実装ガイド

AWS での分散負荷テストソリューション

AWS での分散負荷テストソリューション: 実装ガイド

Table of Contents

ソリューションの概要 1
機能 2
利点 4
ユースケース 5
概念と定義 6
アーキテクチャの概要 8
アーキテクチャ図 8
AWS Well-Architected の設計に関する考慮事項 10
運用上の優秀性 10
セキュリティ 11
信頼性 12
パフォーマンス効率 12
コスト最適化 12
持続可能性 13
アーキテクチャの詳細 14
フロントエンド 14
負荷テスト API 14
ウェブコンソール 15
MCP サーバー (オプション) 15
バックエンド 15
コンテナイメージのパイプライン 16
テストのインフラストラクチャー 16
負荷テストエンジン 16
MCP サーバー 17
AWS AgentCore Gateway 17
DLT MCP サーバー Lambda 17
認証の統合 18
このソリューションで使用している AWS のサービス 18
AWS での分散負荷テストの仕組み 19
MCP サーバーワークフロー (オプション) 22
設計上の考慮事項 23
サポートされているアプリケーション 23
テストタイプ 24
テストのスケジューリング 26

同時テスト 27
ユーザー管理 27
リージョンデプロイ 27
デプロイを計画する 28
Cost 28
MCP サーバーの追加コスト (オプション) 29
セキュリティ 30
IAM ロール 30
Amazon CloudFront 30
Amazon API Gateway 31
AWS Fargate セキュリティグループ 31
Amazon VPC 31
ネットワーク負荷テスト 33
パブリックユーザーインターフェイスへのアクセス制限 34
MCP サーバーのセキュリティ (オプション) 34
サポートしている AWS リージョン 34
MCP サーバーでサポートされている AWS リージョン (オプション) 35
クォータ 36
このソリューション内の AWS サービスのクォータ 36
AWS CloudFormation のクォータ 36
負荷テストのクォータ 36
同時テスト 27
Amazon EC2 テストポリシー 37
Amazon CloudFront の負荷テストポリシー 37
デプロイ後のソリューションのモニタリング 37
CloudWatch アラームの設定 38
エキスパートと連携する 38
AWS での分散負荷テストの AWS Countdown Premium 短期エンゲージメント 38
ソリューションをデプロイする 41
デプロイプロセスの概要 41
AWS Launch Wizard を使用してデプロイする 42
AWS CloudFormation を使用してデプロイする 42
AWS CloudFormation テンプレート 42
スタックを起動する 43
マルチリージョンデプロイ 46
ソリューションを更新する 50

AWS Launch Wizard を使用した更新 50
AWS CloudFormation を使用して更新する 50
v3.3.0 より前のバージョンからの更新のトラブルシューティング 52
リージョンスタックの更新 53
AWS Systems Manager Application Manager 53
トラブルシューティング 54
既知の問題解決 54
AWS サポートに問い合わせる 56
ケースを作成する 56
どのようなサポートをご希望ですか? 56
追加情報 57
ケースの迅速な解決にご協力ください 57
今すぐ解決またはお問い合わせ 57
ソリューションをアンインストールする 58
AWS マネジメントコンソールの使用 58
AWS CloudFormation 58
AWS Launch Wizard 58
AWS コマンドラインインターフェイスの使用 58
Amazon S3 バケットの削除 59
ソリューションを使用する 60
テストシナリオの作成 60
ステップ 1: 全般設定 60
ステップ 2: シナリオ設定 62
ステップ 3: トラフィックシェイプ 64
ステップ 4: 確認して作成する 68
テストシナリオを実行する 68
シナリオの詳細ビュー 69
テスト実行ワークフロー 69
テスト実行のステータス 70
ライブデータを使用したモニタリング 70
テストのキャンセル 72
テスト結果を調べる 73
テスト実行のサマリーメトリクス 73
テスト実行テーブル 73
ベースラインの比較 74
詳細なテスト結果 74

エラータブ 76
アーティファクトタブ 76
S3 の結果の構造 76
MCP サーバー統合 77
ステップ 1: MCP エンドポイントとアクセストークンを取得する 77
ステップ 2: MCP Inspector を使用してテストする 78
ステップ 3: AI 開発クライアントを設定する 80
プロンプトの例 86
デベロッパーガイド 89
ソースコード 89
メンテナンス 89
バージョン 89
コンテナイメージのカスタマイズ 90
分散負荷テスト API 98
GET /stack-info 99
GET /scenarios 100
POST /scenarios 100
OPTIONS /scenarios 102
GET /scenarios/{testId} 103
POST /scenarios/{testId} 105
DELETE /scenarios/{testId} 106
OPTIONS /scenarios/{testId} 106
GET /scenarios/{testId}/testruns 107
GET /scenarios/{testId}/testruns/{testRunId} 110
DELETE /scenarios/{testId}/testruns/{testRunId} 112
GET /scenarios/{testId}/baseline 113
PUT /scenarios/{testId}/baseline 115
DELETE /scenarios/{testId}/baseline 116
GET /tasks 117
OPTIONS /tasks 117
GET /regions 117
OPTIONS /regions 118
コンテナリソースを増加する 119
新しいタスク定義のリビジョンを作成する 119
DynamoDB テーブルのアップデート 120
MCP ツールの仕様 121

list_scenarios 121
get_scenario_details 122
list_test_runs 123
get_test_run 124
get_latest_test_run 125
get_baseline_test_run 126
get_test_run_artifacts 127
リファレンス 129
データ収集 129
寄稿者 129
用語集 130
技術的なプロトコルと形式 130
テストとデータベースの用語 131
AWS およびシステムの用語 132
負荷テストの用語 133
改訂 134
注意 135

ソフトウェアアプリケーションのテストを大規模に自動化

発行日: 2025 年 12 月

AWS での分散負荷テストでは、アプリケーションをリリースする前に、ソフトウェアアプリケーションの大規模なパフォーマンステストを自動化して、ボトルネックを特定することができます。このソリューションは、ある持続頻度で HTTP レコードを生成する数多くの接続ユーザーをシミュレートします。サーバーをプロビジョニングする必要はありません。

このソリューションでは、AWS Fargate で Amazon Elastic Container Service (Amazon ECS) を活用して、負荷テストシミュレーションを実行するコンテナをデプロイし、次の機能を提供しています。

- 単独で実行できる AWS Fargate コンテナに Amazon ECS をデプロイして、アプリケーションの負荷容量をテストできます。
- 複数の AWS リージョンにまたがる数万人の接続ユーザーをシミュレートし、継続的なペースでレコードを生成します。
- JMeter、K6、Locust テストスクリプト、またはシンプルな HTTP エンドポイント設定を使用してアプリケーションテストをカスタマイズします。
- 負荷テストを、すぐに、将来の日時で、または定期的なスケジュールで実行するようにスケジュールします。
- さまざまなシナリオとリージョンで複数の負荷テストを同時に実行します。

この実装ガイドでは、AWS での分散負荷テストソリューションの概要、そのリファレンスアーキテクチャとコンポーネント、デプロイを計画する際の考慮事項、Amazon Web Services (AWS) クラウドにソリューションをデプロイするための設定手順について説明します。これには、セキュリティと可用性に関する AWS のベストプラクティスを使用してこのソリューションをデプロイするために必要な AWS のサービスを起動および設定する AWS CloudFormation テンプレートへのリンクが含まれています。

このソリューションの特徴と機能を環境で使用する場合、対象者には、AWS クラウドでのアーキテクトの実務経験を持つ IT インフラストラクチャアーキテクト、管理者、DevOps プロフェッショナルが含まれます。

このナビゲーションテーブルを使用すると、次の質問に対する回答をすばやく見つけることができます。

質問内容	参照先
このソリューションの実行に必要なコストを確認する。 米国東部 (バージニア北部) リージョンでこのソリューションを実行するための AWS リソースの推定コストは、1 か月あたり 30.90 USD です。	コスト
このソリューションのセキュリティ上の考慮事項を理解する。	セキュリティ
このソリューションのクォータを計画する方法を確認する。	クォータ
どの AWS リージョンでこのソリューションをサポートしているのかを確認する。	サポートしている AWS リージョン
AI 支援型の負荷テスト分析用のオプションの MCP サーバーについて説明します。	MCP サーバー統合
このソリューションに含まれている AWS CloudFormation テンプレートを表示またはダウンロードして、このソリューションのインフラストラクチャリソース (スタック) を自動的にデプロイする。	AWS CloudFormation テンプレート
ソースコードにアクセスし、オプションで AWS Cloud Development Kit (AWS CDK) を使用してソリューションをデプロイする。	GitHub リポジトリ。

機能

このソリューションには次のような特徴があります。

複数のテストフレームワークのサポート

JMeter、K6、Locust のテストスクリプト、およびカスタムスクリプトを必要としないシンプルな HTTP エンドポイントテストをサポートします。詳細については、「アーキテクチャの詳細」セクションの「テストタイプ」を参照してください。

高いユーザー負荷のシミュレーション

数万の同時仮想ユーザーをシミュレートして、現実的な負荷条件下でアプリケーションのストレステストを行います。

マルチリージョン負荷分散

複数の AWS リージョンに負荷テストを分散して、地理的に分散されたユーザートラフィックをシミュレートし、グローバルパフォーマンスを評価します。

柔軟なテストスケジュール

自動リグレッションテストに cron 式を使用して、テストをすぐに、将来の特定の日時に、または定期的なスケジュールで実行するようにスケジュールします。

リアルタイムモニタリング

応答時間、仮想ユーザー数、リクエストの成功率などのリアルタイムメトリクスを使用してテストの進行状況をモニタリングするためのオプションのライブデータストリームを提供します。

包括的なテスト結果

パフォーマンスメトリクス、パーセンタイル (p50、p90、p95、p99)、エラー分析、オフライン分析用のダウンロード可能なアーティファクトを含む詳細なテスト結果を表示します。

ベースラインの比較

パフォーマンス比較用のベースラインテスト実行を指定して、時間の経過に伴う改善やリグレッションを追跡します。

エンドポイントの柔軟性

AWS リージョン、オンプレミス環境、またはその他のクラウドプロバイダー間で HTTP または HTTPS エンドポイントをテストします。

直感的なウェブコンソール

コマンドライン操作を必要とせずに、テストを作成、管理、モニタリングするためのウェブベースのコンソールを提供します。

AI 支援型分析 (オプション)

負荷テストデータのインテリジェントな分析のために、モデルコンテキストプロトコル (MCP) サーバーを介して AI 開発ツールと統合します。

複数のプロトコルのサポート

カスタムテストスクリプトを通じて、HTTP、HTTPS、WebSocket、JDBC、JMS、FTP、gRPC などのさまざまなプロトコルをサポートしています。

利点

このソリューションは次の利点があります。

包括的なパフォーマンステスト

負荷テスト、ストレステスト、耐久性テストをサポートし、さまざまな条件下でアプリケーションのパフォーマンスを徹底的に評価します。

問題の早期検出

本番稼働のデプロイの前に、パフォーマンスのボトルネック、メモリリーク、スケーラビリティの問題を特定し、停止のリスクを軽減します。

実際の使用状況シミュレーション

実際のユーザー動作とトラフィックパターンを正確にシミュレートして、現実的な条件下でアプリケーションのパフォーマンスを検証します。

Actionable Performance Insights

詳細なメトリクス、パーセンタイル、エラー分析を提供して、アプリケーションの動作を理解し、最適化作業をガイドします。

自動化テストワークフロー

手動による介入なしで、継続的なパフォーマンスモニタリングとリグレッションテストのスケジュールされたテストと定期的なテストを有効にします。

コスト効率の高いインフラストラクチャ

サーバーレス AWS Fargate コンテナを従量課金制の価格で使用するため、専用のテストインフラストラクチャや継続的なサブスクリプション料金の必要はありません。

迅速なテストデプロイ

サーバーのプロビジョニングや管理を行わずに、テストインフラストラクチャを数分でデプロイしてスケーリングします。

テスト結果の簡単な調査

オプションの モデルコンテキストプロトコル (MCP) サーバーを介して AI 開発ツールと統合することで、自然言語クエリと負荷テストデータのインテリジェントな分析が可能になり、より迅速なインサイトとトラブルシューティングが可能になります。

ユースケース

本番稼働前の検証

新しいバージョンを起動する前に、本稼働環境と同様の負荷条件下でウェブアプリケーションとモバイルアプリケーションをテストして、パフォーマンスを検証し、問題を特定します。

容量計画

アプリケーションが現在のインフラストラクチャでサポートできる同時ユーザーの最大数を決定し、スケーリングが必要なタイミングを特定します。

ピーク負荷の検証

パフォーマンスを低下させることなく、インフラストラクチャがピーク負荷、季節的なトラフィックスパイク、または予期しない需要の急増に対応できることを確認します。

パフォーマンスの最適化

データベースクエリの遅延、非効率的なコード実行、ネットワークレイテンシー、またはリソースの制約などのパフォーマンスのボトルネックを特定します。

リグレッションテスト

定期的な負荷テストをスケジュールして、新しいコードデプロイやインフラストラクチャの変更によって導入されたパフォーマンスリグレッションを検出します。

グローバルパフォーマンスの評価

複数の地理的リージョンからアプリケーションのパフォーマンスを評価し、グローバルオーディエンスに一貫したユーザーエクスペリエンスを提供します。

API 負荷テスト

REST API、GraphQL エンドポイント、またはマイクロサービスをテストして、負荷時の応答時間、スループット、エラー率を検証します。

CI/CD パイプラインの統合

開発サイクルの早い段階でパフォーマンスの問題を検出するために、自動化されたパフォーマンステストを継続的インテグレーションおよびデプロイパイプラインに統合します。

サードパーティーのサービスのテスト

さまざまな負荷条件下で、アプリケーションが依存するサードパーティー API またはサービスのパフォーマンスと信頼性をテストします。

概念と定義

このセクションでは、主要な概念について説明し、このソリューション固有の用語を定義します。

シナリオ

テストの名前、説明、タスク数、同時実行数、AWS リージョン、ランプアップ、保持、テストタイプ、スケジュール日、繰り返し設定などのテスト定義。

タスク数

テストシナリオを実行するために Fargate クラスターで起動されるコンテナの数。Fargate リソースのアカウント制限に達すると、追加のタスクは作成されません。ただし、既に実行されているタスクは続行されます。

同時実行

同時実行 (タスクごとの同時仮想ユーザーの数)。デフォルト設定に基づき推奨される同時実行は 200 です。同時実行数は CPU とメモリによって制限されます。Apache JMeter に基づくテストでは、同時実行数が多いほど、ECS タスクで JVM が使用するメモリが増加します。デフォルトの ECS タスク定義では、メモリが 4 GB のタスクを作成します。1 つのタスクの同時実行値を低い値から開始して、タスククラスターの ECS CloudWatch メトリクスをモニタリングすることをお勧めします。「Amazon ECS クラスターの使用率メトリクス」を参照してください。

ランプアップ

ゼロからターゲットの同時実行レベルまで徐々に増加する期間。

ホールド

ランプアップの完了後にターゲット同時実行レベルを維持する期間。

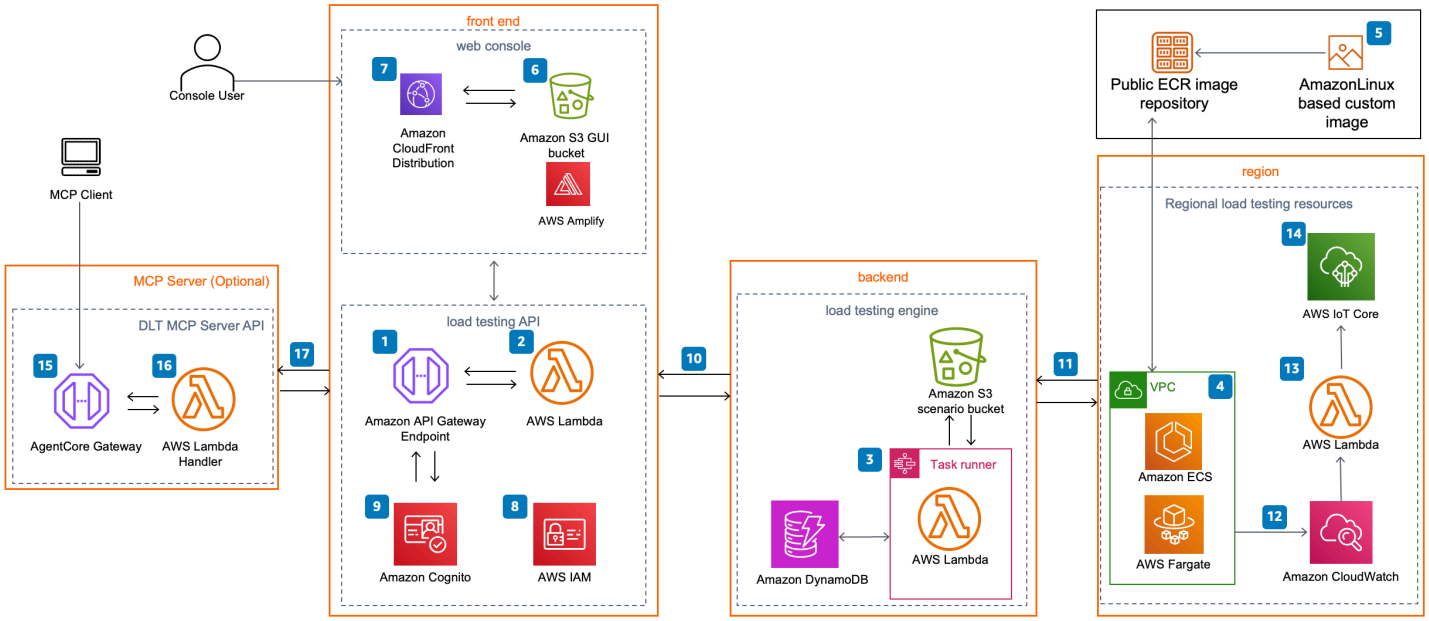
AWS 用語の一般的なリファレンスについては、「AWS 用語集」を参照してください。

アーキテクチャの概要

アーキテクチャ図

このソリューションをデフォルトのパラメータを使用してデプロイすると、AWS アカウントに次のコンポーネントがデプロイされます。

AWS 上の AWS での分散負荷テストのアーキテクチャ



> **Note**
>
> AWS CloudFormation のリソースは、AWS Cloud Development Kit (AWS CDK) のコンストラクトで作成されています。

AWS CloudFormation テンプレートを使用してデプロイされたこのソリューションコンポーネントの大まかなプロセスフローは次のとおりです。

1. 分散負荷テスター API は、Amazon API Gateway を利用してソリューションのマイクロサービス (AWS Lambda 関数) を呼び出します。
2. マイクロサービスでは、テストデータを管理しテストを実行するためのビジネスロジックを提供しています。

3. これらのマイクロサービスは、Amazon Simple Storage Service (Amazon S3)、Amazon DynamoDB、AWS Step Functions とやり取りをしてテストシナリオの詳細と結果を保存し、テスト実行をオーケストレーションします。
4. AWS Fargate で稼働するソリューションの Amazon Elastic Container Service (Amazon ECS) コンテナを含む Amazon Virtual Private Cloud (Amazon VPC) ネットワークトポロジがデプロイされます。
5. このコンテナは、Taurus 負荷テストフレームワークがインストールされた Amazon Linux 2023 ベースイメージを使用します。Taurus は、JMeter、K6、Locust、および他のテストツールをサポートするオープンソースのテスト自動化フレームワークです。コンテナイメージは、Open Container Initiative (OCI) に対応しており、Amazon Elastic Container Registry (Amazon ECR) のパブリックリポジトリで AWS がホストしています。詳細については、このガイドの「コンテナイメージのカスタマイズ」セクションを参照してください。
6. AWS Amplify を活用したウェブコンソールは、静的ウェブホスティング用に設定された S3 バケットにデプロイされます。
7. Amazon CloudFront は、このソリューションのウェブサイトバケットのコンテンツに対して、安全なパブリックアクセスを提供します。
8. 初期設定時に、ソリューションはデフォルトの管理者ロール (IAM ロール) を作成し、お客様が指定したユーザーの E メールアドレスにアクセス招待を送信します。
9. Amazon Cognito ユーザープールは、コンソール、分散負荷テスト API、MCP サーバーへのユーザーアクセスを管理します。
10. このソリューションをデプロイしたら、ウェブコンソールまたは API を使用して、一連のタスクを定義するテストシナリオを作成し、実行できます。
11. マイクロサービスはこのテストシナリオを使用して、指定されたリージョンの Fargate 上で ECS タスクを実行します。
12. テストが完了すると、ソリューションは結果を S3 と DynamoDB に保存し、ログ出力は Amazon CloudWatch に記録します。
13. ライブデータオプションを有効化すると、ソリューションは、テストを実行するリージョンごとに、テスト中に Fargate タスクから Lambda 関数に CloudWatch ログを送信します。
14. Lambda 関数は、メインスタックがデプロイされたリージョンの AWS IoT Core の対応するトピックにデータを公開します。ウェブコンソールはトピックをサブスクライブし、テストの実行中にリアルタイムデータを表示します。

> **Note**
>
> 次の手順では、AI 支援型負荷テスト分析用のオプションの MCP サーバー統合について説明します。このコンポーネントは、ソリューションのデプロイ中に MCP サーバーのオプションを選択した場合にのみデプロイされます。

15MCP クライアント (AI 開発ツール) は AWS AgentCore Gateway エンドポイントに接続して、モデルコンテキストプロトコルを介して分散負荷テストソリューションのデータにアクセスします。AgentCore Gateway は、ユーザーの Cognito 認証トークンを検証して、MCP サーバーへの認可されたアクセスを確認します。

16認証が成功すると、AgentCore Gateway は MCP ツールリクエストを DLT MCP サーバーの Lambda 関数に転送します。Lambda 関数は構造化データを AgentCore Gateway に返し、AI 支援型分析とインサイトのために MCP クライアントに送り返します。

17Lambda 関数はリクエストを処理し、適切な AWS リソース (DynamoDB テーブル、S3 バケット、または CloudWatch ログ) をクエリして、リクエストされた負荷テストデータを取得します。

AWS Well-Architected の設計に関する考慮事項

このソリューションでは、AWS Well-Architected フレームワークのベストプラクティスを使用しています。これにより、お客様は信頼性が高く、安全で、効率的で、コスト効率の高いワークロードをクラウド上で設計し運用することができます。

このセクションでは、Well-Architected Framework の設計原則とベストプラクティスがこのソリューションにどのように役立つかについて説明します。

運用上の優秀性

このセクションでは、オペレーショナルエクセレンスの柱に関する原則とベストプラクティスを用いてこのソリューションをどのように設計したかを説明します。

- すべてのリソースは、AWS CDK コンストラクトから生成された AWS CloudFormation テンプレートを使用して、Infrastructure as Code として定義されます。
- ソリューションは、さまざまな段階でメトリクスを CloudWatch にプッシュして、Lambda 関数、ECS タスク、S3 バケット、その他のソリューションコンポーネントにオブザーバビリティを提供します。

セキュリティ

このセクションでは、このソリューションを設計する際に、セキュリティの柱の原則とベストプラクティスをどのように適用したかについて説明します。

- Cognito は、ウェブコンソールユーザーと API リクエストを認証および認可します。
- すべてのサービス間通信は、必要な最小限のアクセス許可のみを含む、最小特権アクセスを持つ AWS Identity and Access Management (IAM) ロールを使用します。
- S3 バケットや DynamoDB テーブルを含むすべてのデータストレージは、AWS マネージドキーを使用して、保管中のデータを暗号化します。
- ログ記録、トレース、バージョニングは、監査およびコンプライアンスの目的で該当する場合に有効化されます。
- ネットワークアクセスはデフォルトでプライベートで、トラフィックを AWS ネットワーク内に保持できる VPC エンドポイントが有効化されています。

> **Note**
>
> このソリューションは、ログの量とコストの考慮事項に基づいて保持期間が異なる複数の CloudWatch ロググループを作成します。
>
> | ログのタイプ | 保持期間 |
> | --- | --- |
> | ECS コンテナのインサイト | 1 日 |
> | Step Functions、ECS カスタムログ、API ゲートウェイアクセスログ | 1 年 |
> | Lambda ランタイムログ | 2 年 |
> | API ゲートウェイ実行ログ | 有効期限なし |
>
> これらの保持期間は、必要に応じて CloudWatch コンソールで変更できます。

信頼性

このセクションでは、信頼性の柱に関する原則とベストプラクティスを用いてこのソリューションをどのように設計したかを説明します。

- ソリューションでは、可能な限り AWS のサーバーレスサービス (Lambda、API Gateway、Amazon S3、AWS Step Functions、Amazon DynamoDB、AWS Fargate など) を使用して、高可用性とサービス障害からの復旧を確保します。
- すべてのコンピューティング処理には、Lambda 関数または AWS Fargate 上の Amazon ECS を使用します。
- データは DynamoDB と Amazon S3 に保存されるため、デフォルトで複数のアベイラビリティーゾーンに保持されます。

パフォーマンス効率

このセクションでは、パフォーマンス効率の柱に関する原則とベストプラクティスを用いてこのソリューションをどのように設計したかを説明します。

- このソリューションは、必要に応じて水平方向にスケールできるサーバーレスアーキテクチャを使用します。
- このソリューションは、対象の AWS のサービス (AWS Lambda、Amazon API Gateway、Amazon S3、AWS Step Functions、Amazon DynamoDB、Amazon ECS、AWS Fargate、Amazon Cognito など) をサポートする任意のリージョンで起動できます。
- ソリューションでは、全体を通してマネージドサービスを使用しており、リソースのプロビジョニングと管理の運用上の負担を軽減しています。
- このソリューションは、AWS のサービスの変更に応じて一貫性を確保するために毎日自動的にテストおよびデプロイされます。また、実験と改善が必要な分野については、ソリューションアーキテクトや対象分野の エキスパートによってレビューされます。

コスト最適化

このセクションでは、このソリューションを設計する際に、コスト最適化の柱の原則とベストプラクティスをどのように適用したかを説明します。

- ソリューションはサーバーレスアーキテクチャを使用しているため、お客様は使用した分のみ課金されます。

- Amazon DynamoDB は必要に応じてキャパシティをスケールするため、お支払いいただくのは使用したキャパシティに対してのみになります。
- AWS Fargate の AWS ECS では、使用したコンピューティングリソースに対してのみ料金を支払うことが可能で、前払い料金は発生しません。
- AWS AgentCore Gateway は、分散負荷テスト API に対する費用対効果の高い Lambda ベースのプロキシとして機能し、専用インフラストラクチャの必要の削減と、サーバーレスのリクエストごとの支払い料金を通じたコスト削減が行われます。

持続可能性

このセクションでは、このソリューションを設計する際に、持続可能性の柱の原則とベストプラクティスをどのように適用したかを説明します。

- ソリューションでは、マネージドサーバーレスサービスを使用して、継続的に運用されているオンプレミスサービスと比較して、バックエンドサービスの環境への影響を最小限に抑えます。
- サーバーレスサービスでは、必要に応じてスケールアップまたはスケールダウンできます。

アーキテクチャの詳細

このセクションでは、このソリューションを構成するコンポーネントと AWS のサービス、およびこれらのコンポーネントがどのように連携するのかについてのアーキテクチャの詳細について説明します。

AWS ソリューションでの分散負荷テストは、フロントエンド、バックエンド、およびオプションの MCP サーバーの 3 つのハイレベルなコンポーネントで構成されます。

フロントエンド

フロントエンドは、ソリューションとやり取りするためのインターフェイスを提供し、以下のものを含みます。

- プログラムによるアクセスのための負荷テスト API
- パフォーマンステストを作成、スケジューリング、実行するためのウェブコンソール
- テスト結果とエラーの AI 支援型分析のためのオプションの MCP サーバー

負荷テスト API

AWS での分散負荷テストでは、ソリューションの RESTful API をホストするよう Amazon API Gateway を設定します。ユーザーは、付属のウェブコンソール、RESTful API、およびオプションの MCP サーバー を使用して、安全に負荷テストシステムとやり取りできます。API は、Amazon DynamoDB に保存されているテストデータにアクセスするための「フロントドア」として機能します。また、API を使用して、ソリューションに独自に拡張機能を組み込むことも可能です。

このソリューションでは、Amazon Cognito ユーザープールのユーザー認証機能を使用します。ユーザーが正常に認証されると、Amazon Cognito は、コンソールからソリューションの API (Amazon API Gateway エンドポイント) にリクエストの送信を許可する JSON のウェブトークンを発行します。HTTPS リクエストは、トークンを含む認証ヘッダーとともにコンソールから API に送信されます。

リクエストに基づいて、API Gateway は適切な AWS Lambda 関数を呼び出して、DynamoDB テーブルに保存されるデータに対して必要なタスクを実行し、テストシナリオを JSON オブジェクトとして Amazon S3 に保存し、Amazon CloudWatch メトリクスのイメージを取得して、テストシナリオを AWS Step Functions ステートマシンに送信します。

ソリューションの API の詳細については、このガイドの「Distributed load testing API」セクションを参照してください。

ウェブコンソール

このソリューションには、テストの設定と実行、実行中のテストのモニタリング、詳細なテスト結果の表示に使用できるウェブコンソールが含まれています。コンソールは、直感的なウェブアプリケーションを構築するオープンソースの設計システムである Cloudscape で構築された ReactJS アプリケーションです。コンソールは、Amazon S3 でホストされ、Amazon CloudFront を介してアクセスします。アプリケーションには AWS Amplify を使用しており、Amazon Cognito と統合してユーザーを認証します。ウェブコンソールには、実行中のテストのライブデータを表示するオプションもあり、AWS IoT Core の対応するトピックをサブスクライブします。

ウェブコンソールの URL は CloudFront ディストリビューションのドメイン名で、CloudFormation の出力で Console として確認できます。CloudFormation テンプレートを起動すると、ウェブコンソールの URL とそれにログインするためのワンタイムパスワードが記載された E メールも届きます。

MCP サーバー (オプション)

オプションのモデルコンテキストプロトコル (MCP) サーバーは、自然言語でのやり取りを通じて負荷テストデータにアクセスし、分析するための AI 開発ツール用の追加のインターフェイスを提供します。このコンポーネントは、ソリューションのデプロイ中に MCP サーバーのオプションを選択した場合にのみデプロイされます。

MCP サーバーを使用すると、AI エージェントは Amazon Q、Claude、その他の MCP 互換 AI アシスタントなどのツールを使用して、テスト結果のクエリ、パフォーマンスメトリクスの分析、負荷テストデータについてのインサイトを取得できます。MCP サーバーのアーキテクチャと設定の詳細については、このセクションの「MCP Server」を参照してください。

バックエンド

バックエンドは、テスト用の負荷を生成するために使用するコンテナイメージのパイプラインと負荷テストエンジンで構成されます。フロントエンドを介してバックエンドとやり取りします。さらに、テストごとに起動された AWS Fargate 上の Amazon ECS タスクは、一意のテスト識別子 (ID) でタグ付けされます。これらのテスト ID タグは、このソリューションのコストをモニタリングするのに役立ちます。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「User-Defined Cost Allocation Tags」を参照してください。

コンテナイメージのパイプライン

このソリューションでは、Amazon Linux 2023 で構築されたコンテナイメージを、Taurus 負荷テストフレームワークがインストールされたベースイメージとして使用します。Taurus は、JMeter、K6、Locust、および他のテストツールをサポートするオープンソースのテスト自動化フレームワークです。AWS は、Amazon Elastic Container Registry (Amazon ECR) のパブリックリポジトリのイメージをホストします。ソリューションはこのイメージを使用し、AWS Fargate クラスター上の Amazon ECS でタスクを実行します。

詳細については、このガイドの「Container image customization」セクションを参照してください。

テストのインフラストラクチャー

ソリューションは、メインの CloudFormation テンプレートに加えて、複数のリージョンでテストを実行するために必要なリソースを起動するためのリージョン用のテンプレートを提供します。ソリューションは Amazon S3 にこのテンプレートを保存し、そのリンクはウェブコンソールに表示されます。各リージョンスタックには、VPC、AWS Fargate クラスター、ライブデータを処理するための Lambda 関数が含まれています。

追加のリージョンにテストインフラストラクチャをデプロイする方法の詳細については、このガイドの「マルチリージョンデプロイ」セクションを参照してください。

負荷テストエンジン

分散負荷テストソリューションでは、Amazon Elastic Container Service (Amazon ECS) と AWS Fargate を使用して、複数のリージョンにまたがる数千の接続ユーザーをシミュレートし、ある持続頻度で HTTP リクエストを生成します。

テストパラメータは、付属のウェブコンソールを使用して定義します。ソリューションは、これらのパラメータを使用して JSON のテストシナリオを生成し、Amazon S3 に保存します。テストスクリプトとテストパラメータの詳細については、このセクションの「テストタイプ」を参照してください。

AWS Step Functions ステートマシンでは、AWS Fargate クラスターで Amazon ECS タスクを実行してモニタリングします。AWS Step Functions ステートマシンには、ecr-checker AWS Lambda 関数、task-status-checker AWS Lambda 関数、task-runner AWS Lambda 関数、task-canceler AWS Lambda 関数、results-parser AWS Lambda 関数が含まれます。ワークフローの詳細については、このガイドの「Test workflow」セクションを参照してください。テスト結果の詳細については、この

ガイドの「Test results」セクションを参照してください。テストキャンセルワークフローの詳細については、このガイドの「Test cancellation workflow」セクションを参照してください。

ライブデータを選択すると、ソリューションは、そのリージョンの Fargate タスクに対応する CloudWatch Logs で、リージョンごとに real-time-data-publisher Lambda 関数を起動します。その後、ソリューションはデータを処理してから、メインスタックを起動したリージョン内の AWS IoT Core のトピックに公開します。詳細については、このガイドの「Live data」セクションを参照してください。

MCP サーバー

オプションのモデルコンテキストプロトコル (MCP) サーバーの統合により、AI エージェントは自然言語でのやり取りを通じ、プログラムにより負荷テストデータにアクセスし、分析できます。このコンポーネントは、ソリューションのデプロイ中に MCP サーバーのオプションを選択した場合にのみデプロイされます。

MCP サーバーは、AI 開発ツールと DLT デプロイとの橋渡しとして機能し、パフォーマンステスト結果をインテリジェントに分析するための標準化されたインターフェイスを提供します。このアーキテクチャは、いくつかの AWS サービスを統合して、AI エージェントとのやり取りのための安全でスケーラブルなインターフェイスを作成します。

AWS AgentCore Gateway

AWS AgentCore Gateway は、MCP サーバーの標準化されたホスティングとプロトコル管理を提供する完全マネージド型サービスです。このソリューションで、AgentCore Gateway は、AI エージェントが負荷テストデータへのアクセスをリクエストするときに接続するパブリックエンドポイントとして機能します。

このサービスは、ツールの検出、認証トークンの検証、リクエストルーティングなど、すべての MCP プロトコル通信を処理します。AgentCore Gateway は、パブリックエンドポイントへの一般的な脅威に対するセキュリティ保護が組み込まれたマルチテナントサービスとして動作し、リクエストごとに Cognito トークンの署名とクレームを検証します。

DLT MCP サーバー Lambda

DLT MCP サーバー Lambda 関数は、AI エージェントからの MCP リクエストを処理し、DLT リソースに対するクエリに変換するカスタムサーバーレスコンポーネントです。

この Lambda 関数は、MCP 統合のインテリジェンスレイヤーとして機能し、DynamoDB テーブルからテスト結果を取得し、S3 バケットに保存されているパフォーマンスアーティファクトにアクセ

スし、CloudWatch ログに詳細な実行情報をクエリします。Lambda 関数は読み取り専用アクセスパターンを実装し、未処理の DLT データを、エージェントが簡単に解釈して分析できる構造化された AI フレンドリーな形式に変換します。

認証の統合

認証システムは、既存の Cognito ユーザープールインフラストラクチャを活用して、ウェブコンソールと MCP サーバーのインターフェイスの両方で一貫したアクセスコントロールを維持します。

この統合では、OAuth 2.0 トークンベースの認証を使用します。ユーザーは Cognito ログインプロセスを通じて 1 回認証され、UI インタラクションと MCP サーバーアクセスの両方で機能するトークンを受け取ります。システムは、ウェブインターフェイスと同じアクセス許可の境界とアクセスコントロールを維持し、ユーザーが AI エージェント経由でのみアクセスできるようにし、ユーザーはコンソールからアクセスできるのと同じ負荷テストデータにアクセスできます。

このソリューションで使用している AWS のサービス

このソリューションには、次の AWS サービスが含まれています。

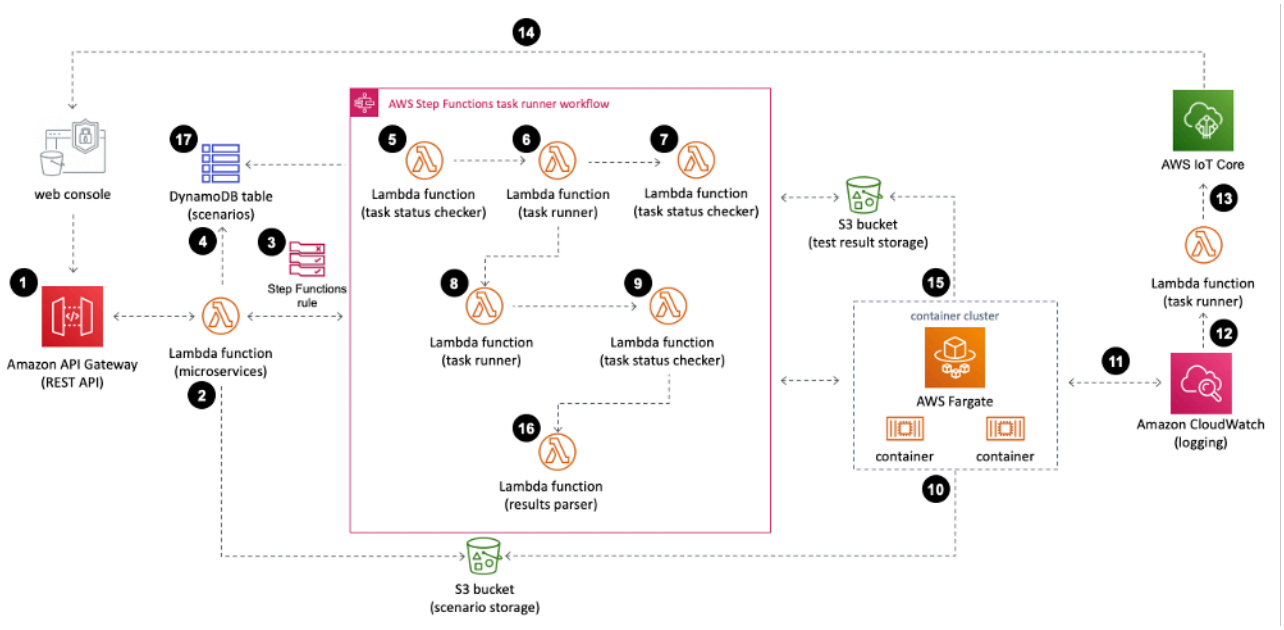
AWS のサービス	説明
Amazon API Gateway	コア。ソリューション内の REST API エンドポイントをホストします。
AWS CloudFormation	コア。ソリューションインフラストラクチャのデプロイを管理します。
Amazon CloudFront	コア。Amazon S3 でホストされるウェブコンテンツを提供します。
Amazon CloudWatch ()	コア。ソリューションのログとメトリクスを保存します。
Amazon Cognito	コア。API でのユーザーの管理と認証を処理します。
– Amazon DynamoDB	コア。デプロイ情報とテストシナリオの詳細と結果を保存します。
Amazon Elastic Container Service	コア。AWS Fargate コンテナ上で独立した Amazon ECS タスクをデプロイおよび管理します。
AWS Fargate	コア。ソリューションの Amazon ECS コンテナをホストします。

AWS のサービス	説明
AWS Identity and Access Management	コア。ユーザーのロールとアクセス許可を管理します。
* AWS Lambda*	コア。API の実装、テスト結果の解析、ワーカー/リーダータスクの起動に関するロジックを提供します。
AWS Step Functions	コア。指定されたリージョンの AWS Fargate タスクでの Amazon ECS コンテナのプロビジョニングをオーケストレーションします。
AWS Amplify	サポート。AWS Amplify で動作するウェブコンソールを提供します。
Amazon CloudWatch Events ()	サポート。特定の日付または定期的な日付でテストが自動的に開始されるようにスケジューリングします。
Amazon Elastic Container Registry	サポート。コンテナイメージをパブリック ECR リポジトリでホストします。
AWS IoT Core	サポート。AWS IoT Core の対応するトピックにサブスクライブして、実行中のテストのライブデータを表示できるようにします。
AWS Systems Manager	サポート。アプリケーションレベルのリソースの監視と、リソースの操作とコストデータの可視化を提供します。
Amazon S3	サポート。静的ウェブコンテンツ、ログ、メトリクス、テストデータをホストします。
Amazon Virtual Private Cloud	サポート。AWS Fargate で実行しているソリューションの Amazon ECS コンテナが含まれます。
Amazon Bedrock AgentCore	サポート、オプション。AI エージェントと API の統合のために、ソリューションのオプションであるリモートモデルコンテキストプロトコル (MCP) サーバーをホストします。

AWS での分散負荷テストの仕組み

次の詳細な内訳は、テストシナリオの実行に関連するステップを示しています。

テストワークフロー



1. ウェブコンソールを使用して、設定の詳細を含むテストシナリオをソリューションの API に送信します。
2. テストシナリオ設定は JSON ファイル (`s3://<bucket-name>/test-scenarios/<$TEST_ID>/<$TEST_ID>.json`) として Amazon Simple Storage Service (Amazon S3) にアップロードされます。
3. AWS Step Functions ステートマシンは、テスト ID、タスク数、テストタイプ、ファイルタイプを AWS Step Functions ステートマシンのインプットとして使用し実行します。テストがスケジューリングされている場合、最初に CloudWatch Events ルールが作成され、その後、指定された日付に AWS Step Functions がトリガーされます。スケジューリングワークフローの詳細については、このガイドの「テストスケジューリングワークフロー」セクションを参照してください。
4. 設定の詳細は、scenarios Amazon DynamoDB テーブルに保存されます。
5. AWS Step Functions タスクランナーワークフローでは、task-status-checker AWS Lambda 関数が、同じテスト ID に対して Amazon Elastic Container Service (Amazon ECS) タスクがすでに実行されているかどうかを確認します。同じテスト ID を持つタスクが実行中であることが判明すると、エラーが発生します。AWS Fargate クラスターで実行中の Amazon ECS タスクがない場合、関数はテスト ID、タスク数、テストタイプを返します。
6. task-runner AWS Lambda 関数は、前のステップからタスクの詳細を取得し、AWS Fargate クラスターで Amazon ECS ワーカータスクを実行します。Amazon ECS API は、RunTask アクションを使用してワーカータスクを実行します。これらのワーカータスクが起動され、テストを開始するためにリーダータスクからの開始メッセージを待ちます。RunTask アクションは、定義ごと

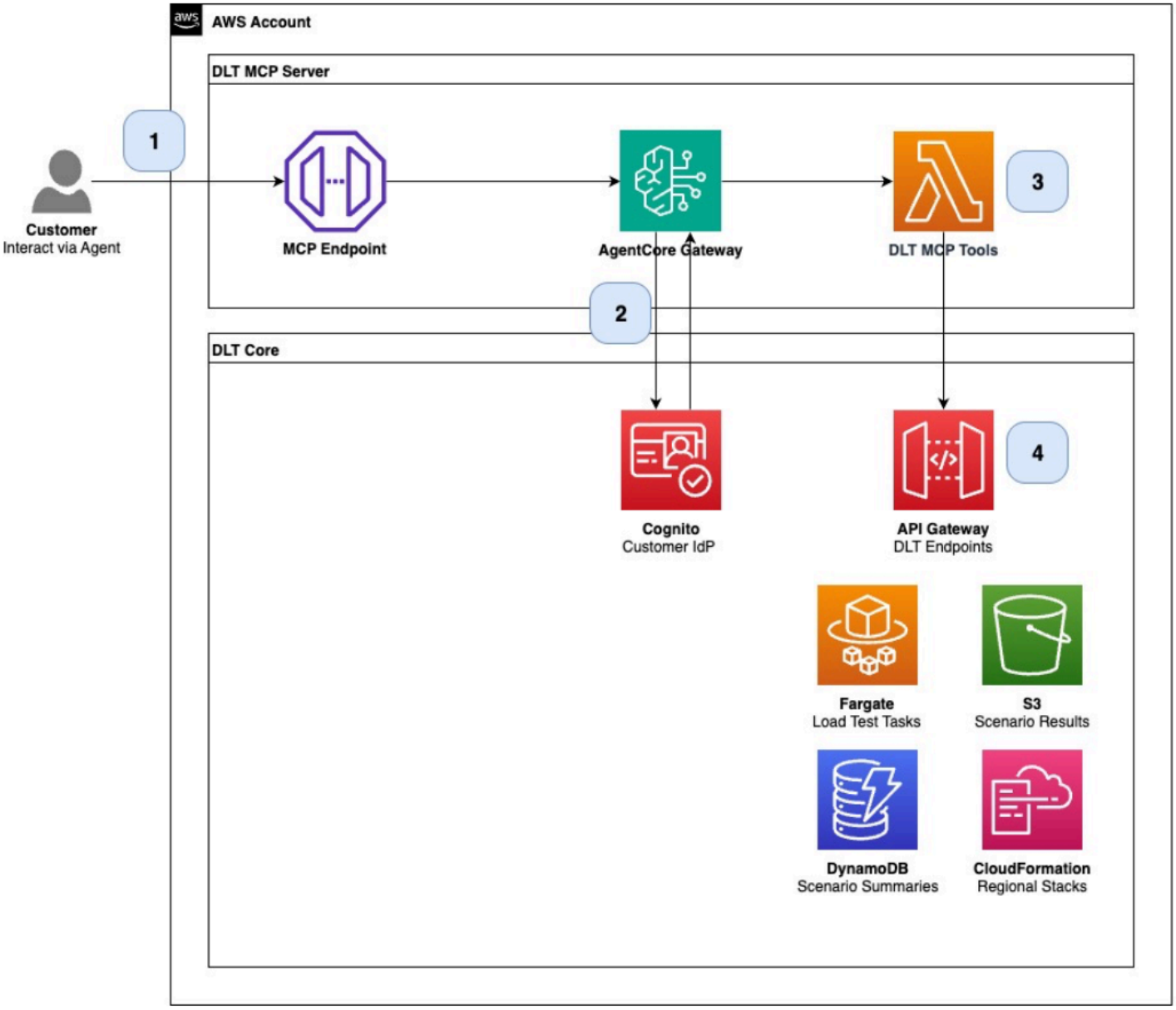
に 10 タスクに制限されています。タスク数が 10 を超える場合は、タスク定義はすべてのワーカータスクが開始されるまで複数回実行されます。この関数は、results-parser AWS Lambda 関数で現在のテストを区別するためのプレフィックスも生成します。

7. task-status-checker AWS Lambda 関数は、すべての Amazon ECS ワーカータスクが同じテスト ID を使用して実行しているかどうかを確認します。タスクがまだプロビジョニング中の場合は、1 分間待ってから再度確認します。すべての Amazon ECS タスクが実行されると、テスト ID、タスク数、テストタイプ、すべてのタスク ID とプレフィックスを返して、task-runner 関数に引き渡されます。
8. task-runner AWS Lambda 関数が再び実行し、今回はリーダーノードとして動作する単一の Amazon ECS タスクを起動します。この ECS タスクは、テストを同時に開始するために、各ワーカータスクにテスト開始メッセージを送信します。
9. task-status-checker AWS Lambda 関数は、Amazon ECS タスクが同じテスト ID を使用して実行しているかどうかを再確認します。タスクがまだ実行中の場合は、1 分間待ってから再度確認します。実行中の Amazon ECS タスクがなくなると、テスト ID、タスク数、テストタイプ、プレフィックスを返します。
10. task-runner AWS Lambda 関数が AWS Fargate クラスターで Amazon ECS タスクを実行すると、各タスクが Amazon S3 からテスト設定をダウンロードしてテストを開始します。
11. テストが実行されると、各タスクの平均応答時間、同時実行ユーザー数、成功したリクエストの数、失敗したリクエストの数が Amazon CloudWatch に記録されるため、CloudWatch ダッシュボードで確認できます。
12. テストにライブデータを含めた場合、ソリューションはサブスクリプションフィルタを使用して CloudWatch のリアルタイムテスト結果をフィルタリングします。その後、ソリューションはデータを Lambda 関数に引き渡します。
13. Lambda 関数は、その後、受信したデータを構造化して AWS IoT Core トピックに公開します。
14. ウェブコンソールはテスト用の AWS IoT Core トピックにサブスクライブして、トピックに公開されたデータを受け取り、テスト実行中のリアルタイムデータをグラフ化します。
15. テストが完了すると、コンテナイメージは、詳細レポートを XML ファイルとして Amazon S3 にエクスポートします。各ファイルには、ファイル名の UUID が割り当てられます。例: s3://dlte-bucket/test-scenarios/<$TEST_ID>/results/<$UUID>.json
16. XML ファイルが Amazon S3 にアップロードされると、results-parser AWS Lambda 関数は、プレフィックスで始まる XML ファイルの結果を読み取り、すべての結果を解析して 1 つの要約された結果に集約します。
17. results-parser AWS Lambda 関数は、集計結果を Amazon DynamoDB テーブルに書き込みます。

MCP サーバーワークフロー (オプション)

オプションの MCP サーバー 統合をデプロイすると、AI エージェントは次のワークフローを通じて負荷テストデータへのアクセスと分析ができます。

MCP サーバーのアーキテクチャ



1. 顧客とのやり取り - 顧客は、AWS AgentCore Gateway によってホストされる MCP エンドポイントを介して DLT の MCP とやり取りします。AI エージェントはこのエンドポイントに接続して、負荷テストデータへのアクセスをリクエストします。

2. 認可 - AgentCore Gateway は、ソリューションの Cognito ユーザープールアプリケーションクライアントに対する認可を処理します。ゲートウェイは、ユーザーの Cognito トークンを検証し、DLT MCP サーバーへのアクセス許可があることを確認します。認可されたユーザーには、読み取り専用オペレーションに制限されたエージェントツールアクセス許可が付与されます。
3. ツール仕様 – AgentCore Gateway は DLT MCP サーバー Lambda 関数に接続します。ツール仕様は、AI エージェントが負荷テストデータとやり取りするために使用できるツールを定義します。
4. 読み取り専用 API アクセス – Lambda 関数は、既存の DLT API ゲートウェイエンドポイントを介した読み取り専用 API アクセスに限定されます。関数は 4 つのプライマリオペレーションを提供します。
   - シナリオの一覧表示 - DynamoDB シナリオテーブルからテストシナリオのリストを取得します
   - シナリオテスト結果の取得 - DynamoDB と S3 から特定のシナリオの詳細なテスト結果にアクセスします
   - Fargate 負荷テストランナーの取得 – ECS クラスターでの Fargate タスクの実行に関するクエリ情報
   - 利用可能なリージョンスタックの取得 – CloudFormation からデプロイされたリージョンインフラストラクチャに関する情報を取得します

MCP サーバー統合は、既存の DLT インフラストラクチャ (API ゲートウェイ、Cognito、DynamoDB、S3) を活用して、AI を活用した分析とインサイトのためのテストデータへの安全な読み取り専用アクセスを提供します。

設計上の考慮事項

このセクションでは、サポートされているアプリケーション、テストタイプ、スケジュールオプション、デプロイに関する考慮事項など、AWS での分散負荷テストソリューションの設計に関する重要な決定事項と設定オプションについて説明します。

サポートされているアプリケーション

このソリューションは、AWS アカウントからアプリケーションへのネットワーク接続があれば、クラウドベースのアプリケーションとオンプレミスのアプリケーションのテストをサポートします。ソリューションでは、HTTP または HTTPS プロトコルを使用する API をサポートしています。

テストタイプ

AWS での分散負荷テストは、シンプルな HTTP エンドポイントテスト、JMeter、K6、Locust の複数のテストタイプをサポートしています。

シンプルな HTTP エンドポイントテスト

ウェブコンソールは、カスタムスクリプトを記述せずに HTTP または HTTPS エンドポイントをテストできる HTTP エンドポイント設定インターフェイスを提供しています。エンドポイント URL を定義し、ドロップダウンメニューから HTTP メソッド (GET、POST、PUT、DELETE など) を選択し、オプションでカスタムリクエストヘッダーと本文ペイロードを追加します。この設定により、アプリケーションに必要なカスタム認可トークン、コンテンツタイプ、またはその他の HTTP ヘッダーやリクエスト本文を使用して API をテストできます。

JMeter テスト

ウェブコンソールを使用してテストシナリオを作成する場合、JMeter のテストスクリプトをアップロードできます。このソリューションは、シナリオ S3 バケットにスクリプトをアップロードします。Amazon ECS タスクを実行すると、S3 から JMeter スクリプトをダウンロードしてテストを実行します。

> **Important**
>
> JMeter スクリプトは同時実行数 (仮想ユーザー)、トランザクションレート (TPS)、ランプアップ時間、およびその他のロードパラメータを定義する場合がありますが、ソリューションはテストの作成時に [Traffic Shape] 画面で指定した値でこれらの設定を上書きします。Traffic Shape 設定は、テスト実行のタスク数、同時実行数 (タスクあたりの仮想ユーザー数)、ランプアップ時間、保留期間を制御します。

JMeter 入力ファイルがある場合は、JMeter スクリプトと一緒に入力ファイルを zip 圧縮できます。テストシナリオを作成するときに、zip ファイルを選択できます。

プラグインを含めると、バンドルされた zip ファイルの /plugins サブディレクトリ内のすべての .jar ファイルが JMeter 拡張ディレクトリにコピーされ、負荷テストに利用できるようになります。

> **Note**
>
> JMeter 入力ファイルを JMeter スクリプトファイルに含める場合は、JMeter スクリプトファイルに入力ファイルの相対パスを含める必要があります。さらに、入力ファイルは必ず相

対パスである必要があります。例えば、JMeter の入力ファイルとスクリプトファイルが /home/user ディレクトリにあり、JMeter スクリプトファイル内の入力ファイルを参照する場合、入力ファイルのパスは ./INPUT_FILES である必要があります。代わりに /home/user/INPUT_FILES を使用すると、入力ファイルを見つけることができないため、テストは失敗します。

JMeter プラグインを含める場合は、.jar ファイルを zip ファイルのルート内の /plugins という名前のサブディレクトリにバンドルする必要があります。zip ファイルのルートを基準にして、jar ファイルへのパスは、./plugins/BUNDLED_PLUGIN.jar でなければなりません。

JMeter スクリプトの使用方法の詳細については、「JMeter User's Manual」を参照してください。

K6 テスト

このソリューションは、K6 フレームワークベースのテストをサポートしています。K6 は AGPL-3.0 ライセンスの下にリリースされています。このソリューションでは、新しい K6 テストを作成するときに、ライセンス確認メッセージが表示されます。アーカイブファイルに必要な入力ファイルと共に K6 テストファイルをアップロードできます。

Important

K6 スクリプトは同時実行数 (仮想ユーザー)、ステージ、しきい値、およびその他のロードパラメータを定義する場合がありますが、ソリューションはテストの作成時に [Traffic Shape] 画面で指定した値でこれらの設定を上書きします。Traffic Shape 設定は、テスト実行のタスク数、同時実行数 (タスクあたりの仮想ユーザー数)、ランプアップ時間、保留期間を制御します。

Locust テスト

このソリューションは、Locust フレームワークベースのテストをサポートしています。アーカイブファイルに必要な入力ファイルと共に Locust テストファイルをアップロードできます。

Important

Locust スクリプトは同時実行数 (ユーザー数)、スポーンレート、およびその他のロードパラメータを定義する場合がありますが、ソリューションはテストの作成時に [Traffic Shape] 画

面で指定した値でこれらの設定を上書きします。Traffic Shape 設定は、テスト実行のタスク数、同時実行数 (タスクあたりの仮想ユーザー数)、ランプアップ時間、保留期間を制御します。

テストのスケジューリング

このソリューションには、負荷テストの実行のために 3 つの実行タイミングオプションがあります。

- 今すぐ実行 - 作成後すぐに負荷テストを実行します
- 1 回実行 - 将来の特定の日時にテストを実行します
- スケジュールに従って実行 – cron 式を使用して定期的なテストを作成し、スケジュールを定義します

[1 回実行] を選択する場合、実行時間を 24 時間形式で指定し、負荷テストの実行を開始する実行日を指定します。

[スケジュールに従って実行] を選択する場合、cron 式を手動で入力するか、一般的な cron パターン (毎時間、特定の時刻に毎日、平日、毎月など) から選択できます。cron 式は、分、時間、日、月、曜日、年のフィールドを含むきめ細かなスケジュール形式を使用します。また、スケジュールされたテストが実行を停止するタイミングを定義する有効期限も指定する必要があります。スケジューリングの仕組みの詳細については、このガイドの「テストスケジューリングワークフロー」セクションを参照してください。

Note

- テスト期間: スケジューリング時には、テストの合計期間を考慮します。例えば、ランプアップ時間が 10 分、保留時間が 40 分のテストは完了までに約 80 分かかります。
- 最小間隔: スケジュールされたテストの間隔が想定されるテスト期間よりも長いことを確認します。例えば、テストに約 80 分かかる場合は、実行頻度を 3 時間は空けるようにスケジュールします。
- 時間単位の制限: 想定されるテスト時間が 1 時間未満であっても、1 時間差でテストをスケジュールすることはできません。

同時テスト

このソリューションは各テストの Amazon CloudWatch ダッシュボードを作成し、Amazon ECS クラスターで実行されるすべてのタスクの出力がリアルタイムで組み合わされて表示されます。CloudWatch ダッシュボードには、平均応答時間、同時ユーザーの数、成功したリクエストの数、失敗したリクエストの数が表示されます。ソリューションは各メトリクスを秒単位で集計し、ダッシュボードを 1 分毎に更新します。

ユーザー管理

初期設定時に、Amazon Cognito がソリューションのウェブコンソールへのアクセスを許可するために使用するユーザー名と E メールアドレスを指定します。コンソールには、ユーザー管理機能はありません。ユーザーを追加するには、Amazon Cognito コンソールを使用する必要があります。詳細については、「Amazon Cognito デベロッパーガイド」の「ユーザープール内のユーザーを管理する」を参照してください。

Amazon Cognito ユーザープールへの既存のユーザーの移行については、AWS ブログ「Approaches for migrating users to Amazon Cognito user pools」を参照してください。

リージョンデプロイ

このソリューションでは、特定の AWS リージョンでのみ利用可能な Amazon Cognito を使用します。そのため、このソリューションを Amazon Cognito が利用可能なリージョンにデプロイする必要があります。リージョン別の現在のサービス提供状況については、AWS リージョン別のサービスのリストを参照してください。

デプロイを計画する

このセクションでは、ソリューションをデプロイする前に確認する必要のある、コスト、セキュリティ、サポートされているリージョン、クォータ、その他の考慮事項について説明します。

Cost

このソリューションの実行中に使用した AWS サービスのコストは、お客様の負担となります。合計コストは、実行する負荷テストの数、それらのテストの所要時間、生成されるデータの量によって異なります。この改訂時点で、米国東部 (バージニア北部) リージョンでデフォルト設定を使用してこのソリューションを実行した場合の推定コストは、約 30.90 USD/月です。

次の表は、このソリューションをデフォルトパラメータで米国東部 (バージニア北部) リージョンに 1 か月間デプロイする場合のコスト内訳の例を示しています。

AWS のサービス	ディメンション	コスト [USD]
AWS Fargate	10 件のオンデマンドタスク (2 つの vCPU と 4 GB のメモリを使用) を 30 時間実行	29.62 USD
Amazon DynamoDB	1,000 件のオンデマンド書き込みキャパシティーユニット 1,000 件のオンデマンド読み込みキャパシティーユニット	0.0015 USD
AWS Lambda	1,000 件のリクエスト 合計所要時間 10 分	1.25 USD
AWS Step Functions	1,000 件の状態遷移	0.025 USD
合計		30.90 USD / 月

ソリューションリソースには Key=SolutionId と Value=SO0062 のタグが付けられます。タグキー SolutionId は、ドキュメントの activating-tags に従ってアクティブ化できます。タグがアクティブ化されたら、ドキュメントに従ってコストカテゴリルールを作成し、コストカテゴリを作成できます。

ソリューションで発生したコストについては、コストカテゴリコンソールをモニタリングし、コストカテゴリ名を選択することで確認できます。

AWS Cost Explorer を使用して予算を作成することをお勧めします。これはコスト管理に役立ちます。料金は変更されることがあります。詳細については、このソリューションで使用される各 AWS サービスの料金ウェブページを確認してください。

> **Note**
>
> デフォルトのタスク設定では、タスクごとに 2 つの vCPU と 4 GB のメモリを使用します。負荷テストでこれらのリソースが必要がない場合は、コスト削減のために削減できます。逆に、より多くのタスクあたりの同時実行数をサポートするために、リソースを増やすことができます。詳細については、このガイドの「コンテナリソースを増加する」セクションを参照してください。

> **Note**
>
> このソリューションには、テストの実行時にライブデータを含めるオプションがあります。この機能を利用するには、追加の AWS Lambda 関数と AWS IoT Core トピックが必要で、追加料金が発生します。

MCP サーバーの追加コスト (オプション)

次の表は、米国東部 (バージニア北部) リージョンでの MCP サーバー統合と 1 か月間の料金のコストの内訳を示しています。

サービスコンポーネント	ディメンション	コスト [USD]
AgentCore Gateway - ツールインデックス	10 ツール × 0.02 USD / 100 ツール	0.002 USD
AgentCore Gateway - 検索 API	10,000 件のインタラクション × 0.025 USD/1,000 件	0.25 USD
AgentCore Gateway - API 呼び出し	50,000 回の呼び出し × 0.005 USD/1,000 回	0.25 USD

サービスコンポーネント	ディメンション	コスト [USD]
AWS Lambda 関数	使用量に基づく変数 (通常のワークロード)	5.00 ~ 20.00 USD
推定追加コストの合計:		5.50 ~ 20.50 USD/月

価格は変更されることがあります。AgentCore Gateway の料金の詳細については、「Amazon Bedrock Pricing」(AgentCore Gateway セクション) を参照してください。Lambda の料金については、「AWS Lambda Pricing」を参照してください。

セキュリティ

AWS インフラストラクチャでシステムを構築すると、お客様と AWS の間でセキュリティ上の責任が分担されます。この責任共有モデルにより、ホストオペレーティングシステムと仮想化レイヤーからサービスが運用されている施設の物理的なセキュリティに至るまでのコンポーネントを AWS が運用、管理、制御するため、お客様の運用上の負担を軽減するのに役立ちます。AWS セキュリティの詳細については、「AWS クラウドセキュリティ」を参照してください。

IAM ロール

AWS Identity and Access Management (IAM) ロールにより、AWS クラウドのサービスとユーザーに対してアクセスポリシーとアクセス許可を詳細に割り当てることができます。このソリューションでは、リージョン別のリソースを作成するために、ソリューションの AWS Lambda 関数にアクセス権を付与する IAM ロールを作成します。

Amazon CloudFront

このソリューションは、Amazon S3 バケットでホストされ、Amazon CloudFront で配布されるウェブ UI をデプロイします。レイテンシーを軽減し、セキュリティを向上させるために、このソリューションには、オリジンアクセスアイデンティティを持つ CloudFront ディストリビューションが含まれています。オリジンアクセスアイデンティティは、ソリューションのウェブサイトのバケットにあるコンテンツに、パブリックアクセスを提供する CloudFront ユーザーです。デフォルトでは、CloudFront ディストリビューションは TLS 1.2 を使用して最高レベルのセキュリティプロトコルを適用します。詳細については、「Amazon CloudFront デベロッパーガイド」の「Amazon S3 オリジンへのアクセスの制限」を参照してください。

CloudFront は追加のセキュリティ対策を有効にして、各ビューワーのレスポンスに HTTP セキュリティヘッダーを追加します。詳細については、「CloudFront レスポンスの HTTP ヘッダーを追加または削除する」を参照してください。

このソリューションでは、デフォルトの CloudFront 証明書を使用します。サポートされる最小のセキュリティプロトコルは TLS v1.0 です。TLS v1.2 または TLS v1.3 の使用を必ず適用するには、デフォルトの CloudFront 証明書の代わりにカスタム SSL 証明書を使用する必要があります。詳細については、「SSL/TLS 証明書を使用するように CloudFront ディストリビューションを設定する方法を教えてください」を参照してください。

Amazon API Gateway

このソリューションは、エッジ最適化 Amazon API Gateway エンドポイントをデプロイして、カスタムドメインではなくデフォルトの API ゲートウェイエンドポイントを使用して負荷テストの機能用の RESTful API を提供します。デフォルトのエンドポイントを使用するエッジ最適化 API については、API ゲートウェイは TLS-1-0 セキュリティポリシーを使用します。詳細については、「Amazon API Gateway デベロッパーガイド」の「REST API の操作」を参照してください。

このソリューションでは、デフォルトの API ゲートウェイの証明書を使用します。サポートされる最小のセキュリティプロトコルは TLS v1.0 です。TLS v1.2 または TLS v1.3 の使用を必ず適用するには、デフォルトの API ゲートウェイの証明書の代わりに SSL 証明書のあるカスタムドメインを使用する必要があります。詳細については、「REST API のカスタムドメイン名を設定する」を参照してください。

AWS Fargate セキュリティグループ

デフォルトで、このソリューションは、AWS Fargate セキュリティグループのアウトバウンドルールを公開します。AWS Fargate がどこにでもトラフィックを送信しないようにするには、アウトバウンドルールを特定の Classless Inter-Domain Routing (CIDR) に変更します。

このセキュリティグループには、同じセキュリティグループに属する任意の送信元へのポート 50,000 のローカルトラフィックを許可するインバウンドルールも含まれています。これは、コンテナが互いに通信できるようにするために使用されます。

Amazon VPC

VPC: Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) は、AWS クラウド内の論理的に隔離されたプライベートネットワークを提供します。

デプロイ中に AWS CloudFormation パラメータで独自の VPC を指定できます。VPC は負荷を生成する ECS タスクでのみ使用されます。ウェブコンソールと API はこの VPC 内にデプロイされません。既存の VPC を指定しない場合、ソリューションは必要なネットワーク設定で新しい VPC を作成します。既存の VPC を使用する場合は、負荷テストタスクを正常に実行するために、次の要件を満たしている必要があります。

VPC の要件

AWS での分散負荷テストで使用する VPC の最小要件を以下に示します。

- VPC には少なくとも 2 つの AZ が含まれている必要があります
- VPC には、それぞれ個別の AZ に少なくとも 2 つのサブネットが含まれている必要があります
- VPC サブネットはパブリックでもプライベートでもかまいませんが、同じ設定 (両方ともパブリック、または両方ともプライベート) を使用する必要があります
- VPC は、ECR、CloudWatch Logs、S3、および IoT Core のエンドポイントへのアクセスを提供する必要があります。
- VPC は、負荷テストの対象となるサービス (複数可) へのアクセスを提供する必要があります。

> **Note**
>
> これらの基準を満たす VPC がない場合は、VPC ウィザードを使用して VPC をすばやく作成できます。詳細については、「Create a VPC (VPC を作成する)」を参照してください。

パブリックサブネットは、以下を含めることでこれらの要件を満たすことができます。

- VPC にアタッチされたインターネットゲートウェイ
- インターネットゲートウェイへのルート (0.0.0.0/0)

プライベートサブネットは、以下に説明するように、NAT ゲートウェイまたは VPC エンドポイントを使用してこれらの要件を満たすことができます。

オプション 1: NAT ゲートウェイ

- プライベートサブネットを持つ各 AZ に NAT ゲートウェイをデプロイする
- インターネット行きのトラフィック (0.0.0.0/0) を NAT ゲートウェイ経由でルーティングするようにルートテーブルを設定する

オプション 2: VPC エンドポイント

VPC に次の VPC エンドポイントを作成します。

- Amazon ECR API エンドポイント: `com.amazonaws.<region>.ecr.api`
- Amazon ECR DKR エンドポイント: `com.amazonaws.<region>.ecr.dkr`
- Amazon CloudWatch Logs エンドポイント: `com.amazonaws.<region>.logs`
- Amazon S3 ゲートウェイエンドポイント: `com.amazonaws.<region>.s3`
- AWS IoT Core エンドポイント (ライブデータチャートを使用する場合は必須) `com.amazonaws.<region>.iot.data`

他の VPC 設定も機能する可能性があります。

> **Important**
>
> 各 VPC エンドポイントインターフェイスにアタッチされたセキュリティグループは、ECS タスクセキュリティグループからのポート 443 でのインバウンド TCP トラフィックを許可する必要があります。

セキュリティグループの設定

デプロイ中、ソリューションは VPC 内にセキュリティグループを作成し、ECS クラスター内のタスクで次のトラフィックを許可します。

- すべてのアウトバウンドトラフィック
- ワーカータスクとリーダータスク間の調整を容易にするために、同じセキュリティグループ内の他のタスクからのポート 50000 のインバウンドトラフィック。

ネットワーク負荷テスト

このソリューションは、「ネットワーク負荷テスト」ポリシーに従って使用していただく必要があります。このポリシーの対象となる状況は、Amazon EC2 インスタンスから他の口ケーション (別の Amazon EC2 インスタンス、AWS のプロパティ/サービス、外部エンドポイントなど) に対して直接、ボリュームの大きなネットワークテストの実行を計画する場合などです。これらのテストは、ストレステスト、負荷テスト、ゲームデイテストと呼ばれることがあります。ほとんどのお客様による

テストはこのポリシーに該当しませんが、合計で 1 分超、1 Gbps (10 億ビット/秒) 超または 1 Gpps (10 億パケット/秒) 超のトラフィック生成が予想される場合は、このポリシーを参照してください。

パブリックユーザーインターフェイスへのアクセス制限

IAM と Amazon Cognito によって提供される認証と認可のメカニズム以外に、パブリックユーザーインターフェイスへのアクセスを制限するには、AWS WAF (ウェブアプリケーションファイアウォール) セキュリティオートメーションソリューションを使用します。

このソリューションは、一般的なウェブベースの攻撃をフィルタリングする一連の AWS WAF ルールを自動的にデプロイします。ユーザーは、AWS WAF のウェブアクセスコントロールリスト (ウェブ ACL) に含まれるルールを定義している事前設定済みの保護機能から選択できます。

MCP サーバーのセキュリティ (オプション)

オプションの MCP サーバー統合をデプロイする場合、ソリューションは AWS AgentCore Gateway を使用して AI エージェントの負荷テストデータへの安全なアクセスを提供します。AgentCore Gateway は、リクエストごとに Amazon Cognito 認証トークンを検証し、認可されたユーザーのみが MCP サーバーにアクセスできることを確認します。MCP サーバー Lambda 関数は読み取り専用アクセスパターンを実装し、AI エージェントがテスト設定や結果を変更できないようにします。すべての MCP サーバーインタラクションは、ウェブコンソールと同じアクセス許可の境界とアクセスコントロールを使用します。

サポートしている AWS リージョン

このソリューションでは Amazon Cognito サービスを使用していますが、現在すべての AWS リージョンで利用できるわけではありません。リージョン別の AWS サービスの最新情報については、AWS リージョン別のサービスのリストを参照してください。

AWS での分散負荷テストは、次の AWS リージョンで利用できます。

リージョン名	
米国東部 (オハイオ)	アジアパシフィック (東京)
米国東部 (バージニア北部)	カナダ (中部)
米国西部 (北カリフォルニア)	欧州 (フランクフルト)

リージョン名	
米国西部 (オレゴン)	欧州 (アイルランド)
アジアパシフィック (ムンバイ)	欧州 (ロンドン)
アジアパシフィック (ソウル)	欧州 (パリ)
アジアパシフィック (シンガポール)	欧州 (ストックホルム)
アジアパシフィック (シドニー)	南米 (サンパウロ)

MCP サーバーでサポートされている AWS リージョン (オプション)

オプションの MCP サーバー統合をデプロイする場合、AWS AgentCore Gateway が利用可能な AWS リージョンにソリューションをデプロイする必要があります。MCP サーバー機能は、以下の AWS リージョンでのみ利用可能です。

リージョン名	リージョンコード
米国東部 (バージニア北部)	us–east–1
米国西部 (オレゴン)	us-west-2
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
アジアパシフィック (東京)	ap-northeast-1
欧州 (フランクフルト)	eu-central-1
欧州 (アイルランド)	eu-west-1
欧州 (ロンドン)	eu-west-2
欧州 (パリ)	eu-west-3

リージョン別の AWS AgentCore Gateway の最新の可用性については、「AWS AgentCore Gateway デベロッパーガイド」の「AWS AgentCore Gateway のエンドポイントとクォータ」を参照してください。

クォータ

サービスクォータ (制限とも呼ばれます) は、AWS アカウントのサービスリソースまたはオペレーションの最大数です。

このソリューション内の AWS サービスのクォータ

このソリューションに実装されている各サービスに十分なクォータがあることを確認してください。詳細については、「AWS のサービスクォータ」を参照してください。

次のリンクを使用して、そのサービスのページに移動します。ページを切り替えずに、ドキュメント内のすべての AWS サービスのサービスクォータを表示するには、この PDF の「Service endpoints and quotas」ページの情報を参照してください。

AWS CloudFormation のクォータ

ご使用の AWS アカウントには AWS CloudFormation のクォータがあり、このソリューションでスタックを起動する際に注意する必要があります。これらのクォータを理解することで、このソリューションを正常にデプロイできなくなるような制限エラーを回避できます。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation クォータを理解する」を参照してください。

負荷テストのクォータ

AWS Fargate 起動タイプを使用して Amazon ECS で実行できるタスクの最大数は、タスクの vCPU サイズによって異なります。AWS での分散負荷テストのデフォルトのタスクサイズは 2 vCPU です。現在のデフォルトクォータを確認するには、「Amazon ECS のサービスクォータ」を参照してください。現在のアカウントのクォータは、記載されているクォータと異なる場合があります。アカウント固有のクォータを確認するには、AWS マネジメントコンソールで Fargate オンデマンド vCPU リソース数のサービスクォータを確認します。増加をリクエストする方法については、「AWS 全般のリファレンスガイド」の「AWS サービスのクォータ」を参照してください。

Amazon Linux 2023 のコンテナイメージ (Taurus がインストール済み) では、タスクごとの同時接続数は制限されていませんが、これは無制限のユーザー数をサポートできるという意味ではありませ

ん。コンテナがテスト用に生成できる同時ユーザーの数を確定するには、このガイドの「ユーザー数を確定する」セクションを参照してください。

> **Note**
>
> デフォルト設定にもとづく同時ユーザーの推奨制限は 200 ユーザーです。

同時テスト

このソリューションは各テストの Amazon CloudWatch ダッシュボードを作成し、Amazon ECS クラスターで実行されるすべてのタスクの出力がリアルタイムで組み合わされて表示されます。CloudWatch ダッシュボードには、平均応答時間、同時ユーザーの数、成功したリクエストの数、失敗したリクエストの数が表示されます。ソリューションは各メトリクスを秒単位で集計し、ダッシュボードを 1 分毎に更新します。

Amazon EC2 テストポリシー

ネットワークトラフィックが 1 Gbps 未満にとどまる場合、このソリューションを使用して負荷テストを実行するのに AWS から承認を得る必要はありません。テストで 1 Gbps を超える場合は、AWS にお問い合わせください。詳細については、「Amazon EC2 テストポリシー」を参照してください。

Amazon CloudFront の負荷テストポリシー

CloudFront エンドポイントの負荷テストを計画している場合は、「Amazon CloudFront 開発者ガイド」の「CloudFront の負荷テスト」を参照してください。また、トラフィックを複数のタスクとリージョンに分散させることをお勧めします。負荷テストには、少なくとも 30 分のランプアップ時間を確保してください。1 秒あたり 500,000 件を超えるリクエストを送信したり、300 Gbps を超えるデータを要求したりする負荷テストでは、まずトラフィック送信の事前承認を得ることをお勧めします。CloudFront は、CloudFront サービスの可用性に影響を与える未承認の負荷テストトラフィックをスロットリングする可能性があります。

デプロイ後のソリューションのモニタリング

ソリューションをデプロイした後は、Amazon CloudWatch アラームとメトリクスを使用してソリューションのリソースを継続的にモニタリングすることをお勧めします。

CloudWatch アラームの設定

[CloudWatch アラーム] を設定して、主要なメトリクスをモニタリングし、しきい値を超えたときに通知を受け取ることができます。次のリソースのアラームの設定を検討してください。

Amazon CloudFront ディストリビューションのメトリクス

CloudFront ディストリビューションのパフォーマンスとエラーをモニタリングします。詳細については、「Amazon CloudFront デベロッパーガイド」の「CloudFront ディストリビューションメトリクスの値」を参照してください。

Amazon API Gateway のメトリクス

API リクエストレート、レイテンシー、エラーをモニタリングします。詳細については、「Amazon API Gateway デベロッパーガイド」の「Amazon API Gateway のディメンションとメトリクス」を参照してください。

AWS Lambda 関数のメトリクス

ソリューションのマイクロサービスの Lambda 関数の呼び出し、期間、エラー、スロットリングをモニタリングします。

Amazon ECS および AWS Fargate メトリクス

負荷テスト中にタスクの CPU とメモリの使用率をモニタリングして、適切なリソースを確保します。

Amazon DynamoDB のメトリクス

読み取りおよび書き込みキャパシティの消費量、スロットリングされたリクエスト、レイテンシーをモニタリングします。

エキスパートと連携する

AWS での分散負荷テストの AWS Countdown Premium 短期エンゲージメント

AWS エンジニアは、パフォーマンステストの基礎、スクリプト開発、結果分析に関するエキスパートからのガイダンスを提供します。今すぐサインアップ。

概要

AWS Countdown Premium (CDP) 短期エンゲージメントは、大規模なパフォーマンステストを実施する組織にエキスパートからのガイダンスを提供します。AWS エンジニアは、コラボレーション「Do It Yourself」モデルを通じて、チームが実行責任を維持しながら、戦略的監視と技術的な専門知識を提供します。エキスパート AWS エンジニアは、サインアップ後 1 週間以内に利用でき、長期契約は必要ありません。

サービスモデル

CDP エンジニアはチームと連携して、パフォーマンステストの実装全体のガイダンスと監視を提供します。このハンドオフアプローチにより、内部機能を構築しながら専門的な指示を受けることができます。このサービスは、AWS での分散負荷テストを効果的に実装するために AWS の専門的な知識を必要とする既存のテスト機能を持つ組織に最適です。

CDP エンジニアが提供するもの

CDP エンジニアは、パフォーマンステストの基礎と AWS での分散負荷テストのアーキテクチャについて説明します。JMeter、K6、Locust スクリプト構造とテストスクリプト開発に関するガイダンスを提供し、CloudFormation テンプレートのデプロイを支援し、パフォーマンス最適化の推奨事項を使用してテスト結果を評価します。サポートには、リソース使用率分析、ベストプラクティスの調整、初期設定から結果分析までのエンドツーエンドのガイダンスが含まれ、チームに知識を伝達できます。

お客様の責任

チームは、アプリケーションレベルの設定、テストスクリプトの開発、テストシナリオの検証を処理します。パフォーマンステストイベントの前、最中、後のすべてのテストアクティビティを含め、実際のテストの実行と運用に対する責任はお客様が負います。

主な利点

CDP 短期エンゲージメントは、エキスパートによる監視、ワークロード固有のコンテキストガイダンス、パフォーマンス最適化の推奨事項、迅速な問題解決、ベストプラクティスの調整、包括的なサポートを通じてリスクを軽減し、チームの所有権と能力開発を維持します。

対応アーキテクチャ

AWS での分散負荷テストは、AWS での分散負荷テストソリューションを活用して、ウェブアプリケーション、API、マイクロサービス、サーバーレスアーキテクチャの大規模なテストをサポートします。テスト機能は、これらの一般的なユースケースをはるかに超えて拡張され、データベー

ス、TCP/UDP プロトコル、LDAP ディレクトリ、SMTP メールサーバー、および負荷時のパフォーマンス検証を必要とする他の多くのシステムやプロトコルが含まれます。

開始方法

AWS での分散負荷テストの CDP 短期エンゲージメントに関心のある組織は、こちらの AWS ウェブサイトから直接サインアップし、フォーカスエリアの「ユースケースの実装」を選択できます。

対象外

CDP は、カスタムテストスクリプト開発 (ガイダンスのみ) を提供したり、テスト実行オペレーションを管理したり、カスタムのハンズオンラボやワークショップを作成したりしません。オンサイトサポートも対象外です。

ソリューションをデプロイする

AWS Launch Wizard は、このソリューションで推奨されるデプロイ方法です。次の機能があります。

- 各ステップで詳細なヘルプパネルを備えたガイド付き設定エクスペリエンス
- すべてのデプロイの正常性をモニタリングするための一元化されたページ
- デプロイまたはアップグレードに利用できるソリューションの最新バージョンがある場合の表示

または、AWS CloudFormation テンプレートを使用してソリューションを直接デプロイすることもできます。

デプロイプロセスの概要

ソリューションをデプロイする前に、このガイドで前述したコスト、アーキテクチャ、セキュリティ、その他の考慮事項を確認してください。

デプロイ時間: メインスタックの場合は約 15 分、追加のリージョンごとに 5 分

> **Note**
>
> このソリューションには、AWS へのデータ収集メトリクスが含まれています。このデータを使用して、ユーザーがこのソリューションおよび関連サービスや製品をどのように使用しているかをよりよく理解します。AWS は、このアンケートを通じて収集されたデータを所有します。データ収集には、AWS プライバシー通知が適用されます。

> **Note**
>
> このソリューションの実行中に使用した AWS サービスのコストは、お客様の負担となります。詳細は、このガイドの「コスト」セクション、およびこのソリューションで使用する各 AWS サービスの料金ページを参照してください。

AWS Launch Wizard を使用してデプロイする

このソリューションには、AWS Launch Wizard を使用したガイド付きデプロイプロセスが含まれています。AWS ソリューションでの分散負荷テストソリューションをアカウントにデプロイするには、次のステップに従ってください。

1. AWS マネジメントコンソールにサインインし、以下のボタンを選択してデプロイプロセスを開始します。

2. ソリューションで使用可能なデプロイパターンが複数ある場合は、ユースケースに最も適したものを選択してください。
3. デプロイするバージョンを選択します。最新バージョンをお勧めします。
4. [起動デプロイウィザード] ボタンをクリックします。

次に、一連のステップに従って、ソリューションをデプロイするために必要な情報を収集します。必要なリソースのプロビジョニングには約 15 分かかります。

[デプロイリスト] からデプロイを選択して、そのステータスを表示します。

AWS CloudFormation を使用してデプロイする

このソリューションは、AWS CloudFormation テンプレートとスタックを使用してデプロイを自動化します。CloudFormation テンプレートは、このソリューションに含まれる AWS リソースとそのプロパティを指定します。CloudFormation スタックは、テンプレートに記述されているリソースをプロビジョニングします。

AWS CloudFormation テンプレート

このソリューションの CloudFormation テンプレートは、デプロイする前にダウンロードできます。このソリューションでは、AWS CloudFormation を使用して、AWS での分散負荷テストのデプロイを自動化します。このソリューションには次の AWS CloudFormation テンプレートが含まれており、デプロイ前にダウンロード可能です。

View template

distributed-load-testing-on-aws.template - このテンプレートを使用して、ソリューションとすべての関連コンポーネントを起動します。デフォルト設定では、「このソリューションの AWS のサービス」セクションに記載しているコアとサポートのサービスがデプロイされますが、特定のニーズに合わせてテンプレートをカスタマイズできます。

> **Note**
>
> AWS CloudFormation のリソースは、AWS Cloud Development Kit (AWS CDK) のコンストラクトで作成されています。すでにこのソリューションをデプロイしている場合は、「ソリューションのアップデート」でアップデートの手順を参照してください。

スタックを起動する

AWS での分散負荷テストソリューションをアカウントにデプロイするには、次のステップに従ってください。この自動化された AWS CloudFormation テンプレートが、AWS での分散負荷テストをデプロイします。

1. AWS マネジメントコンソールにサインインして、 CloudFormation テンプレートを起動するボタンを選択します。

   Launch solution

   または、テンプレートをダウンロードして、独自にカスタマイズすることもできます。
2. テンプレートはデフォルトで米国東部 (バージニア北部) リージョンで起動されます。別の AWS リージョンでこのソリューションを起動するには、コンソールのナビゲーションバーのリージョンセレクターを使用します。

   > **Note**
   >
   > このソリューションでは Amazon Cognito を使用しており、現在特定の AWS リージョンでのみ利用可能です。そのため、このソリューションは Amazon Cognito が利用可能な AWS リージョンで起動する必要があります。リージョン別の現在のサービス提供状況については、AWS リージョン別のサービスのリストを参照してください。

3. [スタックの作成] ページで、正しいテンプレート URL が [Amazon S3 URL] テキストボックスに表示されていることを確認し、[次へ] を選択します。
4. [スタックの詳細を指定] ページで、ソリューションのスタックに名前を割り当てます。
5. [パラメータ] で、テンプレートのパラメータを確認し、必要に応じて変更します。このソリューションでは、次のデフォルト値を使用します。

パラメータ	デフォルト	説明
Administrator Name	<入力必須>	最初のソリューション管理者のユーザー名。
Administrator Email	*<####>*	管理者ユーザーの E メールアドレス。起動後、コンソールログイン手順が記載された E メールがこのアドレスに送信されます。
Existing VPC ID	<オプション入力>	使用したい VPC があり、既に作成されている場合は、スタックをデプロイしたのと同じリージョンにある既存 VPC の ID を入力してください。例: vpc-1a2b3c4d5e6f
First existing subnet	<オプション入力>	既存 VPC 内の 1 つ目のサブネットの ID。このサブネットには、テストを実行するためにコンテナイメージを取り込むためのインターネットへのルートが必要です。例: subnet-7h8i9j0k
Second existing subnet	<オプション入力>	既存 VPC 内の 2 つ目のサブネットの ID。このサブネットには、テストを実行するためにコンテナイメージを取り込むためのインターネット

パラメータ	デフォルト	説明
		へのルートが必要です。例: subnet-1x2y3z
ソリューションで VPC を作成するための有効な CIDR ブロックを提供する	192.168.0.0/16	既存の VPC を使用している場合は、このパラメータを空白のままにしておくことができます。
ソリューションが VPC を作成するために、サブネット A の有効な CIDR ブロックを提供する	192.168.0.0/20	AWS Fargate VPC のサブネット A の CIDR ブロック
ソリューションが VPC を作成するために、サブネット B の有効な CIDR ブロックを提供する	192.168.16.0/20	AWS Fargate VPC のサブネット B の CIDR ブロック
Fargate タスクのアウトバウンドトラフィックを許可するための CIDR ブロックを提供する	0.0.0.0/0	Amazon ECS コンテナのアウトバウンドアクセスを制限する CIDR ブロック。
コンテナイメージを自動更新する	No	次のマイナーリリースまで、最新の安全なイメージを自動的に使用します。[No] を選択すると、セキュリティ更新なしで、最初にリリースされたイメージが取得されます。
オプションの MCP サーバーをデプロイする	No	AgentCore Gateway を使用してオプションのリモート MCP サーバーをデプロイし、AI アプリケーションを AWS での分散負荷テストに接続します。

6. [次へ] を選択します。
7. [スタックオプションの設定] ページで、[次へ] を選択します。
8. [レビュー] ページで、設定を確認して確定します。テンプレートが AWS Identity and Access Management (IAM) リソースを作成することを承認するボックスを必ずオンにします。
9. [スタックの作成] を選択してスタックをデプロイします。

AWS CloudFormation コンソールの [ステータス] 列でスタックのステータスを確認できます。約 15 分で CREATE_COMPLETE ステータスが表示されます。

> **Note**
>
> 主要な AWS Lambda 関数に加えて、このソリューションには custom-resource Lambda 関数が含まれています。この関数は、初期設定時かリソースの更新または削除時にのみ実行されます。

このソリューションを実行すると、custom-resource Lambda 関数は非アクティブになります。ただし、関連付けられたリソースを管理する必要があるため、この関数は削除しないでください。

マルチリージョンデプロイ

デプロイ時間: リージョンごとに約 5 分

複数のリージョンでテストの実行が可能です。

分散負荷テストソリューションをデプロイすると、シナリオの S3 バケットにリージョン CloudFormation テンプレートが作成されます。このテンプレートの URL は、メインスタックの CloudFormation 出力の「RegionalCFTemplate」キーの下に表示されます。

マルチリージョンテストを実行するには、テストを実行する各リージョンにリージョン CloudFormation テンプレートをデプロイする必要があります。

> **Note**
>
> 各 AWS アカウントは、リージョンごとに 1 つのリージョンスタックのみを使用できます。また、リージョンスタックは、メインスタックと同じリージョンでは使用できません。

リージョン用のテンプレートは、次のようにインストールできます。

1. ソリューションのウェブコンソールで、左側のメニューにある [ダッシュボード] に移動します。
2. クリップボードのアイコンを使用して Amazon S3 にある CloudFormation テンプレートのリンクをコピーします。
3. AWS CloudFormation コンソールにサインインして、正しいリージョンを選択します。
4. [スタックの作成] ページで、正しいテンプレート URL が [Amazon S3 URL] テキストボックスに表示されていることを確認し、[次へ] を選択します。
5. [スタックの詳細を指定] ページで、ソリューションのスタックに名前を割り当てます。
6. [パラメータ] で、テンプレートのパラメータを確認し、必要に応じて変更します。このソリューションでは、次のデフォルト値を使用します。

パラメータ	デフォルト	説明
Existing VPC ID	<オプション入力>	使用したい VPC があり、既に作成されている場合は、スタックをデプロイしたのと同じリージョンにある既存 VPC の ID を入力してください。例: vpc-1a2b3c4d5e6f
First existing subnet	<オプション入力>	既存 VPC 内の 1 つ目のサブネットの ID。このサブネットには、テストを実行するためにコンテナイメージを取り込むためのインターネットへのルートが必要です。例: subnet-7h8i9j0k
Second existing subnet	<オプション入力>	既存 VPC 内の 2 つ目のサブネットの ID。このサブネットには、テストを実行するためにコンテナイメージを取り込むためのインターネットへのルートが必要です。例: subnet-1x2y3z

パラメータ	デフォルト	説明
ソリューションで VPC を作成するための有効な CIDR ブロックを提供する	192.168.0.0/16	既存 VPC の値を指定しない場合は、ソリューションによって作成された Amazon VPC の CIDR ブロックには AWS Fargate の IP アドレスが使用されます。
Fargate タスクのアウトバウンドトラフィックを許可するための CIDR ブロックを提供する	0.0.0.0/0	Amazon ECS コンテナのアウトバウンドアクセスを制限する CIDR ブロック。

7. [次へ] を選択します。
8. [スタックオプションの設定] ページで、[次へ] を選択します。
9. [レビュー] ページで、設定を確認して確定します。テンプレートが AWS Identity and Access Management (IAM) リソースを作成することを承認するボックスに必ずチェックを入れてください。
10. [スタックの作成] を選択してスタックをデプロイします。

    AWS CloudFormation コンソールの [ステータス] 列でスタックのステータスを確認できます。約 5 分で CREATE_COMPLETE ステータスが表示されます。

リージョンが正常にデプロイされると、ウェブコンソールに表示されます。テストを作成すると、使用可能なすべてのリージョンが [ダッシュボード] と [シナリオの作成] に表示されます。シナリオ作成の Traffic Shape ステップで、テストにリージョンを追加できます。

ソリューションは、シナリオテーブルでデプロイされたリージョンごとに DynamoDB の項目を作成します。この項目には、そのリージョンのテストリソースに関する必要な情報が含まれます。ウェブコンソールでは、テスト結果をリージョンごとに並べ替えることができます。マルチリージョンテストのすべてのリージョンの集計結果を表示するには、Amazon CloudWatch メトリクスを使用します。テストが完了した後、テスト結果でグラフのソースコードを確認できます。

> **Note**
>
> リージョン用のスタックは、ウェブコンソールなしで起動できます。Amazon S3 の scenarios バケットにあるリージョン用のテンプレートのリンクを取得して、必要なリー

ジョンでリージョン用のスタックを起動する際のソースとして使用してください。または、テンプレートをダウンロードして、必要なリージョンのソースとしてアップロードすることもできます。

ソリューションを更新する

ソリューションを更新すると、デプロイに最新の機能、セキュリティパッチ、バグ修正が適用されます。最新バージョンに更新するには、元のデプロイ方法 (「AWS Launch Wizard」または「AWS CloudFormation」) に基づく適切なセクションを参照してください。

> **Important**
>
> 更新する前に、現在実行中の負荷テストがないことを確認してください。更新プロセスにより、ソリューションの可用性が一時的に中断される場合があります。

AWS Launch Wizard を使用した更新

コンソールでは、デプロイバージョンのドロップダウンに利用可能な最新バージョンのソリューションが自動的に表示されます。ソリューションを以前にデプロイ済みの場合は、この手順に従ってデプロイを最新バージョンに更新します。

1. Launch Wizard Deployments に移動します。
2. 更新するデプロイを選択します。
3. [アクション] を選択し、[デプロイバージョンを更新] を選択します。
4. 利用可能な [デプロイバージョン] から最新バージョンを選択します。
5. 設定を確認します。
6. 各ステップで必要な変更を加えます。
7. 更新を確認します。

AWS CloudFormation を使用して更新する

ソリューションを以前にデプロイ済みの場合は、この手順に従ってソリューションの CloudFormation スタックを最新バージョンに更新します。

1. CloudFormation コンソールにサインインし、既存の CloudFormation スタックを選択して、[スタックを更新] を選択します。
2. [直接更新を実行] を選択します。

3. [既存のテンプレートを置換] を選択します。
4. [テンプレートを指定] で、以下を実行します。
   a. [Amazon S3 URL] を選択します。
   b. 最新のテンプレートのリンクをコピーします。
   c. [Amazon S3 URL] ボックスにリンクを貼り付けます。
   d. テンプレートの正しい URL が [Amazon S3 URL] テキストボックスに表示されていることを確認します。
   e. [次へ] を選択します。
   f. [次へ] をもう一度選択します。
5. [パラメータ] で、テンプレートのパラメータを確認し、必要に応じて変更します。パラメータの詳細については、「スタックを起動する」を参照してください。
6. [次へ] を選択します。
7. [スタックオプションの設定] ページで、[次へ] を選択します。
8. [レビュー] ページで、設定を確認して確定します。
9. テンプレートによって IAM のリソースが作成されることを承認するチェックボックスをオンにします。
10. [変更セットの表示] を選択して、変更を確認します。
11. [スタックの更新] を選択してスタックをデプロイします。

AWS CloudFormation コンソールの [ステータス] 列でスタックのステータスを確認できます。約 15 分で `UPDATE_COMPLETE` ステータスが表示されます。

> **Note**
>
> スタックのアップグレード後にブラウザからログインする際に Amazon Cognito 認証の問題が発生した場合は、ブラウザを更新し (Windows/Linux の場合は Ctrl+Shift+R、Mac の場合は Cmd+Shift+R)、キャッシュされたデータをクリアして再試行してください。

v3.3.0 より前のバージョンからの更新のトラブルシューティング

> **Note**
>
> このセクションは、v3.3.0 より前のバージョンからの更新にのみ適用されます。v3.3.0 以降から更新する場合は、AWS Launch Wizard または AWS CloudFormation を使用して標準の更新手順に従ってください。

1. distributed-load-testing-on-aws.template をダウンロードします。
2. テンプレートを開き、`Conditions:` に移動して `DLTCommonResourcesAppRegistryCondition` を探します。
3. 次のような結果が表示されます。

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

4. 2 番目の `true` の値を `false` に変更します。

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

5. カスタマイズされたテンプレートを使用して、「AWS CloudFormation を使用した更新」の手順に従ってスタックを更新します。
6. この更新により、アプリケーションレジストリ関連のリソースがスタックから削除され、更新は正常に完了します。
7. 最新のテンプレート URL を使用して、別のスタック更新を実行します。

リージョンスタックの更新

ソリューションを複数のリージョンにデプロイした場合は、各リージョンスタックを個別に更新する必要があります。テストインフラストラクチャをデプロイしたリージョン内にある各リージョンの CloudFormation スタックの標準更新手順に従います。

AWS Systems Manager Application Manager

ソリューションを更新すると、AWS Systems Manager Application Manager はソリューションとそのリソースにアプリケーションレベルのビューを提供します。Application Manager を使用すると、以下のことができます。

- リソース、スタックや AWS アカウントにデプロイされたリソースのコスト、ログを一元的にモニタリングします。
- デプロイステータス、CloudWatch アラーム、リソース設定、運用上の問題など、アプリケーションのコンテキストにおけるソリューションのリソースの運用データを表示します。

トラブルシューティング

既知の問題解決には、既知のエラーを軽減するための手順が記載されています。これらの手順で問題が解決しない場合は、「AWS サポートにお問い合わせる」に、このソリューションに関する AWS サポートのケースを開く方法が記載されています。

既知の問題解決

問題: 既存 VPC を使用していて、テストのステータスが Failed で失敗し、次のエラーメッセージが表示されます。

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

- 解決:

指定した VPC にサブネットが存在し、インターネットゲートウェイまたは NAT ゲートウェイのいずれかを使用するインターネットへのルートがあることを確認してください。AWS Fargate は、テストを正常に実行するために、パブリックリポジトリからコンテナイメージを取り込むためのアクセス権が必要です。

問題: テストの実行に時間がかかりすぎている、または、いつまで経っても実行されない

- 解決:

テストをキャンセルし、AWS Fargate をチェックしてすべてのタスクが停止していることを確認します。停止していない場合は、すべての Fargate タスクを手動で停止します。ご自身のアカウントでオンデマンドの Fargate タスクの制限をチェックして、必要な数のタスクが起動されていることを確認します。また、task-runner Lambda 関数用の CloudWatch の口グを確認して、Fargate タスクの起動時の障害を詳しく確認することもできます。実行中の Fargate のコンテナで何が起こっているかの詳細については、CloudWatch の ECS の口グを確認してください。

問題: テストが開始しているが未完了であるか、ECS タスクの状態が不明

- 解決:

ソリューションをデプロイしたアカウントで既存の VPC を指定するオプションを選択した場合は、テスト入力で指定したタスク数を開始するのに十分な空き IP アドレスが、ECS タスクで使用する

VPC にあることを確認してください。ECS タスク定義では、インターネットゲートウェイまたはインターネットへのルートを必要とする ECR イメージを使用して、ECS サービスが aws-solutions/distributed-load-testing-on-aws-load-tester からソリューション ECR イメージをダウンロードしてタスクをプロビジョニングできるようにします。VPC 内のすべてのサブネットがプライベートであるためにインターネットへのルートを指定できない場合は、ECR プルスルーキャッシュを使用してアカウントで ECR イメージをホストできます。新しい ECR イメージ URI でタスク定義を更新し、新しいリビジョンを作成します。タスク定義が更新されたら、DynamoDB テーブルのソリューション設定を更新して新しいリビジョンを使用する必要があります。DynamoDB テーブル名は、CloudFormation の [スタック出力] タブのキー ScenariosTable で確認できます。項目の属性 taskDefinition をキー testId と値 region-[SOLUTION-DEPLOYED-REGION] で更新します。

問題: テストで使用するエンドポイントはプライベートである必要があり、インターネットゲートウェイ経由では利用できない

- 解決:

インターネットゲートウェイ経由でアクセスできないプライベート API エンドポイントをテストする場合は、以下のアプローチを検討してください。

1. ネットワーク設定: ECS タスクで使用するサブネットルートテーブルが、テスト対象のプライベートエンドポイントの IP アドレス範囲へのルートで更新されていることを確認します。これにより、テストトラフィックは VPC 内のプライベートエンドポイントに到達できます。
2. DNS 解決: カスタムドメインの場合は、プライベートエンドポイントのドメイン名を解決するように VPC の DNS 設定を構成します。詳細な手順については、VPC DNS に関するドキュメントを参照してください。
3. VPC エンドポイント: AWS サービスをテストする場合は、VPC エンドポイント (AWS PrivateLink) を使用してプライベート接続を確立することを検討してください。例えば、プライベート API Gateway をテストするには、API Gateway の VPC エンドポイントを作成できます。プライベート API Gateway に関するドキュメントを参照してください。
4. VPC ピアリング: プライベートエンドポイントが別の VPC にある場合は、ソリューションのデプロイ先の VPC とプライベートエンドポイントがある VPC の間に VPC ピアリングを確立します。両方の VPC に適切なルートテーブルを設定します。VPC ピアリングに関するドキュメントを参照してください。
5. Transit Gateway: より複雑なネットワークシナリオで複数の VPC が関わる場合は、ソリューションの VPC とプライベートエンドポイントがある VPC 間のトラフィックをルーティングするため

に AWS Transit Gateway の使用を検討してください。Transit Gateway に関するドキュメントを参照してください。

6. セキュリティグループ: ECS タスクに関連付けられたセキュリティグループがプライベートエンドポイントへのアウトバウンドトラフィックを許可し、プライベートエンドポイントのセキュリティグループが ECS タスクからのインバウンドトラフィックを許可していることを確認します。

内部 Application Load Balancer または EC2 インスタンスをテストする場合は、VPC CIDR 範囲が重複せず、必要なルートがルートテーブルに設定されていることを確認します。

問題: テストは完了しているが、結果を UI で利用できない

- 解決:

テストは完了したが、結果を UI で利用できない場合は、テストを実行した ECS タスクから S3 バケットの結果ファイルを引き続き利用できます。これは、ソリューションの既知の制限です。現在のアーキテクチャでは、ソリューションは結果を解析する Lambda 関数を使用して複数の ECS タスクの結果を要約し、DynamoDB テーブルに 1 つの項目として保存します。DynamoDB テーブルの最大項目サイズは 400 KB に制限されています。テストスクリプトの複雑さ、同時実行数、使用しているタスクの数によっては、この制限に達します。エラーは、テストの失敗ではなく、結果を要約して CRUD オペレーションで DynamoDB テーブルに保存するプロセスの失敗を示しています。結果は、テストシナリオの S3 バケットで引き続き利用できます。

AWS サポートに問い合わせる

AWS ビジネスサポート+、AWS エンタープライズサポート、または Unified Operations をご利用の場合は、AWS サポートセンターを利用して、このソリューションに関するエキスパートのサポートを受けることができます。次のセクションで、その方法を説明します。

ケースを作成する

1. サポートセンターにサインインします。
2. [ケースを作成] を選択します。

どのようなサポートをご希望ですか?

1. [技術] を選択します。

2. サービスで、[ソリューション] を選択します。
3. [カテゴリ] で、[AWS での分散負荷テスト] を選択します。
4. [重要度] で、ユースケースに最も適したオプションを選択します。
5. [サービス]、[カテゴリ]、[重要度] を入力すると、インターフェイスに一般的なトラブルシューティングの質問へのリンクが表示されます。これらのリンクを使用しても問題を解決できない場合は、[次のステップ: 追加情報] を選択してください。

追加情報

1. [件名] に、質問または問題を要約したテキストを入力します。
2. [説明] で、この製品の名前と使用しているバージョン (例: AWS バージョン X.Y.Z での分散負荷テスト) を含めて、問題を詳しく説明します。
3. [ファイルを添付] を選択します。
4. AWS サポートがリクエストを処理するために必要な情報を添付します。

ケースの迅速な解決にご協力ください

1. 必要な情報を記入します。
2. [次のステップ: 今すぐ解決またはお問い合わせ] を選択します。

今すぐ解決またはお問い合わせ

1. [今すぐ解決] で解決策を確認します。
2. これらの解決策で問題を解決できない場合は、[お問い合わせ] を選択し、必要な情報を入力して [送信] を選択します。

ソリューションをアンインストールする

AWS での分散負荷テストソリューションは、AWS マネジメントコンソールまたは AWS コマンドラインインターフェイスを使用してアンインストールできます。このソリューションで作成されたコンソール、シナリオ、ログ記録用の Amazon Simple Storage Service (Amazon S3) バケットを手動で削除する必要があります。AWS ソリューション実装では、ユーザーがデータを保持する場合を考慮し、自動的には削除しません。

> **Note**
>
> リージョン用のスタックをデプロイした場合は、メインスタックを削除する前にリージョン用のスタックを削除する必要があります。

AWS マネジメントコンソールの使用

AWS CloudFormation

1. AWS CloudFormation コンソールにサインインします。
2. [スタック] ページで、このソリューションのインストールスタックを選択します。
3. [削除] を選択します。

AWS Launch Wizard

1. AWS Launch Wizard コンソールにサインインします。
2. Launch Wizard Deployments ページで、このソリューションのデプロイを選択します。
3. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。
4. 削除を確定します。

AWS コマンドラインインターフェイスの使用

AWS コマンドラインインターフェイス (AWS CLI) が環境で使用可能かどうかを判断します。インストール手順については、「AWS CLI ユーザーガイド」の「AWS コマンドラインインターフェイスとは」を参照してください。AWS CLI が使用可能なことを確認したら、次のコマンドを実行します。

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

Amazon S3 バケットの削除

このソリューションは、AWS CloudFormation スタックを削除することを決めた場合、ソリューションで作成した Amazon S3 バケット (オプトインリージョンへのデプロイ用) を保持して、偶発的なデータ損失を防ぐように設定されています。ソリューションをアンインストールした後に、データを保持する必要がない場合は、S3 バケットを手動で削除できます。Amazon S3 バケットを削除するには、次の手順に従います。

1. Amazon S3 コンソール にサインインします。
2. 左側のナビゲーションペインで、[バケット] を選択します。
3. [バケットを名前で検索] フィールドに、このソリューションのスタックの名前を入力します。
4. ソリューションの S3 バケットの 1 つを選択して、[空にする] を選択します。
5. テキスト入力フィールドに「完全に削除」と入力し、[空にする] を選択します。
6. 空にした S3 バケット名を選択し、[削除] を選択します。
7. テキスト入力フィールドに S3 バケット名を入力し、[バケットを削除] を選択します。

すべての S3 バケットを削除するまで、ステップ 4 ~ 7 を繰り返します。

AWS CLI を使用して S3 バケットを削除するには、次のコマンドを実行します。

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

ソリューションを使用する

このセクションでは、最初のテストシナリオの作成から詳細結果の分析まで、AWS での分散負荷テストソリューションを使用するための包括的なガイドを提供します。ワークフローには、テストシナリオの作成、テストの実行、テスト結果の調査が含まれます。

テストシナリオの作成

テストシナリオの作成の主なステップには、全般設定の構成、シナリオの定義、トラフィックパターンの形成、設定の確認の 4 つが含まれます。

ステップ 1: 全般設定

テスト名、説明、全般設定オプションなど、負荷テストの基本的なパラメータを設定します。

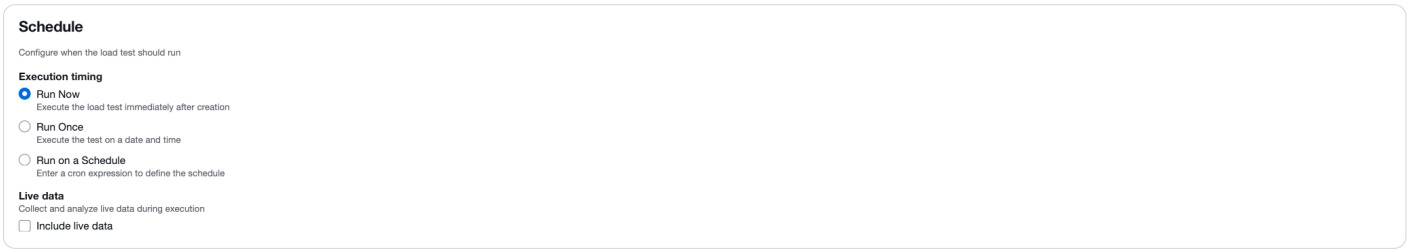
テストの識別

- テスト名 (必須) - テストシナリオのわかりやすい名前
- テストの説明 (必須) - テストの目的と設定に関する追加の詳細
- タグ (オプション) - 最大 5 つのタグを追加して、テストシナリオを分類および整理します。

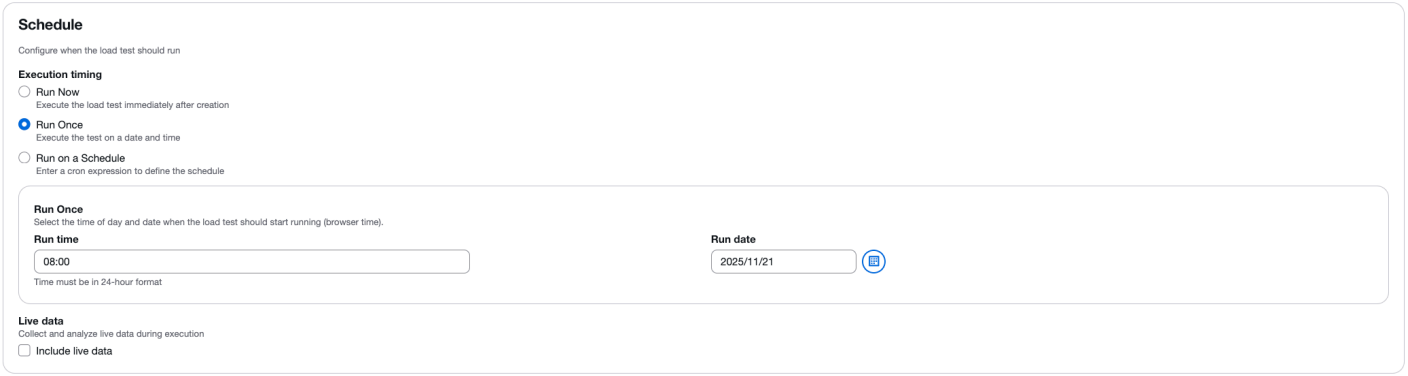
スケジュールオプション

テストを実行するタイミングを設定します。

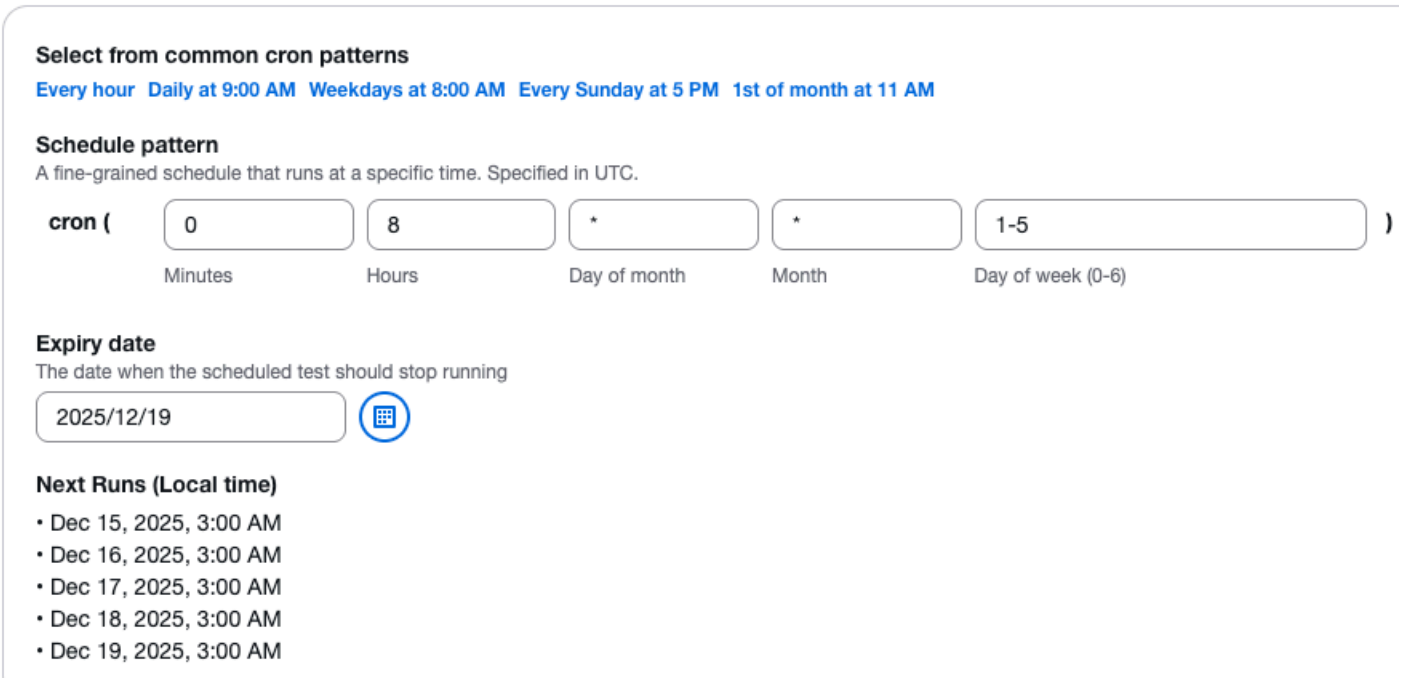
- 今すぐ実行 - 作成後すぐにテストを実行します。



- 1 回実行 - 特定の日時にテストを実行するようにスケジュールします。



- スケジュールに従って実行 - cron ベースのスケジューリングを使用して、定期的にテストを自動的に実行します。一般的なパターン (毎時間、毎日、毎週) から選択することも、カスタム cron 式を定義することもできます。



スケジューリングワークフロー

テストをスケジューリングすると、次のワークフローが発生します。

- スケジュールパラメータは、Amazon API Gateway を介してソリューションの API に送信されます。
- API はパラメータを Lambda 関数に渡し、スケジュールされた CloudWatch Events ルールを作成して、指定した日付に実行されるようにします。

- 1 回限りのテスト (1 回実行) の場合、CloudWatch Events ルールは指定された日に実行し、`api-services` Lambda 関数はテストを実行します。
- 定期的なテスト (スケジュールに従って実行) の場合、CloudWatch Events ルールは指定された日付にアクティブ化され、`api-services` Lambda 関数は指定された頻度に基づいてすぐに、かつ繰り返し実行される新しいルールを作成します。

ライブデータ

テストの実行中にリアルタイムのメトリクスを表示するには、[ライブデータを含める] チェックボックスをオンにします。有効にすると、以下をモニタリングできます。

- 平均応答時間。
- 仮想ユーザー数。
- 成功したリクエストカウント。
- 失敗したリクエストカウント。

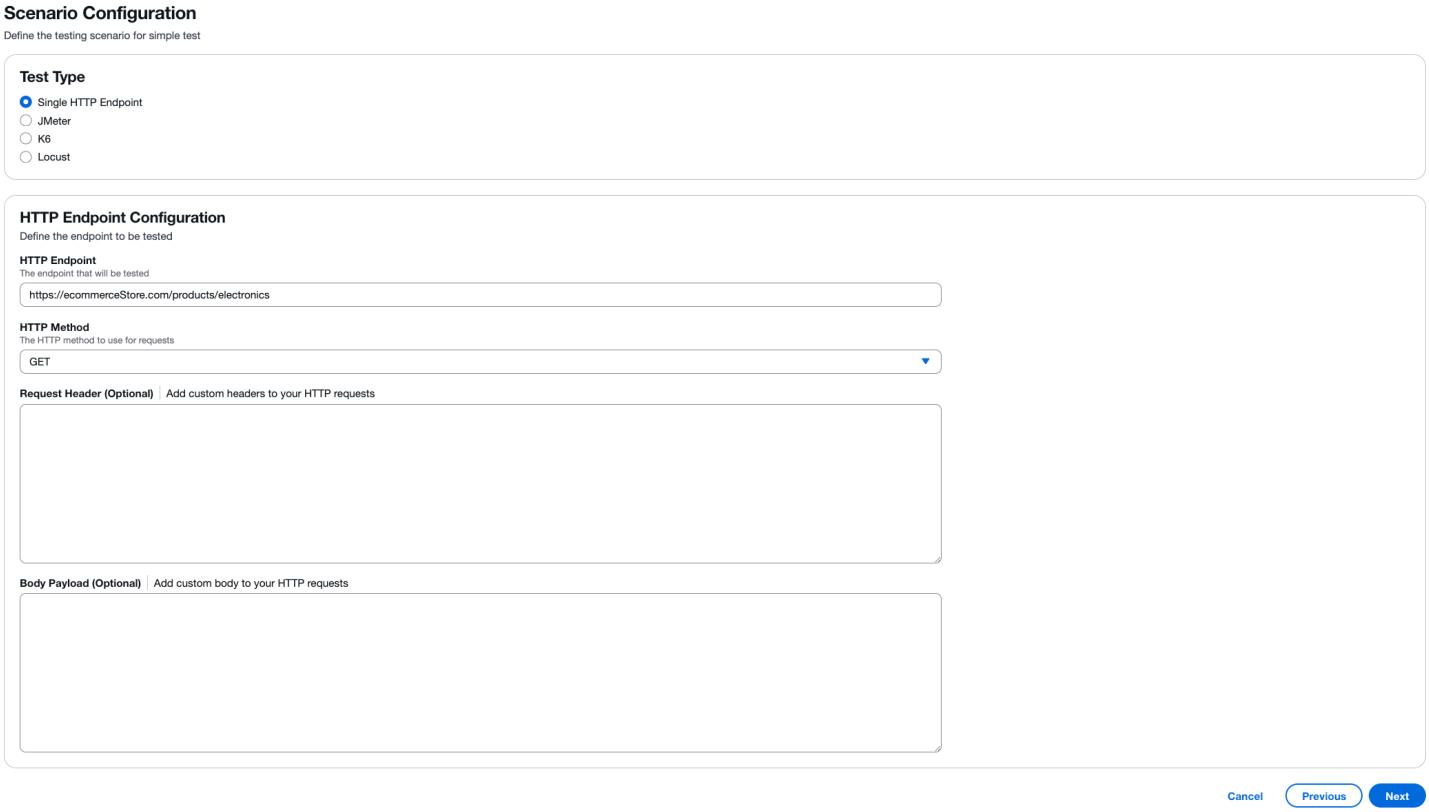
ライブデータ機能は、1 秒間隔で集計されたデータをリアルタイムでグラフ表示します。詳細については、「ライブデータを使用したモニタリング」を参照してください。

ステップ 2: シナリオ設定

特定のテストシナリオを定義し、希望するテストフレームワークを選択します。

テストタイプの選択

実行する負荷テストのタイプを選択します。



- 単一の HTTP エンドポイント - シンプルな設定で単一の API エンドポイントまたはウェブページをテストします。
- JMeter - JMeter テストスクリプト (.jmx ファイルまたは .zip アーカイブ) をアップロードします。
- K6 - K6 テストスクリプト (.js ファイルまたは .zip アーカイブ) をアップロードします。
- Locust - Locust テストスクリプト (.py ファイルまたは .zip アーカイブ) をアップロードします。

HTTP エンドポイント設定 image::images/test-types.png[Select the test type to run] [単一の HTTP エンドポイント] を選択した場合は、以下の設定を行います。

HTTP エンドポイント (必須)

テストするエンドポイントの完全な URL を入力します。例えば、`https://api.example.com/users`。エンドポイントが AWS インフラストラクチャからアクセス可能であることを確認します。

HTTP メソッド (必須)

リクエストの HTTP メソッドを選択します。デフォルトは `GET` です。その他のオプションには、`POST`、`PUT`、`DELETE`、`PATCH`、`HEAD`、`OPTIONS` が含まれます。

リクエストヘッダー (オプション)

リクエストにカスタム HTTP ヘッダーを追加します。一般的な例には、次が含まれます。

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

[ヘッダーの追加] を選択して、複数のヘッダーを含めます。

本文ペイロード (オプション)

POST または PUT リクエストのリクエスト本文のコンテンツを追加します。JSON、XML、またはプレーンテキスト形式をサポートします。例: `{"userId": 123, "action": "test"}`。

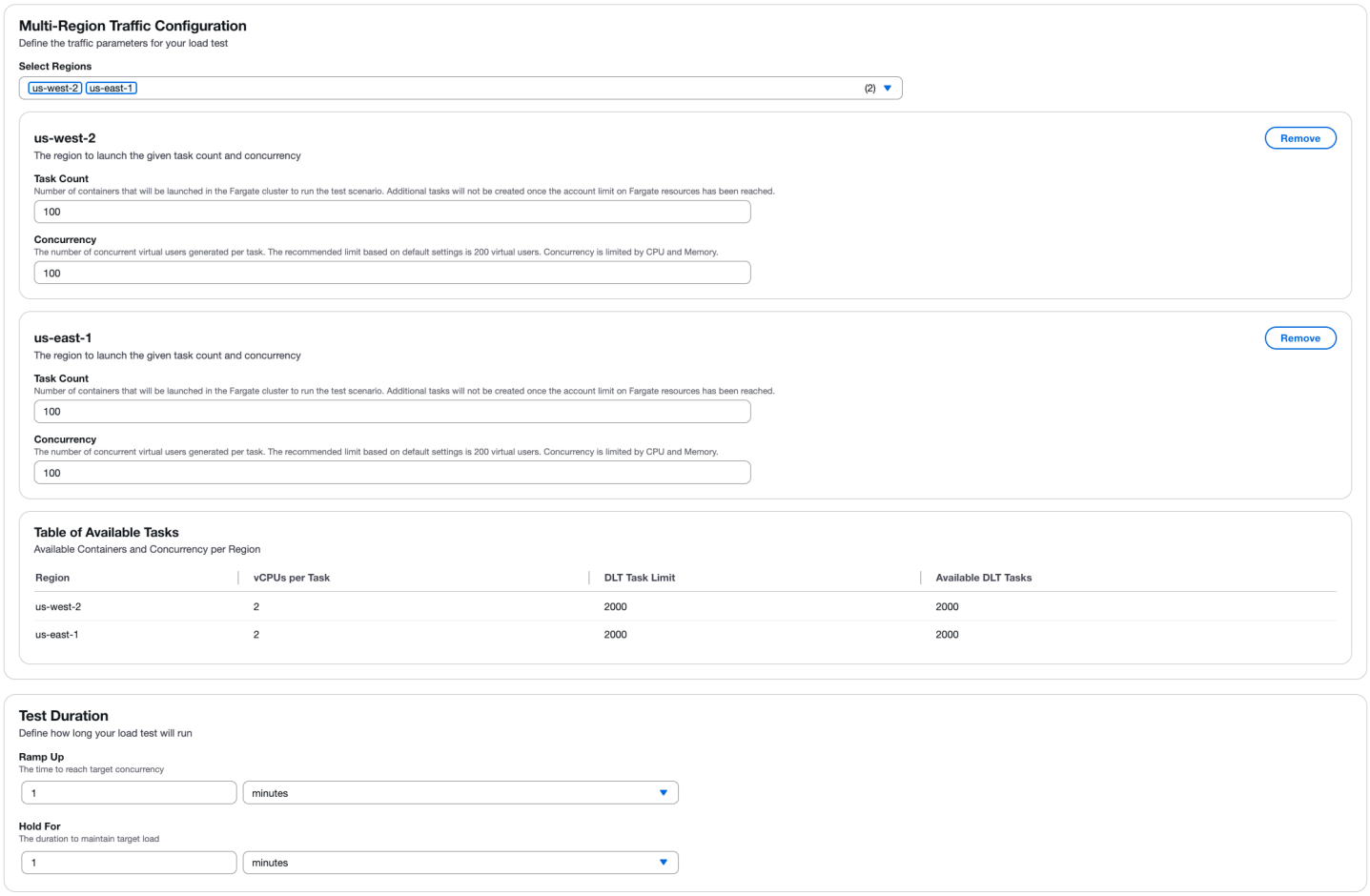
フレームワークスクリプトをテストする

JMeter、K6、または Locust を使用する場合、テストスクリプトファイル、またはテストスクリプトとサポートファイルを含む .zip アーカイブをアップロードします。JMeter では、.zip アーカイブ内の `/plugins` フォルダにカスタムプラグインを含めることができます。

> **Important**
>
> テストスクリプト (JMeter、K6、または Locust) は同時実行数 (仮想ユーザー)、トランザクションレート (TPS)、ランプアップ時間、およびその他のロードパラメータを定義する場合がありますが、ソリューションはテストの作成時に [Traffic Shape] 画面で指定した値でこれらの設定を上書きします。Traffic Shape 設定は、テスト実行のタスク数、同時実行数 (タスクあたりの仮想ユーザー数)、ランプアップ時間、保留期間を制御します。

ステップ 3: トラフィックシェイプ

マルチリージョンのサポートを含め、テスト中にトラフィックを分散する方法を設定します。



マルチリージョンのトラフィック設定

1 つ以上の AWS リージョンを選択して、負荷テストを地理的に分散します。選択したリージョンごとに、次を設定します。

タスク数

テストシナリオのために Fargate クラスターで起動されるコンテナ (タスク) の数。アカウントが「Fargate リソースが制限に達しました」に達すると、追加のタスクは作成されません。

同時実行

タスクごとに生成された同時仮想ユーザーの数。推奨される制限は、タスクごとに 2 つの vCPU のデフォルト設定に基づいています。同時実行数は CPU とメモリのリソースによって制限されます。

ユーザー数を確定する

コンテナがテストでサポートできるユーザー数は、ユーザー数を徐々に増やしつつ、Amazon CloudWatch でパフォーマンスをモニタリングしながら決定することができます。CPU とメモリのパフォーマンスが限界に近づくと、コンテナがデフォルト設定 (2 vCPU と 4 GB のメモリ) でテストをサポートできる最大ユーザー数に達します。

キャリブレーションプロセス

次の例を使用して、テストの同時ユーザー制限の確定を開始できます。

1. 200 人以下のユーザーでテストを作成します。
2. テストの実行中に、CloudWatch コンソールを使用して CPU とメモリをモニタリングします。
   a. 左側のナビゲーションペインの [Container Insights] で、[パフォーマンスのモニタリング] を選択します。
   b. [パフォーマンスのモニタリング] ページで、左側のドロップダウンメニューから [ECS クラスター] を選択します。
   c. 右側のドロップダウンメニューから、Amazon Elastic Container Service (Amazon ECS) クラスターを選択します。
3. モニタリング中は、CPU とメモリを監視します。CPU が 75% を超えない、またはメモリが 85% を超えない (1 回限りのピークは無視) 場合は、より多くのユーザー数で別のテストを実行できます。

テストがリソースの制限を超えなかった場合は、手順の 1～3 を繰り返します。オプションで、コンテナのリソースを増やして、同時実行ユーザー数を増やすことができます。ただし、これによりコストが高くなります。詳細については、「デベロッパーガイド」を参照してください。

> **Note**
>
> 正確な結果を得るには、同時ユーザー制限を決定する際に一度に 1 つのテストのみを実行してください。すべてのテストでは同じクラスターを使用し、CloudWatch Container Insights ではクラスターに基づいてパフォーマンスデータを集計します。これにより、両方のテストが CloudWatch Container Insights に同時にレポートされるため、単一テストでのリソース使用率のメトリクスが不正確になります。

エンジンごとのユーザー調整に関する詳細については、BlazeMeter ドキュメントの「Calibrating a Taurus Test」を参照してください。

> **Note**
>
> このソリューションでは、各リージョンの使用可能な容量情報が表示され、使用可能な制限内でテスト設定を計画するのに役立ちます。

使用可能なタスクの表

[使用可能なタスクの表] には、選択した各リージョンのリソースの可用性が表示されます。

- リージョン - AWS リージョン名。
- タスクあたりの vCPU - 各タスクに割り当てられた仮想 CPU の数 (デフォルト: 2)。
- DLT タスク制限 - アカウントの Fargate 制限に基づいて作成できるタスクの最大数 (デフォルト: 2000)。
- 使用可能な DLT タスク - リージョンで現在、使用可能なタスクの数 (デフォルト: 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

タスクごとに使用可能なタスクまたは vCPU の数を増やすには、デベロッパーガイドを参照してください。

テスト期間

負荷テストの実行期間を定義します。

ランプアップ

ターゲットの同時実行数に達するまでの時間。この期間、負荷は 0 から設定された同時実行レベルまで徐々に増加します。

ホールド

ターゲット負荷を維持する期間。この期間中、テストは完全な同時実行で続行されます。

ステップ 4: 確認して作成する

テストシナリオを作成する前に、すべての設定を確認してください。検証:

- 全般設定 (名前、説明、スケジュール)。
- シナリオ設定 (テストタイプ、エンドポイント、またはスクリプト)。
- トラフィックシェイプ (タスク、ユーザー、期間、リージョン)。

確認後、[作成] を選択してテストシナリオを保存します。

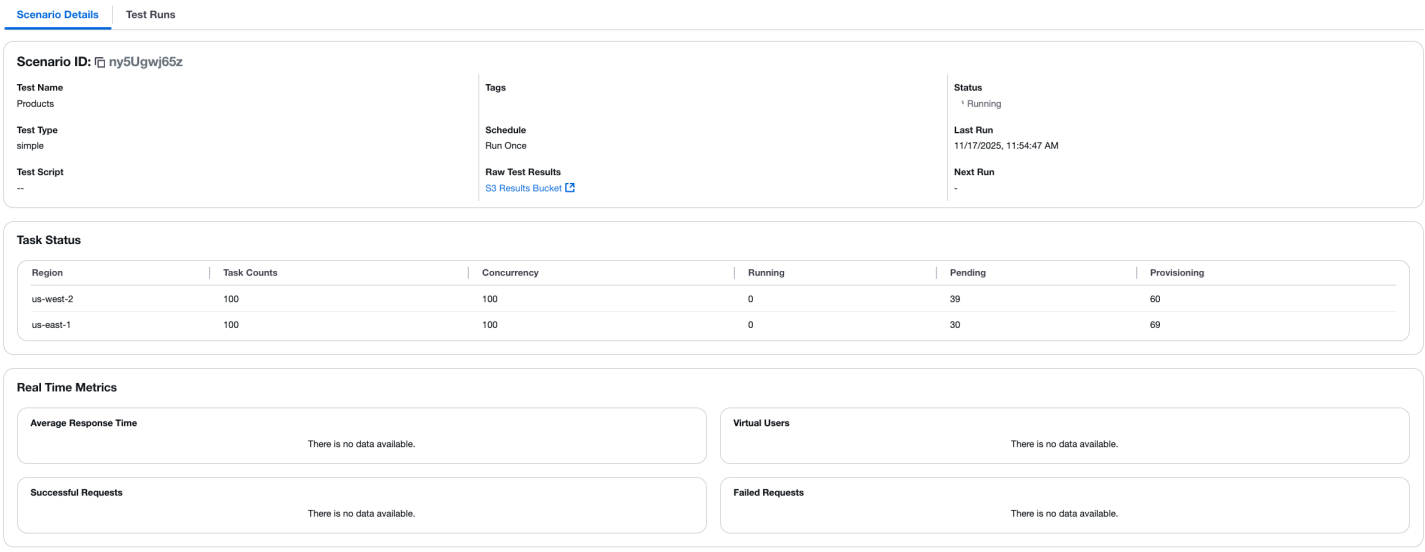
テストシナリオの管理

テストシナリオを作成したら、次のことができます。

- 編集 - テスト設定を変更します。一般的ユースケースには以下が含まれます。
  - トラフィックシェイプを調整して、希望するトランザクションレートを実現します。
- コピー - 既存のテストシナリオを複製してバリエーションを作成します。一般的ユースケースには以下が含まれます。
  - エンドポイントの更新またはヘッダー/本文パラメータの追加。
  - テストスクリプトの追加または変更。
- 削除 - 不要になったテストシナリオを削除します。

テストシナリオを実行する

テストシナリオを作成したら、すぐに実行するか、将来の特定の時間に実行するようにスケジュールできます。実行中のテストに移動すると、コンソールにリアルタイムのタスクステータスとメトリクスを含むシナリオの詳細タブが表示されます。



シナリオの詳細ビュー

シナリオの詳細タブには、テストに関する主要な情報が表示されます。各リージョンのタスクステータステーブルのリアルタイム情報。

タスクステータステーブル

タスクステータステーブルには、各リージョンののリアルタイム情報が表示されます。

- リージョン - タスクが実行されている AWS リージョン
- タスク数 - リージョンに設定されたタスクの合計数
- 同時実行数 - タスクあたりの仮想ユーザー数
- 実行中 - 現在テストを実行しているタスクの数
- 保留中 - 開始を待っているタスクの数
- プロビジョニング - プロビジョニング中のタスクの数

テスト実行ワークフロー

テストが開始されると、次のワークフローが発生します。

1. タスクプロビジョニング - ソリューションは、指定された AWS リージョンにコンテナ (タスク) をプロビジョニングします。タスクは [プロビジョニング中] 列に表示されます。

2. タスクのスタートアップ - ソリューションは、各リージョンでターゲットタスク数に達するまでタスクのプロビジョニングを続行します。タスクは、[プロビジョニング中] から [保留中]、そして [実行中] に移動します。
3. トラフィック生成 - ソリューションがリージョン内のすべてのタスクをプロビジョニングすると、ターゲットエンドポイントへのトラフィックの送信が開始されます。
4. テストの実行 - テストは設定された期間 (ランプアップ + ホールド時間) 実行されます。
5. 結果の解析 - テストが終了すると、バックグラウンド解析ジョブはすべてのリージョンの結果を集計して処理します。

テスト実行のステータス

テスト実行は、次の状態になります。

- スケジュール済み - テストは今後実行されるようスケジュールされています。
- 実行中 - テストは現在進行中です。
- キャンセル済み - ユーザーが進行中のテスト実行をキャンセルしました。
- エラー発生 - テスト実行でエラーが発生しました。
- 完了 - テスト実行は正常に完了し、結果の準備が完了しています。

ライブデータを使用したモニタリング

テストシナリオの作成時にライブデータを有効にした場合、テストの実行中にリアルタイムのメトリクスを表示できます。リアルタイムメトリクスセクションには、テストの進行に伴って継続的に更新される 4 つのグラフが表示され、データは 1 秒間隔で集計されます。

グラフの説明

平均応答時間

各リージョンで処理されたリクエストの平均応答時間を秒単位で示します。Y 軸は応答時間を秒単位で示し、X 軸はその日の時刻を示します。各リージョンは、凡例内の異なる色で表されます。

仮想ユーザー

各リージョンで負荷をアクティブに生成する同時仮想ユーザーの数を示します。グラフには、テスト中に仮想ユーザーがどのように増加し、ターゲットの同時実行レベルが維持されるかが表示されます。

成功したリクエスト

各リージョンで成功したリクエストの累積数を継続的に示します。グラフには、成功したリクエストが処理されるレートが表示されます。

失敗したリクエスト

各リージョンで失敗したリクエストの累積数を継続的に示します。カウントが低いかゼロの場合は、正常なテスト実行であることを示します。

マルチリージョンの視覚化

複数のリージョンでテストを実行する場合、各グラフにはすべてのリージョンのデータが同時に表示されます。各グラフの下部にある凡例で、各リージョンを表す色 (us-west-2 や us-east-1 など) を識別できます。

技術的な実装

Fargate タスクの CloudWatch ロググループには、テスト結果をキャプチャするサブスクリプションフィルタが含まれています。パターンが検出されると、Lambda 関数はデータを構造化して AWS IoT Core トピックに発行します。ウェブコンソールはこのトピックにサブスクライブし、リアルタイムでメトリクスを表示します。

> **Note**
>
> ライブデータはエフェメラルであり、テストの実行中にのみ使用できます。ウェブコンソールには最大 5,000 個のデータポイントが保存され、その後、最も古いデータが最新のデータに置き換えられます。ページが更新されると、グラフは空白になり、次に利用可能なデータポイントから開始されます。テストが完了すると、このソリューションは結果データを DynamoDB と Amazon S3 に保存します。まだ利用可能なデータがない場合、グラフには「利用可能なデータはありません」と表示されます。

テストのキャンセル

実行中のテストは、ウェブコンソールからキャンセルできます。テストをキャンセルすると、次のワークフローが発生します。

1. キャンセルリクエストが `microservices` API に送信されます
2. `microservices` API は、現在起動しているすべてのタスクを停止する `task-canceler` Lambda 関数を呼び出します。
3. `task-runner` Lambda 関数が最初のキャンセル呼び出し後も引き続き実行される場合、タスクは一時的に起動し続けることがあります。
4. `task-runner` Lambda 関数が終了すると、AWS Step Functions は `Cancel Test` ステップに進み、`task-canceler` Lambda 関数を再度実行して、残りのタスクを停止します。

> **Note**
>
> ソリューションがすべてのコンテナを終了するため、キャンセルされたテストはシャットダウンプロセスを完了するのに時間がかかります。すべてのリソースがクリーンアップされると、テストステータスは [キャンセル済み] に変わります。

テスト結果を調べる

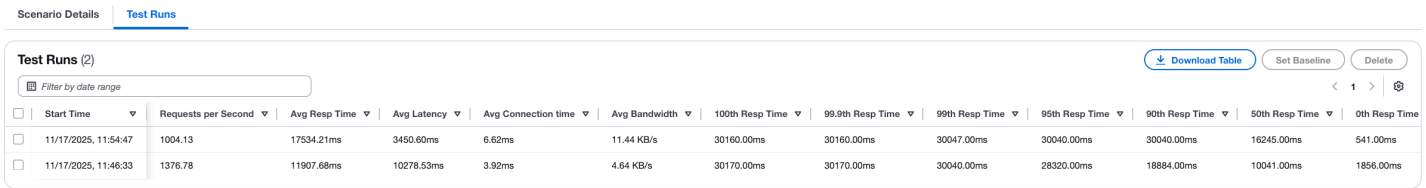
解析ジョブが完了すると、テスト結果は分析のために使用できます。ソリューションは、ロード中のアプリケーションのパフォーマンスを理解するのに役立つ包括的なメトリクスとツールを提供します。

テスト実行のサマリーメトリクス

テストが完了すると、ソリューションは次のメトリクスを含む概要を生成します。

- 平均応答時間 - テストによって生成されたすべてのリクエストの平均応答時間 (秒)。
- 平均レイテンシー - テストによって生成されたすべてのリクエストの平均レイテンシー (秒)。
- 平均接続時間 - テストで生成されたすべてのリクエストについて、ホストへの接続にかかった平均時間 (秒)。
- 平均帯域幅 - テストで生成されたすべてのリクエストの平均帯域幅。
- Total count - リクエストの総数
- Success count - 成功したリクエストの総数
- Error count - エラーの総数
- 1 秒あたりのリクエスト - テストで生成されたすべてのリクエストの 1 秒あたりの平均リクエスト数。
- パーセンタイル - p50 (中央値)、p90、p95、p99 などの応答時間のパーセンタイル。すべてのリクエストの応答時間の分布を示します。

テスト実行テーブル



テスト実行のテーブルには、シナリオのすべての過去のテスト実行が表示されます。以下の操作を実行できます。

- 各テスト実行のサマリーメトリクスを表示する。
- パフォーマンス比較用のベースラインテスト実行を設定する。
- CSV ファイルとしてテーブルをダウンロードする。
- 列を切り替えてビューをカスタマイズする。
- テスト実行を選択して、詳細な結果を表示する。

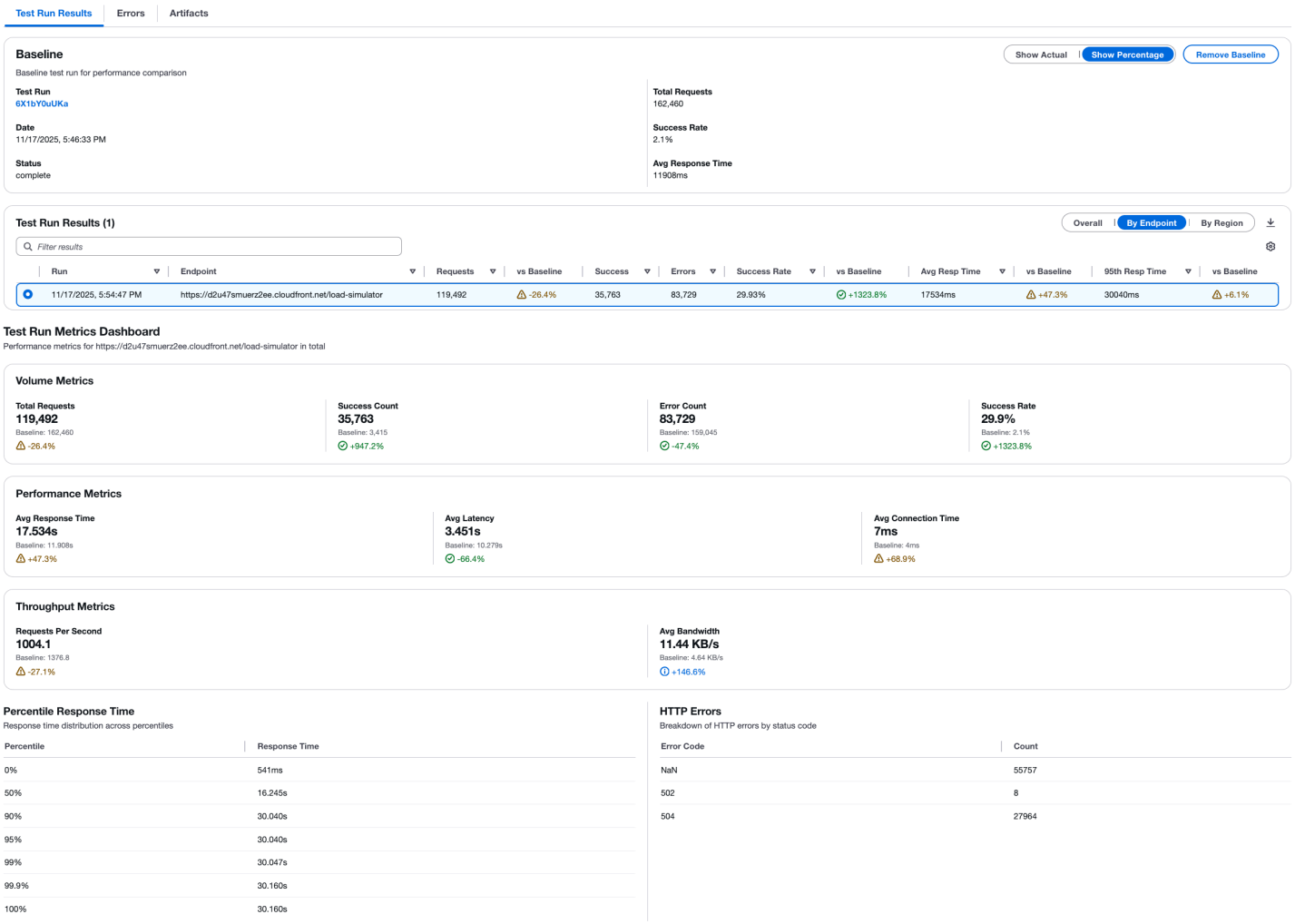
ベースラインの比較

あるテスト実行をベースラインとして指定し、今後のテスト実行と比較できます。ベースラインが設定されている場合:

- テスト実行テーブルには、各メトリクスのベースラインと比較した割合の差 (+/-%) が表示されます。
- ベースラインインジケータは、パフォーマンスの向上やリグレッションを迅速に特定するのに役立ちます。
- ベースラインはいつでも変更またはクリアできます。

詳細なテスト結果

テスト実行を選択すると、テスト実行の結果、エラー、アーティファクトの 3 つのタブのある詳細な結果ビューが開きます。



ベースラインの情報

ベーステストテスト実行が設定されている場合、ページの上部に表示されます。値を表示、割合を表示、またはベースラインを削除を選択して、ベースライン比較の表示方法を制御できます。

テスト実行結果テーブル

結果テーブルには、次の機能を備えた詳細なメトリクスが表示されます。

ディメンションビュー

ディメンションボタンを使用して 3 つのビューを切り替えます。

- 全体 - すべてのエンドポイントとリージョンにわたる集計結果
- エンドポイント別 - 個々のエンドポイント別に分類された結果
- リージョン別 - AWS リージョン別に分類された結果

アクションボタン

- 値を表示 - 実際のメトリクス値を表示します
- 割合を表示 - ベースラインからの差の割合を表示します
- ベースラインを削除 - ベースラインの比較をクリアします

データのエクスポートとカスタマイズ

- CSV ファイルとして結果テーブルをダウンロードする
- 列を切り替えてビューをカスタマイズする
- データのフィルタリングとソートを行い、特定のメトリクスに絞る
- データのフィルタリングとソートを行い、特定のメトリクスに絞ります。

エラータブ

エラータブには、詳細なエラー分析が表示されます。

- タイプ別にエラーの件数を表示します。
- テスト全体またはエンドポイント別に集計されたエラーを参照してください。
- 失敗したリクエストのパターンを特定します。
- 特定のエンドポイントまたはリージョンに関する問題をトラブルシューティングします。

アーティファクトタブ

アーティファクトタブを使用すると、テストの実行中に生成されたすべてのファイルにアクセスできます。

- 個々のアーティファクト (ログ、結果ファイル) を表示します。
- オフライン分析用に特定のアーティファクトをダウンロードします。
- すべてのテスト実行のアーティファクトを 1 つのアーカイブとしてダウンロードします。

S3 の結果の構造

バージョン 4.0 では、組織を改善するために S3 の結果の構造が変更されました。

- 新しい構造 - `scenario-id/test-run-id/results-files`。
- レガシー構造 - バージョン 4.0 より前に実行されたテストでは、シナリオ ID レベルですべての結果ファイルが表示されます。

> **Note**
>
> テスト結果はコンソールに表示されます。`Results` フォルダの Amazon S3 バケットで未処理のテスト結果に直接アクセスすることもできます。Taurus のテスト結果の詳細については、Taurus User Manual の「Generating Test Reports」を参照してください。

MCP サーバー統合

ソリューションのデプロイ中にオプションの MCP サーバーコンポーネントをデプロイした場合、分散負荷テストソリューションをモデルコンテキストプロトコルをサポートする AI 開発ツールと統合できます。MCP サーバーは、AI アシスタントを介して負荷テストを取得、管理、分析するためのプログラムによるアクセスを提供します。

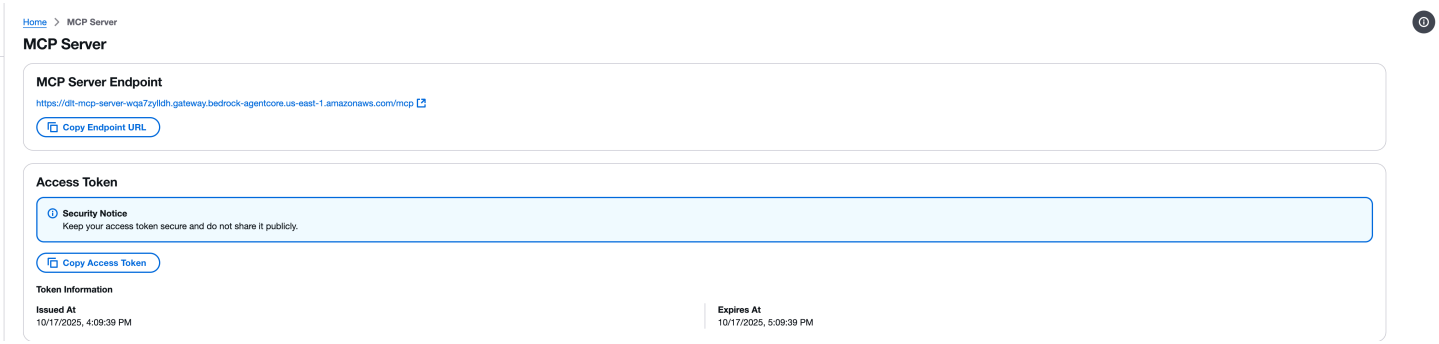
お客様は、選択したクライアント (Amazon Q、Claude など) を使用して DLT MCP サーバーに接続できます。それぞれの設定手順は若干異なります。このセクションでは、MCP Inspector、Amazon Q CLI、Cline、Amazon Q Suite の設定手順について説明します。

ステップ 1: MCP エンドポイントとアクセストークンを取得する

MCP クライアントを設定する前に、DLT ウェブコンソールから MCP サーバーエンドポイントとアクセストークンを取得する必要があります。

1. 分散負荷テストウェブコンソールの [MCP サーバー] ページに移動します。
2. [MCP サーバーエンドポイント] セクションを見つけます。
3. [エンドポイント URL のコピー] ボタンを使用してエンドポイント URL をコピーします。エンドポイント URL の形式は、`https://{gateway-id}.gateway.bedrock-agentcore.{region}.amazonaws.com/mcp` です。
4. [アクセストークン] セクションを見つけます。
5. [アクセストークンをコピー] ボタンを使用して、アクセストークンをコピーします。

> **Important**
>
> アクセストークンは安全に保管し、公開しないでください。トークンは、MCP インターフェイスを介して分散負荷テストソリューションへの読み取り専用アクセスを提供します。



ステップ 2: MCP Inspector を使用してテストする

モデルコンテキストプロトコルは、MCP サーバーに直接接続してツールを呼び出すためのツールである MCP Inspector を提供します。これにより、AI クライアントを設定する前に MCP サーバー接続をテストするのに便利な UI とサンプルネットワークリクエストが提供されます。

> **Note**
>
> MCP Inspector にはバージョン 0.17 以降が必要です。すべてのリクエストを JSON RPC で直接行うこともできますが、MCP Inspector はより使いやすいインターフェイスを提供します。

MCP Inspector をインストールして起動する

1. 必要に応じて、npm をインストールします。
2. 次のコマンドを実行して、MCP Inspector を起動します。

```
npx @modelcontextprotocol/inspector
```

接続を設定する

1. MCP Inspector インターフェイスで、MCP サーバーエンドポイント URL を入力します。

2. アクセストークンで認可ヘッダーを追加します。

3. [接続] をクイックして接続を確立します。

ツールを呼び出す

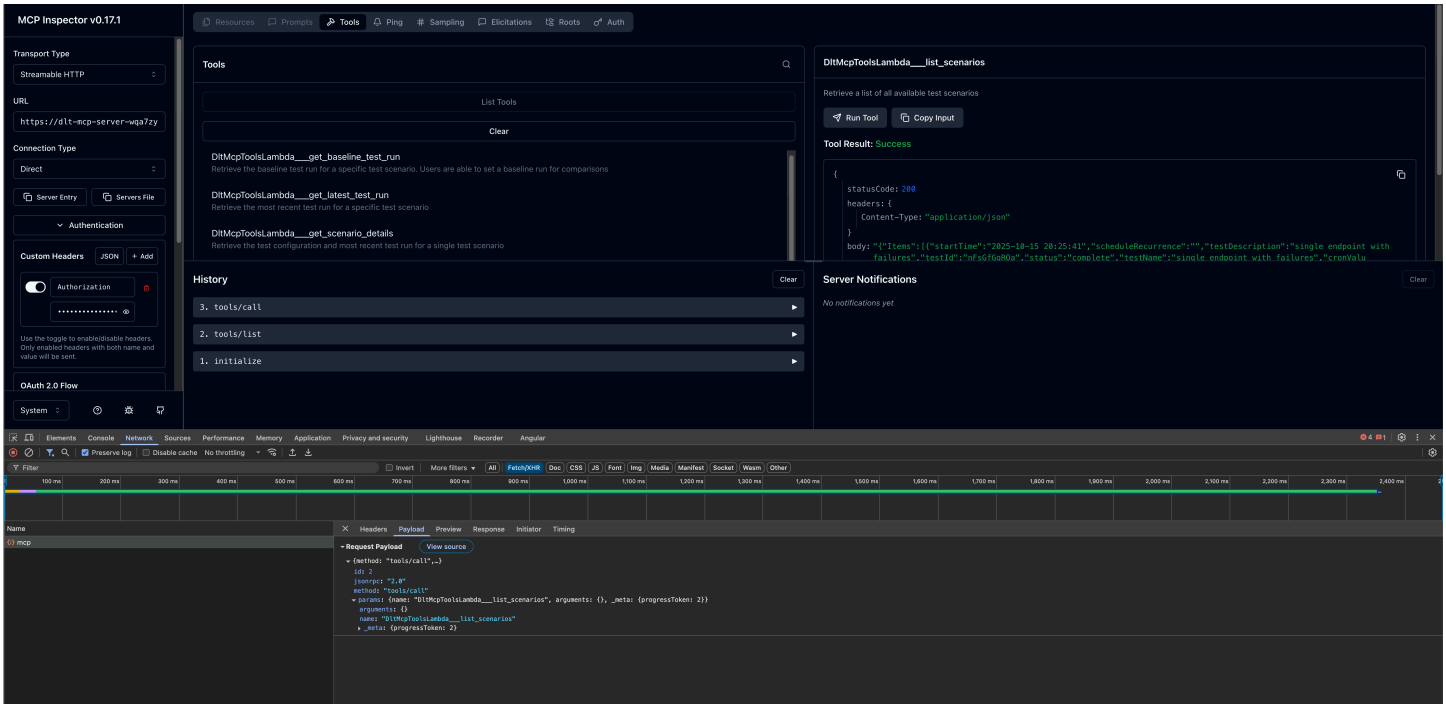
接続したら、利用可能な MCP ツールをテストできます。

1. 左側のパネルで利用可能なツールのリストを参照します。

2. ツール (`list_scenarios` など) を選択します。

3. 必要なパラメータを指定します。

4. [呼び出す] をクリックしてツールを実行し、レスポンスを表示します。



ステップ 3: AI 開発クライアントを設定する

MCP Inspector との MCP サーバー接続を確認したら、希望する AI 開発クライアントを設定できます。

Amazon Q CLI

Amazon Q CLI は、MCP サーバー統合による AI 支援型開発へのコマンドラインアクセスを提供します。

設定手順

1. `mcp.json` 設定ファイルを編集します。設定ファイルの場所の詳細については、「Amazon Q Developer ユーザーガイド」の「リモート MCP サーバーの設定」を参照してください。
2. DLT MCP サーバーの設定を追加します。

```
{
  "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"
      }
    }
  }
}
```

設定の検証:

1. ターミナルで、q と入力して Amazon Q CLI を起動します。
2. /mcp と入力すると、利用可能なすべての MCP サーバーが表示されます。
3. /tools と入力すると、dlt-mcp およびその他の設定済み MCP サーバーが提供し、使用可能なツールが表示されます。
4. dlt-mcp が正常に初期化されたことを確認します。

Cline

Cline は、MCP サーバー統合をサポートする AI コーディングアシスタントです。

設定手順

1. Cline で、[MCP サーバーの管理] > [設定] > [MCP サーバーを設定] に移動します。
2. cline_mcp_settings.json ファイルを更新します。

```
{
  "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. 設定ファイルを保存します。
4. Cline を再起動して変更を適用します。

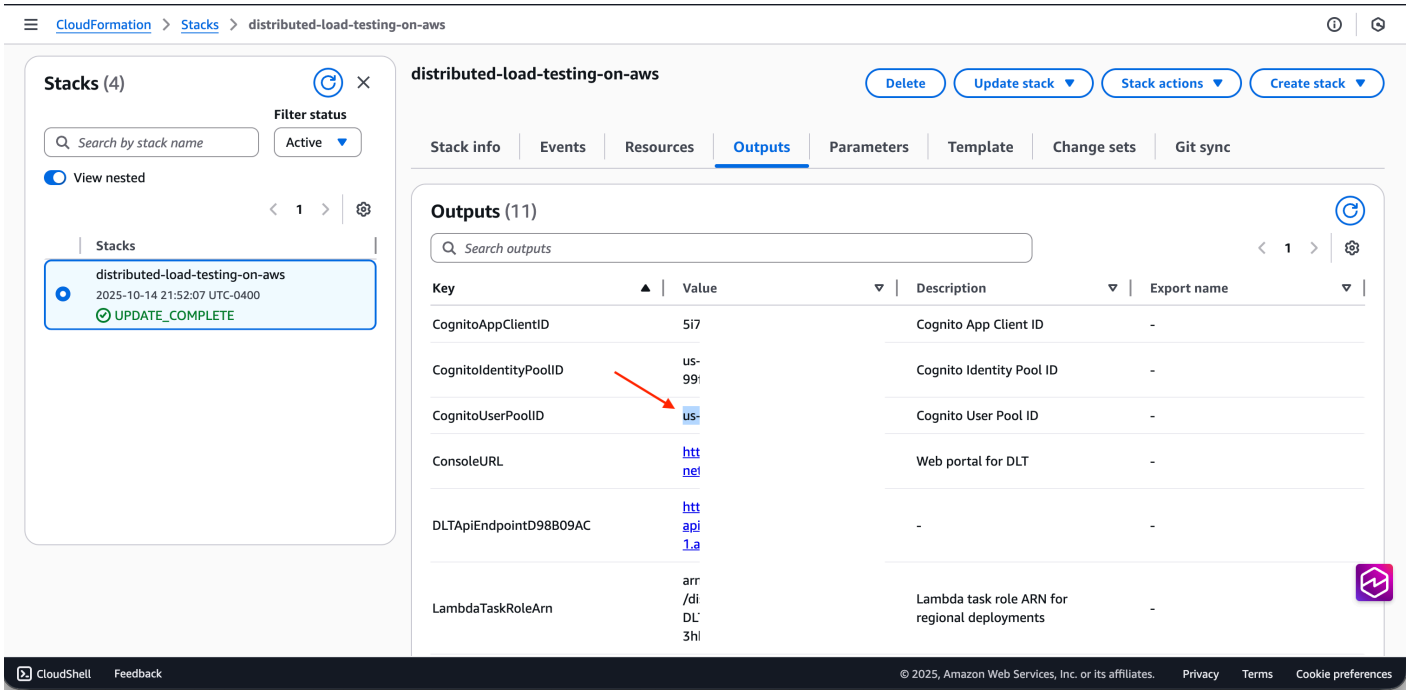
Amazon Q Suite

Amazon Q Suite は、MCP サーバーアクションをサポートする包括的な AI アシスタントプラットフォームを提供します。

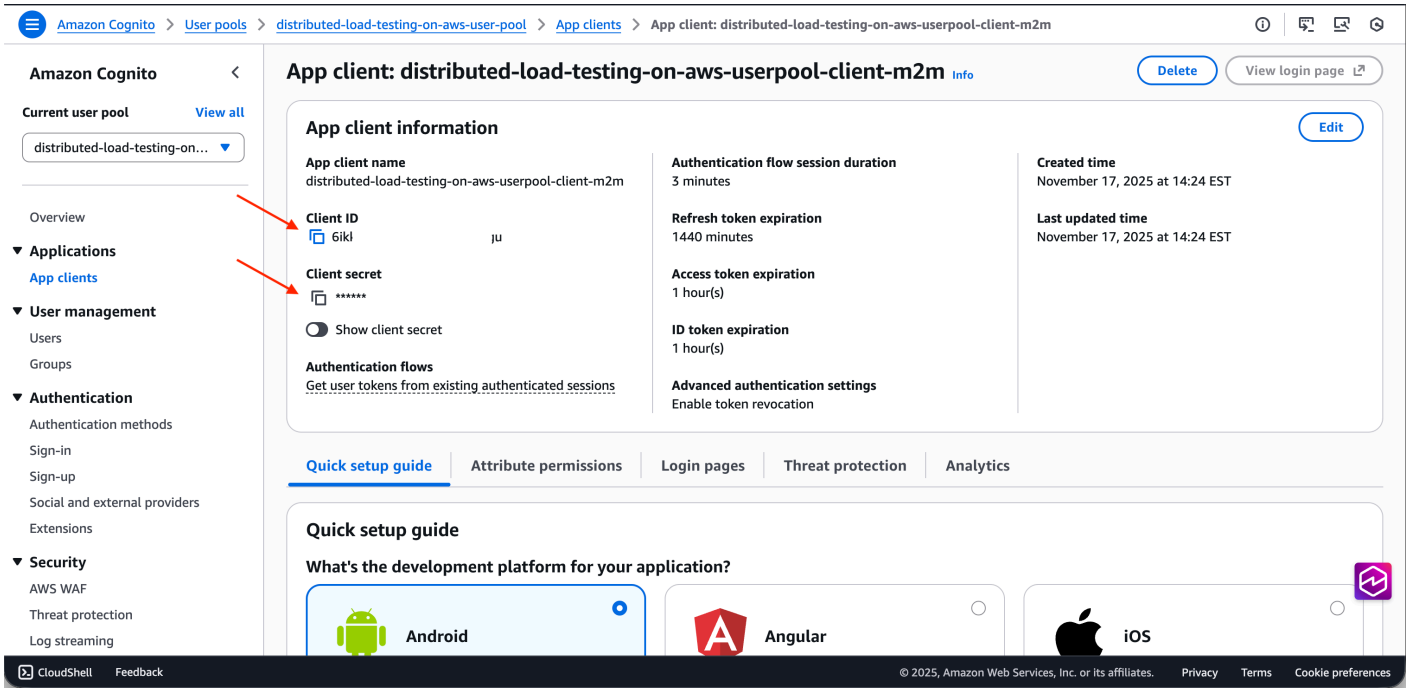
前提条件

Amazon Q Suite で MCP サーバーを設定する前に、DLT デプロイの Cognito ユーザープールから OAuth 認証情報を取得する必要があります。

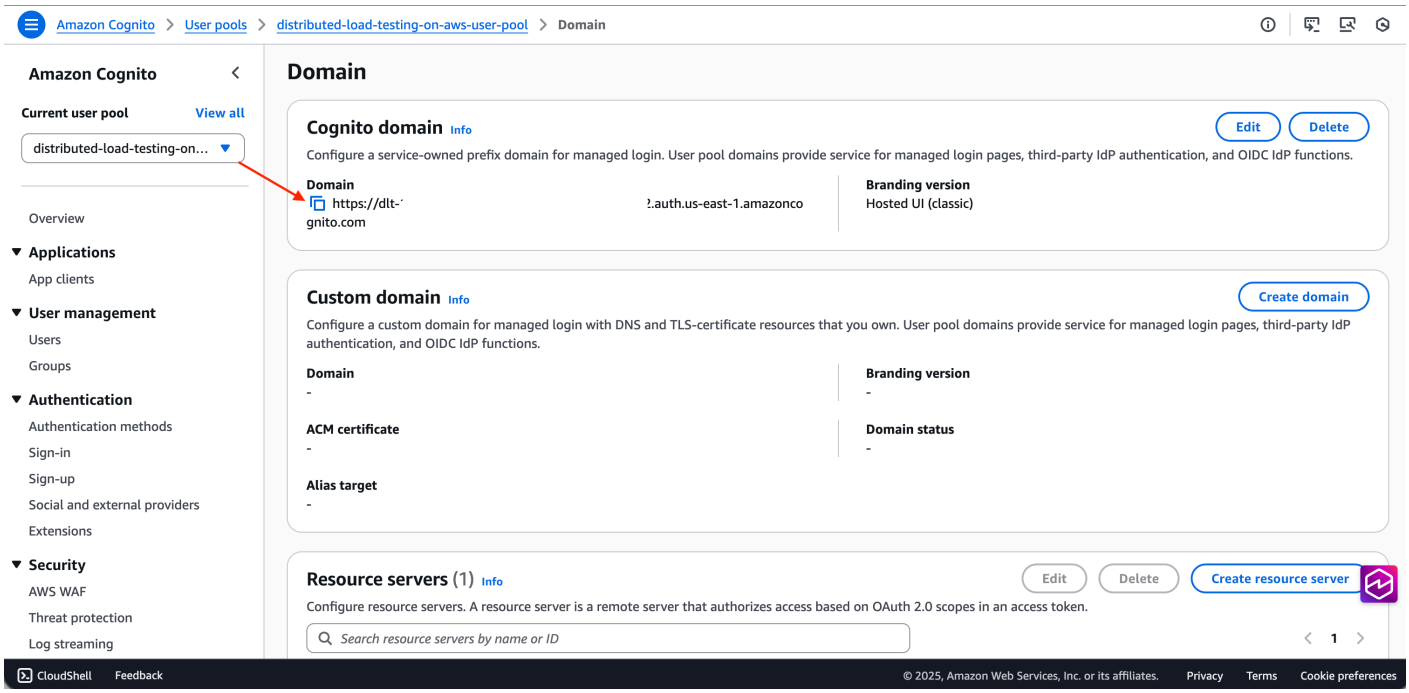
1. AWS CloudFormation コンソールに移動します。
2. 分散負荷テストスタックを選択します。
3. [出力] タブで、DLT デプロイに関連付けられた [Cognito ユーザープール ID] を見つけてコピーします。



4. Amazon Cognitoコンソール に移動します。
5. CloudFormation 出力からユーザープール ID を使用してユーザープールを選択します。
6. 左側のナビゲーションで、[アプリケーションの統合] > [アプリケーションクライアント] を選択します。



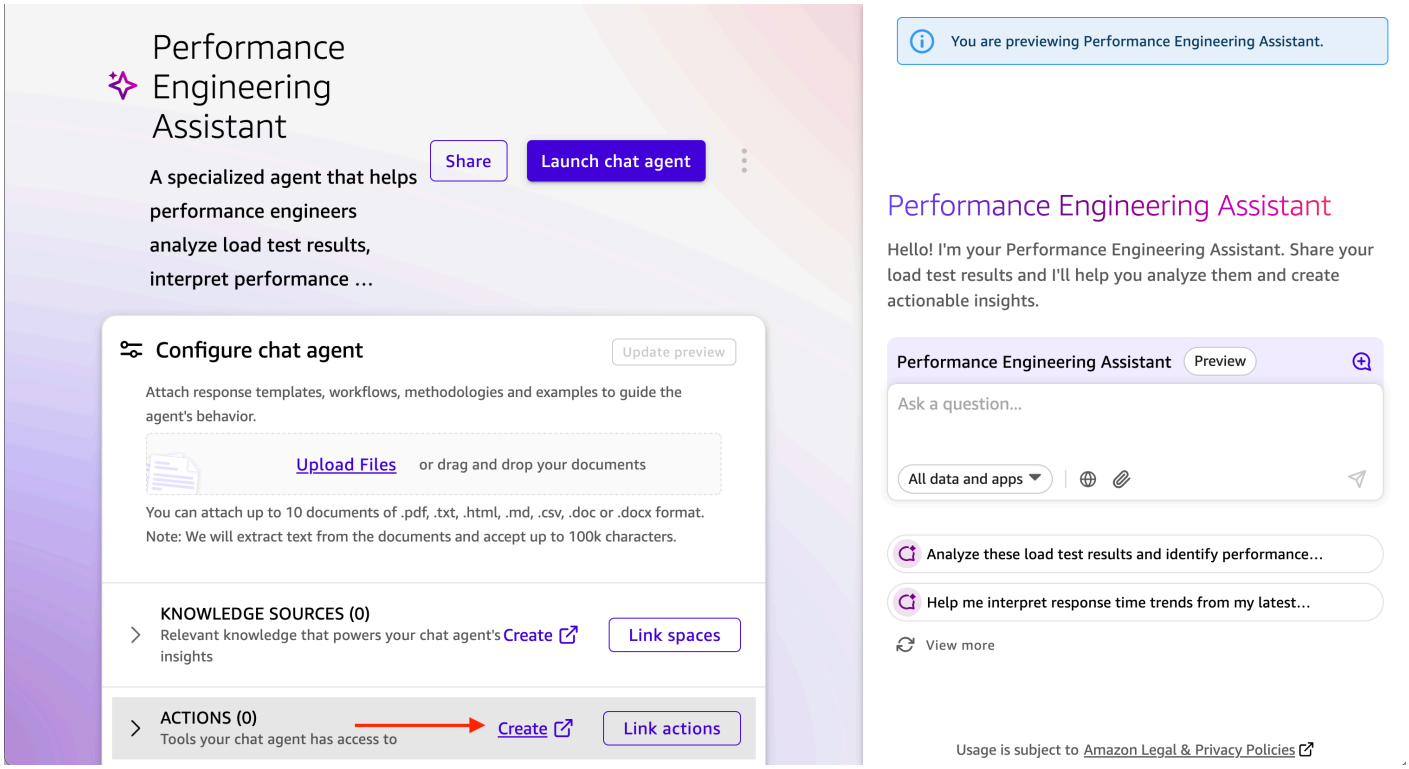
7. m2m (machine-to-machine) で終わる名前のアプリケーションクライアントを見つけます。
8. クライアント ID とクライアントシークレットをコピーします。
9. [ドメイン] タブからユーザープールドメインを取得します。

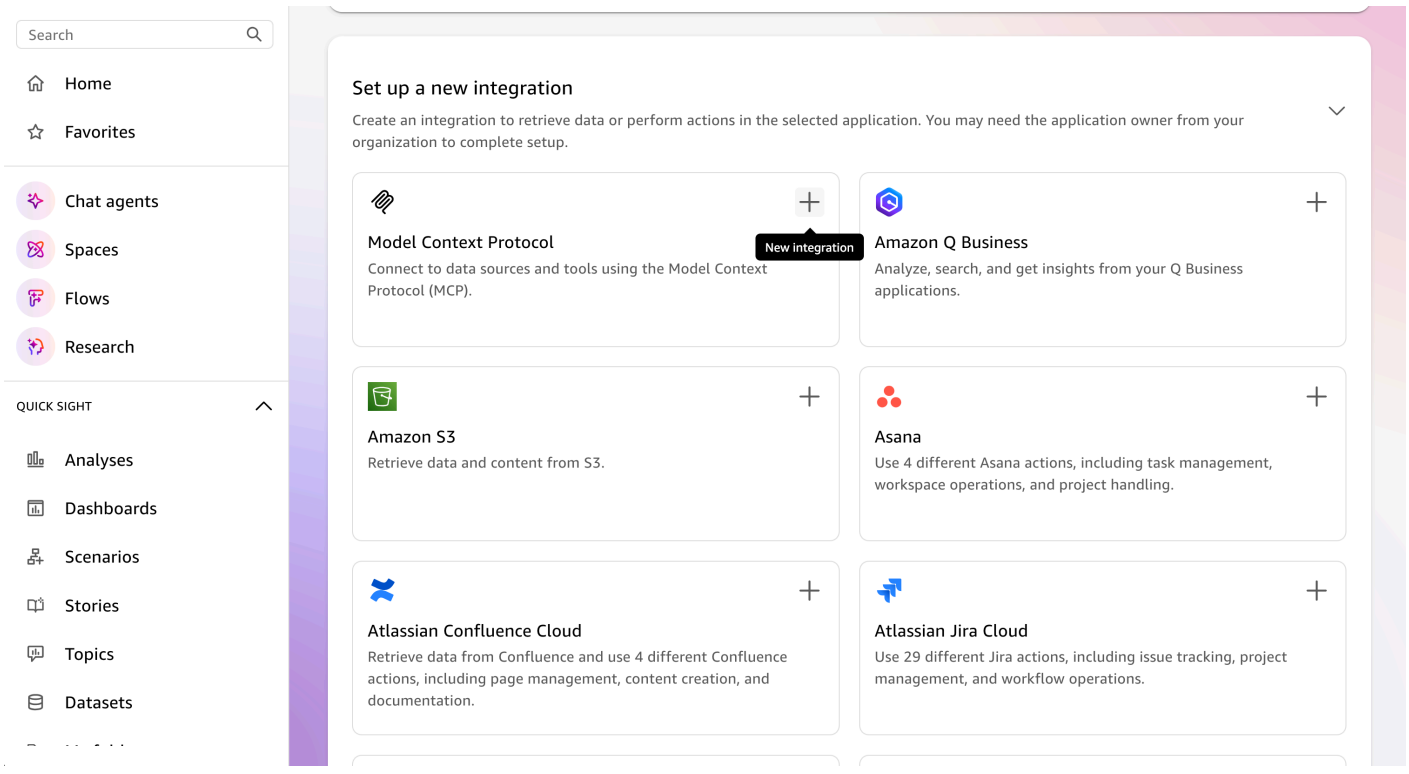


10.ドメインの末尾に /oauth2/token を追加して、トークンエンドポイント URL を作成します。

設定手順

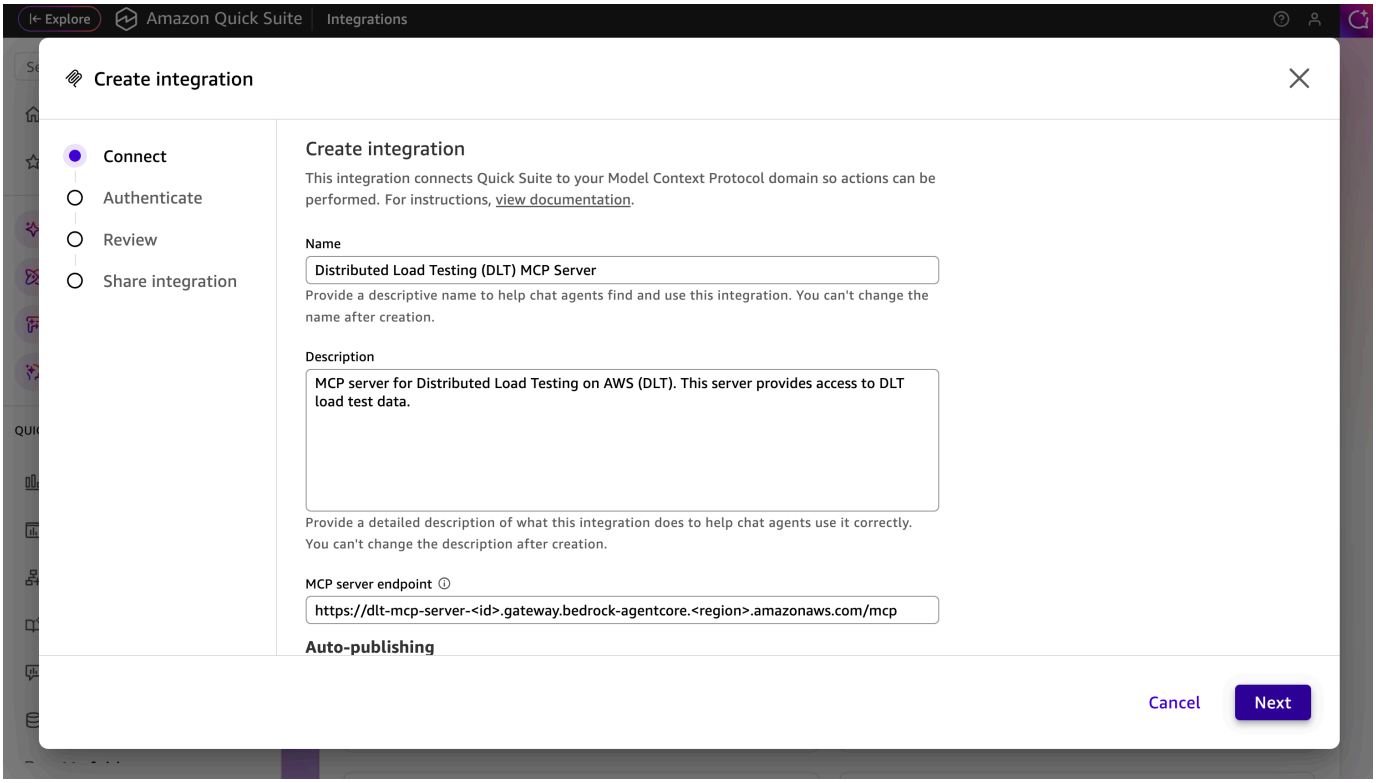
1. Amazon Q Suite で、新しいエージェントを作成するか、既存のエージェントを選択します。
2. DLT MCP サーバーを操作する方法を説明するエージェントプロンプトを追加します。
3. 新しいアクションを追加し、[MCP サーバーアクション] を選択します。



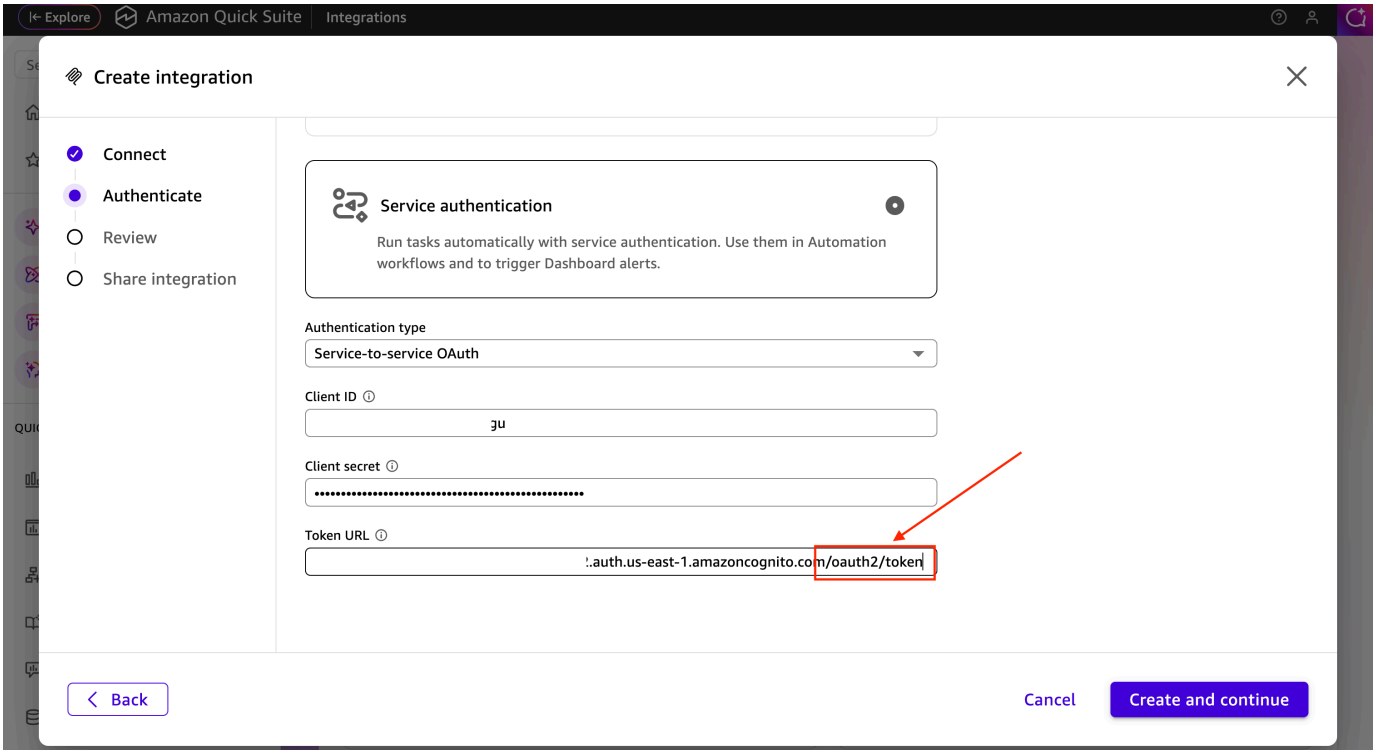


4. MCP サーバーの詳細を設定します。

- MCP サーバー URL: DLT MCP エンドポイント



- 認証タイプ: サービスベースの認証
- トークンエンドポイント: Cognito トークンエンドポイント URL
- クライアント ID: m2m アプリケーションクライアントからのクライアント ID
- クライアントシークレット: m2m アプリケーションクライアントからのクライアントシークレット



5. MCP サーバーアクション設定を保存します。
6. 新しい MCP サーバーアクションをエージェントに追加します。

エージェントを起動してテストする

1. Amazon Q Suite でエージェントを起動します。
2. 自然言語プロンプトを使用してエージェントとの会話を開始します。
3. エージェントは MCP ツールを使用し、負荷テストデータを取得して分析します。

プロンプトの例

次の例は、AI アシスタントとやり取りし、MCP インターフェイスを介して負荷テストデータを分析する方法を示しています。特定のテストニーズに一致するように、テスト ID、日付範囲、基準をカスタマイズします。

使用可能な MCP ツールとそのパラメータの詳細については、「デベロッパーガイド」の「MCP ツールの仕様」を参照してください。

シンプルなテスト結果のクエリ

MCP サーバーとの自然言語でのやり取りは、Show me the load tests that have completed in the last 24 hours with their associated completion status と同じくらいシンプルにすることも、次のような分かりやすいものにすることもできます。

```
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.
```

プログレッシブな開示によるインタラクティブなパフォーマンス分析

```
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.
```

本番稼働の準備状況の検証

```
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]%.
```

パフォーマンスの傾向分析

```
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.
```

失敗したテストのトラブルシューティング

```
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].
```

デベロッパーガイド

このセクションでは、このソリューションのソースコードと追加のカスタマイズについて説明します。

ソースコード

GitHub リポジトリにアクセスして、このソリューションのテンプレートとスクリプトをダウンロードし、カスタマイズした上で他のユーザーと共有できます。

メンテナンス

このソリューションでは、各ソリューションリリースに対応する固定バージョンの Docker イメージを使用します。デフォルトでは、自動更新は無効になっており、デプロイにいつ、どのバージョン更新が適用されるかを完全に制御できます。AWS ソリューションチームは、Amazon ECR 拡張スキャンを使用して、ベースイメージとインストール済みパッケージの共通脆弱性識別子 (CVE) を検出します。可能な場合、チームは、リリースされたソリューションバージョンとの互換性を損なうことなく、同じバージョンタグを持つ、パッチが適用されたイメージを公開して CVE を解決します。

同じマイナーバージョンでイメージにパッチを適用すると、安定したタグが自動的に更新され、追加のイメージタグが `<solution-version>_<date-of-fix>` 形式で作成されます。メジャーバージョンまたはマイナーバージョンがリリースされると、ソリューションのバージョンと一致するように安定したタグが増分されるため、最新のイメージバージョンを取得するにはフルスタック更新の実行が必要になります。

デプロイ中に自動更新をオプトインすると、イメージへの変更 (CVE パッチやマイナーバグ修正など) が、最新の一致するマイナーリリースまで自動的に適用されます。

バージョン

デフォルトでは、このソリューションは自動更新を無効にしてデプロイします。つまり、コンテナイメージのバージョンは、デプロイされたソリューションのバージョンに一致する特定のバージョンにロックされ、バージョンの更新を完全に制御できます。

CloudFormation のデプロイ中に [はい] を選択して自動更新を有効化すると、ソリューションは、一致する最新のマイナーバージョンまでのセキュリティパッチと、最新でないマイナーバグ修正を自動的に受け取ります。例えば、自動更新を有効にしてバージョン 4.0.0 をデプロイすると、4.1.0 以降ではなく、最大 4.0.x までの更新を受け取ります。

コンテナイメージのバージョンを手動で制御するには、タスク定義を編集して、タグ付けされたバージョン形式を使用して特定のイメージバージョンを指定できます。これにより、自動更新の設定に関係なく、特定のイメージバージョンに固定できます。

コンテナイメージのカスタマイズ

このソリューションでは、AWS が管理するパブリック Amazon Elastic Container Registry (Amazon ECR) イメージリポジトリを使用して、設定したテストの実行に使用するイメージを保存します。コンテナイメージをカスタマイズする場合は、イメージを再構築して、各自の AWS アカウントの ECR イメージリポジトリ内にプッシュできます。

このソリューションをカスタマイズする場合は、デフォルトのコンテナイメージを使用するか、ニーズに合わせてこのコンテナを編集してください。このソリューションをカスタマイズする場合は、カスタマイズしたソリューションをビルドする前に、次のコードサンプルを使用して環境変数を定義してください。

```
#!/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
```

コンテナイメージをカスタマイズする場合は、プライベートイメージリポジトリまたは AWS アカウントのパブリックイメージリポジトリのいずれかでホストできます。イメージリソースは、コードベースの `deployment/ecr/distributed-load-testing-on-aws-load-tester` ディレクトリにあります。

イメージをビルドして、ホスト先にプッシュできます。

- プライベート Amazon ECR リポジトリとイメージについては、「Amazon ECR ユーザーガイド」の「Amazon ECR プライベートリポジトリ」と「プライベートイメージ」を参照してください。
- パブリック Amazon ECR リポジトリとイメージについては、「Amazon ECR パブリックユーザーガイド」の「Amazon ECR public repositories」と「Amazon ECR Public image」を参照してください。

独自のイメージを作成したら、カスタマイズしたソリューションをビルドする前に、次の環境変数を定義できます。

```
#!/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
```

次の例は、コンテナファイルを示しています。

```
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"]
```

ディレクトリには、コンテナファイルに加えて、次の bash スクリプトが含まれています。このスクリプトでは Taurus/Blazemeter プログラムを実行する前に Amazon S3 からテスト設定をダウンロードします。

```
#!/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
```

Dockerfile と bash スクリプトに加えて、2 つの Python スクリプトもディレクトリに含まれています。各タスクは、bash スクリプト内から Python スクリプトを実行します。ワーカータスクは ecslistener.py スクリプトを実行し、リーダータスクは ecscontroller.py スクリプトを実行します。ecslistener.py スクリプトはポート 50000 にソケットを作成して、メッセージを待ちます。ecscontroller.py スクリプトはソケットに接続して、ワーカータスクにテスト開始メッセージを送信します。これにより、同時に起動できます。

分散負荷テスト API

この負荷テストソリューションは、テスト結果データを安全な方法で公開するのに役立ちます。API は、Amazon DynamoDB に保存されているテストデータにアクセスするための「フロントドア」として機能します。また、API を使用して、ソリューションに独自に拡張機能を組み込むことも可能です。

このソリューションでは、Amazon API Gateway と統合された Amazon Cognito ユーザープールを使用して、識別と認可を行います。API でユーザープールを使用する場合、クライアントは有効な ID トークンを提供した後にのみ、ユーザープールでアクティブ化されたメソッドを呼び出すことができます。

API 経由でテストを直接実行する方法の詳細については、Amazon API Gateway REST API リファレンスドキュメントの「リクエストの署名」を参照してください。

ソリューションの API では、以下のオペレーションが利用可能です。

> **Note**
>
> `testScenario` やその他のパラメータの詳細については、GitHub リポジトリにあるシナリオとペイロードの例を参照してください。

スタック情報

- GET /stack-info

シナリオ

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

テスト実行

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

ベースライン月

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

タスク

- GET /tasks
- OPTIONS /tasks

Regions

- GET /regions
- OPTIONS /regions

GET /stack-info

説明

`GET /stack-info` オペレーションは、作成時刻、リージョン、バージョンなど、デプロイされたスタックに関する情報を取得します。このエンドポイントはフロントエンドで使用されます。

応答

200 - 成功

名前	説明
created_time	スタックが作成されたときの ISO 8601 タイムスタンプ (例: 2025-09-09T19:40:22Z)
region	スタックがデプロイされている AWS リージョン (例: us-east-1)
version	デプロイされたソリューションのバージョン (例: v4.0.0)

エラーレスポンス

- 403 - 禁止: スタック情報にアクセスするためのアクセス許可が不十分
- 404 - 見つかりません: スタック情報は利用できません
- 500 - 内部サーバーエラー

GET /scenarios

説明

GET /scenarios の操作を使用すると、テストシナリオの一覧を取得できます。

応答

名前	説明
data	各テストの ID、名前、説明、ステータス、実行時間、タグ、合計実行数、最後の実行を含むシナリオのリスト

POST /scenarios

説明

POST /scenarios の操作では、テストシナリオを作成またはスケジューリングできます。

リクエスト本文

名前	説明
testName	テストの名前
testDescription	テストの説明
testTaskConfigs	シナリオで concurrency (並列実行の数)、taskCount (テストの実行に必要なタスクの数)、および region を指定するオブジェクト
testScenario	テストの同時実行、テスト時間、ホスト、メソッドを含むテスト定義
testType	テストのタイプ (例: simple、jmeter)
fileType	アップロードするファイルのタイプ (例: none、script、zip)
tags	テストを分類するための文字列の配列。最大長が 5 のオプションフィールド (例: ["blue", "3.0", "critical"])
scheduleDate	テストを実行する日付。テストがスケジューリングされている場合にのみ提供されます (例: 2021-02-28)。
scheduleTime	テストを実行する時間。テストがスケジューリングされている場合にのみ提供されます (例: 21:07)。
scheduleStep	スケジューリングプロセスのステップ。定期的なテストがスケジューリングされている場合にのみ提供されます。(使用可能な手順には create と start が含まれます)

名前	説明
cronvalue	定期的なスケジューリングをカスタマイズするための cron 値。使用する場合は、scheduleDate および scheduleTime を省略します。
cronExpiryDate	cron が期限切れになり、無期限に実行されないようにするために必要な日付。
recurrence	スケジューリングされたテストの繰り返し。定期的なテストがスケジューリングされている場合にのみ提供されます。(例: daily、weekly、biweekly、monthly)

応答

名前	説明
testId	テストの一意の ID
testName	テストの名前
status	テストのステータス

OPTIONS /scenarios

説明

OPTIONS /scenarios の操作は、正しい CORS レスポンスヘッダーを持つリクエストのレスポンスを提供します。

応答

名前	説明
testId	テストの一意の ID

名前	説明
testName	テストの名前
status	テストのステータス

GET /scenarios/{testId}

説明

GET /scenarios/{testId} の操作を使用すると、特定のテストシナリオの詳細を取得できます。

パラメータのリクエスト

testId

- テストの一意の ID

  タイプ: 文字列

  必須: はい

latest

- 最新のテストランのみを返すクエリパラメータ。デフォルトは true です。

  タイプ: ブール値

  必須: いいえ

history

- テスト実行履歴をレスポンスに含めるクエリパラメータ。デフォルトは true です。false に設定して履歴を除外します

  タイプ: ブール値

  必須: いいえ

応答

名前	説明
testId	テストの一意の ID
testName	テストの名前
testDescription	テストの説明
testType	実行されるテストのタイプ (例: simple、jmeter)
fileType	アップロードするファイルのタイプ (例: none、script、zip)
tags	テストを分類するための文字列の配列
status	テストのステータス
startTime	最後のテストが開始された日時
endTime	最後のテストが終了した日時
testScenario	テストの同時実行、テスト時間、ホスト、メソッドを含むテスト定義
taskCount	テストの実行に必要なタスクの数
taskIds	テストを実行するためのタスク ID の一覧
results	テストの最終結果
history	過去のテストの最終結果の一覧 (history=false の場合、除外されます)
totalRuns	このシナリオのテスト実行の合計数
lastRun	最後のテスト実行のタイムスタンプ

名前	説明
errorReason	エラーが発生したときに生成されるエラーメッセージ
nextRun	次にスケジューリングされている実行 (例: 2017-04-22 17:18:00)
scheduleRecurrence	テストの繰り返し (例: daily、weekly、biweekly、monthly)

POST /scenarios/{testId}

説明

POST /scenarios/{testId} の操作を使用すると、特定のテストシナリオをキャンセルできます。

リクエストパラメータ

testId

- テストの一意の ID

  タイプ: 文字列

  必須: はい

応答

名前	説明
status	テストのステータス

DELETE /scenarios/{testId}

説明

DELETE /scenarios/{testId} の操作を使用すると、特定のテストシナリオに関連するすべてのデータを削除できます。

リクエストパラメータ

testId

- テストの一意の ID

  タイプ: 文字列

  必須: はい

応答

名前	説明
status	テストのステータス

OPTIONS /scenarios/{testId}

説明

OPTIONS /scenarios/{testId} の操作では、正しい CORS レスポンスヘッダーを持つリクエストのレスポンスを提供します。

応答

名前	説明
testId	テストの一意の ID
testName	テストの名前
testDescription	テストの説明

名前	説明
testType	実行されるテストのタイプ (例: simple、jmeter)
fileType	アップロードするファイルのタイプ (例: none、script、zip)
status	テストのステータス
startTime	最後のテストが開始された日時
endTime	最後のテストが終了した日時
testScenario	テストの同時実行、テスト時間、ホスト、メソッドを含むテスト定義
taskCount	テストの実行に必要なタスクの数
taskIds	テストを実行するためのタスク ID の一覧
results	テストの最終結果
history	過去のテストの最終結果の一覧
errorReason	エラーが発生したときに生成されるエラーメッセージ

GET /scenarios/{testId}/testruns

説明

GET /scenarios/{testId}/testruns オペレーションは、特定のテストシナリオのテスト実行 ID を取得し、オプションで時間範囲でフィルタリングします。latest=true の場合、最新のテスト実行を 1 つだけ返します。

パラメータのリクエスト

testId

- テストシナリオ ID

タイプ: 文字列

必須: はい

`latest`

- 最新のテスト実行 ID のみを返します

タイプ: ブール値

デフォルト: `false`

必須: いいえ

`start_timestamp`

- テスト実行をフィルタリングする開始 ISO 8601 タイムスタンプ (含む)。例: `2024-01-01T00:00:00Z`

タイプ: 文字列 (日時形式)

必須: いいえ

`end_timestamp`

- テスト実行をフィルタリングする終了 ISO 8601 タイムスタンプ (含む)。例: `2024-12-31T23:59:59Z`

タイプ: 文字列 (日時形式)

必須: いいえ

`limit`

- 返されるテスト実行の最大数 (`latest=true` の場合は無視)

タイプ: 整数 (最小: 1、最大: 100)

デフォルト: `20`

必須: いいえ

`next_token`

- 次のページを取得するための前のレスポンスからのページ分割トークン

タイプ: 文字列

必須: いいえ

応答

200 - 成功

名前	説明
testRuns	testRunId (文字列) と startTime (ISO 8601 日時) をそれぞれ含むテスト実行オブジェクトの配列
pagination	limit (整数) と next_token (文字列または null) を含むオブジェクト。結果がない場合、トークンは null です

エラーレスポンス

- 400 - 無効なタイムスタンプ形式またはパラメータ
- 404 - テストシナリオが見つかりません
- 500 - 内部サーバーエラー

使用例

- 最新のテスト実行のみ: `GET /scenarios/test123/testruns?latest=true`
- 時間範囲内の最新: `GET /scenarios/test123/testruns?latest=true&start_timestamp=2024-01-01T00:00:00Z`
- 次のページのリクエスト: `GET /scenarios/test123/testruns?limit=20&next_token=eyJ0ZXN0SWQiOiJzZVFVeTEyTEtMIiwic3RhcnRUaW1lIjoiMjAyNC0wMS`

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

説明

GET /scenarios/{testId}/testruns/{testRunId} オペレーションは、特定のテスト実行の完全な結果とメトリクスを取得します。必要に応じて、応答を高速化するために history=false で履歴結果を省略します。

パラメータのリクエスト

testId

- テストシナリオ ID

  タイプ: 文字列

  必須: はい

testRunId

- 特定のテスト実行 ID

  タイプ: 文字列

  必須: はい

history

- レスポンスに履歴配列を含めます。false に設定し、応答の高速化のために履歴を省略します

  タイプ: ブール値

  デフォルト: true

  必須: いいえ

応答

200 - 成功

名前	説明
testId	テストの一意の ID (例: seQUy12LKL)

名前	説明
testRunId	特定のテスト実行 ID (例: 2DEwHItEne)
testDescription	負荷テストの説明
testType	テストのタイプ (例: simple、jmeter)
status	テスト実行のステータス: complete、running、failed、または cancelled
startTime	テストが開始された日時 (例: 2025-09-09 21:01:00)
endTime	テストが終了した日時 (例: 2025-09-09 21:18:29)
succPercent	成功の割合 (例: 100.00)
testTaskConfigs	region、taskCount 、および concurrency を含むタスク設定オブジェクトの配列
completeTasks	完了したタスク数へのオブジェクトマッピングリージョン
results	avg_lt (平均レイテンシー)、パーセンタイル (p0_0、p50_0、p90_0、p95_0、p99_0、p99_9、p1 (平均応答時間)、avg_ct (平均接続時間)、stdev_rt (標準偏差応答時間)、concurrency 、throughput 、succ (成功数)、fail (失敗数)、bytes、testDuration 、metricS3Location 、rc (応答コード配列) および labels 配列を含む詳細なメトリクスを含むオブジェクト
testScenario	execution 、reporting 、および scenarios プロパティを持つテスト設定を含むオブジェクト

名前	説明
history	テスト結果の履歴の配列 (history=false の場合、除外されます)

エラーレスポンス

- 400 - testId または testRunId が無効です
- 404 - テスト実行が見つかりません
- 500 - 内部サーバーエラー

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

説明

DELETE /scenarios/{testId}/testruns/{testRunId} オペレーションは、特定のテスト実行に関連するすべてのデータとアーティファクトを削除します。テスト実行データは DynamoDB から削除されますが、S3 の実際のテストデータは変更されないままです。

パラメータのリクエスト

testId

- テストシナリオ ID

  タイプ: 文字列

  必須: はい

testRunId

- 削除する特定のテスト実行 ID

  タイプ: 文字列

  必須: はい

応答

204 - 成功

テスト実行は正常に削除されました (コンテンツは返されません)

エラーレスポンス

- 400 - testId または testRunId が無効です
- 403 - 禁止: テスト実行を削除するアクセス許可が不十分
- 404 - テスト実行が見つかりません
- 409 - 競合: テスト実行は現在実行中であり、削除できません
- 500 - 内部サーバーエラー

GET /scenarios/{testId}/baseline

説明

GET /scenarios/{testId}/baseline オペレーションは、シナリオの指定されたベースラインテスト結果を取得します。data パラメータに応じて、ベースラインテスト実行 ID または完全なベースライン結果を返します。

パラメータのリクエスト

testId

- テストシナリオ ID

  タイプ: 文字列

  必須: はい

data

- true の場合は完全なベースラインテスト実行データを返し、それ以外の場合は testRunId のみを返します

  タイプ: ブール値

  デフォルト: false

  必須: いいえ

応答

200 - 成功

data=false の場合 (デフォルト):

名前	説明
testId	テストシナリオ ID (例: seQUy12LKL)
baselineTestRunId	ベースラインテスト実行 ID (例: 2DEwHItEne)

data=true の場合:

名前	説明
testId	テストシナリオ ID (例: seQUy12LKL)
baselineTestRunId	ベースラインテスト実行 ID (例: 2DEwHItEne)
baselineData	完全なテスト実行結果オブジェクト (GET /scenarios/{testId}/testruns/{testRunId} と同じ構造)

エラーレスポンス

- 400 - testId パラメータが無効です
- 404 - テストシナリオが見つからないか、ベースラインが設定されていません
- 500 - 内部サーバーエラー

PUT /scenarios/{testId}/baseline

説明

PUT /scenarios/{testId}/baseline オペレーションは、パフォーマンスの比較のベースラインとして特定のテスト実行を指定します。シナリオごとに設定できるベースラインは 1 つだけです。

パラメータのリクエスト

testId

- テストシナリオ ID

  タイプ: 文字列

  必須: はい

リクエスト本文

名前	説明
testRunId	ベースラインとして設定するテスト実行 ID (例: 2DEwHItEne)

応答

200 - 成功

名前	説明
message	確認メッセージ (例: Baseline set successfully)
testId	テストシナリオ ID (例: seQUy12LKL)
baselineTestRunId	設定されたベースラインテスト実行 ID (例: 2DEwHItEne)

エラーレスポンス

- 400 - testId または testRunId が無効です
- 404 - テストシナリオまたはテスト実行が見つかりません
- 409 - 競合: テスト実行をベースラインとして設定できません (テストの失敗など)
- 500 - 内部サーバーエラー

DELETE /scenarios/{testId}/baseline

説明

DELETE /scenarios/{testId}/baseline オペレーションは、シナリオのベースライン値を空の文字列に設定することでクリアします。

パラメータのリクエスト

testId

- テストシナリオ ID

  タイプ: 文字列

  必須: はい

応答

204 - 成功

ベースラインが正常にクリアされました (コンテンツは返されません)

エラーレスポンス

- 400 - testId が無効です
- 500 - 内部サーバーエラー

GET /tasks

説明

GET /tasks の操作を使用すると、実行中の Amazon Elastic Container Service (Amazon ECS) タスクの一覧を取得できます。

応答

名前	説明
tasks	テストを実行するためのタスク ID の一覧

OPTIONS /tasks

説明

OPTIONS /tasks のタスク操作では、正しい CORS レスポンスヘッダーを持つリクエストのレスポンスを提供します。

応答

名前	説明
taskIds	テストを実行するためのタスク ID の一覧

GET /regions

説明

GET /regions の操作では、そのリージョンでテストを実行するために必要なリージョン別のリソース情報を取得できます。

応答

名前	説明
testId	リージョン ID
ecsCloudWatchLogGroup	リージョン内の Amazon Fargate タスク用の Amazon CloudWatch ロググループの名前
region	テーブル内のリソースが存在するリージョン
subnetA	リージョン内のいずれかのサブネットの ID
subnetB	リージョン内のいずれかのサブネットの ID
taskCluster	リージョン内の AWS Fargate クラスターの名前
taskDefinition	リージョン内のタスク定義の ARN
taskImage	リージョン内のタスクイメージの名前
taskSecurityGroup	リージョン内のセキュリティグループの ID

OPTIONS /regions

説明

OPTIONS /regions の操作は、正しい CORS レスポンスヘッダーを持つリクエストのレスポンスを提供します。

応答

名前	説明
testId	リージョン ID
ecsCloudWatchLogGroup	リージョン内の Amazon Fargate タスク用の Amazon CloudWatch ロググループの名前

名前	説明
region	テーブル内のリソースが存在するリージョン
subnetA	リージョン内のいずれかのサブネットの ID
subnetB	リージョン内のいずれかのサブネットの ID
taskCluster	リージョン内の AWS Fargate クラスターの名前
taskDefinition	リージョン内のタスク定義の ARN
taskImage	リージョン内のタスクイメージの名前
taskSecurityGroup	リージョン内のセキュリティグループの ID

コンテナリソースを増加する

負荷テストでシミュレートできる同時仮想ユーザー (同時実行) の数を増やすには、各 Amazon ECS タスクに割り当てられた CPU とメモリのリソースを増やす必要があります。これには、リソース制限が高い新しいタスク定義リビジョンを作成し、ソリューションの DynamoDB 設定を更新して、将来のテスト実行に新しいタスク定義を使用することが含まれます。

新しいタスク定義のリビジョンを作成する

CPU とメモリのリソースを増やした新しいタスク定義を作成するには、次の手順に従います。

1. Amazon Elastic Container Service コンソールにサインインします。
2. 左側のナビゲーションメニューで、[タスク定義] を選択します。
3. このソリューションに対応するタスク定義の横にあるチェックボックスをオンにします。例: [replaceable]<stackName>`-EcsTaskDefinition-*<system-generated-random-Hash>*。
4. [Create new revision (新しいリビジョンを作成)] を選択します。
5. [新しいリビジョンの作成] ページで、次の操作を実行します。
   a. [タスクサイズ] で、タスクメモリとタスク CPU を希望する値に変更します。値を大きくすると、タスクあたりの同時仮想ユーザー数が増えます。

b. [コンテナ定義] で、ハード/ソフトメモリの制限を確認します。この制限が必要なメモリより低い場合は、コンテナを選択してください。

c. [コンテナの編集] ダイアログボックスで、[メモリ制限] に移動し、ハードリミットをタスクメモリ割り当て以下に更新します。

d. [更新] を選択します。

6. [新しいリビジョンの作成] ページで、[作成] を選択します。
7. タスク定義が正常に作成されたら、バージョン番号を含む完全なタスク定義 ARN を記録します。例: [replaceable]<stackName>`-EcsTaskDefinition-*<system-generated-random-Hash>*:[replaceable]<system-generated-versionNumber>.

DynamoDB テーブルのアップデート

新しいタスク定義リビジョンを作成したら、今後のテスト実行で新しいタスク定義が使用されるように、ソリューションの DynamoDB テーブルを更新する必要があります。更新されたタスク定義を使用する AWS リージョンごとに、これらのステップを繰り返します。

1. [DynamoDB console] (DynamoDB コンソール) に移動します。
2. 左側のナビゲーションペインから、テーブル以下にある [項目の検索] を選択します。
3. このソリューションに関連する scenarios-table DynamoDB テーブルを選択します。例: [replaceable]<stackName>`-DLTTestRunnerStorageDLTScenariosTable-*<system-generated-random-Hash>*。
4. 新しいタスク定義リビジョンを作成したリージョンに対応する項目を選択します。例えば、region-[replaceable]<region-name>`。
5. 項目エディタで taskDefinition 属性を見つけ、前のセクションで記録した完全なタスク定義 ARN (バージョン番号を含む) で値を更新します。
6. [Save changes] (変更の保存) をクリックします。

> **Note**
>
> 更新されたタスク定義は、新しいテスト実行でのみ使用されます。現在実行中またはスケジュールされているテストは、引き続き以前のタスク定義を使用します。

MCP ツールの仕様

分散負荷テストソリューションは、AI エージェントがテストシナリオと結果を操作できるようにする一連の MCP ツールを公開します。これらのツールは、AI エージェントが情報を処理する方法に沿った高レベルの抽象化された機能を提供し、詳細な API 契約ではなく分析とインサイトに集中できるようにします。

> **Note**
>
> すべての MCP ツールは、ソリューションのデータへの読み取り専用アクセスを提供します。MCP インターフェイスを介してのテストシナリオや設定の変更はサポートされていません。

list_scenarios

説明

この `list_scenarios` ツールは、基本的なメタデータを使用して、使用可能なすべてのテストシナリオのリストを取得します。

エンドポイント

```
GET /scenarios
```

パラメータ

なし

[Response] (レスポンス)

名前	説明
`testId`	テストシナリオの一意の識別子
`testName`	テストシナリオの名前
`status`	テストシナリオの現在のステータス

名前	説明
startTime	テストがいつ作成されたか、または最後に実行されたか
testDescription	テストシナリオの説明

get_scenario_details

説明

この get_scenario_details ツールは、単一のテストシナリオのテスト設定と最新のテスト実行を取得します。

エンドポイント

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

リクエストパラメータ

test_id

- テストシナリオの一意の識別子

  タイプ: 文字列

  必須: はい

応答

名前	説明
testTaskConfigs	各リージョンのタスク設定
testScenario	テスト定義とパラメータ
status	現在のテストステータス
startTime	テストの開始タイムスタンプ

名前	説明
endTime	テストの終了タイムスタンプ (完了した場合)

list_test_runs

説明

この list_test_runs ツールは、特定のテストシナリオのテスト実行のリストを取得し、最新のものから古い順にソートします。最大 30 件の結果を返します。

エンドポイント

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

または

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

パラメータのリクエスト

test_id

- テストシナリオの一意の識別子

  タイプ: 文字列

  必須: はい

limit

- 返されるテスト実行の最大数

  タイプ: 整数

  デフォルト: 20

  最大: 30

  必須: いいえ

`start_date`

- 特定の日付から実行をフィルタリングする ISO 8601 タイムスタンプ

  タイプ: 文字列 (日時形式)

  必須: いいえ

`end_date`

- 特定の日付まで実行をフィルタリングする ISO 8601 タイムスタンプ

  タイプ: 文字列 (日時形式)

  必須: いいえ

応答

名前	説明
`testRuns`	各実行のパフォーマンスメトリクスとパーセンタイルを含むテスト実行の概要の配列

get_test_run

説明

この `get_test_run` ツールは、リージョンとエンドポイントの内訳を含む 1 回のテスト実行の詳細な結果を取得します。

エンドポイント

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

パラメータのリクエスト

`test_id`

- テストシナリオの一意の識別子

  タイプ: 文字列

必須: はい

`test_run_id`

- 特定のテスト実行の一意の識別子

  タイプ: 文字列

  必須: はい

応答

名前	説明
`results`	リージョンの結果の内訳、エンドポイント固有のメトリクス、パフォーマンスパーセンタイル (p50、p90、p95、p99)、成功と失敗の数、応答時間とレイテンシー、実行に使用されるテスト設定を含む完全なテスト実行データ

get_latest_test_run

説明

`get_latest_test_run` ツールは、特定のテストシナリオの最新のテスト実行を取得します。

エンドポイント

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

> **Note**
>
> 結果はグローバルセカンダリインデックス (GSI) を使用して時間別にソートされ、最新のテスト実行が返されます。

リクエストパラメータ

`test_id`

- テストシナリオの一意の識別子

  タイプ: 文字列

  必須: はい

応答

名前	説明
`results`	`get_test_run` と同じ形式の最新のテスト実行データ

get_baseline_test_run

説明

`get_baseline_test_run` ツールは、特定のテストシナリオのベーステストテスト実行を取得します。ベースラインはパフォーマンスを比較する目的で使用されます。

エンドポイント

`GET /scenarios/<test_id>/baseline`

リクエストパラメータ

`test_id`

- テストシナリオの一意の識別子

  タイプ: 文字列

  必須: はい

応答

名前	説明
baselineData	指定されたベースライン実行のすべてのメトリクスと設定を含む、比較目的のベースラインテスト実行データ

get_test_run_artifacts

説明

get_test_run_artifacts ツールは、ログ、エラーファイル、結果などのテストアーティファクトにアクセスするための Amazon S3 バケット情報を取得します。

エンドポイント

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

パラメータのリクエスト

test_id

- テストシナリオの一意の識別子

  タイプ: 文字列

  必須: はい

test_run_id

- 特定のテスト実行の一意の識別子

  タイプ: 文字列

  必須: はい

応答

名前	説明
bucketName	アーティファクトが保存されている S3 バケット名
testRunPath	現在のアーティファクトストレージのパスプレフィックス (バージョン 4.0 以上)
testScenarioPath	レガシーアーティファクトストレージのパスプレフィックス (バージョン 4.0 より前)

> **Note**
>
> すべての MCP ツールは、既存の API エンドポイントを活用します。MCP 機能をサポートするのに、基盤となる API を変更する必要はありません。

参照資料

このセクションには、データ収集、関連リソースへのポインタ、このソリューションに貢献したビルダーのリストに関する情報が含まれています。

データ収集

このソリューションは、このソリューションの使用に関するオペレーションメトリクスを AWS (「データ」) に送信します。AWS ではこのデータを使用して、ユーザーがこのソリューション、関連サービスおよび製品をどのように使用しているかをよりよく理解し、提供するサービスや製品の改善に役立てます。AWS によるこのデータの収集には、AWS プライバシー通知が適用されます。

寄稿者

- 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

用語集

この用語集は、AWS での分散負荷テスト実装ガイド全体で使用されている頭字語と略語を定義しています。

技術的なプロトコルと形式

AGPL

Affero General Public License。K6 で使用されるオープンソースソフトウェアライセンス。

API

アプリケーションプログラミングインターフェイス。ソフトウェアアプリケーションを構築し、異なるシステム間の通信を可能にするための一連のプロトコルとツール。

CLI

コマンドラインインターフェイス。ソフトウェアやオペレーティングシステムを操作するためのテキストベースのインターフェイス。

CORS

クロスオリジンリソース共有。あるオリジンで実行されているウェブアプリケーションが別のオリジンのリソースにアクセスすることを許可または制限するセキュリティ機能。

CSV

カンマ区切り値。表形式のデータをプレーンテキストで保存するために使用するファイル形式。一般的にデータのエクスポートに使用されます。

gRPC

gRPC リモートプロシージャの呼び出し。リモートプロシージャの呼び出し用の高性能のオープンソースフレームワーク。

HTTP

ハイパーテキスト転送プロトコル。World Wide Web でデータを送信するために使用する基本プロトコル。

HTTPS

HTTP Secure。ネットワークを介した安全な通信の暗号化に使用する HTTP の拡張。

JSON

JavaScript Object Notation。人間が読み書きしやすく、マシンが解析して生成しやすい軽量のデータ交換形式。

JWT

JSON ウェブトークン。認証と認可のために二者間で転送されるクレームを表す、コンパクトで URL セーフな方法。

OAuth

認可を開きます。一般的にトークンベースの認証と認可に使用されるアクセス委任のオープンスタンダード。

REST

Representational State Transfer。ステートレスな通信と標準の HTTP メソッドを使用してネットワークアプリケーションを設計するためのアーキテクチャスタイル。

SSE

Server-Sent Events。クライアントが HTTP 接続を介してサーバーから自動更新を受信できるようにするサーバープッシュテクノロジー。

UI

ユーザーインターフェイス。ユーザーがソフトウェアアプリケーションを操作するビジュアル要素とコントロール。

URL

Uniform Resource Locator。インターネット上のリソースへのアクセスに使用されるアドレス。

XML

拡張マークアップ言語。人間が読み取れる形式と機械が読み取れる形式でドキュメントをエンコードするためのルールを定義するマークアップ言語。

テストとデータベースの用語

FTP

File Transfer プロトコル。クライアントとサーバー間でファイルを転送するために使用される標準ネットワークプロトコル。

GSI

グローバルセカンダリインデックス。代替キーを使用してデータをクエリできる DynamoDB 機能。

JDBC

Java Database Connectivity。データベースに接続してクエリを実行するための Java API。

JMS

Java Message Service。2 つ以上のクライアント間でメッセージを送信するための Java API。

TPS

1 秒あたりのトランザクション数。システムが 1 秒間に処理できるトランザクションの数の測定値。

AWS およびシステムの用語

ARN

Amazon リソースネーム。AWS のサービス全体でリソースを指定するために使用される AWS リソースの一意の識別子。

ISO

国際標準化機構。国際標準を開発する独立した非政府組織。ISO 8601 タイムスタンプ形式について、このガイドで参照されています。

SLA

サービスレベルアグリーメント。予想されるサービスのレベルを定義するサービスプロバイダーと顧客間のコミットメント。

UUID

Universally Unique Identifier。コンピュータシステム内の情報を一意に識別するために使用される 128 ビットの番号。

vCPU

仮想中央処理装置。仮想マシンまたはコンテナに割り当てられた仮想プロセッサ。物理 CPU の処理能力の一部を表します。

負荷テストの用語

同時実行

タスクごとの同時仮想ユーザーの数。このパラメータは、負荷テスト中に各 Fargate タスクが生成するシミュレートされたユーザーの数を制御します。

リージョン用のスタック

マルチリージョン負荷テスト用にテストインフラストラクチャを提供するために AWS リージョンにデプロイされた CloudFormation スタック。

タスク数

テストシナリオを実行するために起動された Fargate コンテナ (タスク) の数。生成された合計負荷は、タスク数に同時実行数を掛けた値に等しくなります。

テストシナリオ

テストタイプ、ターゲットエンドポイント、タスク数、同時実行数、期間、その他のパラメータを含む、設定済みの負荷テスト。

改訂

GitHub リポジトリの CHANGELOG.md にアクセスして、バージョン固有の改善と修正を追跡します。

注意

お客様は、この文書に記載されている情報を独自に評価する責任を負うものとします。本書は、(a) 情報提供のみを目的とし、(b) AWS の現在の製品と慣行について説明しており、これらは予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤー、またはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または黙示を問わず、一切の保証、表明、条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は AWS 契約によって規定されています。また、本文書は、AWS とお客様との間の契約に属するものではなく、また、当該契約が本文書によって修正されることもありません。

AWS での分散負荷テストは、Apache Software Foundation で閲覧可能な Apache ライセンスバージョン 2.0 の条項に基づいてライセンスされます。