Implementation Guide

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*

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 provides an overview of the Distributed Load Testing on AWS solution, its reference architecture and components, considerations for planning the deployment, and configuration steps for deploying the solution to 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 intended audience for using this solution's features and capabilities in their environment includes IT infrastructure architects, administrators, and DevOps professionals who have practical experience architecting in the AWS Cloud.

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

If you want to . . .	Read . . .
Know the cost for running this solution.	Cost

If you want to . . .	Read . . .
The estimated cost for running this solution in the US East (N. Virginia) Region is USD $ 30.90 per month for AWS resources.	
Understand the security considerations for this solution.	Security
Know how to plan for quotas for this solution.	Quotas
Know which AWS Regions support this solution.	Supported AWS Regions
View or download the AWS CloudFormation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution.	AWS CloudFormation template
Access the source code and optionally use the AWS Cloud Development Kit (AWS CDK) to deploy the solution.	GitHub repository

Features

The solution provides the following features:

Out-of-the-Box Configurable Performance Tests

Includes pre-configured performance tests available for immediate use.

Customizable Application Tests

Allows for flexible and precise test customization to identify potential issues and tailors tests to specific requirements and scenarios using JMeter scripts.

Simulates High User Load

Capable of simulating tens of thousands of connected users to stress test your application.

Continuous Transaction Generation

Generates transactional records continuously to evaluate performance under constant load.

Real-Time Monitoring

Provides real-time monitoring of test progress and results and allows you to schedule tests to start automatically on specified dates or at recurring intervals.

Regional Request Simulation

Simulate user requests from any region to assess global performance.

Endpoint Flexibility

Test any endpoint across AWS regions, on-premises environments, or other cloud providers.

Detailed Test Results

View comprehensive test results, including average response time, number of concurrent users, successful requests, and failed requests.

Intuitive Web Console

Offers an easy-to-use web console for managing and monitoring tests.

Supports Multiple Protocols

Compatible with various protocols such as WebSocket, HTTP, HTTPS, JDBC, JMS, FTP, and gRPC.

Benefits

The solution provides the following benefits:

Supports Comprehensive Performance Testing

Facilitates load, stress, and endurance testing for thorough application evaluation.

Early Detection of Performance Issues

Identifies performance issues and bottlenecks prior to production release.

Real-World Usage Simulation

Accurately mirrors real-world usage patterns to highlight bottlenecks and optimization areas.

Detailed Performance Insights

Provides insights into software performance and resilience under significant load.

Automated Performance Assessment

Enables regular performance evaluations without manual intervention.

Cost-Efficient Testing

Offers a pay-as-you-go model, eliminating the need for a dedicated testing infrastructure and subscription fees.

Use cases

Simulate Production Load

Test web and mobile applications under production-like conditions before launching a new version.

Validate Application Performance

Ensure your application can handle expected user traffic without degradation. Test application limits using default resources and assess infrastructure scalability.

Manage Peak Loads

Verify that your infrastructure can manage peak loads or unexpected traffic spikes, ensuring stability under high demand.

Optimize Performance

Understand your application's performance profile and identify bottlenecks such as inefficient code execution, database queries, and network latency.

Quick Testing Start-Up

Begin testing quickly with out-of-the-box performance tests.

Customizable Tests

Tailor tests to specific scenarios and requirements, adjusting the number of concurrent users and tasks launched.

Scheduled Testing

Schedule tests for regression testing and continuous performance monitoring, ensuring consistent application performance.

Geographical Performance Evaluation

Evaluate application performance across different geographical regions to ensure global efficiency.

CI/CD Pipeline Integration

Integrate performance testing into your CI/CD pipeline for seamless and automated testing during development cycles.

Concepts and definitions

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

scenario

Test definition including the test's name, description, task count, concurrency, AWS Region, ramp-up, hold-for, test type, schedule date, and recurrence configurations.

task count

Number of containers that will be launched in the Fargate cluster to run the test scenario. Additional tasks will not be created once the account limit on Fargate resources has been reached. However, tasks already running will continue.

concurrency

The number of concurrent virtual users generated per task. The recommended limit based on default settings is 200 virtual users. Concurrency is limited by CPU and memory. For tests based on Apache JMeter, the higher the number of virtual users, the higher the memory utilized by the JVM on the ECS task. The default ECS Task Definition creates tasks with 4 GB of memory. It is recommended to start with lower values of virtual users for 1 task and monitor the ECS CloudWatch metrics for the Task Cluster. Refer to Amazon ECS cluster utilization metrics.

ramp-up

The time to reach target concurrency.

hold-for

Time to hold target concurrency.

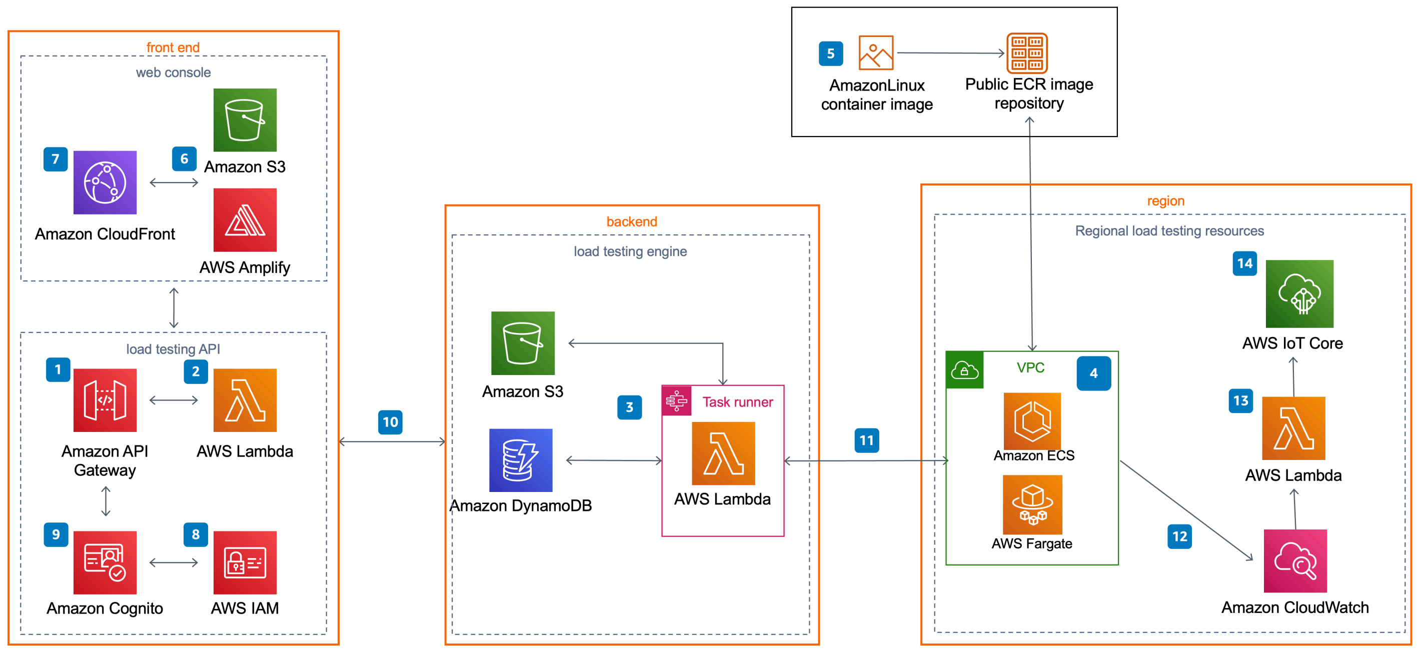
For a general reference of AWS terms, see the AWS Glossary.

Architecture overview

Architecture diagram

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

Distributed Load Testing on AWS architecture on AWS



Note

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

The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

1. A distributed load tester API, which leverages [Amazon API Gateway](https://aws.amazon.com/api-gateway/) to invoke the solution's microservices ([AWS Lambda](https://aws.amazon.com/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 AmazonLinux (with blazemeter load testing framework installed) Open Container Initiative (OCI) compliant container image, which is used to generate load for testing your application's performance. Taurus/Blazemeter 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 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.

AWS Well-Architected design considerations

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

This section describes how the design principles and best practices of the Well-Architected Framework benefit this solution.

Operational excellence

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

- Resources defined as infrastructure as code using CloudFormation.
- The solution pushes metrics to Amazon CloudWatch at various stages to provide observability into the infrastructure: Lambda functions, Amazon ECS tasks, Amazon S3 buckets, and the rest of the solution components.

Security

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

- Amazon Cognito authenticates and authorizes web UI app users.
- All interservice communications use applicable AWS Identity and Access Management (IAM) roles.
- All roles used by the solution follow least privilege access. They only contain the minimum permissions required to accomplish the task.
- All data storage, including the S3 buckets, encrypts the data at rest.
- An Amazon Cognito user pool manages user access to the console and the distributed load tester API Gateway endpoints.
- Logging, tracing, and versioning is turned on where applicable.
- Network access is private by default with Amazon Virtual Private Cloud (Amazon VPC) endpoints being turned on where available.

Note

Distributed Load Testing on AWS creates multiple CloudWatch log groups based on the service being used. The log retention period for the logs varies based on the storage and cost of the volume of log events generated. The solution creates container insight logs for the ECS tasks; these logs have log retention configured to 1 day. The logs created for the services AWS Step Functions, ECS Load Testing custom logs, and API Gateway logs are configured with a retention period of 1 year. AWS Lambda runtime logs have log retention configured to 2 years. API Gateway execution logs have log retention configured to never expire. You can change the log retention periods based on your requirements in the CloudWatch console.

Reliability

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

- The solution uses AWS serverless services wherever possible (examples: Lambda, API Gateway, Amazon S3, AWS Step Functions, Amazon DynamoDB, and AWS Fargate) to ensure high availability and recovery from service failure.
- All compute processing uses Lambda functions or Amazon ECS on AWS Fargate.
- Data is stored in DynamoDB and Amazon S3, so it persists in multiple Availability Zones by default.

Performance efficiency

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

- The solution uses a serverless architecture with the ability to scale horizontally as needed.
- The solution can be launched in any Region that supports the AWS services in this solution, such as: AWS Lambda, Amazon API Gateway, Amazon S3, AWS Step Functions, Amazon DynamoDB, Amazon ECS, AWS Fargate, and Amazon Cognito.
- The solution uses managed services throughout to reduce the operational burden of resource provisioning and management.
- The solution is automatically tested and deployed daily to achieve consistency as AWS services change, as well as reviewed by solution architects and subject matter experts for areas to experiment and improve.

Cost optimization

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

- The solution uses serverless architecture; therefore, customers only get charged for what they use.
- Amazon DynamoDB scales capacity on demand, so you only pay for the capacity you use.
- AWS ECS on AWS Fargate allows you to pay only for the compute resources you use, with no upfront expenses.

Sustainability

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

- The solution uses managed serverless services to minimize the environmental impact of the backend services compared to continually operating on-premises services.
- Serverless services allow you to scale up or down as needed.

Architecture details

This section describes the components and AWS services that make up this solution and the architecture details on how these components work together.

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 AnAm

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.

AWS services in this solution

The following AWS services are included in this solution:

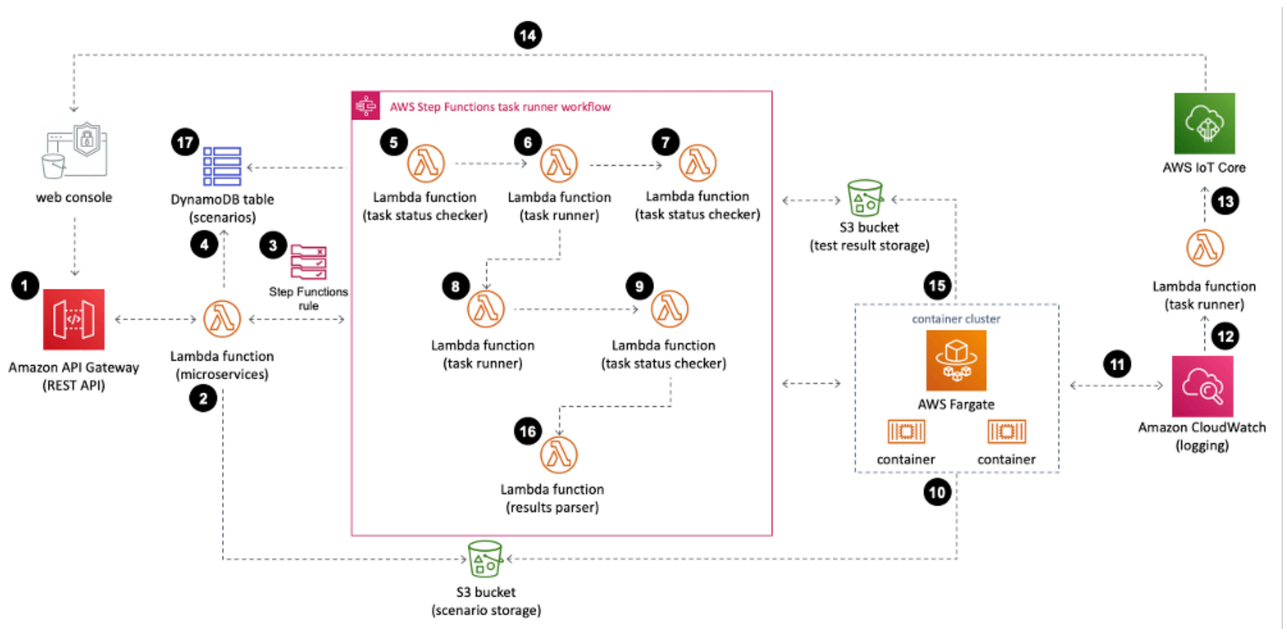
AWS service	Description
Amazon API Gateway	**Core.** Hosts REST API endpoints in the solution.
AWS CloudFormation	**Core.** Manages deployments for the solution infrastructure.
Amazon CloudFront	**Core.** Serves the web content hosted in Amazon S3.
Amazon CloudWatch	**Core.** Stores the solution logs and metrics.
Amazon Cognito	**Core.** Handles user management and authentication for the API.

AWS service	Description
Amazon DynamoDB	**Core.** Stores deployment information and tests scenario details and results.
Amazon Elastic Container Service	**Core.** Deploys and manages independent Amazon ECS tasks on AWS Fargate containers.
AWS Fargate	**Core.** Hosts solution's Amazon ECS containers
AWS Identity and Access Management	**Core.** Handles user role and permissions management.
AWS Lambda	**Core.** Provides logic for APIs implementation, tests results parsing, and launching workers/leader tasks.
AWS Step Functions	**Core.** Orchestrates the provisioning of Amazon ECS containers on AWS Fargate tasks in the specified regions
AWS Amplify	**Supporting.** Provides a web console powered by AWS Amplify.
Amazon CloudWatch Events	**Supporting**. Schedules tests to automatically begin at a specified date or on recurring dates.
Amazon Elastic Container Registry	**Supporting**. Hosts the container image in a public ECR repository.
AWS IoT Core	**Supporting.** Enables viewing live data for a running test by subscribi ng to the corresponding topic in AWS IoT Core.
AWS Systems Manager	**Supporting.** Provides application-level resource monitoring and visualization of resource operations and cost data.
Amazon S3	**Supporting.** Hosts the static web content, logs, metrics, and tests data.
Amazon Virtual Private Cloud	**Supporting.** Contains the solution's Amazon ECS containers running on AWS Fargate.

How Distributed Load Testing on AWS works

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

Test workflow


14
web console
17
DynamoDB table (scenarios)
AWS Step Functions task runner workflow
5
Lambda function (task status checker)
6
Lambda function (task runner)
7
Lambda function (task status checker)
S3 bucket (test result storage)
AWS IoT Core
13
Lambda function (task runner)
4
3
Step Functions rule
1
Amazon API Gateway (REST API)
Lambda function (microservices)
2
8
Lambda function (task runner)
9
Lambda function (task status checker)
15
container cluster
AWS Fargate
container
container
11
12
Amazon CloudWatch (logging)
16
Lambda function (results parser)
10
S3 bucket (scenario storage)

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-parse 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 to 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 detailed 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.

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.

K6 script support

The solution supports K6 framework-based testing. K6 is released with AGPL-3.0 license. The solution displays a license acknowledgment message when creating a new test for K6. The K6 test file along with any necessary input files can be included in an archive file and uploaded for the test scenario using the upload option.

Locust script support

The solution supports Locust framework-based testing. The Locust test file along with any necessary input files can be included in an archive file and uploaded for the test scenario using the upload option.

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.

Starting in version 3.3.0, Distributed Load Testing on AWS allows users to schedule load tests using cron expressions. Select **Run on Schedule** and then the **CRON tab** to either manually enter a cron value or use the drop-down fields. The cronExpiryDate must match the scheduled test run date. Review the **Next Run Dates (UTC)** to confirm your schedule.

Note

- Test duration: Consider the total duration of tests when scheduling. For example, a test with a 10-minute ramp-up time and 40-minute hold time will take approximately 80 minutes to complete.
- Minimum interval: Ensure the interval between scheduled tests is longer than the estimated test duration. For example, if the test takes about 80 minutes, schedule it to run no more frequently than every 3 hours.
- Hourly limitation: The system does not allow tests to be scheduled with only a one-hour difference even if the estimated test duration is less than an hour.

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.

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*.

For migrating existing users to Amazon Cognito user pools, refer to the AWS blog Approaches for migrating users to Amazon Cognito user pools.

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.

Plan your deployment

This section describes the cost, security, Regions, and other considerations prior to deploying the solution.

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 following table provides a sample cost breakdown for deploying this solution with the default parameters in the US East (N. Virginia) Region for one month.

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**

The solution resources are tagged with the Key=SolutionId and Value=SO0062. You can activate the tag key SolutionId by following the documentation activating-tags. Once the tag is activated, you can create a cost category rule by following the documentation to create cost categories. You can view the cost incurred for the solution by monitoring the cost categories console and selecting the cost category name.

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, see 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.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, 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 the AWS Cloud. This solution creates IAM roles that grant the solution's AWS Lambda functions access to create Regional resources.

Amazon CloudFront

This solution deploys a web UI hosted in an Amazon S3 bucket, which is distributed by Amazon CloudFront. To help reduce latency and improve security, this solution includes a CloudFront distribution with an origin access identity, which is a CloudFront user that provides public access to the solution website's bucket contents. By default, the CloudFront distribution uses TLS 1.2 to enforce the highest level of security protocol. For more information, refer to Restricting access to an Amazon S3 origin in the *Amazon CloudFront Developer Guide*.

CloudFront activates additional security mitigations to append HTTP security headers to each viewer response. For more information, refer to Adding or removing HTTP headers in CloudFront responses.

This solution uses the default CloudFront certificate, which has a minimum supported security protocol of TLS v1.0. To enforce the use of TLS v1.2 or TLS v1.3, you must use a custom SSL certificate instead of the default CloudFront certificate. For more information, refer to How do I configure my CloudFront distribution to use an SSL/TLS certificate.

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 everywhere, 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 when you are planning to run 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).

Supported AWS Regions

This solution uses the Amazon Cognito service, which is not currently available in all AWS Regions. For the most current availability of AWS services by Region, see the AWS Regional Services List.

Distributed Load Testing on AWS is available in the following AWS Regions:

Region name	
US East (Ohio)	Asia Pacific (Tokyo)
US East (N. Virginia)	Canada (Central)
US West (Northern California)	Europe (Frankfurt)
US West (Oregon)	Europe (Ireland)
Asia Pacific (Mumbai)	Europe (London)
Asia Pacific (Seoul)	Europe (Paris)
Asia Pacific (Singapore)	Europe (Stockholm)
Asia Pacific (Sydney)	South America (São Paulo)

Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the services implemented in this solution. For more information, see AWS service quotas.

Use the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

AWS CloudFormation quotas

Your AWS account has AWS CloudFormation quotas that you should be aware of when launching the stack in this solution. By understanding these quotas, you can avoid limitation errors that would prevent you from deploying this solution successfully. For more information, see AWS CloudFormation quotas in the in the *AWS CloudFormation User's 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 AmazonLinux image (with Blazemeter installed) 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.

Monitoring the solution post deployment

Enabling or setting up CloudWatch Alarms

It is recommended to continuously monitor the solution’s resources post-deployment. You can consider setting up CloudWatch Metrics for the resources created by the solution.

Amazon CloudFront Distribution Metrics

Review the CloudFront Distribution metrics and setting up alarms to monitor and receive notifications.

Amazon API Gateway Metrics

Review the Amazon API Gateway dimensions and metrics and setting up alarms to monitor and receive notifications.

Deploy the solution

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

Deployment process overview

Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Before you launch the solution, review the cost, architecture, network security, and other considerations discussed earlier in this guide.

Time to deploy: Approximately 15 minutes

AWS CloudFormation template

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

distributed-load-testing-on-aws.template - Use this template to launch the solution and all associated components. The default configuration deploys the core and supporting services found in the AWS services in this solution section, but you can customize the template to meet your specific needs.

Note

AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs. If you have previously deployed this solution, see Update the solution for update instructions.

Launch the stack

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

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 through this survey. Data collection is subject to the AWS Privacy Notice.
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 updated template and deploy the solution. For more information, see the Anonymized data collection section of this guide.

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. For more details, visit the Cost section in this guide and refer to the pricing webpage for each AWS service used in this solution.

1. Sign in to the AWS Management Console and select the button 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
Administrator Name	*<Requires input>*	User name for the initial solution administrator.
Administrator Email	`<Requires input>`	Email address of the administrator user. After launch, an email will be sent to this address with console login instructions.
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

Parameter	Default	Description
		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.
Provide valid CIDR block for the solution to create VPC	192.168.0.0/16	You may leave this parameter blank if you are using existing VPC
Provide valid CIDR block for subnet A for the solution to create VPC	192.168.0.0/20	CIDR block for subnet A of the AWS Fargate VPC
Provide valid CIDR block for subnet B for the solution to create VPC	192.168.16.0/20	CIDR block for subnet B of the AWS Fargate VPC
Provide CIDR block for allowing outbound traffic of Fargate tasks	0.0.0.0/0	CIDR block that restricts Amazon ECS container outbound access.

Parameter	Default	Description
Auto-Update Container Image	`Yes`	Automatically use the most up to date and secure image up until the next minor release. Selecting No will pull the image as originall y released, without any security updates.

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 where 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 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

Parameter	Default	Description
		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.
Provide valid CIDR block for the solution to create VPC	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.
Provide CIDR block for allowing outbound traffic of Fargate tasks	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 will 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 view the aggregate results of all Regions in a multi-Region test only 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.

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.

Update the solution

If you have previously deployed the solution, follow this procedure to update the solution's CloudFormation stack to get the latest version of the solution's framework.

1. Sign in to the CloudFormation console, select your existing CloudFormation stack, and select **Update stack**.
2. Select **Make a direct update**.
3. Select **Replace existing template**.
4. Under **Specify template**:
   a. 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.
   e. Choose **Next**.
   f. Choose **Next** again.
5. Under **Parameters**, review the parameters for the template and modify them as necessary. Refer to Launch the stack for details about the parameters.
6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings.
9. Select the box acknowledging that the template might create IAM resources.
10. Choose **View change set** and verify the changes.
11. 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 a `UPDATE_COMPLETE` status in approximately 15 minutes.

When updating from DLT versions older than v3.2.6 to v3.3.x and v3.3.x to latest, updating the stack fails

1. Download the distributed-load-testing-on-aws.template.

2. Open the template and navigate to Conditions: and look for DLTCommonResourcesAppRegistryCondition
3. You should see something similar to 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.

Troubleshooting

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

Known issue resolution

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.
```

- Resolution:

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.

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

- Resolution:

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.

Issue: Tests are starting but failing to complete or the state of the ECS tasks is unknown

- Resolution:

If you selected the option to provide an existing VPC in the account where the solution has been deployed, ensure that the VPC being used by the ECS Tasks has enough free IP addresses to start the number of tasks provided in the test input. The ECS task definition uses the ECR image that needs an internet gateway or a route to the internet so that the ECS service can provision the tasks by downloading the solution ECR image from aws-solutions/distributed-load-testing-on-aws-load-tester. If you cannot provide a route to the internet since all subnets in the VPC are private, you can host the ECR image in your account using ECR pull through cache. Update the task definition with the new ECR image URI and create a new revision. Once the task definition is updated, the solution configuration in the DynamoDB table needs to be updated to use the new revision. The DynamoDB table name can be found in the CloudFormation stack outputs tab under the key ScenariosTable. Update the attribute taskDefinition for the item with the key testId and value region-[SOLUTION-DEPLOYED-REGION].

Issue: Tests need to use an endpoint which is private or not available through the internet gateway

- Resolution:

When testing private API endpoints that aren't accessible through the internet gateway, consider the following approaches:

1. **Network Configuration**: Ensure the subnet route tables used by the ECS tasks are updated with a route to the IP address range of the private endpoint being tested. This allows the test traffic to reach the private endpoint within your VPC.
2. **DNS Resolution**: For custom domains, configure the DNS settings in your VPC to resolve the private endpoint's domain name. Refer to VPC DNS documentation for detailed instructions.
3. **VPC Endpoints**: If testing AWS services, consider using VPC endpoints (AWS PrivateLink) to establish private connectivity. For example, to test a private API Gateway, you can create a VPC endpoint for API Gateway. See Private API Gateway documentation.
4. **VPC Peering**: If the private endpoint is in a different VPC, establish VPC peering between the VPC where the solution is deployed and the VPC containing the private endpoint. Configure appropriate route tables in both VPCs. See VPC Peering documentation.
5. **Transit Gateway**: For more complex networking scenarios involving multiple VPCs, consider using AWS Transit Gateway to route traffic between the solution's VPC and the VPC containing the private endpoint. See Transit Gateway documentation.
6. **Security Groups**: Ensure that the security groups associated with your ECS tasks allow outbound traffic to the private endpoint, and the security groups of the private endpoint allow inbound traffic from the ECS tasks.

For testing internal Application Load Balancers or EC2 instances, ensure that the VPC CIDR ranges don’t overlap and that the necessary routes are configured in the route tables.

Issue: Tests are completing but the results are not available on the UI

- Resolution:

If the test has completed but the results are not available in the UI, the result files should still be available in the S3 Bucket from the ECS tasks which ran the tests. This is a known limitation in the solution. In the current architecture, the solution uses a result parsing Lambda function to summarize the results from multiple ECS tasks, which are then stored as an item in the DynamoDB table. The DynamoDB table has a limit of 400 KB maximum item size. This limitation is reached depending on the complexity of the test script, the concurrency, and the number of tasks being used. The error does not mean the test is failing; it indicates that the process to summarize the results and store them in the DynamoDB table for CRUD operations has failed. The results are still available in the S3 bucket for the test scenario.

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 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 buckets (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. Select the S3 bucket you just emptied and choose **Delete**.
7. Enter the S3 bucket name in the verification field and choose **Delete bucket**.

Repeat steps 4 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
```

Use the solution

This section includes information on how to use the Distributed Load Testing on AWS solution, including Test results, Test scheduling workflow, and Live data.

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 second 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*.

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 that 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 test workflow.
- 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 its 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 container 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.

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 the 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.

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.

Developer guide

This section provides the source code for the solution and additional customizations.

Source code

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

Maintenance

This solution uses Docker images with fixed versions that match each solution release if automatic updates is not selected. The AWS Solutions team uses ECR Enhanced Scanning to detect Common Vulnerabilities and Exposures (CVEs) in the base image and installed packages. When possible, the team will publish patched images with the same version tag to resolve CVEs, without breaking compatibility with the released solution version. When images are patched, if they are on the same minor version, the stable tag will be automatically updated, and an additional image tag will be created in the format `<solution-version>_<date-of-fix>`. If a major or minor version is released, a full stack update will be required to get the latest image version as the stable tag will be incremented so that its version matches the version of the solution. If opting-in to automatic updates the changes to the image, including the CVEs and minor bug fixes, will automatically be applied to the image up to the latest matching minor release.

Versions

Customers on the latest solution version will receive security patches and minor, non-breaking, bug fixes automatically if they opt-in to automatic image updates. The image will automatically pull the latest image up to the latest matching minor version. In order to lock the container to a specific version, the task definition can be edited to specify the container to use a specific image version by using the tagged version of the image. Automatic updates can also be turned off by selecting **No** to automatic updates in CloudFormation when launching the stack. This will launch the image version matching the solution version.

Container image customization

This solution uses a public Amazon Elastic Container Registry (Amazon ECR) image repository managed by AWS to store the 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.

```
#!/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.

```
#!/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, v3.4.0
```

The following example shows the container file.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:2023-minimal

RUN dnf update -y && \
    dnf install -y python3.11 python3.11-pip java-21-amazon-corretto bc procps jq
 findutils unzip && \
    dnf clean all

ENV PIP_INSTALL="pip3.11 install --no-cache-dir"


# install bzt
RUN $PIP_INSTALL --upgrade bzt awscli setuptools==78.1.1 h11 urllib3==2.2.2 && \
    $PIP_INSTALL --upgrade bzt
COPY ./.bzt-rc /root/.bzt-rc
RUN chmod 755 /root/.bzt-rc

# install bzt tools
RUN bzt -install-tools -o modules.install-
checker.exclude=selenium,gatling,tsung,siege,ab,k6,external-results-
loader,locust,junit,testng,rspec,mocha,nunit,xunit,wdio,robot,newman
RUN rm -rf /root/.bzt/selenium-taurus
RUN mkdir /bzt-configs /tmp/artifacts
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
RUN chmod 755 /bzt-configs/jar_updater.py
RUN python3.11 /bzt-configs/jar_updater.py

# Remove jar files from /tmp
RUN rm -rf /tmp/jmeter-plugins-manager-1* && \
    rm -rf /usr/local/lib/python3.11/site-packages/setuptools-65.5.0.dist-info && \
    rm -rf /usr/local/lib/python3.11/site-packages/urllib3-1.26.17.dist-info

# Add settings file to capture the output logs from bzt cli
```

```
RUN mkdir -p /etc/bzt.d && echo '{"settings": {"artifacts-dir": "/tmp/artifacts"}}' > /
etc/bzt.d/90-artifacts-dir.json

WORKDIR /bzt-configs
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/Blazemeter 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}"
echo "MAIN_STACK_REGION:: ${MAIN_STACK_REGION}"

cat /proc/self/cgroup
TASK_ID=$(grep -oE '[a-f0-9]{32}' /proc/self/cgroup | head -n 1)
echo $TASK_ID

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 --region
 $MAIN_STACK_REGION

# Set the default log file values to jmeter
LOG_FILE="jmeter.log"
```

```
OUT_FILE="jmeter.out"
ERR_FILE="jmeter.err"
KPI_EXT="jtl"

# download JMeter jmx file
if [ "$TEST_TYPE" != "simple" ]; then
  # setting the log file values to the test type
  LOG_FILE="${TEST_TYPE}.log"
  OUT_FILE="${TEST_TYPE}.out"
  ERR_FILE="${TEST_TYPE}.err"

  # set variables based on TEST_TYPE
  if [ "$TEST_TYPE" == "jmeter" ]; then
    EXT="jmx"
    TYPE_NAME="JMeter"
    # 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 $JMETER_LIB_PATH
  elif [ "$TEST_TYPE" == "k6" ]; then
    curl --output /tmp/artifacts/k6.rpm https://dl.k6.io/rpm/x86_64/k6-v0.58.0-
amd64.rpm
    rpm -ivh /tmp/artifacts/k6.rpm
    dnf install -y k6
    rm -rf /tmp/artifacts/k6.rpm
    EXT="js"
    KPI_EXT="csv"
    TYPE_NAME="K6"
  elif [ "$TEST_TYPE" == "locust" ]; then
    EXT="py"
    TYPE_NAME="Locust"

  fi

  if [ "$FILE_TYPE" != "zip" ]; then
    aws s3 cp s3://$S3_BUCKET/public/test-scenarios/$TEST_TYPE/$TEST_ID.$EXT ./ --
region $MAIN_STACK_REGION
  else
    aws s3 cp s3://$S3_BUCKET/public/test-scenarios/$TEST_TYPE/$TEST_ID.zip ./ --region
 $MAIN_STACK_REGION
    unzip $TEST_ID.zip
    echo "UNZIPPED"
    ls -l
    # If zip and locust, make sure to pick locustfile
    if [ "$TEST_TYPE" != "locust" ]; then
      TEST_SCRIPT=$(find . -name "*.${EXT}" | head -n 1)
    else
      TEST_SCRIPT=$(find . -name "locustfile.py" | head -n 1)
    fi
    # only looks for the first test script file.
    TEST_SCRIPT=`find . -name "*.${EXT}" | head -n 1`
    echo $TEST_SCRIPT
    if [ -z "$TEST_SCRIPT" ]; then
      echo "There is no test script (.${EXT}) in the zip file."
      exit 1
    fi

    sed -i -e "s|$TEST_ID.$EXT|$TEST_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
    python3.11 -u $SCRIPT  $TIMEOUT &
    pypid=$!
    wait $pypid
    pypid=0
else
```

```
    aws s3 cp s3://$S3_BUCKET/Container_IPs/${TEST_ID}_IPHOSTS_${AWS_REGION}.txt ./ --
region $MAIN_STACK_REGION
    export IPHOSTS=$(cat ${TEST_ID}_IPHOSTS_${AWS_REGION}.txt)
    python3.11 -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 $TEST_ID.$EXT | grep filename > results.txt
  else
    cat $TEST_SCRIPT | grep filename > results.txt
  fi

  if [ -f results.txt ]; then
    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`)
    extensions=()
    for f in "${files[@]}"; do
      ext="${f##*.}"
      if [[ ! " ${extensions[@]} " =~ " ${ext} " ]]; then
        extensions+=("$ext")
      fi
    done

    # Find all files in the current folder with the same extensions
    all_files=()
    for ext in "${extensions[@]}"; do
      for f in *."$ext"; do
        all_files+=("$f")
      done
    done

    for f in "${all_files[@]}"; do
      p="s3://$S3_BUCKET/results/$TEST_ID/${TYPE_NAME}_Result/$PREFIX/$UUID/$f"
      if [[ $f = /* ]]; then
        p="s3://$S3_BUCKET/results/$TEST_ID/${TYPE_NAME}_Result/$PREFIX/$UUID$f"
      fi

        echo "Uploading $p"
        aws s3 cp $f $p --region $MAIN_STACK_REGION
    done
    fi
fi

if [ -f /tmp/artifacts/results.xml ]; then

  # Insert the Task ID at the same level as <FinalStatus>
  curl -s $ECS_CONTAINER_METADATA_URI_V4/task
  Task_CPU=$(curl -s $ECS_CONTAINER_METADATA_URI_V4/task | jq '.Limits.CPU')
  Task_Memory=$(curl -s $ECS_CONTAINER_METADATA_URI_V4/task | jq '.Limits.Memory')
  START_TIME=$(curl -s "$ECS_CONTAINER_METADATA_URI_V4/task" | jq -r
 '.Containers[0].StartedAt')
  # Convert start time to seconds since epoch
  START_TIME_EPOCH=$(date -d "$START_TIME" +%s)
  # Calculate elapsed time in seconds
  CURRENT_TIME_EPOCH=$(date +%s)
  ECS_DURATION=$((CURRENT_TIME_EPOCH - START_TIME_EPOCH))


  sed -i.bak 's/<\/FinalStatus>/<TaskId>'"$TASK_ID"'<\/TaskId><\/FinalStatus>/' /tmp/
artifacts/results.xml
  sed -i 's/<\/FinalStatus>/<TaskCPU>'"$Task_CPU"'<\/TaskCPU><\/FinalStatus>/' /tmp/
artifacts/results.xml
  sed -i 's/<\/FinalStatus>/<TaskMemory>'"$Task_Memory"'<\/TaskMemory><\/
FinalStatus>/' /tmp/artifacts/results.xml
  sed -i 's/<\/FinalStatus>/<ECSDuration>'"$ECS_DURATION"'<\/ECSDuration><\/
FinalStatus>/' /tmp/artifacts/results.xml

  echo "Validating Test Duration"
  TEST_DURATION=$(grep -E '<TestDuration>[0-9]+.[0-9]+</TestDuration>' /tmp/artifacts/
results.xml | sed -e 's/<TestDuration>//' | sed -e 's/<\/TestDuration>//')

  if (( $(echo "$TEST_DURATION > $CALCULATED_DURATION" | bc -l) )); then
```

```
    echo "Updating test duration: $CALCULATED_DURATION s"
    sed -i.bak.td 's/<TestDuration>[0-9]*\.[0-9]*<\/TestDuration>/
<TestDuration>'"$CALCULATED_DURATION"'<\/TestDuration>/' /tmp/artifacts/results.xml
  fi

  if [ "$TEST_TYPE" == "simple" ]; then
    TEST_TYPE="jmeter"
  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 --region $MAIN_STACK_REGION
  aws s3 cp /tmp/artifacts/bzt.log s3://$S3_BUCKET/results/${TEST_ID}/bzt-${PREFIX}-
${UUID}-${AWS_REGION}.log --region $MAIN_STACK_REGION
  aws s3 cp /tmp/artifacts/$LOG_FILE s3://$S3_BUCKET/results/${TEST_ID}/${TEST_TYPE}-
${PREFIX}-${UUID}-${AWS_REGION}.log --region $MAIN_STACK_REGION
  aws s3 cp /tmp/artifacts/$OUT_FILE s3://$S3_BUCKET/results/${TEST_ID}/${TEST_TYPE}-
${PREFIX}-${UUID}-${AWS_REGION}.out --region $MAIN_STACK_REGION
  aws s3 cp /tmp/artifacts/$ERR_FILE s3://$S3_BUCKET/results/${TEST_ID}/${TEST_TYPE}-
${PREFIX}-${UUID}-${AWS_REGION}.err --region $MAIN_STACK_REGION
  aws s3 cp /tmp/artifacts/kpi.${KPI_EXT} s3://$S3_BUCKET/results/${TEST_ID}/kpi-
${PREFIX}-${UUID}-${AWS_REGION}.${KPI_EXT} --region $MAIN_STACK_REGION

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.

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
`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`)

Name	Description
`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`)
`cronvalue`	The cron value for customizing recurring scheduling. If used, omit scheduleDate and scheduleTime.
`cronExpiryDate`	Required date so the cron expires and doesn't run indefinitely.
`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

Name	Description
`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`)
`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`)
`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

Name	Description
`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
`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
`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

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, `[replaceable]`<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: `[replaceable]`<stackName>`` ` ``-EcsTaskDefinition-*`<system-generated-random-Hash>`*:[replaceable]<`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, `[replaceable]`<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.

Reference

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

Anonymized data collection

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

- **Solution ID** - The AWS solution identifier
- **Unique ID (UUID)** - Randomly generated, unique identifier for each 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 Deploy the solution section of this guide.

Contributors

- Tom Nightingale
- Fernando Dingler
- Beomseok Lee
- George Lenz
- Erin McGill
- Dimitri Lopez
- Kamyar Ziabari
- Bassem Wanis
- Garvit Singh
- Nikhil Reddy
- Simon Kroll

Revisions

Visit the CHANGELOG.md in our GitHub repository to track version-specific improvements and fixes.

Notices