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: *December 2025*

Distributed Load Testing on AWS helps you automate performance testing of your software applications at scale to identify bottlenecks before you release your application. This solution simulates thousands of connected users generating HTTP requests at a sustained rate without the need to provision servers.

This solution leverages Amazon Elastic Container Service (Amazon ECS) on AWS Fargate to deploy containers that run your load test simulations and offers the following capabilities:

- Deploy Amazon ECS on AWS Fargate containers that run independently to test the load capacity of your application.
- Simulate tens of thousands of concurrent users across multiple AWS Regions generating requests at a continuous pace.
- Customize your application tests using JMeter, K6, Locust test scripts, or simple HTTP endpoint configuration.
- Schedule load tests to run immediately, at a future date and time, or on a recurring schedule.
- Run multiple load tests concurrently across different scenarios and regions.

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.<br><br>The estimated cost for running this solution in the US East (N. Virginia) Region is USD $ 30.90 per month for AWS resources. | Cost |
| 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 |
| Learn about the optional MCP Server for AI-assisted load testing analysis. | MCP server integration |
| 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:

**Multiple Test Framework Support**

Supports JMeter, K6, and Locust test scripts, as well as simple HTTP endpoint testing without requiring custom scripts. For more information, refer to Test types in the Architecture details section.

**High User Load Simulation**

Simulates tens of thousands of concurrent virtual users to stress test your application under realistic load conditions.

**Multi-Region Load Distribution**

Distributes load tests across multiple AWS Regions to simulate geographically distributed user traffic and assess global performance.

**Flexible Test Scheduling**

Schedules tests to run immediately, at a specific future date and time, or on a recurring schedule using cron expressions for automated regression testing.

**Real-Time Monitoring**

Provides optional live data streaming to monitor test progress with real-time metrics including response times, virtual user counts, and request success rates.

**Comprehensive Test Results**

Displays detailed test results with performance metrics, percentiles (p50, p90, p95, p99), error analysis, and downloadable artifacts for offline analysis.

**Baseline Comparison**

Designates baseline test runs for performance comparison to track improvements or regressions over time.

**Endpoint Flexibility**

Tests any HTTP or HTTPS endpoint across AWS Regions, on-premises environments, or other cloud providers.

**Intuitive Web Console**

Provides a web-based console for creating, managing, and monitoring tests with no command-line interaction required.

**AI-Assisted Analysis (Optional)**

Integrates with AI development tools through the Model Context Protocol (MCP) server for intelligent analysis of load testing data.

**Multiple Protocol Support**

Supports various protocols including HTTP, HTTPS, WebSocket, JDBC, JMS, FTP, and gRPC through custom test scripts.

# Benefits

The solution provides the following benefits:

**Comprehensive Performance Testing**

Supports load testing, stress testing, and endurance testing to thoroughly evaluate application performance under various conditions.

**Early Issue Detection**

Identifies performance bottlenecks, memory leaks, and scalability issues before production deployment, reducing the risk of outages.

**Real-World Usage Simulation**

Accurately simulates real-world user behavior and traffic patterns to validate application performance under realistic conditions.

**Actionable Performance Insights**

Provides detailed metrics, percentiles, and error analysis to understand application behavior and guide optimization efforts.

**Automated Testing Workflows**

Enables scheduled and recurring tests for continuous performance monitoring and regression testing without manual intervention.

**Cost-Efficient Infrastructure**

Uses serverless AWS Fargate containers with pay-per-use pricing, eliminating the need for dedicated testing infrastructure and ongoing subscription fees.

**Rapid Test Deployment**

Deploys and scales test infrastructure in minutes without provisioning or managing servers.

**Easy Interrogation of Test Results**

Integrates with AI development tools through an optional Model Context Protocol (MCP) server, enabling natural language queries and intelligent analysis of load testing data for faster insights and troubleshooting.

# Use cases

### Pre-Production Validation

Test web and mobile applications under production-like load conditions before launching a new version to validate performance and identify issues.

### Capacity Planning

Determine the maximum number of concurrent users your application can support with current infrastructure and identify when scaling is required.

### Peak Load Verification

Verify that your infrastructure can handle peak loads, seasonal traffic spikes, or unexpected surges in demand without performance degradation.

### Performance Optimization

Identify performance bottlenecks such as slow database queries, inefficient code execution, network latency, or resource constraints.

### Regression Testing

Schedule recurring load tests to detect performance regressions introduced by new code deployments or infrastructure changes.

### Global Performance Assessment

Evaluate application performance from multiple geographic regions to ensure consistent user experience for a global audience.

### API Load Testing

Test REST APIs, GraphQL endpoints, or microservices to validate response times, throughput, and error rates under load.

### CI/CD Pipeline Integration

Integrate automated performance testing into continuous integration and deployment pipelines to catch performance issues early in the development cycle.

**Third-Party Service Testing**

Test the performance and reliability of third-party APIs or services that your application depends on under various load conditions.

# 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 concurrency (number of concurrent virtual users per task). The recommended concurrency based on default settings is 200. Concurrency is limited by CPU and memory. For tests based on Apache JMeter, higher concurrency increases 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 concurrency values for 1 task and monitor the ECS CloudWatch metrics for the Task Cluster. Refer to Amazon ECS cluster utilization metrics.

**ramp-up**

The time period to gradually increase from zero to the target concurrency level.

**hold for**

The time period to maintain the target concurrency level after ramp-up completes.

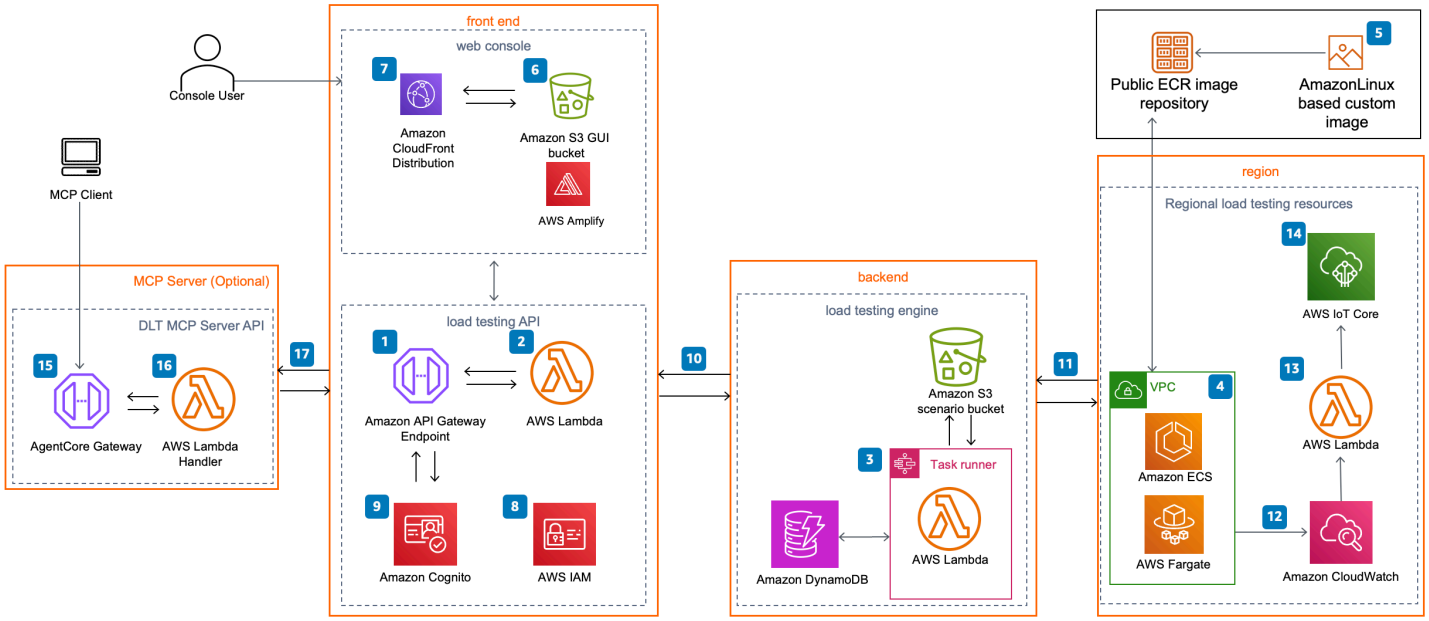
For a general reference of AWS terms, see the AWS Glossary.

# 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 leverages Amazon API Gateway to invoke the solution's microservices (AWS Lambda functions).
2. The microservices provide the business logic to manage test data and run the tests.

3. These microservices interact with Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, and AWS Step Functions to store test scenario details and results and to orchestrate test execution.
4. An Amazon Virtual Private Cloud (Amazon VPC) network topology deploys containing the solution's Amazon Elastic Container Service (Amazon ECS) containers running on AWS Fargate.
5. The containers use an Amazon Linux 2023 base image with the Taurus load testing framework installed. Taurus is an open-source test automation framework that supports JMeter, K6, Locust, and other testing tools. The container image is Open Container Initiative (OCI) compliant and hosted by AWS in an Amazon Elastic Container Registry (Amazon ECR) public repository. For more information, refer to Container image customization.
6. A web console powered by AWS Amplify deploys into an 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, the solution creates a default 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, the distributed load tester API, and the MCP Server.
10. After you deploy this solution, you can use the web console or APIs to create and run test scenarios that define a series of tasks.
11. The microservices use this test scenario to run ECS tasks on Fargate in the specified Regions.
12. When the test completes, the solution stores results in S3 and DynamoDB and logs output in Amazon CloudWatch.
13. If you enable the live data option, the solution sends CloudWatch logs from the Fargate tasks to a Lambda function during the test for each Region where the test runs.
14. The Lambda function 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 displays real-time data while the test runs.

> **Note**
>
> The following steps describe the optional MCP Server integration for AI-assisted load testing analysis. This component is only deployed if you select the MCP Server option during solution deployment.

15. An MCP client (AI development tool) connects to the AWS AgentCore Gateway endpoint to access the Distributed Load Testing solution's data through the Model Context Protocol. AgentCore Gateway validates the user's Cognito authentication token to ensure authorized access to the MCP server.
16. Upon successful authentication, AgentCore Gateway forwards the MCP tool request to the DLT MCP Server Lambda function. The Lambda function returns the structured data to AgentCore Gateway, which sends it back to the MCP client for AI-assisted analysis and insights.
17. The Lambda function processes the request and queries the appropriate AWS resources (DynamoDB tables, S3 buckets, or CloudWatch logs) to retrieve the requested load testing data.

# 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.

- All resources are defined as infrastructure as code using AWS CloudFormation templates generated from AWS CDK constructs.
- The solution pushes metrics to CloudWatch at various stages to provide observability into Lambda functions, ECS tasks, S3 buckets, and other solution components.

## Security

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

- Cognito authenticates and authorizes web console users and API requests.
- All interservice communications use AWS Identity and Access Management (IAM) roles with least privilege access, containing only the minimum permissions required.

- All data storage, including S3 buckets and DynamoDB tables, encrypts data at rest using AWS managed keys.
- Logging, tracing, and versioning are enabled where applicable for audit and compliance purposes.
- Network access is private by default with VPC endpoints enabled where available to keep traffic within the AWS network.

**Note**

The solution creates multiple CloudWatch log groups with varying retention periods based on log volume and cost considerations:

| Log Type | Retention Period |
|---|---|
| ECS container insights | 1 day |
| Step Functions, ECS custom logs, API Gateway access logs | 1 year |
| Lambda runtime logs | 2 years |
| API Gateway execution logs | Never expire |

You can modify these retention periods in the CloudWatch console based on your requirements.

## 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.
- AWS AgentCore Gateway serves as a cost-effective Lambda-based proxy to the distributed load testing API, eliminating the need for dedicated infrastructure and reducing costs through serverless pay-per-request pricing.

# 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 three high-level components: a front end, a backend, and an optional MCP Server.

## Front end

The front end provides the interfaces for interacting with the solution and includes:

- A load testing API for programmatic access
- A web console for creating, scheduling, and running performance tests
- An optional MCP Server for AI-assisted analysis of test results and errors

### Load testing API

Distributed Load Testing on AWS configures Amazon API Gateway to host the solution's RESTful API. Users can interact with the load testing system securely through the included web console, RESTful API, and optional MCP Server. 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 built with Cloudscape, an open-source design system for building intuitive web applications. The console is hosted in Amazon S3 and accessed through Amazon CloudFront. The application leverages AWS Amplify to integrate with Amazon Cognito to authenticate users. The web console also contains an option to view live data for a running test, in which it subscribes to the corresponding topic in AWS IoT Core.

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

## MCP Server (Optional)

The optional Model Context Protocol (MCP) Server provides an additional interface for AI development tools to access and analyze load testing data through natural language interactions. This component is only deployed if you select the MCP Server option during solution deployment.

The MCP Server enables AI agents to query test results, analyze performance metrics, and gain insights into your load testing data using tools like Amazon Q, Claude, and other MCP-compatible AI assistants. For detailed information about the MCP Server architecture and configuration, refer to MCP Server in this section.

# Backend

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

## Container image pipeline

This solution uses a container image built with Amazon Linux 2023 as the base image with the Taurus load testing framework installed. Taurus is an open-source test automation framework that supports JMeter, K6, Locust, and other testing tools. AWS hosts this image in an Amazon Elastic Container Registry (Amazon ECR) public repository. The solution uses this image to run tasks in the Amazon ECS on AWS Fargate cluster.

For more information, refer to the Container image customization section of this guide.

## Testing infrastructure

In addition to the main CloudFormation template, the solution provides a regional template to launch the required resources for running tests in multiple Regions. The solution stores this template in Amazon S3 and provides a link to it in the web console. Each regional stack includes a VPC, an AWS Fargate cluster, and a Lambda function for processing live data.

For more information on how to deploy testing infrastructure in additional Regions, refer to the Multi-Region deployment section of this guide.

## Load testing engine

The Distributed Load Testing solution uses Amazon Elastic Container Service (Amazon ECS) and AWS Fargate to simulate thousands of concurrent users across multiple Regions, generating HTTP requests at a sustained rate.

You define the test parameters using the included web console. The solution uses these parameters to generate a JSON test scenario and stores it in Amazon S3. For more information about test scripts and testing parameters, refer to Test types in this section.

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.

# MCP Server

The optional Model Context Protocol (MCP) Server integration enables AI agents to programmatically access and analyze your load testing data through natural language interactions. This component is only deployed if you select the MCP Server option during solution deployment.

The MCP Server acts as a bridge between AI development tools and your DLT deployment, providing a standardized interface for intelligent analysis of performance testing results. The architecture integrates several AWS services to create a secure, scalable interface for AI agent interactions:

## AWS AgentCore Gateway

AWS AgentCore Gateway is a fully managed service that provides standardized hosting and protocol management for MCP servers. In this solution, AgentCore Gateway serves as the public endpoint that AI agents connect to when requesting access to your load testing data.

The service handles all MCP protocol communication, including tool discovery, authentication token validation, and request routing. AgentCore Gateway operates as a multi-tenant service with built-in security protections against common threats to public endpoints, while validating Cognito token signatures and claims for each request.

## DLT MCP Server Lambda

The DLT MCP Server Lambda function is a custom serverless component that processes MCP requests from AI agents and translates them into queries against your DLT resources.

This Lambda function acts as the intelligence layer of the MCP integration, retrieving test results from DynamoDB tables, accessing performance artifacts stored in S3 buckets, and querying CloudWatch logs for detailed execution information. The Lambda function implements read-only access patterns and transforms raw DLT data into structured, AI-friendly formats that agents can easily interpret and analyze.

## Authentication integration

The authentication system leverages your existing Cognito user pool infrastructure to maintain consistent access controls across both the web console and MCP Server interfaces.

This integration uses OAuth 2.0 token-based authentication. Users authenticate once through the Cognito login process and receive tokens that work for both UI interactions and MCP Server access. The system maintains the same permission boundaries and access controls as the web interface, ensuring that users can only access through AI agents the same load testing data they can access through the console.

# AWS services in this solution

The following AWS services are included in this solution:

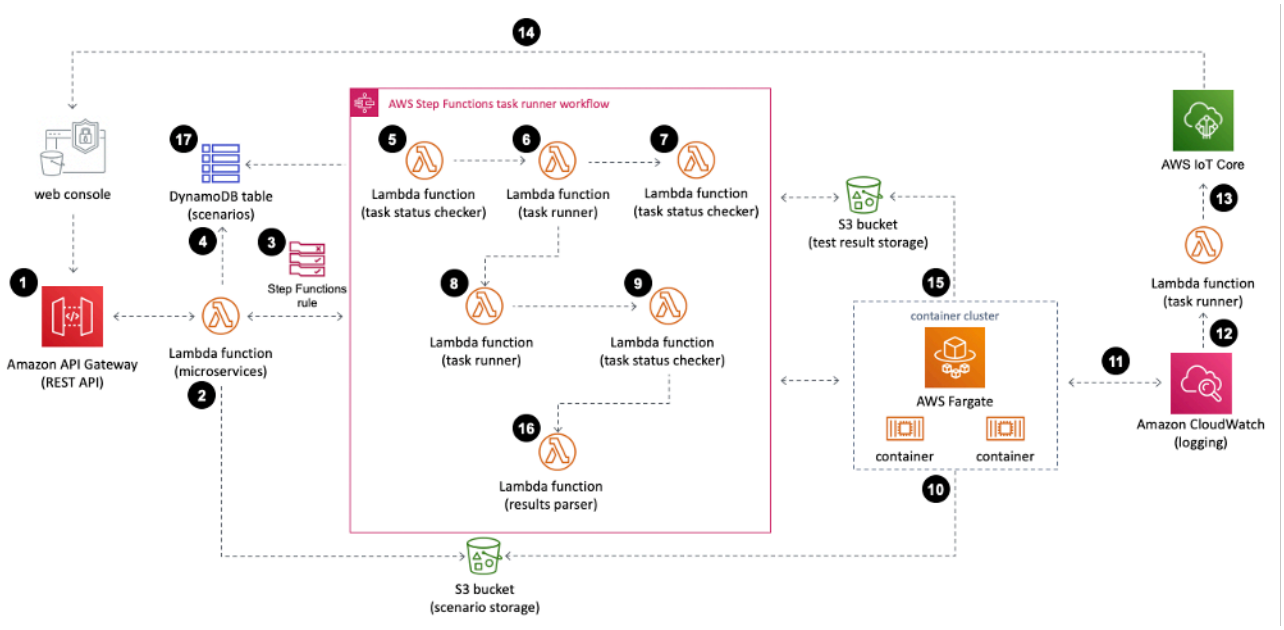
| AWS service | Description |
|---|---|
| Amazon API Gateway | **Core.** Hosts REST API endpoints in the solution. |
| AWS CloudFormation | **Core.** Manages deployments for the solution infrastructure. |
| Amazon CloudFront | **Core.** Serves the web content hosted in Amazon S3. |
| Amazon CloudWatch | **Core.** Stores the solution logs and metrics. |
| Amazon Cognito | **Core.** Handles user management and authentication for the API. |
| 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 service | Description |
| --- | --- |
| AWS IoT Core | **Supporting.** Enables viewing live data for a running test by subscribing 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. |
| Amazon Bedrock AgentCore | **Supporting, Optional.** Hosts the solution's optional Remote Model Context Protocol (MCP) Server for AI agent integration with API. |

# How Distributed Load Testing on AWS works

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

### Test workflow



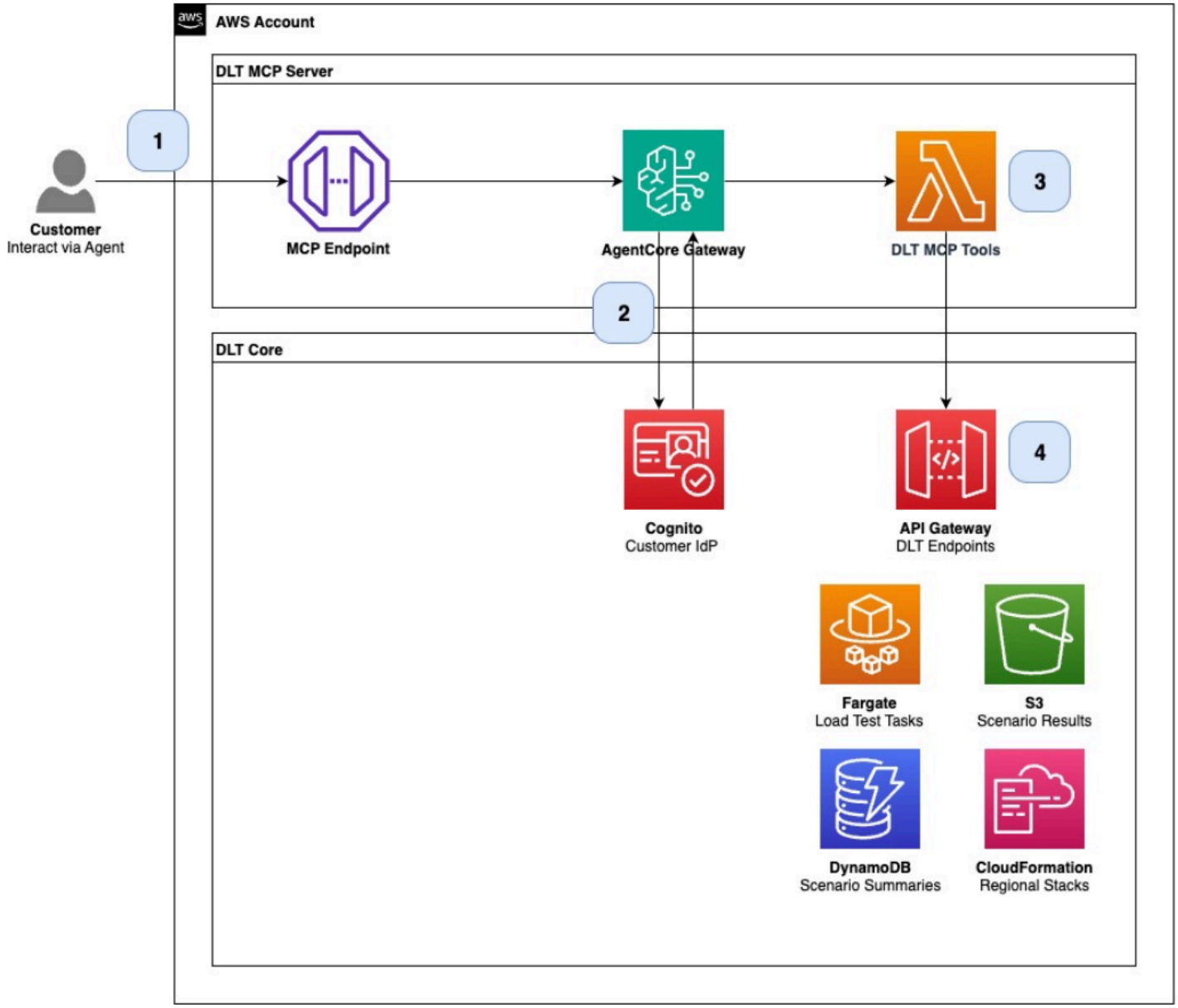
1. You use the web console to submit a test scenario that includes the configuration details to the solution's API.
2. The test scenario configuration is uploaded to the Amazon Simple Storage Service (Amazon S3) as a JSON file (`s3://<bucket-name>/test-scenarios/<$TEST_ID>/<$TEST_ID>.json`).
3. An AWS Step Functions state machine runs using the test ID, task count, test type, and file type as the AWS Step Functions state machine input. If the test is scheduled, it will first create a CloudWatch Events rule, which triggers AWS Step Functions on the specified date. For more details on the scheduling workflow, refer to the Test scheduling workflow section of this guide.
4. Configuration details are stored in the scenarios Amazon DynamoDB table.
5. In the AWS Step Functions task runner workflow, the task-status-checker AWS Lambda function checks if Amazon Elastic Container Service (Amazon ECS) tasks are already running for the same test ID. If tasks with the same test ID are found running, it causes an error. If there are no Amazon ECS tasks running in the AWS Fargate cluster, the function returns the test ID, task count, and test type.
6. The task-runner AWS Lambda function gets the task details from the previous step and runs the Amazon ECS worker tasks in the AWS Fargate cluster. The Amazon ECS API uses the RunTask action to run the worker tasks. These worker tasks are launched and then wait for a start message from the leader task in order to begin the test. The RunTask action is limited to 10 tasks per definition. If your task count is more than 10, the task definition will run multiple times until all worker tasks have been started. The function also generates a prefix to distinguish the current test in the results-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.

## MCP Server workflow (Optional)

If you deploy the optional MCP Server integration, AI agents can access and analyze your load testing data through the following workflow:

### MCP Server architecture




1. **Customer interaction** - The customer interacts with DLT's MCP via the MCP Endpoint hosted by AWS AgentCore Gateway. AI agents connect to this endpoint to request access to load testing data.
2. **Authorization** - AgentCore Gateway handles authorization against the Solution Cognito user pool application client. The gateway validates the user's Cognito token to ensure they have permission to access the DLT MCP server. Authorized users are granted access with agent tool access limited to read-only operations.

3. **Tool specification** - AgentCore Gateway connects to the DLT MCP Server Lambda function. A tool specification defines the available tools that AI agents can use to interact with your load testing data.
4. **Read-only API access** - The Lambda function is scoped to read-only API access through the existing DLT API Gateway endpoints. The function provides four primary operations:
   - **List scenarios** - Retrieve a list of test scenarios from the DynamoDB scenarios table
   - **Get scenario test results** - Access detailed test results for specific scenarios from DynamoDB and S3
   - **Get Fargate load test runners** - Query information about running Fargate tasks in the ECS cluster
   - **Get available Regional stacks** - Retrieve information about deployed regional infrastructure from CloudFormation

The MCP Server integration leverages the existing DLT infrastructure (API Gateway, Cognito, DynamoDB, S3) to provide secure, read-only access to test data for AI-powered analysis and insights.

# Design considerations

This section describes important design decisions and configuration options for the Distributed Load Testing on AWS solution, including supported applications, test types, scheduling options, and deployment considerations.

## Supported applications

This solution supports testing cloud-based applications and on-premises applications as long as you have network connectivity from your AWS account to your application. The solution supports APIs that use HTTP or HTTPS protocols.

## Test types

Distributed Load Testing on AWS supports multiple test types: simple HTTP endpoint tests, JMeter, K6, and Locust.

### Simple HTTP endpoint tests

The web console provides an HTTP Endpoint Configuration interface that allows you to test any HTTP or HTTPS endpoint without writing custom scripts. You define the endpoint URL, select the HTTP method (GET, POST, PUT, DELETE, etc.) from a dropdown menu, and optionally add custom request headers and body payloads. This configuration enables you to test APIs with custom authorization tokens, content types, or any other HTTP headers and request bodies required by your application.

### JMeter tests

When creating a test scenario using the web console, you can upload a JMeter test script. The solution uploads the script to the scenarios S3 bucket. When Amazon ECS tasks run, they download the JMeter script from S3 and execute the test.

> **Important**
>
> Although your JMeter script may define concurrency (virtual users), transaction rates (TPS), ramp-up times, and other load parameters, the solution will override these configurations with the values you specify in the Traffic Shape screen during test creation. The Traffic Shape configuration controls the task count, concurrency (virtual users per task), ramp-up duration, and hold duration for the test execution.

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 tests

The solution supports K6 framework-based testing. K6 is released under the AGPL-3.0 license. The solution displays a license acknowledgment message when creating a new K6 test. You can upload the K6 test file along with any necessary input files in an archive file.

> **Important**
>
> Although your K6 script may define concurrency (virtual users), stages, thresholds, and other load parameters, the solution will override these configurations with the values you specify in the Traffic Shape screen during test creation. The Traffic Shape configuration controls the task count, concurrency (virtual users per task), ramp-up duration, and hold duration for the test execution.

### Locust tests

The solution supports Locust framework-based testing. You can upload the Locust test file along with any necessary input files in an archive file.

> **Important**
>
> Although your Locust script may define concurrency (user count), spawn rate, and other load parameters, the solution will override these configurations with the values you specify in the Traffic Shape screen during test creation. The Traffic Shape configuration controls the task count, concurrency (virtual users per task), ramp-up duration, and hold duration for the test execution.

## Scheduling tests

The solution provides three execution timing options for running load tests:

- **Run Now** - Execute the load test immediately after creation

- **Run Once** - Execute the test on a specific date and time in the future
- **Run on a Schedule** - Create recurring tests using cron expressions to define the schedule

When you select **Run Once**, you specify the run time in 24-hour format and the run date when the load test should start running.

When you select **Run on a Schedule**, you can either manually enter a cron expression or select from common cron patterns (such as every hour, daily at a specific time, weekdays, or monthly). The cron expression uses a fine-grained schedule format with fields for minutes, hours, day of month, month, day of week, and year. You must also specify an expiry date, which defines when the scheduled test should stop running. For more information on how scheduling works, refer to the Test scheduling workflow section of this guide.

**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 creates an Amazon CloudWatch dashboard for each test that displays the combined output of all tasks running in the Amazon ECS cluster in real time. The CloudWatch dashboard shows average response time, number of concurrent users, number of successful requests, and number of failed requests. The solution aggregates each metric by the second and updates the dashboard 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 cost, security, supported Regions, quotas, and other considerations you should review before deploying the solution.

## Cost

You are responsible for the cost of the AWS services used while running this solution. The total cost depends on the number of load tests run, the duration of those tests, and the amount of data generated. As of this revision, the estimated 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<br>1,000 on-demand read capacity units | $0.0015 |
| AWS Lambda | 1,000 requests<br>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.

> **Note**
>
> The default task configuration uses 2 vCPUs and 4 GB of memory per task. If your load tests do not require these resources, you can reduce them to lower costs. Conversely, you can increase resources to support higher concurrency per task. For more 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.

## MCP Server additional costs (Optional)

The following table provides a cost breakdown for the MCP Server integration with pricing in the US East (N. Virginia) Region for one month.

| Service component | Dimensions | Cost [USD] |
|---|---|---|
| AgentCore Gateway - Tool Indexing | 10 tools × $0.02 per 100 tools | $0.002 |
| AgentCore Gateway - Search API | 10,000 interactions × $0.025 per 1,000 | $0.25 |
| AgentCore Gateway - API Invocations | 50,000 invocations × $0.005 per 1,000 | $0.25 |

| Service component | Dimensions | Cost [USD] |
| --- | --- | --- |
| AWS Lambda Function | Variable based on usage (typical workloads) | $5.00 - $20.00 |
| **Total estimated additional cost:** | | **$5.50 - $20.50 per month** |

Prices are subject to change. For full details on AgentCore Gateway pricing, refer to Amazon Bedrock Pricing (AgentCore Gateway section). For Lambda pricing, refer to AWS Lambda Pricing.

# 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.

## Amazon API Gateway

This solution deploys edge-optimized Amazon API Gateway endpoints to provide RESTful APIs for the load testing functionality using the default API Gateway endpoint rather than a custom domain. For edge-optimized APIs using the default endpoint, API Gateway uses the TLS-1-0 security policy. For more information, refer to Working with REST APIs in the *Amazon API Gateway Developer Guide*.

This solution uses the default API Gateway 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 domain with a custom SSL certificate instead of the default API Gateway certificate. For more information, refer to Setting up custom domain names for REST APIs.

## 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.

## Amazon VPC

**VPC**: A virtual private cloud (VPC) based on the Amazon VPC service gives you a private, logically isolated network in the AWS Cloud.

You can specify your own VPC in AWS CloudFormation parameters during deployment. The VPC is used exclusively by the ECS tasks that generate load; the web console and API are not deployed within this VPC. If you do not specify an existing VPC, the solution will create a new VPC with the required networking configuration. If you choose to use an existing VPC, it must meet the following requirements to run load testing tasks successfully.

## VPC requirements

The minimum requirements for a VPC to be used with Distributed Load Testing on AWS are listed below.

- The VPC must contain at least two AZs
- The VPC must contain at least two subnets, each in a separate AZ
- VPC subnets can be either public or private, but they must use the same configuration (both public OR both private)
- The VPC must provide access to endpoints for ECR, CloudWatch Logs, S3, and IoT Core.
- The VPC must provide access to the service(s) being targeted by load tests.

> **Note**
>
> If you don't have a VPC that meets these criteria, you can create a VPC with the VPC wizard quickly. For more information, see Create a VPC.

Public subnets may meet these requirements by including the following:

- An internet gateway attached to the VPC
- A route to the internet gateway (0.0.0.0/0)

Private subnets may meet these requirements through the use of NAT Gateways or VPC endpoints, as described below.

**Option 1: NAT Gateway**

- Deploy a NAT Gateway in each AZ with private subnets
- Configure route tables to route internet-bound traffic (0.0.0.0/0) through the NAT Gateway

**Option 2: VPC Endpoints**

Create the following VPC endpoints in your VPC:

- Amazon ECR API Endpoint: `com.amazonaws.<region>.ecr.api`
- Amazon ECR DKR Endpoint: `com.amazonaws.<region>.ecr.dkr`

- Amazon CloudWatch Logs endpoint: `com.amazonaws.<region>.logs`
- Amazon S3 Gateway endpoint: `com.amazonaws.<region>.s3`
- AWS IoT Core endpoint (required if using the live data charts) `com.amazonaws.<region>.iot.data`

Other VPC configurations may also work.

> **Important**
>
> The security group attached to each VPC endpoint interface must allow inbound TCP traffic on port 443 from the ECS task security group.

#### Security Group Configuration

During deployment, the solution will create a security group within your VPC to permit the following traffic with tasks in the ECS cluster:

- All outbound traffic
- Inbound traffic on port 50000 from other tasks in the same security group, to facilitate coordination between worker and leader tasks.

## 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).

### MCP Server security (Optional)

If you deploy the optional MCP Server integration, the solution uses AWS AgentCore Gateway to provide secure access to load testing data for AI agents. AgentCore Gateway validates Amazon Cognito authentication tokens for each request, ensuring that only authorized users can access the MCP Server. The MCP Server Lambda function implements read-only access patterns, preventing AI agents from modifying test configurations or results. All MCP Server interactions use the same permission boundaries and access controls as the web console.

## 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) |

## MCP Server supported AWS Regions (Optional)

If you plan to deploy the optional MCP Server integration, you must deploy the solution in an AWS Region where AWS AgentCore Gateway is available. The MCP Server feature is only available in the following AWS Regions:

| Region name | Region code |
|---|---|
| US East (N. Virginia) | us-east-1 |
| US West (Oregon) | us-west-2 |
| Asia Pacific (Singapore) | ap-southeast-1 |
| Asia Pacific (Sydney) | ap-southeast-2 |
| Asia Pacific (Tokyo) | ap-northeast-1 |
| Europe (Frankfurt) | eu-central-1 |
| Europe (Ireland) | eu-west-1 |
| Europe (London) | eu-west-2 |
| Europe (Paris) | eu-west-3 |

For the most current availability of AWS AgentCore Gateway by Region, refer to AWS AgentCore Gateway endpoints and quotas in the *AWS AgentCore Gateway Developer Guide*.

# 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 Amazon Linux 2023 container image (with Taurus installed) does not limit concurrent connections per task, but that does not mean it can support an unlimited number of users. To determine the number of concurrent users the containers can generate for a test, refer to the 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 creates an Amazon CloudWatch dashboard for each test that displays the combined output of all tasks running in the Amazon ECS cluster in real time. The CloudWatch dashboard shows average response time, number of concurrent users, number of successful requests, and number of failed requests. The solution aggregates each metric by the second and updates the dashboard 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

After deploying the solution, we recommend continuously monitoring the solution's resources using Amazon CloudWatch alarms and metrics.

## Setting up CloudWatch alarms

You can set up CloudWatch alarms to monitor key metrics and receive notifications when thresholds are exceeded. Consider setting up alarms for the following resources:

### Amazon CloudFront distribution metrics

Monitor CloudFront distribution performance and errors. For more information, refer to CloudFront distribution metrics in the *Amazon CloudFront Developer Guide*.

### Amazon API Gateway metrics

Monitor API request rates, latency, and errors. For more information, refer to Amazon API Gateway dimensions and metrics in the *Amazon API Gateway Developer Guide*.

### AWS Lambda function metrics

Monitor Lambda function invocations, duration, errors, and throttles for the solution's microservices.

#### Amazon ECS and AWS Fargate metrics

Monitor task CPU and memory utilization during load tests to ensure adequate resources.

#### Amazon DynamoDB metrics

Monitor read and write capacity consumption, throttled requests, and latency.

# Engage an Expert

## AWS Countdown Premium Short Term Engagements for Distributed Load Testing on AWS

Our AWS engineers provide expert guidance on performance testing fundamentals, script development, and results analysis. Sign up now.

#### Overview

AWS Countdown Premium (CDP) Short Term Engagements provide expert guidance for organizations conducting performance testing at scale. Through a collaborative "do-it-yourself" model, AWS engineers deliver strategic oversight and technical expertise while your team maintains execution responsibility. Expert AWS engineers are available within one week of signup with no long-term contracts required.

#### Service Model

CDP engineers work alongside your team to provide guidance and oversight throughout your performance testing implementation. This hands-off approach ensures you receive expert direction while building internal capabilities. The service is ideal for organizations with existing testing capabilities who need specialized AWS expertise to implement Distributed Load Testing on AWS effectively.

#### What CDP Engineers Provide

CDP engineers guide you through performance testing fundamentals and Distributed Load Testing on AWS architecture. They provide guidance on JMeter, K6, and Locust script structure and test script development, assist with CloudFormation template deployment, and evaluate test results with performance optimization recommendations. Support includes resource utilization analysis, best practices alignment, and end-to-end guidance from initial setup through results analysis, enabling knowledge transfer to your team.

### Customer Responsibilities

Your team handles application-level configurations, test script development, and test scenario verification. You maintain responsibility for actual test execution and operations, including all testing activities before, during, and after performance testing events.

### Key Benefits

CDP Short Term Engagements deliver reduced risk through expert oversight, contextual guidance specific to your workload, performance optimization recommendations, faster issue resolution, best practice alignment, and comprehensive support while maintaining your team's ownership and capability development.

### Supported Architectures

Distributed Load Testing on AWS supports testing for web applications, APIs, microservices, and serverless architectures at scale, leveraging the Distributed Load Testing on AWS solution. The testing capabilities extend far beyond these common use cases to include databases, TCP/UDP protocols, LDAP directories, SMTP mail servers, and many other systems and protocols that require performance validation under load.

### Getting Started

Organizations interested in CDP Short Term Engagements for Distributed Load Testing on AWS can sign up directly through the AWS website here and select "Use Case Implementation" for your Focus Area.

### Out of Scope

CDP does not provide custom test script development (guidance only), manage test execution operations, or create custom hands-on labs or workshops. On-site support is also out of scope.

# Deploy the solution

AWS Launch Wizard is the recommended deployment method for this solution. It provides:

- A guided configuration experience with detailed help panels at each step
- A centralized page to monitor the health of all your deployments
- Indication when there is a more recent version of the solution available for deployment or upgrade

Alternatively, you can deploy the solution directly using an AWS CloudFormation template.

## Deployment process overview

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

**Time to deploy:** Approximately 15 minutes for the main stack, plus 5 minutes for each additional Region

> **Note**
>
> This solution includes data collection 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.

> **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.

## Deploy using AWS Launch Wizard

This solution features a guided deployment process using AWS Launch Wizard. Follow these steps to deploy Distributed Load Testing on AWS into your account.

1. Sign in to the AWS Management Console and select the button below to start the deployment process.


2. If there are more than one deployment patterns available for the solution, select the one that's most applicable to your use case.
3. Select a version to deploy. The latest version is recommended.
4. Click on the **Launch deployment wizard** button.

You will then follow a series of steps to collect the information needed to deploy the solution. It will take approximately 15 minutes to provision the required resources.

Select your deployment from the Deployment list to view its status.

# Deploy using AWS CloudFormation

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.

## 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:


**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

Follow these steps to deploy the Distributed Load Testing on AWS solution into your account. This automated AWS CloudFormation template deploys Distributed Load Testing on AWS.

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

   Launch solution

   Alternatively, you can 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 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 |

| Parameter | Default | Description |
| --- | --- | --- |
| **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. |
| **Auto-Update Container Image** | No | 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. |
| **Deploy Optional MCP Server** | No | Deploy the optional remote MCP server, using AgentCore Gateway to connect AI applications to Distributed Load Testing on AWS. |

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 5 minutes per Region

You can run tests across multiple Regions.

When you deploy the Distributed Load Testing solution, it creates a regional CloudFormation template in the scenarios S3 bucket. The URL for this template is listed in the CloudFormation outputs of your main stack under the key "RegionalCFTemplate".

To run a multi-Region test, you must deploy the regional CloudFormation template in each Region where you want to run the test.

> **Note**
>
> Each AWS account can use only one regional stack per region. Also, the regional stack cannot be used in the same region as the main stack.

You can install the regional template as follows:

1. In the solution's web console, navigate to **Dashboard** in the left 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 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. |

| Parameter | Default | Description |
| --- | --- | --- |
| **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 appear in the web console. When you create a test, all available Regions are listed on the **Dashboard** and in **Scenario Creation**. You can add a Region to a test in the **Traffic Shape** step of Scenario Creation.

The solution creates a DynamoDB item for each deployed Region in the scenarios table, which contains the necessary information about the testing resources in that Region. You can sort test results in the web console by Region. To view aggregate results across all Regions in a multi-Region test, use Amazon CloudWatch metrics. You can find the source code for the graph in the test results after the test completes.

> **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

Updating the solution applies the latest features, security patches, and bug fixes to your deployment. To update to the latest version, refer to the appropriate section based on your original deployment method: AWS Launch Wizard or AWS CloudFormation.

> **Important**
>
> Before updating, ensure no load tests are currently running. The update process may temporarily disrupt the solution's availability.

## Update using AWS Launch Wizard

The console automatically displays the newest available version of the solution in the **Deployment version** dropdown. If you have previously deployed the solution, follow this procedure to update your deployment to the latest version.

1. Go to Launch Wizard Deployments.
2. Select the deployment you want to update.
3. Choose **Actions**, then **Update deployment version**.
4. Select the latest version from the available **Deployment versions**.
5. Review configuration.
6. Make changes needed on each step.
7. Confirm the update.

## Update using AWS CloudFormation

If you have previously deployed the solution, follow this procedure to update the CloudFormation stack to the latest version.

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.

> **Note**
>
> If you experience Amazon Cognito authentication issues while logging in from your browser after stack upgrade, please refresh your browser (Ctrl+Shift+R on Windows/Linux or Cmd +Shift+R on Mac) to clear cached data and try again.

# Troubleshooting updates from versions prior to v3.3.0

> **Note**
>
> This section applies only to updates from versions prior to v3.3.0. If you are updating from v3.3.0 or later, follow the standard update procedure via either AWS Launch Wizard or AWS CloudFormation.

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 by following the steps in Update using AWS CloudFormation.
6. This update removes app registry-related resources from the stack, allowing the update to complete successfully.
7. Perform another stack update using the latest template URL.

## Updating regional stacks

If you have deployed the solution in multiple Regions, you must update each regional stack separately. Follow the standard update procedure for each regional CloudFormation stack in the Regions where you have deployed testing infrastructure.

## AWS Systems Manager Application Manager

After updating the solution, AWS Systems Manager Application Manager provides an application-level view into the solution and its resources. You can use Application Manager to:

- Monitor resources, costs for deployed resources across stacks and AWS accounts, and logs from a central location.

- View operations data for the solution's resources 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 Business Support+, AWS Enterprise Support, or Unified Operations, you can use AWS 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, including the name of this product and the version you are using, such as this example: Distributed Load Testing on AWS vX.Y.Z.
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

### AWS CloudFormation

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack.
3. Choose **Delete**.

### AWS Launch Wizard

1. Sign in to the AWS Launch Wizard console.
2. On the Launch Wizard Deployments page, select this solution's deployment.
3. Choose **Actions**, then **Delete**.
4. Confirm the deletion.

## 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 provides a comprehensive guide to using the Distributed Load Testing on AWS solution, from creating your first test scenario to analyzing detailed results. The workflow includes creating a test scenario, running a test, and exploring test results.

# Create a test scenario

Creating a test scenario involves four main steps: configuring general settings, defining the scenario, shaping traffic patterns, and reviewing your configuration.

## Step 1: General settings

Configure the basic parameters for your load test including test name, description, and general configuration options.

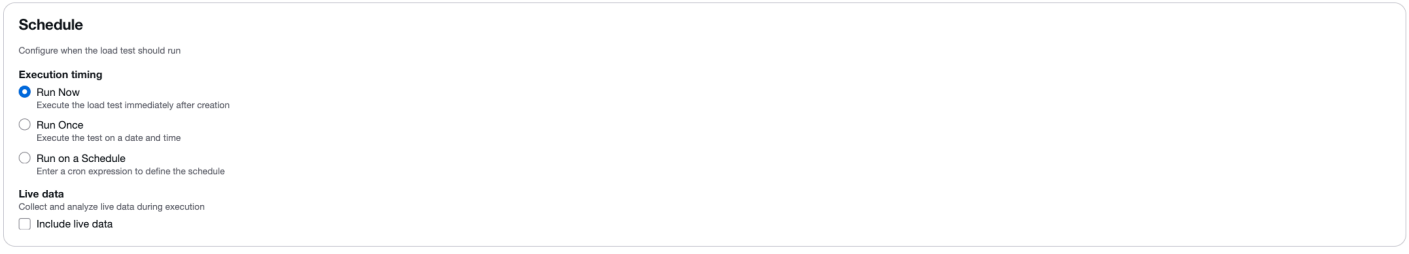
### Test identification

- **Test name** (Required) - A descriptive name for your test scenario
- **Test description** (Required) - Additional details about the test purpose and configuration
- **Tags** (Optional) - Add up to 5 tags to categorize and organize your test scenarios

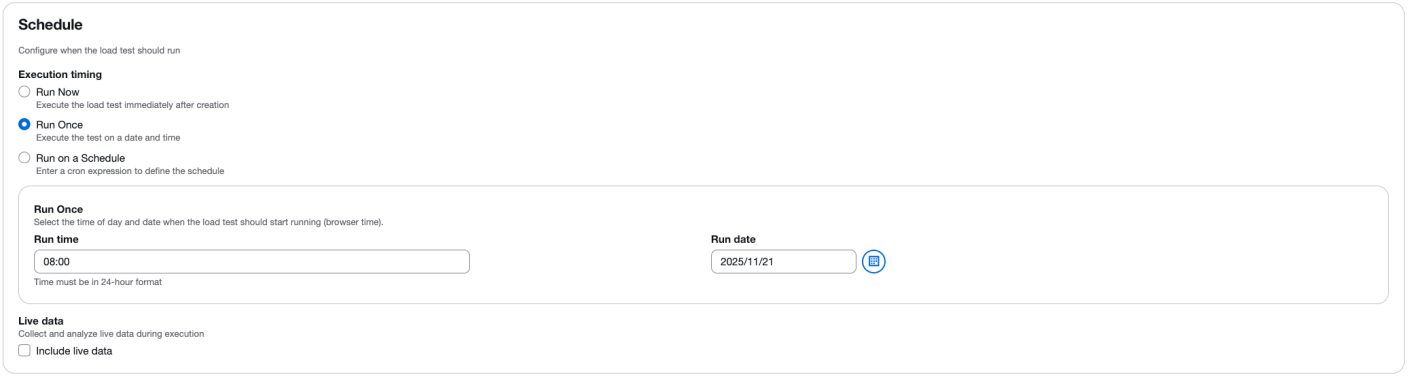
### Scheduling options

Configure when the test should run:

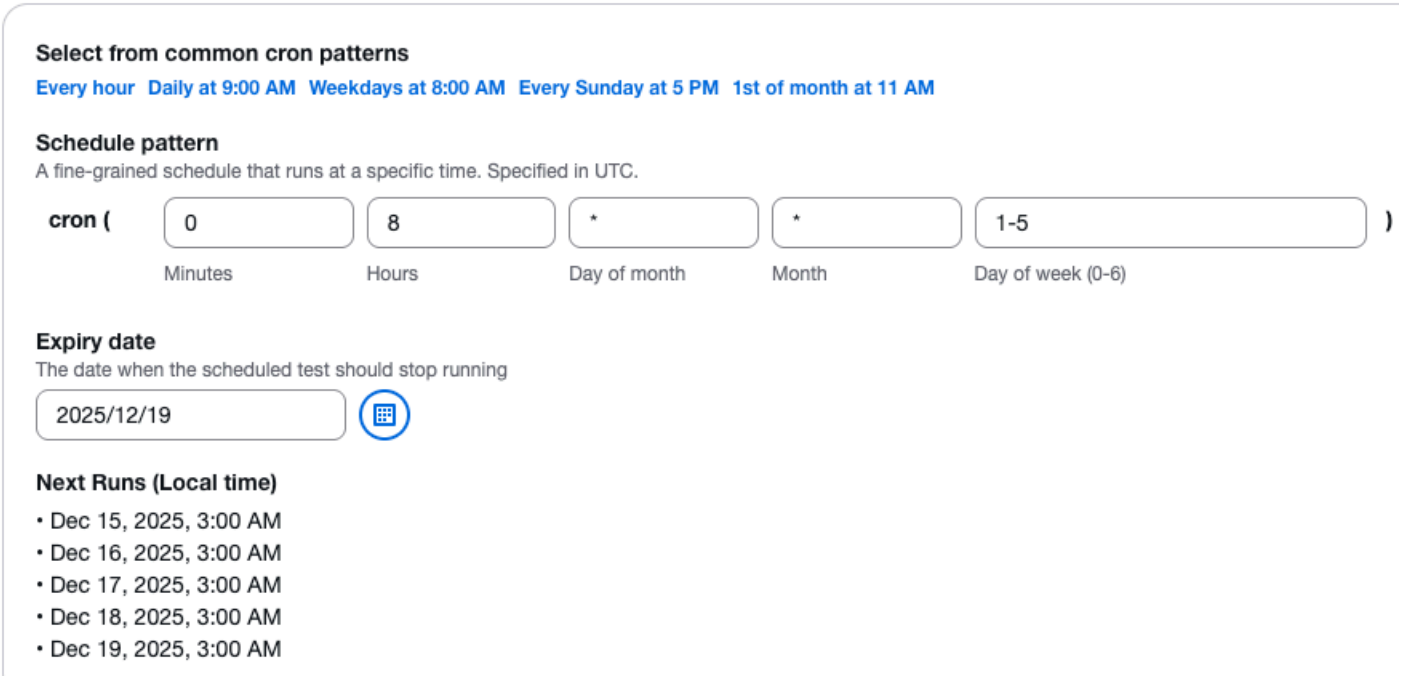
- **Run Now** - Execute the test immediately after creation.




- **Run Once** - Schedule the test to run at a specific date and time.




- **Run on a Schedule** - Use cron-based scheduling to run tests automatically at regular intervals. You can select from common patterns (every hour, daily, weekly) or define a custom cron expression.




## Scheduling workflow

When you schedule a test, the following workflow occurs:

- The schedule parameters are sent to the solution's API via Amazon API Gateway.
- The API passes the parameters to a Lambda function that creates a CloudWatch Events rule scheduled to run on the specified date.
- For one-time tests (Run Once), the CloudWatch Events rule runs on the specified date and the `api-services` Lambda function executes the test.

- For recurring tests (Run on a Schedule), the CloudWatch Events rule activates on the specified date, and the `api-services` Lambda function creates a new rule that runs immediately and recurrently based on the specified frequency.

#### Live data

Select the **Include live data** checkbox to view real-time metrics while your test is running. When enabled, you can monitor:

- Average response time.
- Virtual user counts.
- Successful request counts.
- Failed request counts.

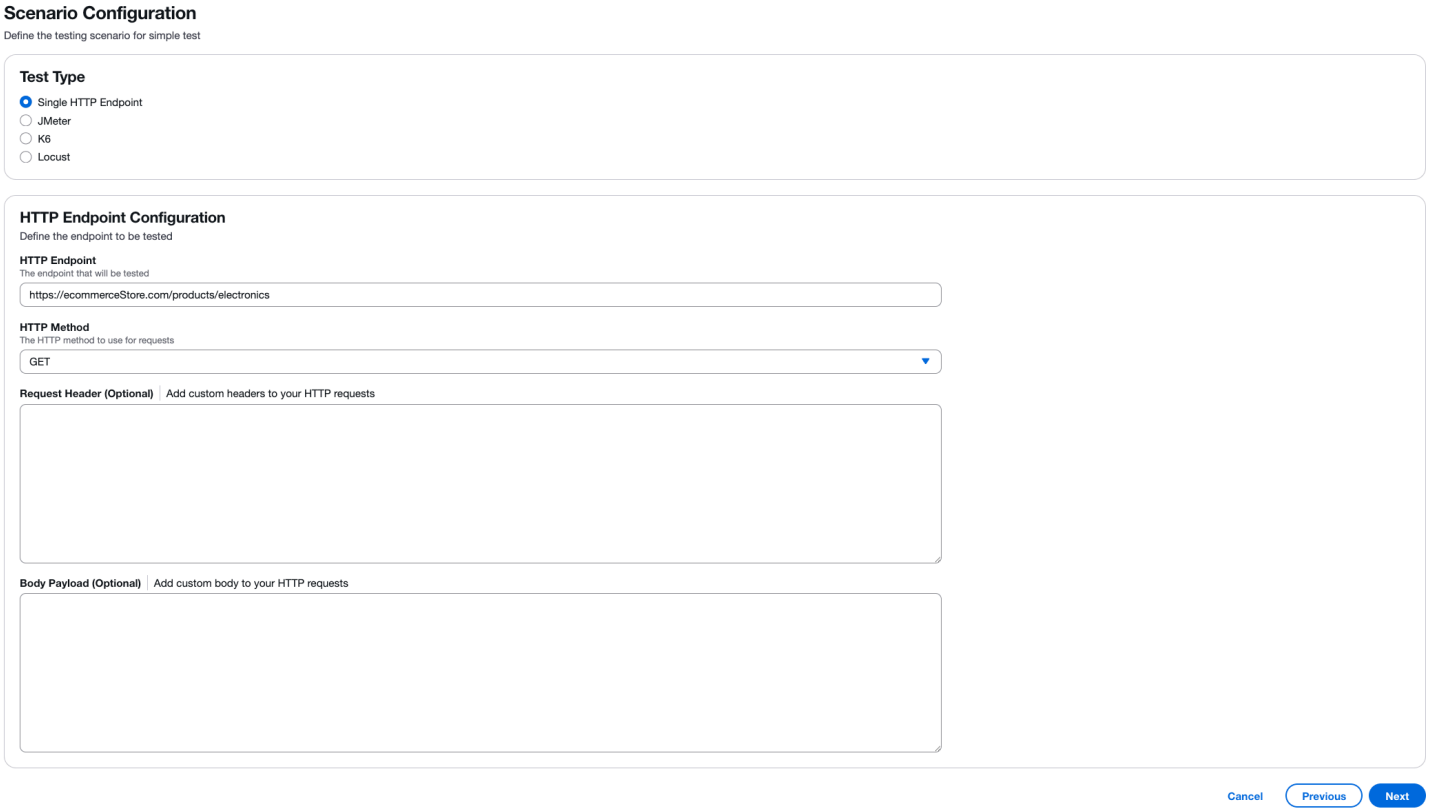
The live data feature provides real-time charting with data aggregated at one-second intervals. For more information, refer to Monitoring with live data.

## Step 2: Scenario configuration

Define the specific testing scenario and select your preferred testing framework.

#### Test type selection

Choose the type of load test you want to perform:




- **Single HTTP Endpoint** - Test a single API endpoint or web page with simple configuration.
- **JMeter** - Upload JMeter test scripts (.jmx files or .zip archives).
- **K6** - Upload K6 test scripts (.js files or .zip archives).
- **Locust** - Upload Locust test scripts (.py files or .zip archives).

**HTTP endpoint configuration** image::images/test-types.png[Select the test type to run] When "Single HTTP Endpoint" is selected, configure these settings:

**HTTP Endpoint** (Required)

Enter the full URL of the endpoint you want to test. For example, `https://api.example.com/users`. Ensure the endpoint is accessible from AWS infrastructure.

**HTTP Method** (Required)

Select the HTTP method for your requests. Default is `GET`. Other options include `POST`, `PUT`, `DELETE`, `PATCH`, `HEAD`, and `OPTIONS`.

**Request Header** (Optional)

Add custom HTTP headers to your requests. Common examples include:

- `Content-Type: application/json`
- `Authorization: Bearer <token>`
- `User-Agent: LoadTest/1.0`

Choose **Add Header** to include multiple headers.

**Body Payload** (Optional)

Add request body content for POST or PUT requests. Supports JSON, XML, or plain text formats. For example: `{"userId": 123, "action": "test"}`.

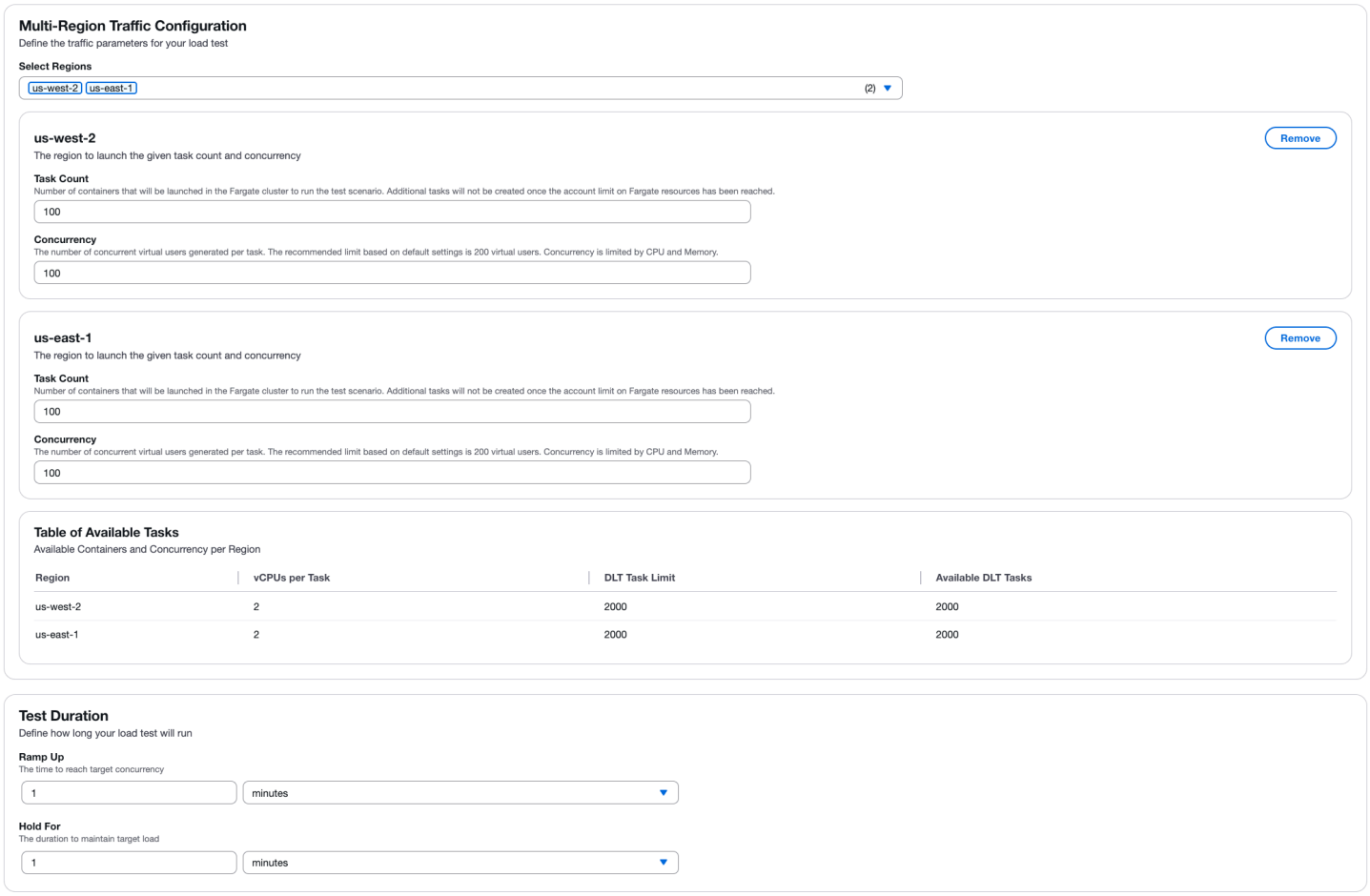
#### Test framework scripts

When using JMeter, K6, or Locust, upload your test script file or a .zip archive containing your test script and supporting files. For JMeter, you can include custom plugins in a `/plugins` folder within your .zip archive.

> **Important**
>
> Although your test script (JMeter, K6, or Locust) may define concurrency (virtual users), transaction rates (TPS), ramp-up times, and other load parameters, the solution will override these configurations with the values you specify in the Traffic Shape screen during test creation. The Traffic Shape configuration controls the task count, concurrency (virtual users per task), ramp-up duration, and hold duration for the test execution.

## Step 3: Traffic shape

Configure how traffic will be distributed during your test, including multi-region support.




## Multi-region traffic configuration

Select one or more AWS regions to distribute your load test geographically. For each selected region, configure:

## Task Count

The number of containers (tasks) that will be launched in the Fargate cluster for the test scenario. Additional tasks will not be created once the account has reached the "Fargate resource has been reached" limit.

## Concurrency

The number of concurrent virtual users generated per task. The recommended limit is based on default settings of 2 vCPUs per task. Concurrency is limited by CPU and Memory resources.

## 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).

### Calibration process

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, you can increase the container resources to allow for a higher number of concurrent users. However, this results in a higher cost. For details, refer to the Developer 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.

> **Note**
>
> The solution displays available capacity information for each region, helping you plan your test configuration within available limits.

### Table of available tasks

The **Table of Available Tasks** displays resource availability for each selected region:

- **Region** - The AWS region name.
- **vCPUs per Task** - The number of virtual CPUs allocated to each task (default: 2).
- **DLT Task Limit** - The maximum number of tasks that can be created based on your account's Fargate limits (default: 2000).
- **Available DLT Tasks** - The current number of tasks available for use in the region (default: 2000).

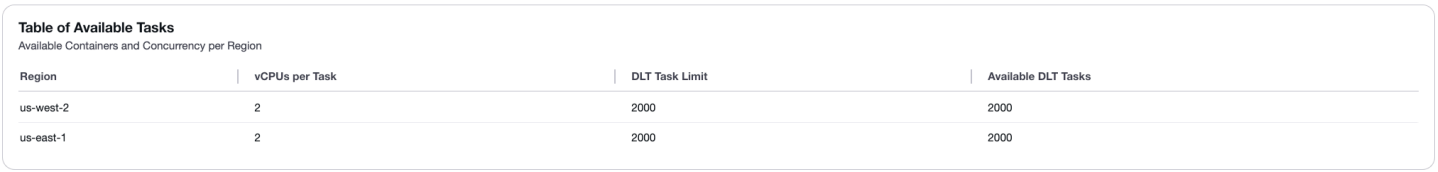


**Table of Available Tasks**

Available Containers and Concurrency per Region

| Region | vCPUs per Task | DLT Task Limit | Available DLT Tasks |
|---|---|---|---|
| us-west-2 | 2 | 2000 | 2000 |
| us-east-1 | 2 | 2000 | 2000 |

To increase the number of available tasks or vCPUs per task, refer to the Developer Guide.

### Test duration

Define how long your load test will run:

#### Ramp Up

The time to reach target concurrency. The load gradually increases from 0 to the configured concurrency level over this period.

#### Hold For

The duration to maintain target load. The test continues at full concurrency for this period.

## Step 4: Review and create

Review all your configurations before creating the test scenario. Verify:

- General settings (name, description, schedule).
- Scenario configuration (test type, endpoint or script).
- Traffic shape (tasks, users, duration, regions).

After reviewing, choose **Create** to save your test scenario.

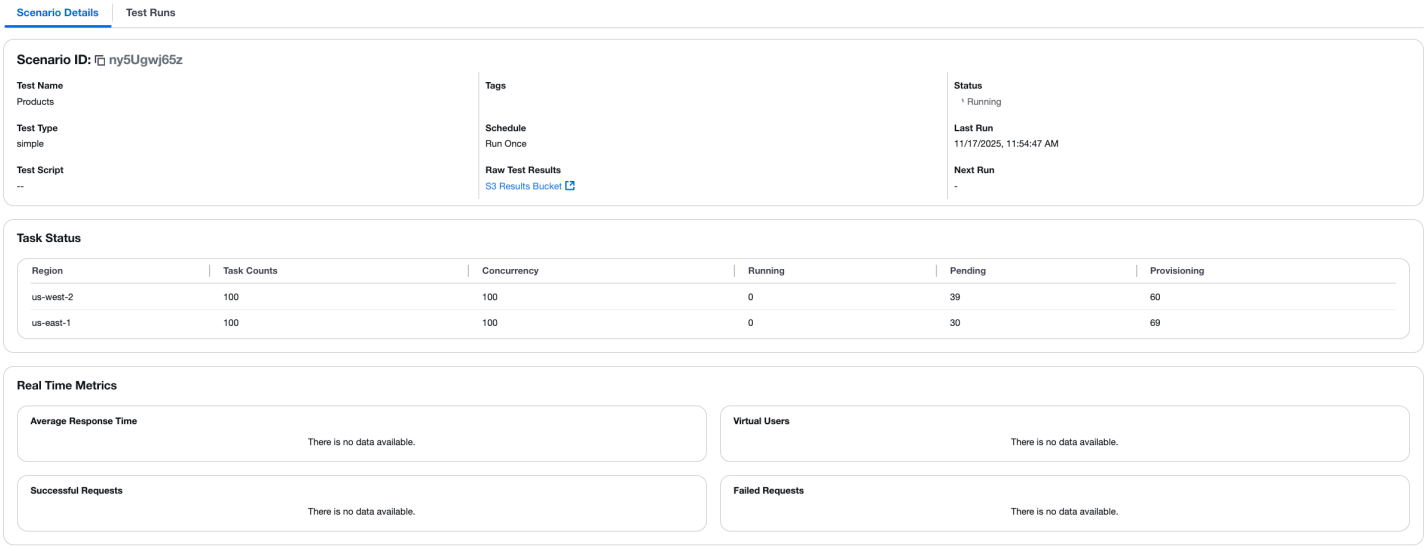
### Managing test scenarios

After creating a test scenario, you can:

- **Edit** - Modify the test configuration. Common use cases include:
  - Refining traffic shape to achieve the desired transaction rate.
- **Copy** - Duplicate an existing test scenario to create variations. Common use cases include:
  - Updating endpoints or adding headers/body parameters.
  - Adding or modifying test scripts.
- **Delete** - Remove test scenarios you no longer need.

## Run a test scenario

After creating a test scenario, you can run it immediately or schedule it to run at a specific time in the future. When you navigate to a running test, the console displays the Scenario Details tab with real-time task status and metrics.



## Scenario details view

The Scenario Details tab displays key information about your test. The task staus table real-time information for every region.

### Task status table

The Task Status table shows real-time information for each region:

- **Region** - The AWS region where tasks are running
- **Task Counts** - The total number of tasks configured for the region
- **Concurrency** - The number of virtual users per task
- **Running** - Number of tasks currently executing the test
- **Pending** - Number of tasks waiting to start
- **Provisioning** - Number of tasks being provisioned

## Test execution workflow

When a test starts, the following workflow occurs:

1. **Task provisioning** - The solution provisions containers (tasks) in the specified AWS regions. Tasks appear in the "Provisioning" column.
2. **Task startup** - The solution continues to provision tasks until the target task count is reached in each region. Tasks move from "Provisioning" to "Pending" to "Running".
3. **Traffic generation** - After the solution provisions all tasks in a region, they begin sending traffic to your target endpoint.
4. **Test execution** - The test runs for the configured duration (ramp-up + hold time).
5. **Results parsing** - When the test ends, a background parsing job aggregates and processes results from all regions.

## Test run statuses

Test runs can have the following statuses:

- **Scheduled** - The test is scheduled to run in the future.
- **Running** - The test is currently in progress.

- **Cancelled** - A user cancelled an in-progress test run.
- **Errored** - The test run encountered an error.
- **Complete** - The test run completed successfully and results are ready.

## Monitoring with live data

If you enabled live data when creating the test scenario, you can view real-time metrics while the test is running. The Real Time Metrics section displays four graphs that update continuously as the test progresses, with data aggregated at one-second intervals.


### Graph descriptions

#### Average Response Time

Displays the average response time in seconds for requests processed by each region. The Y-axis shows response time in seconds, and the X-axis shows the time of day. Each region is represented by a different color in the legend.

#### Virtual Users

Shows the number of concurrent virtual users actively generating load in each region. The graph displays how virtual users ramp up during the test and maintains the target concurrency level.

**Successful Requests**

Displays the cumulative count of successful requests over time for each region. The graph shows the rate at which successful requests are being processed.

**Failed Requests**

Shows the cumulative count of failed requests over time for each region. A low or zero count indicates healthy test execution.

**Multi-region visualization**

When running tests across multiple regions, each graph displays data for all regions simultaneously. The legend at the bottom of each graph identifies which color represents each region (for example, us-west-2 and us-east-1).

**Technical implementation**

The CloudWatch log group for the Fargate tasks contains a subscription filter that captures test results. When the pattern is detected, a Lambda function structures the data and publishes it to an AWS IoT Core topic. The web console subscribes to this topic and displays the metrics in real-time.

> **Note**
>
> Live data is ephemeral and only available while the test is running. 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. Once a test is complete, the solution stores the results data in DynamoDB and Amazon S3. If no data is available yet, the graphs display "There is no data available."

## Cancelling a test

You can cancel a running test from the web console. When you cancel a test, the following workflow occurs:

1. The cancellation request is sent to the `microservices` API
2. The `microservices` API calls the `task-canceler` Lambda function which stops all currently launched tasks

3. If the `task-runner` Lambda function continues to run after the initial cancellation call, tasks may continue to launch briefly
4. Once the `task-runner` Lambda function finishes, AWS Step Functions proceeds to the `Cancel Test` step, which runs the `task-canceler` Lambda function again to stop any remaining tasks

> **Note**
>
> Cancelled tests take time to complete the shutdown process as the solution terminates all containers. The test status will change to "Cancelled" once all resources are cleaned up.

# Explore test results

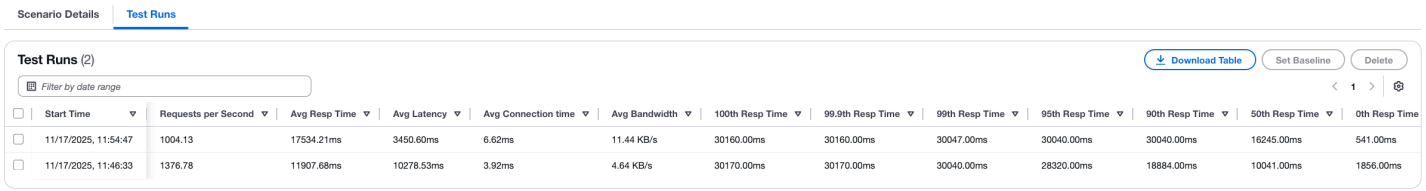
Once the parsing job completes, test results are available for analysis. The solution provides comprehensive metrics and tools to help you understand your application's performance under load.

## Test run summary metrics

When a test completes, the solution generates a summary that includes the following metrics:

- **Average response time** - The average response time, in seconds, for all requests generated by the test.
- **Average latency** - The average latency, in seconds, for all requests generated by the test.
- **Average connection time** - The average time, in seconds, it takes to connect to the host for all requests.
- **Average bandwidth** - The average bandwidth for all 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 requests generated by the test.
- **Percentiles** - Response time percentiles including p50 (median), p90, p95, and p99, showing the distribution of response times across all requests.

# Test runs table




The test runs table displays all historical test runs for a scenario. You can:

- View summary metrics for each test run.
- Set a baseline test run for performance comparison.
- Download the table as a CSV file.
- Toggle columns to customize your view.
- Select a test run to view detailed results.

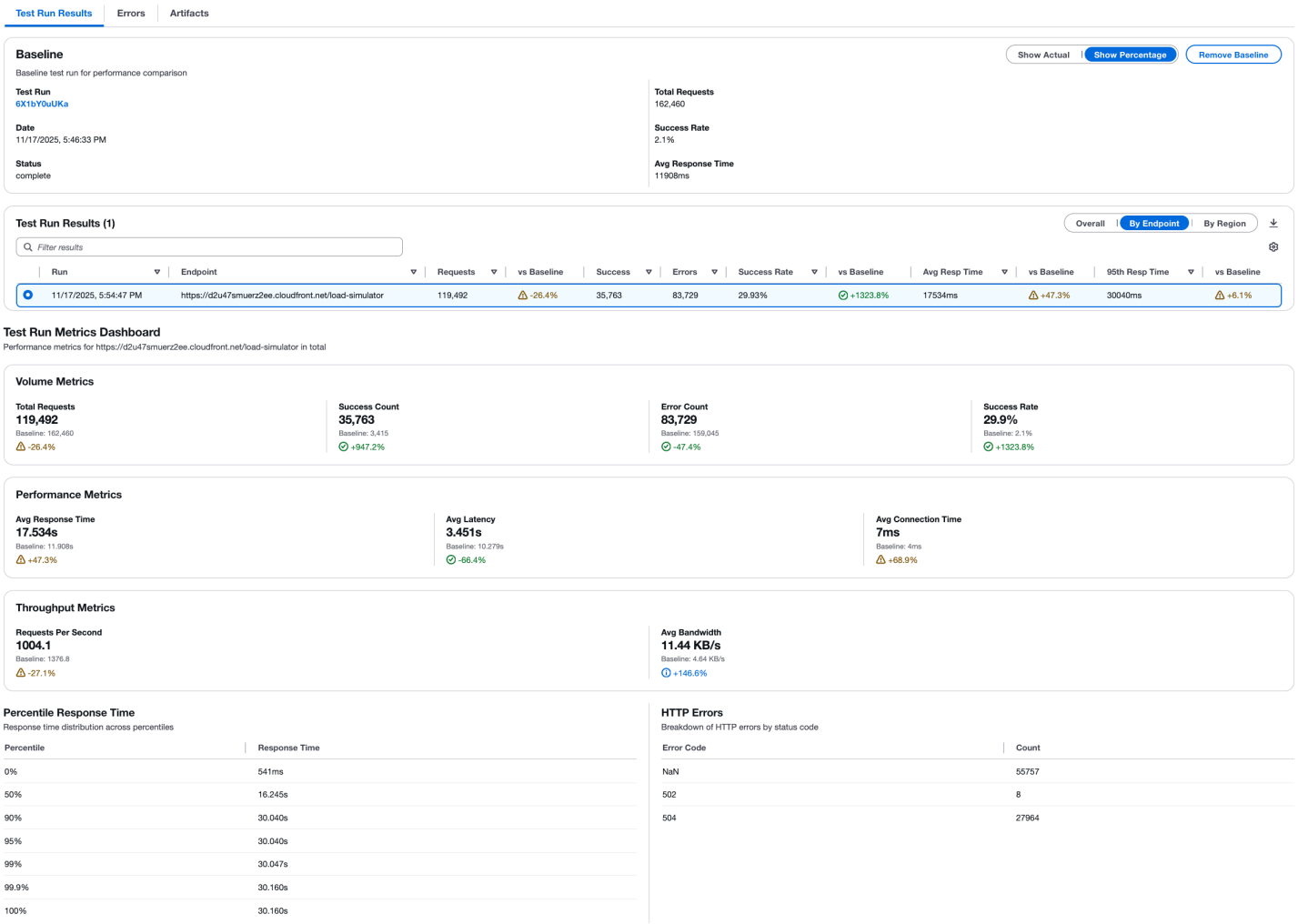
## Baseline comparison

You can designate a test run as a baseline to compare future test runs against it. When a baseline is set:

- The test runs table shows percentage differences (+/-%) compared to the baseline for each metric.
- The baseline indicator helps you quickly identify performance improvements or regressions.
- You can change or clear the baseline at any time.

## Detailed test results

Selecting a test run opens the detailed results view with three tabs: Test Run Results, Errors, and Artifacts.




## Baseline information

If a baseline test run is set, it displays at the top of the page. You can choose **Show Actual**, **Show Percentage**, or **Remove Baseline** to control how baseline comparisons are displayed.

## Test Run Results table

The results table provides detailed metrics with the following features:

## Dimension views

Toggle between three views using the dimension buttons:

- **Overall** - Aggregated results across all endpoints and regions
- **By Endpoint** - Results broken down by individual endpoints
- **By Region** - Results broken down by AWS region

**Action buttons**

- **Show Actual** - Display actual metric values
- **Show Percentage** - Display percentage differences from baseline
- **Remove Baseline** - Clear the baseline comparison

**Data export and customization**

- Download the results table as a CSV file
- Toggle columns to customize your view
- Filter and sort data to focus on specific metrics
- Filter and sort data to focus on specific metrics.

## Errors tab

The errors tab provides detailed error analysis:

- View error counts by type.
- See errors aggregated by overall test or by endpoint.
- Identify patterns in failed requests.
- Troubleshoot issues with specific endpoints or regions.

## Artifacts tab

The artifacts tab allows you to access all files generated during the test run:

- View individual artifacts (logs, results files).
- Download specific artifacts for offline analysis.
- Download all test run artifacts as a single archive.

## S3 results structure

In version 4.0, the S3 results structure has changed to improve organization:

- **New structure** - `scenario-id/test-run-id/results-files`.

- **Legacy structure** - Tests run before version 4.0 show all result files at the scenario ID level.

> **Note**
>
> Test results are displayed in the console. You can also access the raw test results directly in the Amazon S3 bucket under the `Results` folder. For more information on Taurus test results, see Generating Test Reports in the *Taurus User Manual*.

# MCP server integration

If you deployed the optional MCP server component during solution deployment, you can integrate the Distributed Load Testing solution with AI development tools that support the Model Context Protocol. The MCP server provides programmatic access to retrieve, manage, and analyze load tests through AI assistants.

Customers can connect to the DLT MCP Server using the client of their choice (Amazon Q, Claude, etc.), which each have slightly different configuration instructions. This section provides setup instructions for MCP Inspector, Amazon Q CLI, Cline, and Amazon Q Suite.

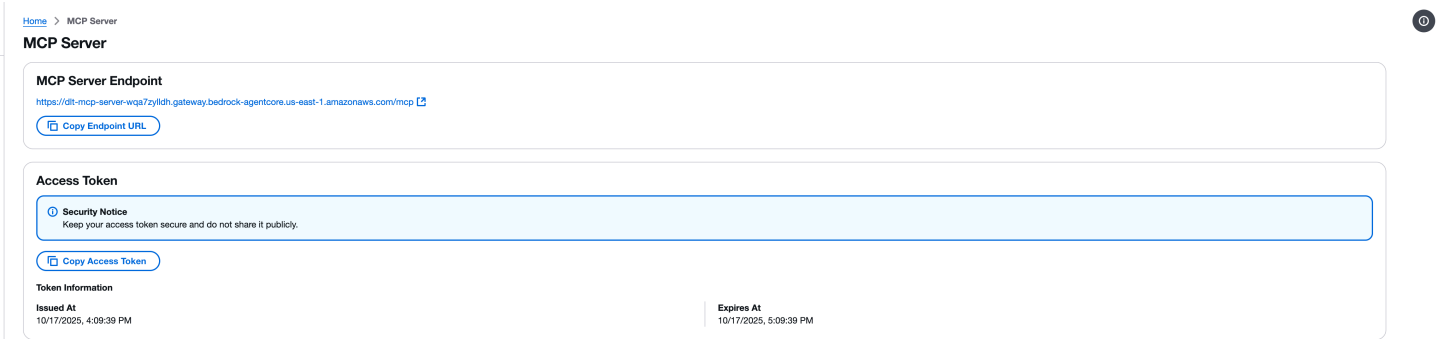
## Step 1: Get MCP endpoint and access token

Before configuring any MCP client, you need to retrieve your MCP server endpoint and access token from the DLT web console.

1. Navigate to the **MCP Server** page in the Distributed Load Testing web console.
2. Locate the **MCP Server Endpoint** section.
3. Copy the endpoint URL using the **Copy Endpoint URL** button. The endpoint URL follows the format: `https://{gateway-id}.gateway.bedrock-agentcore.{region}.amazonaws.com/mcp`
4. Locate the **Access Token** section.
5. Copy the access token using the **Copy Access Token** button.

**Important**

Keep your access token secure and do not share it publicly. The token provides read-only access to your Distributed Load Testing solution through the MCP interface.




## Step 2: Test with MCP Inspector

The Model Context Protocol offers MCP Inspector, a tool to directly connect to MCP servers and invoke tools. This provides a convenient UI and sample network requests for testing your MCP server connection before configuring AI clients.

**Note**

MCP Inspector requires version 0.17 or later. All requests can also be made with JSON RPC directly, but MCP Inspector provides a more user-friendly interface.

### Install and launch MCP Inspector

1. Install npm if necessary.
2. Run the following command to launch MCP Inspector:

```
npx @modelcontextprotocol/inspector
```

### Configure the connection

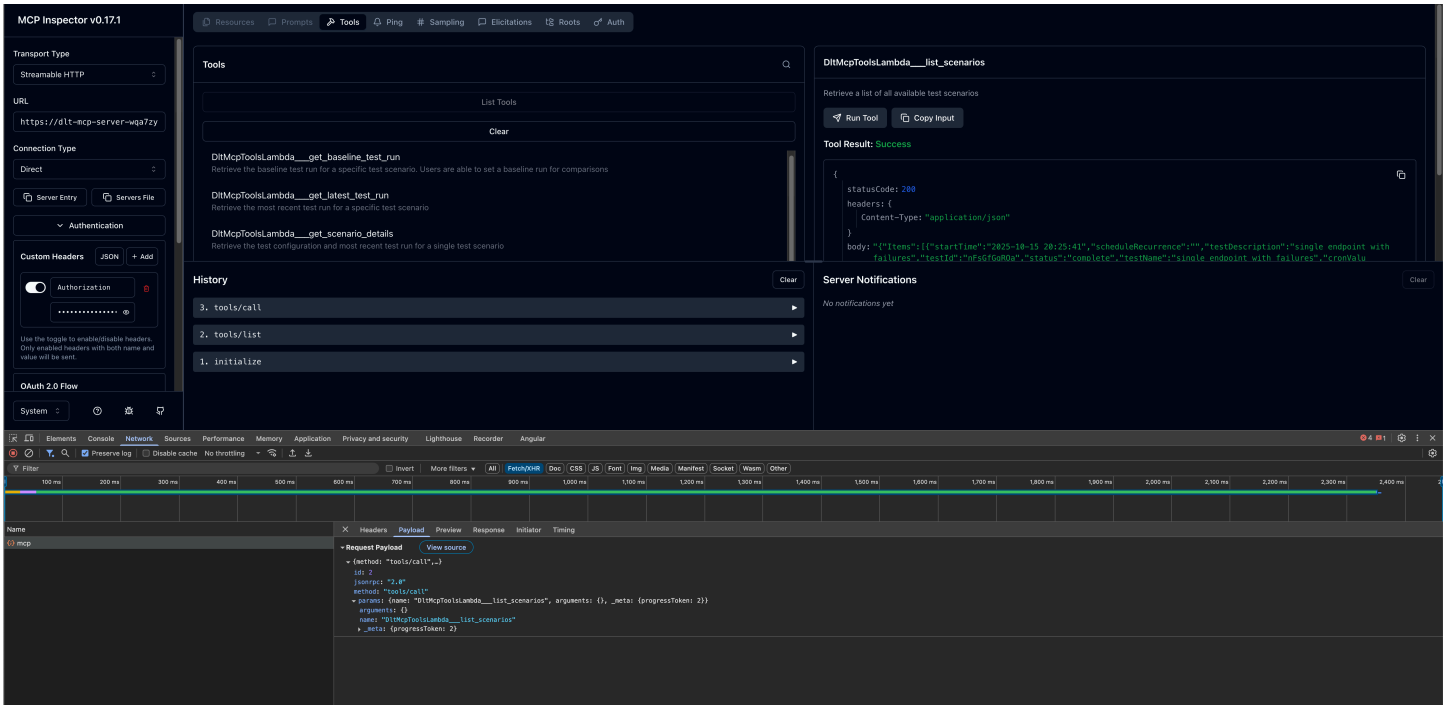
1. In the MCP Inspector interface, enter your MCP Server Endpoint URL.
2. Add an Authorization header with your access token.

3. Click **Connect** to establish the connection.


## Invoke tools

Once connected, you can test the available MCP tools:

1. Browse the list of available tools in the left panel.
2. Select a tool (for example, `list_scenarios`).
3. Provide any required parameters.
4. Click **Invoke** to execute the tool and view the response.




# Step 3: Configure AI development clients

After verifying your MCP server connection with MCP Inspector, you can configure your preferred AI development client.

## Amazon Q CLI

Amazon Q CLI provides command-line access to AI-assisted development with MCP server integration.

### Configuration steps

1. Edit the `mcp.json` configuration file. For more information on configuration file location, refer to Configuring remote MCP servers in the *Amazon Q Developer User Guide*.
2. Add your DLT MCP server configuration:

```
{
  "mcpServers": {
    "dlt-mcp": {
      "type": "http",
      "url": "https://[api-id].execute-api.[region].amazonaws.com/[stage]/gateway/
backend-agent/sse/mcp",
      "headers": {
```

```
        "Authorization": "your_access_token_here"
      }
    }
  }
}
```

### Verify the configuration

1. In a terminal, type q to launch Amazon Q CLI.
2. Type `/mcp` to see all available MCP servers.
3. Type `/tools` to see available tools provided by `dlt-mcp` and other configured MCP servers.
4. Verify that `dlt-mcp` successfully initializes.

## Cline

Cline is an AI coding assistant that supports MCP server integration.

### Configuration steps

1. In Cline, navigate to **Manage MCP Servers** > **Configure** > **Configure MCP Servers**.
2. Update the `cline_mcp_settings.json` file:

```
{
  "mcpServers": {
    "dlt-mcp": {
      "type": "streamableHttp",
      "url": "https://[api-id].execute-api.[region].amazonaws.com/[stage]/gateway/
backend-agent/sse/mcp",
      "headers": {
        "Authorization": "your_access_token_here"
      }
    }
  }
}
```

3. Save the configuration file.
4. Restart Cline to apply the changes.

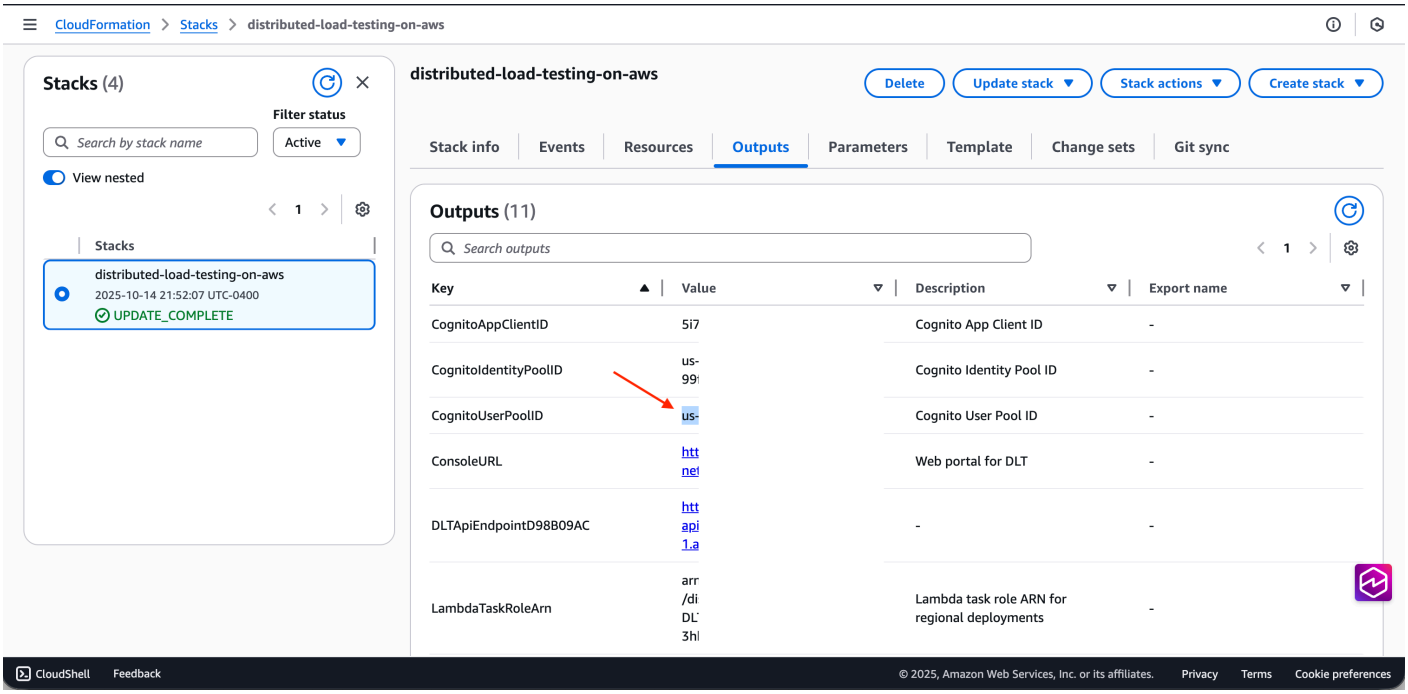
## Amazon Q Suite

Amazon Q Suite provides a comprehensive AI assistant platform with support for MCP server actions.

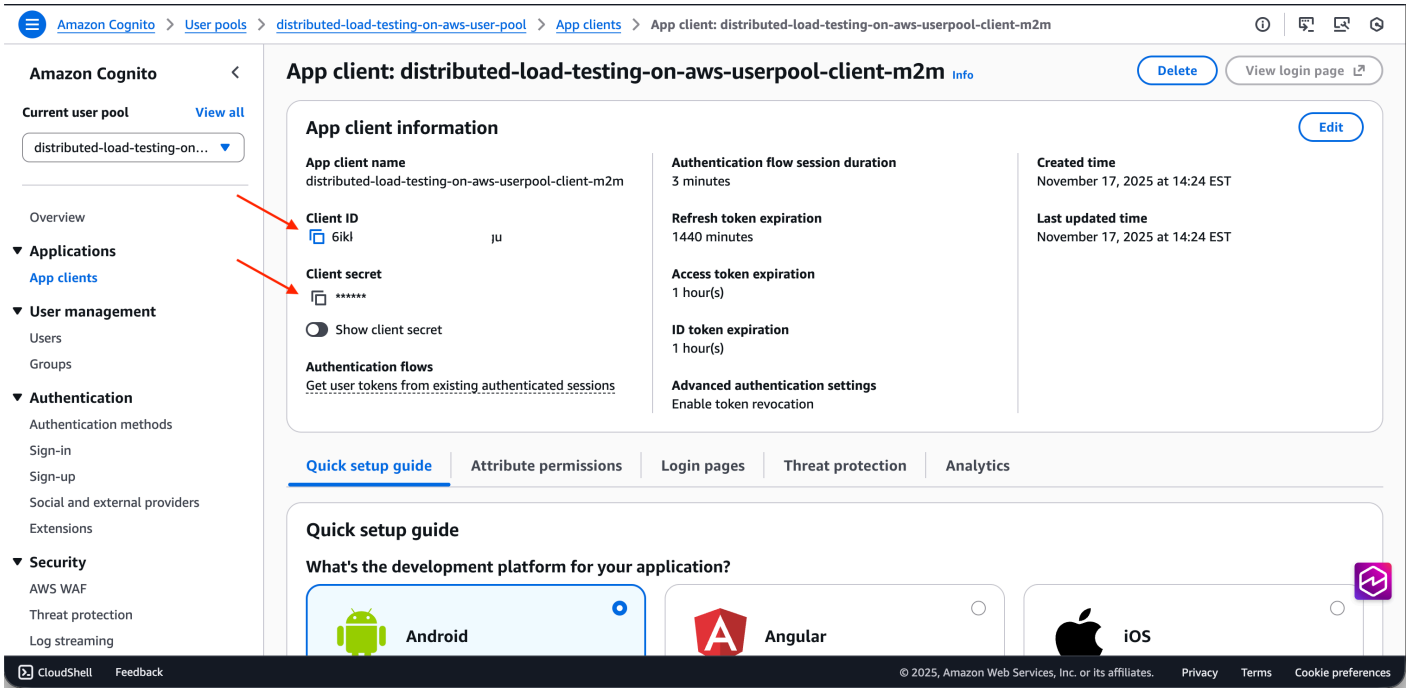
### Prerequisites

Before configuring the MCP server in Amazon Q Suite, you need to retrieve OAuth credentials from your DLT deployment's Cognito user pool:

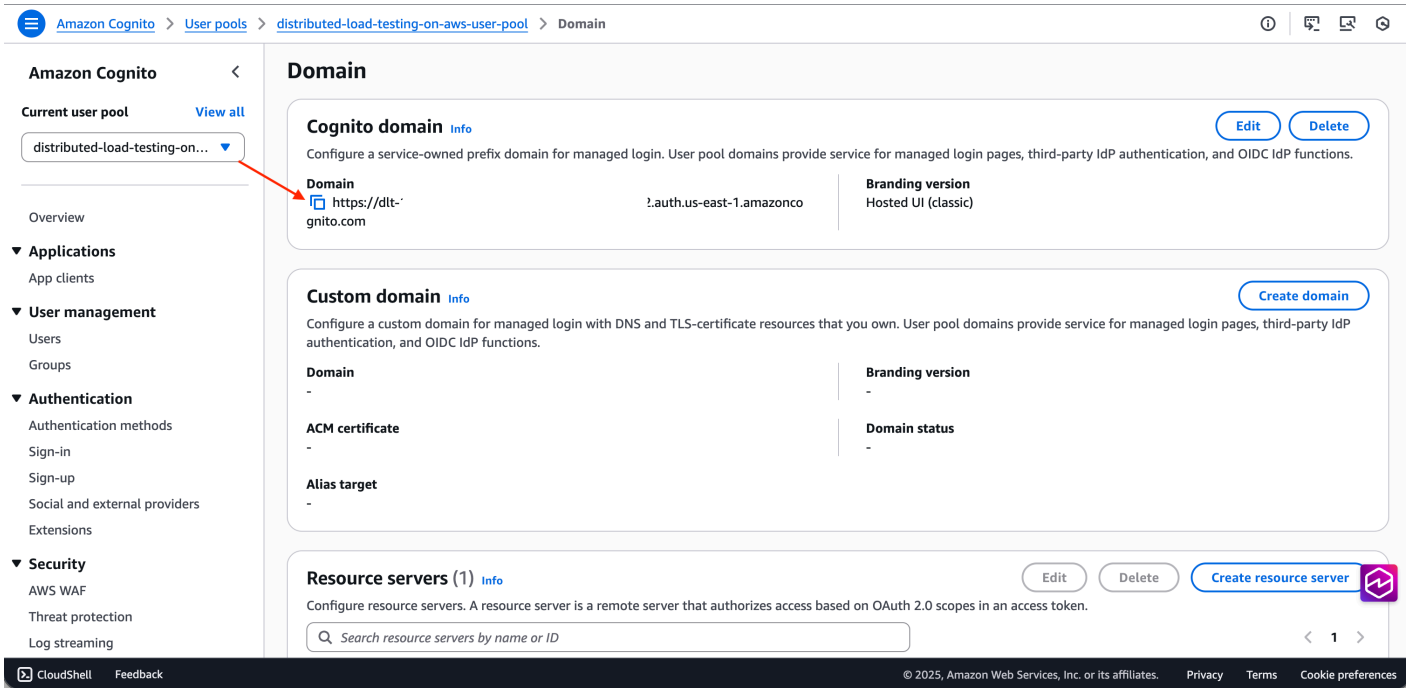
1. Navigate to the AWS CloudFormation console.
2. Select the Distributed Load Testing stack.
3. In the **Outputs** tab, locate and copy the **Cognito User Pool ID** associated with your DLT deployment.




4. Navigate to the Amazon Cognito console.
5. Select the user pool using the User Pool ID from the CloudFormation outputs.
6. In the left navigation, select **App integration** > **App clients**.



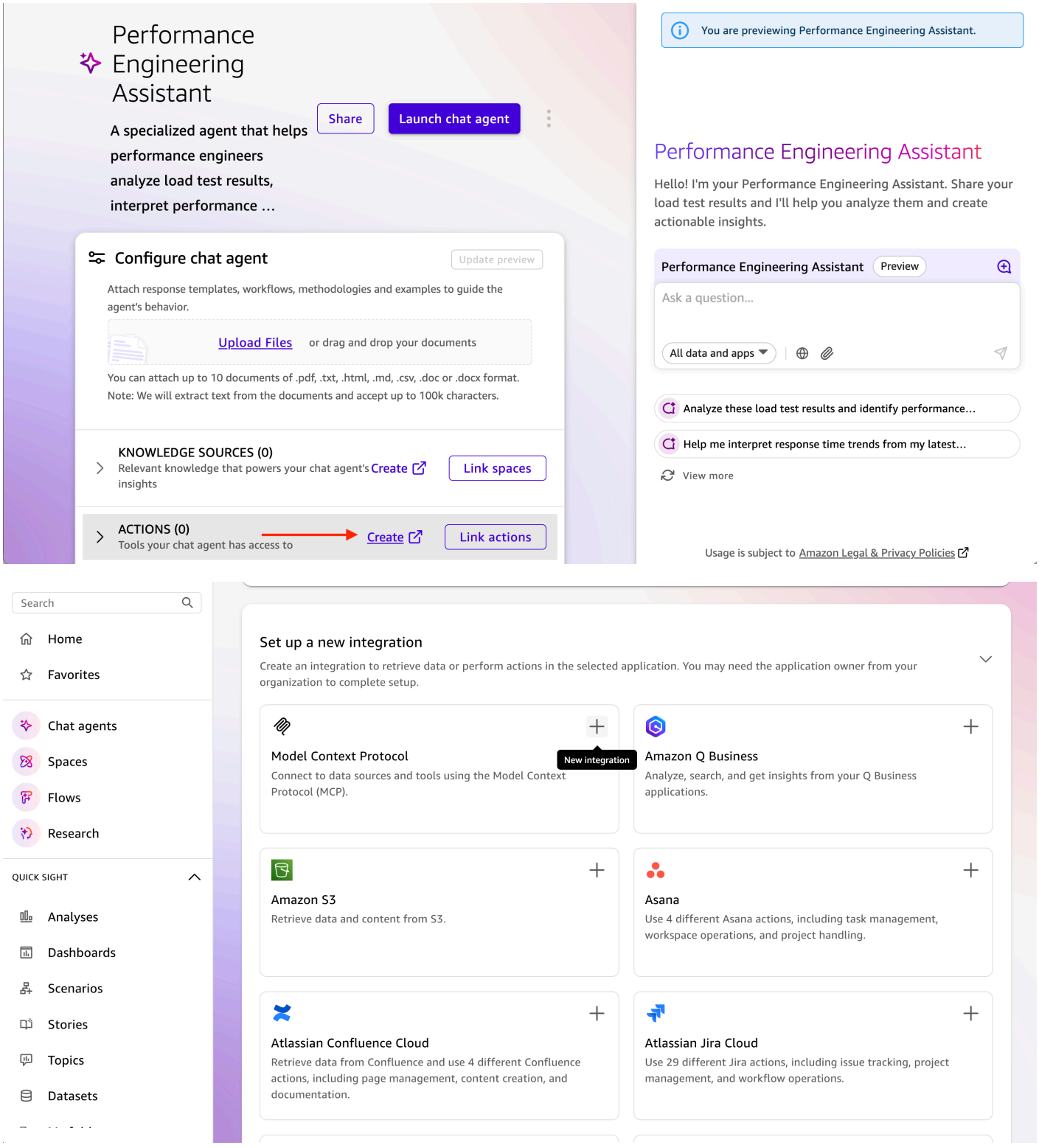

7. Locate the app client with the name ending in `m2m` (machine-to-machine).
8. Copy the **Client ID** and **Client secret**.
9. Get the user pool domain from the **Domain** tab.




10. Construct the token endpoint URL by appending `/oauth2/token` to the end of the domain.

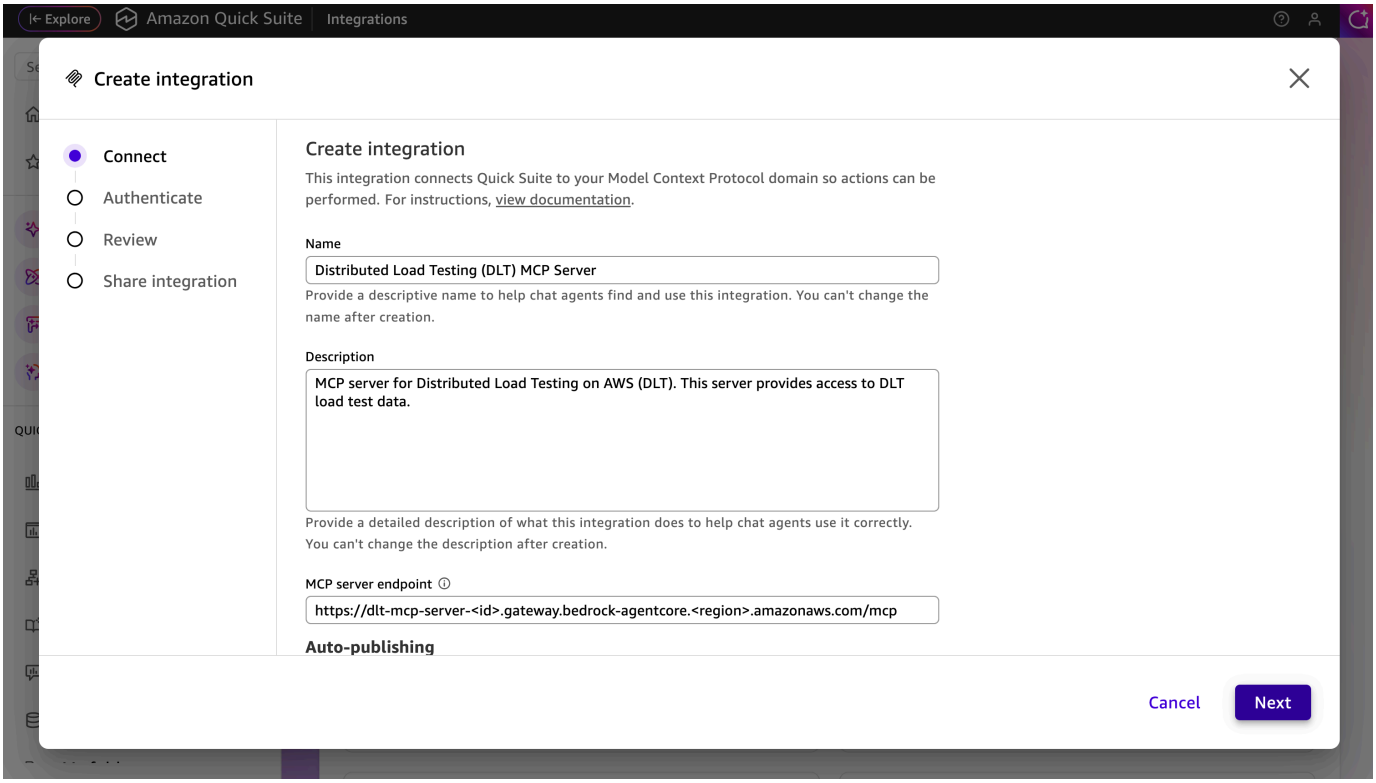
## Configuration steps

1. In Amazon Q Suite, create a new agent or select an existing agent.
2. Add an agent prompt that describes how to interact with the DLT MCP server.
3. Add a new action and select **MCP server action**.

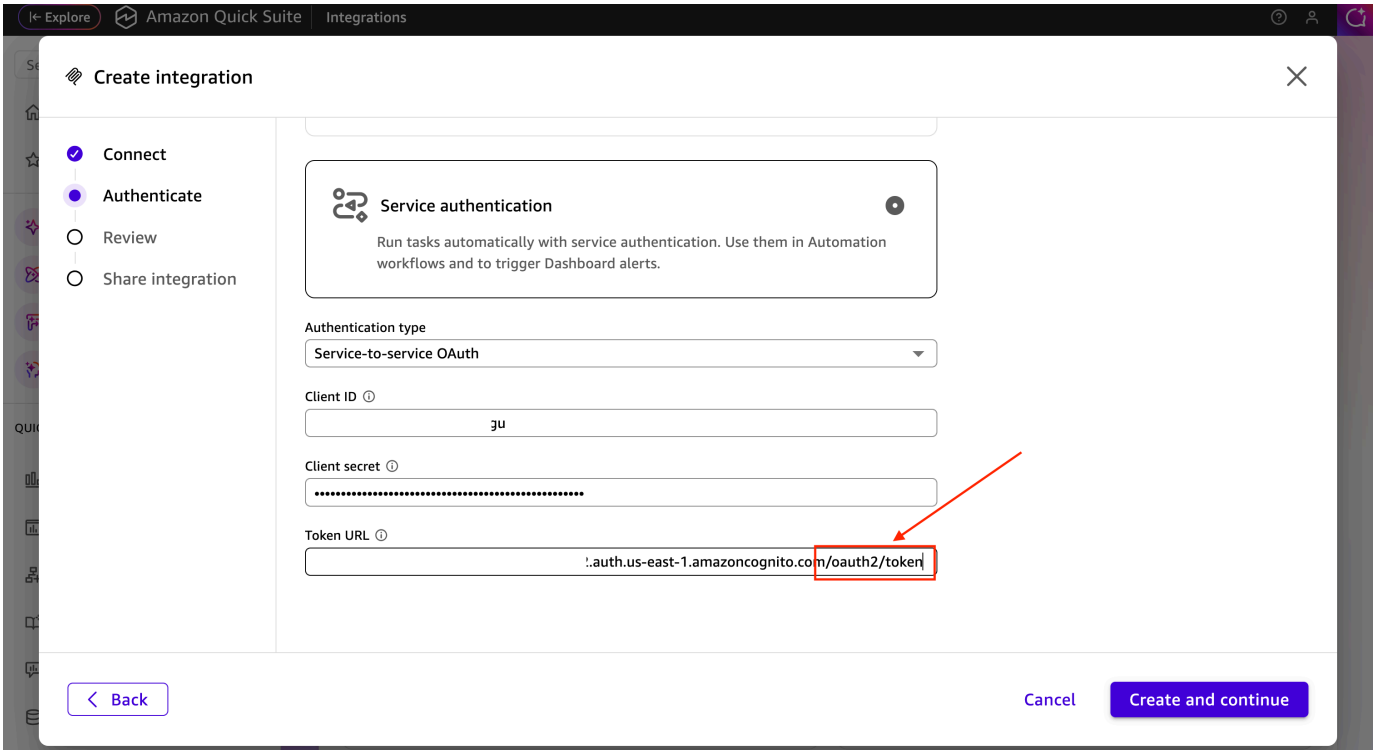


4. Configure the MCP server details:

- **MCP Server URL**: Your DLT MCP endpoint




- **Authentication Type**: Service-based authentication
- **Token Endpoint**: Your Cognito token endpoint URL
- **Client ID**: The client ID from the m2m app client
- **Client Secret**: The client secret from the m2m app client




5. Save the MCP server action configuration.
6. Add the new MCP server action to your agent.

### Launch and test the agent

1. Launch the agent in Amazon Q Suite.
2. Start a conversation with the agent using natural language prompts.
3. The agent will use the MCP tools to retrieve and analyze your load testing data.

## Example prompts

The following examples demonstrate how to interact with your AI assistant to analyze load testing data through the MCP interface. Customize the test IDs, date ranges, and criteria to match your specific testing needs.

For detailed information about available MCP tools and their parameters, refer to MCP tools specification in the Developer Guide.

## Simple test results query

Natual language interaction with the MCP Server can be as simple as `Show me the load tests that have completed in the last 24 hours with their associated completion status` or can be more descriptive such as

```
Use list_scenarios to find my load tests. Then use get_latest_test_run to show me the
 basic execution data and performance metrics for the most recent test. If the results
 look concerning, also get the detailed performance metrics using get_test_run.
```

## Interactive performance analysis with progressive disclosure

```
I need to analyze my load test performance, but I'm not sure which specific tests to
 focus on. Please help me by:

1. First, use list_scenarios to show me available test scenarios
2. Ask me which tests I want to analyze based on the list you show me
3. For my selected tests, use list_test_runs to get the test run history
4. Then use get_test_run with the test_run_id to get detailed response times,
 throughput, and error rates
5. If I want to compare tests, use get_baseline_test_run to compare against the
 baseline
6. If there are any issues, use get_test_run_artifacts to help me understand what went
 wrong

Please guide me through this step by step, asking for clarification whenever you need
 more specific information.
```

## Production readiness validation

```
Help me validate if my API is ready for production deployment:

1. Use list_scenarios to find recent test scenarios
2. For the most recent test scenario, use get_latest_test_run to get basic execution
 data
3. Use get_test_run with that test_run_id to get detailed response times, error rates,
 and throughput
4. Use get_scenario_details with the test_id to show me what load patterns and
 endpoints were tested
5. If I have a baseline, use get_baseline_test_run to compare current results with the
 baseline
```

```
6. Provide a clear go/no-go recommendation based on the performance data
7. If there are any concerns, use get_test_run_artifacts to help identify potential
 issues

My SLA requirements are: response time under [X]ms, error rate under [Y]%.
```

## Performance trend analysis

```
Analyze the performance trend for my load tests over the past [TIME_PERIOD]:

1. Use list_scenarios to get all test scenarios
2. For each scenario, use list_test_runs with start_date and end_date to get tests from
 that period
3. Use get_test_run for the key test runs to get detailed metrics
4. Use get_baseline_test_run to compare against the baseline
5. Identify any significant changes in response times, error rates, or throughput
6. If you detect performance degradation, use get_test_run_artifacts on the problematic
 tests to help identify causes
7. Present the trend analysis in a clear format showing whether performance is
 improving, stable, or degrading

Focus on completed tests and limit results to [N] tests if there are too many.
```

## Troubleshooting failed tests

```
Help me troubleshoot my failed load tests:

1. Use list_scenarios to find test scenarios
2. For each scenario, use list_test_runs to find recent test runs
3. Use get_test_run with the test_run_id to get the basic execution data and failure
 information
4. Use get_test_run_artifacts to get detailed error messages and logs
5. Use get_scenario_details to understand what was being tested when it failed
6. If I have a similar test that passed, use get_baseline_test_run to identify
 differences
7. Summarize the causes of failure and suggest next steps for resolution

Show me the most recent [N] failed tests from the past [TIME_PERIOD].
```

# 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 correspond to each solution release. By default, automatic updates are disabled, giving you complete control over when and which version updates are applied to your deployment. The AWS Solutions team uses Amazon ECR Enhanced Scanning to detect Common Vulnerabilities and Exposures (CVEs) in the base image and installed packages. When possible, the team publishes patched images with the same version tag to resolve CVEs without breaking compatibility with the released solution version.

When images are patched on the same minor version, the stable tag is automatically updated, and an additional image tag is created in the format `<solution-version>_<date-of-fix>`. If a major or minor version is released, you must perform a full stack update to get the latest image version, as the stable tag is incremented to match the solution version.

If you opt in to automatic updates during deployment, changes to the image, including CVE patches and minor bug fixes, are automatically applied up to the latest matching minor release.

## Versions

By default, this solution deploys with automatic updates disabled. This means the container image version is locked to the specific version matching your deployed solution version, providing you with full control over version updates.

If you choose to enable automatic updates by selecting **Yes** during CloudFormation deployment, the solution will automatically receive security patches and minor, non-breaking bug fixes up to the latest matching minor version. For example, if you deploy version 4.0.0 with automatic updates enabled, you will receive updates up to 4.0.x, but not 4.1.0 or higher.

To manually control the container image version, you can edit the task definition to specify a particular image version using the tagged version format. This allows you to pin to a specific image version regardless of the automatic updates setting.

## 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.

### Stack Info

- GET /stack-info

### Scenarios

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

### Test Runs

- GET /scenarios/{testId}/testruns
- GET /scenarios/{testId}/testruns/{testRunId}
- DELETE /scenarios/{testId}/testruns/{testRunId}

### Baseline

- GET /scenarios/{testId}/baseline
- PUT /scenarios/{testId}/baseline
- DELETE /scenarios/{testId}/baseline

#### Tasks

- GET /tasks
- OPTIONS /tasks

#### Regions

- GET /regions
- OPTIONS /regions

## GET /stack-info

### Description

The `GET /stack-info` operation retrieves information about the deployed stack including creation time, region, and version. This endpoint is used by the front-end.

### Response

#### 200 - Success

| Name | Description |
| --- | --- |
| `created_time` | ISO 8601 timestamp when the stack was created (for example, `2025-09-09T19:40:22Z` ) |
| `region` | AWS region where the stack is deployed (for example, `us-east-1` ) |
| `version` | Version of the deployed solution (for example, `v4.0.0`) |

### Error Responses

- `403` - Forbidden: Insufficient permissions to access stack information
- `404` - Not found: Stack information not available
- `500` - Internal server error

## 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, run time, tags, total runs, and last run 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 |

| Name | Description |
| --- | --- |
| | 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`) |
| `tags` | An array of strings for categorizing tests. Optional field with a maximum length of 5 (for example, `["blue", "3.0", "critical"]` ) |
| `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 parameters

`testId`

- The unique ID of the test

Type: String

Required: Yes

`latest`

- Query parameter to return only the latest test run. Default is `true`

  Type: Boolean

  Required: No

`history`

- Query parameter to include test run history in the response. Default is `true`. Set to `false` to exclude history

  Type: Boolean

  Required: No

## 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`) |
| `tags` | An array of strings for categorizing tests |
| `status` | The status of the test |
| `startTime` | The time and date when the last test started |

| Name | Description |
| --- | --- |
| 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 (excluded when history=false ) |
| totalRuns | The total number of test runs for this scenario |
| lastRun | The timestamp of the last test run |
| 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 |
| `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 |

| Name | Description |
| --- | --- |
| `errorReason` | An error message generated when an error occurs |

# GET /scenarios/{testId}/testruns

## Description

The `GET /scenarios/{testId}/testruns` operation retrieves test run IDs for a specific test scenario, optionally filtered by time range. When `latest=true`, returns only the single most recent test run.

## Request parameters

`testId`

- The test scenario ID

  Type: String

  Required: Yes

`latest`

- Return only the most recent test run ID

  Type: Boolean

  Default: `false`

  Required: No

`start_timestamp`

- ISO 8601 timestamp to filter test runs from (inclusive). For example, `2024-01-01T00:00:00Z`

  Type: String (date-time format)

  Required: No

`end_timestamp`

- ISO 8601 timestamp to filter test runs until (inclusive). For example, `2024-12-31T23:59:59Z`

  Type: String (date-time format)

  Required: No

`limit`

- Maximum number of test runs to return (ignored when `latest=true`)

  Type: Integer (minimum: 1, maximum: 100)

  Default: `20`

  Required: No

`next_token`

- Pagination token from previous response to get next page

  Type: String

  Required: No

## Response

### 200 - Success

| Name | Description |
| --- | --- |
| `testRuns` | Array of test run objects, each containing `testRunId` (string) and `startTime` (ISO 8601 date-time) |
| `pagination` | Object containing `limit` (integer) and `next_token` (string or null). Token is null if no more results |

### Error Responses

- `400` - Invalid timestamp format or parameters
- `404` - Test scenario not found
- `500` - Internal server error

### Example usage

- Latest test run only: `GET /scenarios/test123/testruns?latest=true`
- Latest within time range: `GET /scenarios/test123/testruns?latest=true&start_timestamp=2024-01-01T00:00:00Z`
- Next page request: `GET /scenarios/test123/testruns?limit=20&next_token=eyJ0ZXN0SWQiOiJzZVFVeTEyTEtMIiwic3RhcnRUaW1lIjoiMjAyNC0wMS`

## GET /scenarios/{testId}/testruns/{testRunId}

### Description

The `GET /scenarios/{testId}/testruns/{testRunId}` operation retrieves complete results and metrics for a specific test run. Optionally omit history results with `history=false` for faster response.

### Request parameters

`testId`

- The test scenario ID

  Type: String

  Required: Yes

`testRunId`

- The specific test run ID

  Type: String

  Required: Yes

`history`

- Include history array in response. Set to `false` to omit history for faster response

Type: Boolean

Default: `true`

Required: No

## Response

### 200 - Success

| Name | Description |
|---|---|
| `testId` | The unique ID of the test (for example, `seQUy12LKL` ) |
| `testRunId` | The specific test run ID (for example, `2DEwHItEne` ) |
| `testDescription` | Description of the load test |
| `testType` | The type of test (for example, `simple`, `jmeter`) |
| `status` | The status of the test run: `complete`, `running`, `failed`, or `cancelled` |
| `startTime` | The time and date when the test started (for example, `2025-09-09 21:01:00` ) |
| `endTime` | The time and date when the test ended (for example, `2025-09-09 21:18:29` ) |
| `succPercent` | Success percentage (for example, `100.00`) |
| `testTaskConfigs` | Array of task configuration objects containing `region`, `taskCount` , and `concurrency` |
| `completeTasks` | Object mapping regions to completed task counts |

| Name | Description |
| --- | --- |
| `results` | Object containing detailed metrics including `avg_lt` (average latency), percentiles (`p0_0`, `p50_0`, `p90_0`, `p95_0`, `p99_0`, `p99_9`, `p100_0`), `avg_rt` (average response time), `avg_ct` (average connection time), `stdev_rt` (standard deviation response time), `concurrency` , `throughput` , `succ` (success count), `fail` (failure count), `bytes`, `testDuration` , `metricS3Location` , `rc` (response codes array), and `labels` array |
| `testScenario` | Object containing test configuration with `execution` , `reporting` , and `scenarios` properties |
| `history` | Array of historical test results (excluded when `history=false` ) |

**Error Responses**

- `400` - Invalid testId or testRunId
- `404` - Test run not found
- `500` - Internal server error

# DELETE /scenarios/{testId}/testruns/{testRunId}

## Description

The `DELETE /scenarios/{testId}/testruns/{testRunId}` operation deletes all data and artifacts related to a specific test run. The test run data is removed from DynamoDB, while the actual test data in S3 remains unchanged.

### Request parameters

`testId`

- The test scenario ID

  Type: String

  Required: Yes

`testRunId`

- The specific test run ID to delete

  Type: String

  Required: Yes

### Response

#### 204 - Success

Test run successfully deleted (no content returned)

#### Error Responses

- `400` - Invalid testId or testRunId
- `403` - Forbidden: Insufficient permissions to delete test run
- `404` - Test run not found
- `409` - Conflict: Test run is currently running and cannot be deleted
- `500` - Internal server error

## GET /scenarios/{testId}/baseline

### Description

The `GET /scenarios/{testId}/baseline` operation retrieves the designated baseline test result for a scenario. Returns either the baseline test run ID or the full baseline results depending on the `data` parameter.

## Request parameters

`testId`

- The test scenario ID

  Type: String

  Required: Yes

`data`

- Return full baseline test run data if `true`, otherwise just the testRunId

  Type: Boolean

  Default: `false`

  Required: No

## Response

### 200 - Success

When `data=false` (default):

| Name | Description |
|---|---|
| `testId` | The test scenario ID (for example, `seQUy12LKL` ) |
| `baselineTestRunId` | The baseline test run ID (for example, `2DEwHItEne` ) |

When `data=true`:

| Name | Description |
|---|---|
| `testId` | The test scenario ID (for example, `seQUy12LKL` ) |

| Name | Description |
| --- | --- |
| `baselineTestRunId` | The baseline test run ID (for example, `2DEwHItEne` ) |
| `baselineData` | Complete test run results object (same structure as `GET /scenarios/{testId}/testruns/{testRunId}` ) |

**Error Responses**

- `400` - Invalid testId parameter
- `404` - Test scenario not found or no baseline set
- `500` - Internal server error

# PUT /scenarios/{testId}/baseline

## Description

The `PUT /scenarios/{testId}/baseline` operation designates a specific test run as the baseline for performance comparison. Only one baseline can be set per scenario.

## Request parameters

`testId`

- The test scenario ID

  Type: String

  Required: Yes

### Request body

| Name | Description |
| --- | --- |
| `testRunId` | The test run ID to set as baseline (for example, `2DEwHItEne` ) |

### Response

#### 200 - Success

| Name | Description |
| --- | --- |
| `message` | Confirmation message (for example, `Baseline set successfully` ) |
| `testId` | The test scenario ID (for example, `seQUy12LKL` ) |
| `baselineTestRunId` | The baseline test run ID that was set (for example, `2DEwHItEne` ) |

#### Error Responses

- `400` - Invalid testId or testRunId
- `404` - Test scenario or test run not found
- `409` - Conflict: Test run cannot be set as baseline (for example, failed test)
- `500` - Internal server error

## DELETE /scenarios/{testId}/baseline

### Description

The `DELETE /scenarios/{testId}/baseline` operation clears the baseline value for a scenario by setting it to an empty string.

### Request parameters

`testId`

- The test scenario ID

  Type: String

  Required: Yes

### Response

#### 204 - Success

Baseline successfully cleared (no content returned)

#### Error Responses

- `400` - Invalid testId
- `500` - Internal server error

## 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 |

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

# 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 |

| Name | Description |
| --- | --- |
| taskSecurityGroup | The ID of the security group in the Region |

# Increase the container resources

To increase the number of concurrent virtual users (concurrency) your load tests can simulate, you need to increase the CPU and memory resources allocated to each Amazon ECS task. This involves creating a new task definition revision with higher resource limits, then updating the solution's DynamoDB configuration to use the new task definition for future test runs.

## Create a new task definition revision

Follow these steps to create a new task definition with increased CPU and memory resources:

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 **Task CPU** to your desired values. Higher values allow more concurrent virtual users per task.
   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 match or be less than your task memory allocation.
   d. Choose **Update**.
6. On the **Create new revision** page, choose **Create**.
7. After the task definition is successfully created, record the complete task definition ARN including the version number. For example: [replaceable]<stackName>`-EcsTaskDefinition-*<system-generated-random-Hash>*:[replaceable]<system-generated-versionNumber>.

## Update the DynamoDB table

After creating the new task definition revision, you must update the solution's DynamoDB table so that future test runs use the new task definition. Repeat these steps for each AWS Region where you want to use the updated task definition:

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 created the new task definition revision. For example, `region-[replaceable]`<region-name>`.
5. In the item editor, locate the **taskDefinition** attribute and update its value with the complete task definition ARN you recorded in the previous section (including the version number).
6. Choose **Save changes**.

> **Note**
>
> The updated task definition will only be used for new test runs. Any tests that are currently running or scheduled will continue to use the previous task definition.

# MCP tools specification

The Distributed Load Testing solution exposes a set of MCP tools that enable AI agents to interact with test scenarios and results. These tools provide high-level, abstracted capabilities that align with how AI agents process information, allowing them to focus on analysis and insights rather than detailed API contracts.

> **Note**
>
> All MCP tools provide read-only access to the solution's data. No modifications to test scenarios or configurations are supported through the MCP interface.

# list_scenarios

## Description

The `list_scenarios` tool retrieves a list of all available test scenarios with basic metadata.

## Endpoint

`GET /scenarios`

## Parameters

None

## Response

| Name | Description |
|---|---|
| `testId` | Unique identifier for the test scenario |
| `testName` | Name of the test scenario |
| `status` | Current status of the test scenario |
| `startTime` | When the test was created or last run |
| `testDescription` | Description of the test scenario |

# get_scenario_details

## Description

The `get_scenario_details` tool retrieves the test configuration and most recent test run for a single test scenario.

## Endpoint

`GET /scenarios/<test_id>?history=false&results=false`

### Request parameter

`test_id`

- The unique identifier for the test scenario

  Type: String

  Required: Yes

### Response

| Name | Description |
| --- | --- |
| `testTaskConfigs` | Task configuration for each region |
| `testScenario` | Test definition and parameters |
| `status` | Current test status |
| `startTime` | Test start timestamp |
| `endTime` | Test end timestamp (if completed) |

## list_test_runs

### Description

The `list_test_runs` tool retrieves a list of test runs for a specific test scenario, sorted newest to oldest. Returns a maximum of 30 results.

### Endpoint

```
GET /scenarios/<testid>/testruns/?limit=<limit>
```

or

```
GET /scenarios/<testid>/testruns/?
limit=30&start_date=<start_date>&end_date=<end_date>
```

## Request parameters

`test_id`

- The unique identifier for the test scenario

  Type: String

  Required: Yes

`limit`

- Maximum number of test runs to return

  Type: Integer

  Default: 20

  Maximum: 30

  Required: No

`start_date`

- ISO 8601 timestamp to filter runs from specific date

  Type: String (date-time format)

  Required: No

`end_date`

- ISO 8601 timestamp to filter runs until specific date

  Type: String (date-time format)

  Required: No

## Response

| Name | Description |
| --- | --- |
| `testRuns` | Array of test run summaries with performance metrics and percentiles for each run |

# get_test_run

## Description

The `get_test_run` tool retrieves detailed results for a single test run with regional and endpoint breakdowns.

## Endpoint

`GET /scenarios/<testid>/testruns/<testrunid>`

## Request parameters

`test_id`

- The unique identifier for the test scenario

  Type: String

  Required: Yes

`test_run_id`

- The unique identifier for the specific test run

  Type: String

  Required: Yes

## Response

| Name | Description |
| --- | --- |
| `results` | Complete test run data including regional results breakdown, endpoint-specific metrics, performance percentiles (p50, p90, p95, p99), success and failure counts, response times and latency, and test configuration used for the run |

# get_latest_test_run

## Description

The `get_latest_test_run` tool retrieves the most recent test run for a specific test scenario.

## Endpoint

`GET /scenarios/<testid>/testruns/?limit=1`

> **Note**
>
> Results are sorted by time using a Global Secondary Index (GSI), ensuring the most recent test run is returned.

## Request parameter

`test_id`

- The unique identifier for the test scenario

  Type: String

  Required: Yes

## Response

| Name | Description |
| --- | --- |
| `results` | Latest test run data with the same format as `get_test_run` |

# get_baseline_test_run

## Description

The `get_baseline_test_run` tool retrieves the baseline test run for a specific test scenario. The baseline is used for performance comparison purposes.

### Endpoint

```
GET /scenarios/<test_id>/baseline
```

### Request parameter

`test_id`

- The unique identifier for the test scenario

  Type: String

  Required: Yes

### Response

| Name | Description |
| --- | --- |
| `baselineData` | Baseline test run data for comparison purposes, including all metrics and configura tion from the designated baseline run |

## get_test_run_artifacts

### Description

The `get_test_run_artifacts` tool retrieves Amazon S3 bucket information for accessing test artifacts including logs, error files, and results.

### Endpoint

```
GET /scenarios/<testid>/testruns/<testrunid>
```

### Request parameters

`test_id`

- The unique identifier for the test scenario

  Type: String

Required: Yes

`test_run_id`

- The unique identifier for the specific test run

  Type: String

  Required: Yes

## Response

| Name | Description |
| --- | --- |
| `bucketName` | S3 bucket name where artifacts are stored |
| `testRunPath` | Path prefix for current artifact storage (version 4.0+) |
| `testScenarioPath` | Path prefix for legacy artifact storage (pre-version 4.0) |

> **Note**
>
> All MCP tools leverage existing API endpoints. No modifications to the underlying APIs are required to support MCP functionality.

# Reference

This section includes information about data collection, pointers to related resources, and a list of builders who contributed to this solution.

## Data collection

This solution sends operational metrics to AWS (the "Data") about the use of this solution. We use this Data to better understand how customers use this solution and related services and products. AWS's collection of this Data is subject to the AWS Privacy Notice.

## Contributors

- Tom Nightingale
- Fernando Dingler
- Beomseok Lee
- George Lenz
- Erin McGill
- Dimitri Lopez
- Kamyar Ziabari
- Bassem Wanis
- Garvit Singh
- Nikhil Reddy
- Simon Kroll
- Ahern Knox
- Ian Downard
- Owen Brady
- Jim Thario
- Thyag Ramachandran
- Yang Qin
- James Wang

# Glossary

This glossary defines acronyms and abbreviations used throughout the Distributed Load Testing on AWS Implementation Guide.

## Technical Protocols and Formats

AGPL

Affero General Public License. An open-source software license used by K6.

API

Application Programming Interface. A set of protocols and tools for building software applications and enabling communication between different systems.

CLI

Command Line Interface. A text-based interface for interacting with software and operating systems.

CORS

Cross-Origin Resource Sharing. A security feature that allows or restricts web applications running at one origin to access resources from a different origin.

CSV

Comma-Separated Values. A file format used to store tabular data in plain text, commonly used for data export.

gRPC

gRPC Remote Procedure Call. A high-performance, open-source framework for remote procedure calls.

HTTP

Hypertext Transfer Protocol. The foundation protocol used for transmitting data on the World Wide Web.

HTTPS

HTTP Secure. An extension of HTTP that uses encryption for secure communication over a network.

JSON

JavaScript Object Notation. A lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate.

JWT

JSON Web Token. A compact, URL-safe means of representing claims to be transferred between two parties for authentication and authorization.

OAuth

Open Authorization. An open standard for access delegation commonly used for token-based authentication and authorization.

REST

Representational State Transfer. An architectural style for designing networked applications using stateless communication and standard HTTP methods.

SSE

Server-Sent Events. A server push technology enabling a client to receive automatic updates from a server via an HTTP connection.

UI

User Interface. The visual elements and controls through which users interact with software applications.

URL

Uniform Resource Locator. The address used to access resources on the internet.

XML

Extensible Markup Language. A markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.

## Testing and Database Terms

FTP

File Transfer Protocol. A standard network protocol used for transferring files between a client and server.

GSI

Global Secondary Index. A DynamoDB feature that allows you to query data using an alternate key.

JDBC

Java Database Connectivity. A Java API for connecting and executing queries with databases.

JMS

Java Message Service. A Java API for sending messages between two or more clients.

TPS

Transactions Per Second. A measure of the number of transactions a system can process in one second.

## AWS and System Terms

ARN

Amazon Resource Name. A unique identifier for AWS resources used to specify resources across AWS services.

ISO

International Organization for Standardization. An independent, non-governmental organization that develops international standards. Referenced in this guide for ISO 8601 timestamp format.

SLA

Service Level Agreement. A commitment between a service provider and a customer that defines the level of service expected.

UUID

Universally Unique Identifier. A 128-bit number used to uniquely identify information in computer systems.

vCPU

Virtual Central Processing Unit. A virtual processor assigned to a virtual machine or container, representing a portion of the physical CPU's processing power.

# Load Testing Terms

concurrency

The number of concurrent virtual users per task. This parameter controls how many simulated users each Fargate task generates during a load test.

regional stack

A CloudFormation stack deployed in an AWS Region to provide testing infrastructure for multi-region load tests.

task count

The number of Fargate containers (tasks) launched to run a test scenario. The total load generated equals task count multiplied by concurrency.

test scenario

A configured load test including test type, target endpoints, task count, concurrency, duration, and other parameters.

# Revisions

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

# Notices

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