Distributed Load Testing on AWS

Implementation Guide

Distributed Load Testing on AWS: Implementation Guide

Table of Contents

Distributed Load Testing on AWS

AWS Implementation Guide

AWS Solutions Builder Team

November 2019 (last update (p. 28): December 2020)

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.

Overview

Many Amazon Web Services (AWS) customers want to accelerate time to release by minimizing application bugs and the time required for testing in a continuous integration and delivery development model.

Software test automation tools enable you to simplify testing and reduce time to release by automating functional tests for your applications. But, some testing automation tools can require you to provision additional infrastructure to run tests, and can be difficult and costly to set up.

To help make it easier for customers to automate application testing, AWS offers Distributed Load Testing on AWS. Distributed Load Testing is a solution that leverages AWS Fargate to run containers to easily create and simulate thousands of connected users generating a select number of transactions per second without having to provision servers. With this solution, you can understand how your application will perform at scale and at load, and identify bottlenecks before you release your application.

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. At the date of publication, the cost for running this solution with default settings in the US East (N. Virginia) Region is approximately **$9.12 per month**. The cost estimate assumes the following factors:

- 10 AWS Fargate tasks running for 30 hours
- 1,000 Amazon DynamoDB write capacity units using the On-Demand capacity mode
- 1,000 DynamoDB read capacity units using the On-Demand capacity mode
- 1,000 AWS Lambda function requests
- 10 minutes duration running AWS Lambda functions
- 1,000 AWS Step Functions state transitions

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.

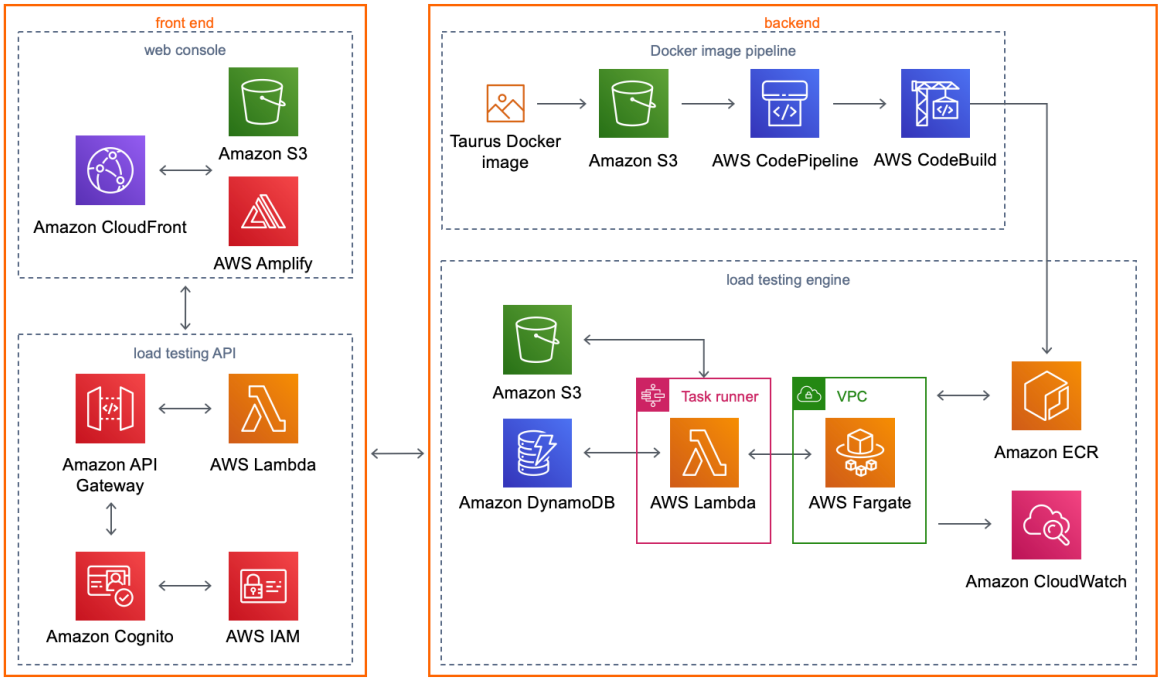

Figure 1: Distributed Load Testing on AWS architecture

The AWS CloudFormation template deploys a distributed load tester API, which leverages Amazon API Gateway to invoke the solution's microservices (AWS Lambda functions). The microservices provide the business logic to manage test data and run the tests. These microservices interact with Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, and AWS Step Functions (Amazon SQS) to provide storage for the test scenario details and run test scenarios.

The solution also deploys an Amazon Virtual Private Cloud (Amazon VPC) network topology that contains the solution's Amazon Elastic Container Service (Amazon ECS) containers running on AWS Fargate. The containers contain a Taurus load testing Docker image which is used to generate load for testing your application's performance. Taurus is an open-source test automation framework. AWS CodePipeline, AWS CodeBuild, and Amazon S3 help manage the image.

The solution creates a web console powered by AWS Amplify and deploys it into an Amazon S3 bucket configured for static web hosting. Amazon CloudFront is used to provide secure, public access to the solution's website bucket contents. During initial configuration, the solution also creates a default administrator role and sends an access invite to a customer-specified user email address. The solution uses an Amazon Cognito user pool to manage user access to the console and the load tester API.

After you deploy the solution, you can use the web console to create a test scenario that defines a series of tasks. This solution's microservices use this test scenario to run AWS Fargate tasks. When each task is complete, the results are stored in Amazon S3 and the output is logged in Amazon CloudWatch. Once all tasks are complete, the results are stored in Amazon DynamoDB.

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 Simple Storage Service (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, see Appendix D (p. 19).

Web Console

This solution includes a simple web console 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 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.

Backend

The backend consists of a Docker image pipeline and load testing engine you use to generate load for the tests. You interact with the backend through the front end.

Docker Image Pipeline

This solution leverages a Docker image of the Taurus Load Testing framework. During launch, a Docker file is uploaded to an Amazon S3 bucket. AWS CodePipeline uses AWS CodeBuild to create a new image

from the Docker file and registers the image with Amazon Elastic Container Registry. The image is used to run tasks in the AWS Fargate cluster.

CodePipeline is only used to register the container image during initial deployment. But, you can also use CodePipeline to deploy updates to the image. For more information, see Appendix A (p. 14).

Load Testing Engine

The Distributed Load Testing solution uses Amazon Elastic Container Service (Amazon ECS) and AWS Fargate to simulate thousands of connected users 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 Simple Storage Service (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, and a `results-parser` AWS Lambda function. For more information on the workflow, see Appendix B (p. 16). For more information on test results, see Appendix C (p. 18).

Considerations

Supported Applications

Distributed Load Testing on AWS supports cloud-based applications, and on-prem 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.

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.

Load Testing Limits

The maximum number of tasks that can be running in Amazon ECS using the AWS Fargate launch type is 100 per AWS Region, per account. For more information, see Amazon ECS Service Limits. For instructions on how to request an increase, see AWS Service Limits in the *AWS General Reference Guide*.

The Taurus load testing Docker image can generate up to 200 concurrent connections per task which means that the maximum load for a single test is 20,000 concurrent requests (200 connections x 100 running tasks at a time). The maximum execution time for a single test is four hours.

Concurrent Tests

This solution includes an Amazon CloudWatch dashboard that displays the combined output of all tasks running in the Amazon ECS cluster in real time. Only one CloudWatch log group can be assigned to an ECS cluster. This solution's web console supports one running test as a time. If you run more than one test at a time, the dashboard will show the combined results from all tests.

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, see the Amazon EC2 Testing Policy.

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, see 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, see AWS service offerings by region.

Solution Updates

If you have previously deployed the solution, you must update the solution's CloudFormation stack to get the latest version of the solution's framework. For details, refer to Update the Stack (p. 10).

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:

distributed-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), Amazon Elastic Container Registry (Amazon ECR), AWS Fargate, Amazon Virtual Private Cloud (Amazon VPC), AWS Lambda, AWS CodePipeline, AWS CodeBuild, 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

Before you launch the automated deployment, please 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

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

Note
You are responsible for the cost of the AWS services used while running this solution. See the Cost (p. 2) section for more details. For full details, see the pricing webpage for each AWS service you will be using 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.

   

   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, see the AWS service offerings by region.
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>*	The 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
AWS Fargate VPC CIDR Block	`192.168.0.0/16`	CIDR block for the solution-created Amazon VPC that will contain AWS Fargate
AWS Fargate Subnet A CIDR Block	`192.168.0.0/20`	CIDR block for VPC subnet A
AWS Fargate Subnet B CIDR Block	`192.168.16.0/20`	CIDR block for 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. Be sure to 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 see a status of **CREATE_COMPLETE** in approximately 20 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.

Update the Stack

If you have previously deployed the solution, follow this procedure to update the Distributed Load Testing on AWS CloudFormation stack to get the latest version of the solution framework.

1. Sign in to the AWS CloudFormation console, select your existing Distributed Load Testing on AWS stack, and choose **Update**.
2. On the **Update stack** page, verify that **Replace current template** is selected.

   a. In the **Specify template** section, select **Amazon S3 URL**.
   b. Copy the link of the latest template.
   c. Paste the link in the **Amazon S3 URL** box.
   d. Verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.
3. On the **Specify stack details** page, under **Parameters**, review the parameters for the template and modify them as necessary. Refer to Launch the Stack (p. 9) for details about the parameters.
4. Choose **Next**.
5. On the **Configure stack options** page, choose **Next**.
6. 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.
7. Choose **View change set** and verify the changes.
8. Choose **Update 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 an **UPDATE_COMPLETE** status in approximately 15 minutes.

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 the AWS Security Center.

IAM Role

AWS Identity and Access Management (IAM) roles enable 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).

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 1 Gpps (1 billion packets per second).

Additional Resources

AWS services

- Amazon Elastic Container Service
- Amazon Elastic Container Registry
- AWS Fargate
- AWS Lambda
- Amazon Simple Storage Service
- Amazon DynamoDB
- Amazon Simple Queue Service
- Amazon CloudWatch
- Amazon API Gateway
- AWS Step Functions
- Amazon Cognito
- Amazon CloudFront
- Amazon Virtual Private Cloud
- AWS CodePipeline
- AWS CodeBuild
- AWS Identity and Access Management
- AWS Amplify
- AWS CloudFormation

Other services

- Taurus

Appendix A: Updating the Docker Image

Distributed Load Testing on AWS uses AWS CodePipeline to help build and register the Taurus Docker image in Amazon Elastic Container Registry (Amazon ECR). During initial deployment, a container ZIP file is uploaded to Amazon Simple Storage Service (Amazon S3). AWS CodeBuild unpacks the ZIP file, builds the Docker image, and sends it to Amazon ECR for registration.

Figure 2: Docker image pipeline

To update the Taurus Docker image, upload a new or updated container ZIP file to Amazon S3. AWS CodePipeline will automatically start the build and registration process with the new file.

The following example shows the Docker 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

# Taurus working directory = /bzt-configs
ADD ./load-test.sh /bzt-configs/
RUN chmod 755 /bzt-configs/load-test.sh

ENTRYPOINT ["sh", "-c","./load-test.sh"]
```

In addition to a Docker file, the container ZIP file contains the following Bash script that downloads the test configuration from Amazon S3 before running the Taurus executable.

```
#!/bin/bash

# set a uuid for the results xml file name in S3
UUID=$(cat /proc/sys/kernel/random/uuid)

echo "S3_BUCKET:: ${S3_BUCKET}"
echo "TEST_ID:: ${TEST_ID}"
echo "TEST_TYPE:: ${TEST_TYPE}"
echo "PREFIX:: ${PREFIX}"
echo "UUID ${UUID}"

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

```
# download JMeter jmx file
if [ "$TEST_TYPE" != "simple" ]; then
  aws s3 cp s3://$S3_BUCKET/public/test-scenarios/$TEST_TYPE/$TEST_ID.jmx ./
fi

echo "Running test"
bzt test.json -o modules.console.disable=true

# 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
  cat $TEST_ID.jmx | grep filename > results.txt
  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

echo "Uploading results"
aws s3 cp /tmp/artifacts/results.xml s3://$S3_BUCKET/results/${TEST_ID}/${PREFIX}-
${UUID}.xml
```

Appendix B: Test Workflow

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

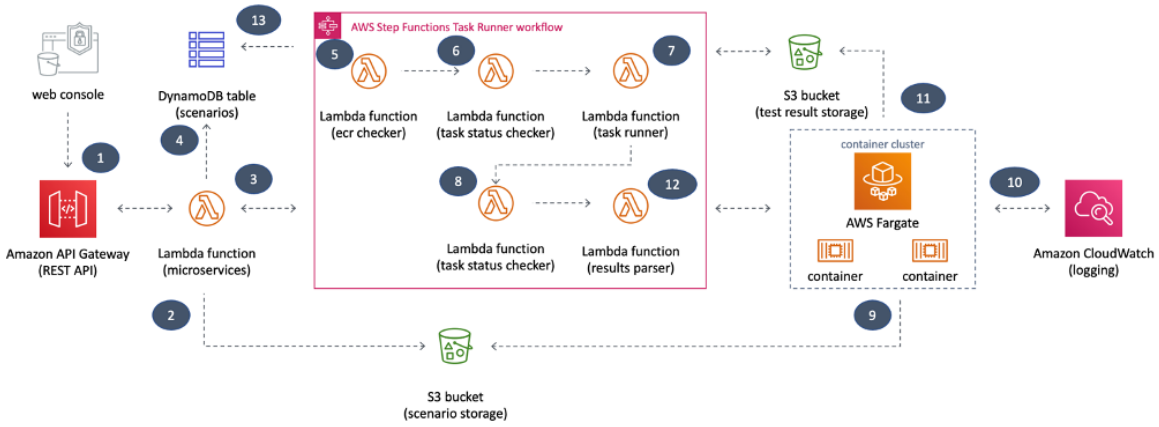

Figure 3: Test workflow

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.
4. Configuration details are stored in the `scenarios` Amazon DynamoDB table.
5. In the AWS Step Functions task runner workflow, the `ecr-checker` AWS Lambda function checks whether the `ContainerCodeBuild` AWS CodeBuild project has succeeded. If the CodeBuild project remains in progress, the AWS Lambda function continues checking in one minute intervals. When the CodeBuild project is completed, the AWS Lambda function checks whether the `latest` tag is applied to the newest image in the Amazon Elastic Container Registry (Amazon ECR) image repository. If this tag has not been applied, the AWS Lambda function creates an error message.
6. 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.
7. The `task-runner` AWS Lambda function gets the task details from the previous step and runs the Amazon Elastic Container Service (Amazon ECS) tasks in the AWS Fargate cluster. The Amazon ECS API uses the RunTask action to run the task. 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 tasks are complete. The function also generates a prefix to distinguish the current test in the `results-parser` AWS Lambda function.
8. 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.
9. 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.
10. The average response time for each task is logged in Amazon CloudWatch and can be viewed in a CloudWatch dashboard.

11. 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`.
12. 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.
13. The `results-parser` AWS Lambda function writes the aggregate result to an Amazon DynamoDB table.

Appendix C: 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%.

For more information on Taurus test results, see Generating Test Reports in the *Taurus User Manual*.

Appendix D: Distributed Load Testing API

This load testing solution enables 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 enabled methods after they provide a valid identity token.

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

Scenarios

Tasks

GET /scenarios

Description

The `GET /scenarios` operation enables 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 enables you to create a test scenario.

Request Body

Name	Description
`testName`	The name of the test
`testDescription`	The description of the test
`taskCount`	The number of tasks needed to run the test
`testScenario`	The test definition including concurrency, test time, host, and method for the test

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 enables 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 (such as simple, jmeter)
`fileType`	The type of file that is uploaded (such as 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

POST /scenarios/{testId}

Description

The `POST /scenarios/{testId}` operation enables 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 enables 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

Name	Description
`testName`	The name of the test
`testDescription`	The description of the test
`testType`	The type of test that is run (such as simple, jmeter)
`fileType`	The type of file that is uploaded (such as 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

GET /tasks

Description

The `GET /tasks` operation enables 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

Appendix E: Collection of Operational Metrics

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. When enabled, 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

Note that AWS will own the data gathered via this survey. Data collection will be subject to the AWS Privacy Policy. To opt out of this feature, modify the AWS CloudFormation template mapping section as follows:

```
AnonymousData:
  SendAnonymousData
    Data: "Yes"
```

to

```
AnonymousData:
  SendAnonymousData
    Data: "No"
```

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

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

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.