point for your own
   implementation.

2. The template is launched in the US East (N. Virginia) Region by default. To launch this solution
   in a different AWS Region, use the region selector in the console navigation bar.

   > ⓘ **Note**
   >
   > This solution uses Amazon Cognito, which is currently available in specific AWS Regions
   > only. Therefore, you must launch this solution in an AWS Region where Amazon
   > Cognito is available. For the most current service availability by Region, refer to the
   > AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL shows 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 the template and modify them as necessary.
   This solution uses the following default values.

Parameter	Default	Description
Admin name	*<Requires input>*	User name for the initial solution administrator.
Admin email	*<Requires input>*	Email address of the administrator user. After launch, an email will be sent to this address with console login instructions.

Parameter	Default	Description
Existing VPC ID	<Optional input>	If you have a VPC that you want to use and is already created, enter the ID of an existing VPC in the same Region where the stack was deployed. For example, `vpc-1a2b3c4d5e6f` .
First existing subnet	<Optional input>	The ID of the first subnet within your existing VPC. This subnet needs a route to the internet to pull the container image for running tests. For example, `subnet-7h8i9j0k` .
Second existing subnet	<Optional input>	The ID of the second subnet within the existing VPC. This subnet needs a route to the internet to pull the container image for running tests. For example, `subnet-1x2y3z` .
AWS Fargate VPC CIDR Block	`192.168.0.0/16`	If you do not provide values for an existing VPC, the CIDR block for the solution-created Amazon VPC contains the IP address for AWS Fargate.

Parameter	Default	Description
AWS Fargate Subnet A CIDR Block	`192.168.0.0/20`	If you do not provide values for an existing VPC, the CIDR block contains the IP address for the Amazon VPC subnet A.
AWS Fargate Subnet B CIDR Block	`192.168.16.0/20`	If you do not provide values for an existing VPC, the CIDR block contains the IP address for the Amazon VPC subnet B.
AWS Fargate Security Group CIDR Block	`0.0.0.0/0`	CIDR block that restricts Amazon ECS container outbound access.

6. Choose **Next**.

7. On the **Configure stack options** page, choose **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.

> ⓘ **Note**
>
> In addition to the primary AWS Lambda function, this solution includes the `custom-resource` Lambda function, which runs only during initial configuration or when resources are updated or deleted.
> When running this solution, the `custom-resource` Lambda function is inactive. However, do not delete this function as it is necessary to manage associated resources.

Multi-Region deployment

Time to deploy: Approximately five minutes

You can run tests across multiple Regions. When you deploy the Distributed Load Testing solution, it creates three Amazon S3 buckets. The solution creates a secondary regional stack and stores it in the Amazon S3 `scenarios` bucket.

> ⓘ **Note**
>
> The bucket naming convention is <*stack-name*>-dlttestrunnerstoragedltscenariosbucket<*[0-9][0-9]..-<[0-9][0-9]..*> with the keyword scenarios in the bucket name which you can locate by navigating to the S3 console, then **Buckets**.

To run a multi-Region deployment, you must deploy the regional CloudFormation template, which is stored in the Amazon S3 `scenarios` bucket, in the Regions you want to run the test. You can install the regional template by doing the following:

1. In the solution's web console, navigate to **Manage Regions** in the top menu.
2. Use the clipboard icon to copy the CloudFormation template link in Amazon S3.
3. Sign in to the [AWS CloudFormation console](#) and select the correct Region.
4. On the **Create stack** page, verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.
5. On the **Specify stack details** page, assign a name to your solution stack.
6. Under **Parameters**, review the parameters for the template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
Existing VPC ID	<Optional input>	If you have a VPC that you want to use and is already created, enter the ID of an existing VPC in the same Region where the stack

Parameter	Default	Description
		was deployed. For example, `vpc-1a2b3c4d5e6f` .
First existing subnet	<Optional input>	The ID of the first subnet within your existing VPC. This subnet needs a route to the internet to pull the container image for running tests. For example, `subnet-7h8i9j0k` .
Second existing subnet	<Optional input>	The ID of the second subnet within the existing VPC. This subnet needs a route to the internet to pull the container image for running tests. For example, `subnet-1x2y3z` .
AWS Fargate VPC CIDR Block	`192.168.0.0/16`	If you do not provide values for an existing VPC, the CIDR block for the solution-created Amazon VPC contains the IP address for AWS Fargate.
AWS Fargate Subnet A CIDR Block	`192.168.0.0/20`	If you do not provide values for an existing VPC, the CIDR block contains the IP address for the Amazon VPC subnet A.

Parameter	Default	Description
AWS Fargate Subnet B CIDR Block	`192.168.16.0/20`	If you do not provide values for an existing VPC, the CIDR block contains the IP address for the Amazon VPC subnet B.
AWS Fargate Security Group CIDR Block	`0.0.0.0/0`	CIDR block that restricts Amazon ECS container outbound access.

7. Choose **Next**.

8. On the **Configure stack options** page, choose **Next**.

9. On the **Review** page, review and confirm the settings. Be sure to check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.

10. 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 five minutes.

When the Regions have been successfully deployed, they appear in the web console. When you create a test, the new Region will be listed in the **Manage Regions** modal. You can use this Region in a test by selecting it upon test creation. The solution creates a DynamoDB item for each Region launched in the scenarios table, which contains the necessary information regarding the testing resources in that Region. You can sort test results in the web console by Region. Due to API constraints, you can only view the aggregate results of all Regions in a multi-Region test by graphing them in Amazon CloudWatch metrics. You can find the source code for the graph in the test results once the test has finished.

> (i) **Note**
>
> You can launch the regional stack without the web console. Obtain a link to the regional template in the Amazon S3 `scenarios` bucket and provide it as the source when launching the regional stack in the required Region. Alternatively, you can download the template and upload it as the source for the Region you want.

Additional resources

AWS services	
[Amazon Elastic Container Service](#)	[Amazon Cognito](#)
[Amazon Elastic Container Registry](#)	[Amazon CloudFront](#)
[AWS Fargate](#)	[Amazon Virtual Private Cloud](#)
[AWS Lambda](#)	[AWS Service Catalog](#)
[Amazon Simple Storage Service](#)	[AWS Systems Manager](#)
[Amazon DynamoDB](#)	[AWS Identity and Access Management](#)
[Amazon CloudWatch](#)	[AWS Amplify](#)
[Amazon CloudWatch Events](#)	[AWS CloudFormation](#)
[Amazon API Gateway](#)	[AWS IoT Core](#)
[AWS Step Functions](#)	

Other resources

- [Taurus](#)

Monitoring the solution with Service Catalog AppRegistry

The Distributed Load Testing on AWS solution includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both 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, such as deployment status, CloudWatch alarms, resource configurations, and operational issues.

The following figure depicts an example of the application view for the Distributed Load Testing on AWS stack in Application Manager.

Distributed Load Testing 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

1. Open 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**.
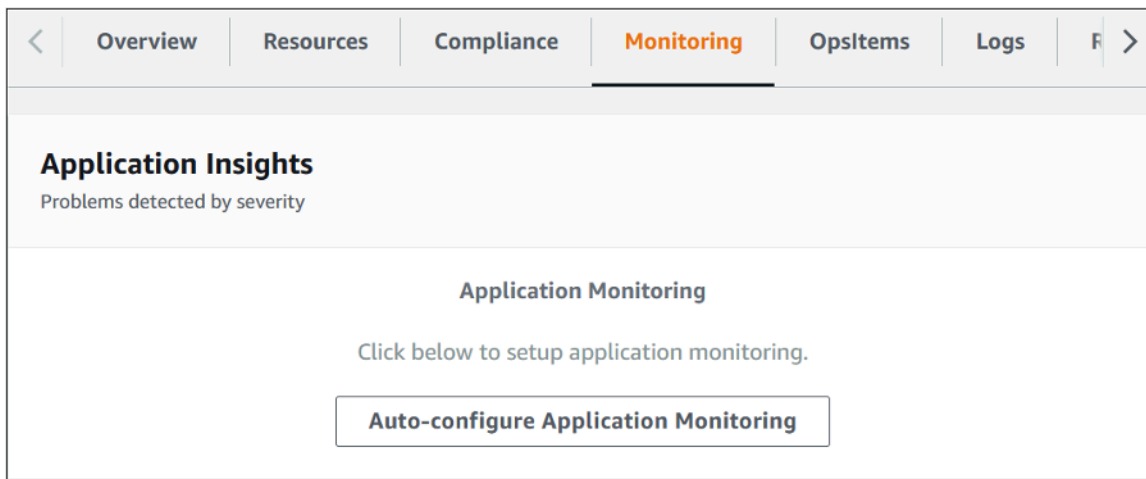
6. Monitoring for your applications is now activated and the following status box appears:
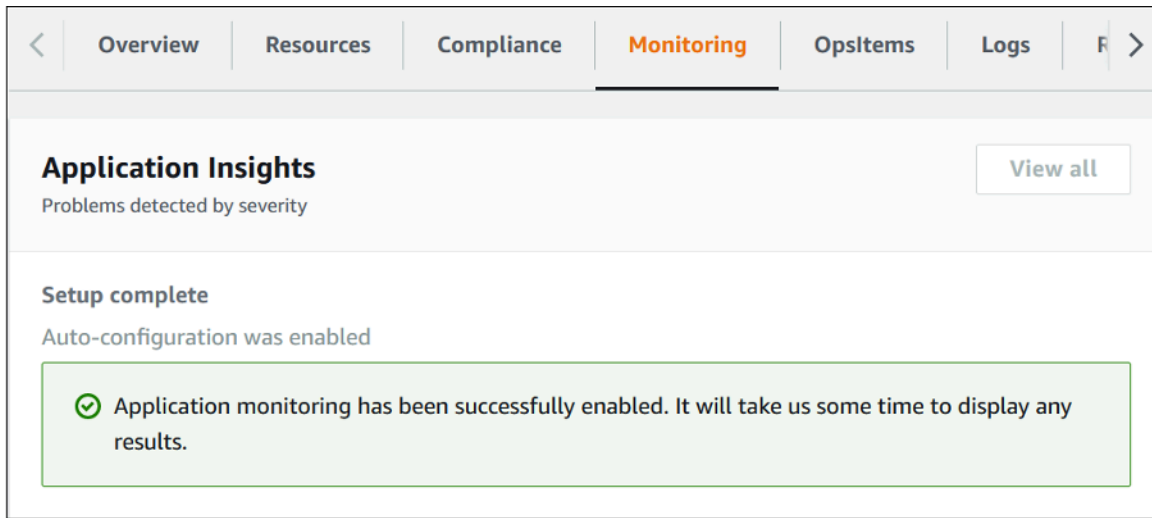
Activate AWS Cost Explorer

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. 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 left navigation menu, select **Cost Explorer**.
3. On the **Welcome to Cost Explorer** page, choose **Launch Cost Explorer**.

The activation process can take up to 24 hours to complete. Once activated, you can open the Cost Explorer user interface to further analyze cost data for the solution.

Activate cost allocation tags associated with the solution

After you activate Cost Explorer, you must activate a cost allocation tag 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.

Confirm cost tags associated with the solution

After you activate cost allocation tags associated with the solution, you must confirm the cost allocation tags to see the costs for this solution. To confirm cost allocation tags:

1. Sign in to the [Systems Manager console](#).
2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, choose the application name for this solution and select it.
4. In the **Overview** tab, in **Cost**, select **Add user tag**.
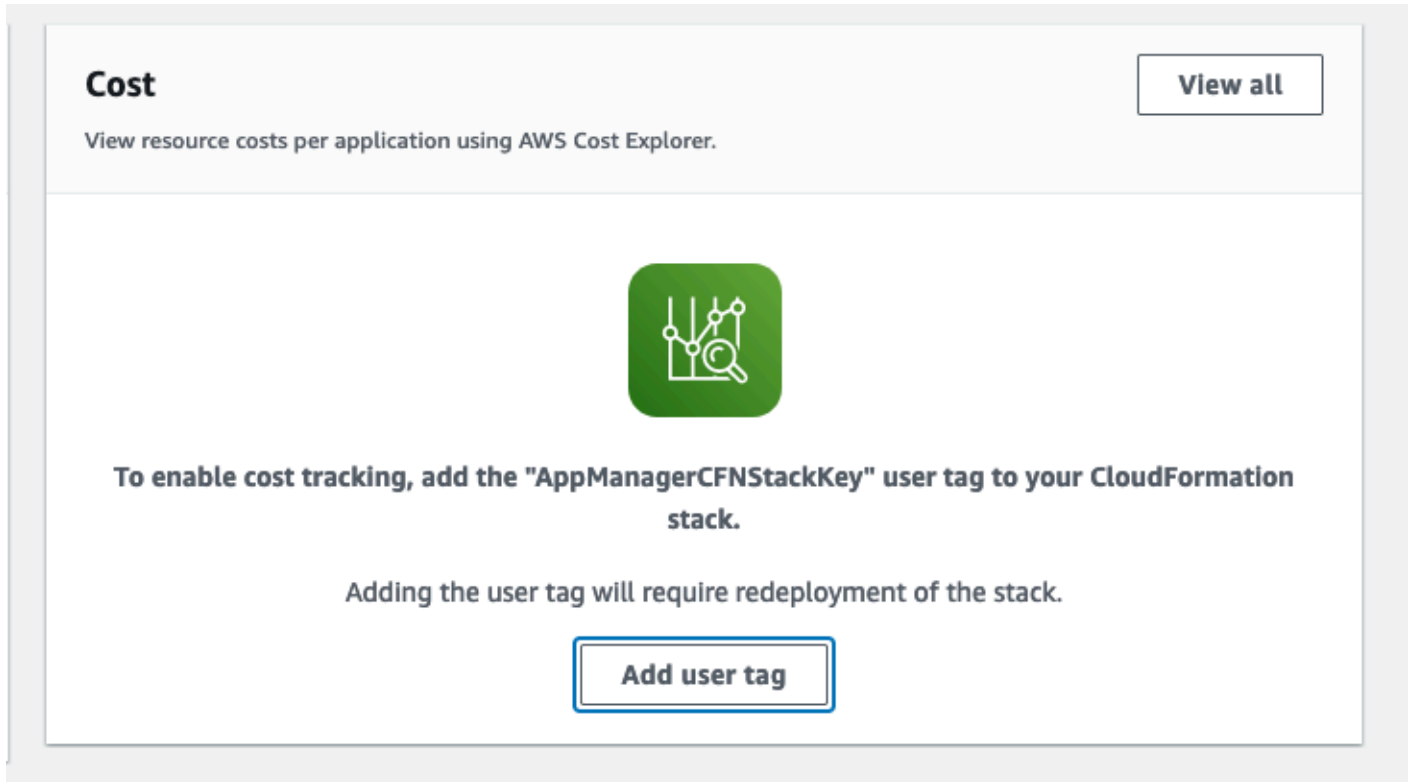
5. On the **Add user tag** page, enter `confirm`, then select **Add user tag**.

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

Container image customization

This solution uses a public Amazon Elastic Container Registry (Amazon ECR) image repository managed by AWS to store the Taurus image that is used to run the configured tests. If you want to customize the container image, you can rebuild and push the image into an ECR image repository in your own AWS account.

If you want to customize this solution, you can use the default container image or, edit this container to fit your needs. If you customize the solution, use the following code sample to declare the environment variables before building your customized solution.

```bash
#!/bin/bash

export REGION=aws-region-code # the AWS region to launch the solution (e.g. us-east-1)
export BUCKET_PREFIX=my-bucket-name # prefix of the bucket name without the region code
export BUCKET_NAME=$BUCKET_PREFIX-$REGION # full bucket name where the code will reside
export SOLUTION_NAME=my-solution-name
export VERSION=my-version # version number for the customized code
export PUBLIC_ECR_REGISTRY=public.ecr.aws/awssolutions/distributed-load-testing-on-
aws-load-tester # replace with the container registry and image if you want to use a
 different container image export PUBLIC_ECR_TAG=v3.1.0 # replace with the container
 image tag if you want to use a different container image
```

If you choose to customize the container image, you can host it in either a private image repository, or a public image repository in your AWS account. The image resources are in the `deployment/ecr/distributed-load-testing-on-aws-load-tester` directory, located in the code base.

You can build and push the image to the host destination.

- For private Amazon ECR repositories and images, refer to [Amazon ECR private repositories](#) and [private images](#) in the *Amazon ECR User Guide*.
- For public Amazon ECR repositories and images, refer to [Amazon ECR public repositories](#) and [public images](#) in the *Amazon ECR Public User Guide*.

Once you create your own image, you can declare the following environment variables before building your customized solution.

```bash
#!/bin/bash
```

```
export PUBLIC_ECR_REGISTRY=YOUR_ECR_REGISTRY_URI # e.g. YOUR_ACCOUNT_ID.dkr.ecr.us-
east-1.amazonaws.com/YOUR_IMAGE_NAME
export PUBLIC_ECR_TAG=YOUR_ECR_TAG # e.g. latest, v2.0.0
```

The following example shows the container file.

```
FROM blazemeter/taurus
# taurus includes python and pip
RUN /usr/bin/python3 -m pip install --upgrade pip
RUN pip install --no-cache-dir awscli
RUN apt-get -y install xmlstarlet bc procps
# Taurus working directory  = /bzt-configs
ADD ./load-test.sh /bzt-configs/
ADD ./*.jar /bzt-configs/
ADD ./*.py /bzt-configs/
RUN chmod 755 /bzt-configs/load-test.sh
RUN chmod 755 /bzt-configs/ecslistener.py
RUN chmod 755 /bzt-configs/ecscontroller.py
ENTRYPOINT ["./load-test.sh"]
```

In addition to a container file, the directory contains the following bash script that downloads the
test configuration from Amazon S3 before running the Taurus program.

```
#!/bin/bash

# set a uuid for the results xml file name in S3
UUID=$(cat /proc/sys/kernel/random/uuid)
pypid=0
echo "S3_BUCKET:: ${S3_BUCKET}"
echo "TEST_ID:: ${TEST_ID}"
echo "TEST_TYPE:: ${TEST_TYPE}"
echo "FILE_TYPE:: ${FILE_TYPE}"
echo "PREFIX:: ${PREFIX}"
echo "UUID:: ${UUID}"
echo "LIVE_DATA_ENABLED:: ${LIVE_DATA_ENABLED}"

sigterm_handler() {
  if [ $pypid -ne 0 ]; then
    echo "container received SIGTERM."
    kill -15 $pypid
    wait $pypid
    exit 143 #128 + 15
  fi
```

```
}
trap 'sigterm_handler' SIGTERM

echo "Download test scenario"
aws s3 cp s3://$S3_BUCKET/test-scenarios/$TEST_ID-$AWS_REGION.json test.json

# download JMeter jmx file
if [ "$TEST_TYPE" != "simple" ]; then
  # Copy *.jar to JMeter library path. See the Taurus JMeter path: https://
gettaurus.org/docs/JMeter/
  JMETER_LIB_PATH=`find ~/.bzt/jmeter-taurus -type d -name "lib"`
  echo "cp $PWD/*.jar $JMETER_LIB_PATH"
  cp $PWD/*.jar $JMETER_LIB_PATH

  if [ "$FILE_TYPE" != "zip" ]; then
    aws s3 cp s3://$S3_BUCKET/public/test-scenarios/$TEST_TYPE/$TEST_ID.jmx ./
  else
    aws s3 cp s3://$S3_BUCKET/public/test-scenarios/$TEST_TYPE/$TEST_ID.zip ./
    unzip $TEST_ID.zip
    # only looks for the first jmx file.
    JMETER_SCRIPT=`find . -name "*.jmx" | head -n 1`
    if [ -z "$JMETER_SCRIPT" ]; then
      echo "There is no JMeter script in the zip file."
      exit 1
    fi

    sed -i -e "s|$TEST_ID.jmx|$JMETER_SCRIPT|g" test.json

    # copy bundled plugin jars to jmeter extension folder to make them available to
 jmeter
    BUNDLED_PLUGIN_DIR=`find $PWD -type d -name "plugins" | head -n 1`
    # attempt to copy only if a /plugins folder is present in upload
    if [ -z "$BUNDLED_PLUGIN_DIR" ]; then
      echo "skipping plugin installation (no /plugins folder in upload)"
    else
      # ensure the jmeter extensions folder exists
      JMETER_EXT_PATH=`find ~/.bzt/jmeter-taurus -type d -name "ext"`
      if [ -z "$JMETER_EXT_PATH" ]; then
        # fail fast - if plugins bundled they will be needed for the tests
        echo "jmeter extension path (~/.bzt/jmeter-taurus/**/ext) not found - cannot
 install bundled plugins"
        exit 1
      fi
      cp -v $BUNDLED_PLUGIN_DIR/*.jar $JMETER_EXT_PATH
```

```
        fi
    fi
fi

#Download python script
if [ -z "$IPNETWORK" ]; then
    python3 -u $SCRIPT $TIMEOUT &
    pypid=$!
    wait $pypid
    pypid=0
else
    python3 -u $SCRIPT $IPNETWORK $IPHOSTS
fi

echo "Running test"
stdbuf -i0 -o0 -e0 bzt test.json -o modules.console.disable=true | stdbuf -i0 -o0 -e0
 tee -a result.tmp | sed -u -e "s|^|$TEST_ID $LIVE_DATA_ENABLED |"
CALCULATED_DURATION=`cat result.tmp | grep -m1 "Test duration" | awk -F ' ' '{ print
 $5 }' | awk -F ':' '{ print ($1 * 3600) + ($2 * 60) + $3 }'`

# upload custom results to S3 if any
# every file goes under $TEST_ID/$PREFIX/$UUID to distinguish the result correctly
if [ "$TEST_TYPE" != "simple" ]; then
  if [ "$FILE_TYPE" != "zip" ]; then
    cat $TEST_ID.jmx | grep filename > results.txt
  else
    cat $JMETER_SCRIPT | grep filename > results.txt
  fi
  sed -i -e 's/<stringProp name="filename">//g' results.txt
  sed -i -e 's/<\/stringProp>//g' results.txt
  sed -i -e 's/ //g' results.txt

  echo "Files to upload as results"
  cat results.txt

  files=(`cat results.txt`)
  for f in "${files[@]}"; do
    p="s3://$S3_BUCKET/results/$TEST_ID/JMeter_Result/$PREFIX/$UUID/$f"
    if [[ $f = /* ]]; then
      p="s3://$S3_BUCKET/results/$TEST_ID/JMeter_Result/$PREFIX/$UUID$f"
    fi

    echo "Uploading $p"
    aws s3 cp $f $p
```

```
    done
 fi


 if [ -f /tmp/artifacts/results.xml ]; then
   echo "Validating Test Duration"
   TEST_DURATION=`xmlstarlet sel -t -v "/FinalStatus/TestDuration" /tmp/artifacts/
 results.xml`

   if (( $(echo "$TEST_DURATION > $CALCULATED_DURATION" | bc -l) )); then
     echo "Updating test duration: $CALCULATED_DURATION s"
     xmlstarlet ed -L -u /FinalStatus/TestDuration -v $CALCULATED_DURATION /tmp/
 artifacts/results.xml
   fi

   echo "Uploading results, bzt log, and JMeter log, out, and err files"
   aws s3 cp /tmp/artifacts/results.xml s3://$S3_BUCKET/results/${TEST_ID}/${PREFIX}-
 ${UUID}-${AWS_REGION}.xml
   aws s3 cp /tmp/artifacts/bzt.log s3://$S3_BUCKET/results/${TEST_ID}/bzt-${PREFIX}-
 ${UUID}-${AWS_REGION}.log
   aws s3 cp /tmp/artifacts/jmeter.log s3://$S3_BUCKET/results/${TEST_ID}/jmeter-
 ${PREFIX}-${UUID}-${AWS_REGION}.log
   aws s3 cp /tmp/artifacts/jmeter.out s3://$S3_BUCKET/results/${TEST_ID}/jmeter-
 ${PREFIX}-${UUID}-${AWS_REGION}.out
   aws s3 cp /tmp/artifacts/jmeter.err s3://$S3_BUCKET/results/${TEST_ID}/jmeter-
 ${PREFIX}-${UUID}-${AWS_REGION}.err
 else
   echo "An error occurred while the test was running."
 fi
```

In addition to the [Dockerfile](#) and the bash script, two Python scripts are also included in the directory. Each task runs a Python script from within the bash script. The worker tasks run the `ecslistener.py` script, while the leader task will run the `ecscontroller.py` script. The `ecslistener.py` script creates a socket on port 50000 and waits for a message. The `ecscontroller.py` script connects to the socket and sends the start test message to the worker tasks, which allows them to start simultaneously.

Test workflow

The following detailed breakdown shows the steps involved in running a test scenario.
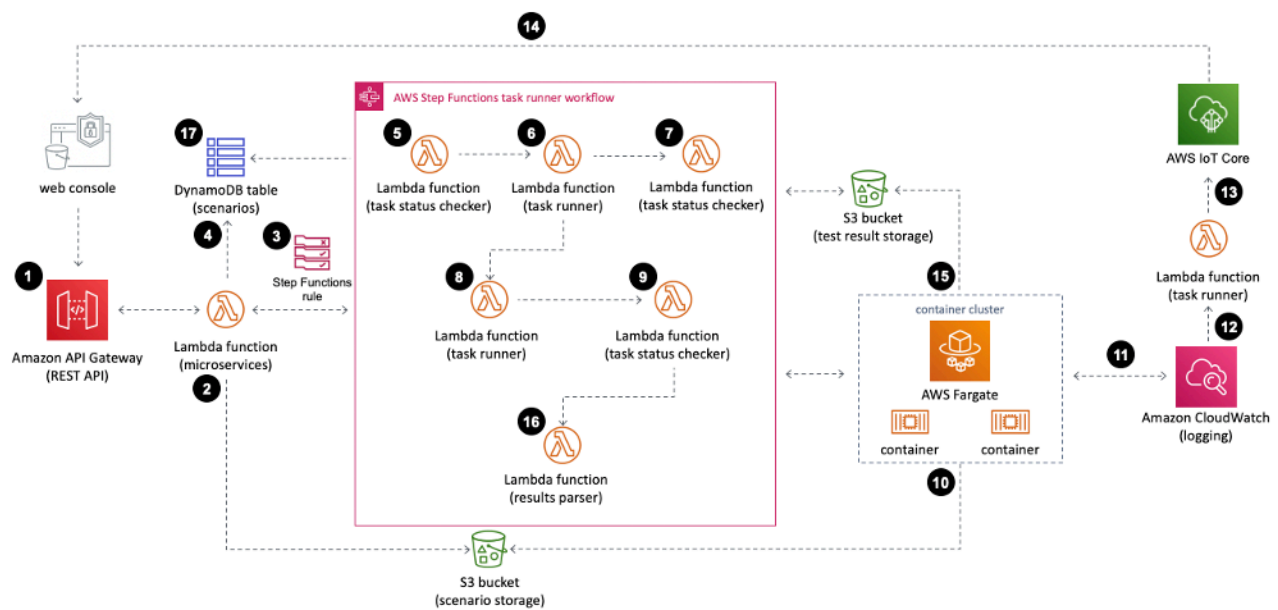
Test workflow

1. You use the web console to submit a test scenario that includes the configuration details to the solution's API.

2. The test scenario configuration is uploaded to the Amazon Simple Storage Service (Amazon S3) as a JSON file (s3://*<bucket-name>*/test-scenarios/*<$TEST_ID>*/*<$TEST_ID>*.json).

3. An AWS Step Functions state machine runs using the test ID, task count, test type, and file type as the AWS Step Functions state machine input. If the test is scheduled, it will first create a CloudWatch Events rule, which triggers AWS Step Functions on the specified date. For more details on the scheduling workflow, refer to the [Test scheduling workflow](#) section of this guide.

4. Configuration details are stored in the `scenarios` Amazon DynamoDB table.

5. In the AWS Step Functions task runner workflow, the `task-status-checker` AWS Lambda function checks if Amazon Elastic Container Service (Amazon ECS) tasks are already running for the same test ID. If tasks with the same test ID are found running, it causes an error. If there are no Amazon ECS tasks running in the AWS Fargate cluster, the function returns the test ID, task count, and test type.

6. The `task-runner` AWS Lambda function gets the task details from the previous step and runs the Amazon ECS worker tasks in the AWS Fargate cluster. The Amazon ECS API uses the

RunTask action to run the worker tasks. These worker tasks are launched and then wait for a start message from the leader task in order to begin the test. The RunTask action is limited to 10 tasks per definition. If your task count is more than 10, the task definition will run multiple times until all worker tasks have been started. The function also generates a prefix to distinguish the current test in the results-parser AWS Lambda function.

7. The task-status-checker AWS Lambda function checks if all the Amazon ECS worker tasks are running with the same test ID. If tasks are still provisioning, it waits for one minute and checks again. Once all Amazon ECS tasks are running, it returns the test ID, task count, test type, all task IDs and prefix and passes it to the task-runner function.

8. The task-runner AWS Lambda function runs again, this time launching a single Amazon ECS task to act as the leader node. This ECS task sends a start test message to each of the worker tasks in order to start the tests simultaneously.

9. The task-status-checker AWS Lambda function again checks if Amazon ECS tasks are running with the same test ID. If tasks are still running, it waits for one minute and checks again. Once there are no running Amazon ECS tasks, it returns the test ID, task count, test type, and prefix.

10. When the task-runner AWS Lambda function runs the Amazon ECS tasks in the AWS Fargate cluster, each task downloads the test configuration from Amazon S3 and starts the test.

11. Once the tests are running the average response time, number of concurrent users, number of successful requests, and number of failed requests for each task is logged in Amazon CloudWatch and can be viewed in a CloudWatch dashboard.

12. If you included live data in the test, the solution filters real-time test results in CloudWatch using a subscription filter. Then the solution passes the data to a Lambda function.

13. The Lambda function then structures the data received and publishes it to an AWS IoT Core topic.

14. The web console subscribes the AWS IoT Core topic for the test and receives the data published to the topic to graph the real-time data while the test is running.

15. When the test is complete, the container images export a detail report as an XML file to Amazon S3. Each file is given a UUID for the filename. For example, s3://dlte-bucket/test-scenarios/*<$TEST_ID>*/results/*<$UUID>*.json.

16. When the XML files are uploaded to Amazon S3, the results-parser AWS Lambda function reads the results in the XML files starting with the prefix and parses and aggregates all the results into one summarized result.

17.The `results-parser` AWS Lambda function writes the aggregate result to an Amazon
   DynamoDB table.

Test results

Distributed Load Testing on AWS leverages the Load Testing framework to run application testing at scale. When a test is complete, a detailed report is generated containing the following results.

- **Average response time** - The average response time, in seconds, for all the requests generated by the test.

- **Average latency** - The average latency, in seconds, for all the requests generated by the test.

- **Average connection time** - The average time, in seconds, it takes to connect to the host for all the requests generated by the test.

- **Average bandwidth** - The average bandwidth for all the requests generated by the test.

- **Total count** - The total number of requests.

- **Success count** - The total number of successful requests.

- **Error count** - The total number of errors.

- **Requests per second** - The average requests per seconds for all the requests generated by the test.

- **Percentile** - The percentile of the response time for the test. The maximum response time is 100%; the minimum response time is 0%.

> ⓘ **Note**
>
> Test results are displayed in the console. You can view the XML files for the raw test results in the `Results` folder of the `Scenarios` Amazon S3 bucket.

For more information on Taurus test results, see [Generating Test Reports](#) in the *Taurus User Manual*.

Distributed load testing API

This load testing solution helps you to expose test result data in a secure manner. The API acts as a "front door" for access to testing data stored in Amazon DynamoDB. You can also use the APIs to access any extended functionality you build into the solution.

This solution uses an Amazon Cognito user pool integrated with Amazon API Gateway for identification and authorization. When a user pool is used with the API, clients are only allowed to call user pool activated methods after they provide a valid identity token.

For more information on running tests directly via the API, refer to Signing Requests in the Amazon API Gateway REST API Reference documentation.

The following operations are available in the solution's API.

> ⓘ **Note**
>
> For more information about `testScenario` and other parameters, refer to scenarios and payload example in the GitHub repository.

Scenarios

- GET /scenarios
- POST /scenarios
- OPTIONS /scenarios
- GET /scenarios/{testId}
- POST /scenarios/{testId}
- DELETE /scenarios/{testId}
- OPTIONS /scenarios/{testId}

Tasks

- GET /tasks
- OPTIONS /tasks

Regions

- [GET /regions](#)
- [OPTIONS /regions](#)

GET /scenarios

Description

The GET /scenarios operation allows you to retrieve a list of test scenarios.

Response

Name	Description
data	A list of scenarios including the ID, name, description, status, and run time for each test

POST /scenarios

Description

The POST /scenarios operation allows you to create or schedule a test scenario.

Request body

Name	Description
testName	The name of the test
testDescription	The description of the test
testTaskConfigs	An object that specifies concurrency (the number of parallel runs), taskCount (the number of tasks needed to run a test), and region for the scenario

Name	Description
testScenario	The test definition including concurrency, test time, host, and method for the test
testType	The test type (for example, simple, jmeter)
fileType	The upload file type (for example, none, script, zip)
scheduleDate	The date to run a test. Only provided if scheduling a test (for example, 2021-02-28)
scheduleTime	The time to run a test. Only provided if scheduling a test (for example, 21:07)
scheduleStep	The step in the schedule process. Only provided if scheduling a recurring test. (Available steps include create and start)
recurrence	The recurrence of a scheduled test. Only provided if scheduling a recurring test (for example, daily, weekly, biweekly, or monthly)

Response

Name	Description
testId	The unique ID of the test
testName	The name of the test
status	The status of the test

OPTIONS /scenarios

Description

The `OPTIONS /scenarios` operation provides a response for the request with the correct CORS response headers.

Response

Name	Description
testId	The unique ID of the test
testName	The name of the test
status	The status of the test

GET /scenarios/{testId}

Description

The `GET /scenarios/{testId}` operation allows you to retrieve the details of a specific test scenario.

Request parameter

testId

    The unique ID of the test

    Type: String

    Required: Yes

Response

Name	Description
testId	The unique ID of the test
testName	The name of the test
testDescription	The description of the test
testType	The type of test that is run (for example, simple, jmeter)
fileType	The type of file that is uploaded (for example, none, script, zip)
status	The status of the test
startTime	The time and date when the last test started
endTime	The time and date when the last test ended
testScenario	The test definition including concurrency, test time, host, and method for the test
taskCount	The number of tasks needed to run the test
taskIds	A list of task IDs for running tests
results	The final results of the test
history	A list of final results of past tests
errorReason	An error message generated when an error occurs
nextRun	The next scheduled run (for example, 2017-04-22 17:18:00)

Name	Description
scheduleRecurrence	The recurrence of the test (for example, `daily`, `weekly`, `biweekly`, `monthly`)

POST /scenarios/{testId}

Description

The `POST /scenarios/{testId}` operation allows you to cancel a specific test scenario.

Request parameter

`testId`

    The unique ID of the test

    Type: String

    Required: Yes

Response

Name	Description
status	The status of the test

DELETE /scenarios/{testId}

Description

The `DELETE /scenarios/{testId}` operation allows you to delete all data related to a specific test scenario.

Request parameter

`testId`

> The unique ID of the test
>
> Type: String
>
> Required: Yes

Response

Name	Description
`status`	The status of the test

OPTIONS /scenarios/{testId}

Description

The `OPTIONS /scenarios/{testId}` operation provides a response for the request with the correct CORS response headers.

Response

Name	Description
`testId`	The unique ID of the test
`testName`	The name of the test
`testDescription`	The description of the test
`testType`	The type of test that is run (for example, `simple`, `jmeter`)
`fileType`	The type of file that is uploaded (for example, `none`, `script`, `zip`)

Name	Description
status	The status of the test
startTime	The time and date when the last test started
endTime	The time and date when the last test ended
testScenario	The test definition including concurrency, test time, host, and method for the test
taskCount	The number of tasks needed to run the test
taskIds	A list of task IDs for running tests
results	The final results of the test
history	A list of final results of past tests
errorReason	An error message generated when an error occurs

GET /tasks

Description

The GET /tasks operation allows you to retrieve a list of running Amazon Elastic Container Service (Amazon ECS) tasks.

Response

Name	Description
tasks	A list of task IDs for running tests

OPTIONS /tasks

Description

The `OPTIONS /tasks` tasks operation provides a response for the request with the correct CORS response headers.

Response

Name	Description
`taskIds`	A list of task IDs for running tests

GET /regions

Description

The `GET /regions` operation allows you to retrieve the regional resource information necessary to run a test in that Region.

Response

Name	Description
`testId`	The Region ID
`ecsCloudWatchLogGroup`	The name of the Amazon CloudWatch log group for the Amazon Fargate tasks in the Region
`region`	The Region in which the resources in the table exist
`subnetA`	The ID of one of the subnets in the Region
`subnetB`	The ID of one of the subnets in the Region

Name	Description
taskCluster	The name of the AWS Fargate cluster in the Region
taskDefinition	The ARN of the task definition in the Region
taskImage	The name of the task image in the Region
taskSecurityGroup	The ID of the security group in the Region

OPTIONS /regions

Description

The OPTIONS /regions operation provides a response for the request with the correct CORS response headers.

Response

Name	Description
testId	The Region ID
ecsCloudWatchLogGroup	The name of the Amazon CloudWatch log group for the Amazon Fargate tasks in the Region
region	The Region in which the resources in the table exist
subnetA	The ID of one of the subnets in the Region
subnetB	The ID of one of the subnets in the Region
taskCluster	The name of the AWS Fargate cluster in the Region

Name	Description
taskDefinition	The ARN of the task definition in the Region
taskImage	The name of the task image in the Region
taskSecurityGroup	The ID of the security group in the Region

Test scheduling workflow

Use the web console to schedule a load test. When scheduling a test, the following workflow runs:

- When a load test is created with the option to schedule, the schedule parameters are sent to the solution's API via Amazon API Gateway.

- The API then passes the parameters to a Lambda function which creates a CloudWatch Events rule, which will be scheduled to run on the date specified.

- If the test is a one-time test, the CloudWatch Events rule runs on the specified date. The `api-services` Lambda function runs a new test through the workflow specified in the Test workflow section of this guide.

- If the test is a recurring test, the CloudWatch Events rule activates on the specified date. The `api-services` Lambda function runs, which deletes the current CloudWatch Events rule and creates another rule that runs immediately when created, and recurrently thereafter based on the specified recurrence frequency.

Determine the number of users

The number of users a container can support for a test can be determined by gradually increasing the number of users, and monitoring performance in Amazon CloudWatch. Once you observe that CPU and memory performance are approaching their limits, you've reached the maximum number of users a container can support for that test in their default configuration (2 vCPU and 4 GB of memory). You can begin determining the concurrent user limits for your test by using the following example:

1. Create a test with no more than 200 users.
2. While the test runs, monitor the CPU and Memory using the [CloudWatch console](#):
   a. From the left navigation pane, under **Container Insights**, select **Performance Monitoring**.
   b. On the **Performance monitoring** page, from the left drop down menu, select **ECS Clusters**.
   c. From the right drop down menu, select your Amazon Elastic Container Service (Amazon ECS) cluster.
3. While monitoring, watch the CPU and Memory. If the CPU does not surpass 75% or the Memory does not surpass 85% (ignore one-time peaks), you can run another test with a higher number of users.

Repeat steps 1-3 if the test did not exceed the resource limits. Optionally, the containers resources can be increased to allow for a higher number of concurrent users. However, this results in a higher cost. For details, refer to the [Increase the container resources](#) section of this guide.

> (i) **Note**
>
> For accurate results, run only one test at a time when determining concurrent user limits. All tests use the same cluster and CloudWatch container insights aggregates the performance data based on cluster. This causes both tests to be reported to CloudWatch container insights simultaneously, which results in inaccurate resource utilization metrics for a single test.

For more information on calibrating users per engine, refer to [Calibrating a Taurus Test](#) in the BlazeMeter documentation.

Increase the container resources

To increase the number of users currently supported, increase the container resources. This allows you to increase the CPUs and memory to handle the increase in concurrent users.

Create a new task definition revision

1. Sign in to the [Amazon Elastic Container Service console](#).

2. In the left navigation menu, select **Task Definitions**.

3. Select the checkbox next to the task definition that corresponds to this solution. For example, *<stackName>*-EcsTaskDefinition-*<system-generated-random-Hash>*.

4. Choose **Create new revision**.

5. On the **Create new revision** page, take the following actions:

   a. Under **Task size**, modify the **Task memory** and the **Task CPU**.

   b. Under **Container Definitions**, review the **Hard/Soft memory limits**. If this limit is lower than your desired memory, choose the container.

   c. In the **Edit container** dialog box, go to **Memory Limits** and update the **Hard Limit** to your desired memory.

   d. Choose **Update**.

6. On the **Create new revision** page, choose **Create**.

7. After the task definition is successfully created, record the name of the new task definition. This name includes the version number, for example: *<stackName>*-EcsTaskDefinition-*<system-generated-random-Hash>*:*<system-generated-versionNumber>*.

Update the DynamoDB table

1. Navigate to the [DynamoDB console](#).

2. From the left navigation pane, select **Explore Items** under **Tables**.

3. Select the scenarios-table DynamoDB table associated with this solution. For example, *<stackName>*-DLTTestRunnerStorageDLTScenariosTable-*<system-generated-random-Hash>*.

4. Select the item that corresponds to the Region in which you have modified the task definition. For example, region-*<region-name>*.

5. Update the **taskDefinition** attribute with the new task definition.

Live data

You can optionally include live data when running a test to gain real-time insights into what is occurring. The CloudWatch log group for the Fargate tasks contains a subscription filter for results from tests that include the live data option. If the solution finds the pattern, it initiates a Lambda function that structures the data and publishes it to an AWS IoT Core topic. The web console subscribes to the topic, receives the incoming data, and graphs the data aggregated at one-second intervals. The web console contains four graphs: average response time, virtual users, successes, and failures.

> ⓘ **Note**
>
> The data is ephemeral and is only for use to see what is happening while the test is running. Once a test is complete, the solution stores the results data in DynamoDB and Amazon S3. The web console persists a maximum of 5,000 data points, after which the oldest data is replaced with the newest. If the page refreshes, the graphs will be blank and start from the next available data point.

Test cancellation workflow

When you cancel a load test from the web console, the solution runs the following test cancellation workflow.

1. The cancellation request is sent to the `microservices` API.

2. The `microservices` API calls the `task-canceler` Lambda function which cancels tasks until all the currently launched tasks are stopped.

3. If the `task-runner` Lambda function continues to run after the initial call to the `task-canceler` Lambda function, then tasks will continue to be launched. Once the `task-runner` Lambda function finishes, AWS Step Functions continues to the `Cancel Test` step, which runs the `task-canceler` Lambda function again to stop any remaining tasks.

Troubleshooting

This section provides troubleshooting instructions for deploying and using the solution.

Known issue resolution provides instructions to mitigate known errors. If these instructions don't address your issue, see the Contact AWS Support section for instructions on opening an AWS Support case for this solution.

Known issues

Issue: You are using an existing VPC and your tests fail with a status of `Failed`, resulting in the following error message:

```
Test might have failed to run.
```

Solution: Ensure that the subnets exist in the VPC specified and that they have a route to the internet with either an internet gateway or a NAT gateway. AWS Fargate needs access to pull the container image from the public repository to successfully run tests.

Isuue: Tests are taking too long to run or are stuck indefinitely running.

Solution: Cancel the test and check AWS Fargate to ensure that all tasks have stopped. If they have not stopped, manually stop all Fargate tasks. Check the on-demand Fargate task limits on your account to ensure that you can launch the number of tasks desired. You can also check the CloudWatch logs for the Lambda `task-runner` function for more insight into failures when launching Fargate tasks. Check the CloudWatch ECS logs for details of what is happening in Fargate containers that are running.

When updating from DLT versions older than v3.2.6 to latest, updating the stack fails.
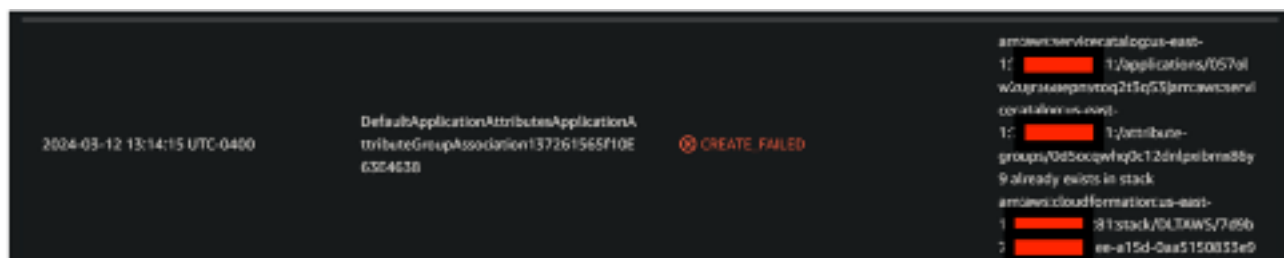
Solution: This is known issue when updating DLT from versions prior to v3.2.6 to the latest, where stack fails due to failure in creation of app registry application attribute group association like below image. You can follow our suggested solution to resolve the issue and continue updating the stack.

1. Use the following [link](#) to download the `distributed-load-testing-on-aws.template`

2. Open the template and navigate to `Conditions:` and look for `DLTCommonResourcesAppRegistryCondition`

3. You should be seeing something similar the following:

```
Conditions:
DLTCommonResourcesAppRegistryConditionCCEF54F8:

Fn::Equals:
- "true"
- "true"
```

4. Change the second true value to `false:`

```
Conditions:
 DLTCommonResourcesAppRegistryConditionCCEF54F8:
Fn::Equals:
- "true"
- "false"
```

5. Use the customized template to update your stack.

6. This stack removes app registry related resources from the stack. Hence the update should be completed.

7. Perform another stack update using the latest template url to add back app registry application resources to your stack.

> ⓘ **Note**
>
> AWS Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:
> 1. Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.

2. View operations data for the resources of this solution in the context of an application, such as deployment status, CloudWatch alarms, resource configurations, and operational issues.

Contact AWS Support

If you have [AWS Developer Support](#), [AWS Business Support](#), or [AWS Enterprise Support](#), you can use the Support Center to get expert assistance with this solution. The following sections provide instructions.

Create case

1. Sign in to [Support Center](#).

2. Choose **Create case.**

How can we help?

1. Choose **Technical**

2. For **Service**, select **Solutions**.

3. For **Category**, select **Distributed Load Testing on AWS**.

4. For **Severity**, select the option that best matches your use case.

5. When you enter the **Service**, **Category**, and **Severity**, the interface populates links to common troubleshooting questions. If you can't resolve your questions with these links, choose **Next step: Additional information**.

Additional information

1. For **Subject**, enter text summarizing your question or issue.

2. For **Description**, describe the issue in detail.

3. Choose **Attach files**.

4. Attach the information that AWS Support needs to process the request.

Help us resolve your case faster

1. Enter the requested information.

2. Choose **Next step: Solve now or contact us**.

Solve now or contact us

1. Review the **Solve now** solutions.

2. If you can't resolve your issue with these solutions, choose **Contact us**, enter the requested information, and choose **Submit**.

Uninstall the solution

You can uninstall the Distributed Load Testing on AWS solution from the AWS Management Console or by using the AWS Command Line Interface. You must manually delete the console, scenario, and logging Amazon Simple Storage Service (Amazon S3) buckets created by this solution. AWS Solutions Implementations do not automatically delete them in case you have stored data to retain.

> ⓘ **Note**
>
> If you have deployed regional stacks, you must delete the stacks in those Regions before deleting the main stack.

Using the AWS Management Console

1. Sign in to the [AWS CloudFormation console](#).

2. On the **Stacks** page, select this solution's installation stack.

3. Choose **Delete**.

Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, refer to [What Is the AWS Command Line Interface](#) in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>
```

Deleting the Amazon S3 buckets

This solution is configured to retain the solution-created Amazon S3 bucket (for deploying in an opt-in Region) if you decide to delete the AWS CloudFormation stack to prevent accidental data loss. After uninstalling the solution, you can manually delete this S3 bucket if you do not need to retain the data. Follow these steps to delete the Amazon S3 bucket.

1. Sign in to the [Amazon S3 console](#).

2. Choose **Buckets** from the left navigation pane.

3. In the **Find buckets by name** field, enter the name of this solution's stack.

4. Select one of the solution's S3 buckets and choose **Empty**.

5. Enter **permanently delete** in the verification field and choose **Empty**.

6. Choose the S3 bucket name you just emptied and choose **Delete**.

7. Enter the S3 bucket name in the verification field and choose **Delete bucket**.

Repeat steps 3 through 7 until you delete all the S3 buckets.

To delete the S3 bucket using AWS CLI, run the following command:

```
$ aws s3 rb s3://<bucket-name> --force
```

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 activated, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Unique ID (UUID)** - Randomly generated, unique identifier for each solution deployment
- **Timestamp** - Data-collection timestamp
- **Test Type** - The type of test that is run
- **File Type** - The type of file that is uploaded
- **Task Count** - The task count for each test submitted through the solution's API
- **Task Duration** - The total run time for all tasks needed to run a test
- **Test Result** - The result of the test that was run

AWS owns the data gathered via this survey. Data collection is subject to the [AWS Privacy Policy](). To opt out of this feature, complete the following steps before launching the AWS CloudFormation template.

1. Download the [AWS CloudFormation template]() to your local hard drive.
2. Open the AWS CloudFormation template with a text editor.
3. Modify the AWS CloudFormation template mapping section from:

```
Solution:
  Config:
    SendAnonymousData: "Yes"
```

   to:

```
Solution:
  Config:
    SendAnonymousData: "No"
```

4. Sign in to the [AWS CloudFormation console]().
5. Select **Create stack**.

6. On the **Create stack** page, **Specify template** section, select **Upload a template file**.

7. Under **Upload a template file**, choose **Choose file** and select the edited template from your local drive.

8. Choose **Next** and follow the steps in [Launch the stack](#) in the Automated deployment section of this guide.

Source code

Visit our GitHub repository to download the templates and scripts for this solution, and to share your customizations with others.

Contributors

The following individuals contributed to this document:

- Tom Nightingale

- Fernando Dingler

- Beomseok Lee

- George Lenz

- Erin McGill

- Dimitri Lopez

- Kamyar Ziabari

Revisions

Date	Change
November 2019	Initial release
September 2020	Release version 1.1.0: Replaced Amazon SQS with AWS Step Functions and updated the architecture diagram and components information to detail the changed AWS service; added support for JMeter scripts; for more information, refer to the CHANGELOG.md file in the GitHub repository.
December 2020	Release version 1.2.0: Added Amazon ECR checker to AWS Step Functions; added support for zip file uploads for JMeter, enabling the ability to use JMeter plugins; for more information, refer to the CHANGELOG.md file in the GitHub repository.
April 2021	Release version 1.3.0: Added support for running concurrent tests; added support for starting tests simultaneously across tasks belonging to the same test; added support for scheduling tests; increased task limit to 1,000 tasks; removed concurrent users limit; for more information, refer to the CHANGELOG.md file in the GitHub repository.
September 2021	Release version 2.0.0: Added support to view complete test configuration, test data, and Amazon CloudWatch dashboard from previous test runs; the solution container image is now managed by AWS, removing the requirement to create AWS CodePipeline, AWS CodeBuild, and Amazon ECR image repository in the customer account; updated the CloudWatch dashboard to show maximum data points; added support for an existing Amazon VPC; propagated CloudFormation tags to Fargate tasks; Fargate tasks for tests are launched in multiple availability zones. For more information, refer to the CHANGELOG.md file in the GitHub repository.

Date	Change
December 2021	Release version 2.0.1: Updated AWS SDK version in development dependencies for AWS Lambda functions; resolved issue with displaying a large number of tests; resolved ValidationException error with DynamoDB. For more information, refer to the CHANGELOG.md file in the GitHub repository.
August 2022	Release version 3.0.0: Updated to AWS CDK V2, added multi-Region functionality, added live date functionality, added JMeter extensions compatibility, and additional enhancements and bug fixes. For more information, refer to the CHANGELOG.md file in the GitHub repository.
November 2022	Release version 3.1.0: Added AppRegistry support for the solution. For more information, refer to the CHANGELOG.md file in the GitHub repository.
November 2022	Release version 3.1.1: Bug fix. For more information, refer to the CHANGELOG.md file in the GitHub repository.
March 2023	Release version 3.2.0: Updated task limiting to be based on account quota, added auto-refresh functionality, added linting and consistent formatting, upgraded to bootstrap 5, and implemented minor bug fixes. For more information, refer to the CHANGELOG.md file in the GitHub repository.
April 2023	Release version 3.2.1: 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 repository.
July 2023	Release version 3.2.2: Updated to `react-scripts v5.0.1`; other bug fixes and minor updates. For more information, refer to the CHANGELOG.md file in the GitHub repository.
August 2023	Documentation update: Added AWS Support section for contacting support.

Date	Change
October 2023	Release version 3.2.3: Updated CDK libraries and Docker image, other bug fixes and security updates. For more information, refer to the CHANGELOG.md file in the GitHub repository.
November 2023	Documentation update: Added AWS Developer Support and merged Contact AWS Support into the Troubleshooting section.
November 2023	Release version 3.2.4: Updated Lambda runtimes to use Node.js 18, updated Taurus image to version 1.16.26, updated JMeter and Taurus dependencies within the Docker image to enhance the security of the Docker image, and additional bug fixes. For more information, refer to the CHANGELOG.md file in the GitHub repository.
November 2023	Documentation update: Added the section called "Confirm cost tags associated with the solution" to the Monitoring the solution with AWS Service Catalog AppRegistry section.
January 2024	Release version 3.2.5: Bug fix to resolve issues with automatic plugins installation. For more information, refer to the CHANGELOG.md file in the GitHub repository.
March 2024	Release version 3.2.6: Bug fix to resolves issues with enabled tag resource authentication in ECS. Updating version of taurus, react, chart.js, amplify, amplify/ui-react. For more information, refer to the CHANGELOG.md file in the GitHub repository.
March 2024	Release version 3.2.7: Updated Changelog to include information on previous release. Added package-lock.json. For more information, refer to the CHANGELOG.md file in the GitHub repository.

Date	Change
April 2024	Release version 3.2.8: Updated packages to resolve vulnerabilities in npm packages and in docker image. Bug fix to resolve issues with scheduling. For more information, refer to the [CHANGELOG.md](#) file in the GitHub repository.
June 2024	Release version 3.2.9: Patch update to mitigate security vulnerabilities. For more information, refer to the [CHANGELOG.md](#) file in the GitHub repository.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Distributed Load Testing on AWS is licensed under the terms of the of the Apache License Version 2.0 available at The Apache Software Foundation.

AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.