AWS での分散負荷テスト

実装ガイド

2019 年 11 月

最終更新日: 2022 年 8 月 (改訂を参照)

目次

はじめに

「AWS での分散負荷テスト」ソリューションでは、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および負荷時の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、一定のペースでトランザクションレコードを生成する数多くの接続ユーザーを作成および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、すべてのシミュレーションを実行でき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、テスト対象のソフトウェアの負荷能力をテストできます。
- 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、継続的なペースでトランザクションレコードを生成します。
- カスタマイズ可能な JMeter スクリプトを使用して、アプリケーションのテストをカスタマイズします。
- 未来の日付または定期的な日付に負荷テストが自動的に開始されるようにスケジュールを組みます。
- アプリケーションのロードテストを同時に実行するか、複数のテストを同時に実行します。

この実装ガイドでは、アマゾン ウェブ サービス (AWS) クラウドに「AWS での分散負荷テスト」ソリューションをデプロイするためのアーキテクチャ上の考慮事項と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。

このガイドは、AWS クラウドにおけるアーキテクチャの設計の実務経験がある 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を対象としています。

コスト

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。このソリューションを実行するための合計コストは、実行する負荷テストの数、それらの負荷テストの所要時間、およびテストの一部として使用されるデータの量によって異なります。2022 年 8 月の時点で、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するための推定コストは、**1 か月あたり約 30.90 USD** です。コストの見積もりは、次の要因を想定しています。

AWS サービス	ディメンション	コスト [USD]
AWS Fargate	10 個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**

重要: バージョン 1.3.0 以降、CPU は 2 つの vCPU に増加し、メモリは 4 GB に増加しました。これらの変更により、このソリューションの以前のバージョンと比較して推定コストが増加します。AWS リソースに対する負荷テストでこれらの増加を必要としない場合は、減らすことができます。詳細については、このガイドの「コンテナのリソース追加」セクションを参照してください。

注意: 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。

料金は変更される可能性があります。詳細については、このソリューションで使用する AWS の各サービスの料金表ウェブページを参照してください。

アーキテクチャの概要

このソリューションをデフォルトのパラメータでデプロイすると、AWS クラウドに次の環境が構築されます。

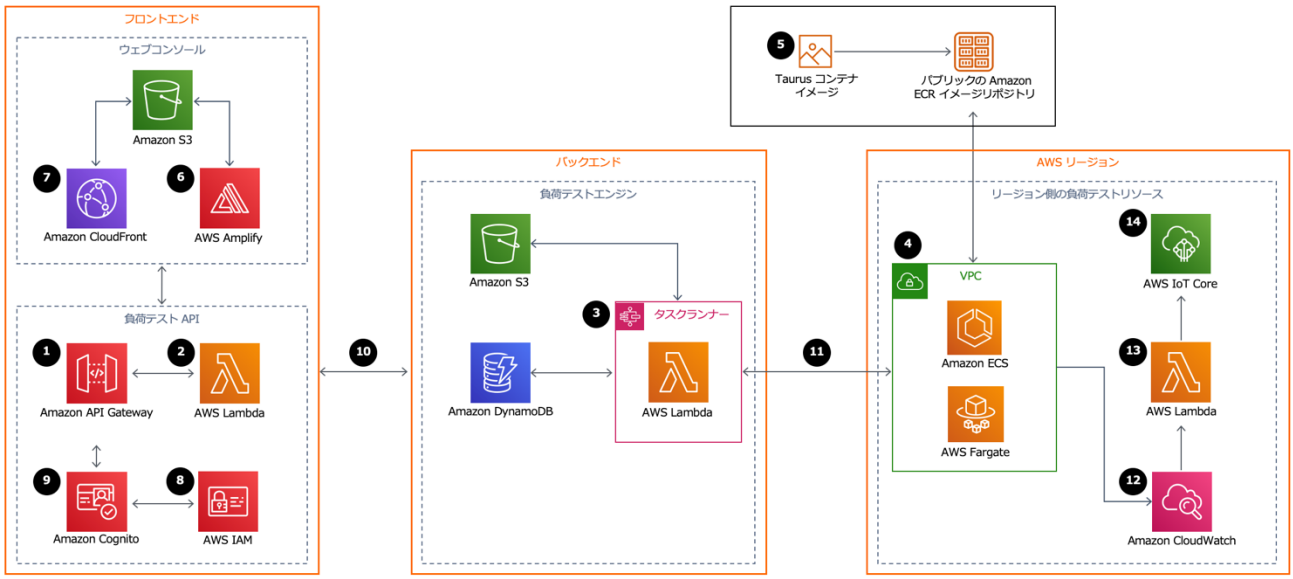

図 1: 「AWS での分散負荷テスト」ソリューションのアーキテクチャ

AWS CloudFormation テンプレートでは、次のリソースをデプロイします。

1. `distributed load tester` API は、Amazon API Gateway を利用してこの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 と通信し、テストシナリオを実行して、そのテストシナリオの詳細と結果用のストレージを提供します。
4. AWS Fargate で実行中の 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をデプロイします。

5. このコンテナには、アプリケーションのパフォーマンステスト用の負荷を生成する Taurus 負荷テストオープンコンテナ規格 (OCI) に準拠しているコンテナイメージが含まれています。Taurus は、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、AWS が Amazon Elastic Container Registry (Amazon ECR) のパブリックリポジトリでホストしています。Amazon ECR イメージリポジトリの詳細については、「コンテナイメージのカスタマイズ」を参照してください。
6. AWS Amplify を利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した Amazon 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. ソリューションは初期設定時にデフォルトのソリューション管理者ロール (IAM ロール) を作成し、指定されたユーザーの 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、コンソールと `distributed load tester` API 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後、ウェブコンソールを使用して、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 AWS リージョンで AWS Fargate 上で Amazon ECS のタスクを実行します。
12. Amazon S3 と Amazon DynamoDB にその結果を保存することに加えて、テストが完了したら、その出力は Amazon CloudWatch に記録されます。
13. ライブデータオプションを選択すると、このソリューションは、テストが実行された AWS リージョンごとに、テスト中に AWS Fargate タスクの Amazon CloudWatch Logs を AWS Lambda 関数に送信します。
14. AWS Lambda 関数は、メインスタックがデプロイされた AWS リージョンの AWS IoT Core の対応するトピックにデータを発行します。ウェブコンソールはそのトピックをサブスクライブし、ウェブコンソールでテスト実行中にデータを確認することができます。

ソリューションコンポーネント

「AWS での分散負荷テスト」ソリューションは、フロントエンドとバックエンドの 2 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、このソリューションのバックエンドとやり取りするために使用する負荷テスト API とウェブコンソールで構成されます。

負荷テスト API

「AWS での分散負荷テスト」ソリューションでは、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と RESTful API を使用して、安全にテストデータとやり取りできます。この API は、Amazon DynamoDB に保存されたデータにアクセスするための「フロントドア」のように動作します。また、この API を使用して、この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。オペレーターが正常に認証されると、Amazon Cognito は、コンソールからこのソリューションの 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、コンソールからトークンを含む認証ヘッダーとともにスマートプロダクトソリューション API に送信されます。

リクエストに基づいて、Amazon API Gateway は適切な AWS Lambda 関数を呼び出して、Amazon DynamoDB テーブルに保存されてい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。

このソリューションの API の詳細については、このガイドの「分散負荷テストの 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。このコンソールは、Amazon S3 でホストされ、Amazon CloudFront を介してアクセスされる ReactJS アプリケーションです。このアプリケーションは AWS Amplify を使用して、Amazon Cognito と統合し、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。

ウェブコンソールでは、この負荷テストソリューションとどのようにやり取りできるのかをデモするために設計されています。本番環境では、独自のニーズに合わせてこのウェブコンソールをカスタマイズするか、ユースケースに合わせて独自のコンソールを構築することをお勧めします。

ウェブコンソールの URL は Amazon CloudFront ディストリビューションドメイン名で、AWS CloudFormation の出力で **Console** として確認できます。AWS 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。

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 のタスクは、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 のタグは、このソリューションのコストを監視するために使用できます。詳細については、 *AWS Billing and Cost Management* ユーザーガイドの「ユーザー定義のコスト配分タグ」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、Taurus 負荷テストフレームワークのコンテナイメージを使用します。このイメージは Amazon Elastic Container Registry (Amazon ECR) のパブリックリポジトリでホストされます。このイメージは、AWS Fargate クラスターで Amazon ECS のタスクを実行するために使用されます。

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。

テストのインフラストラクチャー

このソリューションでは、メインのテンプレートに加えて、複数の AWS 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補助的なテンプレートを作成します。そのテンプレートは Amazon S3 に保存され、テンプレートへのリンクはウェブコンソールに表示されます。この補助的なテンプレートは、VPC、AWS Fargate クラスター、ライブデータを処理するための AWS Lambda 関数を作成します。

セカンダリリージョンの起動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

「AWS での分散負荷テスト」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 AWS リージョンにまたがる数千の接続ユーザーでシミュレートし、1 秒間に特定の数のトランザクションを生成します。

付属のウェブコンソールを使用して、テストの一部として実行されるタスクのパラメータを定義します。このソリューションでは、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。

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、`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「ワークフローのテスト」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「テスト結果」セクションを参照してください。キャンセルワークフローのテストの詳細については、このガイドの「キャンセルワークフローのテスト」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、このソリューションは、その AWS リージョンの AWS Fargate タスクに対応する Amazon CloudWatch Logs で、AWS リージョンごとに AWS Lambda の `real-time-data-publisher` 関数を起動します。その後、この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 AWS リージョン内の AWS IoT Core のトピックに発行します。詳細については、このガイドの「ライブデータ」セクション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場合、セキュリティ上の責任はお客様と AWS の間で共有されます。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システムの物理的なセキュリティに至るまでのコンポーネントについて、AWS が運用、管理、および制御します。そのため、お客様の運用上の負担を軽減するのに役立ちます。AWS の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。

AWS IAM ロール

AWS Identity and Access Management (IAM) ロールにより、AWS 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、いくつかの IAM ロールが作成されます。それには、このソリューションの AWS Lambda 関数にこのソリューションで使用する他のサービスへのアクセスを許可するロールが含まれます。

Amazon CloudFront

このソリューションでは、Amazon S3 バケットでホストする静的ウェブサイトをデプロイします。レイテンシーを減らし、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 Amazon CloudFront ディストリビューションが含まれています。これは、このソリューションのウェブサイトの S3 バケットにあるコンテンツにセキュアなパブリックアクセスを提供するのに役立つ特別な CloudFront ユーザーです。詳細については、「オリジンアクセスアイデンティティを使用して Amazon S3 コンテンツへのアクセスを制限する」を参照してください。

AWS Fargate セキュリティグループ

このソリューションは、デフォルトで AWS Fargate セキュリティグループのアウトバウンドルールは公開設定となっています。AWS Fargate がどこにでもトラフィックを送信することをブロックしたい場合は、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からポート 50,000 へ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、コンテナが相互に通信できるようにするために使用されます。

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。このポリシーは、Amazon EC2 インスタンスから、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の他のロケーションに向けて、ボリュームの大きなネットワークテストの実行を計画している場合などの状況を対象としています。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。テストを行う大抵のユーザーは、このポリシーに該当することはありません。ただし、1 Gbps (10 億ビット / 秒) または 1 Gpps (10 億パケット / 秒) を超えて、合計で 1 分以上維持するトラフィックを生成すると思われる場合は、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

AWS 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では、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 ルールが自動的にデプロイされます。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された保護機能から選択できます。

設計に関する考慮事項

サポートし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アプリケーションをサポートします。このソリューションは、HTTP または HTTPS のいずれかを使用する API をサポートします。また、HTTP リクエストヘッダーを制御できるので、トークンまたは API キーを渡すための認証ヘッダーまたはカスタムヘッダーを追加できます。

JMeter スクリプトのサポート

このソリューションのユーザーインターフェイス (UI) を使用してテストシナリオを作成する場合は、JMeter テストスクリプトを使用できます。JMeter スクリプトファイルを選択すると、`<stack-name>-scenariosbucket` Amazon Simple Storage Service (Amazon S3) バケットにアップロードされます。AWS Fargate タスクで Amazon Elastic Container Service (Amazon ECS) タスクが実行されている場合は、JMeter スクリプトは `<stack-name>-scenariosbucket` Amazon S3 バケットからダウンロードされ、テストが実行されます。

JMeter の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圧縮できます。テストシナリオの作成時に、zip ファイルを選択できます。

プラグインを含める場合は、バンドルされた zip ファイルの `/plugins` サブディレクトリに含まれるすべての `.jar` ファイルが JMeter の拡張ディレクトリにコピーされ、負荷テストに使用できるようになります。

注意: JMeter の入力ファイルを 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。さらに付け加えると、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している場合は、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ることができないため、テストは失敗します。

JMeter のプラグインを含めると、`.jar` ファイルは、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。

テストのスケジューリング

未来の日付で実行するようにスケジュールするか、**Run Now** オプションを使用して、テストをスケジュールできます。テストを未来の日付で 1 回限りの実行としてスケジュールすることも、初回実行日と繰り返しの計画を指定して定期テストを設定することもできます。繰り返しを指定するオプションには、毎日、毎週、隔週、月次、が含まれます。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「スケジューリングワークフローのテスト」セクションを参照してください。

負荷テストの制限

AWS Fargate の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、AWS リージョンごとで各アカウントにつき 1,000 件です。すべてのアカウントがデフォルトで、この制限をサポートしているわけではありません。アカウントの特定のサービスクォータを確認してください。詳細については、「Amazon ECS のサービスクォータ」を参照してください。サービスの上限引き上げ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 のサービスクォータ」を参照してください。

Taurus での負荷テスト用のコンテナイメージは、タスクごとの同時接続を制限していませんが、無制限のユーザー数をサポートできるわけではありません。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認するには、このガイドの「ユーザー数の決定」セクションを参照してください。

注意: デフォルト設定に基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には各テストの Amazon CloudWatch ダッシュボードが含まれており、Amazon ECS クラスターでそのテストで実行されるすべてのタスクの合計出力をリアルタイムで表示します。Amazon CloudWatch の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。各メトリクスは秒単位で集計され、ダッシュボードは 1 分毎に更新されます。

Amazon EC2 のテストポリシー

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」をご参照ください。

ユーザー管理

初期設定時に、この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 Amazon Cognito が使用するユーザー名と E メールアドレスを指定します。このコンソールは、ユーザー管理を提供しません。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* 開発者ガイドの「ユーザープールのユーザーの管理」を参照してください。

デプロイ可能な AWS リージョン

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、Amazon Cognito の使用できるリージョンでこのソリューションを起動する必要があります。AWS リージョンで利用可能な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」をご参照ください。

ソリューションアップデート

リソースのデプロイが変更されたため、AWS CloudFormation コンソールを使用して以前のバージョンからバージョン 3.0.0 にアップデートすることはできません。バージョン 3.0.0 を使用するには、以前のバージョンからデータを移行する代わりに、AWS CloudFormation テンプレートのバージョン 3.0.0 を使用して新しいスタックを作成し、テストをこの最新バージョンに移行する必要があります。テストを移行したら、このソリューションの以前のバージョンをアンインストールできます。

AWS CloudFormation テンプレート

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」ソリューション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。

テンプレートを表示

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、Amazon Elastic Container Service (Amazon ECS)、AWS Fargate、Amazon Virtual Private Cloud (Amazon VPC)、AWS Lambda、Amazon Simple Storage Service (Amazon S3)、AWS Step Functions、Amazon DynamoDB、Amazon CloudWatch Logs、Amazon API Gateway、Amazon Cognito、AWS Identity and Access Management (IAM)、Amazon CloudFront がデプロイされますが、特定のネットワークニーズに基づいてテンプレートをカスタマイズすることもできます。

自動デプロイ

自動デプロイを開始する前に、このガイドで説明されているアーキテクチャ、およびその他の考慮事項をよくお読みください。このセクションの手順に従って、AWS で分散負荷テストを設定してアカウントにデプロイします。

デプロイ時間：約 15 分

スタックの起動

> **重要:** このソリューションには、匿名の運用メトリクスを AWS に送信するオプションが含まれています。当社はこのデータを使用して、お客様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 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 プライバシーポリシーが適用されます。
>
> この機能を無効にするには、テンプレートをダウンロードして、AWS CloudFormation のマッピングセクションを変更し、AWS CloudFormation コンソールを使用してテンプレートをアップロードし、このソリューションをデプロイします。詳細については、このガイドの「運用メトリックの収集」セクションを参照してください。

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」ソリューションをデプロイします。

> **注意:** 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、本ガイドの「コスト」セクション、および本ソリューションで使用する各 AWS のサービスの料金ページを参照してください。

1. AWS マネジメントコンソールにサインインし、右側のボタンをクリックして、`distributed-load-testing-on-aws` AWS CloudFormation テンプレートを起動します。

   ソリューションの起動

   または、独自にカスタマイズするためにテンプレートをダウンロード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。

> **注意:** 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、このサービスを提供している AWS リージョンでこのソリューションを開始する必要があります。AWS リージョンで利用可能な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」をご参照ください。

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、この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
Admin Name	<入力が必須>	最初のソリューション管理者のユーザー名。
Admin Email	<入力が必須>	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	<入力は任意>	使用したい Amazon VPC があり、すで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 AWS リージョンにある既存の Amazon VPC の ID を入力してください。(例: `vpc-1a2b3c4d5e6f`)
First existing subnet	<入力は任意>	既存の Amazon 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	<入力は任意>	既存の Amazon 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`)
AWS Fargate VPC CIDR Block	`192.168.0.0/16`	既存の Amazon VPC の値を指定しない場合は、この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。
AWS Fargate Subnet A CIDR Block	`192.168.0.0/20`	既存の Amazon VPC の値を指定しない場合は、CIDR ブロックには Amazon VPC サブネット A の IP アドレスが使用されます。

パラメータ	デフォルト	説明
AWS Fargate Subnet B CIDR Block	`192.168.16.0/20`	既存の Amazon VPC の値を指定しない場合は、CIDR ブロックには Amazon VPC サブネット B の IP アドレスが使用されます。
AWS Fargate Security Group CIDR Block	`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。

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、設定を確認します。テンプレートが IAM リソースを作成することを承認するチェック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。

   スタックのステータスは、AWS CloudFormation コンソールの**ステータス**列で確認できます。約 15 分で CREATE_COMPLETE ステータスが表示されます。

> **注意 :** 主要な AWS Lambda 関数に加えて、このソリューションには `custom-resource` AWS 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` AWS 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、この関数を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: 約 15 分

複数の AWS リージョンでテストの実行が可能です。「AWS での分散負荷テスト」ソリューションをデプロイすると、セカンダリリージョン用のスタックが作成され、Amazon S3 の `scenarios` バケットに保存されます。マルチリージョンデプロイを実行するには、Amazon S3 の `scenarios` バケットに保存されているリージョン用の AWS CloudFormation テンプレートを、テストを実行したい AWS

リージョンにデプロイする必要があります。リージョン用のテンプレートは、次の方法でインストールできます。

1. このソリューションのウェブコンソールで、トップメニューにある **Manage Regions** 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 AWS 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、AWS リージョンを選択します。
4. **スタックの作成**ページで、**Amazon S3 URL** テキストエリアにリンクを貼り付け、[**次へ**] を選択します。
5. **スタックの詳細を指定**ページで、この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	<入力は任意>	使用したい Amazon VPC があり、すで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 AWS リージョンにある既存の Amazon VPC の ID を入力してください。(例: `vpc-1a2b3c4d5e6f`)
First existing subnet	<入力は任意>	既存の Amazon 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	<入力は任意>	既存の Amazon 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
AWS Fargate VPC CIDR Block	`192.168.0.0/16`	既存の Amazon VPC の値を指定しない場合は、この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。
AWS Fargate Subnet A CIDR Block	`192.168.0.0/20`	既存の Amazon VPC の値を指定しない場合は、CIDR ブロックには Amazon VPC サブネット A の IP アドレスが使用されます。
AWS Fargate Subnet B CIDR Block	`192.168.16.0/20`	既存の Amazon VPC の値を指定しない場合は、CIDR ブロックには Amazon VPC サブネット B の IP アドレスが使用されます。
AWS Fargate Security Group CIDR Block	`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チェックボックスを必ずオンにします。
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。

    スタックのステータスは、AWS CloudFormation コンソールの**ステータス**列で表示できます。約 5 分で CREATE_COMPLETE ステータスが表示されます。

AWS 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、新しい AWS リージョンが **Manage Regions** のモーダルに表示されます。テスト作成時にこの AWS リージョンを選択すると、テストで使用できます。このソリューションでは、scenarios テーブルで起動された AWS リージョンごとに Amazon 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 AWS リージョンごとに並べ替えることができます。API の制約により、マルチリージョンテストのすべての AWS リージョンの集計結果は、Amazon CloudWatch メトリックスでグラフ化することによってのみ表示できます。テストが終了すると、テスト結果でグラフのソースコードを確認できます。

注意: 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 AWS 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 AWS 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。

その他のリソース

AWS のサービス

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

その他のリソース

- Taurus

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 が管理している Amazon Elastic Container Registry (Amazon ECR) コンテナイメージのパブリックリポジトリを使用して、設定されたテストの実行に使用される Taurus 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、独自の AWS アカウントの Amazon ECR イメージリポジトリに保存できます。

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=v3.0.0 # replace with the container image
tag if you want to use a different container image
```

コンテナイメージのカスタマイズを選択する場合は、プライベートリポジトリまたは AWS アカウントのパブリックリポジトリのいずれかでホストできます。イメージのリソースは、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。

- Amazon ECR のプライベート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の概念」と「プライベートイメージ」を参照してください。

- Amazon ECR のパブリック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 public repositories」と「Public images」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, v2.0.0
```

次の例は、コンテナファイルを示しています。

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

コンテナファイルに加えて、ディレクトリには、Taurus を実行する前に Amazon S3 からテスト設定をダウンロードする次の bash スクリプトが含まれています。

```
#!/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}"
```

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

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

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

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

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

    # copy bundled plugin jars to jmeter extension folder to make
them available to jmeter
    BUNDLED_PLUGIN_DIR=`find $PWD -type d -name "plugins" | head
-n 1`
    # attempt to copy only if a /plugins folder is present in
upload
    if [ -z "$BUNDLED_PLUGIN_DIR" ]; then
      echo "skipping plugin installation (no /plugins folder in
upload)"
    else
      # ensure the jmeter extensions folder exists
      JMETER_EXT_PATH=`find ~/.bzt/jmeter-taurus -type d -name
"ext"`
      if [ -z "$JMETER_EXT_PATH" ]; then
```

```
        # fail fast - if plugins bundled they will be needed for
the tests
        echo "jmeter extension path (~/.bzt/jmeter-taurus/**/ext)
not found - cannot install bundled plugins"
        exit 1
      fi
      cp -v $BUNDLED_PLUGIN_DIR/*.jar $JMETER_EXT_PATH
    fi
  fi
fi

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

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

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

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

  files=(`cat results.txt`)
  for f in "${files[@]}"; do

p="s3://$S3_BUCKET/results/$TEST_ID/JMeter_Result/$PREFIX/$UUID/$
f"
    if [[ $f = /* ]]; then
```

```
p="s3://$S3_BUCKET/results/$TEST_ID/JMeter_Result/$PREFIX/$UUID$f
"
    fi

    echo "Uploading $p"
    aws s3 cp $f $p
  done
fi

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

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

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

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。

ワークフローのテスト

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。

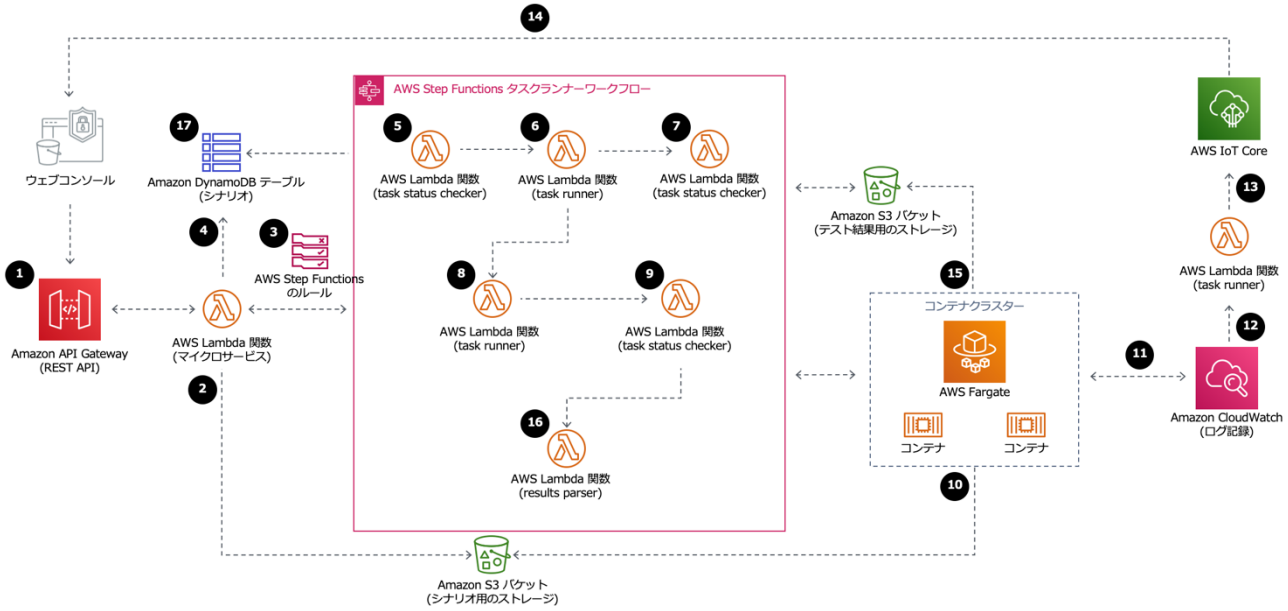

図 2: ワークフローのテスト

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 ステートマシンの入力として使用して実行します。テストがスケジュールされている場合は、最初に AWS CloudWatch Events ルールが作成され、その後、指定した日付に 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「スケジューリングワークフローのテスト」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 関数は、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わかった場合に、エラーが発生します。AWS Fargate クラスターで実行中の Amazon ECS タスクがない場合に、この関数はテスト 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 個のタスクに制限されています。タスク数が 10 件を超える場合は、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 関数で現行のテストを区別するためのプレフィックスを生成します。
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 で実行され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されている場合は、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、`task-runner` AWS Lambda 関数に引き渡されます。
8. `task-runner` AWS Lambda 関数は再び実行され、今回はリーダーノードとして機能する単一の Amazon ECS タスクを起動します。この Amazon 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 で実行されているかどうかを再度確認します。タスクがまだ実行されている場合は、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、平均応答時間、同時ユーザー数、成功したリクエストの数、各タスクに対して失敗したリクエストの数が Amazon CloudWatch に記録され、Amazon CloudWatch の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、この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 Amazon 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、そのデータを AWS Lambda 関数に引き渡します。
13. AWS Lambda 関数は、受信したデータを構造化して AWS IoT Core トピックに発行します。
14. ウェブコンソールはテスト用の AWS IoT Core トピックをサブスクライブし、そのトピックに発行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、ファイル名として UUID が割り当てられます。(例: `s3://dlte-bucket/test-scenarios/<$TEST_ID>/results/<$prefix>-<$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。

テスト結果

「AWS での分散負荷テスト」ソリューションでは、負荷テストフレームワークを使用して、大規模なアプリケーションテストを実行します。テストが完了すると、次の結果を含む詳細なレポートが生成されます。

- **Average response time -** 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)
- **Average latency -** 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)

- **Average connection time -** 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)
- **Average bandwidth -** 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
- **Requests Per Second -** 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
- **Percentile -** テストの応答時間のパーセンタイル値 (最大応答時間は 100%、最小応答時間は 0%)

Taurus のテスト結果の詳細については、*Taurus* ユーザーマニュアルの「Generating Test Reports」を参照してください。

分散負荷テストの API

「AWS での分散負荷テスト」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。この API は、Amazon DynamoDB に保存されたデータにアクセスするための「フロントドア」のように動作します。また、この API を使用して、この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、識別と承認を行います。この 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、ユーザープールが有効なメソッドを呼び出すことができます。

API を使用して直接テストを実行する方法の詳細については、Amazon API Gateway の「Signing Requests」ドキュメントを参照してください。

このソリューションの API では、次のオペレーションを使用できます。

> **注意:** `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。

シナリオ

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

タスク

- GET /tasks
- OPTIONS /tasks

リージョン

- GET /regions
- OPTIONS /regions

GET /scenarios

説明

`GET /scenarios` オペレーションを使用すると、テストシナリオの一覧を取得できます。

レスポンス

名前	説明
`data`	各テストの ID、名前、説明、ステータス、実行時間を含むシナリオのリスト

POST /scenarios

説明

`POST /scenarios` オペレーションでは、テストシナリオを作成またはスケジュールできます。

リクエストボディ

名前	説明
`testName`	テストの名前
`testDescription`	テストの説明
`taskCount`	テストの実行に必要なタスクの数
`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`)
`scheduleDate`	テストを実行する日付。テストがスケジュール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ルされている場合にのみ提供されます (例: `21:07`)。
`scheduleStep`	スケジュールプロセスのステップ。定期的なテストがスケジュール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)
`recurrence`	スケジュールしたテストの繰り返し。定期的なテストがスケジュールされている場合にのみ提供されます。(例: `daily`、`weekly`、`biweekly`、`monthly`)

レスポンス

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` オペレーションは、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。

レスポンス

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

GET /scenarios/{testId}

説明

`GET /scenarios/{testId}` オペレーションを使用すると、特定のテストシナリオの詳細を取得できます。

リクエストパラメーター

`testId`

テストの一意の ID

タイプ: 文字列

必須：はい

レスポンス

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`	実行するテストのタイプ (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
`nextRun`	次にスケジュールされている実行 (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}` オペレーションを使用すると、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ー

`testId`

テストの一意の ID

タイプ: 文字列

必須：はい

レスポンス

名前	説明
`status`	テストのステータス

DELETE /scenarios/{testId}

説明

`DELETE /scenarios/{testId}` オペレーションを使用すると、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ー

`testId`

テストの一意の ID

タイプ: 文字列

必須：はい

レスポンス

名前	説明
`status`	テストのステータス

OPTIONS /scenarios/{testId}

説明

`OPTIONS /scenarios/{testId}` オペレーションは、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。

レスポンス

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`	実行するテストのタイプ (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 /tasks

説明

`GET /tasks` オペレーションを使用すると、実行中の Amazon Elastic Container Service (Amazon ECS) タスクの一覧を取得できます。

レスポンス

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` オペレーションは、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。

レスポンス

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` オペレーションでは、そのリージョンでテストを実行するために必要な AWS リージョンのリソース情報を取得できます。

レスポンス

名前	説明
testId	AWS リージョンの 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 AWS 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` オペレーションは、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。

レスポンス

名前	説明
testId	AWS リージョンの 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

名前	説明
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 Amazon 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

スケジューリングワークフローのテスト

ウェブコンソールを使用して、負荷テストをスケジュールします。テストをスケジュールすると、次のワークフローが実行されます。

- スケジュールするオプションを使用して負荷テストを作成すると、スケジュールパラメータが Amazon API Gateway 経由でこのソリューションの API に送信されます。
- 次に、この API はパラメータを AWS Lambda 関数にひき渡し、Amazon CloudWatch Events ルールを作成して、指定された日付で実行するようにスケジュールします。
- このテストが 1 回限りのテストである場合は、Amazon CloudWatch Events ルールは指定された日付に実行されます。`api-services` AWS Lambda 関数は、このガイドの「ワークフローのテスト」セクションで指定されたワークフローを通じて新しいテストを実行します。
- このテストが定期的なテストである場合は、Amazon CloudWatch Events ルールは指定された日付に実行されます。`api-services` AWS Lambda 関数が実行されると、現在の Amazon CloudWatch Events ルールが削除され、作成するとすぐに実行される別のルールを作成します。その後、指定された繰り返し頻度に基づいて定期的に実行されます。

ユーザー数の決定

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

ポートできるユーザーの最大数に達します。次の例を使用して、テストの同時ユーザー制限を決定できます。

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、Amazon 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 Clusters**] 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、このガイドの「コンテナのリソース追加」セクションを参照してください。

> **注意:** 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。すべてのテストで同じクラスターを使用しており、Amazon CloudWatch の 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ています。これにより、両方のテストが Amazon CloudWatch の Container Insights に同時にレポートされ、単一のテストで不正確なリソース使用率のメトリクスが発生します。

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。

コンテナのリソース追加

現在サポートしているユーザーの数を増やすには、コンテナのリソースを増やします。これにより、CPU とメモリを増やして、同時ユーザーの増加に対応できます。

新しいタスク定義のリビジョン作成

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: `<stackName>-EcsTaskDefinition-<system-generated-random-Hash>`)
4. [**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** を変更します。
   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**を必要なメモリに更新します。
   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、新しいタスク定義の名前を記録します。この名前にはバージョン番号が含まれます。(例: `<stackName>-EcsTaskDefinition-<system-generated-random-Hash>:<system-generated-versionNumber>`)

Amazon DynamoDB テーブルのアップデート

1. Amazon DynamoDB の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 Amazon DynamoDB の `scenarios-table` テーブルを選択します。(例: *<stackName>*-DLTTestRunnerStorageDLTScenariosTable-*<system-generated-random-Hash>*)
4. タスク定義を変更したリージョンに対応する項目を選択します。(例: region-*<region-name>*)
5. **taskDefinition** 属性を新しいタスク定義で更新します。

ライブデータ

オプションで、テストの実行時にライブデータを含めて、何が起こっているかをリアルタイムで把握できます。AWS Fargate タスクの Amazon CloudWatch ロググループには、ライブデータオプションを含むテスト結果用のサブスクリプションフィルタが含まれています。このソリューションはパターンを見つけると、データを構造化する AWS Lambda 関数が起動し、AWS IoT Core トピックに発行します。ウェブコンソールはそのトピックをサブスクライブし、受信データを受け取り、1 秒間隔で集計されたデータをグラフ化します。ウェブコンソールには、平均応答時間、仮想ユーザー、成功、失敗の 4 つのグラフが含まれています。

> **注意:** このデータは一時的なもので、テストの実行中に何が起きているかを確認するためだけに使用されます。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 Amazon DynamoDB と Amazon S3 に保存します。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。

キャンセルワークフローのテスト

ウェブコンソールから負荷テストをキャンセルすると、このソリューションでは次のキャンセルワークフローのテストが実行されます。

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が停止するまでタスクをキャンセルする `task-canceller` AWS Lambda 関数を呼び出します。

3. `task-runner` AWS Lambda 関数が `task-canceler` AWS Lambda 関数の最初に呼び出され、その後も引き続き実行されている場合は、タスクは引き続き起動されます。`task-runner` AWS Lambda 関数の実行が終了すると、AWS Step Functions はテストを Cancel Test ステップに進み、`task-canceler` AWS Lambda 関数を再度実行して、残りのタスクを停止します。

トラブルシューティング

問題: 既存の Amazon 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.
```

解決方法: 使用しているサブネットに、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。

解決方法: 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 AwS 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 AWS 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、AWS Lambda の `task-runner` 関数用の Amazon CloudWatch Logs を確認して、AWS Fargate タスクの起動時の障害詳細を確認することもできます。実行中の AWS Fargate のコンテナで何が起こっているかの詳細については、Amazon CloudWatch の Amazon ECS のログを確認してください。

ソリューションのアンインストール

「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から、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。保持したいデータが格納されている場合を考慮し、AWS ソリューションではそれらを自動的に削除しません。

> **注意:** 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

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をインストールしたスタックを選択します。
3. [**削除**] を選択します。

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) がお客様の環境で使用できるかどうかを確認します。インストール手順については、*AWS CLI* ユーザーガイド の「AWS Command Line Interface とは」を参照してください。AWS CLI が使用可能になったことを確認したら、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では、AWS CloudFormation スタックを削除して、誤ってデータを損失しないようにするために、このソリューションで作成された Amazon S3 バケット (オプトインリージョンにデプロイするため) を保持するように設定しています。この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、Amazon S3 バケットを手動で削除できます。次の手順に従って、Amazon S3 バケットを削除してください。

1. Amazon S3 コンソールにサインインします。
2. 左のナビゲーションペインから [**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. このソリューションの Amazon S3 バケットの 1 つを選択し、[**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. 空にしたばかりの Amazon 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 Amazon 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 Amazon S3 バケットを削除するまで、ステップ 3 ～ 7 を繰り返してください。

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

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

運用メトリクスの収集

このソリューションには、匿名の運用メトリクスを AWS に送信するオプションが含まれています。当社はこのデータを使用して、お客様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。有効にすると、次の情報が収集され、AWS に送信されます。

- **Solution ID -** AWS ソリューション識別子
- **Unique ID (UUID) -** ソリューションのデプロイごとにランダムに生成された一意の識別子
- **Timestamp -** データ収集タイムスタンプ
- **Test Type -** 実行されるテストのタイプ
- **File Type -** アップロードされるファイルのタイプ
- **Task Count -** このソリューションの API を通じて送信された各テストのタスク数

- **Task Duration -** テストの実行に必要なすべてのタスクの合計実行時間
- **Test Result -** 実行されたテストの結果

AWS は、この調査で収集されたデータを所有します。データ収集には、AWS プライバシーポリシーが適用されます。この機能を無効にするには、AWS CloudFormation テンプレートを起動する前に、次の手順を実施してください。

1. AWS CloudFormation テンプレートをローカルのハードドライブにダウンロードします。
2. テキストエディタで AWS CloudFormation テンプレートを開きます。
3. AWS CloudFormation テンプレートのマッピングセクションを次のように変更します。

```
Solution:
    Config:
      SendAnonymousData: “Yes”
```

次のように変更します。

```
Solution:
    Config:
      SendAnonymousData: “No”
```

4. AWS CloudFormation コンソールにサインインします。
5. [**スタックの作成**] を選択します。
6. **スタックの作成**ページの**テンプレートの指定**セクションで、[**テンプレートファイルのアップロード**] を選択します。
7. **テンプレートファイルのアップロード**で、[**ファイルの選択**] を選択し、ローカルドライブから編集したテンプレートを選択します。
8. [**次へ**] を選択し、このガイドの「自動デプロイメント」セクションの「スタックの起動」の手順に従います。

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。

改訂

日付	変更
2019 年 11 月	初回リリース
2020 年 9 月	リリースバージョン 1.1.0: Amazon SQS を AWS Step Functions に置き換え、アーキテクチャ図とコンポーネント情報を更新して、変更された AWS サービスの詳細を記載し、JMeter スクリプトのサポートを追加。(詳細については、GitHub リポジトリの CHANGELOG.md ファイルを参照)
2020 年 12 月	リリースバージョン 1.2.0: AWS Step Functions に Amazon ECR チェッカーを追加。JMeter の zip ファイルアップロードのサポートが追加され、JMeter プラグインが使用可能になりました。(詳細については、GitHub リポジトリの CHANGELOG.md ファイルを参照)
2021 年 4 月	リリースバージョン 1.3.0: 同時テストの実行するためのサポート、同じテストに属するタスク間でテストを同時に開始するサポート、スケジューリングテストのためのサポート、を追加し、タスクの上限を 1,000 タスクに増加し、同時ユーザー制限を削除しました。(詳細については、GitHub リポジトリの CHANGELOG.md ファイルを参照)
2021 年 9 月	リリースバージョン 2.0.0: このソリューションのコンテナイメージは AWS によって管理されるようになり、ユーザーの AWS アカウントでビルドパイプラインと Amazon ECR イメージリポジトリを作成する必要がなくなりました。以前のテスト実行からのすべてのテスト設定、テストデータ、Amazon CloudWatch ダッシュボードを表示するためのサポートを追加しました。Amazon CloudWatch ダッシュボードを更新して最大データポイントを表示しました。既存の Amazon VPC のサポートを追加しました。AWS Fargate タスクに AWS CloudFormation タグを追加しました。テスト用の AWS Fargate タスクが複数のアベイラビリティーゾーンで起動するようになりました。詳細については、GitHub リポジトリの CHANGELOG.md ファイルを参照してください。

日付	変更
2021 年 12 月	リリースバージョン 2.0.1: AWS Lambda 関数の開発依存関係で AWS SDK バージョンを更新。多数のテストを表示する際の問題を解決し、Amazon DynamoDB での ValidationException エラーを解決しました。詳細については、GitHub リポジトリの CHANGELOG.md ファイルを参照してください。
2022 年 8 月	リリースバージョン 3.0.0: AWS CDK V2 に更新し、マルチリージョン機能の追加、ライブデータ機能の追加、JMeter 拡張の互換性の追加し、その他の機能強化とバグ修正を行いました。詳細については、GitHub リポジトリの CHANGELOG.md ファイルを参照してください。

寄稿者

- Tom Nightingale
- Fernando Dingler
- Beomseok Lee
- George Lenz
- Erin McGill

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。このドキュメントは、(a) 情報提供のみを目的としており、(b) AWS の現行製品とプラクティスを表したものであり、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義務や確約を意味するものではありません。AWS の製品やサービスは、明示または暗示を問わず、いかなる保証、表明、条件を伴うことなく「現状のまま」提供されます。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で行われるいかなる契約の一部でもなく、そのような契約の内容を変更するものでもありません。

「AWS での分散負荷テスト」ソリューションは、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。