ユーザーガイド

AWS Mainframe Modernization

AWS Mainframe Modernization: ユーザーガイド

Table of Contents

AWS Mainframe Modernization とは

AWS Mainframe Modernization は、メインフレームアプリケーションを AWS マネージドランタイム環境にモダナイズするのに役立ちます。移行とモダナイズの計画および実装に役立つツールとリソースを提供します。既存のメインフレームアプリケーションの分析、また COBOL や PL/I を使用してメインフレームアプリケーションの開発または更新が可能です。自動パイプラインを実装することもでき、アプリケーションの継続的インテグレーションと継続的デリバリー (CI/CD) を実現できます。クライアントのニーズに応じて、自動リファクタリングパターンとリプラットフォーミングパターンのどちらかを選択することができます。クライアントがメインフレームのワークロードを移行するのを支援しているのであれば、初期計画から移行後のクラウド運用まで、移行とモダナイゼーションの過程のあらゆる段階で AWS Mainframe Modernization ツールを使用できます。

AWS Mainframe Modernization を使用すると、メインフレームアプリケーション用のランタイム環境を で AWS 効率的に作成および管理し、モダナイズされたアプリケーションを管理およびモニタリングできます。

トピック

> **Note**
>
> メインフレームのモダナイゼーションプロジェクトのために AWS メインフレーム移行コンピテンシーパートナーまたは AWS プロフェッショナルサービスとやり取りしたことはありますか？ 利用したことがない場合は、専門家と連携してプロジェクトを実行することを強くお勧めします。
>
> - AWS Mainframe Modernization コンピテンシーパートナー
> - AWS プロフェッショナルサービス

AWS Mainframe Modernization の特徴とユースケースは、進化的なモダナイゼーションアプローチをサポートしています。これは、俊敏性と、後で最適化してイノベーションを起こすさまざまな機会を改善することで、短期的な成功を提供します。詳細については、「モダナイゼーションアプローチ」を参照してください。

AWS Mainframe Modernization の機能

AWS Mainframe Modernization の機能は、次のユースケースをサポートしています。

- 評価: AWS Mainframe Modernization の評価機能は、移行とモダナイゼーションプロジェクトの評価、範囲設定、計画に役立ちます。
- リファクタリング: AWS Blu Age を搭載したリファクタリングを使用して、レガシーアプリケーションプログラミング言語の変換、マクロサービスまたはマイクロサービスの作成、ユーザーインターフェイス (UIsとアプリケーションソフトウェアスタックのモダナイズを行うことができます。

  AWS Blu Insights がシングルサインオン AWS Management Console を通じて から利用可能になりました。別の AWS Blu Insights 認証情報を管理する必要はありません。 AWS AWS Blu Age Codebase と変換センターの機能には、 から直接アクセスできます AWS Management Console。
- リプラットフォーム: Micro Focus Enterprise のソリューションを使用しているため、アプリケーションのソースコードの大部分を変更せずに再コンパイルしたアプリケーションを移植することができます。
- デベロッパー IDE: AWS Mainframe Modernization はオンデマンドの統合開発環境 (IDE) を提供するため、デベロッパーはスマート編集とデバッグ、コードの即時コンパイル、ユニットテストにより、コードをより迅速に記述できます。
- マネージドランタイム: AWS Mainframe Modernization マネージド実行環境は、クラスターを継続的にモニタリングして、自己修復コンピューティングと自動スケーリングでエンタープライズワークロードを実行し続けることができます。
- 継続的インテグレーションとデリバリー (CI/CD): AWS メインフレーム・モダナイゼーションの CI/CD 機能は、アプリケーション開発チームがコード変更をより頻繁に確実に提供できるよう支援します。これにより、移行速度が速くなり、品質が向上し、新しいビジネス機能のリリース time-to-market を減らすことができます。
- 他の AWS サービスとの統合: AWS Mainframe Modernization は AWS CloudFormation、、、および をサポートし AWS PrivateLink、 AWS Key Management Service 繰り返しデプロイとセキュリティとコンプライアンスを強化します。

- 可用性の拡張: AWS Mainframe Modernization が米国東部 (オハイオ) 、米国西部 (北カリフォルニア) 、アジアパシフィック (ムンバイ) 、アジアパシフィック (ソウル) 、アジアパシフィック (シンガポール) 、アジアパシフィック (東京) 、欧州 (ロンドン) 、欧州 (パリ) で利用可能になりました。

AWS Mainframe Modernization 機能の詳細については、「」を参照してくださいhttps://aws.amazon.com/mainframe-modernization/features/。

パターン

AWS Blu Age による自動リファクタリングパターンは、機能的な等価性を維持しながら、レガシーアプリケーションスタックとそのデータレイヤー全体を最新の Java ベースのアプリケーションに変換することで、モダナイゼーションを加速することに重点を置いています。この自動変換中に、Angular ベースのフロントエンド、API 対応の Java バックエンド、最新のデータストアにアクセスするデータレイヤーを持つ多層アプリケーションが作成されます。リファクタリングプロセスによって従来のスタックと同等の機能が発揮されて、プロジェクトの自動化が強化されます。これによりスピード、品質、コスト削減が実現し、ビジネスでの利益をより早く得ることができます。詳細については、「AWS Mainframe Modernization Automated Refactor」を参照してください。

Micro Focus Enterprise スイートを使用したリプラットフォームパターンは、アプリケーションアセットやチームへの影響を最小限に抑えるために、アプリケーション言語、コード、アーティファクトを保存することに重点を置いています。これにより、お客様はアプリケーションの知識とスキルを維持することができます。アプリケーションの変更は限定的ですが、このパターンを使用すればインフラストラクチャとプロセスの最新化も容易になります。インフラストラクチャは最新のクラウドベースのマネージドサービスに変更され、プロセスもアプリケーション開発と IT 運用のベストプラクティスに従うように変更されます。詳細については、「AWS Mainframe Modernization Replatform」を参照してください。

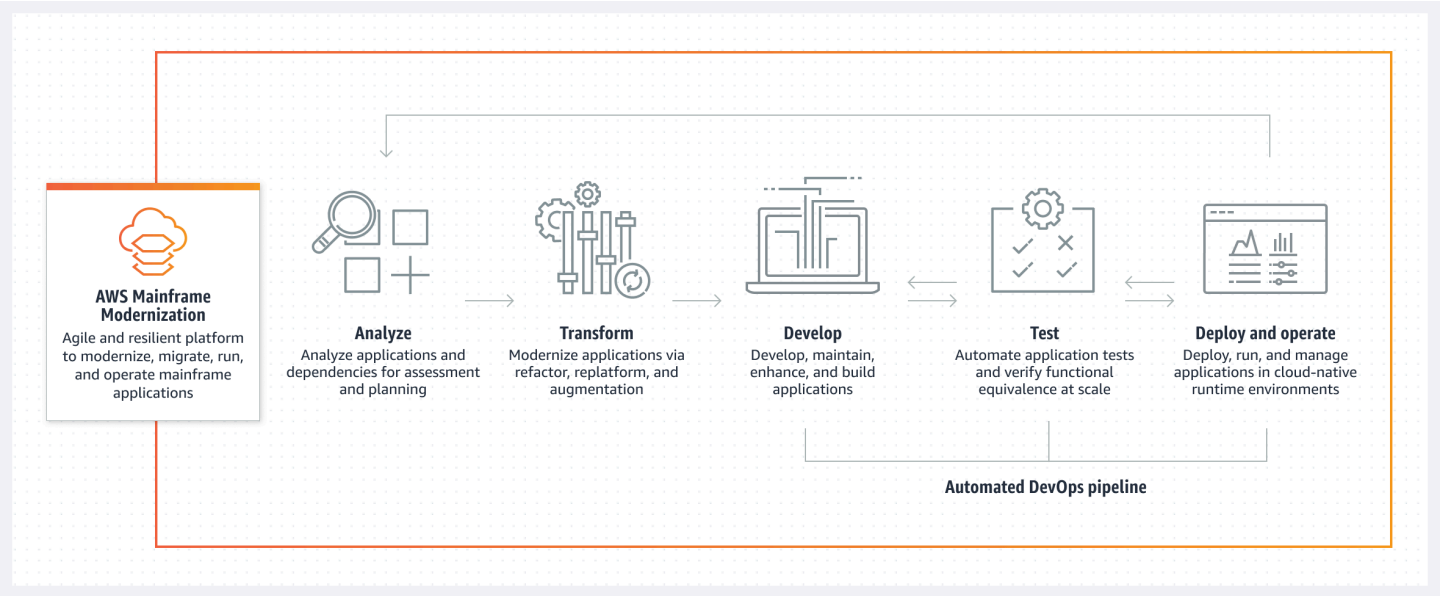
AWS Mainframe Modernization の使用を開始する方法

試してみましょう！AWS Mainframe Modernization が提供する内容を理解するのに役立つチュートリアルとサンプルアプリケーションを提供しています。完全な step-by-step チュートリアルチュートリアル: Micro Focus のマネージドランタイムについては、 チュートリアル: AWS Blu Age のマネージドランタイムまたは を選択します。

自動リファクタリングに関心がある場合は、「」で AWS Blu Age ツールを確認してくださいBluInsights。 AWS Blu Age デベロッパー IDE、または Micro Focus Enterprise Analyzer と Micro

Focus Enterprise Developer ツールにアクセスするように AppStream 2.0 を設定することもできます。

チュートリアルとサンプルアプリケーションは、 AWS Mainframe Modernization が提供する内容を理解するだけです。モダナイゼーションプロジェクトを開始する準備ができたら、「モダナイゼーションアプローチ」を参照してモダナイゼーションプロジェクトの段階とタスクの詳細を確認してください。



関連サービス

自動リファクタリングのための Blu Insights に加えて、 AWS Mainframe Modernization で次の AWS サービスを使用できます。

- 移行したデータベースをホストできる Amazon RDS。
- アプリケーションバイナリと定義ファイルを保存できる Amazon S3。
- アプリケーションデータを保存できる Amazon FSx または Amazon EFS。
- Micro Focus Enterprise Analyzer および Micro Focus Enterprise Developer ツールにアクセス AppStream するための Amazon。
- AWS CloudFormation 移行したアプリケーションの CI/CD をセットアップするために使用できる自動 DevOps パイプライン用の 。
- AWS Migration Hub
- AWS DMS データベースを移行するための 。

AWS Mainframe Modernization へのアクセス

現在、https://console.aws.amazon.com/m2/ のコンソールから AWS Mainframe Modernization にアクセスできます。 AWS Mainframe Modernization が利用可能なリージョンのリストについては、「」のAWS 「 Mainframe Modernization エンドポイントとクォータ」を参照してくださいAmazon Web Services 全般のリファレンス。

AWS Mainframe Modernization を初めてお使いになる方向けの情報

AWS Mainframe Modernization を初めて使用する場合は、まず以下のセクションを読むことをお勧めします。

- ご利用開始にあたって
- セットアップ

料金

AWS Mainframe Modernization では、マネージドランタイム環境をサポートするインスタンスの使用に対して料金が発生します。さらに、 AWS Mainframe Modernization は追加料金なしで一部のツールを提供しています。 AWS Mainframe Modernization に関連して使用する他の AWS サービスに対して発生する料金はお客様の負担となります。 Mainframe Modernization の使用に対して料金の変更が反映されるまで 30 AWS 日前に AWS から通知されます。詳細については、「 Mainframe Modernization with AWS」を参照してください。

AWS Blu Insights では、変換センターの使用に対して料金が発生します。詳細については、「AWS Mainframe Modernization pricing」を参照してください。

AWS Mainframe Modernization のセットアップ

AWS Mainframe Modernization の使用を開始する前に、ユーザーまたは管理者がいくつかのステップを完了する必要があります。

トピック

にサインアップする AWS アカウント

がない場合は AWS アカウント、次のステップを実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手続きの一環として、通話呼び出しを受け取り、電話のキーパッドを用いて検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウント内のすべての AWS のサービス とリソースにアクセスできます。セキュリティのベストプラクティスとして、管理ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行します。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理ユーザーの作成

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、「AWS サインイン ユーザーガイド」の「Signing in as the root user」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理ユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。
2. IAM アイデンティティセンターで、管理ユーザーに管理アクセス権を付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセスを設定する IAM アイデンティティセンターディレクトリAWS IAM Identity Center 」を参照してください。

管理ユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインイン ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

AWS Mainframe Modernization の開始方法

AWS Mainframe Modernization の使用を開始するには、サービスと各ランタイムエンジンを紹介するチュートリアルに従います。

トピック

学習を続けるには、以下のチュートリアルを参照してください。

チュートリアル: AWS Blu Age のマネージドランタイム

このチュートリアルでは、AWS Blu Age のモダナイズされたアプリケーションを AWS メインフレームモダナイゼーションのランタイム環境にデプロイする方法を示します。

トピック

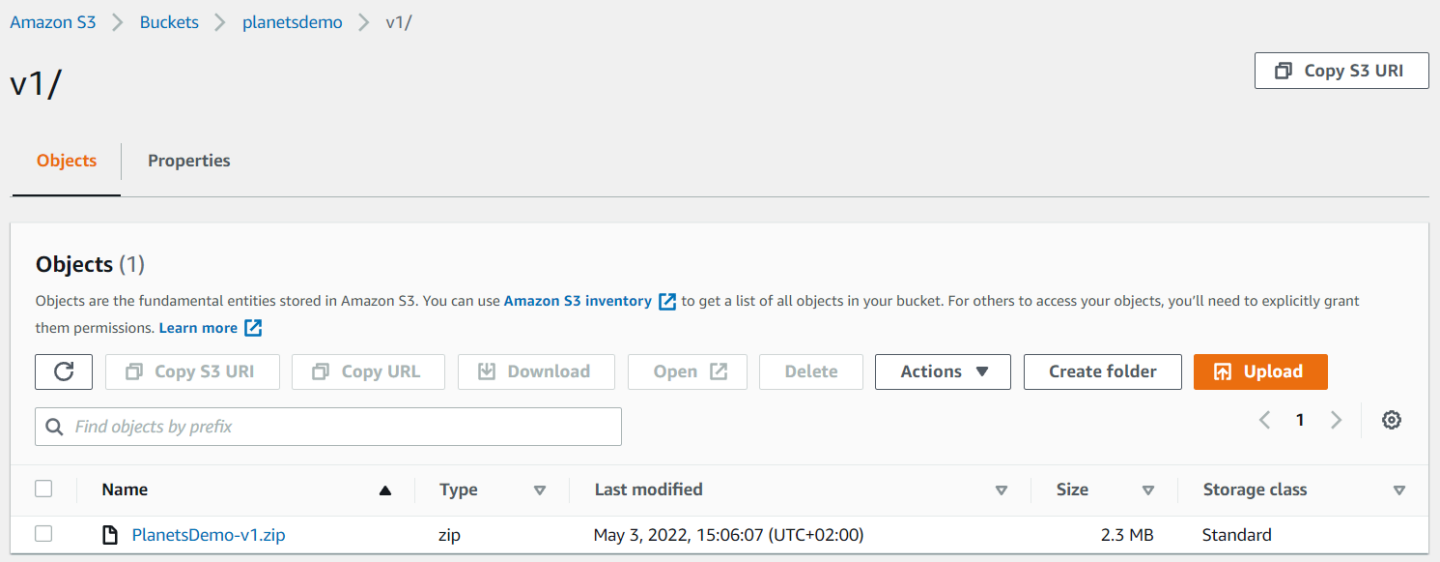
前提条件

このチュートリアルを完了するには、デモアプリケーションアーカイブ PlanetsDemo-v1.zip をダウンロードします。

デモアプリケーションを実行するには、最新のブラウザが必要です。このブラウザをデスクトップから実行するか、VPC 内などの Amazon Elastic Compute Cloud インスタンスから実行するかによって、セキュリティ設定が決まります。

ステップ 1: デモアプリケーションをアップロードする

Amazon S3 バケットにデモアプリケーションをアップロードします。このバケットがアプリケーションをデプロイする場所と同じ AWS リージョン にあることを確認します。次の例は、v1 という名前のキープレフィックスまたはフォルダを持つ planetsdemo という名前のバケットと、`planetsdemo-v1.zip` という名前のアーカイブを示しています。



> **Note**
>
> バケット内のフォルダは必須です。

ステップ 2: アプリケーション定義を作成する

マネージドランタイムにアプリケーションをデプロイするには、AWS Mainframe Modernization アプリケーション定義が必要です。この定義は、アプリケーションの場所と設定を記述した JSON ファイルです。以下の例は、このようなデモアプリケーションのアプリケーション定義です。

```
{
    "template-version": "2.0",
    "source-locations": [{
        "source-id": "s3-source",
        "source-type": "s3",
        "properties": {
            "s3-bucket": "planetsdemo",
            "s3-key-prefix": "v1"
        }
    }],
    "definition": {
        "listeners": [{
            "port": 8196,
            "type": "http"
        }],
        "ba-application": {
            "app-location": "${s3-source}/PlanetsDemo-v1.zip"
        }
    }
}
```

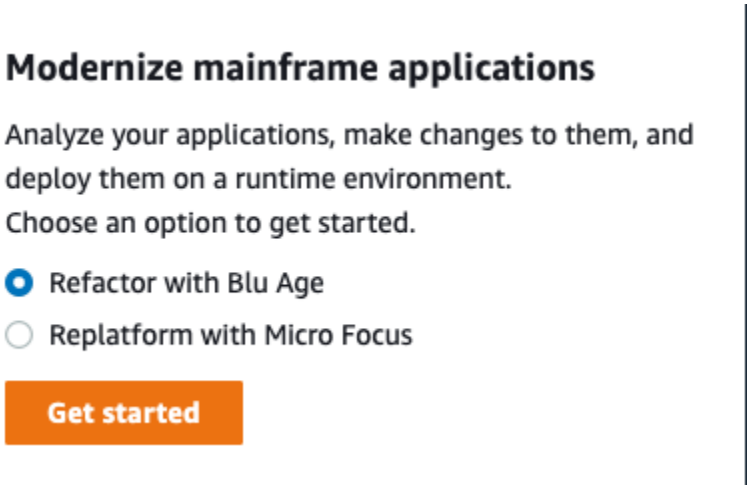
s3-bucket エントリを、サンプルアプリケーションの zip ファイルを保存したバケットの名前に変更します。

アプリケーション定義の詳細については、「AWS Blu Age アプリケーション定義のサンプル」を参照してください。

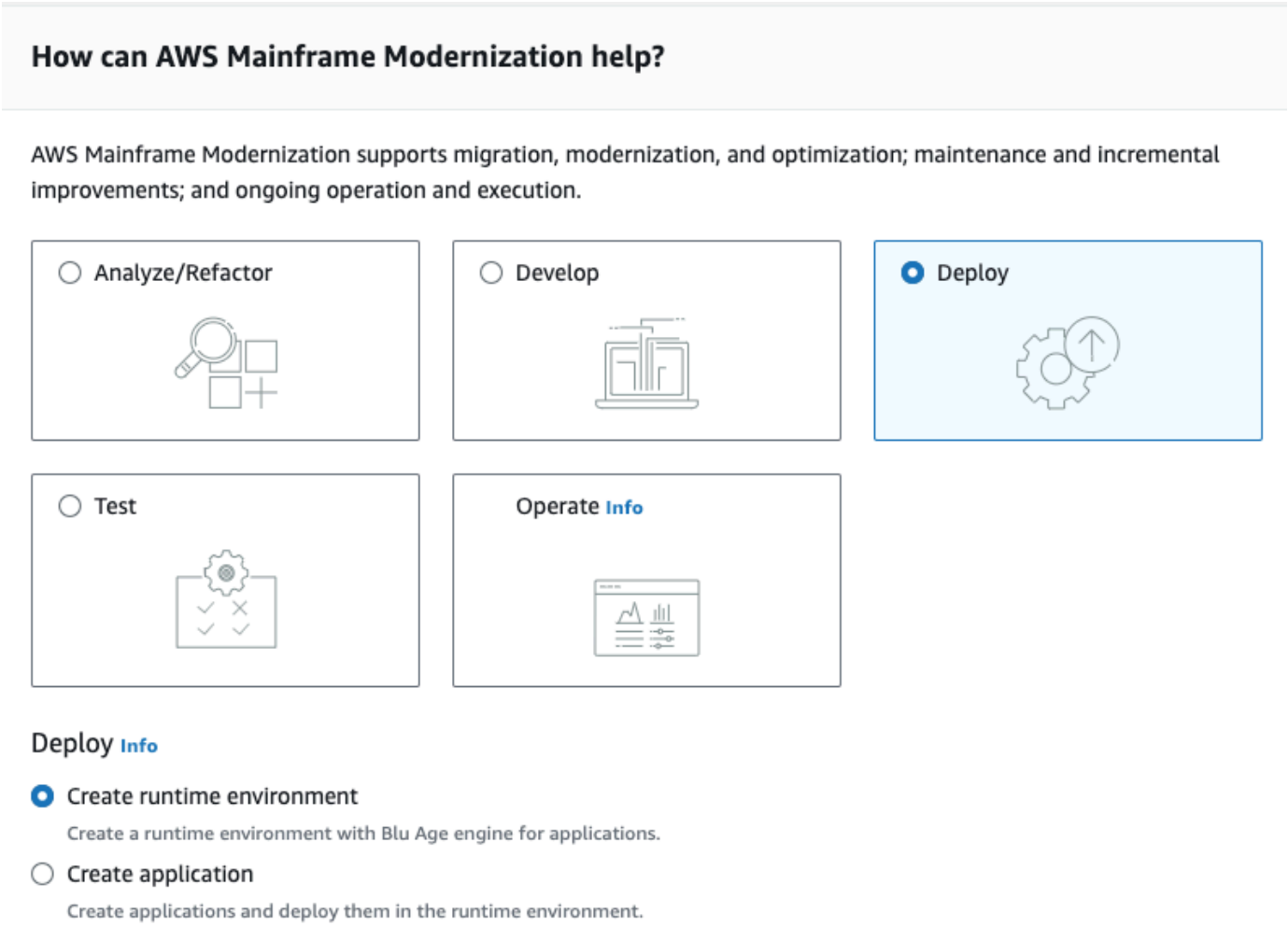
ステップ 3: ランタイム環境を作成する

AWS Mainframe Modernization ランタイム環境を作成するには、次のステップを実行します。

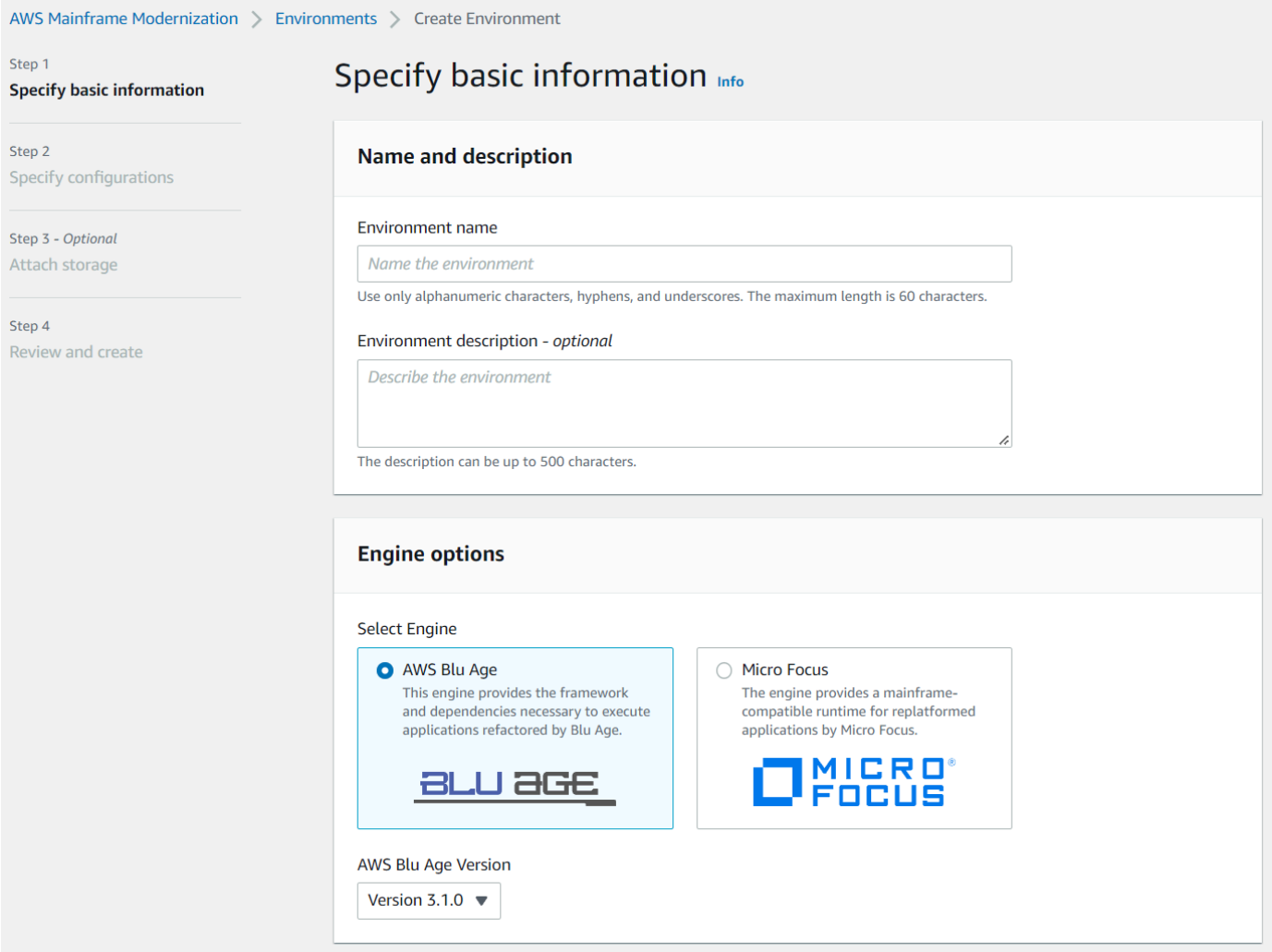
1. AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、環境を作成するリージョンを選択します。この AWS リージョンは ステップ 1: デモアプリケーションをアップロードする で S3 バケットを作成したリージョンと一致する必要があります。
3. [メインフレームアプリケーションのモダナイズ] で [Blu Age でリファクタリングする] を選択し、[今すぐ始める] を選択します。



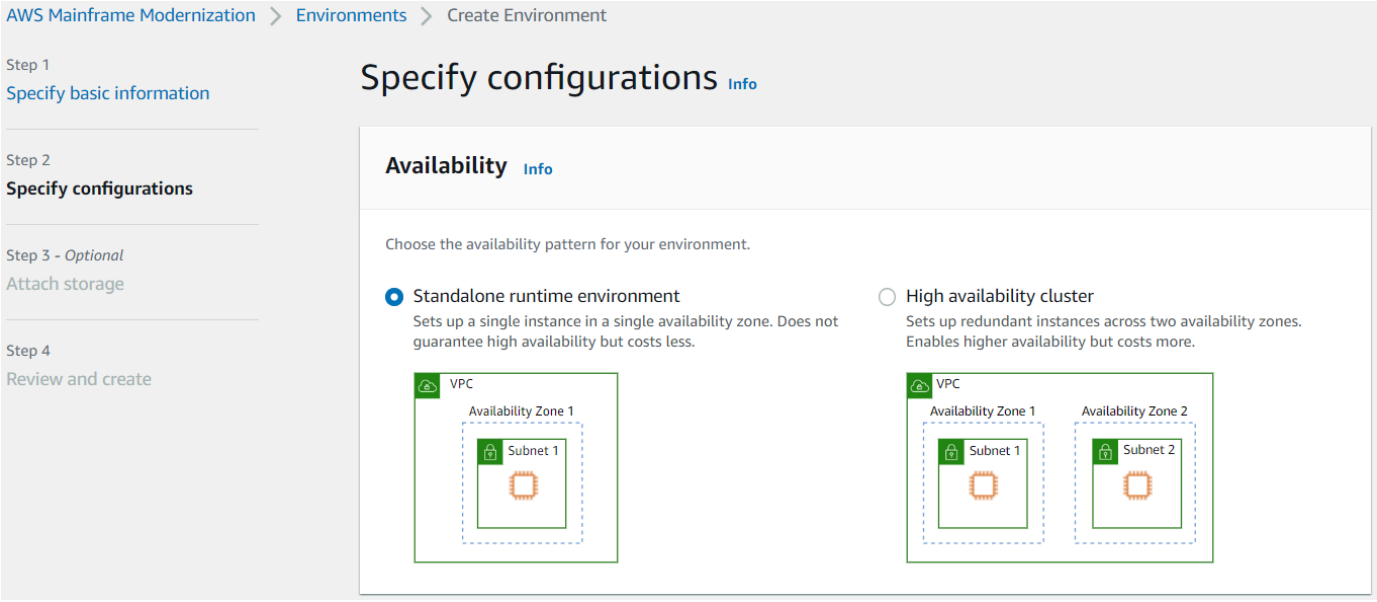
4. [AWS Mainframe Modernization はどのように役立つのか] で、[デプロイ] と [ランタイム環境を作成] を選択します。



5. ナビゲーションペインで、[環境] を選択し、[環境を作成] を選択します。[基本情報を指定] ページで、環境の名前と説明を入力し、AWS Blu Age エンジンが選択されていることを確認します。オプションで、作成したリソースにタグを追加できます。次いで、[次へ] を選択します。

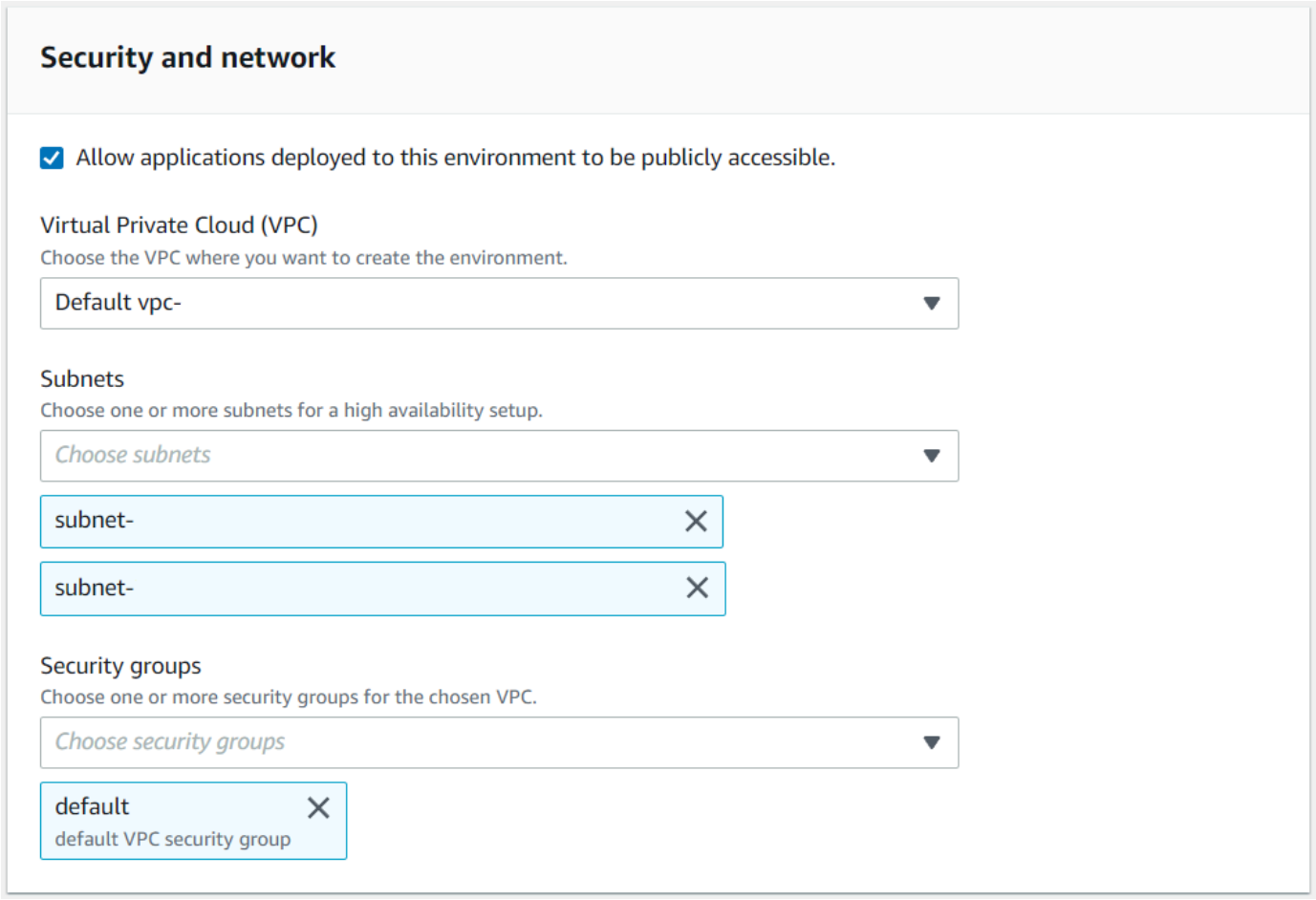


6. [設定を指定] ページで、[スタンドアロンランタイム環境] を選択します。



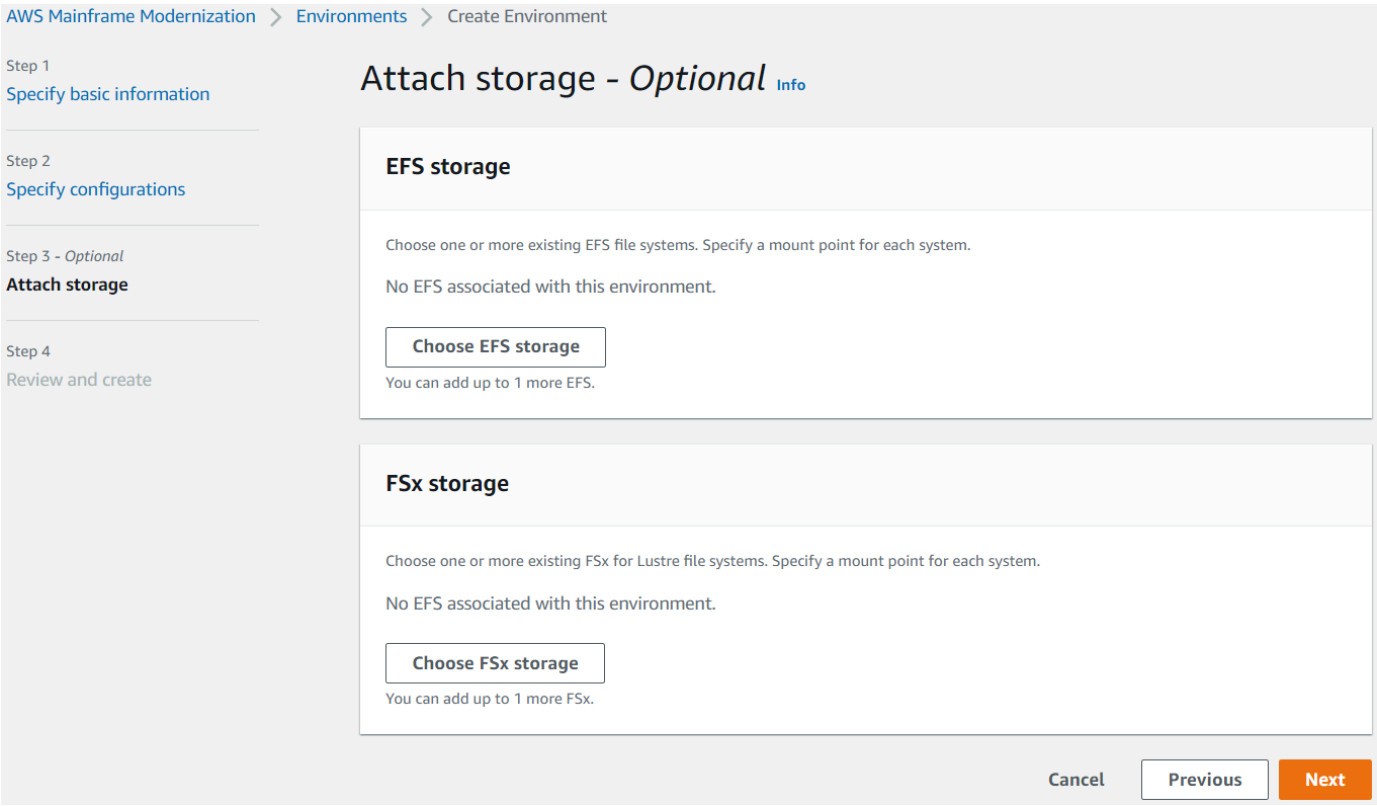
7. [セキュリティとネットワーク] で、以下の変更を行います。

   - [この環境にデプロイされたアプリケーションをパブリックにアクセスできるようにする] を選択します。このオプションでは、アプリケーションにパブリック IP アドレスを割り当てて、デスクトップからアクセスできるようにします。
   - VPC を選択します。[デフォルト] を使用できます。
   - 2 つのサブネットを選択します。サブネットがパブリック IP アドレスの割り当てを許可していることを確認してください。
   - [セキュリティグループ] をクリックします。[デフォルト] を使用できます。選択したセキュリティグループが、ブラウザの IP アドレスからアプリケーション定義の `listener` プロパティで指定したポートへのアクセスを許可していることを確認してください。詳細については、「ステップ 2: アプリケーション定義を作成する」を参照してください。



選択した VPC の外部からアプリケーションにアクセスする場合は、その VPC のインバウンドルールが適切に設定されていることを確認してください。詳細については、「アプリケーションの URL にはアクセスできません。」を参照してください。

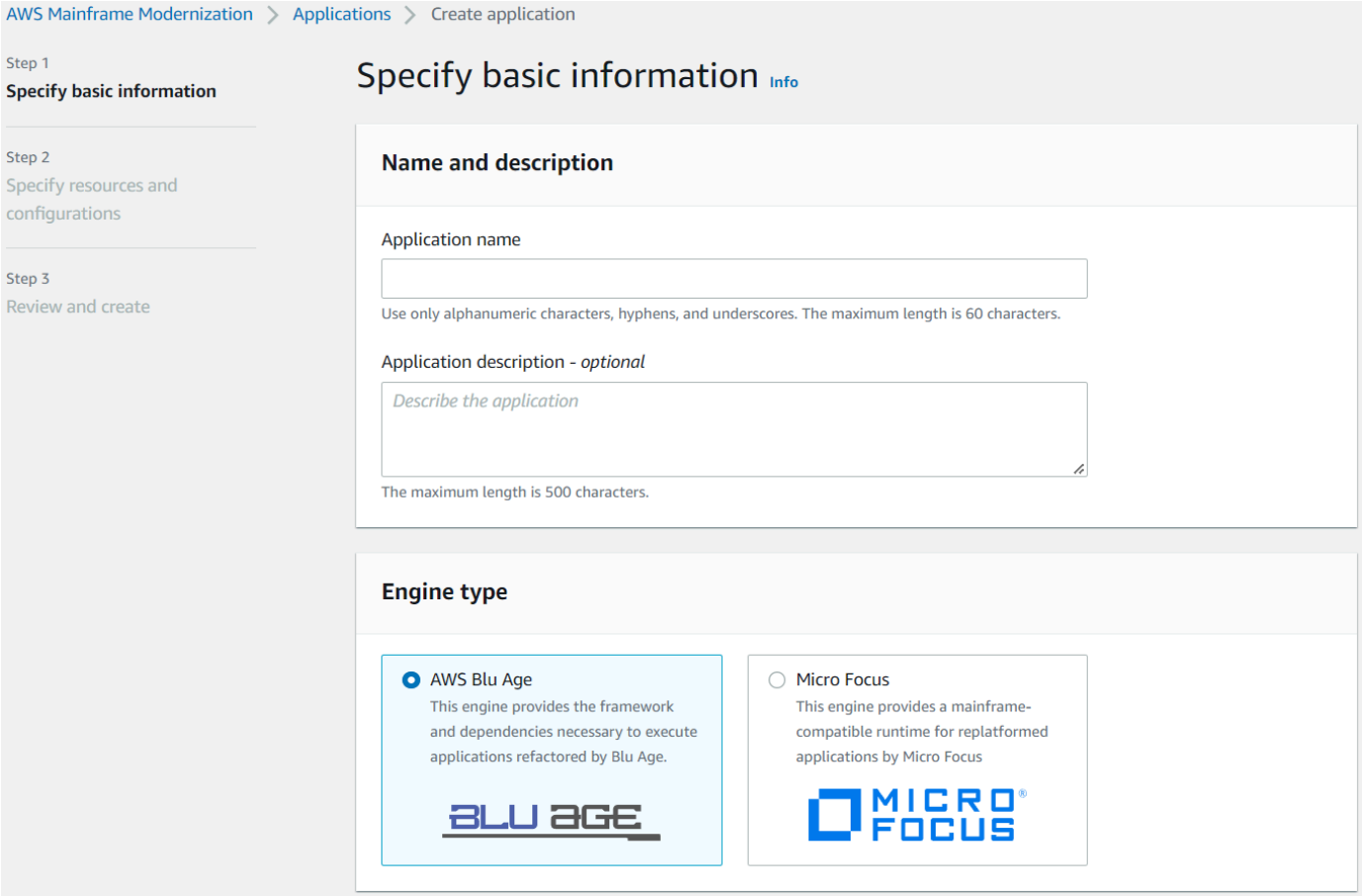
8. [次へ] を選択します。
9. [ストレージをアタッチ - オプション] では、デフォルトの選択をそのままにして [次へ] を選択します。



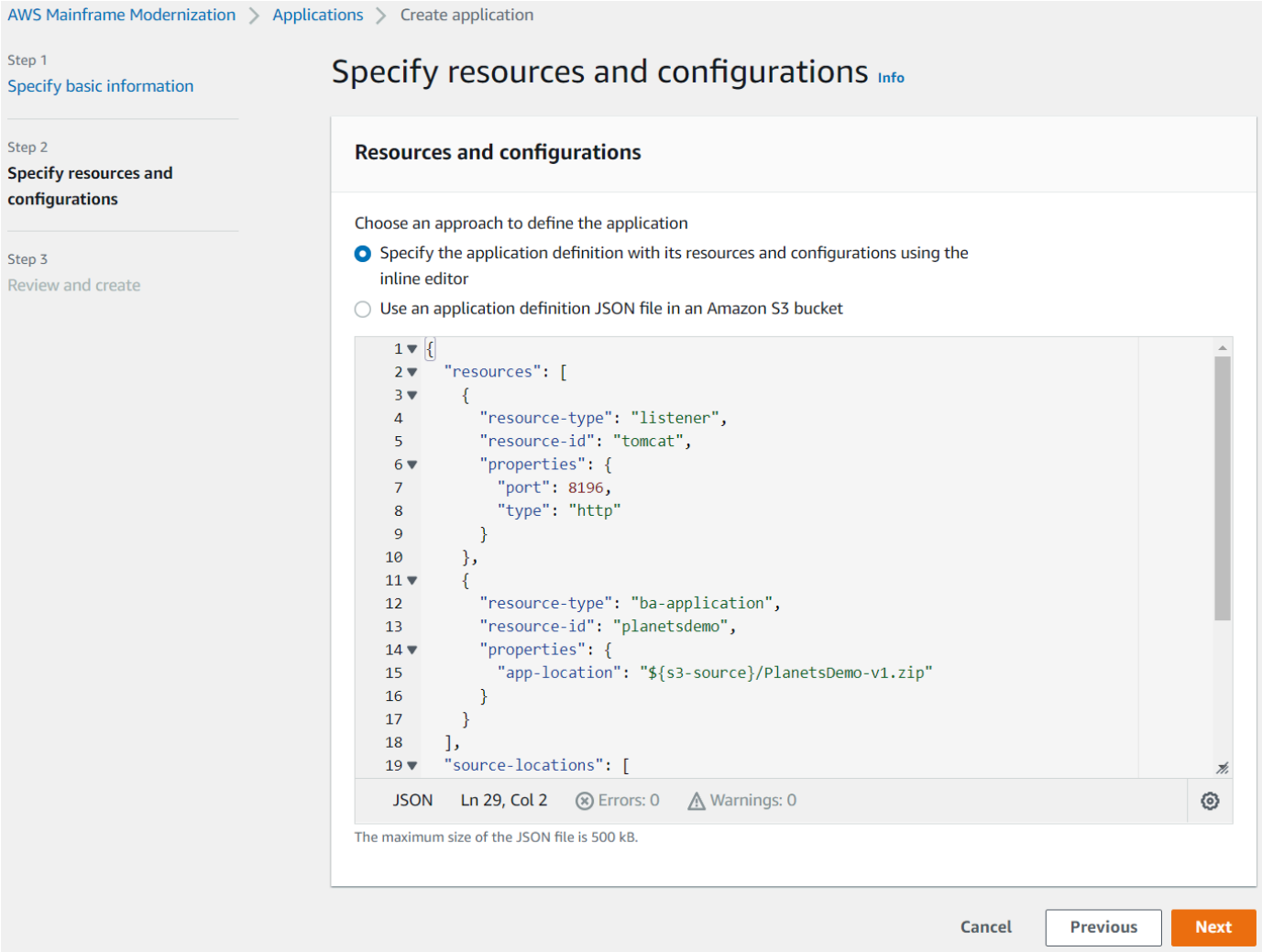
10. [メンテナンスをスケジュール] で、[指定なし] を選択し、[次へ] を選択します。
11. [確認して作成] で情報を確認し、[環境の作成] を選択します。

ステップ 4: アプリケーションを作成する

1. AWS Management Console で [AWS Mainframe Modernization] に移動します。
2. ナビゲーションペインで、[アプリケーション] を選択し、[アプリケーションの作成] を選択します。[基本情報を指定] ページで、アプリケーションの名前と説明を入力し、AWS Blu Age エンジンが選択されていることを確認します。次いで、[次へ] を選択します。



3. [リソースと設定を指定] ページで、the section called “ステップ 2: アプリケーション定義を作成する” で作成し手更新したアプリケーション定義 JSON をコピーして貼り付けます。

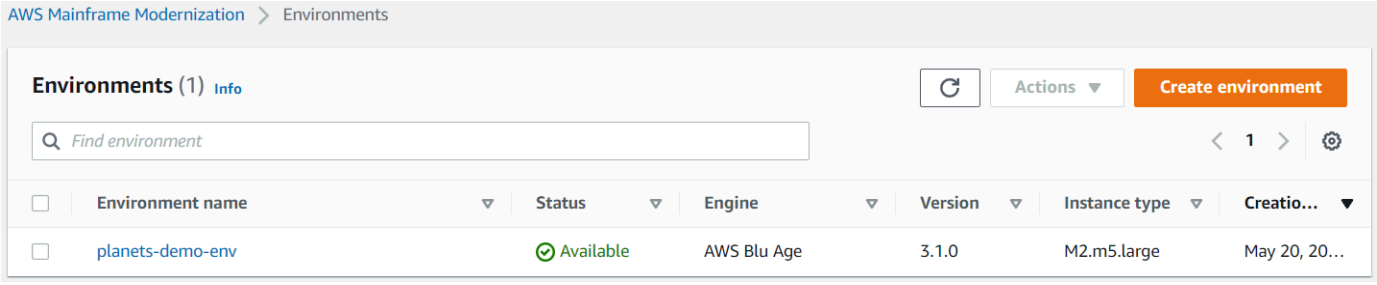


4. [確認と作成] で選択内容を見直してから [アプリケーションの作成] を選択します。

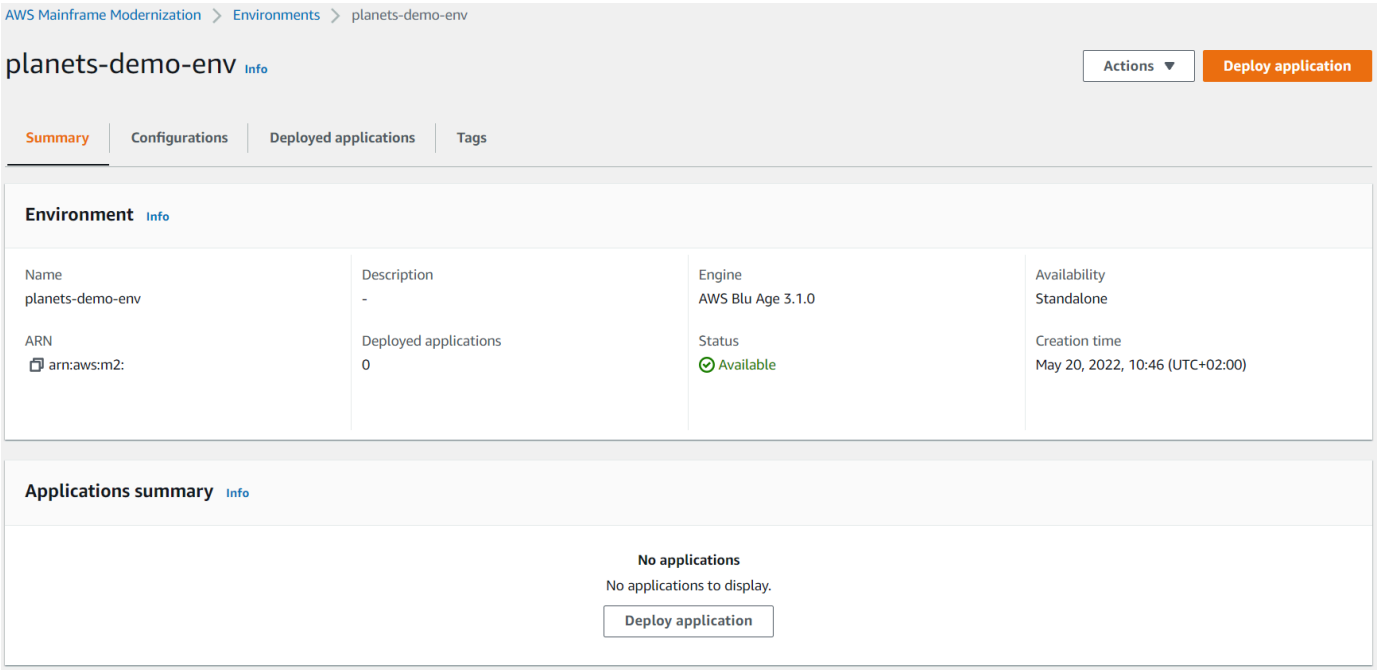
ステップ 5: アプリケーションをデプロイする

AWS Mainframe Modernization ランタイム環境とアプリケーションの両方が正常に作成され、両方が [利用可能] 状態になったら、アプリケーションをランタイム環境にデプロイできます。そのためには、以下のステップを完了します。

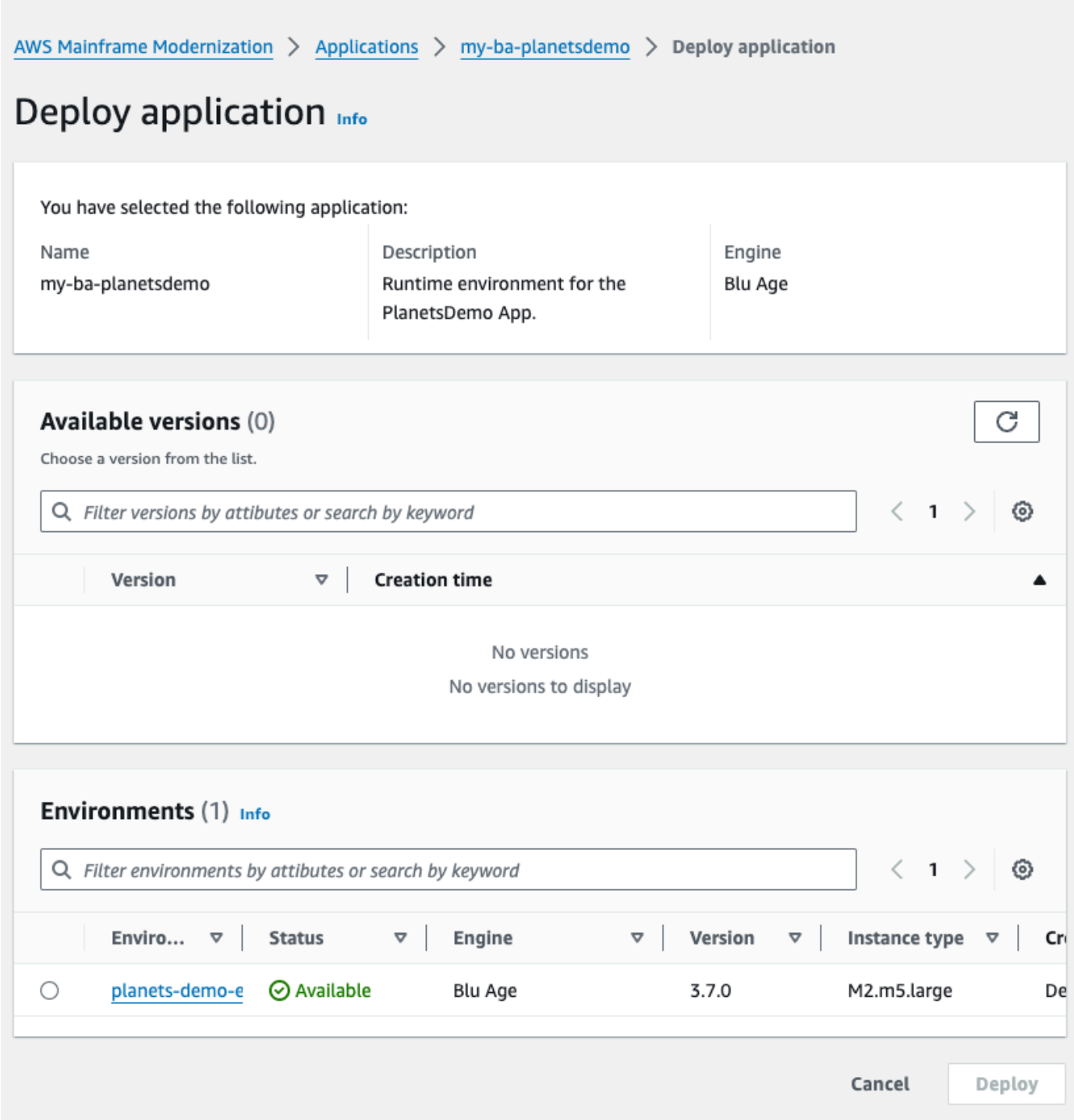
1. AWSマネジメントコンソールの [AWS Mainframe Modernization] に移動します。ナビゲーションペインで [環境] を選択します。[環境] リストページが表示されます。



2. 以前に作成したランタイム環境を選択します。[環境の詳細] ページが表示されます。
3. [アプリケーションのデプロイ] を選択します。



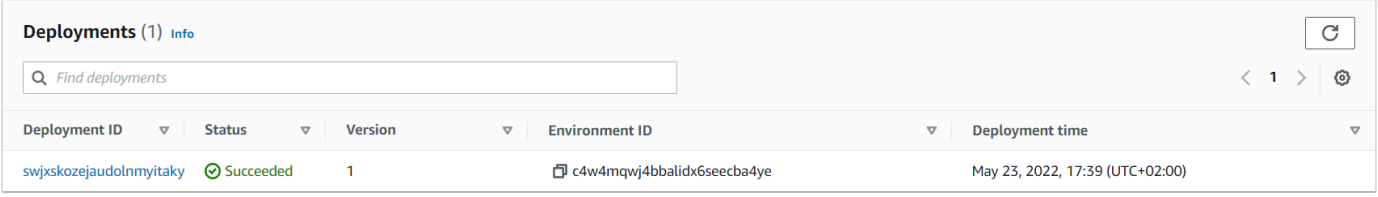
4. 前に作成したアプリケーションを選択し、アプリケーションをデプロイするバージョンを選択します。その後、[デプロイ] を選択します。



5. アプリケーションがデプロイを完了するまでお待ちください。「アプリケーションが正常にデプロイされました」というメッセージのバナーが表示されます。

ステップ 6: アプリケーションを開始する

1. AWS Management Consoleの [AWS Mainframe Modernization] に移動し、[アプリケーション] を選択します。
2. アプリケーションを選択してから、[デプロイ] に移動します。アプリケーションのステータスは [成功] になっているはずです。



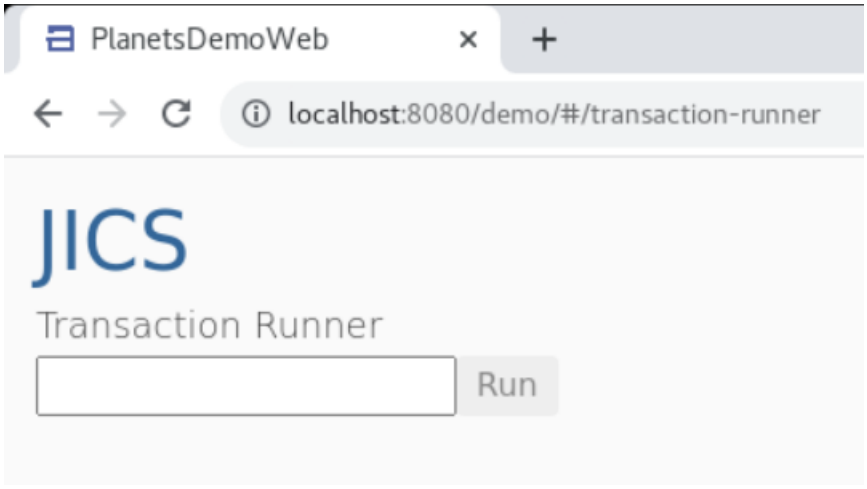
3. [アクション] を選択してから、[アプリケーションを開始] を選択します。

ステップ 7: アプリケーションにアクセスする

1. アプリケーションが [実行中] 状態になるまで待ちます。「アプリケーションが正常に開始されました」というメッセージのバナーが表示されます。
2. アプリケーションの DNS ホスト名をコピーします。このホスト名はアプリケーションの [アプリケーション情報] セクションにあります。
3. ブラウザで、`http://{hostname}:{portname}/PlanetsDemo-web-1.0.0/` に移動します。ここで:

   - `hostname` は以前にコピーした DNS ホスト名です。
   - `portname` は ステップ 2: アプリケーション定義を作成する で作成したアプリケーション定義で定義されている Tomcat ポートです。

   JICS 画面が表示されます。



アプリケーションにアクセスできない場合は、「アプリケーションの URL にはアクセスできません。」を参照してください。

> **Note**
>
> アプリケーションにアクセスできず、セキュリティグループのインバウンドルールによりポート 8196 で「My IP」が選択されている場合は、ポート 8196 で LB i/p からのトラフィックを許可するルールを指定します。

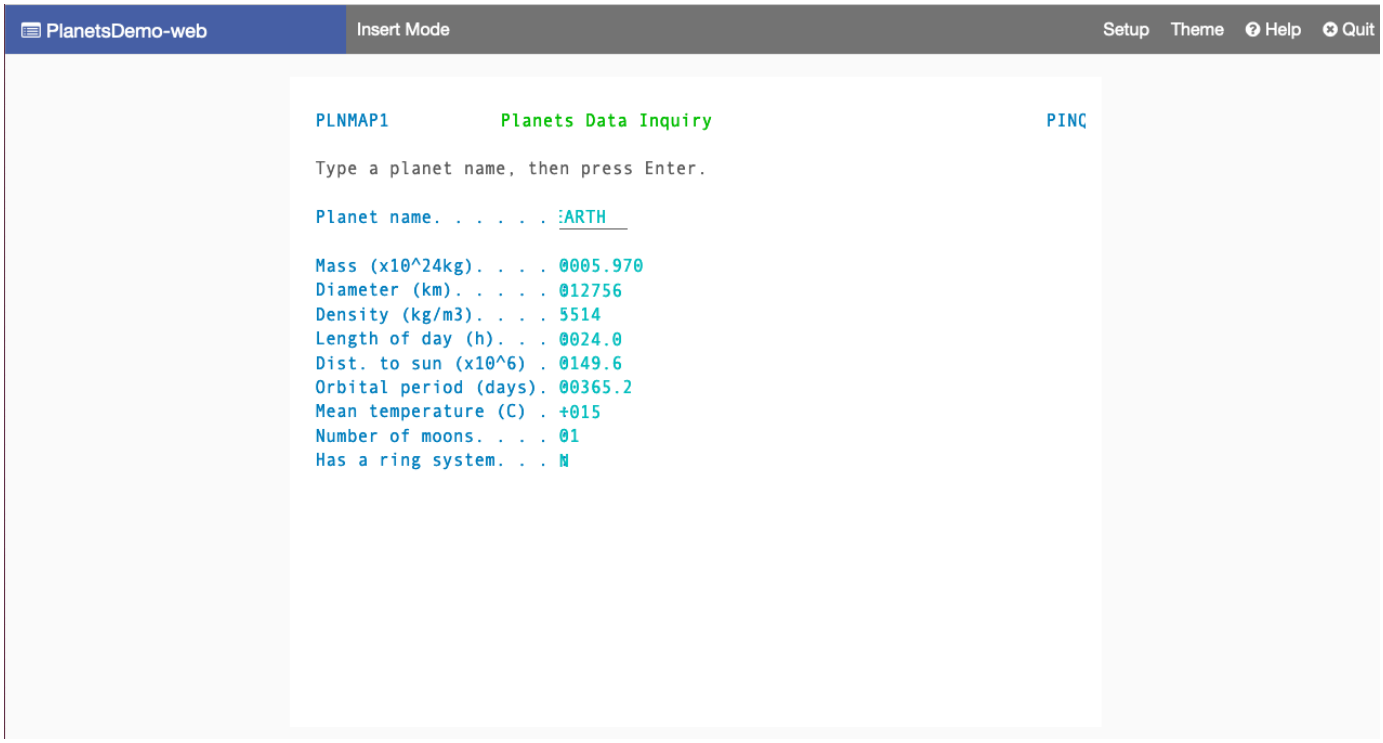
ステップ 8: アプリケーションのテスト

このステップでは、移行したアプリケーションでトランザクションを実行します。

1. JICS 画面で、入力フィールドで PINQ を入力し、[実行] を選択 (または Enter を押す) してアプリケーショントランザクションを開始します。

   デモアプリ画面が表示されます。

2. 対応するフィールドに惑星名を入力し、Enter キーを押します。



惑星の詳細が表示されます。

リソースをクリーンアップする

このチュートリアルのために作成したリソースが不要になった場合は、追加料金が発生しないように削除します。そのためには、以下のステップを実行します。

- AWS Mainframe Modernization アプリケーションがまだ実行中の場合は、終了させます。
- アプリケーションを削除します。詳細については、「AWS Mainframe Modernization アプリケーションの削除」を参照してください。
- ランタイム環境を削除します。詳細については、「AWS Mainframe Modernization ランタイム環境を削除する」を参照してください。

チュートリアル: Micro Focus のマネージドランタイム

このチュートリアルでは、Micro Focus ランタイムエンジンを使用して、 AWS Mainframe Modernization マネージドランタイム環境で CardDemo サンプルアプリケーションをデプロイして実行する方法を示します。 CardDemo サンプルアプリケーションは、メインフレームのモダナイゼーションのユースケースで AWSとパートナーテクノロジーをテストおよび紹介するために開発されたシンプルなクレジットカードアプリケーションです。

このチュートリアルでは、他の AWS のサービス でリソースを作成します。これには、Amazon Simple Storage Service、Amazon Relational Database Service、、AWS Key Management Serviceおよび が含まれますAWS Secrets Manager。

トピック

- リソースをクリーンアップする
- 次のステップ

前提条件

- CICS 接続を使用するには、3270 エミュレータにアクセスできることを確認してください。無料およびトライアル 3270 エミュレータは、サードパーティーのウェブサイトから入手できます。または、 AWS Mainframe Modernization AppStream 2.0 Micro Focus インスタンスを起動し、Rumba 3270 エミュレータを使用することもできます (無料では使用できません)。

  AppStream 2.0 の詳細については、「」を参照してくださいthe section called “チュートリアル: Enterprise Analyzer と Enterprise Developer 向け AppStream 2.0 のセットアップ”。

  > **Note**
  >
  > スタックを作成するときは、Enterprise Analyzer (EA) ではなく、Enterprise Developer (ED) オプションを選択します。

- CardDemo サンプルアプリケーションをダウンロードし、ダウンロードしたファイルを任意のローカルディレクトリに解凍します。このディレクトリには、 というタイトルのサブディレクトリが含まれます`CardDemo`。
- このチュートリアルで作成したリソースを定義できるアカウント内の VPC を特定します。VPC には、少なくとも 2 つのアベイラビリティーゾーンのサブネットが必要です。Amazon VPC の詳細については、「Amazon VPC の仕組み」を参照してください。

ステップ 1: Amazon S3 バケットを作成してロードする

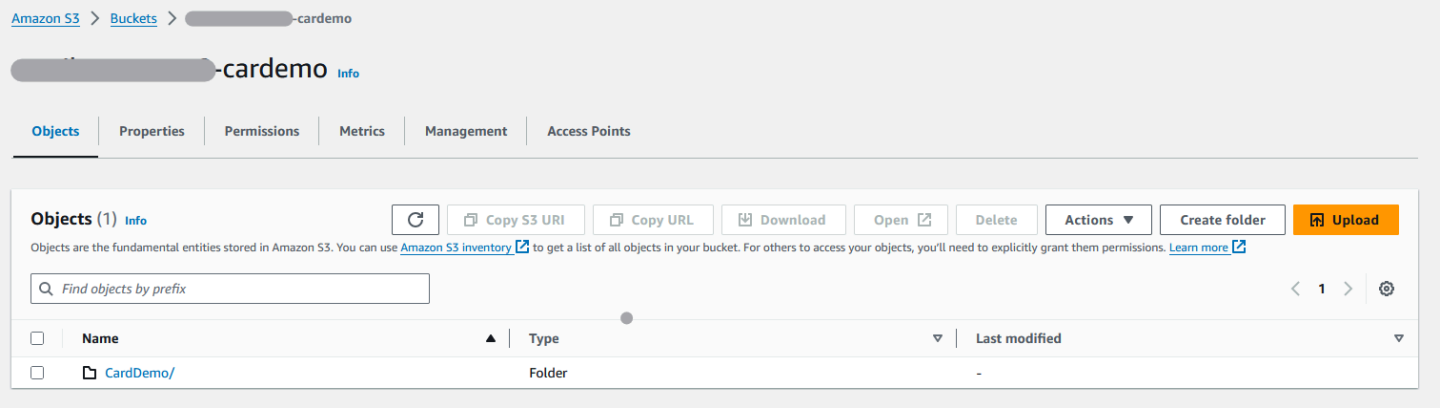
このステップでは、Amazon S3 バケットを作成し、このバケットに CardDemo ファイルをアップロードします。このチュートリアルの後半では、これらのファイルを使用して、 AWS Mainframe Modernization Micro Focus Managed Runtime 環境で CardDemo サンプルアプリケーションをデプロイして実行します。

> **Note**
>
> 新しい S3 バケットを作成する必要はありませんが、選択するバケットは、このチュートリアルで使用する他のリソースと同じリージョンにある必要があります。

Amazon S3 バケットを作成するには

1. Amazon S3 コンソール を開き、バケットの作成 を選択します。
2. 全般設定 で、 AWS Mainframe Modernization Micro Focus Managed Runtime を構築する AWS リージョンを選択します。
3. バケット名を入力しますyourname-aws-region-carddemo。例えば、。デフォルト設定のままで、バケットの作成を選択します。または、既存の Amazon S3 バケットから設定をコピーし、バケットの作成 を選択することもできます。
4. 先ほど作成したバケットを選択し、アップロードを選択します。
5. アップロードセクションで、フォルダを追加を選択し、ローカルコンピュータからCardDemoディレクトリを参照します。
6. アップロードを選択してアップロードプロセスを開始します。アップロード時間は、接続速度によって異なります。
7. アップロードが完了したら、すべてのファイルが正常にアップロードされたことを確認してから、 を閉じるを選択します。

Amazon S3 バケットに CardDemoフォルダが含まれるようになりました。



S3 バケットの詳細については、「Amazon S3 バケットの作成、設定、操作」を参照してください。

ステップ 2: データベースを作成して設定する

このステップでは、Amazon Relational Database Service (Amazon RDS) に PostgreSQL データベースを作成します。 Amazon Relational Database Service このチュートリアルでは、このデータベースには、 CardDemo サンプルアプリケーションがクレジットカード取引に関する顧客タスクに使用するデータセットが含まれています。

Amazon RDS にデータベースを作成するには

1. Amazon RDS コンソールを開きます。
2. データベースインスタンスを作成する AWS リージョンを選択します。
3. ナビゲーションペインから、[Databases] (データベース) を選択します。
4. データベースの作成 を選択し、標準作成 を選択します。
5. [エンジンタイプ] として、[PostgreSQL] を選択します。
6. 15 以上のエンジンバージョンを選択します。

> **Note**
>
> このチュートリアルの後半で必要になるため、エンジンバージョンを保存します。

7. [テンプレート] で、[無料利用枠] を選択します。
8. DB インスタンス識別子を、 など、わかりやすいものに変更しますMicroFocus-Tutorial。
9. でマスター認証情報を管理することを忘れないでくださいAWS Secrets Manager。代わりに、マスターパスワードを入力して確認します。

> **Note**
>
> データベースに使用するユーザー名とパスワードを保存します。これらは、このチュートリアルの次のステップで安全に保存します。

10. 接続 で、 AWS Mainframe Modernization マネージドランタイム環境を作成する VPC を選択します。
11. [データベースの作成] を選択します。

Amazon RDS でカスタムパラメータグループを作成するには

1. Amazon RDS コンソールのナビゲーションペインで、パラメータグループ を選択し、パラメータグループ の作成 を選択します。
2. パラメータグループの作成ウィンドウで、パラメータグループファミリー で、データベースバージョンに一致する Postgres オプションを選択します。

> **Note**
>
> 一部の Postgres バージョンでは、タイプ が必要です。必要に応じて DB パラメータグループを選択します。パラメータグループのグループ名と説明を入力します。

3. [作成] を選択します。

カスタムパラメータグループを設定するには

1. 新しく作成したパラメータグループを選択します。
2. [Actions] (アクション) を選択して、[Edit] (編集) を選択します。
3. をフィルタリング`max_prepared_transactions`し、パラメータ値を 100 に変更します。
4. [変更を保存] を選択します。

カスタムパラメータグループをデータベースに関連付けるには

1. Amazon RDS コンソールのナビゲーションペインで、データベース を選択し、変更するデータベースインスタンスを選択します。
2. [Modify] (変更) を選択します。Modify DB instance ページが表示されます。

> **Note**
>
> データベースの作成とバックアップが完了するまで、Modify オプションは使用できません。これには数分かかる場合があります。

3. DB インスタンスの変更ページで、追加設定 に移動し、DB パラメータグループをパラメータグループに変更します。パラメータグループがリストにない場合は、正しいデータベースバージョンで作成されたかどうかを確認します。
4. 続行を選択し、変更の概要を確認します。
5. すぐに適用を選択して、変更をすぐに適用します。
6. [DB インスタンスを変更] を選択して、変更を保存します。

パラメータグループの詳細については、「パラメータグループを使用する」を参照してください。

> **Note**
>
> AWS Mainframe Modernization で Amazon Aurora PostgreSQL データベースを使用することもできますが、無料利用枠オプションはありません。詳細については、「Amazon Aurora postgreSQL の使用」を参照してください。

ステップ 3: を作成して設定する AWS KMS key

Amazon RDS インスタンスの認証情報を安全に保存するには、まず を作成しますAWS KMS key。

AWS KMS key を作成するには

1. Key Management Service コンソールを開きます。
2. [Create Key] (キーを作成) を選択します。
3. キータイプには Symmetric を、キーの使用には暗号化と復号化はデフォルトのままにします。
4. [次へ] をクリックします。
5. キーに などのエイリアスMicroFocus-Tutorial-RDS-Keyとオプションの説明を付けます。
6. [次へ] をクリックします。
7. ユーザーまたはロールの横にあるチェックボックスをオンにして、キー管理者を割り当てます。
8. Next を選択し、もう一度 Next を選択します。
9. レビュー画面で、キーポリシー を編集し、次のように入力します。

```
{
     "Sid" : "Allow access for Mainframe Modernization Service",
     "Effect" : "Allow",
        "Principal" : {
           "Service" : "m2.amazonaws.com"
                 },
      "Action" : "kms:Decrypt",
      "Resource" : "*"
},
```

   このポリシーは、この特定のキーポリシーを使用して AWS Mainframe Modernization の復号化のアクセス許可を付与します。

10. [Finish] (完了) を選択し、KMS キーを作成します。

詳細については、「 AWS Key Management Serviceデベロッパーガイド」の「キーの作成」を参照してください。

ステップ 4: AWS Secrets Manager データベースシークレットを作成して設定する

ここで、 AWS Secrets Managerと を使用してデータベース認証情報を安全に保存しますAWS KMS key。

AWS Secrets Manager データベースシークレットを作成して設定するには

1. Secrets Manager コンソールを開きます。
2. ナビゲーションペインで [シークレット] を選択します。
3. 「シークレット」で、「新しいシークレットを保存する」を選択します。
4. シークレットタイプを Amazon RDS データベースの認証情報に設定します。
5. データベースの作成時に指定した認証情報を入力します。
6. 暗号化キー で、ステップ 3 で作成したキーを選択します。
7. データベースセクションで、このチュートリアル用に作成したデータベースを選択し、次へを選択します。
8. シークレット名 に、 などの名前MicroFocus-Tutorial-RDS-Secretとオプションの説明を入力します。
9. 「リソースのアクセス許可」セクションで「アクセス許可の編集」を選択し、内容を次のポリシーに置き換えます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect" : "Allow",
            "Principal" : {
                "Service" : "m2.amazonaws.com"
            },
            "Action" : "secretsmanager:GetSecretValue",
            "Resource" : "*"
        }
    ]
}
```

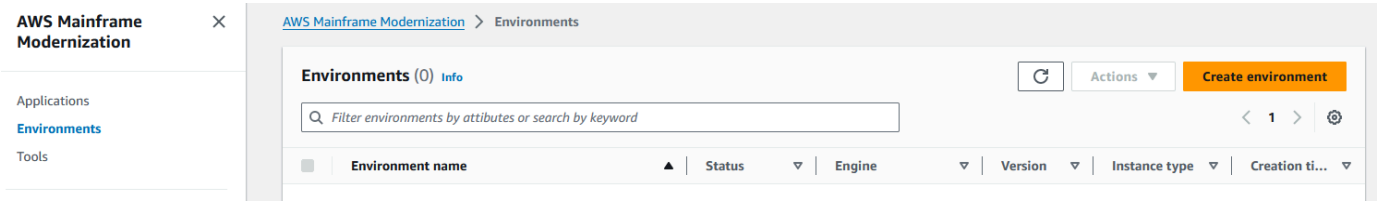
10. [保存] を選択します。
11. 後続の画面で次へを選択し、保存を選択します。シークレットリストを更新して、新しいシークレットを確認します。
12. チュートリアルの後半で必要`Secret ARN`になるため、新しく作成したシークレットを選択し、を書き留めます。
13. シークレットの概要タブで、シークレット値の取得を選択します。
14. 編集 を選択し、行を追加 を選択します。
15. の値`sslMode`を持つ のキーを追加します`verify-full`。

16. [保存] を選択します。

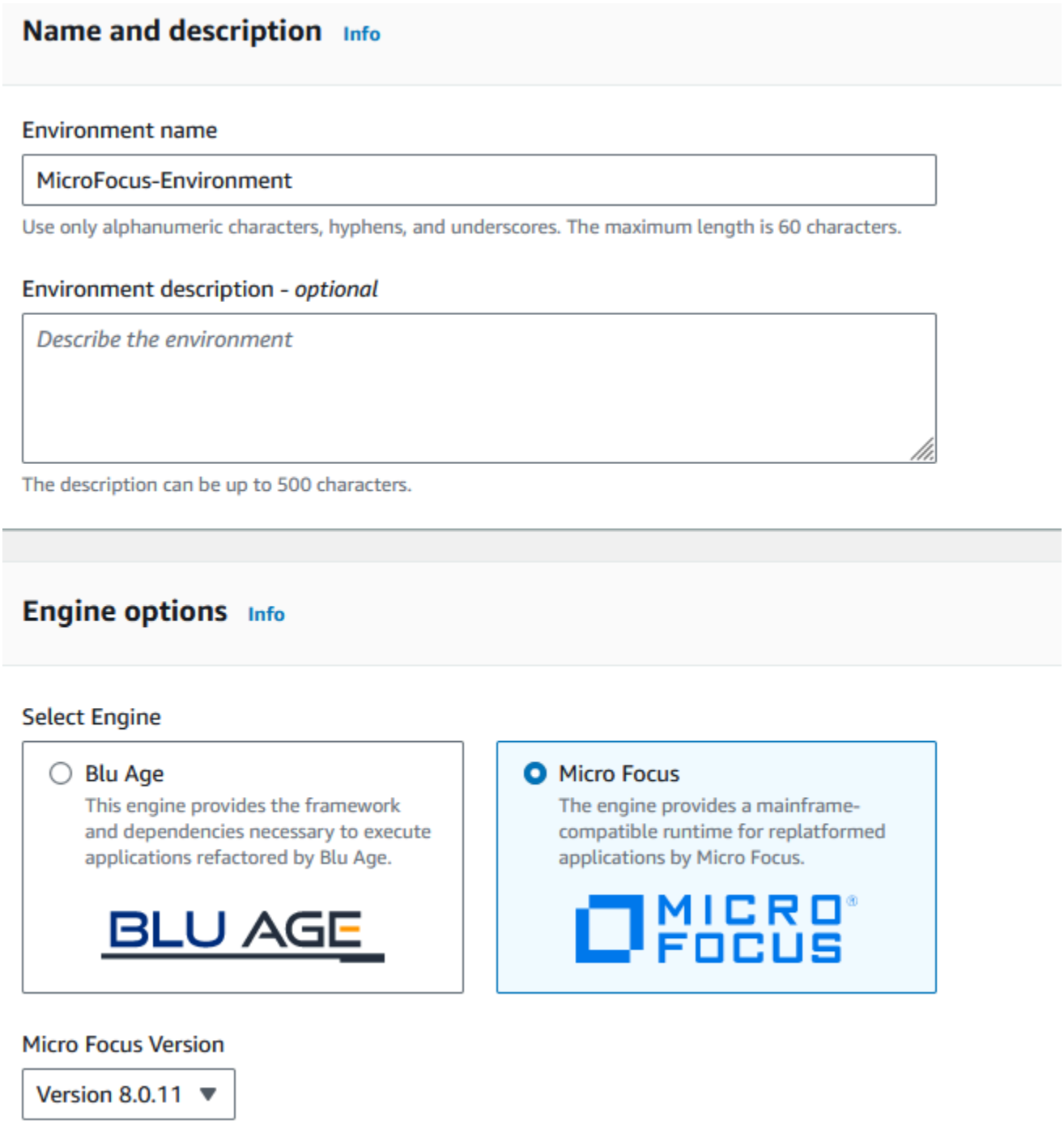
ステップ 5: ランタイム環境を作成する

ランタイム環境を作成するには

1. AWS Mainframe Modernization コンソールを開きます。
2. ナビゲーションペインで [環境] を選択します。次に、環境の作成を選択します。



3. 「基本情報の指定」で、
   a. 環境名に`MicroFocus-Environment`「」と入力します。
   b. エンジンオプションで、Micro Focus が選択されていることを確認します。
   c. 最新の Micro Focus バージョン を選択します。
   d. [次へ] をクリックします。



4. 環境を設定する

   a. [可用性] で [高可用性クラスター] を選択します。

   b. リソース で、インスタンスタイプ`M2.m5.large`に `M2.c5.large`または のいずれかを選択し、必要なインスタンスの数を選択します。最大 2 つのインスタンスを指定します。

c. セキュリティとネットワーク で、この環境にデプロイされたアプリケーションをパブリックにアクセス可能にし、少なくとも 2 つのパブリックサブネットを選択します。

d. [次へ] をクリックします。

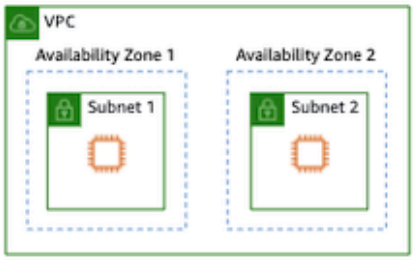
Specify configurations Info

Availability Info

Choose the availability pattern for your environment.

- Standalone runtime environment
  Sets up a single instance in a single availability zone. Does not guarantee high availability but costs less.

- High availability cluster
  Sets up redundant instances across two availability zones. Enables higher availability but costs more.



Resources

Instance type
Choose the instance type for your high availability cluster.

M2.m5.large

Desired capacity
Specify the desired number of instances.

2

Security and network

☑ Allow applications deployed to this environment to be publicly accessible.

Virtual Private Cloud (VPC)
Choose the VPC where you want to create the environment.

Default vpc-15

Subnets
Choose one or more subnets for a high availability setup.

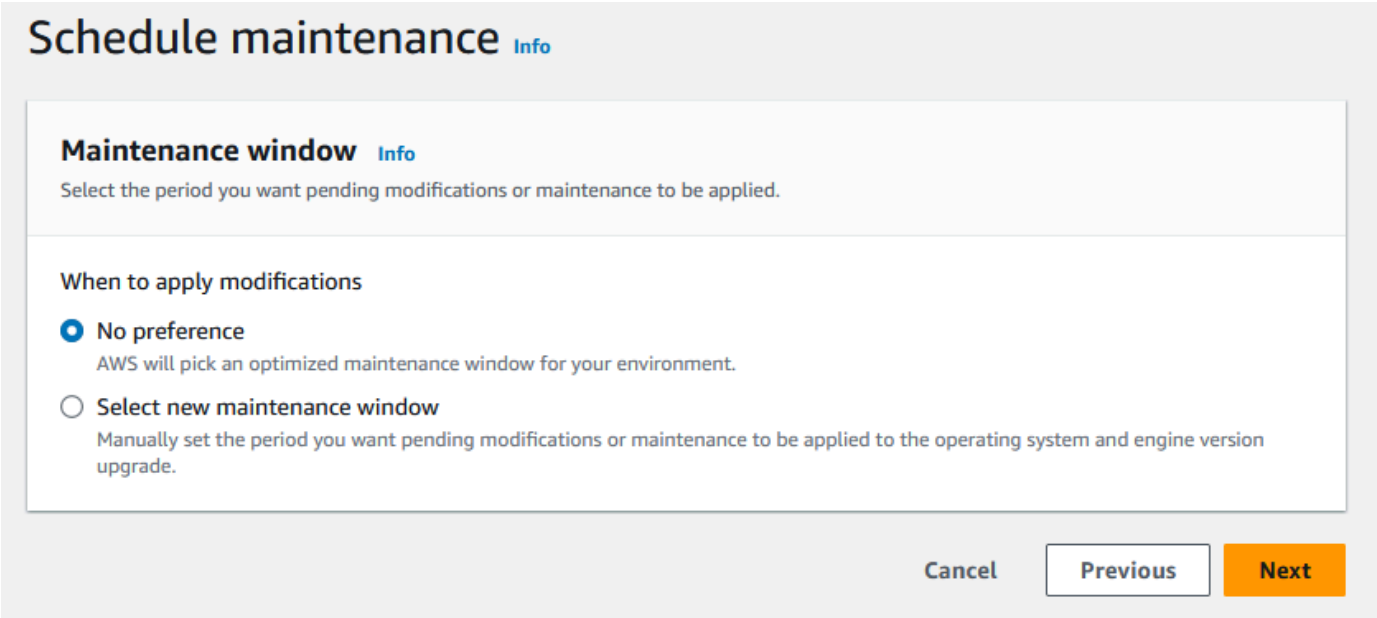
Choose subnets

subnet-56f1a | us-west-2a

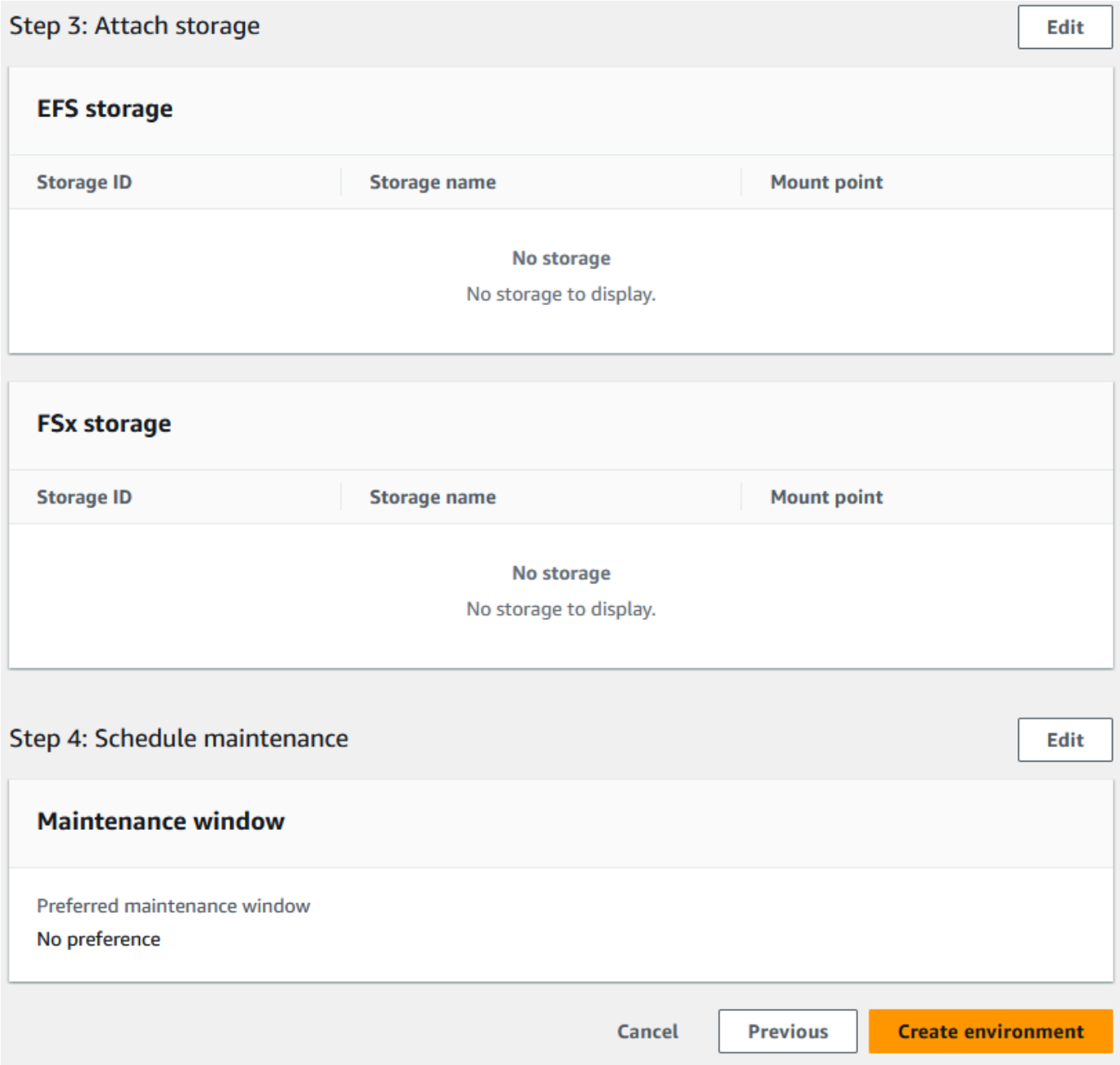
subnet-6685 | us-west-2b

Security groups
Choose one or more security groups for the chosen VPC.

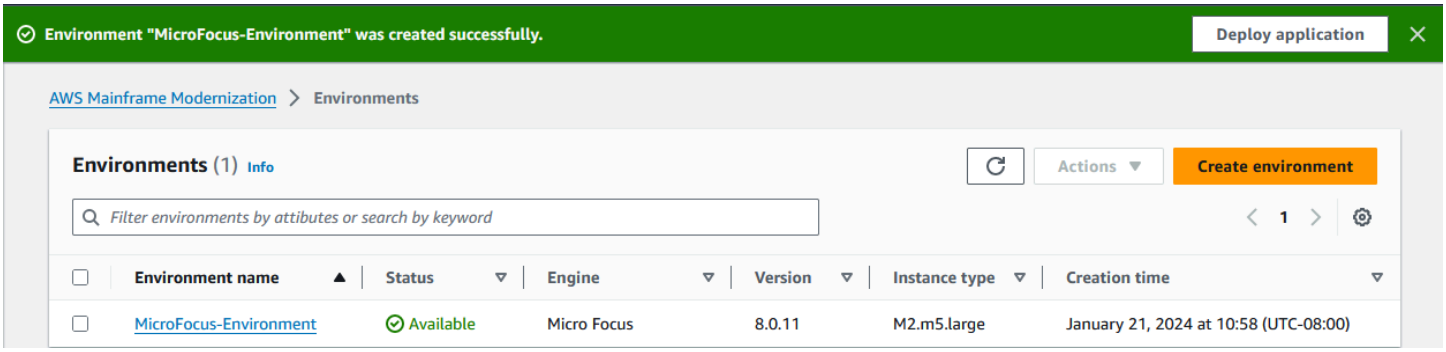
5. [ポリシーをアタッチ] ページで、[次へ] を選択します。
6. スケジュールメンテナンスページで、「設定なし」を選択し、「次へ」を選択します。



7. 確認と作成 ページで、ランタイム環境に指定したすべての設定を確認し、環境の作成 を選択します。



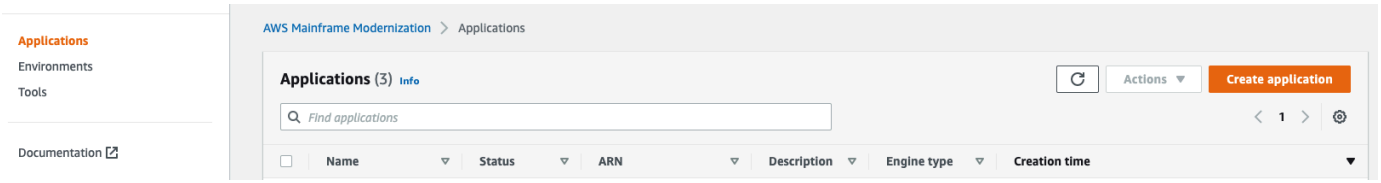
環境を作成すると、`Environment` *name* `was created successfully` というバナーが表示され、[ステータス] フィールドが [使用可能] に変わります。環境の作成プロセスには数分かかりますが、実行中は次のステップに進むことができます。



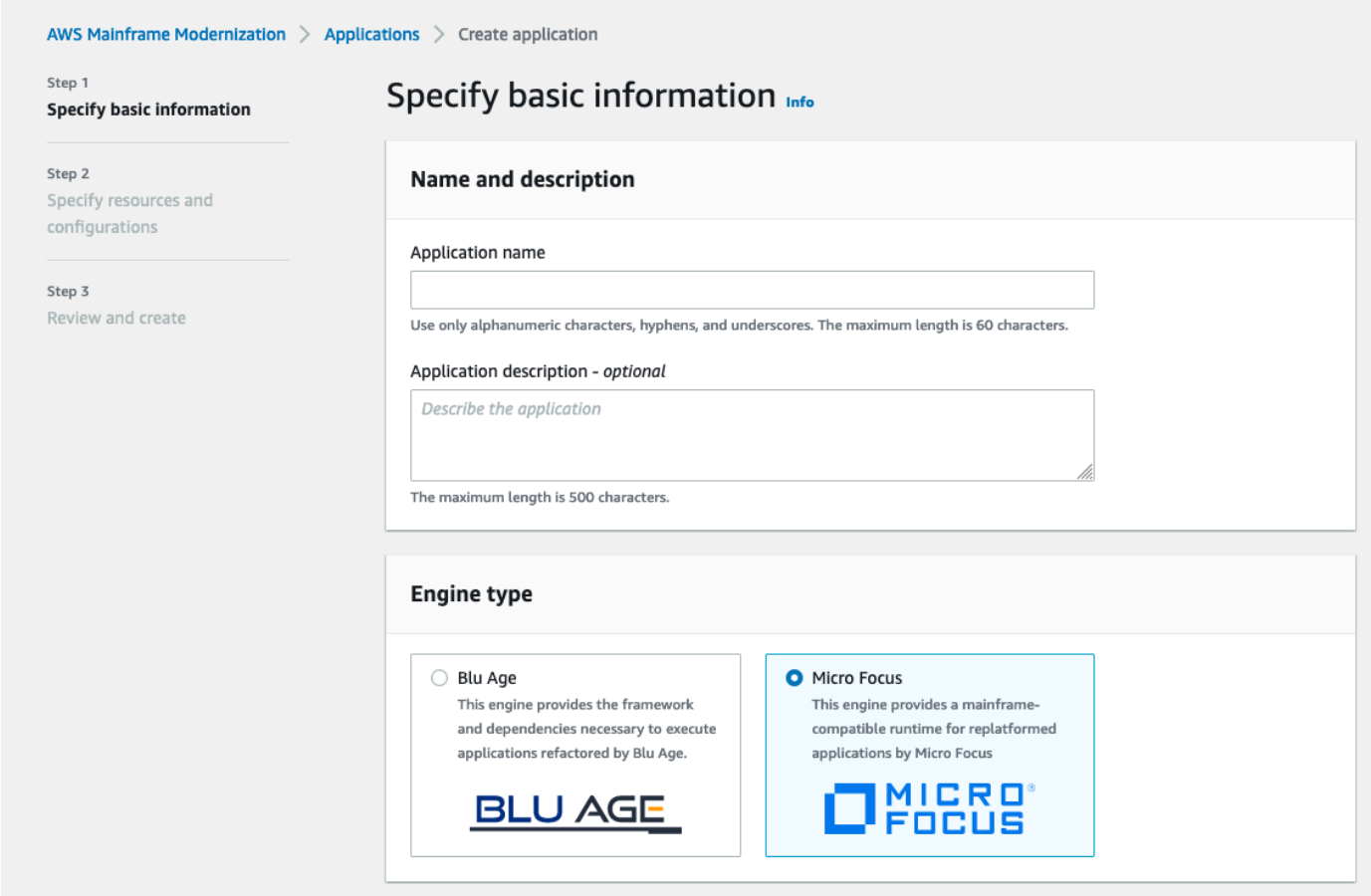
ステップ 6: アプリケーションを作成する

アプリケーションを作成するには

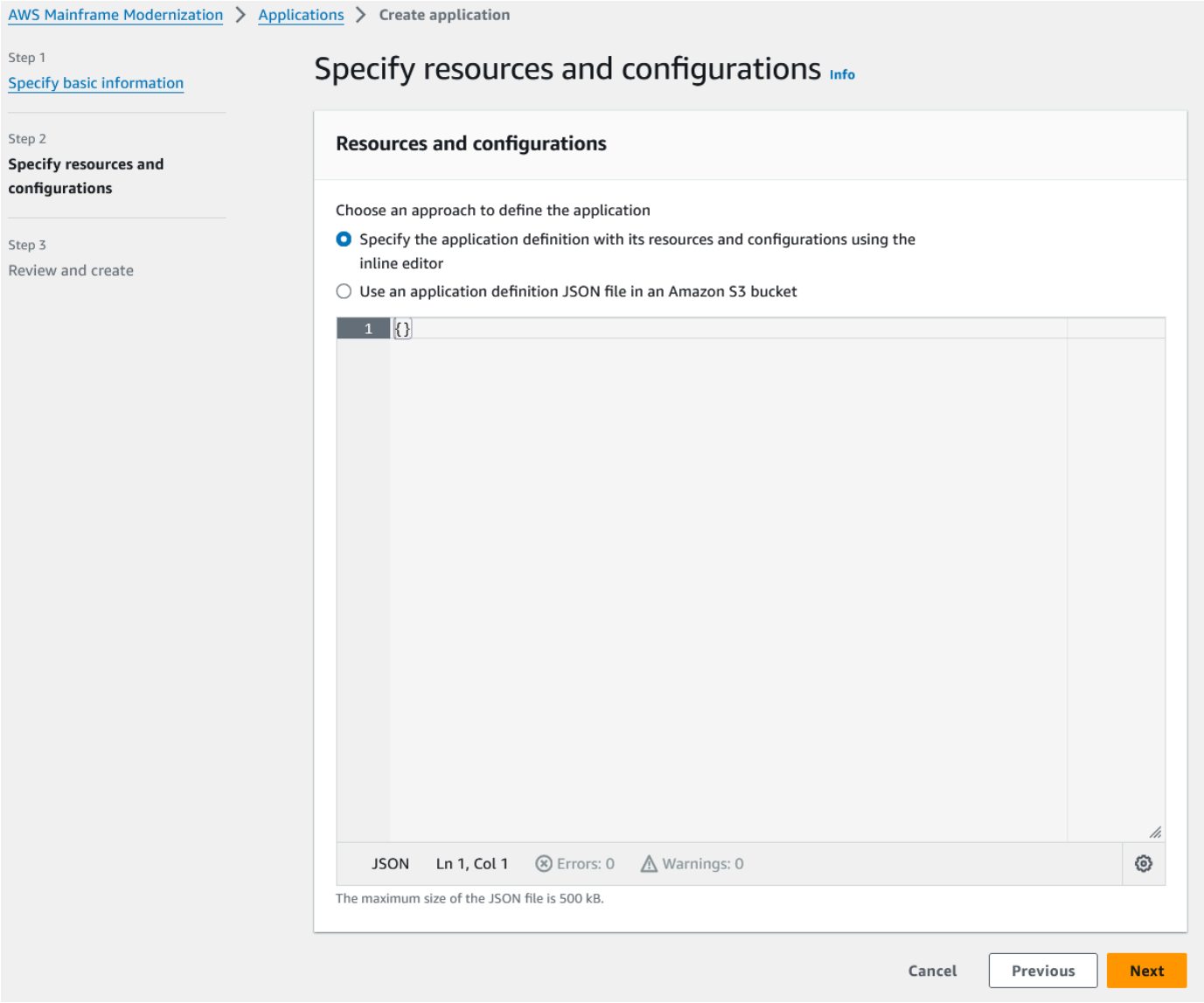
1. ナビゲーションペインで、[アプリケーション] を選択します。次に [アプリケーションの作成] を選択します。



2. 「アプリケーションの作成」ページの「基本情報の指定」で、アプリケーション名に「」と入力し、「エンジンタイプ」で「Micro Focus が選択されていることを確認します。 `MicroFocus-CardDemo`次いで、[次へ] を選択します。



3. 「リソースと設定の指定」で、インラインエディタを使用して、リソースと設定でアプリケーション定義を指定するオプションを選択します。



エディタに次のアプリケーション定義を入力します。

```
{
  "template-version": "2.0",
  "source-locations": [
    {
      "source-id": "s3-source",
      "source-type": "s3",
      "properties": {
        "s3-bucket": "yourname-aws-region-carddemo",
        "s3-key-prefix": "CardDemo"
      }
    }
```

```
  ],
  "definition": {
    "listeners": [
      {
        "port": 6000,
        "type": "tn3270"
      }
    ],
    "dataset-location": {
      "db-locations": [
        {
          "name": "Database1",
          "secret-manager-arn":
 "arn:aws:secretsmanager:Region:123456789012:secret:MicroFocus-Tutorial-RDS-Secret-
xxxxxx"
        }
      ]
    },
    "batch-settings": {
      "initiators": [
        {
          "classes": [
               "A",
               "B"
            ],
          "description": "initiator_AB...."
        },
        {
          "classes": [
                "C",
                "D"
             ],
          "description": "initiator_CD...."
        }
      ],
      "jcl-file-location": "${s3-source}/catalog/jcl"
    },
    "cics-settings": {
      "binary-file-location": "${s3-source}/loadlib",
      "csd-file-location": "${s3-source}/rdef",
      "system-initialization-table": "CARDSIT"
    },
    "xa-resources": [
      {
```

```
        "name": "XASQL",
        "secret-manager-arn":
 "arn:aws:secretsmanager:Region:123456789012:secret:MicroFocus-Tutorial-RDS-Secret-
xxxxxx",
        "module": "${s3-source}/xa/ESPGSQLXA64.so"
      }
    ]
  }
}
```

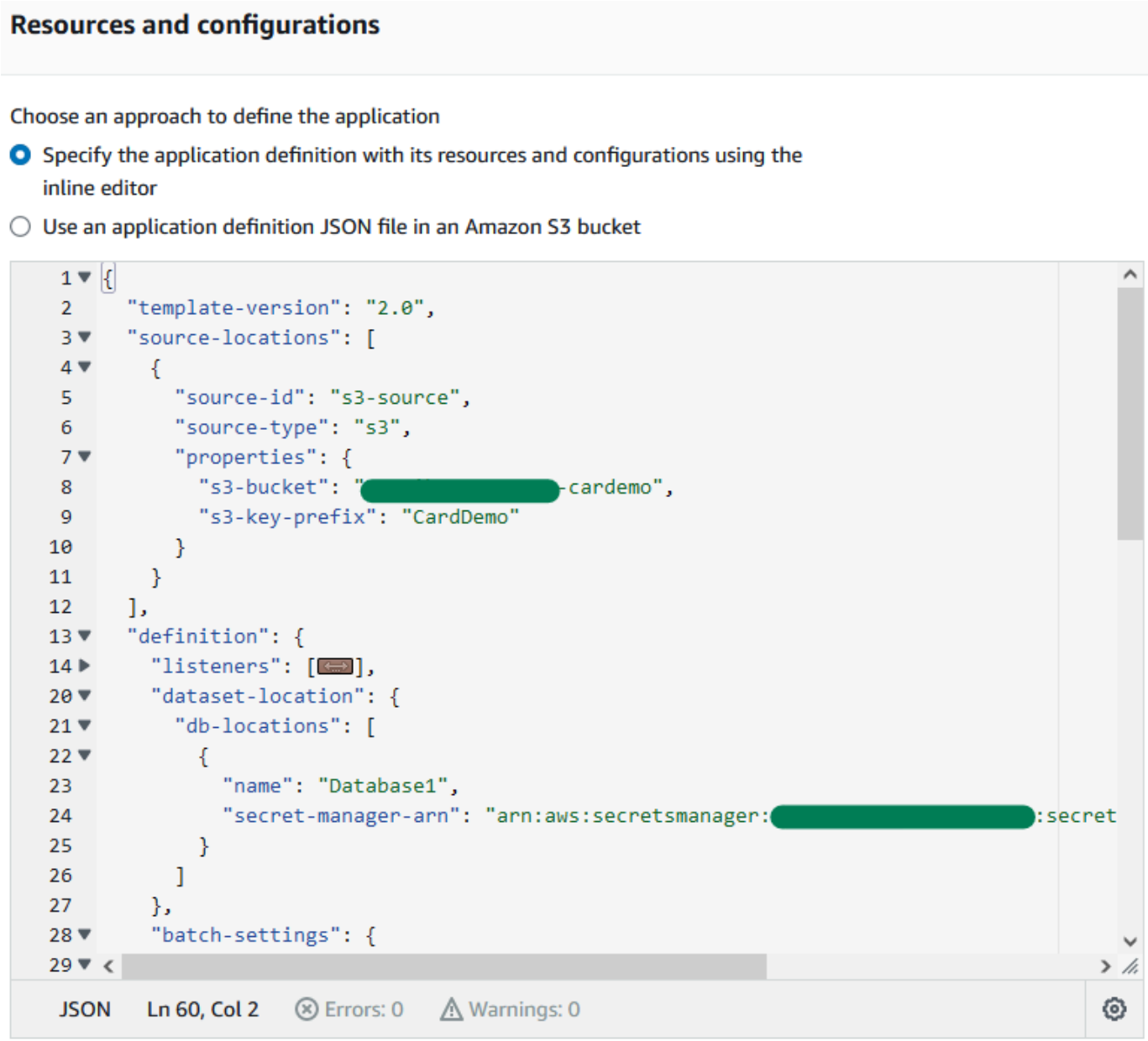
> **Note**
>
> このファイルは変更される可能性があります。

4. source-locations の properties オブジェクトのアプリケーション JSON を次のように編集します。

   a. の値を、ステップ 1 で作成した Amazon S3 バケットの名前s3_bucketに置き換えます。

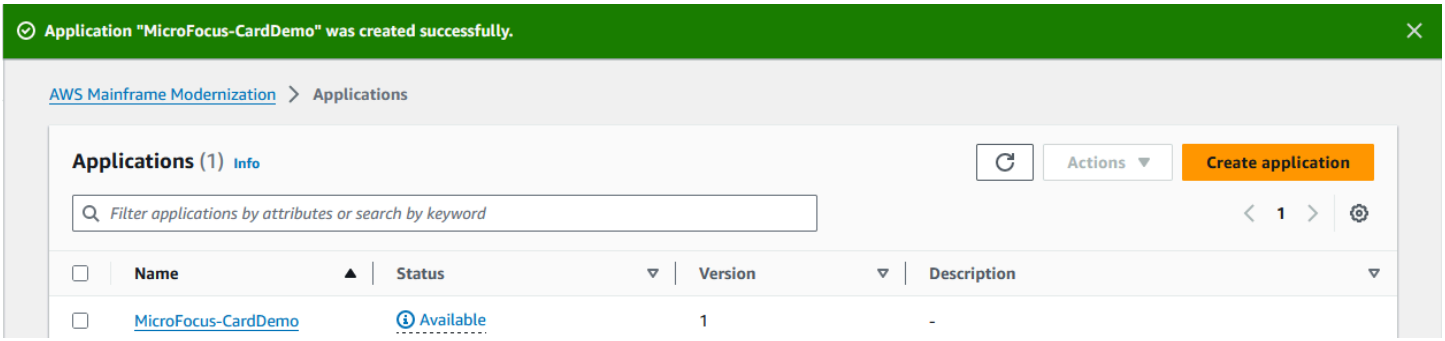
   b. の値を、 CardDemo サンプルファイルをアップロードしたフォルダ (キープレフィックス) s3-key-prefixに置き換えます。CardDemo ディレクトリを Amazon S3 バケットに直接アップロードした場合、 を変更s3-key-prefixする必要はありません。

   c. 両方のsecret-manager-arn値を、ステップ 4 で作成したデータベースシークレットの ARN に置き換えます。



アプリケーション定義の詳細については、「Micro Focus アプリケーション定義」を参照してください。

5. [次へ] を選択して続行します。
6. 確認と作成 ページで、指定した情報を確認し、アプリケーションの作成 を選択します。

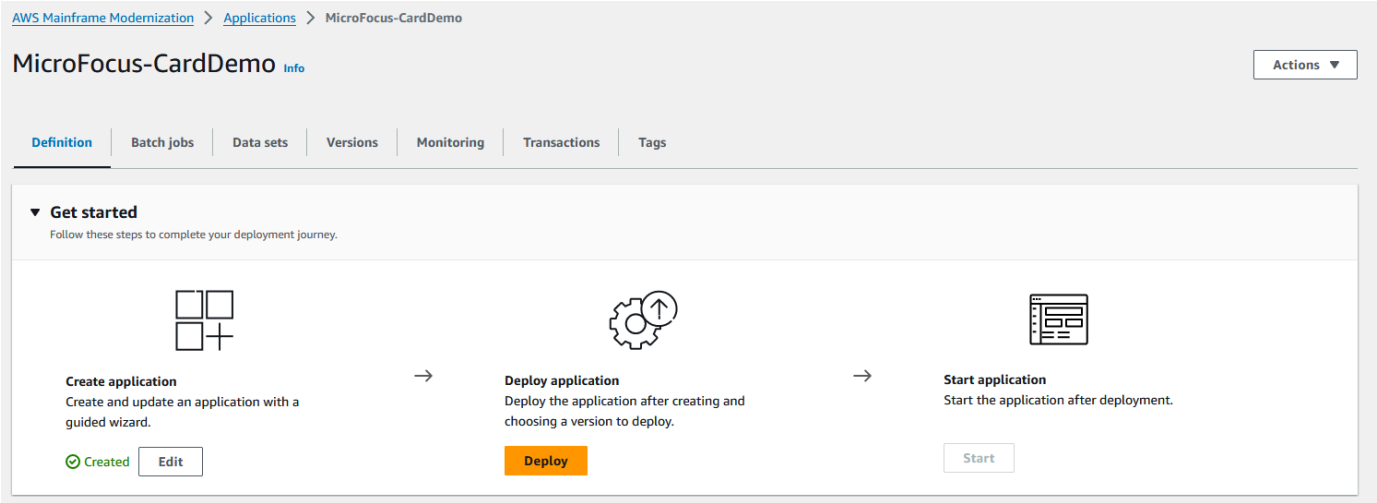


アプリケーションを作成すると、 というバナーが表示されますApplication *name* was created successfully。ステータス フィールドが Available に変わります。

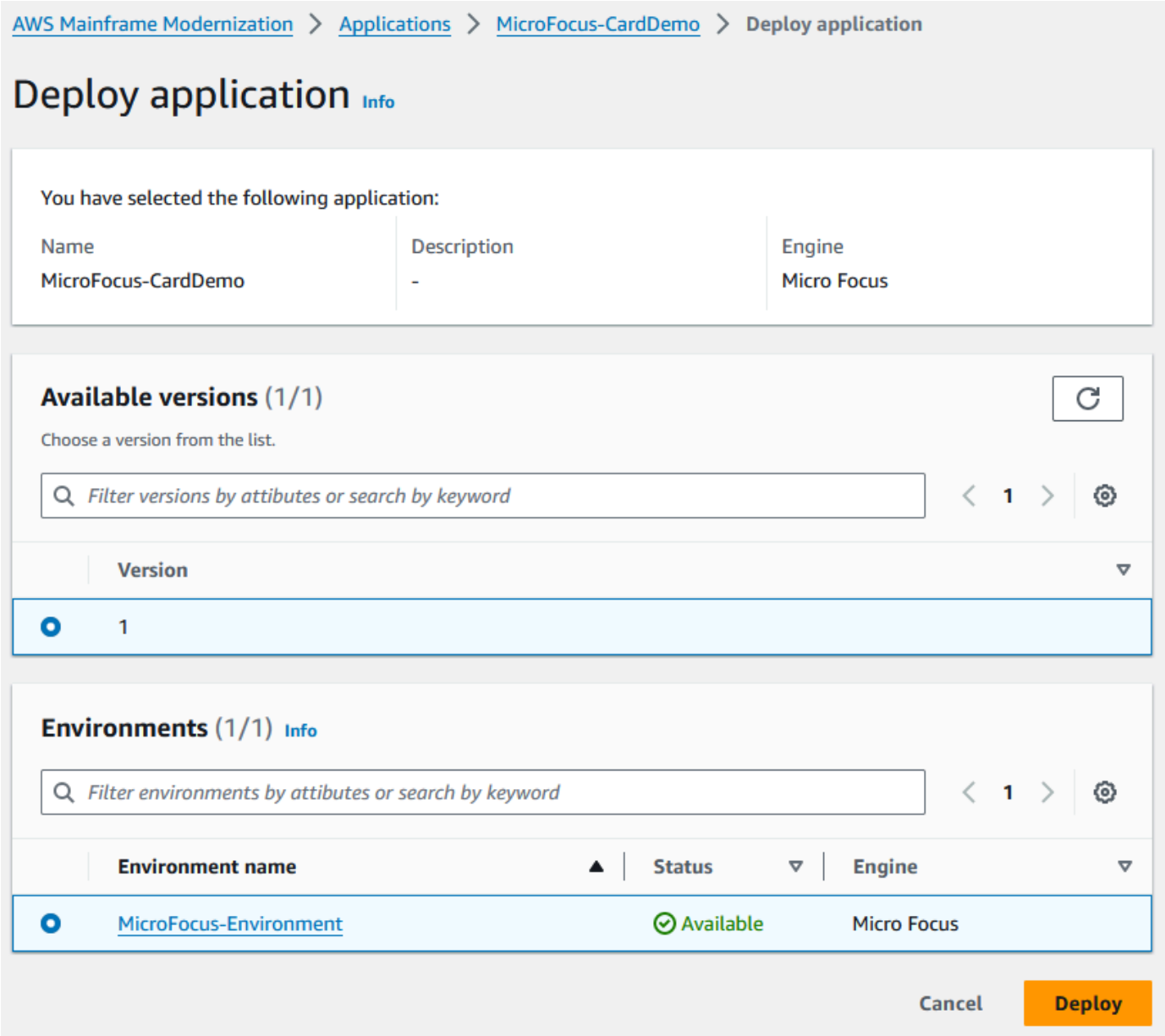
ステップ 7: アプリケーションをデプロイする

アプリケーションをデプロイするには

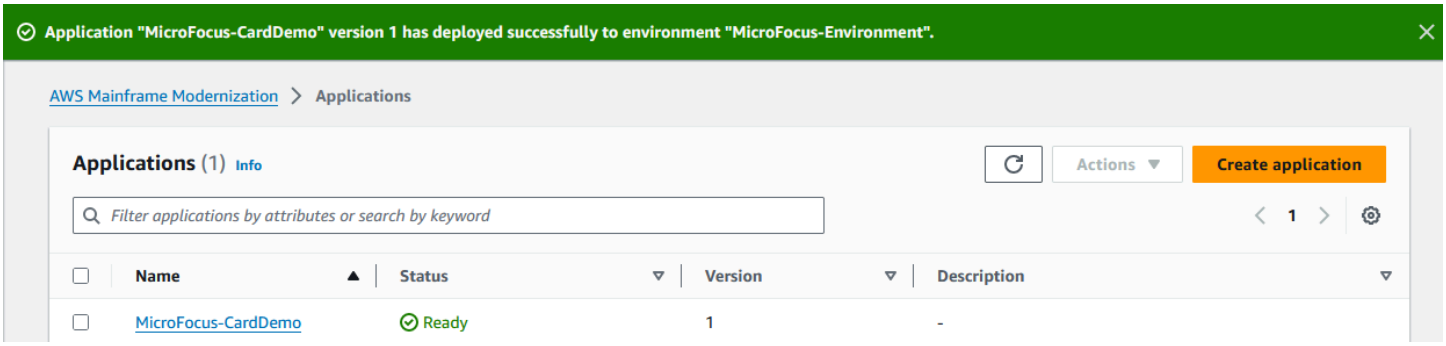
1. ナビゲーションペインで、アプリケーション を選択し、 を選択しますMicroFocus-CardDemo。
2. 「アプリケーションのデプロイ」で、「 のデプロイ」を選択します。



3. 以前に作成したアプリケーションの最新バージョンと環境を選択し、デプロイを選択します。



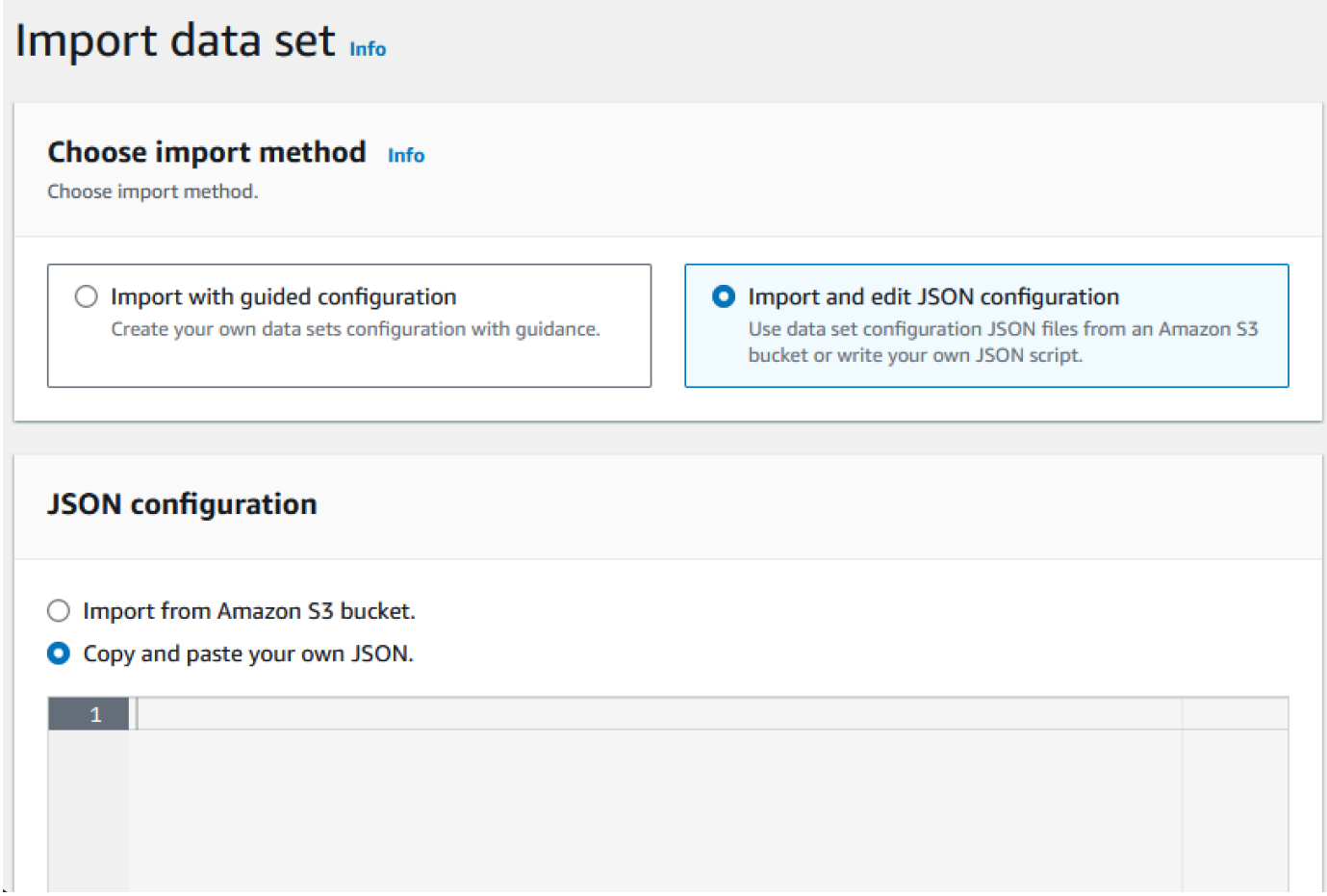
CardDemo アプリケーションが正常にデプロイされると、ステータスは Ready に変わります。



ステップ 8: データセットをインポートする

データセットをインポートするには

1. ナビゲーションペインで、アプリケーション を選択し、アプリケーションを選択します。
2. [データセット] タブを選択します。次に、[インポート] を選択します。
3. JSON 設定のインポートと編集を選択し、コピーして独自の JSON オプションを貼り付けます。



4. 次の JSON をコピーして貼り付けますが、まだ「送信」を選択しないでください。この JSON には、デモアプリケーションに必要なすべてのデータセットが含まれていますが、Amazon S3 バケットの詳細が必要です。

```
{
    "dataSets": [
        {
            "dataSet": {
                "storageType": "Database",
```

```
                "datasetName": "AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 11,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 300,
                    "max": 300
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX.PATH",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 11,
                            "offset": 16
                        }
                    }
                },
                "recordLength": {
                    "min": 150,
                    "max": 150
                }
            },
            "externalLocation": {
```

```
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 150,
                    "max": 150
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
```

```
                "recordLength": {
                    "min": 50,
                    "max": 50
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 9,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 500,
                    "max": 500
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
```

```
                        "primaryKey": {
                            "length": 11,
                            "offset": 25
                        }
                    }
                },
                "recordLength": {
                    "min": 50,
                    "max": 50
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 350,
                    "max": 350
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
```

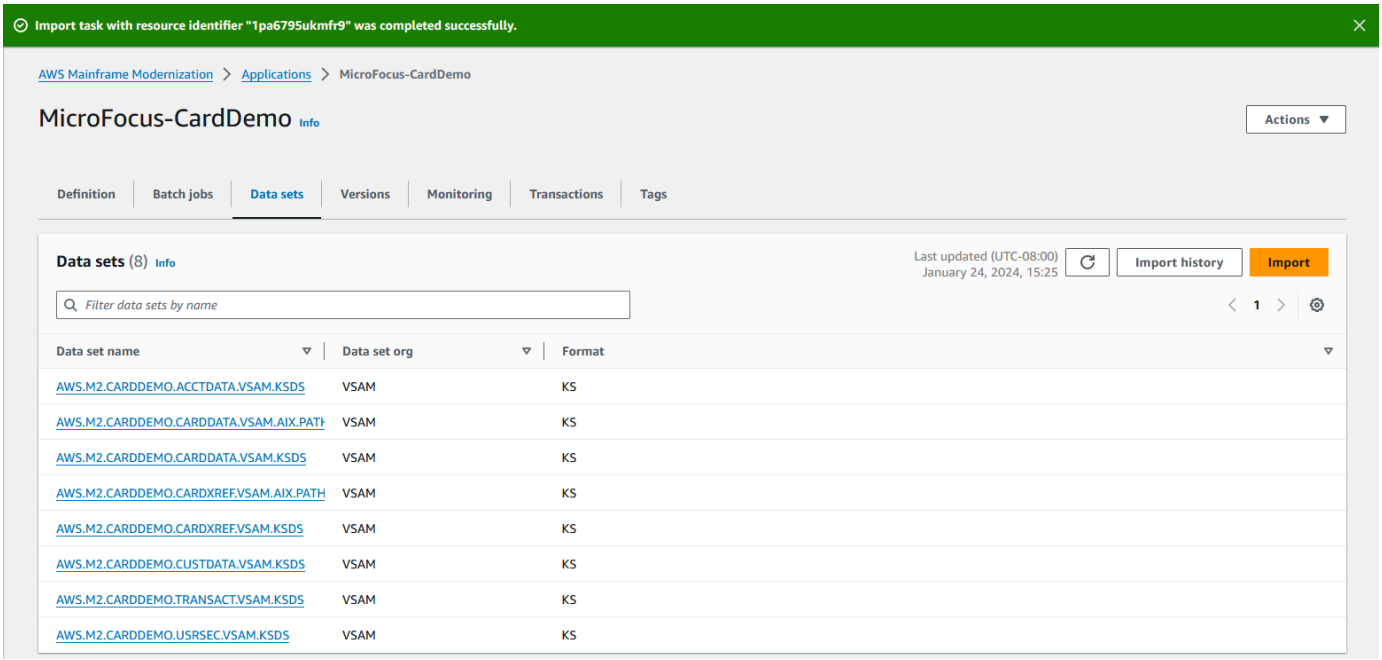
```
                "datasetName": "AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 8,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 80,
                    "max": 80
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo/catalog/data/
AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS.DAT"
            }
        }
    ]
}
```

5. 各出現 <s3-bucket-name> (8 つある) を、 CardDemo フォルダを含む Amazon S3 バケットの名前に置き換えます。例えば、 ですyour-name-aws-region-carddemo。

> **Note**
>
> Amazon S3 内のフォルダの Amazon S3 URI をコピーするには、フォルダを選択し、Amazon Amazon S3URI のコピーを選択します。

6. 送信 を選択します。

インポートが完了すると、インポートされたデータセットImport task with resource identifier *name* was completed successfully.のリストを示すバナーが表示されます。

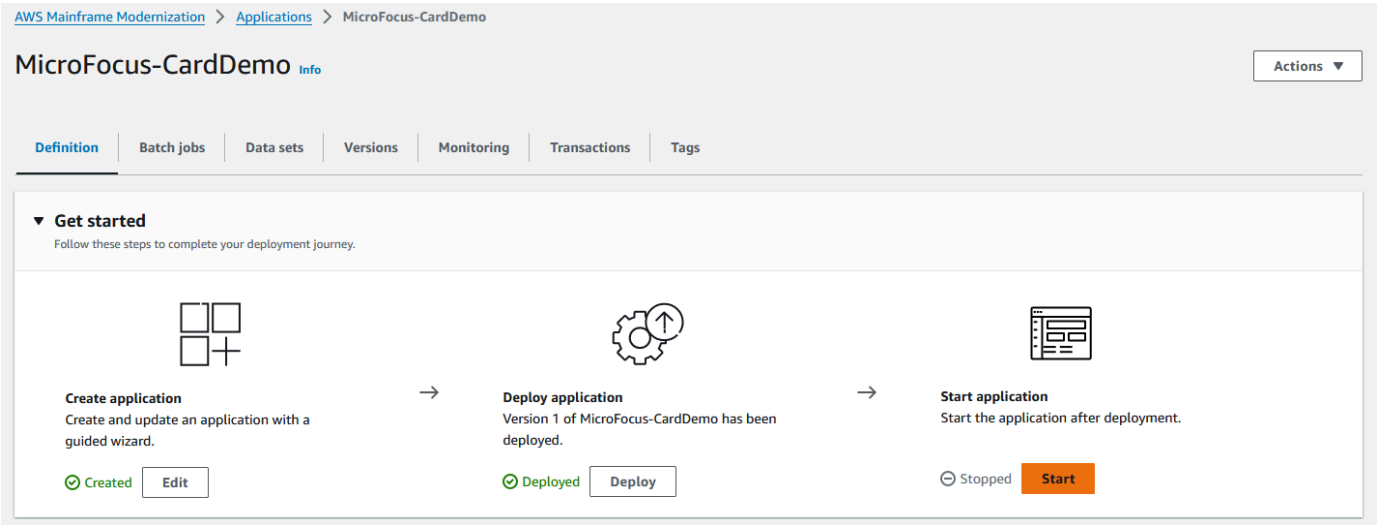


データセットタブでインポート履歴を選択して、すべてのデータセットのインポートのステータスを表示することもできます。

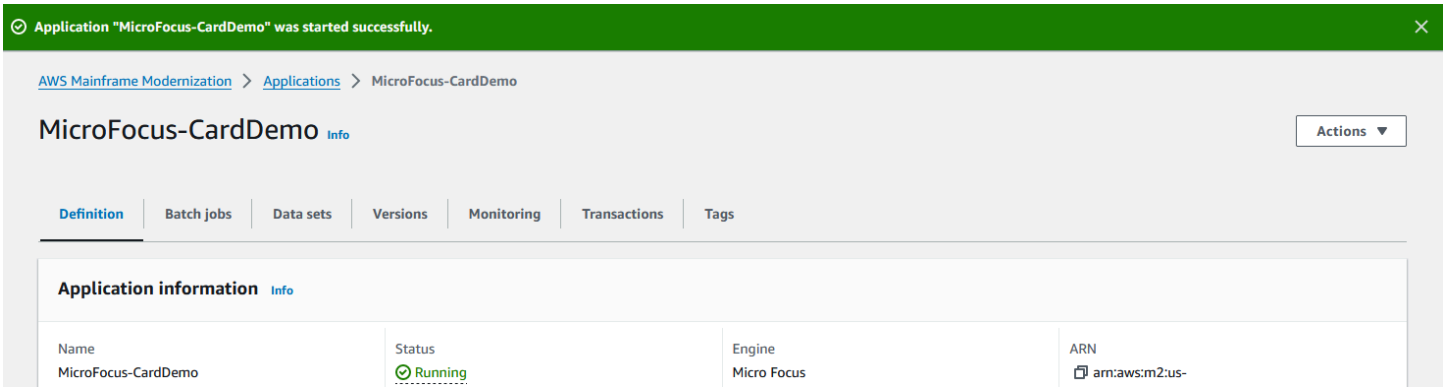
ステップ 9: アプリケーションを開始する

アプリケーションを起動するには

1. ナビゲーションペインで、アプリケーション を選択し、アプリケーションを選択します。
2. アプリケーションの開始を選択します。



CardDemo アプリケーションが正常に実行を開始すると、バナーに というメッセージが表示されますApplication *name* was started successfully。ステータス フィールドが実行中 に変わります。



ステップ 10: CardDemo CICS アプリケーションに接続する

接続する前に、アプリケーションに指定した VPC とセキュリティグループが、接続元のネットワークインターフェイスに適用したものと同じであることを確認します。

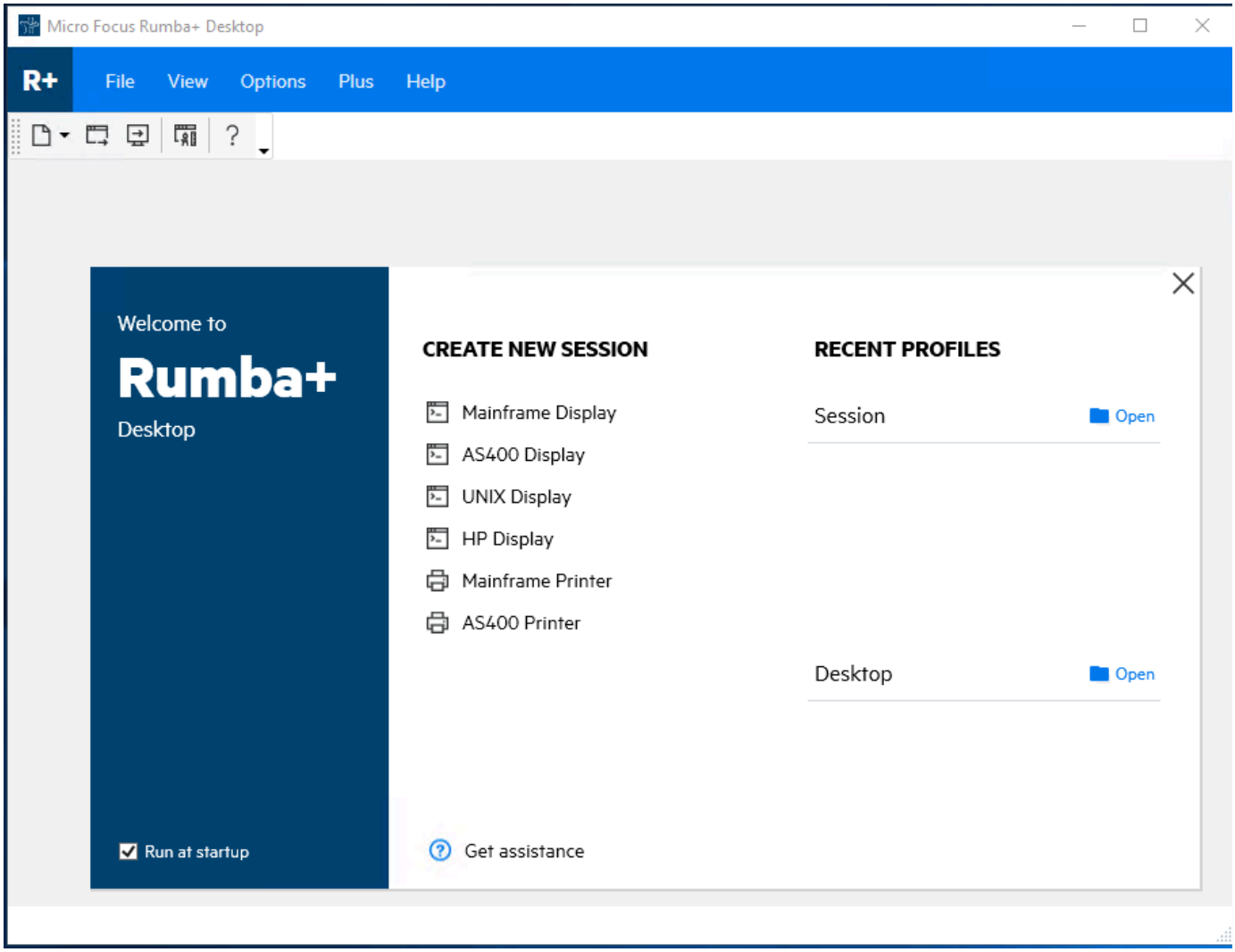
TN3270 接続を設定するには、DNS ホスト名とアプリケーションのポートも必要です。

ターミナルエミュレータを使用してアプリケーションを設定し、メインフレームに接続するには

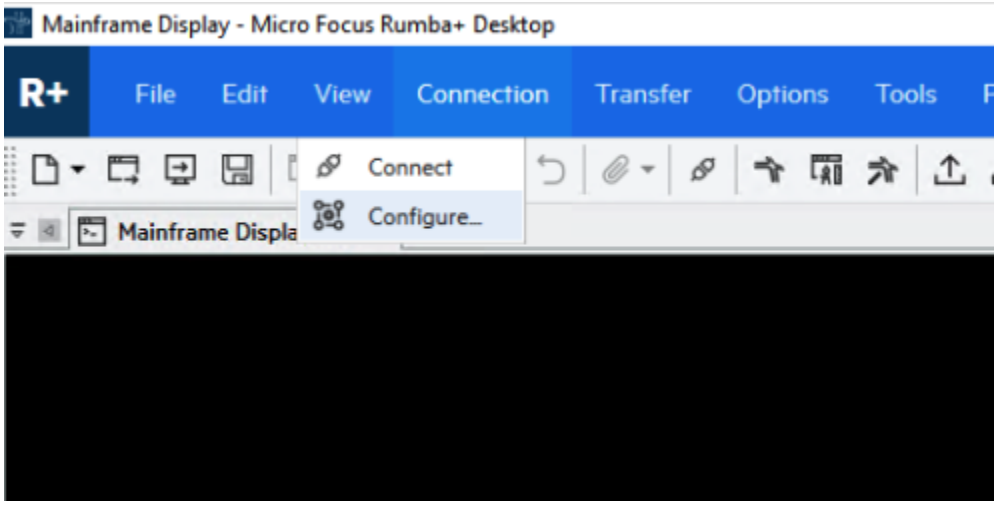
1. AWS Mainframe Modernization コンソールを開き、アプリケーション を選択し、 を選択しますMicroFocus-CardDemo。
2. コピーアイコンを選択して、DNS ホスト名 をコピーします。また、ポート番号も書き留めておきます。
3. ターミナルエミュレータを起動します。このチュートリアルでは、Micro Focus Rumba+ を使用しています。

   > **Note**
   >
   > 設定手順はエミュレータによって異なります。

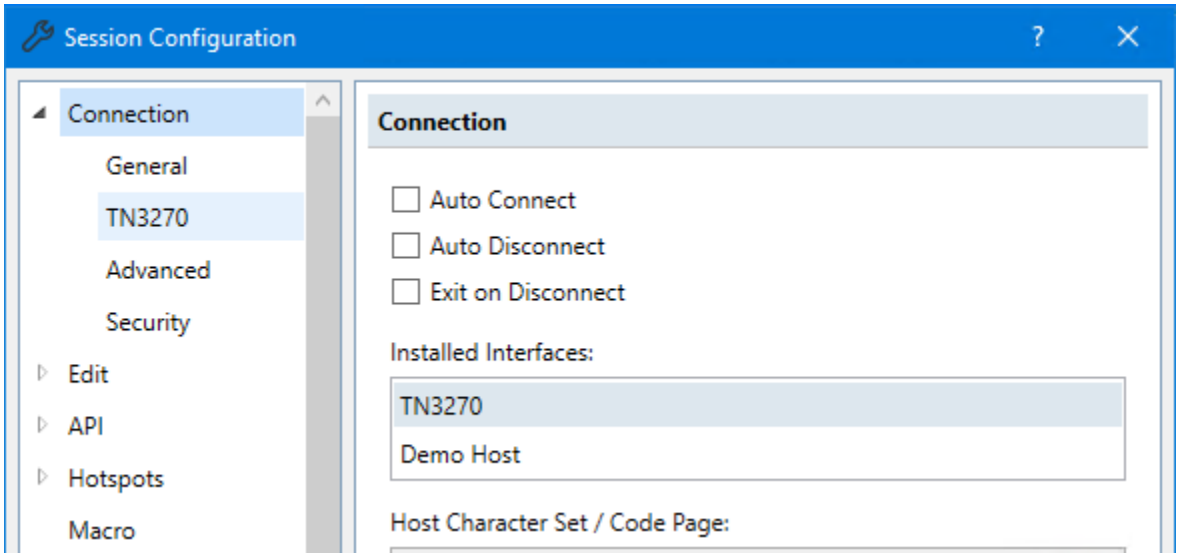
4. メインフレームディスプレイ を選択します。



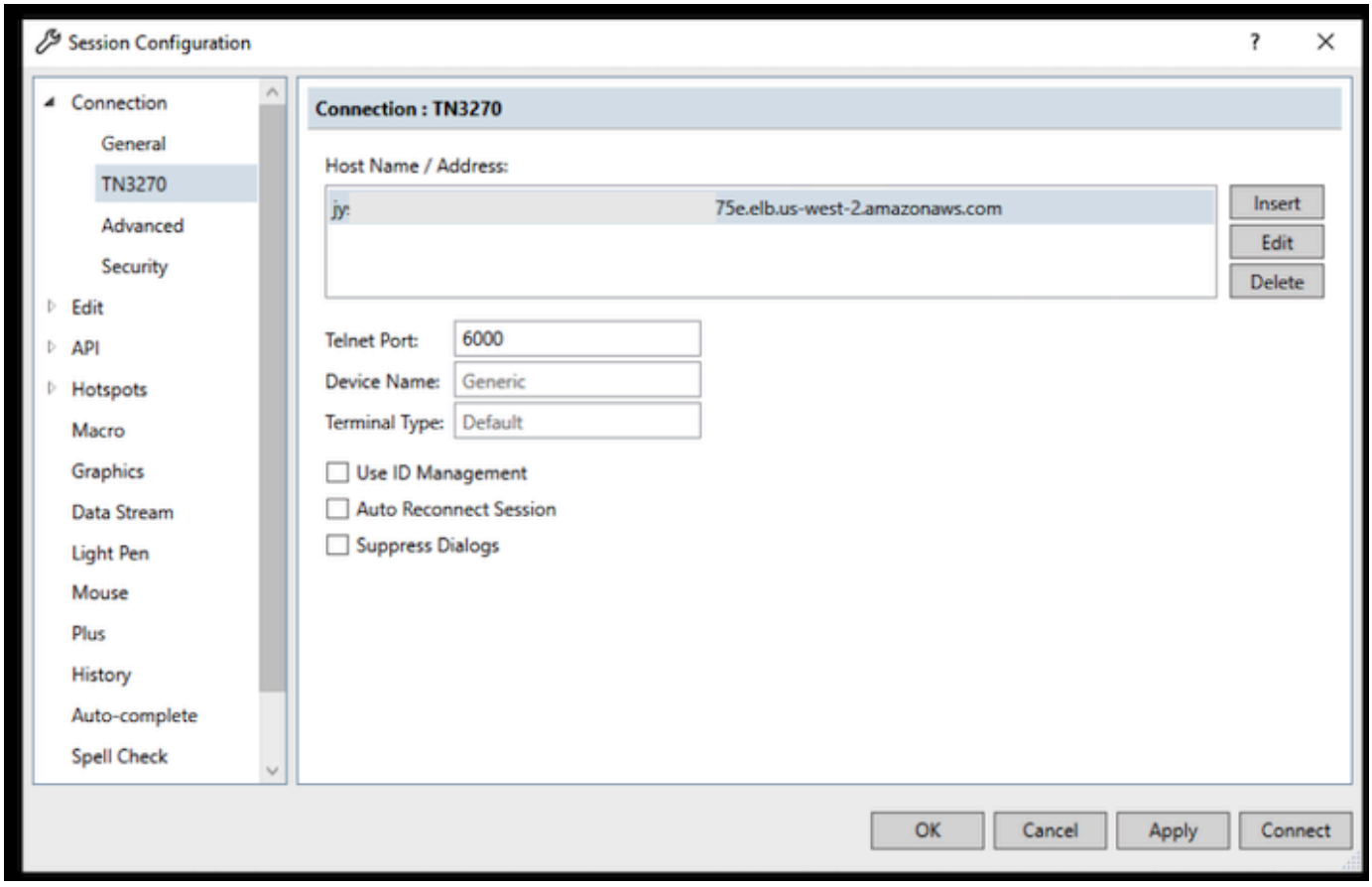
5. 接続 を選択し、設定 を選択します。



6. インストール済みインターフェイス で、 を選択しTN3270、接続メニューでTN3270もう一度を選択します。



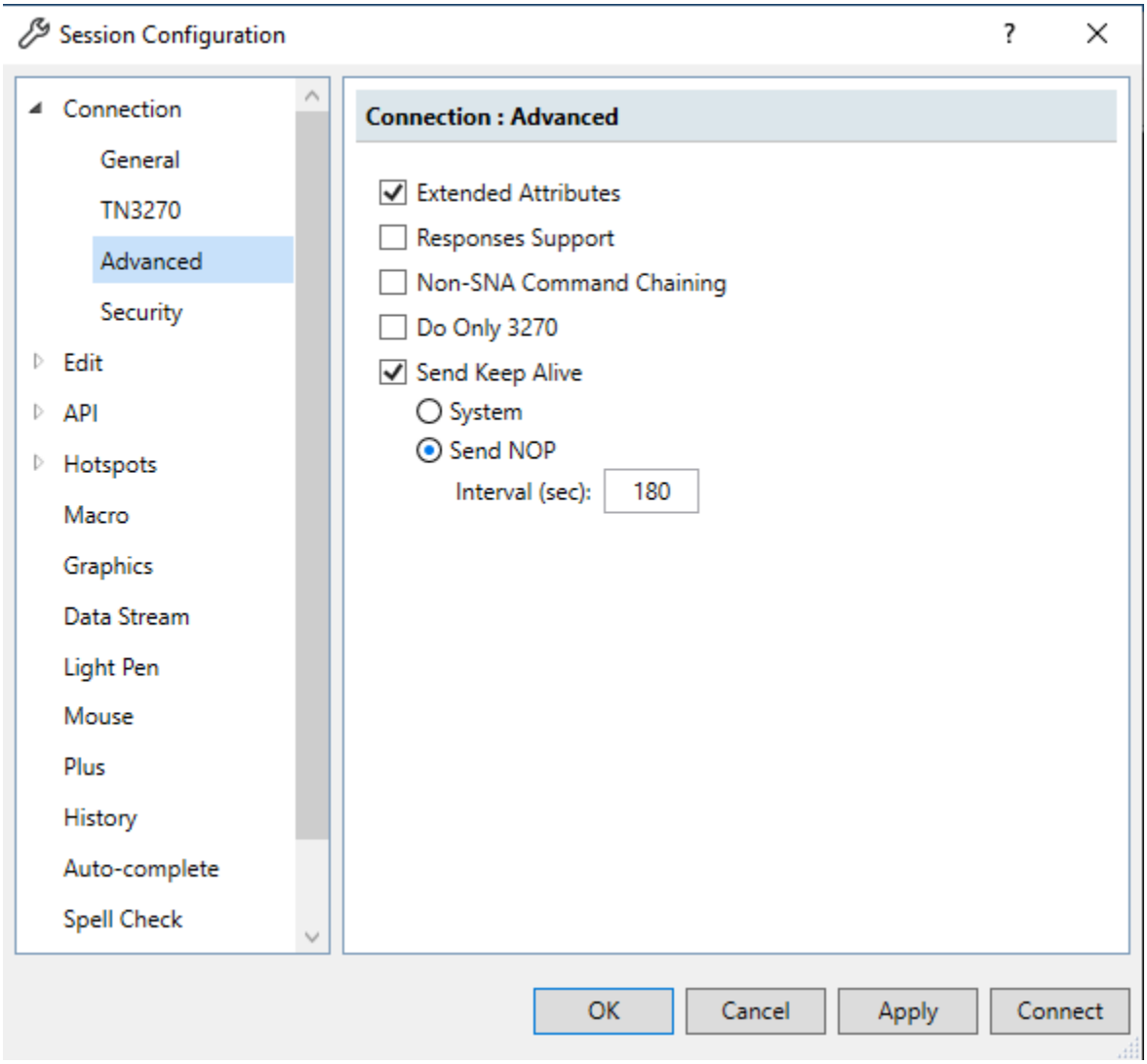
7. 挿入を選択し、アプリケーションの を貼り付けDNS Hostnameます。Telnet ポート6000に を指定します。



> **Note**
>
> ブラウザで AWS AppStream 2.0 を使用していて、値の貼り付けに問題がある場合は、AppStream 「2.0 ユーザーの問題のトラブルシューティング」を参照してください。

8. 「接続」で「アドバンスト」を選択し、「キープアライブの送信」と「NOP の送信」を選択し、間隔に 180 と入力します。

> **Note**
>
> TN3270 ターミナルのキープアライブ設定を少なくとも 180 秒に設定すると、Network Load Balancer が接続を切断しないようにできます。



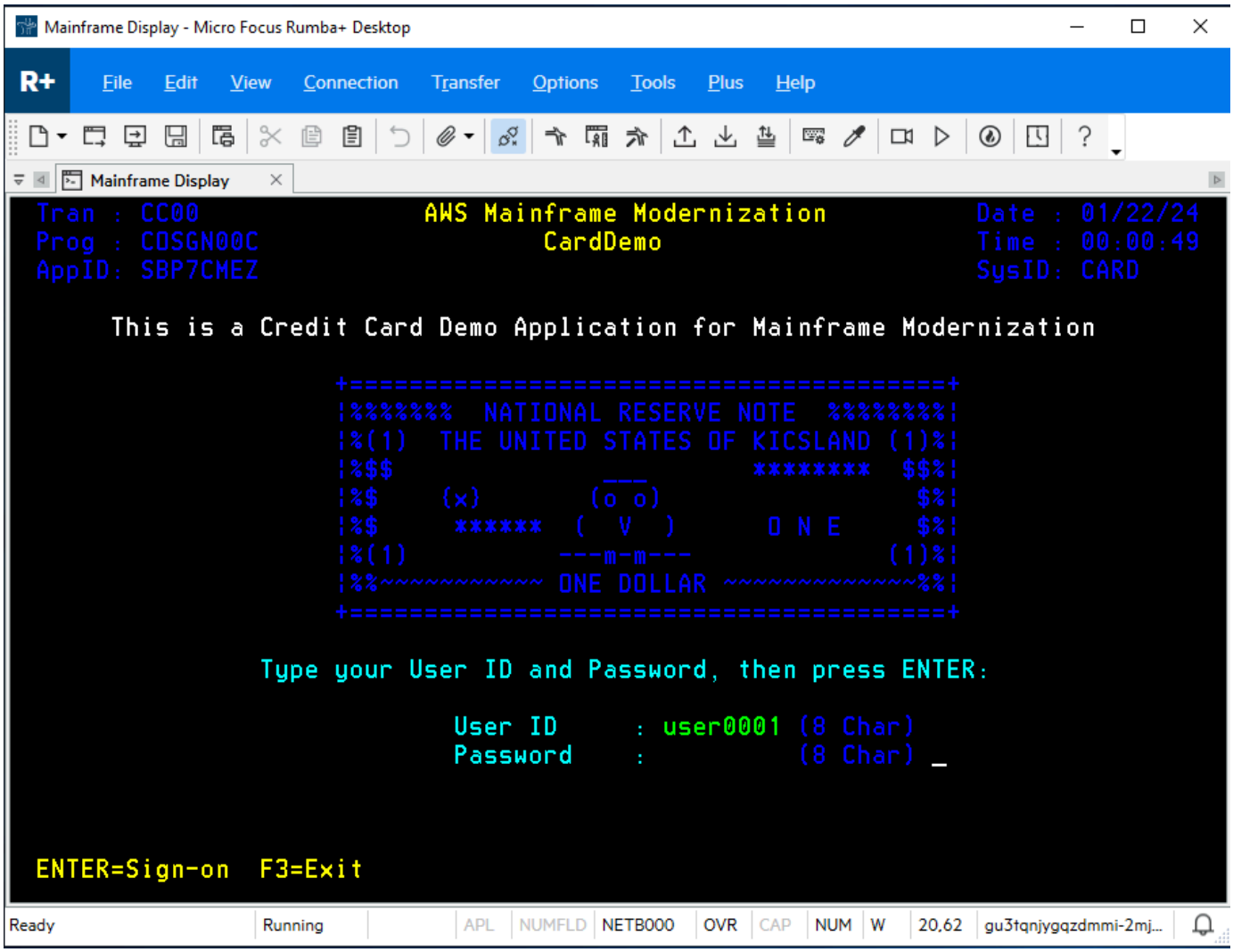
9. [接続]を選択します。

> **Note**
>
> 接続が失敗した場合：
>
> - AppStream 2.0 を使用している場合は、アプリケーションの環境に指定された VPC とセキュリティグループが AppStream 2.0 フリートと同じであることを確認します。
> - VPC Reachability Analyzer を使用して接続を分析します。Reachability Analyzer には、コンソール からアクセスできます。
> - 診断ステップとして、アプリケーションのセキュリティグループのインバウンドルールを追加または変更して、任意の場所 (CIDR ブロック 0.0.0.0/0) からのポート 6000

のトラフィックを許可してみてください。正常に接続できたら、セキュリティグループがトラフィックをブロックしていることがわかります。セキュリティグループのソースをより具体的に変更します。セキュリティグループの詳細については、「セキュリティグループの基本」を参照してください。

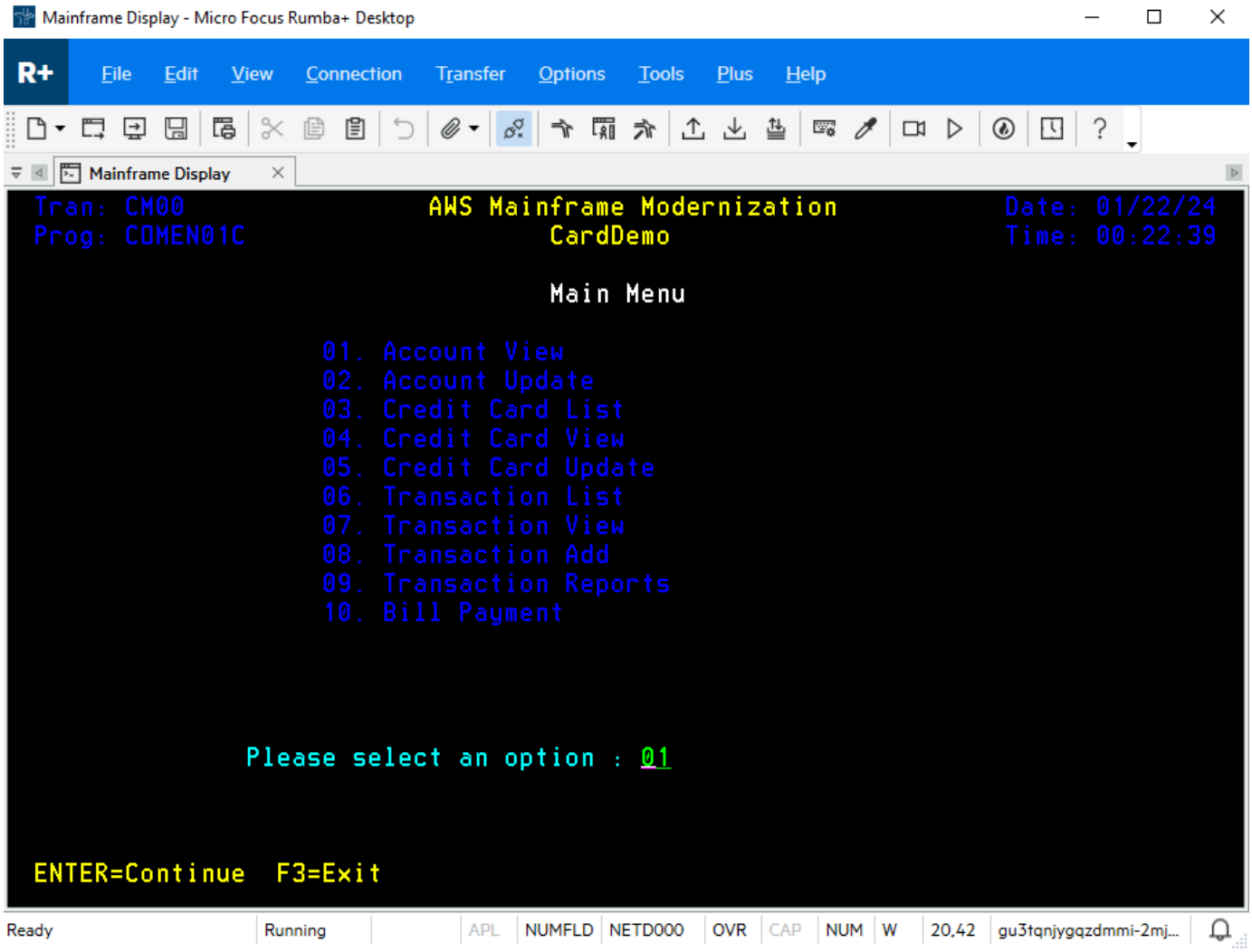
10. ユーザー名USER0001に 、パスワードに password を入力します。

> **Note**
>
> Rumba では、クリアのデフォルトは で ctrl-shift-z、リセットのデフォルトは ctrl-r です。



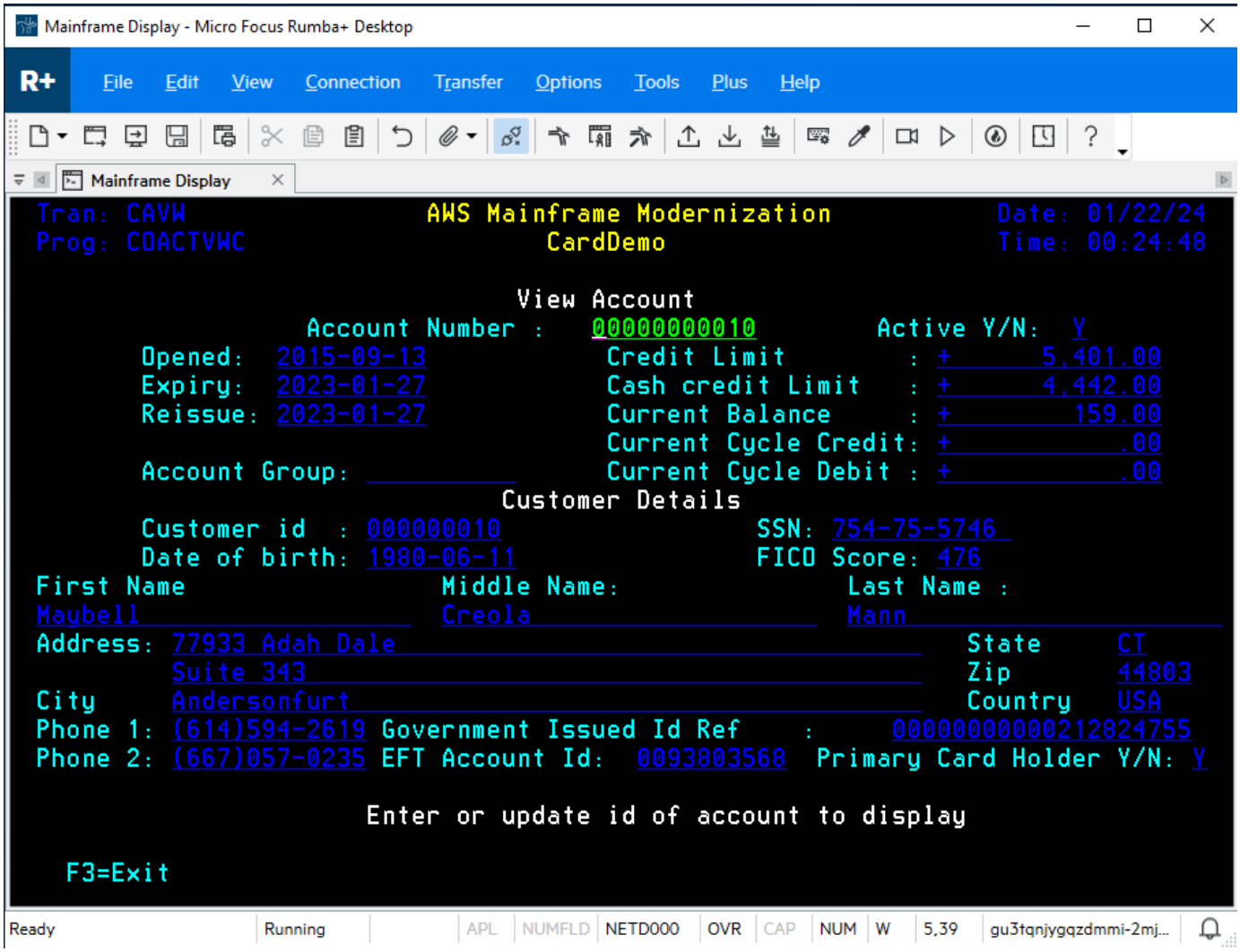
11. ログインに成功すると、アプリケーション内 CardDemoを移動できます。

12. アカウントビューに01「」と入力します。



13. アカウント番号00000000010に を入力し、キーボードの Enter キーを押します。

> **Note**
>
> その他の有効なアカウントは 0000000011と です00000000020。



14. F3 を押してメニューを終了し、 F3 を押してトランザクションを終了します。

リソースをクリーンアップする

このチュートリアルのために作成したリソースが不要になった場合は、追加料金が発生しないように削除します。そのためには、以下のステップを実行します。

- 必要に応じて、アプリケーションを停止します。
- アプリケーションを削除します。詳細については、「AWS Mainframe Modernization アプリケーションの削除」を参照してください。
- ランタイム環境を削除します。詳細については、「AWS Mainframe Modernization ランタイム環境を削除する」を参照してください。

- このチュートリアル用に作成した Amazon S3 バケットを削除します。詳細については、「Amazon S3 ユーザーガイド」の「バケットの削除」を参照してください。
- このチュートリアル用に作成したAWS Secrets Managerシークレットを削除します。詳細については、「シークレットの削除」を参照してください。
- このチュートリアル用に作成した KMS キーを削除します。詳細については、「AWS KMS キーの削除」を参照してください。
- このチュートリアル用に作成した Amazon RDS データベースを削除します。詳細については、「Amazon RDS ユーザーガイド」の「EC2 インスタンスと DB インスタンスの削除」を参照してください。
- ポート 6000 のセキュリティグループルールを追加した場合は、ルールを削除します。

次のステップ

モダナイズされたアプリケーションの開発環境を設定する方法については、「チュートリアル: Micro Focus Enterprise Analyzer および Micro Focus Enterprise Developer で使用するための AppStream 2.0 のセットアップ」を参照してください。

モダナイゼーションアプローチ

移行は複雑で、さまざまな不確定要素が多くあります。AWSMainframe Modernization は、短期的な成果をもたらす発展的なアプローチです。この成果は、俊敏性を向上させ、後で最適化や革新を行う機会を増やすことによって得られます。さらに、AWS Mainframe Modernization はプロセスを簡素化するのに役立ちますが、クライアントの会社やビジネスの特性を尊重します。AWS Mainframe Modernization がサポートする主な 2 つのアプローチは、自動リファクタリングとリプラットフォームです。どちらを選択するかは、クライアントの状況によって異なります。

自動リファクタリングでは、AWS Blu Age のツールを使用して、コードやデータ、依存関係を最新の言語、データストア、フレームワークに自動的に変換し、同じビジネス機能との機能的等価性を保証します。

リプラットフォームでは、Micro Focus のツールを使用して、メインフレームのワークロードを AWS のアジャイルサービスに変換します。

モダナイゼーションのジャーニーについては段階的に考えることができます。第 1 段階には、評価、準備、移行とモダナイズという 3 つのフェーズがあります。次の段階は運用と最適化の段階で、イノベーションの機会をさらに明確にすることができます。

トピック

評価フェーズ

全体を俯瞰して、評価フェーズでは移行の準備が整っているかどうかを調べます。ビジネスケースを定義し、AWS が提供するワークショップやイマージョンデー (デモとラボ) でチームを教育します。ワークショップとイマージョンデーでは、さまざまなトピックを取り上げています。これらのタスクは AWS Mainframe Modernization 以外で行われます。

準備フェーズ

準備フェーズでは、キックオフを行ってプロジェクトを開始し、メインフレームアプリケーションからデータを抽出して移行ツールに取り込む検出プロセスを実行します。移行するアプリケーションを

特定し、試行するアプリケーションをいくつか選択します。ビジネスケースを絞り込み、移行計画を立て、セキュリティとコンプライアンス、アカウントガバナンス、運用モデルをどのように処理するかを決定します。チームから適切な人材を集め、クラウドセンターオブエクセレンスを設置します。パイロット運用を実施し、わかったことを文書化します。移行計画とビジネスケースを改善します。これらのタスクの多くは、AWS Mainframe Modernization 以外で行われます。

移行とモダナイズフェーズ

移行とモダナイズのフェーズは各アプリケーションに適用され、人員の割り当て、詳細な調査、AWS の適切なアプリケーションアーキテクチャの検討、アプリケーションランタイム環境の設定、コードのリプラットフォームやリファクタリング、他のシステムとの統合、テストなど、いくつかのタスクで構成されます。フェーズの最後に、リプラットフォームまたはリファクタリングされたアプリケーションを本番環境にデプロイし、AWS の新しいシステムに移行します。これらのタスクのほとんどまたはすべてが、AWS Mainframe Modernization、別の AWS サービス、または AWS Mainframe Modernization がアクセスを提供するツールで実行されます。

自動リファクタリングを使用する場合は、Blu Insights を参照してください。AWSBlu Insights は、シングルサインオンで AWS Management Console から利用できるようになりました。今後は AWS Blu Insights の認証情報を個別に管理する必要はありません。AWS AWS Blu Age Codebase と Transformation Center の両機能には、AWS Management Console から直接アクセスすることができます。

メインフレームから AWS へデータを移行する際、AWS SCT と AWS Database Migration Service を使用することをお勧めします。詳細については、AWS Schema Conversion Tool ユーザーガイドの「What is the AWS Schema Conversion Tool?」および AWS Database Migration Serviceユーザーガイドの「What is AWS Database Migration Service?」を参照してください。

運用と最適化フェーズ

運用と最適化のフェーズでは、デプロイしたアプリケーションの監視、リソースの管理、セキュリティとコンプライアンスを最新の状態に保つことに重点を置きます。また、移行したワークロードを最適化する機会を評価します。

概念

AWS Mainframe Modernization には、 でメインフレームワークロードを移行、モダナイズ、実行するためのツールとリソースが用意されています AWS。

トピック

アプリケーション

AWS Mainframe Modernization で実行中のメインフレームワークロード。一連のバッチジョブ、インタラクティブトランザクション (CICS または IMS)、またはその他のコンポーネントによりアプリケーションが構成されます。スコープはユーザーが定義します。CICS トランザクションやバッチジョブなど、ワークロードに必要なコンポーネントやリソースを定義および指定する必要があります。

アプリケーション定義

AWS Mainframe Modernization で実行されているアプリケーション (メインフレームワークロード) に必要なコンポーネントとリソースの定義または仕様。定義をアプリケーション自体から分離するこ

とは重要です。なぜなら、同じ定義を異なるランタイム環境の複数の段階 (製造前、実稼働) で再利用できるからです。

バッチジョブ

ユーザーによる操作を必要とせずに実行するように設定されたスケジュールプログラムです。AWS Mainframe Modernization では、バッチジョブ JCL ファイルとバッチジョブバイナリの両方を Amazon S3 バケットに格納し、両方の場所をアプリケーション定義ファイルに記載する必要があります。バッチジョブを実行すると、 AWS Mainframe Modernization は次のステータス値をレポートします。

送信中

バッチジョブの送信処理中です。

保持

バッチジョブは保持状態です。

ディスパッチ中

バッチジョブのディスパッチ処理中です。

実行中

バッチジョブは現在実行中です。

[キャンセル中]

バッチジョブはキャンセル処理中です。

キャンセル

バッチジョブはキャンセルされました。

成功

バッチジョブの実行は正常に終了しました。

[失敗]

バッチジョブが失敗しました。

警告ありで成功

バッチジョブは正常に終了しましたが、軽微なエラーが報告されました。 GetBatchJobExecution レスポンスの一部として返されるジョブ条件コードは、エラーの原因を示します。

構成

環境またはアプリケーションの特性です。環境設定には、エンジンタイプ、エンジンバージョン、可用性パターン、オプションのファイルシステム設定などが含まれます。

アプリケーション設定は静的または動的にすることができます。静的設定は、新しいバージョンをデプロイしてアプリケーションを更新した場合にのみ変更されます。動的設定は、通常はトレーシングのオン/オフなどの運用上のアクティビティですが、更新するとすぐに変更されます。

データセット

アプリケーションが使用するデータを含むファイルです。

環境

AWS コンピューティングリソース、ランタイムエンジン、および 1 つ以上のアプリケーションをホストするために作成された設定の詳細の名前付きの組み合わせ。

Mainframe Modernization

レガシーメインフレーム環境から にアプリケーションを移行するプロセス AWS。

移行ジャーニー

レガシーアプリケーションの移行とモダナイズ end-to-end のプロセスで、通常は評価、準備、移行とモダナイズ、運用と最適化のフェーズで構成されています。

マウントポイント

システム内に保存されているファイルへのアクセスを提供するファイルシステム内のディレクトリです。

自動リファクタリング

レガシーアプリケーションのアーティファクトを最新のクラウド環境で実行できるようにモダナイズするプロセスです。これにはコードやデータ変換が含まれる場合があります。詳細については、「AWS Mainframe Modernization Automated Refactor」を参照してください。

リプラットフォーム

アプリケーションとアプリケーションアーティファクトを、あるコンピューティングプラットフォームから別のコンピューティングプラットフォームに移動するプロセスです。詳細については、「AWS Mainframe Modernization Replatform」を参照してください。

リソース

コンピュータシステム内の物理コンポーネントまたは仮想コンポーネントです。

ランタイムエンジン

アプリケーションの実行を容易にするソフトウェアです。

AWS Blu Age を使用したアプリケーションの自動リファクタリング

AWS Blu Age による自動リファクタリングは、メインフレームアプリケーションの移行とモダナイズのための end-to-end ソリューションを提供します。リファクタリングプロセスのステップは、次のとおりです。

- インベントリを分析する
- 依存関係を分析する
- コードを自動的に変換する
- テストシナリオのキャプチャと管理

AWS Mainframe Modernization コンソールからシングルサインオンで利用できる Blu Insights ツールで、前のステップを完了できます。Blu Insights について詳しくは、「Blu Insights のドキュメント」を参照してください。

変換されたソースコードに問題がなければ、 に移動し AWS、次のステップを実行します。

- リファクタリングしたアプリケーションをビルドしてデプロイします。
- AWS Mainframe Modernization でアプリケーションをデプロイしてモニタリングします。

AWS Blu Age ランタイム (非マネージド) は、 Mainframe Modernization サービスの提供の 1 AWS つであり、 AWS Blu Age マネージドサービスでもあります。 AWS Blu Age マネージドを使用すると、モダナイズされたアプリケーションを AWSマネージド環境にデプロイしてエクスペリエンスを簡素化できるため、モダナイズされたアプリケーションを実行する基盤となるインフラストラクチャを管理する必要はありません。対照的に、 AWS Blu Age ランタイム (非マネージド) では、モダナイズされたアプリケーションを自分の AWS アカウントにデプロイできるため、独自のインフラストラクチャを管理できます。 AWS Blu Age ランタイム (非マネージド) を使用すると、モダナイズされたアプリケーションを必要な方法で実行するために必要なすべての技術コンポーネントを柔軟に運用できます。

AWS Blu Age ランタイム (非マネージド) は、Amazon EC2 および の Amazon ECS でデプロイできます AWS Fargate。

トピック

- AWS Blu Age リリースノート
- AWS Blu Age ランタイムの概念
- AWS Blu Age ランタイム設定および設定ファイル
- AWS Blu Age ランタイム APIs
- AWS Blu Age ランタイム (非マネージド) セットアップ
- Blu Age デベロッパー IDE でソースコードを修正する

AWS Blu Age リリースノート

このセクションでは、バージョン 3.5.0 以降の AWS Blu Age ランタイムおよびモダナイゼーションツールのリリースノートを、バージョン番号別にまとめています。

> **Note**
>
> このドキュメントより前のリリースノートについては、 AWS Blu Age 配信サービスにお問い合わせください。Blu Insights の最新機能については、Blu Insights のリリースに関するページを参照してください。

トピック

- リリースノート 3.10.0
- ランタイムリリース 3.10.0
- モダナイゼーションツールリリース 3.10.0
- リリースノート 3.9.0
- ランタイムリリース 3.9.0
- モダナイゼーションツールリリース 3.9.0
- リリースノート 3.8.0
- ランタイムリリース 3.8.0
- モダナイゼーションツールリリース 3.8.0
- リリースノート 3.7.0
- ランタイムリリース 3.7.0
- モダナイゼーションツールリリース 3.7.0

- リリースノート 3.6.0
- ランタイムリリース 3.6.0
- モダナイゼーションツールリリース 3.6.0
- リリースノート 3.5.0
- ランタイムリリース 3.5.0
- モダナイゼーションツールリリース 3.5.0

リリースノート 3.10.0

AWS Blu Age ランタイムとモダナイゼーションツールのこのリリースでは、すべての変換および実行ステップでパフォーマンスと堅牢性を向上させるために、製品全体のコアベースラインのアップグレードと改善に焦点を当てています。このリリースの主な機能と変更点は次のとおりです。

- Java 8 から Java 17 へのバージョンアップグレードにより、セキュリティとパフォーマンスが向上し、お客様はより最新の言語で実装されたアプリケーションをデプロイして実行し、最新のサードパーティーフレームワークバージョンを使用できます。
- ユーザーまたはジョブ間の大きな共有メモリスペースを管理し、アプリケーションまたはインスタンスの再起動後に再利用可能なデータを保存するための追加サポート。
- Blusam の大規模なデータセットへの高速アクセスには、レコードのサブセットを段階的に取得できるページ分割メカニズムを使用します。

このリリースに含まれる変更の詳細については、以下のセクションを参照してください。

ランタイムリリース 3.10.0

このランタイムは Java17, Spring2.7、Angular16 に基づいています。

zOS

新しい特徴

- Blusam - ページを使用してインデックスを保存およびロードするページ分割されたメカニズムによる大規模なデータセットのサポートを追加

改良点

- 文字列から数値への優先順位の低い変換を処理するように DataUtils.compare を強化
- YML プロパティ dataSimplifier .byteRangeBoundsCheck を使用して、不適切な値で ByteRange が作成されていないことを確認するサポートを追加しました。
- 空の文字 GraphicAlphanumericType を持つ の初期化をサポートするように removeSOSI () を強化
- ジョブオペレーションの堅牢性と安全な GDG 状態の読み取りを追加
- Blusam - CoreBluesamManager.removeCache () という名前の新しいメソッドを使用して Blusam データセットの Ehcache をクリアするサポートを追加しました
- Blusam - 通常の Blusam データセットの削除/名前変更の動作が改善されました
- Redis - データセットのロック解除とレコードロックのクリアのサポートの強化
- JICS - 失敗したリクエストのエラーメッセージを改善しました
- JCL - ドット文字に基づく ControlM 変数連結のサポートを追加
- JCL - GDG ファイルの書き込み ADVANCING (ADV) のサポートを追加
- JCL - すべての GDG ファイルを削除した後の現行世代番号のサポート強化
- JCL - データセット作成時のカタログからの rdw/recordSize 読み取りのサポートの強化
- JCL - データ出力レコードのサイズでファイルを開くときに、リソースオブジェクトを (から AbstractSequentialFile) 更新するサポートを追加
- JCL - IDCAMS のパフォーマンスが向上しました
- JCL - 「CHARACTER」のエイリアスとして「CHAR」を追加することで、PRINT STATEMENT のサポートを強化
- SORT - Blusam 固定長データセットから可変長のデータセットへのコピーオペレーションのサポートを強化
- SORT - 一部の特定のステートメントを処理するためのソート文法を強化しました

AS400

新しい特徴

- ユーザースペースとその関連 APIsのサポートを追加
- SNDPGMMSG の TOMSGQ パラメータと実装されたメッセージキューのサポートを追加
- CL - OVRPRTF コマンドの FILE パラメータと SPLFNAME パラメータのサポートを追加
- CL - CPYF コマンドを使用した対応するパーティションテーブルのライブラリ処理のサポートを追加

- CL - CHGCURLIB コマンドを処理し、クエリを構築するときに現在のライブラリを考慮するためのサポートを追加しました
- CL - 呼び出しスタックトレースの一部として cl コマンドを処理するためのサポートが追加されました

改良点

- コールスタックトレースエントリの処理 MessageHandlingBuilder を改善
- contextPreconstruct 機能の並列実行が改善されました
- TAKTAKZ によってレコードが作成されたときの表示属性が改善されました
- SAVOBJ が改善され、複数の出力ファイルを処理できるようになりました。
- Java プログラムから呼び出された programCallStack ときに に追加することで groovy プログラムの処理を改善しました
- ヘルプモーダルの最上位測位検出を改善
- SNDPGMMSG に toMsgQ パラメータが提供されるときの toPgmQ toMsgQ 機能が改善されました
- メッセージローダーの定義済みメッセージと機能のフェッチが改善されました
- コンテンツ内の区切り文字の CPYTOIMPF 処理が改善されました
- READ レコードのリリースロックの改善

横断的な機能

新しい特徴

- フロントエンドのシステムメッセージの翻訳を追加
- プログラム呼び出しスタック ExecutionContext を返す新しいメソッドが に追加されました
- 実際の環境に関係なく、行区切り文字 (データ区切り記号用) を設定する
- SQL モデルの JSON パスを設定する可能性を追加

改良点

- パディングが含まれる場合の比較方法 DataUtils.compareAlphInt() を改善しました。
- カーソルクエリで例外発生時のカスタム動作を許可するフラグの作成
- グラフィックの LOWVALUES db 変換を改善

サードパーティー

- CVE-2024-21634, CVE-2023-34055, CVE-2023-34462, IN1-JAVA-ORGSPRINGFRAMEWORKSECURITY-5905484, CVE-2023-46120, CVE-2023-6481, CVE-2023-6378, CVE-2023-5072を軽減するためにアップグレード

モダナイゼーションツールリリース 3.10.0

zOS

改良点

- COBOL - ABS 関数のサポートを追加
- JCL - 拡張変数スコープ: JOB の代わりに TAK にアタッチ
- 低/高値の拡張カーソルパラメータインジェクション
- CSD 解析が改善されました。特にリモートトランザクションの場合

AS400

改良点

- コントロールレベルインジケータの空白のチェックを削除しました
- IMPORT/EXPORT キーワードの外部名のサポートを追加
- フィールドでの %LEN のサポートを追加
- CL - CLLE 言語の新しい演算子のサポートを追加
- CL - ネストされた IF のサポートを追加
- COBOL - 複数のキーで使用する場合の START コマンドの処理が改善されました
- TAKF - レコード番号によるカーソル位置の処理が改善されました
- TAKF - 符号付き数値、数値のみのフィールド、および大規模のフィールドのフォーマットを改善
- TAKF - スクリーン全般ヘルプのタイトルの決定が改善されました
- TAKF - 入力/出力仕様のサポート向上
- TAKF - 数値フィールドの検証中のグループ化区切り文字の処理が改善されました
- マッピング出力/DDS レコードの改善
- プリンターファイルの REFFLT キーワードが参照フィールドを解決する機能を改善

- RPG - 「ALL free」ステートメントのサポート強化
- RPG - 条件解析が改善され、結果 TAG なしで CABXX を処理するためのサポートが追加されました
- RPG - 数値フィールドの入力仕様の処理が改善されました
- RPG - IF/ELSEIF/WHEN 条件内でのプロシージャ呼び出しの処理が改善されました
- RPG - dspf ファイルで呼び出されたときの READ コマンドの処理が改善されました
- RPG - 存在しない DDS を参照するファイルのサポートを改善
- 物理レコード形式名を渡したときの REFFLD の処理を改善
- 「return」を db 列名として使用するサポートを追加

横断的な機能

新しい特徴

- Oracle - SYS よりもユーザーを定義して組み込み関数を保存できるようになりました

改良点

- Java バージョンを v8 から v17 にアップグレードしました
- クラスター列名による SQL 条件の改善
- ORDER BY 句のビューからのサポートを追加

リリースノート 3.9.0

AWS Blu Age ランタイムおよびモダナイゼーションツールのこのリリースでは、高可用性アーキテクチャでパフォーマンスを向上させるための製品全体の複数のトランスバーサル機能強化と、次のレベルにジョブの実行を引き上げる新機能に焦点を当てています。このリリースの主な機能と変更点は次のとおりです。

- Angular 13 から Angular 16 へのバージョンアップにより、セキュリティが強化され、お客様のオンラインアプリケーションのパフォーマンス向上に役立つ新機能にアクセスできます。
- AS400 にクロスジョブ機能のサポートが追加されます。主な特長として、ジョブ間で同期的に照会メッセージを送信できるため、モダナイズされたジョブにおける疎結合化が可能になります。
- 接続プールの最適化、接続時の高度なセキュリティ、データセットロックメカニズムのアップグレードなど、Redis の使用に関するパフォーマンスが向上しました。

このリリースに含まれる変更の詳細については、以下のセクションを参照してください。

ランタイムリリース 3.9.0

zOS

新しい特徴

- ソートプログラム: VSAM 入力を固定長に更新
- JHDB DB: 設定可能なタイムアウトを追加

改良点

- ファイル連結で使用する場合のストリームへの行区切りのサポートを強化
- 連結されたシーケンシャルファイルを開くためのサポートを強化 ファイルを開いて DataSetIndex から初期化する
- NumericEditedType が数値に影響を受ける場合の仮想小数区切り文字のサポートの強化
- 負の値 NumericEditedType での のサポートの強化
- IDCAMS: SYSIN カードは、 application-utility-pgm.yml で定義された「エンコード」プロパティを使用して読み取られるようになりました
- IDCAMS: DEFINE CLUSTER ステートメントの FILE(..) 引数をサポートするように文法を更新
- INFUTILB: DD SYSREC の DCB パラメータをオーバーライドするための DFSIGDCB 引数のサポートを追加
- INFUTIL:「DFSIGDCB YES」パラメータのサポートを強化
- SPLICE が巨大な入力ファイルを処理するように改良
- DFSORT: コメントフィールドの処理を改善
- DFSORT: (符号付き/符号なし) 自由形式数値フォーマット (SFF/UFF) のサポートを追加
- SORT: OPTION PRINT ステートメントと OPTION ROUTE ステートメントの解析サポートを追加
- SORT/ICEMAN: 囲まれた除算演算 (DIV 演算子を含むフィールド) のサポートを追加
- ジェネリックキーを使用した CICS READ のサポートを強化
- グラフィックタイプから SOSI を削除するための関数 StringUtils.chargraphic fixed
- DataUtils.isDoubleByteEncoding のパフォーマンスを強化する
- JCL: 一時データセットの KEEP 処理モードのサポートを強化。システムは処理を PASS に変更します。

- JCL: DCB パラメータを動的に処理
- JCL: 正しくない値に対する SUM FIELDS 出力を強化
- JCL: CommonDDUtils::getContent がカタログ内の recordSize を検索するようになりました
- JCL: データセットの作成時にカタログから rdw/recordSize 属性を読み取る
- JCL: 同じジョブステップで DD の DCB パラメータを別の DD にコピーするための DCB=.MYDD のサポートを追加
- JCL: レコードサイズ継承システムの改良
- JCL: (Redis) 排他的データセットロックを追加
- Redis: スタンドアロンモード向けの SSL サポートを追加
- Redis: ロックと同期した Redis ロックカウントを追加
- Redis: Redis ロックでサポートされるプールパラメータ
- Redis: Redis によるメタデータの更新を最適化
- Redis: Redis クラスターのサポートを強化
- IO モードでのオープンロックの改善
- データセットのロックパフォーマンスの向上と未使用のロックのクリア
- ファイルの登録解除時のデータセットのパスを強化
- プリフェッチウィンドウキャッシュの無効化の改善
- スレッドセーフユーティリティのデータソースプロバイダの使用に対するサポートを追加
- datasetState の無効性チェックの強化
- 既に開いているデータセットを再度開かないためのサポートを強化
- ジョブの最終オペレーションに対する堅牢性を追加
- 重複を許可するキーのインデックス順序のサポートを強化
- スキップリストのシリアル化順序のサポートを強化
- インデックスの順序に関する問題の診断に役立つデバッグダンプ機能のサポートを追加
- メタデータ更新のサポートの強化
- Blusam 一括読み取りのサポート強化

AS400

新しい特徴

- アプリケーションコンテキストレジストリの作成
- DSPF キーワード CLRL(NO) Support レコードロックのモリタリングをサポート
- キー付き のサポート DataQueue
- バッチジョブの INQUIRY メッセージのサポート
- AS400 COBOL 用のプログラム記述プリンタファイルのサポートを追加
- RMVJOBSCDE cl コマンドを処理
- RUNSQL/DLYJOB の改良
- CHKOBJ: パラメータ LIB のレガシーエラーコードを発生
- SNDPGMMSG: 文字列パラメータのサポート
- RTVDTAARA: LDA の部分文字列の改善
- DSPFD: 特定のファイル名に対する FILE パラメータのサポートを追加
- RUNQRY: QRY PARAM 内の SQL ファイルのサポート
- CRTDUPOB: データ域間でのデータのコピーをサポート
- SBMJOB: 使用する指示を変換します JobQueueManager
- OPNQRYF: Qtemp ライブラリのサポートを追加
- CRTDUPOBJ: パーティションの内容をコピーするロジックを改善
- CRTDUPOBJ: Qtemp のビュー向けサポートを追加
- RTVSYSVAL: CL コマンドでの SYSVAL 値、QDATFMT のサポート
- CHKOBJ: OUTQ のサポートを追加
- RTVJOBA: SWS パラメータのサポート
- SNDPGMMSG および RCVMSG:
  MSGF、MSGFLIB、MSGDTA、MSGTYPE、KEYVAR、MSGKEY、MSGID の追加パラメータのサポート

改良点

- WORKSTATION I/O カードのサポートの強化
- 前のメッセージをオーバーレイする設定済みメッセージの処理を改善
- array-messageline に関する追加のメッセージ情報のサポート
- EVAL、SortA、figuratives 内のスタンドアロン配列ラッパーのアクセスを改善
- オンラインアプリケーション終了時の DAO クリーニングを改善

- 他の日付形式のサポートを追加し、文字列入力の処理を改善
- CL コマンドからシステム値ヘルパークラスデコードおよびビルドパラメータを追加することで、SYSVAL の CVTDAT 処理を改善しました SbmJob
- gapwalk-cl-command コンポーネントスキャンからパッケージ com.netfective.bluage.gapwalk.rt.blu4iv を削除しました
- メッセージキュー API の事前定義済みメッセージのサポートを強化
- 別のプログラムで書き込まれたレコード retrieveSubfileRecord に対する のサポートが改善されました
- メッセージキュー API の即時メッセージのサポートを強化
- ジョブ送信時のローカルデータ域の処理を改善
- サーバーの起動時に JobQueues 自動的に起動
- SBMJOB のパラメータのデコードに applicationContext 設定を使用
- システム提供のエラーメッセージの改善
- ネストされたサブディレクトリ内の .properties ファイルの検索を RTVMSG に許可
- 不正/無効なポインタにバインドされたエンティティのリセットを処理
- msgId と MsgFile name を RCVMSG の文字列として表示 MessageHandlingBuilder するように改善しました
- メッセージキューイング API withMsgFileの名前メソッドの改善
- データ域のロックメカニズムを改善
- RTVMBRD: パラメータ FILE の小文字と大文字をサポート
- CRTDUPOBJ: ビューの処理を改善
- CPYTOSTMF: 接続の処理を改善
- CPYF: フラットファイルからのコピー時のディレクトリ名の処理を改善
- RCVF: Groovy および Java 用の DEV/RCDFMT パラメータと RCDFMT の変換を適切に処理
- RCVF: 後続の呼び出しを処理し、カーソルのリセットを回避
- CPYF: フラットファイルからの書き込みのサポートを追加
- CRTDUPOBJ: Qtemp ライブラリによる新しいオブジェクトの処理を追加
- CHGDTAARA: データ域の最大長を 256 から 2000 に増加
- SAVOBJ: 保存されたレコードが挿入順になっていることを確認
- RTVDTAARA: 取得された値 (トリミングはしない)

- CHKOBJ: メンバーが存在しない場合に正しいモニタリングメッセージを返す
- RTVDTAARA: LDA サブストリングのサポートを追加
- RTVDTAARA: RTNVAR パラメータで指定された変数の長さまで空白を返す
- RTVDTAARA: 開始と長さの整数パラメータをサポートし、最新の変換形式をサポート
- CHGDTAARA: 下限と上限を含むパラメータのサポートを追加
- CHKOBJ: パラメータオブジェクトタイプの VIEW 値を処理
- CHKOBJ: ビューが存在する場合は、メンバーに関係なく結果が true に設定される

横断的な機能

新しい特徴

- .txt ファイルへのレポートの生成を処理
- CurrentSchema XA データソースプロパティをシークレットマネージャーに追加
- 既に開いているカーソルを開くときにフレームワークが SQLCODE エラー 502 を発生させるように、database.cursor.raise.already.opened.error yml プロパティを追加

改良点

- Amazon EC2 パッケージの AWS Blu Age に gapwalk poms を追加
- 新しいシグナルハンドラーパラダイムをデフォルトで使用します。
- 処理が MOD または OLD の場合のロックのサポートを追加
- データベースの日付/時刻パターンを保存するキャッシュを追加
- のチェック機能の改善 PackedType
- でレコードの DataUtils.setTo 関数を改善する VariableSizeArray
- MQ SYNCPOINT オプションを実行ユニットと見なして処理
- ロールバックトランザクションで SQLCODE を設定するフレームワークを有効化
- エンジンキーシークレットに従った自動ドライバークラス名を追加
- プログラム/トランザクションタイムアウト
- カーソルへのアクセス時、ロールバックの後にカーソル位置を復元

サードパーティー

- SnakeYAML 、Redisson、Amazon SDK のアップグレード、削除 YamlBeans (CVE-2022-25857, CVE-2023-24621, CVE-2023-42809, CVE-2023-44487を緩和)

モダナイゼーションツールリリース 3.9.0

zOS

改良点

- 文字列タイプのターゲットのソースとしての XML-TEXT のサポートを強化
- X/(Y/Z) 除算パターンをサポートするよう、STM から UML へのワークフローを強化
- JHDB DB: データベースの更新前に ROLLBACK 呼び出しを受け入れる
- JHDB DB: トランザクションが終了しても ROLLBACK を受け入れる (NOP)
- JCL: ステップ検証関数の強化
- SORT: ゾーン 10 進数の負の値を持つ SUM 関数を処理
- COBOL: 文字列リテラルにおける一重/二重引用符のエスケープのサポートを追加

AS400

改良点

- 先行ゼロを追加することで、組み込み関数 %editc の編集コード X の処理を改善
- 入力のみのフィールドの初期値の処理を改善
- ヘルプダイアログにアクションキーを追加
- 動的テーブルのフッターレコードを下部に表示
- 実際の RECORD-KEY を指定するファイルの START コマンドを KEY PHASE なしで処理
- 浮動小数点型と NumberUtils::pow 型のデフォルト値を追加
- LIKE(IN) を使用した変数定義のサポートを追加
- オプション要素の省略をサポートするように、FOR ループ処理を更新
- レコードを CTDATA 配列名に関連付けるように RPG 解析を更新
- CABxx ステートメントのインジケータの処理を改善
- COMMIT キーワードのオプションパラメータをサポート
- LF での FORMAT キーワードサポートの強化

- 「高い」と「等しい」(または「低い」と「等しい」) インジケータを持つ LOOKUP オペレーションコードを管理
- 二重引用符内で宣言された PF キー名を処理
- 先行ゼロが非表示にならないように、EDTCDE X の処理を改善
- 名前のないラベルが生成されないように、プリンタファイル内の MSGCON のサポートを強化
- フィールド CONTENT が複数のデータ構造で共有される
- ERRSFL パラメータを SFLMSG/SFLMSGID と組み合わせて処理
- 完全フリー RPG の proc 宣言前スコープのメインコードを改善
- 解析の条件付き制御仕様を追加
- dataholdermapper での setErrSfl() メソッドのサポートが改善されました
- 内部で作成された変数の型解決を改善
- Z-ADD opcode のサポートを強化
- DFT 値を含む定数フィールドの処理を改善
- プログラムステータス ds 内の整数フィールドのサポートを強化
- ENTRY パラメータのインジケータ割り当てを処理
- ref/reffield キーワードで伝播されるキーワードのフィルターを改善
- サポートされている名前のない DataArea データ構造
- ポインタデータ型の処理を改善
- LIKE キーワードを持つ変数の定義に使用される配列要素を処理し、出力フィールドでの配列アクセスをサポート
- 数字のみが表示されるように、符号付き数値のサポートを強化
- O カードでの論理関係をサポート
- 英数字の %CHAR のテストケース
- 制御仕様キーワード main をサポート
- プリンタファイルに 2 つのパラメータを含む EDTCDE
- FullFreeRPG 解析の改善
- フッターが正しく配置されるように、動的テーブルを強化
- ALL 表意定数を持つ数値型の初期化のサポートを追加
- 同じ物理ファイルを参照する複数の RPG 論理ファイルの処理を改善
- 最新の画面で変更されたフィールドの検出を改善
- 動的フィールドとのモーダル同期

- 出力のみの符号付き数値フィールドの処理を改善
- WORKSTATION I/O カードのサポートの強化

横断的な機能

新しい特徴

- データ移行ツール: バイトを読み取るときに ebcdicFilesWithVarcharInVARCHAR 2 バイトの長さを考慮に入れるように VB プロパティを追加
- エラーを口グ記録する共通 API を実装
- COBOL2Model BluAgeErrorDictionaryUtils 、RPG 、CycleBuilder Definitions2Model、および のエラーや情報を口グに記録するための一般的な API の実装と使用 COBOL2Model FieldsProcessor
- さまざまな isolation-clause の定義がサポートされるように、SQL 文法を改善

改良点

- Angular バージョンを v16 にアップグレード
- Angular: ajv バージョンを 6 から 8.9 にアップグレード

サードパーティー

- Groovy をバージョン 2.4.15 にアップグレード

リリースノート 3.8.0

AWS Blu Age ランタイムおよびモダナイゼーションツールのこのリリースでは、製品全体の複数のトランスバーサル機能強化に重点を置いて、品質とセキュリティを向上させ、キャッシュのパフォーマンスを向上させ、1 つのディストリビューションで がサポートするコマンドを統合しました。このリリースの主な機能と変更点は次のとおりです。

- Spring 2.5 から Spring 2.7 へのバージョンアップにより、プラットフォームのメンテナンスサポート、パフォーマンス、セキュリティが向上しました。
- 以前は CL スクリプトを使用していたモダナイズされたアプリケーションの使用とデプロイを容易にするため、82 を超える CL コマンドの統一は ディス over-the-counterトリビューションの一部として をサポートします。

- マネージドサービスへの統合インポート、データセットのメタデータ情報をリストアップする機能など、BluSAM データセットのオペレーションと対話を改善するための新しい API が利用できるようになりました。
- クラスターモードでの可用性、高可用性データの取得、シークレットの使用の標準化など、パフォーマンスの向上と Redis の使用の拡張を実現しました。

このリリースに含まれる変更の詳細については、以下のセクションを参照してください。

ランタイムリリース 3.8.0

zOS

新しい特徴

- の文字列としてのキー定義の処理 DynamicFileBuilder
- DFSORT: OUTFIL TRAILER1 + DFSORT 文法の初期化における複数項目のサポートが追加
- CommonDDUtils ツール: インストリームデータのレコードサイズの処理
- インデックスファイル: GENKEY オプションの処理

改良点

- 外部化された BluSAM ローディングサービスを別の jar に格納
- 一時ファイルを保存する場所の設定へのサポートの追加
- 複数ノードの場合の共有キャッシュメカニズムの改良
- 共有キャッシュの使用: IDCAMS 検証の最適化
- 埋め込み選択の ROWID インジェクションの改良
- JCL: 各インストリームジョブプロシージャを個別の Groovy ファイルで生成
- IDCAMS JCL カードの対象範囲を card-demo-v2 つ確保する
- BluSAM: 複数のインスタンスを使用する場合のウォームアップの重複を回避
- キャッシュハイドレーション時のメモリ使用量の削減
- Jedis プール設定のサポート
- ファイル連結で使用する場合、ストリームに行区切りを追加
- IDCAMS ユーティリティで EBCDIC カード + ブロックコメント (/* ... /) をサポート
- データベースサポートクエリ: SQL への level49 の変換で 2 バイト文字列をサポート

- DFSORT 文法: 17 の制御ステートメントを実装 + そのうちの 2 つの制御ステートメントを統合 (OMIT/INCLUDE)
- GRAPHIC 列のフェッチ INFUTILB の改良
- 可変サイズのテーブルを持つファイルの読み込みをサポート
- 最後のバイトの最初のビットが「E」である二ブル署名 ZonedType 付き での のサポート
- DFSORT/ICETOOL でレコードが CHANGE 検索定数のいずれも一致しない場合、NOMATCH=(..) 引数のサポートを追加
- Redis クラスターの互換性
- groovy 終了コードに基づくジョブステータス (失敗) の処理
- CICS SYNCPOINT ROLLBACK サポート向上
- Redis キャッシュの使用の最適化のためのプリフェッチウィンドウ
- JCL/GROOVY: DISP=(,PASS) の場合、前のステップのデータセットから isRDW プロパティを継承
- 可変サイズの配列でデータの部分コピー処理

AS400

新しい特徴

- ディスプレイファイル用の I/O カードのサポート
- DSPF キーワード ERRMSGID と CHKMSGID に関する追加のメッセージ情報のサポート
- フロントエンド画面での複数のエラーメッセージのサポート
- アプリケーション内での gapwalk-cl-command 82 CL コマンドのサポートを追加または改善

改良点

- コミットメントコントロール下の DELETE と READ のサポート向上
- ConvertDate 組み込み %dec の内部
- XSS セキュリティヘッダーの適用
- STM 生成の堅牢性と一貫性の向上 (自由形式 rpg の継続行、小数部のカンマ、定義/宣言内の自由形式ブロックの処理の向上)
- DataHolderMapper 生成の向上
- での堅牢性と変更範囲の追加 DataAreaFactory

- タブキーでのフォーカスシフトの改良
- Jasper レポート生成のパフォーマンス向上
- パディング 0 による小数点表示の改良
- INFDS の ROW/COL フィールドのサポート向上
- 画面からの変更項目のサポート向上
- 生成されたレポート名とパスのゲッターの追加
- Dataqueue の長さの改良
- Spring Boot 2.7 の新標準に合わせたジョブキューの自動設定の改良
- 複数の同時セッションに対するワークステーションの更新の改良

横断的な機能

新しい特徴

- Packed の無効データなしの許容範囲のサポート
- データセットのエンドポイントを一覧表示するページ分割/フィルタリングを追加

改良点

- 空文字列と列比較における ORACLE クエリ変換手法が強化されました。
- DSNTEP および INFUTILB ユーティリティプログラムによる BLOB DB2 の処理。BLOB DB2 は、BYTEA タイプの postgres にモダナイズされました。
- カーソルの最後のアイテムの削除の改良
- RRDS ファイルの削除のサポートの強化
- AWS Blusam シークレットのパフォーマンスが向上しました
- SQL フレームワークでのデータベース接続の処理の改良
- 標準化された AWS マルチデータソースシークレットマネージャーキー
- パフォーマンスリグレッションの修正
- のチェック機能の改善 PackedType
- の LOW-VALUE の処理を改善しました PackedType
- cognito 接続用の Spring セキュリティパッケージのアップグレード
- DB2 対象のデータベースで codeshiftpoint のエンコードとデコードを適用しない

サードパーティー

- Spring Boot 2.5 から 2.7 へのアップグレード

モダナイゼーションツールリリース 3.8.0

zOS

新しい特徴

- JCL: キャリッジリターン「\r」を含むストリームの処理

改良点

- ON SIZE ERROR を持つ DIVIDE をモダナイズする場合に、0 で除算されないようにロギングを改良
- JCL: プロシージャ内のプロシージャ呼び出しのサポート向上
- あいまいなフィールドがある場合の FORMATTIME CICS コマンドの OF キーワードのサポート
- JCL: 変数内の「Â¥」文字のサポート
- JCL: 前のステップに基づいて RC を計算
- PL1 SUBSTR の使用時に文字列の代わりにバイトを比較する
- 単一ソースからの多次元配列の初期化の改良
- IF ブロックに 1 つの SQL クエリが含まれる場合の COBOL の解析を改良

AS400

新しい特徴

- CL でのネストされた IF ステートメントのサポート
- RPG フリーフォームでの ENDDO ステートメントのサポート向上

改良点

- コントロールレベルのコンディショニングのサポート向上
- LIKE によるプロトタイプの戻り値の改良
- 関数 %months、%year、%days の処理のサポート向上

- 画面全体のヘルプ機能のサポート
- パラメータとして渡された形象的な BLANKS の処理
- "" 演算子を使った式 EVAL の改良
- KEY PHASE なしの START コマンドの処理
- LIKEREC キーワードの処理の改良
- 名前のないサブフィールドの改良
- 符号なしのタイプ返すプロシージャの改良
- RESET オペレーション (フリー RPG)、%CHAR、%DEC ビルトインのサポート向上
- ビルトイン関数 %LOOKUPXX の改良
- プロトタイプなしのプロシージャでの LIKEDS キーワードのサポート向上
- Dim キーワード配列タイプ (VAR、AUTO) の処理
- XFOOT のサポート向上
- COBOL: RENAMES フィールドのサポート向上
- CL: while (true) 条件をサポート
- LIKE キーワードを使ったスタンドアロン配列の処理の改良
- ビルトイン関数 %INT の改良
- RPG Full Free 解析の改良
- リンケージでの配列のサポート向上
- CL2GROOVY: 選択ステートメントのサポート
- DSPF キーワード「ERRMSGID」の改良
- 先頭にゼロが付いたバイトの初期化の処理の改良
- 数値フィールドの authorizedValues の改良
- フリーフォーム EVAL ステートメントの拡張 H の処理
- CL から Groovy へ:LDA のサブ文字列のサポート
- レコードの RESET のサポート向上
- リファレンス付きの EDTCDE と EDTWRD の処理の改良
- DDS フィールド付きの入力フィールドマッピングの改良
- MOVEA 文字を IN 配列に変換するためのサポート向上
- LIKEDS キーワードによるプロトタイプの改良
- DSPF キーワード DSPATR のサポート向上

- +/-を使用した D カードの解析の改良
- プログラム呼び出しに堅牢性を追加
- フィールド解決プロセスの堅牢性の向上

横断的な機能

改良点

- FrontEnd: IME 入力の貼り付けイベントをシミュレートする

サードパーティー

- Spring Boot 2.5 から 2.7 へのアップグレード

リリースノート 3.7.0

AWS Blu Age ランタイムおよびモダナイゼーションツールのこのリリースには、主に、コマンドとユーティリティのサポート向上、AWS Secrets Manager との統合機能、および新しいモニタリング機能が含まれています。今回のリリースの主な変更点は次のとおりです。

- 複数のランタイムコンポーネントが AWS Secrets Manager を使用して、ユーティリティデータソース、TS キューの Redis、 BluSam キャッシュ、ロックに主に関連し、モダナイズされたアプリケーションのセキュリティ設定を強化できるようになりました。
- ステータス、期間、ボリュームなど、リソース使用量の最適化と運用管理を目的としたトランザクション、バッチ、JVM のメトリクスを取得できるモニタリングエンドポイント。
- RPG の IBM MQ コールをサポートする新機能に加え、JCL SORT と IDCAMS の変換範囲が拡大しました。

このリリースに含まれる変更の詳細については、以下のセクションを参照してください。

ランタイムリリース 3.7.0

トピック

zOS

新しい特徴

- SQL に似た文法を使用して、プログラムユーティリティアプリケーションに関連するクエリの解析を改良します。(V7-9401)
- オフセット時にインデックス付き可変サイズ配列を処理 (V7-9904)
- 24:00:00 時間形式で DB2 への SQL TIME 列の INSERT をサポート (V7-10023)
- FOR ROWS と ATOMIC オプションで配列からの INSERT SQL クエリをサポート (V7-10105)
- JCL SORT - IFTHEN による OUTREC TranscodeTool のサポートを強化 (V7-10124)
- JCL SORT - OUTREC コマンドに DATE キーワードのサポートを追加 (V7-10125)
- JCL - インストリームプロシージャのサポートを追加 (V7-10223)

改良点

- 「PASS」処理が付いたデータセットは、すべてのジョブステップで利用可能 (V7-9504)
- JCL 属性の SCHENV のサポート (V7-9570)
- CTLCHAR オプションによる SEND のサポート (V7-9714)
- COBOL - ACCEPT ステートメントで異なる行区切り文字セットの処理 (V7-9875)
- 複数のロールバックを回避 (V7-9958)
- GDG ファイルの末尾に追加する MOD 処理の使用を許可 (V7-10031)
- 最適化: putAll リファクタリングの追加 (V7-10063)
- PutAll リファクタリング: ページ分割の追加 (V7-10063)
- Jedis クライアントの読み取りタイムアウトが設定可能 (V7-10063)
- UseSsl スタンドアロンモードの サポート (V7-10114)
- ファイルを正常に開いた後で EIBDS をサポート (V7-10147)
- ファイルコントロールリクエスト後に EIBDS をサポート (V7-10147)
- CICS SYNCPOINT のサポート向上 (V7-10187)
- BluesamRedisSerializer: metadataPersistence の問題 (V7-10202)
- TS キューの Redis AWS Secrets Manager のサポート (V7-10204)
- DD 名のサイズのカスタマイズに関する JCLBCICS のサポート (V7-10224)
- IDCAMS DELETE ステートメントに絶対パスのサポートを追加 (V7-10308)

AS400

新しい特徴

- AS400 スクリーンのヘルプ機能の実装 (V7-9673)

改良点

- INFDS 内のレコード数 (V7-9377)

横断的な機能

新しい特徴

- Amazon に ログを送信するための EC2 でのランタイムのサポート CloudWatch (D87990246)
- バッチ、トランザクション、JVM に関するメトリックスを取得するための新しいエンドポイントを追加 (D88393832)

改良点

- ユーティリティ pgm 用の AWS Secrets Manager の datasources のサポート (V7-9570)
- DSNUTILB DISCARD の Db2 のサポートを追加 (V7-9798)
- デフォルトの SYSPRINT ファイルと SYSPUNCH ファイルのデフォルトのシステム出力ストリームの代わりに ログーへの書き込みをサポート (V7-10098)
- Redis BluSam キャッシュをサポートし、AWS Secrets Manager で接続プロパティを ロックする (V7-10238)
- Db2 XA AWS シークレットでの SSL 接続をサポート (V7-10258)
- IDCAMS REPRO および VERIFY のメタデータの更新 (V7-10281)
- IDCAMS Abend Return Code Management の改良 (V7-10307)

モダナイゼーションツールリリース 3.7.0

トピック

- 横断的な機能

zOS

新しい特徴

- PLI - 配列の断面と 2 次元配列の割り当ての改善 (V7-9830)

AS400

新しい特徴

- コントロールレベルインジケータの処理 (V7-9227)
- EXTNAME パラメータ *INPUT のサポート (V7-9897)
- Goto リライト機能の強化: SELECT OTHER ステートメントにあるタグのサポート (V7-9973)
- REFSHIT DSPF キーワードのサポート (V7-10049)

改良点

- ファイル記述キーワード EXTIND(*INUx) の処理を改良 (V7-7404)
- SQLDDS ファイル変換の改良 (V7-7687)
- AS400 ファイルのファイルオブジェクトは生成されなくなりました (V7-9062)
- ファイル記述キーワード EXTDESC の処理を改良 (V7-9268)
- %CHAR ビルトインの処理を改良 (V7-9311)
- SFLEND を使用しない最後のレコードのページダウンのサポート向上 (V7-9322)
- プレフィックス付きデータ構造のサポート向上 (V7-9436)
- %SIZE で定義されたディメンションのサポート (V7-9472)
- 二重引用符で宣言された PF フィールド名の処理のサポート (V7-9557)
- ファイルオペレーションの改良 - 大文字と小文字の区別なし (V7-9785)
- *USER に初期化されたフィールドのサポート (V7-9806)
- AS400 の COMP タイプのサポート (V7-9840)
- (Not)(InvalidKey V7-9922) での COBOL400 解析が改善されました
- SCAN オペレーションの処理の改良 (V7-9971)

- GOTO オペレーションコードのサポート向上 (V7-9973)
- EXCEPT オペレーションの処理を改良 (V7-9977)
- プレフィックスのサポート向上 (V7-10000)
- RPG での MQ コールのサポート (V7-10007)
- %LOOKUP ビルトイン (キー付き配列データ構造) の改良 (V7-10022)
- Close *All オペレーションのサポート (V7-10036)
- UPDATE AS ROW CHANGE SQLDDS ステートメントのサポート (V7-10051)
- Long 型リテラル値の処理の改良 (V7-10073)
- RPG 文法の改良 (サブルーチンの名前としてキーワード INZ を使用) (V7-10074)
- RPG 文法を改善し、小数部が空の数値をサポート (V7-10077)
- CL と外部ファイル間で共有されるフィールドのサポート向上 (V7-10081)
- DDS 条件付きインジケータのサポート向上 (V7-10084)
- COBOL プログラムによる DDS バイナリタイプのサポート (V7-10100)
- リンケージによる名前のコリジョンを改良 (V7-10109)
- メインプロシージャとエクスポートプロシージャの混合のサポート (V7-10112)
- サブプロシージャ DataStructure での のサポート向上 (V7-10113)
- CLEAR のサポート向上 (V7-10126)
- DO ループのサポート向上 (V7-10134)
- Full-Free RPG で SQLTYPE をサポート (V7-10151)
- DDS キーワードの条件の解析を改良 (V7-10155)
- DSL 生成の改良 (V7-10163)
- 条件がバイナリ表現の場合の processIndicators の改良。(V7-10164)
- Else 条件による GOTO の改良 (V7-10168)
- DSPF での Time と Timestamp のサポート (V7-10173)
- DDS の継続行の解析の改良 (V7-10183)
- COBOL による RENAMES FLD OF RECORD のサポート (V7-10195)
- DSPF フィールドでの条件付きインジケータの解析を改良 (V7-10221)
- DDS キーワード NOALTSEQ の解析のサポート (V7-10288)
- ヘルプメニューと隠しフィールドサポート (V7-10314)
- DSPF ヘルプキーワードのサニティチェックの改良 (V7-10328)

- Ref フィールドのすべてのキーワードが反映されなくなりました (V7-10347)

横断的な機能

新しい特徴

- Data Migrator - CLOB データの処理 (V7-9665)

改良点

- による JCL プロパティ SCHENV の JOB から PROC GROOVY 定義 JobContextへの伝播 (V7-10225)
- FrontEnd - 境界線がない場合のウィンドウサイズの調整 (V7-10358)

リリースノート 3.6.0

AWS Blu Age ランタイムとモダナイゼーションツールのこのリリースでは、zOS と AS400 のレガシー移行の両方に新機能が提供されます。主に CICS サポートメカニズムの拡張、JCL 機能の補完、同時および大量の機能のパフォーマンスの最適化、 multi-data-source 機能の追加を目的としています。今回のリリースの主な変更点は次のとおりです。

- JCL の動的ファイル処理の強化、現在のステートメントの拡張と連結されたデータセットの管理、単一ブロックでの複数のステートメントの実行、バッチからプログラムへのデータ転送。
- 複数の CICS リソースタイプに対する問い合わせを含む、複数の CICS コマンドのサポートが強化されました。
- Blu Age ランタイムユーティリティを使用する際に異なるデータベースを使用できるため、ビジネスデータが複数のソースに分散しているシナリオに最適です。

このリリースに含まれる変更の詳細については、以下のセクションを参照してください。

ランタイムリリース 3.6.0

トピック

zOS

新しい特徴

- JCL DynamicFileBuilder - 拡張ファイル処理管理 (V7-9408)
- INFUTILB UNLOAD ユーティリティの呼び出し時に、一部のビルトイン SQL DB2 関数形式の変換が拡張 (V7-9554)
- PLI 多次元配列の割り当ての強化 (V7-9592)
- ファイルへの sysout リダイレクトの処理 (V7-9992)

改良点

- DB2 RDBMS のストアドプロシージャのトリガー機能を追加 (V7-9155)
- SORT によって PDF 形式への変換を処理 (V7-9286)
- JCL/GROOVY - DUMMY データセットをサポートするように REPRO ステートメントを拡張 (V7-9424)
- CICS UNLOCK のサポート向上 (V7-9606)
- Union のデフォルト値サイズの処理 (V7-9648)
- JCL/GROOVY によって連結されたデータセット内の異なる終了/処理を処理 (V7-9653)
- blusam データセットの pageSize が設定可能 (V7-9680)
- DSNUTIL - DB2LUW に 24:00:00 を有効な TIME としてロード可能 (V7-9697)
- NumberUtils.ne() /. NumberUtilseq() (V7-9731) での高値 (0xff) 比較をサポート
- JCL/GROOVY - DO ... のサポート IDCAMS IF-THEN-ELSE 句の THEN キーワードを使用して 1 つのブロックで複数のステートメントを実行 (V7-9750)
- JHDB 外のプログラムと呼ばれる無効な JHDB BatchRunner (V7-9782)
- SORT OUTFIL コントロールカードで空白文字をサポート (V7-9808)
- CICS READ PREV のサポート向上 (V7-9845)
- データセットインデックスの同時アクセスの改良 (V7-9864)
- CICS REWRITE のサポート向上 (V7-9873)
- COBOL - バッチ (JCL) からプログラム (COBOL) にデータを渡すための ACCEPT ステートメントでの複数行の SYSIN のサポート (V7-9875)
- Groovy - ファイル作成ステップ ConcatenatedFileConfiguration での の処理の向上 (V7-9876)
- IDCAMS UTILITY - DEFINE PATH ステートメントの処理 (V7-9878)

- SORT BUILD - TRAN オプションの調整と暗黙的な空白の処理 (V7-9925)
- GENERIC オプションのサポートによる CICS DELETE の改良 (V7-9939)
- CICS STARTBR と ENDBR のサポート向上 (V7-9952)
- 同時アクセス時のクローズパフォーマンスの向上 (V7-9953)
- 開始時のファイルステータス処理の改良 (V7-9991)
- Groovy - での getDisposition ()/getNormalTermination()/getAbnormalTermination() の呼び出しを許可する ConcatenatedFileConfiguration (V7-10012)

AS400

新しい特徴

- COMMIT キーワードの外部インジケータをサポート (V7-6035)
- SFLCTL 書き込み後の ReadC ループのリセット (V7-8061)
- CALL の LR インジケータのサポート (V7-9250)
- 複数行の入力フィールドを処理する新しいタイプの動的フィールド (分割) を追加 (V7-9370)
- プライマリ/セカンダリファイルをサポート (V7-9390)
- ローカルデータエリアはジョブの送信時に呼び出されたジョブに渡されるようになる (V7-9775)
- データ領域の QTEMP のサポートとデータ領域の値の作成をサポート (V7-9916)
- Commitment Control - コミットメントコントロールの有効化/無効化をサポート (V7-9956)
- COMMIT キーワードの外部インジケータのサポート

改良点

- 0 値の表示と EDTWRD の改良 (V7-8933)
- DSPF キーワード「CHKMSGID」のサポート (V7-9125)
- バッチ終了時の SQL コミットトランザクション (V7-9232)
- フィールドとデータ構造の EXPORT および IMPORT キーワードのサポート向上 (V7-9265)
- での小文字のサポート DateHelper (V7-9461)
- *CYMD から*ISO (数値) への変換をサポート (V7-9488)
- 可変フィールド (式の左側と右側) のビルトイン %len の処理の改良 (V7-9733)
- ビルトイン関数「%LOOKUPXX」(XX: LE、LT、GE、GT) のサポート向上 (V7-10064)

横断的な機能

新しい特徴

- CICS - オプションステータスのトランザクション照会の改良 (V7-9712)
- JCL - システム出力ファイルによる sysprint のロードを改良 (V7-9797)
- CICS - INQUIRE TSQUEUE の改良 (V7-9823)
- CICS - オプションユーザー ID の問い合わせターミナルの改良 (V7-9906)

改良点

- 空白の場合の比較処理の改良 (V7-8047)
- Jics と BluSam (V7-8847) のログ記録を改善
- ダイナミックフィールド用の BMS 拡張属性 SOSI とプログラムシンボル F8 のサポート (V7-8857)
- プログラムパラメータのバッファオーバーフロー処理 (V7-9138)
- Blusam ロックレジストリのスレッド書き込みの同時実行性を改良 (V7-9505)
- Utility-pgm の複数のデータソース構成のサポート (V7-9570)
- Blusam レコードレベルのロック専用モード (V7-9626)
- メタデータの永続性を確保し、サーバーの再起動の耐久度を向上 (V7-9748)
- 例外 (ブラウザを閉じる) 発生時の DAO クリーンアップを改良 (V7-9790)
- INFUTILB SYSPUNCH DummyFile のサポート (V7-9799)
- での負の値のサポートを強化する NumericEditedType (V7-9935)

モダナイゼーションツールリリース 3.6.0

トピック

- zOS
- AS400
- 横断的な機能

zOS

新しい特徴

- JCL - プロシージャ終了時の口ギングの強化 (V7-8509)
- PL1 - データ型のバッグ生成の強化 PakedLong (V7-8917)
- JCL - ファイルに「終了」マーカー「//」が含まれている場合のプロシージャ終了時の口ギングの強化 (V7-9509)
- PL1 - 固定小数点ストリームと SYSIN ストリームによる GET EDIT のサポート向上 (V7-9593)
- DB2 - VARGRAPHIC DB2 タイプのサポート向上 (V7-9809)
- CICS - オプションの LOGMESSAGE のコマンド QUERY SECURITY の改良 (V7-9969)
- PL1 - CHARG/chargraphic ビルトインのバッグ生成の改良 (V7-9989)

改良点

- PL1 - INCLUDEX キーワードのサポート向上 (V7-9588)
- PL/I - CHARGRAPHIC キーワードを任意のメソッド呼び出しの有効なパラメータとして処理 (V7-9589)
- @ # $ § という特定の文字で命名された場合の PL1 ホスト変数の解像度を改良 (V7-9654)
- COBOL - 解析ステップでの WRITE ADVANCING ステートメントのパラメータとしての C01...C12 と S01...S05 キーワードのサポート (V7-9669)

AS400

新しい特徴

- アナライザーでの SQL-DDS 変換をサポート (V7-7687)
- SQL-DDS ファイル検出の自動化 (V7-7687)
- SQL-DDS 前処理の実装 (V7-7687)
- ALIGN キーワードのサポート (V7-9254)
- TAKF および multi-dim 配列 ExtName のサポート (V7-9663)
- COBOL WRITE の Support InvalidKey ステートメント (V7-9793)

改良点

- TESTB オペレーションコードの改良 (V7-8865)
- フォーカス時の DECFMT のサポート向上 (V7-8933)

- MOVE に表示されるインジケータの処理 (V7-9224)
- フィールドとデータ構造のキーワード TEMPLATE のサポート向上 (V7-9278)
- LIKEDS の改良 (LIKEDS を使用して定義された DS を自動的に修飾) (V7-9302)
- COBOL - インジケータ構造の生成の改良 (V7-9423)
- プロトタイプの Const パラメータは読み取り専用ではありません (V7-9437)
- 編集コード「Y」による EDTCDE キーワードの改良 (V7-9443)
- PSDS および INFDS での *ROUTINE フィールドの生成をサポート (V7-9487)
- フィールド XXX のスタンドアロンへの書き換えを改良 (書き換え時にデフォルト値が失われる) (V7-9522)
- DSPF キーワードのサポート向上 (V7-9658)
- バイナリでの ZEROES デフォルト値の処理 (V7-9666)
- 暗黙的ポインタのサポート (V7-9719)
- 1 つのパラメータによるビルトイン呼び出し %size の処理を改良 (V7-9730)
- ビルトイン呼び出し (%ELEM) でのデータ構造参照の処理を改良 (V7-9736)
- 定義仕様の LIKE 参照フィールドの符号付き長の処理を改良 (V7-9738)
- REWRITE の改良 (V7-9791)
- DDS ファイルからのインデックス生成の改良 (V7-9803)
- 無効な数値によるマッパーの堅牢性を向上 (V7-9813)
- SQLModel と allIndexes ファイル生成の改良 (V7-9818)
- 修飾 DS のサポート向上 (V7-9863)
- LOOKUP のサポート向上 (パラメータに DS のようなスタンドアロンフィールドを持つ) (V7-9961)
- インジケータの LIKE の改良 (V7-9985)
- MVR に表示されるインジケータの処理 (V7-9995)
- チルダ付きの文字「N」をサポート (V7-10021)
- SQLDDS レガシーファイルからの最新の DDL ファイル生成の改良 (V7-10067)

横断的な機能

新しい特徴

- yml プロパティを使用してリソースの場所をカスタマイズ (D88816105)
- COBOL - GO TO /PERFORM...THROUGH を使用せずにインライン PERFORM を終了するための EXIT PERFORM ステートメントのサポート (V7-9582)
- グローバルメタデータを考慮したデフォルトのレガシーエンコーディングの指定 (V7-9883)

改良点

- マスク生成の改良 (V7-9602)
- コンテキストのウォームアップの改良 (V7-9621)
- Charset CUSTOM930 をスレッドセーフ化 (V7-9674)
- MOVEA での改良 (V7-9773)

リリースノート 3.5.0

AWS Blu Age ランタイムとモダナイゼーションツールのこのリリースでは、zOS と AS400 のレガシー移行の両方に新しい機能が提供されます。主にデータセットとメッセージングの最適化、および変換プロセスの結果として得られる拡張 Java 機能を対象としています。今回のリリースの主な変更点は次のとおりです。

- 既存の groovy スクリプト機能に加え、CL プログラムを Java に移行でき、他のモダナイズされたプログラムと簡単に統合できるようになります。プログラミング言語を統一することにより、顧客の習得が容易になります。
- 新しいデータバルク機能により、Redis でのデータセットをロードする時間を短縮し、パフォーマンスの最適化を実現しました。
- ジョブステップ内でデータセットを操作して渡すことができるため、従来のデータセットの動作をモダナイズできます。
- SQL マイグレーションを拡張して VB 入力ファイルと Java 11 によるシンプルな移行がサポートされるようになりました。
- ヘッダーの追加、拡張 GET/PUT のサポート、キューメタデータの自動取得など、IBM MQ との高速統合のための多くの新しいメカニズム。
- データセットのメタデータと S3 バケットからのデータセットのインポートのための REST エンドポイント。

このリリースに含まれる変更の詳細については、以下のセクションを参照してください。

ランタイムリリース 3.5.0

トピック

zOS

新しい特徴

- JCL SORT - 新しいキーワードオーバーレイの処理 (V7-9409)
- ZOS COBOL - フローティング文字のサポート向上 (V7-9404)
- & RedisTemplate ListOperations (V7-9212) への RedisJicsTSQueue のポート
- ZOS JCL - UserDefinedParameters (V7-9012) で定義されている場合は、ファイルディレクトリを使用して一時ディレクトリのパスを強化します。
- FUNCTION ORD-MAX を ALL (すべての配列項目) で処理 (V7-9366)
- Redis に TS キューを格納する際に、人間が読める接頭辞付きのキーを使用 (V7-9212)
- blusam API のデータセットエンドポイントの取得を追加
- JCL - # のような特殊文字を含む名前のバッチジョブのサポートを追加 (V7-9136)
- TSModel のフェッチがオンデマンドで確実に実行されるようになりました (V7-9212)

改良点

- LNK ファイルでのバージョニング非対応の INCLUDE をサポート (V7-6022)
- MQ - エンコーディングのサポート向上 (V7-9652)
- さまざまな文字タイプに対応する 2 バイトまたは混合文字セットのサポート向上 (V7-9596)
- JCL - IDCAMS delete NONVSAM ステートメントにおける filesDirectory 構成のサポート (V7-9609)
- ESDS と RRDS のデータセットをファイルからロードするバルクモードをサポート (V7-8639)
- 入力モードで空の ESDS を開く処理 (V7-9287)
- ORD/UNORD 省略形のサポートによる DEFINE CLUSTER ステートメントの向上 (V7-9451)

- BluSam Redis ロックパフォーマンスの向上 (V7-8639)
- DEFINE CLUSTER ステートメントを拡張し、DATA() 引数のスコープで指定された RECORDSIZE をサポート (V7-9337)
- DEFINE CLUSTER ステートメントに BUFFERSPACE/UNIQUE 属性のサポートを追加 (V7-9419)
- 可変長レコードデータセットの BluSam 読み取りオペレーションを改善します。(V7-9391)
- CICS ADDRESS が欠落している CWA を null として適切に表示 (V7-9491)
- エンドロック時の不要な書き込みを削除 (V7-8639)
- キャッシュ内の Redis キャッシュテンプレートインジェクションの処理 (V7-9510)
- BPXWDYN パラメータを正しくデコード (V7-9417)
- LISTCAT エクスポートの消費量の改善 (V7-9201)
- BluSam TS キュー名での印刷不可能な文字のサポート (V7-9212)
- マップセットが null のフィールドの受信マップ構築の処理 (V7-9486)
- 動的アクセスモード BluesamRelativeFile の削除および書き換えオペレーションを改善します。(V7-8989)

AS400

新しい特徴

- 標準 DS/STM ピボットを使用して CL ファイルを Java プログラムとして生成する機能を追加 (V7-9427)
- ADD モードでの入力ファイルをサポート (V7-9378)
- cl コマンド OPNQRYF (Open Query File) をサポートするソート順序と取得管理が改善され、 で SHARE パラメータのサポートが追加されました OverrideItem。(V7-9364)

改良点

- での SFLNXTCHG のサポート UpdateSubfile (V7-8061)
- CL コマンドの実行時に CL コンテキストのスコープを変更 (V7-9624)
- プログラム BPXWDYN のリターンコードの処理 (V7-9417)
- ローカルモニタの消去 (V7-9624)
- DSPF キーワード RTNCSRLOC のサポート (V7-9389)

- setOnGreaterOrEqual() が Equal を 1 に設定していない (V7-9342)
- でのフィールドキャッシュの更新 UpdateSubfileRecord (V7-9376)
- SFLNXTCHG のサポート向上 (V7-8061)

横断的な機能

新しい特徴

- リテラルグラフィック文字列の G のプレフィックスを無視 (V7-9420)
- ZOS COBOL - 一部の特殊構造に対する Fiedl.initialize() のサポート向上 (V7-9485)
- プログラムスタートアップ時のパフォーマンス向上のため、コンテキストの非同期初期化を許可 (V7-9446)
- SQL リリースは、開かれた準備ステートメント および を明示的にリリースします ResulSet。 (V7-9422)
- JMS MQ の向上 - MQ PUT /V7-7085 の MQRFH2 をサポート - デフォルトキューマネージャーのサポート (V7-9400)
- SQL 管理: SET コマンドのパラメータの Lambda 変換を有効化 (V7-9492)
- ZOS MQ JMS - MQCOMIT と MQBACK にサポートを追加 (V7-9399)
- ZOS IBMMQ - MQINQ のサポート向上 (V7-9544)
- 2 バイトエンコーディングを使用する場合、CONCAT オペレーションを文字列ではなくバイトで処理 (V7-8932)
- ZOS IBMMQ - オプションの SET_ALL_CONTEXT による PUT コマンドのサポート向上 (V7-9544)

改良点

- $ 文字を使用した GDG ファイル名の処理 (V7-9066)
- SQL 診断では、直前の SQL ステートメントが成功すると NUMBER 句として 1 を返します (V7-9410)
- 長さが null ではないフィールドのアウトライン (V7-7536)
- ビルトイン PL1 GRAPHIC 関数のサポート (V7-9245)
- MQ - MQGMO フィールド設定用のバージョンのサポートを追加 (V7-9500)
- JMS MQ GET - メッセージが返す dataLength の改良 (V7-9502)
- sqlerrd(3) に ROWSET コンテキストのフェッチされた項目数を設定 (V7-9371)

モダナイゼーションツールリリース 3.5.0

トピック

zOS

新しい特徴

- ZOS PLI - バイナリ式の代入でアスタリスクインデックスをサポート (V7-9178)
- JCL から BatchScript - ジョブ実行の終了を示す「//」 (V7-9304)
- ZOS PLI - 数値編集タイプのフローティング文字とサインインのサポート向上 (V7-8982)
- COBOL - ビルトイン SUM 関数のサポート (V7-9367)
- JCL - オプションで null ステートメント (//) の後にデッドコードをコメント (V7-9202)
- JCL - 条件ステートメントにおける演算子「|」のサポート (V7-9499)
- PL/I - 解析の例外を防ぐための前処理ステップでのプリコンパイラディレクティブのコメント (V7-9507)

改良点

- 区切り文字によるストリーム定義の処理 (V7-9615)
- LISTCAT エクスポート処理の改良 (V7-9201)
- PL/I - 暗黙的な「null」引数をサポートするための機能拡張 (V7-9204)

AS400

新しい特徴

- DDS キーワード CONCAT のサポート (V7-9439)
- 生成された Java コードを DSPF キーワード用にリファクタリング (V7-7700)
- データ構造定義内のフィールドでの可変キーワードのサポート (V7-9029)

改良点

- 論理関係 AND/OR の解析を改良 (V7-9352)
- COBOL の vo と dsEntity 間のマッピングを改良 (V7-9449)
- 数値入力にフォーカスされている場合に空の値を表示 (V7-9374)
- SQL 宣言カーソル内のローカル変数 (V7-9456)
- DS が空の場合のスコープの問題 (V7-9466)
- 解析前に 80 列以降の行を切り捨て (V7-9632)
- 定義仕様のキーワード (DIM、LIKE など) のフィールド参照とビルトイン呼び出しの処理を改良 (V7-9358)
- SQL コメント (--) をサポート (V7-9632)
- FullFree 解析、タイプ日付/時刻/タイムスタンプ (V7-9542)
- FullFree 解析からの SQLCA を含める (V7-9333)
- コントロールレベルのサポート向上 (V7-9610)
- *BLANKS による DS の比較処理 (V7-9668)
- DDS での複数のインジケータのサポート向上 (V7-9318)
- 複数の DSPF プログラムのサポート向上 (V7-9657)
- LIKE によるフィールドの処理向上 (データ構造が似ている場合と配列内のデータ構造が似ている場合) (V7-9213)
- フリー RPG、リテラルでの継続処理 (V7-9686)
- プログラム終了のレコードのサポート向上 (V7-9452)
- CALL ステートメントの LINKAGE フレーズのサポート (V7-9685)
- CASXX オペレーションコード (CASXX グループのない CASBB) (V7-9357)
- FullFreeRPG 解析の改善 (V7-9457)
- ビルトイン %LEN は DS を引数としてサポートしていません (V7-9267)
- ファクター 2 が *ALL'X...' の場合の MOVEA の改良 (V7-9228)
- RENAME フィールドによる割り当てのサポート (V7-9385)

横断的な機能

新しい特徴

- SQL Migrator ツール - ebcdic ローディングステップで可変レコード長の OID オプションを追加 (V7-9380)
- SQL Migrator ツール - OID オプションでの Java 11 のサポート (V7-9599)

改良点

- ネストされた配列のサポート向上 (V7-9595)
- 元のエンコーディングで「 Â¬」がサポートされている場合、「Â¬」を「!」に置き換える (V7-9465)
- JCL - ジョブステップ間でデータセットを共有するための PASS 正常終了のサポート (V7-9504)
- ORACLE で VARCHAR と null 許容型の DB 列タイプを扱う場合、カラム定義に null を適用 (V7-9681)
- Spring インジェクションコンプライアンスの向上 (V7-9635)

AWS Blu Age ランタイムの概念

AWS Blu Age ランタイムの基本概念を理解することは、自動リファクタリングによってアプリケーションがどのようにモダナイズされるかを理解するのに役立ちます。

トピック

- AWS Blu Age ランタイムハイレベルアーキテクチャ
- AWS モダナイズされたアプリケーションの Blu Age 構造
- Data Simplifier

AWS Blu Age ランタイムハイレベルアーキテクチャ

レガシープログラムを Java にモダナイズするための AWS Blu Age ソリューションの一環として、AWS Blu Age ランタイムは、レガシーコンストラクトを提供し、プログラムコード組織を標準化するライブラリを通じて、モダナイズされたアプリケーション用の統合 REST ベースのエントリポイントと、そのようなアプリケーションの実行フレームワークを提供します。

このようなモダナイズされたアプリケーションは、メインフレームプログラムと中間プログラム (次のドキュメントでは「レガシー」と呼びます) をウェブベースのアーキテクチャにモダナイズするための AWS Blu Age 自動リファクタリングプロセスの結果です。

AWS Blu Age Runtime の目標は、tomcat、Spring、getters/setters、fluent APIs...

トピック

AWS Blu Age ランタイムコンポーネント

AWS Blu Age ランタイム環境は、次の 2 種類のコンポーネントで構成されています。

- しばしば「共有フォルダ」と呼ばれ、従来の構造やステートメントを提供する java ライブラリ (jar ファイル) のセット。
- モダナイズされたプログラムに共通のフレームワークとサービスを提供する、Spring ベースの一連のウェブアプリケーション (war ファイル)。

次のセクションでは、これらの両方のコンポーネントの役割について詳しく説明します。

AWS Blu Age ライブラリ

AWS Blu Age ライブラリは、モダナイズされたすべての Java プログラムで使用できるように、標準の tomcat クラスパスに追加されたshared/サブフォルダに保存されている一連の jar ファイルです。これは Java プログラミング環境ではそのまま簡単に利用することはできませんが、従来の開発環境によくある機能を提供することを目的としています。これらの機能は、Java 開発者ができるだけ使い慣れた方法 (ゲッター/セッター、クラスベース、fluent API) で公開されています。重要な例としては、(COBOL、PL1、または RPG 言語に存在する) 従来のメモリレイアウトと操作コンストラクトを Java プログラムに提供する Data Simplifier ライブラリがあります。それらの jar は、レガシープログラムから生成されたモダナイズされた Java コードの主要な依存関係です。Data Simplifier の詳細については、「Data Simplifier」を参照してください。

ウェブアプリケーション

ウェブアプリケーションアーカイブ (WAR) は、コードとアプリケーションを tomcat アプリケーションサーバーにデプロイするための標準的な方法です。 AWS Blu Age ランタイムの一部として提供されるものは、レガシー環境とトランザクションモニター (JCL バッチ、CICS、IMS...)、および関連する必要な サービスを再生成する一連の実行フレームワークを提供することを目指しています。

最も重要なのものは gapwalk-application (しばしば「Gapwalk」と略される) で、これはトランザクション、プログラム、バッチの実行をトリガーおよび制御するための REST ベースのエントリポイントの統合セットを提供します。詳細については、「AWS Blu Age ランタイム APIs」を参照してください。

このウェブアプリケーションは Java 実行スレッドとリソースを割り当て、モダナイズされたプログラムを設計時のコンテキストで実行します。このような再現環境の例については、次のセクションで詳しく説明します。

他のウェブアプリケーションは、レガシープログラムから使用可能な、またはレガシープログラムから呼び出し可能なプログラムをエミュレートするプログラムを、実行環境 (より正確には、後述の「プログラムレジストリ」) に追加します。以下の、2 つの重要なカテゴリがあります。

- OS が提供するプログラムのエミュレーション: JCL 駆動のバッチは、特に標準環境の一部としてさまざまなファイルやデータベースを操作するプログラムを呼び出せる必要があります。例には SORT/DFSORT または IDCAMS が含まれます。この目的のために、このような動作を再現し、従来のものと同じ規則を使用して呼び出し可能な Java プログラムが用意されています。
- 「ドライバー」とは、実行フレームワークまたはミドルウェアがエントリポイントとして提供する特殊なプログラムです。1 つの例は CBLTDLI で、IMS 環境で実行される COBOL プログラムは、IMS 関連サービス (IMS DB、MFS を介したユーザーダイアログなど) へのアクセスに依存しています。

プログラムレジストリ

これらのコンストラクト、フレームワーク、サービスに追加して活用するために、レガシープログラムからモダナイズされた Java プログラムは、AWS モダナイズされたアプリケーションの Blu Age 構造 に記載されている特定の構造に準拠しています。起動時に、 AWS Blu Age ランタイムはそのようなプログラムをすべて共通の「Programs Registry」に収集し、後で呼び出す (および相互に呼び出す) ことができるようにします。プログラムレジストリは疎結合であり、分解の可能性もあります (相互に呼び出すプログラムは同時にモダナイズする必要がないため)。

実行環境

よく見かけるレガシー環境やコレオグラフィは以下のとおりです。

- JCL 駆動型バッチは Java プログラムと Groovy スクリプトにモダナイズされると、同期 (ブロッキング) または非同期 (デタッチ) で起動できます。後者の場合、その実行は REST エンドポイントを通じて監視できます。

- AWS Blu Age サブシステムは、以下を通じて CICS と同様の実行環境を提供します。
  - CICS の「実行レベル」のコレオグラフィを順守しながら、CICS トランザクションを開始して関連プログラムを実行するために使用されるエントリポイント。
  - リソース定義用の外部ストレージ
  - EXEC CICS ステートメントを再現する同質の Java fluent API セット
  - 一時ストレージキュー、一時データキュー、ファイルアクセスなどの CICS サービスを再現するプラガブルクラスのセット (通常、Amazon Managed Service for Apache Flink、Amazon Simple Queue Service、または RabbitMQ for TD Queues などを複数実装することができます)。
  - ユーザー向けアプリケーションでは、BMS 画面記述形式が Angular ウェブアプリケーションにモダナイズされ、対応する「疑似会話」ダイアログがサポートされます。
- 同様に、別のサブシステムには IMS メッセージベースのコレオグラフィが用意されており、MFS 形式の UI 画面のモダナイズをサポートします。
- さらに、3 つ目のサブシステムを使用すると、DSPF (ファイル表示) 指定画面のモダナイズなどを含む、iSeries のような環境でプログラムを実行できます。

これらの環境はすべて、次のような一般的な OS レベルのサービスに基づいて構築されています。

- 従来のメモリ割り当てとレイアウトのエミュレーション (Data Simplifier)、
- COBOL 「実行ユニット」の実行とパラメータ受け渡しメカニズム (CALL ステートメント) の Java スレッドベースでの再現、
- フラット、連結、VSAM (Bluesam ライブラリセットを使用) 、GDG データセット組織のエミュレーション、
- RDBMS (EXEC SQL ステートメント) などのデータストアへのアクセス。

ステートレス性とセッション処理

AWS Blu Age ランタイムの重要な機能は、モダナイズされたプログラムを実行するときに高可用性 (HA) と水平スケーラビリティのシナリオを有効にすることです。

その基礎となるのがステートレス性で、その重要な例は HTTP セッション処理です。

セッション処理

tomcat はウェブベースであるため、このための重要なメカニズムは HTTP セッション処理 (tomcat と Spring が提供している) とステートレス設計です。こうしたステートレス設計は以下に基づいています。

- ユーザーは HTTPS 経由で接続し、
- アプリケーションサーバーはロードバランサーの背後にデプロイされ、
- ユーザーがアプリケーションに初めて接続すると認証が行われ、アプリケーションサーバーは (通常は cookie 内で) 識別子を作成します。
- この識別子は、ユーザーコンテキストを外部キャッシュに保存、および外部キャッシュから取得するためのキーとして使用されます (データストア)。

Cookie 管理は AWS Blu Age フレームワークと基盤となる tomcat サーバーによって自動的に行われます。これはユーザーには透過的です。ユーザーのインターネットブラウザがこれを自動的に管理します。

Gapwalk ウェブアプリケーションは、セッション状態 (コンテキスト) を以下のさまざまなデータストアに保存できます。

- Amazon ElastiCache for Redis
- Redis クラスター
- メモリマップ内 (開発環境とスタンドアロン環境のみで、HA には適していません)。

高可用性とステートレス性

より一般的には、 AWS Blu Age フレームワークの設計原則はステートレスです。レガシープログラムの動作を再現するために必要なほとんどの非一時的な状態は、アプリケーションサーバー内に保存されるのではなく、外部で共通の「信頼できる唯一のソース」を通じて共有されます。

このような状態の例は、CICS の一時ストレージキューまたはリソース定義で、それらの一般的な外部ストレージは Redis 互換のサーバーやリレーショナルデータベースです。

この設計をロードバランサーおよび共有セッションと組み合わせると、ほとんどのユーザー向けダイアログ (OLTP、「オンライントランザクション処理」) を複数の「ノード」(ここでは tomcat インスタンス) に分散できるようになります。

実際、ユーザーはどのサーバーでもトランザクションを実行でき、次のトランザクションの呼び出しが別のサーバーで実行されても構いません。そうすれば、(自動スケーリングのため、または正常でないサーバーを置き換えるために) 新しいサーバーが作成されたときに、接続可能で正常なサーバーがトランザクションを期待どおりに実行し、適切な結果 (期待される戻り値、データベース内での予想されるデータ変更など) が得られるようになります。

AWS モダナイズされたアプリケーションの Blu Age 構造

このドキュメントでは、デベロッパーが次のようなさまざまなタスクを実行できるように、モダナイズされたアプリケーションの構造 (AWS Mainframe Modernization リファクタリングツールを使用) について詳しく説明します。

- アプリケーションへのスムーズなナビゲーション。
- モダナイズされたアプリケーションから呼び出せるカスタムプログラムの開発。
- モダナイズされたアプリケーションの安全なリファクタリング。

次の内容に関する基本的な知識を既に保持していることを前提としています。

- レコード、データセット、レコードへのアクセスモード (インデックス付き、シーケンシャル)、VSAM、実行ユニット、jcl スクリプト、CICS 概念など、従来の共通コーディング概念。
- Spring フレームワークを使用した Java コーディング。
- ドキュメント全体をとおして、読みやすさを考慮して `short class names` を使用しています。詳細については、AWS Blu Age 完全修飾名マッピング「」を参照して AWS Blu Age ランタイム要素に対応する完全修飾名を取得し、サードパーティーの完全修飾名マッピング「」を参照してサードパーティー要素に対応する完全修飾名を取得します。
- すべてのアーティファクトとサンプルは、サンプル COBOL/CICS CardDemo アプリケーションのモダナイゼーションプロセスの出力から取得されます。

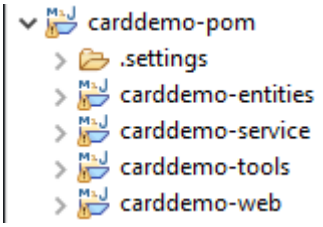
トピック

- アーティファクトの組織
- プログラムの実行と呼び出し
- 独自のプログラムの作成
- 完全修飾名マッピング

アーティファクトの組織

AWS Blu Age のモダナイズされたアプリケーションは、JEE サーバーにデプロイできる java ウェブアプリケーション (.war) としてパッケージ化されています。通常、サーバーは AWS Blu Age Velocity ランタイムを埋め込む Tomcat インスタンスであり、現在 Springboot および Angular (UI 部分用) フレームワークに基づいて構築されています。

war は複数のコンポーネントアーティファクト (.jar) を統合します。各 jar は専用の Java プロジェクトを (maven ツールを使用して) コンパイルして生成され、その要素はモダナイズプロセスの結果生じます。



基本的な組織は以下の構造に基づいています。

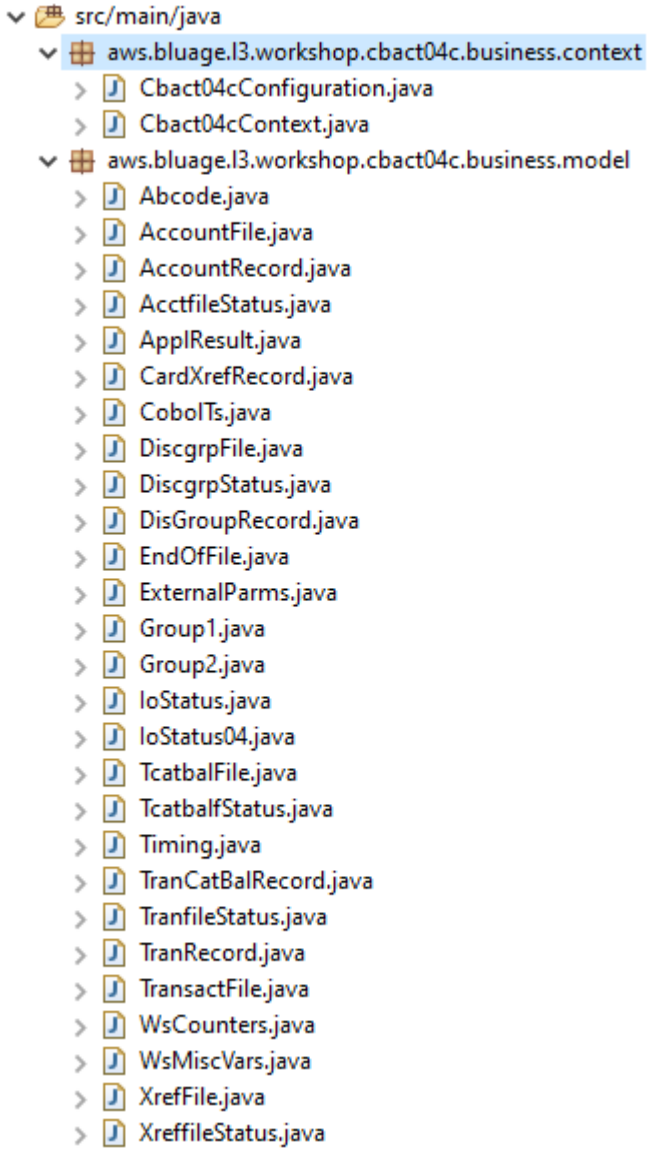
- エンティティプロジェクト: ビジネスモデルとコンテキスト要素が含まれます。プロジェクト名は通常「-entities」で終わります。通常は、レガシー COBOL プログラムであると仮定すると、これは I/O セクション (データセット) とデータ部のモダナイズに相当します。複数のエンティティプロジェクトを使用することができます。
- サービスプロジェクト: レガシービジネスロジックのモダナイズ要素が含まれています。通常は COBOL プログラムの手続き部です。複数のサービスプロジェクトを使用することもできます。
- ユーティリティプロジェクト: 他のプロジェクトで使用されている、共通のツールやユーティリティが含まれています。
- ウェブプロジェクト: UI 関連の要素をモダナイズしたもの (該当する場合) が含まれます。バッチのみのモダナイズプロジェクトには使用されません。これらの UI 要素は、CICS BMS マップ、IMS MFS コンポーネント、その他のメインフレーム UI ソースに由来する場合があります。複数のウェブプロジェクトを使用することができます。

エンティティプロジェクトの内容

> **Note**
>
> 以下の説明は COBOL と PL/I のモダナイゼーション出力にのみ適用されます。RPG モダナイゼーション出力は異なるレイアウトに基づいています。

リファクタリングを行う前に、エンティティプロジェクトのパッケージ構成をモダナイズプログラムと関連付けます。これを実行するための、いくつか方法があります。推奨される方法は、コード生成メカニズムを起動する前に動作する、リファクタリングツールボックスを使用することです。これは、 BluAge トレーニングで説明されている高度な操作です。詳細については、「リファクタリングワークショップ」を参照してください。このアプローチでは後で Java コードを再生成する機能を維

持できるため、将来さらに改善できるというメリットがあります。もう 1 つの方法は、自己責任でお好みの Java リファクタリングアプローチを使用して、生成されたソースコードに対して直接、標準の Java リファクタリングを行うことです。



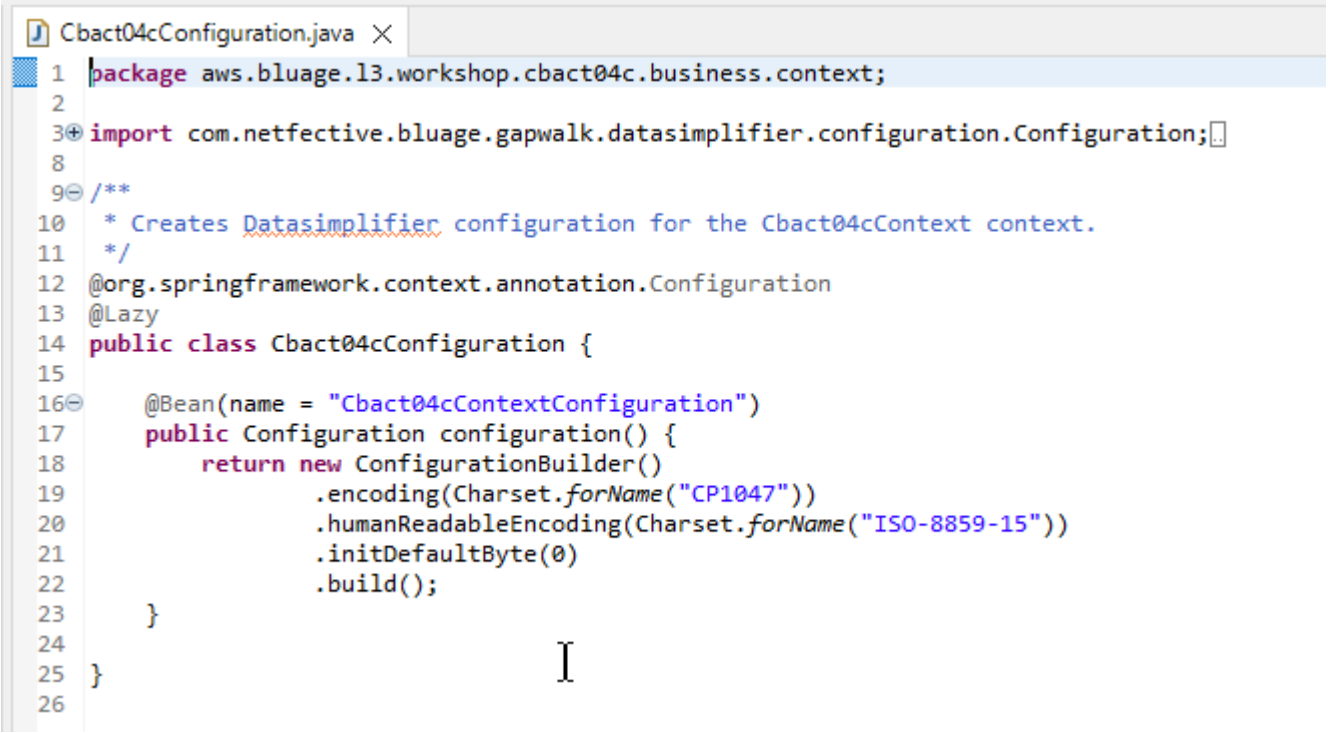
プログラム関連クラス

各モダナイズプログラムは、business.context パッケージと business.model パッケージという、2 つのパッケージに関連付けられます。

- *base package*.*program*.business.context

  business.context サブパッケージには、構成クラスとコンテキストクラスという 2 つのクラスが含まれています。

- プログラムの 1 つの構成クラスには、文字ベースのデータ要素を表すために使用する文字セット、データ構造要素をパディングするためのデフォルトバイト値など、特定のプログラム固有の構成情報が含まれます。クラス名は「Configuration」で終わります。`@org.springframework.context.annotation.Configuration` アノテーションが付いており、正しくセットアップされた `Configuration` オブジェクトを返す必要がある単一メソッドが含まれています。


Cbact04cConfiguration.java

```
1  package aws.bluage.l3.workshop.cbact04c.business.context;
2
3  import com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration;
8
9  /**
10  * Creates Datasimplifier configuration for the Cbact04cContext context.
11  */
12 @org.springframework.context.annotation.Configuration
13 @Lazy
14 public class Cbact04cConfiguration {
15
16     @Bean(name = "Cbact04cContextConfiguration")
17     public Configuration configuration() {
18         return new ConfigurationBuilder()
19                 .encoding(Charset.forName("CP1047"))
20                 .humanReadableEncoding(Charset.forName("ISO-8859-15"))
21                 .initDefaultByte(0)
22                 .build();
23     }
24
25 }
26
```

- 1 つのコンテキストクラスは、プログラムサービスクラス (以下を参照) と、モデルサブパッケージ (以下を参照) のデータ構造 (`Record`) やデータセット (`File`) との間のブリッジとして機能します。クラス名は「Context」で終わり、`RuntimeContext` クラスのサブクラスです。

Cbact04cContext.java

```
@Component("aws.bluage.l3.workshop.cbact04c.business.context.Cbact04cContext")
@Import({
    aws.bluage.l3.workshop.cbact04c.business.model.TcatbalFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.XrefFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.DiscgrpFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.AccountFile.class
,   aws.bluage.l3.workshop.cbact04c.business.model.TransactFile.class
})
@Lazy
@Scope("prototype")
public class Cbact04cContext extends JicsRuntimeContext {

    @Autowired
    private TcatbalFile tcatbalFile;

    @Autowired
    private XrefFile xrefFile;

    @Autowired
    private DiscgrpFile discgrpFile;

    @Autowired
    private AccountFile accountFile;

    @Autowired
    private TransactFile transactFile;

    private IndexedFile tcatbalFileFile;

    private IndexedFile xrefFileFile;

    private IndexedFile discgrpFileFile;

    private IndexedFile accountFileFile;

    private SequentialFile transactFileFile;

    private TranCatBalRecord tranCatBalRecord;
    private TcatbalfStatus tcatbalfStatus;
    private CardXrefRecord cardXrefRecord;
```

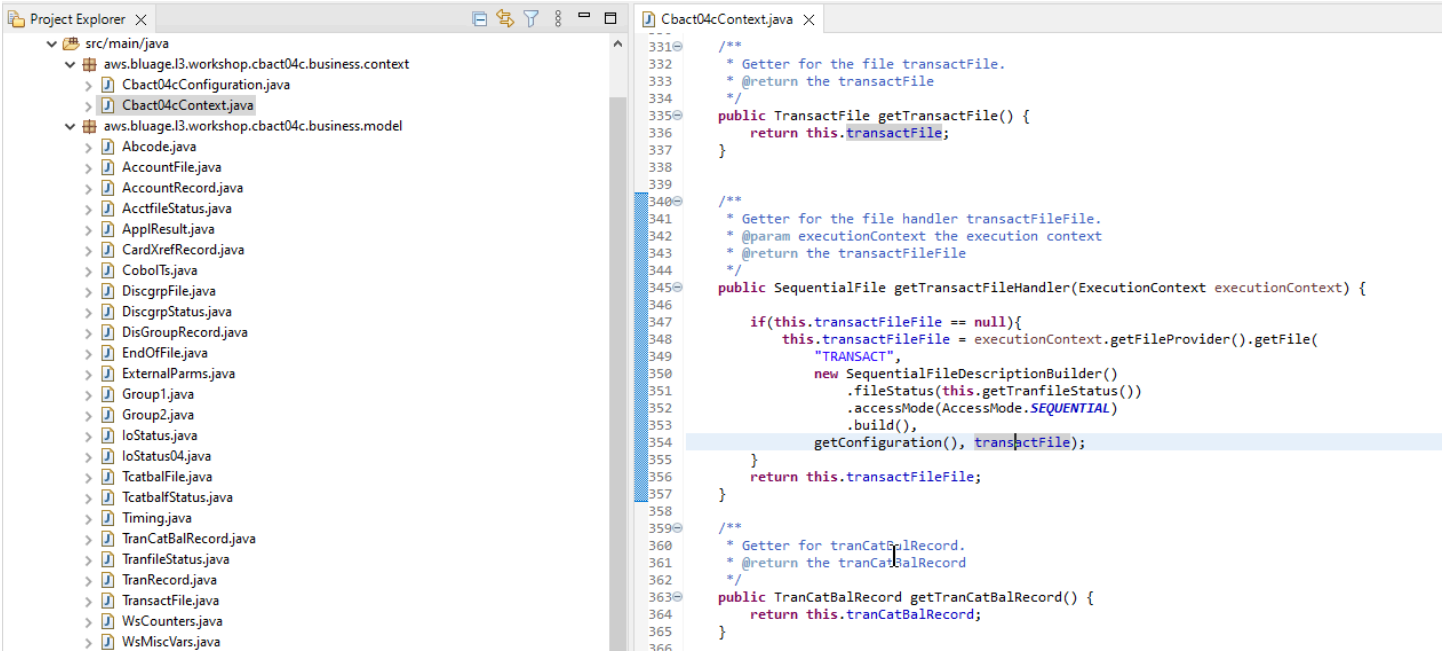
- *base package*.*program*.business.model

  モデルサブパッケージには、特定のプログラムが使用できるすべてのデータ構造が含まれています。例えば、01 レベルの COBOL データ構造はいずれもモデルサブパッケージ内のクラスに対応します (下位レベルのデータ構造は、それぞれ独自の 01 レベル構造のプロパティです)。01 データ構造をモダナイズする方法については、「Data Simplifier」を参照してください。

DiscgrpFile.java

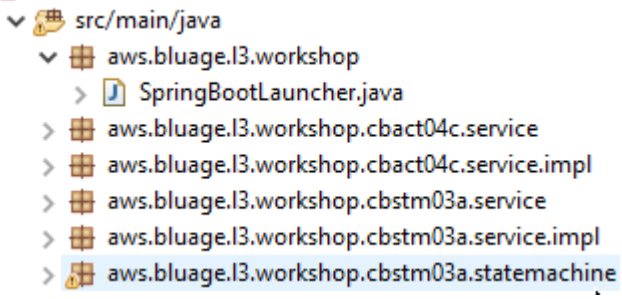
```java
package aws.bluage.l3.workshop.cbact04c.business.model;

import com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration;
import com.netfective.bluage.gapwalk.datasimplifier.data.structure.Elementary;
import com.netfective.bluage.gapwalk.datasimplifier.data.structure.Group;
import com.netfective.bluage.gapwalk.datasimplifier.entity.ElementaryRangeReference;
import com.netfective.bluage.gapwalk.datasimplifier.entity.RangeReference;
import com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity;
import com.netfective.bluage.gapwalk.datasimplifier.metadata.type.AlphanumericType;
import com.netfective.bluage.gapwalk.datasimplifier.metadata.type.ZonedType;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

/**
 * Data simplifier file DiscgrpFile.
 *
 * <p>About 'fdDiscgrpRec' field, <br>uml entity: aws.bluage.l3.workshop.cbact04c.business.model.FdDiscgrpRec
 * <br></p>
 *
 */
@Component("aws.bluage.l3.workshop.cbact04c.business.model.DiscgrpFile")
@Lazy
@Scope("prototype")
public class DiscgrpFile extends RecordEntity {

    private final Group root = new Group(getData());
    private final Group fdDiscgrpRec = new Group(root);
    private final Group fdDiscgrpKey = new Group(fdDiscgrpRec);
    private final Elementary fdDisAcctGroupId = new Elementary(fdDiscgrpKey,new AlphanumericType(10));
    private final Elementary fdDisTranTypeCd = new Elementary(fdDiscgrpKey,new AlphanumericType(2));
    private final Elementary fdDisTranCatCd = new Elementary(fdDiscgrpKey,new ZonedType(4, 0, false));
    private final Elementary fdDiscgrpData = new Elementary(fdDiscgrpRec,new AlphanumericType(34));
```

すべてのクラスは、ビジネスレコード表現へのアクセスを表す `RecordEntity` クラスを拡張します。レコードの中には、`File` にバインドされている特別な目的を持つものもあります。`Record` との間のバインディング`File`は、ファイルオブジェクトの作成時にコンテキストクラスにある対応する *FileHandler methods で行われます。例えば、次のリストは、 が TransactfileFile `File` `Record` (モデルサブパッケージから) transactFile にどのようにバインドされるかを示しています。



サービスプロジェクトの内容

すべてのサービスプロジェクトには、アーキテクチャのバックボーンとして使用される、専用の Springboot アプリケーションが付属しています。これは、サービス java ソースのベースパッケージにある、SpringBootLauncher という名前のクラスを通じて実現されます。



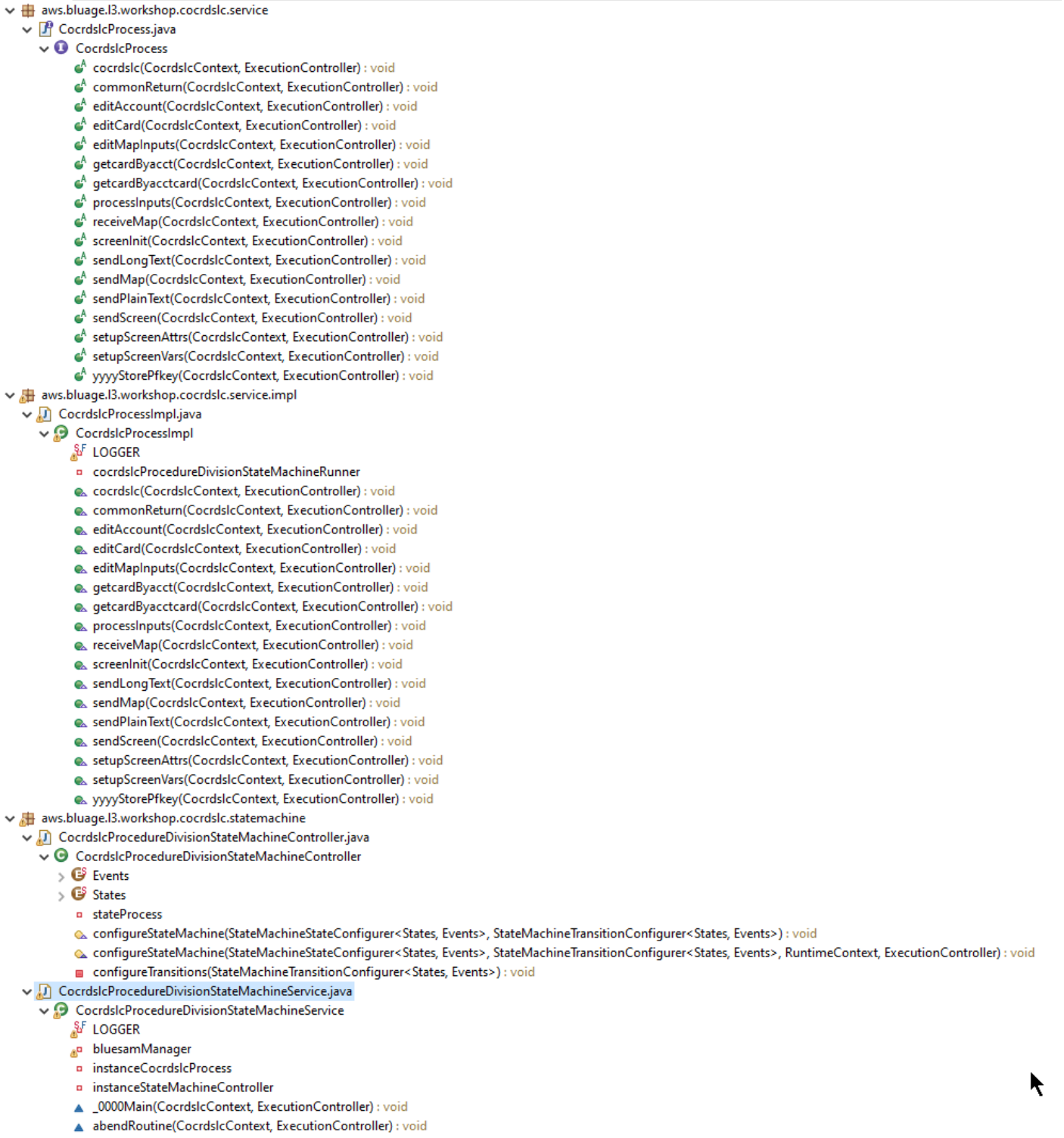
このクラスは特に以下の役割を果たします。

- プログラムクラスとマネージドリソース (データソース、トランザクションマネージャー、データセットマッピングなど) を結合します。
- プログラムに ConfigurableApplicationContext を提供します。
- spring コンポーネント (@Component) とマークされたクラスをすべて検出します。
- プログラムが ProgramRegistry に正しく登録されていることを確認します。この登録を管理する初期化メソッドを参照してください。

```
/**
 * Initialization method called when the spring application is ready.
 * Register all programs and services to the gapwalk shared context.
 * @param event the application ready event
 */
@EventListener
public void initialize(ApplicationReadyEvent event) {
    Map<String, ProgramContainer> programContainers = event.getApplicationContext().getBeansOfType(ProgramContainer.class);
    programContainers.values().forEach(ProgramRegistry::registerProgram);
    Map<String, ServiceContainer> serviceContainers = event.getApplicationContext().getBeansOfType(ServiceContainer.class);
    serviceContainers.values().forEach(ServiceRegistry::registerService);
}
```

プログラム関連アーティファクト

事前にリファクタリングを行わなくても、ビジネスロジックのモダナイゼーション出力は、レガシープログラムごとに以下の 2 つまたは 3 つのパッケージにまとめられます。



最も包括的なケースには、次の 3 つのパッケージがあります。

- *base package*.*program*.service: ProgramProcess という名前のインターフェイスが含まれ、これには従来の実行制御フローを維持したままビジネスロジックを処理するビジネスメソッドが含まれています。
- *base package*.*program*.service.impl: には、プログラム ProcessImplという名前のクラスが含まれています。これは、前述のプロセスインターフェイスの実装です。レガシーステートメントが java ステートメントに「変換」される場所は、 AWS Blu Age フレームワークによって異なります。

CocrdslcProcessImpl.java

```
210     /**
211      * Process operation sendScreen.
212      *
213      * @param ctx
214      * @param ctrl
215      */
216     @Override
217     public void sendScreen(final CocrdslcContext ctx, final ExecutionController ctrl) {
218         ctx.getCcWorkAreas().setCcardNextMapset(ctx.getWsLiterals().getLitThismapset());
219         ctx.getCcWorkAreas().setCcardNextMap(ctx.getWsLiterals().getLitThismap());
220         ctx.getCarddemoCommarea().setCdemoPgmReenter(true);
221         SendMapBuilder.newInstance(ctx.getDfheiblk(), ctx)
222         .withMap(ctx.getCcWorkAreas().getCcardNextMap())
223         .withMapset(ctx.getCcWorkAreas().getCcardNextMapset())
224         .withData(ctx.getGroup1().getCcrdslaoReference())
225         .withCursor()
226         .withErase()
227         .withFreeKB()
228         .execute();
229         ctx.getWsMiscStorage().setWsRespCd(ctx.getDfheiblk().getEibresp());
230     }
231
232     /**
233      * Process operation processInputs.
234      *
235      * @param ctx
236      * @param ctrl
237      */
238     @Override
239     public void processInputs(final CocrdslcContext ctx, final ExecutionController ctrl) {
240         receiveMap(ctx, ctrl);
241         editMapInputs(ctx, ctrl);
242         ctx.getCcWorkAreas().setCcardErrorMsg(ctx.getWsMiscStorage().getWsReturnMsg());
243         ctx.getCcWorkAreas().setCcardNextProg(ctx.getWsLiterals().getLitThispgm());
244         ctx.getCcWorkAreas().setCcardNextMapset(ctx.getWsLiterals().getLitThismapset());
245         ctx.getCcWorkAreas().setCcardNextMap(ctx.getWsLiterals().getLitThismap());
246     }
247
```

- *base package*.*program*.statemachine: このパッケージが常にあるとは限りません。これは、レガシー制御フローのモダナイゼーションで、レガシー実行フローを適切にカバーするためにステートマシンアプローチ (Spring StateMachine フレームワーク を使用する) を使用する必要がある場合に必要です。

その場合、statemachine サブパッケージには次の 2 つのクラスが含まれます。

- *Program*ProcedureDivisionStateMachineController: Spring ステートマシン構造の駆動に使用される StateMachineController (ステートマシンの実行の制御に必要な操作を定義) インターフェイスと StateMachineRunner (ステートマシンの実行に必須の操作を定義) インターフェイスを実装しているクラスを拡張するクラス (ケース例の SimpleStateMachineController など)。

CocrdslcProcedureDivisionStateMachineController.java

```java
package aws.bluage.l3.workshop.cocrdslc.statemachine;

import aws.bluage.l3.workshop.cocrdslc.business.context.CocrdslcContext;

/**
 * Controller managing the state machine "CocrdslcProcedureDivisionStateMachine" execution.
 */
@Component("aws.bluage.l3.workshop.cocrdslc.statemachine.CocrdslcProcedureDivisionStateMachineController")
@Import({
aws.bluage.l3.workshop.cocrdslc.statemachine.CocrdslcProcedureDivisionStateMachineService.class
})
@Lazy
public class CocrdslcProcedureDivisionStateMachineController extends SimpleStateMachineController<States, Events> {

    /**
     * State machine states.
     */
    public enum States {
        _0000_MAIN_1, _0000_MAIN, ABEND_ROUTINE, FINAL, LOCAL_FINAL
    }

    /**
     * State machine events.
     */
    public enum Events {
        TO__0000_MAIN_1, TO__0000_MAIN, TO_ABEND_ROUTINE, TO_FINAL, TO_LOCAL_FINAL
    }

    /**
     * State machine state process service provider.
     */
    @Autowired
    @Lazy
    private CocrdslcProcedureDivisionStateMachineService stateProcess;

    @Override
    protected void configureStateMachine(StateMachineStateConfigurer<States, Events> states, StateMachineTransitionConfigurer<States, Events> transitions) throws Exception {
        throw new UnsupportedOperationException("Please use the four arguments configureStateMachine method instead: configureStateMachine(StateMachineStateConfigurer<States, Events> states, "
                + "StateMachineTransitionConfigurer<States, Events> transitions, RuntimeContext ctx, ExecutionController ctrl)");
    }

    @Override
    protected void configureStateMachine(StateMachineStateConfigurer<States, Events> states, StateMachineTransitionConfigurer<States, Events> transitions, RuntimeContext ctx, ExecutionController ctrl) throws Exception {
        StateConfigurer<States, Events> configurer = states.withStates();
        configurer.initial(States._0000_MAIN_1).end(States.FINAL);
        configurer.state(States._0000_MAIN_1);
        configurer.state(States.FINAL);

        StateConfigurer<States, Events> subConfigurer = states.withStates().parent(States._0000_MAIN_1);
        subConfigurer.initial(States._0000_MAIN).end(States.LOCAL_FINAL);
        CocrdslcContext lctx = (CocrdslcContext) ctx;
        subConfigurer.state(States._0000_MAIN, buildAction(() -> {stateProcess._0000Main(lctx, ctrl);}), null);
        subConfigurer.state(States.ABEND_ROUTINE, buildAction(() -> {stateProcess.abendRoutine(lctx, ctrl);}), null);

        configureTransitions(transitions);
    }

    /**
     * Declare state machine transitions.
     * @param transitions the transitions configuration helper
     */
    private void configureTransitions(StateMachineTransitionConfigurer<States, Events> transitions) throws Exception {
        transitions.withLocal().source(States._0000_MAIN_1).target(States.ABEND_ROUTINE).event(Events.TO_ABEND_ROUTINE);
        transitions.withExternal().source(States.ABEND_ROUTINE).target(States.FINAL).event(Events.TO_FINAL);
    }
}

```

ステートマシンコントローラは、発生する可能性のあるさまざまな状態、およびそれらの間の遷移を定義します。これにより、特定のプログラムの従来の実行制御フローが再現されます。

ステートマシンを構築する際、コントローラはステートマシンパッケージ内の関連するサービスクラスで定義されているメソッドを参照します。以下で説明します。

```
subConfigurer.state(States._0000_MAIN, buildAction(() ->
 {stateProcess._0000Main(lctx, ctrl);}), null);
subConfigurer.state(States.ABEND_ROUTINE, buildAction(() ->
 {stateProcess.abendRoutine(lctx, ctrl);}), null);
```

- *Program*ProcedureDivisionStateMachineService: このサービスクラスは、前述のように、ステートマシンコントローラが作成するステートマシンにバインドする必要のある、一部のビジネスロジックを表します。

  このクラスのメソッド内のコードは、ステートマシンコントローラで定義された以下のイベントを使用します。

CocrdslcProcedureDivisionStateMachineService.java

```
59      /**
60       * State process operation _0000Main.
61       *
62       * @param ctx
63       * @param ctrl
64       */
65      void _0000Main(CocrdslcContext ctx, ExecutionController ctrl) {
66          ctx.getDfheiblk().bind(ArgUtils.get(ctx, 0));
67          ctx.getDfhcommarea().bind(ArgUtils.get(ctx, 1));
68
69          /*
70          *****************************************************************
71          Program:     COCRDSLC.CBL                                      *
72          Layer:       Business logic                                    *
73          Function:    Accept and process credit card detail request     *
74          *****************************************************************
75          Copyright Amazon.com, Inc. or its affiliates.
76          All Rights Reserved.
77          Licensed under the Apache License, Version 2.0 (the \"License\").
78          You may not use this file except in compliance with the License.
79          You may obtain a copy of the License at
80          http://www.apache.org/licenses/LICENSE-2.0
81          Unless required by applicable law or agreed to in writing,
82          software distributed under the License is distributed on an
83          \"AS IS\" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
84          either express or implied. See the License for the specific
85          language governing permissions and limitations under the License
86          *****************************************************************
87          Ver: CardDemo_v1.0-15-g27d6c6f-68 Date: 2022-07-19 23:16:00 CDT */
88          instanceStateMachineController.registerSignalHandler(Events.TO_ABEND_ROUTINE, "!ABEND");
89          HandleAbendBuilder.newInstance(ctx.getDfheiblk(), ctx).execute().handleException();
90          ctx.getCcWorkAreas().getCcWorkAreaReference().getField().initialize();
91          ctx.getWsMiscStorage().getField().initialize();
92          DataUtils.initialize(ctx.getWsCommarea().getWsCommareaReference());
93
```

CocrdslcProcedureDivisionStateMachineService.java

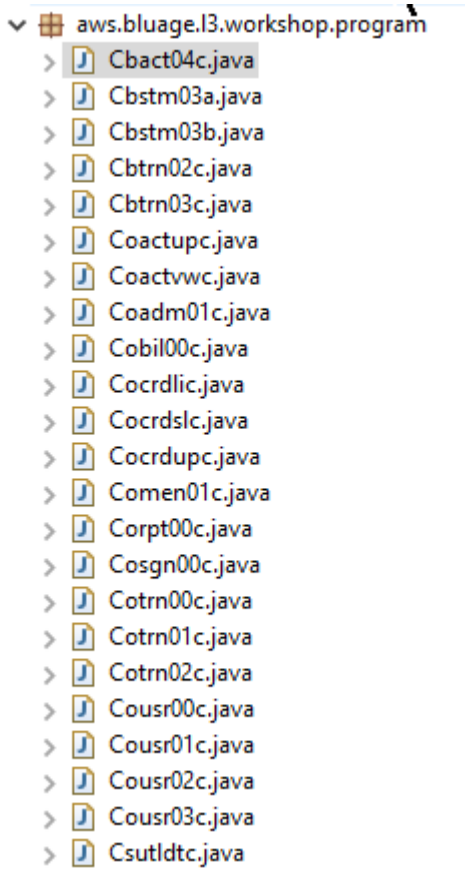
```
221      *
222      * @param ctx
223      * @param ctrl
224      */
225     void abendRoutine(CocrdslcContext ctx, ExecutionController ctrl) {
226         if (DataUtils.isLowValue(ctx.getAbendData().getAbendMsgReference())) {
227             ctx.getAbendData().setAbendMsg("UNEXPECTED ABEND OCCURRED.");
228         }
229         ctx.getAbendData().setAbendCulprit(ctx.getWsLiterals().getLitThispgm());
230         SendTextBuilder.newInstance(ctx.getDfheiblk(), ctx)
231         .withData(ctx.getAbendData())
232         .withLength(134)
233         .execute();
234         HandleAbendBuilder.newInstance(ctx.getDfheiblk(), ctx).cancel().execute().handleException();
235         AbendBuilder.newInstance(ctx.getDfheiblk(), ctx).withAbendCode("9999").execute().handleException();
236
237         /*
238         Ver: CardDemo_v1.0-15-g27d6c6f-68 Date: 2022-07-19 23:12:33 CDT */
239         instanceStateMachineController.sendEvent(Events.TO_FINAL);
240
241     }
242 }
243
```

また、ステートマシンサービスは、以前説明した以下のプロセスサービス実装を呼び出します。

CocrdslcProcedureDivisionStateMachineService.java

```
166
167             /*
168             ************************************************************
169             COMING FROM CREDIT CARD LIST SCREEN
170             SELECTION CRITERIA ALREADY VALIDATED
171             ************************************************************ */
172         } else if (ctx.getCarddemoCommarea().isCdemoPgmEnter() && DataUtils.compare(ctx.getCarddemoCommarea().getCdemoFromProgramReference(), ctx.getWsLiterals().getLitCclistpgmReference()) == 0) {
173             ctx.getWsMiscStorage().setInputOk(true);
174             ctx.getCcWorkAreas().setCcAcctIdN(ctx.getCarddemoCommarea().getCdemoAcctId());
175             ctx.getCcWorkAreas().setCcCardNumN(ctx.getCarddemoCommarea().getCdemoCardNum());
176             instanceCocrdslcProcess.getcardByacctcard(ctx, ctrl);
177             instanceCocrdslcProcess.sendMap(ctx, ctrl);
178             instanceCocrdslcProcess.commonReturn(ctx, ctrl);
179         } else if (ctx.getCarddemoCommarea().isCdemoPgmEnter()) {
180
181             /*
182             ************************************************************
183             COMING FROM SOME OTHER CONTEXT
184             SELECTION CRITERIA TO BE GATHERED
185             ************************************************************ */
186             instanceCocrdslcProcess.sendMap(ctx, ctrl);
187             instanceCocrdslcProcess.commonReturn(ctx, ctrl);
188         } else if (ctx.getCarddemoCommarea().isCdemoPgmReenter()) {
189             instanceCocrdslcProcess.processInputs(ctx, ctrl);
190             if (ctx.getWsMiscStorage().isInputError()) {
191                 instanceCocrdslcProcess.sendMap(ctx, ctrl);
192                 instanceCocrdslcProcess.commonReturn(ctx, ctrl);
193             } else {
194                 instanceCocrdslcProcess.getcardByacctcard(ctx, ctrl);
195                 instanceCocrdslcProcess.sendMap(ctx, ctrl);
196                 instanceCocrdslcProcess.commonReturn(ctx, ctrl);
197             }
198         } else {
199             ctx.getAbendData().setAbendCulprit(ctx.getWsLiterals().getLitThispgm());
200             ctx.getAbendData().setAbendCode("0001");
201             DataUtils.setToBlank(ctx.getAbendData().getAbendReasonReference());
202             ctx.getWsMiscStorage().setWsReturnMsg("UNEXPECTED DATA SCENARIO");
203             instanceCocrdslcProcess.sendPlainText(ctx, ctrl);
204         }
205
206         /*
207         If we had an error setup error message that slipped through
208         Display and return */
209         if (ctx.getWsMiscStorage().isInputError()) {
210             ctx.getCcWorkAreas().setCcardErrorMsg(ctx.getWsMiscStorage().getWsReturnMsg());
211             instanceCocrdslcProcess.sendMap(ctx, ctrl);
212             instanceCocrdslcProcess.commonReturn(ctx, ctrl);
213         }
214         instanceCocrdslcProcess.commonReturn(ctx, ctrl);
```

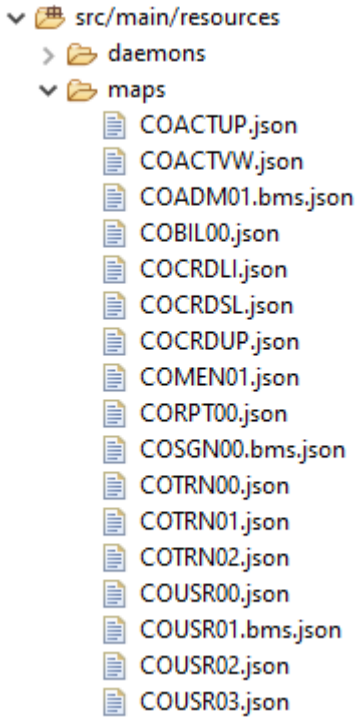
それに加えて、*base package*.program という名前のパッケージは、プログラムごとにプログラムのエントリポイントとなる 1 つのクラスを集めるという、重要な役割を果たします (これについては後で詳しく説明します)。各クラスはプログラムのエントリポイントのマーカーである Program インターフェイスを実装しています。



その他のアーティファクト

- BMS MAP コンパニオン

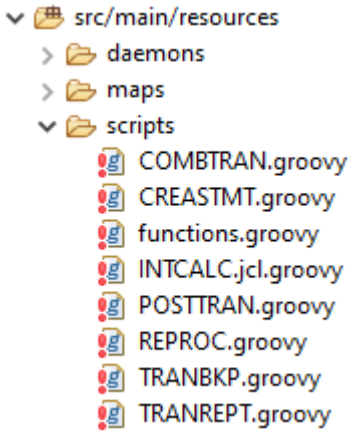
  プログラム関連のアーティファクトに加えて、サービスプロジェクトにはさまざまな目的のアーティファクトを含めることができます。CICS オンラインアプリケーションをモダナイズする場合、モダナイゼーションプロセスでは json ファイルが生成され、/src/main/resources フォルダの map フォルダに格納されます。



Blu Age ランタイムは、これらの json ファイルを利用して、SEND MAP ステートメントで使用されるレコードを画面フィールドとバインドします。

- Groovy スクリプト

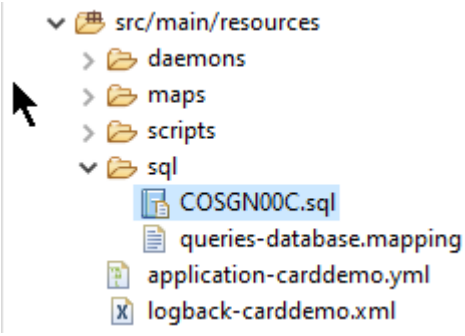
  レガシーアプリケーションに JCL スクリプトがあった場合、それらは groovy スクリプトとしてモダナイズされ、/src/main/resources/scripts フォルダに保存されます (この特定の場所については後で詳しく説明します)。



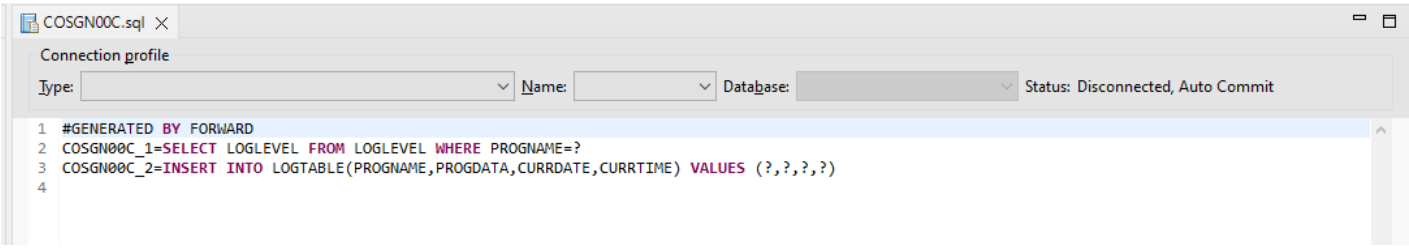
  これらのスクリプトは、バッチジョブ (専用の、インタラクティブではない、CPU 負荷の高いデータ処理ワークロード) を起動するために使用されます。

- SQL ファイル

レガシーアプリケーションが SQL クエリを使用していた場合、対応するモダナイズ SQL クエリは、program.sql という命名パターンを持つ専用のプロパティファイルにまとめられています。ここで、program は、そのクエリを使用するプログラムの名前です。



これらの sql ファイルの内容は (key=query) エントリの集合で、各クエリは固有のキーに関連付けられており、モダナイズプログラムはこれを使用して特定のクエリを実行します。

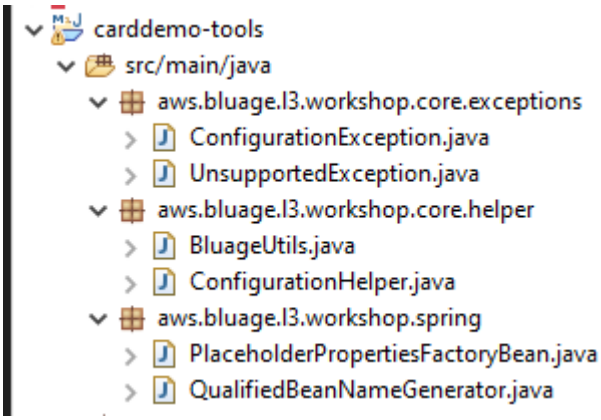


例えば、COSGN00C プログラムは「COSGN00C_1」(sql ファイルの最初のエントリ) というキーを使用してクエリを実行します。

```
327     /**
328      * Process operation getProgramLogLevel.
329      *
330      * ****************************************************************
331      *                       GET PROGRAM LOG LEVEL
332      * ****************************************************************
333      *
334      * @param ctx
335      * @param ctrl
336      */
337     @Override
338     public void getProgramLogLevel(final Cosgn00cContext ctx, final ExecutionController ctrl) {
339         SQLExecutorBuilder.newInstance(ctrl, ctx, ctx.getSqlca())
340             .mapInParameter(SQLParameterBuilder.newInstance(ctx.getLogData().getLogProgramName()).type(JDBCType.CHAR))
341             .mapOutParameter(SQLParameterBuilder.newInstance(ctx.getLogData().getLogProgramLevelReference()))
342             .execute("COSGN00C_1");
343         if (ctx.getSqlca().getSqlcode() == 100) {
344             ctx.getLogData().setLogProgramLevel("N");
345         }
346     }
347
```

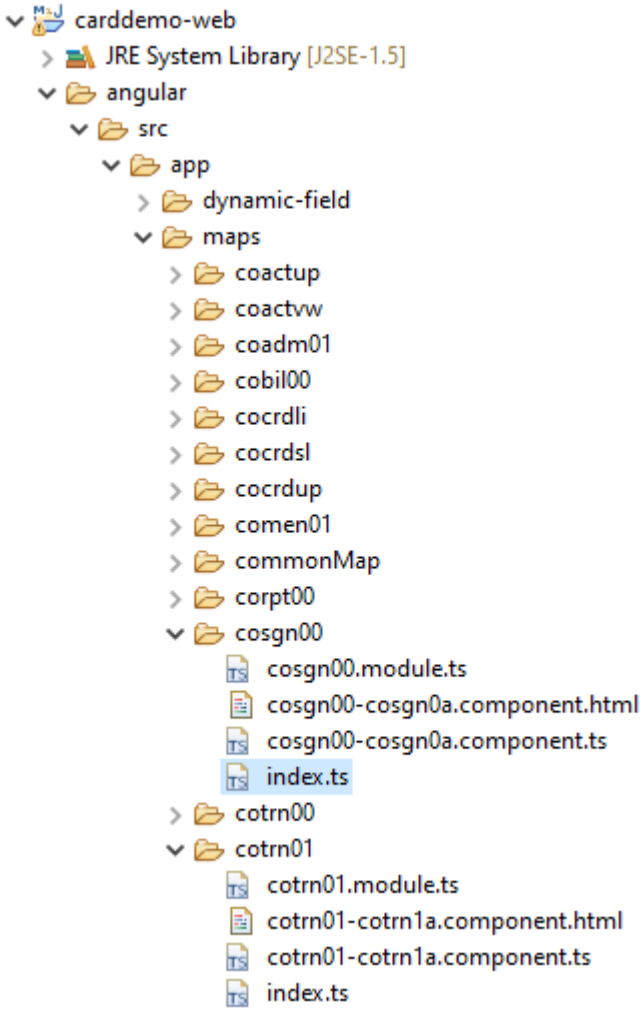
ユーティリティプロジェクトの内容

名前が「-tools」で終わるユーティリティプロジェクトには、他のすべてのプロジェクトで使用できるテクニカルユーティリティのセットが含まれています。



ウェブプロジェクトの内容

ウェブプロジェクトはレガシー UI 要素をモダナイズする場合にのみ存在します。モダナイズされたアプリケーションのフロントエンドの構築に使用されるモダン UI 要素は Angular に基づいています。モダナイゼーションアーティファクトを示すために使用されるアプリケーション例は、メインフレーム上で実行される COBOL/CICS アプリケーションです。CICS システムは MAP を使用して UI 画面を表します。すべてのマップにおいて、対応するモダン要素は Typescript ファイルを伴った HTML ファイルになります。



ウェブプロジェクトはアプリケーションのフロントエンド部分のみを処理します。ユーティリティプロジェクトとエンティティプロジェクトに依存するサービスプロジェクトは、バックエンドサービスを提供します。フロントエンドとバックエンド間のリンクは、標準の AWS Blu Age ランタイムディストリビューションの一部である Gapwalk-Application という名前のウェブアプリケーションを介して行われます。

プログラムの実行と呼び出し

レガシーシステムでは、プログラムはスタンドアロンの実行ファイルとしてコンパイルされ、COBOL CALL ステートメントなどの CALL メカニズムを通じて自らを呼び出し、必要に応じて引数を渡すことができます。モダナイズされたアプリケーションでも機能は同じですが、関連するアーティファクトの性質が従来のものとは異なるため、別のアプローチを使用します。

モダナイズされた側では、プログラムのエントリポイントは `Program` インターフェイスを実装する特定のクラス、Spring コンポーネント (@Component) であり、サービスプロジェクト内の *`base package`*`.program` という名前のパッケージにあります。

プログラムの登録

モダナイズされたアプリケーションをホストする Tomcat サーバーが起動するたびに、サービス Springboot アプリケーションも起動し、これによりプログラムの登録が開始されます。`ProgramRegistry` という名前の専用レジストリにはプログラムエントリが格納され、各プログラムは既知のプログラム ID につき 1 つのエントリを使用して登録されます。つまり、プログラムが複数の異なる識別子で認識されている場合、レジストリには識別子と同じ数のエントリが含まれます。

特定のプログラムの登録は、 getProgramIdentifiers() メソッドによって返される識別子のコレクションによって異なります。

Cbact04c.java

```java
package aws.bluage.l3.workshop.program;

import aws.bluage.l3.workshop.SpringBootLauncher;

/**
 * Reference the spring application of program CBACT04C.
 *  Provides an access to the contained program for the run unit.
 */
@Component
@Import({
    aws.bluage.l3.workshop.cbact04c.business.context.Cbact04cConfiguration.class,
    aws.bluage.l3.workshop.cbact04c.business.context.Cbact04cContext.class,
    aws.bluage.l3.workshop.cbact04c.service.impl.Cbact04cProcessImpl.class
})
public class Cbact04c implements Program {
    /**
     * Unique identifiers for the contained program.
     */
    private static final Set<String> programIdentifiers = Collections.unmodifiableSet(Stream.of("CBACT04C").collect(Collectors.toSet()));

    /**
     * Main program identifier for the contained program.
     */
    private static final String programIdentifier = "CBACT04C";
    @Autowired
    PlatformTransactionManager transactionManager;

    @Autowired
    Map<String, DataSource> datasources;
    @Autowired
    BeanFactory beanFactory;
    /**
     * {@inheritDoc}
     */
    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        return SpringBootLauncher.getCac();
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void updateExecutionContext(ExecutionContext executionContext) {
        executionContext.setDatasources(datasources);
        executionContext.setDatabaseSupport(ExecutionContext.DatabaseSupport.POSTGRE);
        executionContext.setSqlcaVersion(ExecutionContext.SqlcaVersion.getEnum("ansi-comp5"));
        executionContext.setTransactionManager(transactionManager);
        executionContext.setUseSQLDateNewParadigm(true);
        executionContext.setUseSQLTrimStringType(false);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public Set<String> getProgramIdentifiers() {
        return programIdentifiers;
    }
```

この例では、プログラムは 'CBACT04C' という名前で 1 回登録されています (programIdentifiers コレクションの内容を見てください)。tomcat のログには、すべてのプログラムの登録が表示されます。プログラム登録は宣言されたプログラム ID にのみ依存し、プログラムクラス名自体には依存しません (ただし、通常、プログラム ID とプログラムクラス名は一致しています)。

Blu Age AWS ランタイムディストリビューションの一部であるさまざまなユーティリティ AWS Blu Age ウェブアプリケーションによって提供されるユーティリティプログラムにも同じ登録メカニズムが適用されます。例えば、Gapwalk-Utility-Pgm ウェブアプリは z/OS システムユーティリティ (IDCAMS、ICEGENER、SORT など) と同等の機能を備えており、モダナイズされたプログラムや

スクリプトから呼び出すことができます。Tomcat のスタートアップ時に登録された利用可能なユーティリティプログラムはすべて Tomcat ログに記録されます。

スクリプトとデーモンの登録

/src/main/resources/scripts フォルダ階層にある groovy スクリプトでも、同様の登録プロセスが Tomcat のスタートアップ時に行われます。scripts フォルダ階層がトラバースされ、見つかったすべての groovy スクリプト (特別な functions.groovy 予約スクリプトを除く) が、そのショートネーム (スクリプトファイル名の最初のドット文字の前にある部分) を取得用キーとして使用して `ScriptRegistry` に登録されます。

> **Note**
>
> - 複数のスクリプトのファイル名を使用して同じ登録キーが生成される場合、最後のものだけが登録され、そのキーに対して以前に登録されたものは上書きされます。
> - 登録メカニズムによって階層は平坦になり、予期しない上書きが発生する可能性があるため、サブフォルダを使用する場合は上記の点を考慮してご注意ください。階層は登録プロセスでは考慮されません。通常 /scripts/A/myscript.groovy と /scripts/B/myscript.groovy の順に生成されると /scripts/B/myscript.groovy により /scripts/A/myscript.groovy が上書きされます。

/src/main/resources/daemons フォルダにある groovy スクリプトの扱いは少し異なります。これらは通常のスクリプトとして登録されていますが、それに加えて、Tomcat のスタートアップ時に一度だけ非同期的に直接起動されます。

スクリプトが `ScriptRegistry` に登録されると、Gapwalk-Application が公開する専用のエンドポイントを使用して REST を呼び出すとスクリプトを起動できます。詳細については、対応するドキュメントをご参照ください。

プログラム呼び出しプログラム

各プログラムは他のプログラムをサブプログラムとして呼び出し、パラメータを渡すことができます。プログラムは、`ExecutionController` インターフェイスの実装を使用し (ほとんどの場合、これは `ExecutionControllerImpl` インスタンスです)、`CallBuilder` という fluent API メカニズムと連動してプログラム呼び出し引数を構築します。

すべてのプログラムのメソッドは `RuntimeContext` と `ExecutionController` の両方をメソッド引数と解釈するため、`ExecutionController` は常に他のプログラムを呼び出すことができます。

例えば、CBST03A プログラムが CBST03B プログラムをサブプログラムとして呼び出し、パラメータを渡す方法を示す次の図を参照してください。

Cbstm03aProcessImpl.java

```
67      /**
68       * Process operation xreffileGetNext.
69       *
70       * ----------------------------------------------------------------*
71       *
72       * @param ctx
73       * @param ctrl
74       */
75      @Override
76      public void xreffileGetNext(final Cbstm03aContext ctx, final ExecutionController ctrl) {
77          ctx.getWsM03bArea().setWsM03bDd("XREFFILE");
78          ctx.getWsM03bArea().setM03bRead(true);
79          DataUtils.setToZeroes(ctx.getWsM03bArea().getWsM03bRcReference());
80          DataUtils.setToBlank(ctx.getWsM03bArea().getWsM03bFldtReference());
81          ctrl.callSubProgram("CBSTM03B", CallBuilder.newInstance()
82              .byReference(ctx.getWsM03bArea())
83              .getArguments(), ctx);
84          if (DataUtils.compare(ctx.getWsM03bArea().getWsM03bRcReference(), "00") == 0) {
85
86              /*
87              Do nothing */
88          } else if (DataUtils.compare(ctx.getWsM03bArea().getWsM03bRcReference(), "10") == 0) {
89              ctx.getMiscVariables().setEndOfFile("Y");
90          } else {
91              if (LOGGER.isInfoEnabled()) LOGGER.info("ERROR READING XREFFILE");
92              if (LOGGER.isInfoEnabled()) LOGGER.info("{}{}" , "RETURN CODE: " , ctx.getWsM03bArea().getWsM03bRc());
93              abendProgram(ctx, ctrl);
94          }
95          ctx.getCardXrefRecord().setBytes(ctx.getWsM03bArea().getWsM03bFldtReference().getBytes());
96      }
97
```

- `ExecutionController.callSubProgram` の最初の引数は、呼び出すプログラムの識別子です (つまり、プログラム登録に使用される識別子のうちの 1 つです。上の段落を参照してください)。
- 2 番目の引数は、`CallBuilder` をビルドした結果で、呼び出し元から呼び出し先に渡されたデータに対応する `Record` の配列です。
- 3 番目の、最後の引数は呼び出し元の `RuntimeContext` インスタンスです。

3 つの引数はすべて必須で NULL にできませんが、2 番目の引数は空の配列でも構いません。

呼び出し先が渡されたパラメータを処理できるのは、元々そのように設計されていた場合のみです。レガシー COBOL プログラムでは、LINKAGE 要素を利用するため、手続き部に LINKAGE 節と USING 句が必要です。

例えば、対応する CBSTM03B.CBL COBOL ソースファイルを参照してください。

github.com/aws-samples/aws-mainframe-modernization-carddemo/blob/main/app/cbl/CBSTM03B.CBL

```
 98
 99         LINKAGE SECTION.
100         01  LK-M03B-AREA.
101             05  LK-M03B-DD          PIC X(08).
102             05  LK-M03B-OPER        PIC X(01).
103               88  M03B-OPEN       VALUE 'O'.
104               88  M03B-CLOSE      VALUE 'C'.
105               88  M03B-READ       VALUE 'R'.
106               88  M03B-READ-K     VALUE 'K'.
107               88  M03B-WRITE      VALUE 'W'.
108               88  M03B-REWRITE    VALUE 'Z'.
109             05  LK-M03B-RC          PIC X(02).
110             05  LK-M03B-KEY         PIC X(25).
111             05  LK-M03B-KEY-LN      PIC S9(4).
112             05  LK-M03B-FLDT        PIC X(1000).
113
114         PROCEDURE DIVISION USING LK-M03B-AREA.
115
```

そのため、CBSTM03B プログラムはパラメータ (サイズ 1 の配列) として単一 `Record` を取ります。これこそが、`CallBuilder` が byReference() メソッドと getArguments() メソッド連鎖を使って構築しているものです。

`CallBuilder` fluent API クラスには、呼び出し先に渡す引数の配列を投入するためのメソッドがいくつか用意されています。

- asPointer (RecordAdaptable) : ポインタの種類をリファレンスとして引数を追加します。ポインタは対象となるデータ構造のアドレスを表します。
- byReference (RecordAdaptable): 参照による引数を追加します。呼び出し側は呼び出し先が実行する変更を確認します。
- byReference (RecordAdaptable...): 前のメソッドの varargs バリアント。
- byValue(Object): `Record` に変換された引数を、値ごとに追加します。呼び出し側は呼び出し先が実行する変更を確認しません。
- byValue (RecordAdaptable): 前のメソッドと同じですが、引数は として直接使用できますRecordAdaptable。
- byValueWithBounds(Object, int, int): 引数を追加し、 に変換してRecord、指定された境界で定義されたバイト配列部分を値別に抽出します。

最後に、getArguments メソッドは追加された引数をすべて収集し、Record の配列として返します。

> **Note**
>
> 呼び出し元は、引数の配列が必要なサイズであること、項目が適切に順序付けられており、リンケージ要素で想定されるレイアウトとメモリレイアウトに関し互換性があることを確認する責任があります。

スクリプト呼び出しプログラム

登録済みプログラムを groovy スクリプトから呼び出すには、その MainProgramRunner インターフェイスを実装するクラスインスタンスを使用する必要があります。通常、このようなインスタンスの取得は Spring ApplicationContext の使用を通じて実現されます。

REPROC.groovy

```groovy
// Import
import com.netfective.bluage.gapwalk.rt.provider.ScriptRegistry
import com.netfective.bluage.gapwalk.rt.call.MainProgramRunner
import com.netfective.bluage.gapwalk.io.support.FileConfigurationUtils
import com.netfective.bluage.gapwalk.rt.job.support.DefaultJobContext
import com.netfective.bluage.gapwalk.rt.utils.GroovyUtils
import com.netfective.bluage.gapwalk.rt.io.support.FileConfiguration
import com.netfective.bluage.gapwalk.rt.shared.AbendException
import com.netfective.bluage.gapwalk.rt.call.exception.GroovyExecutionException
// Variables
mpr = applicationContext.getBean("com.netfective.bluage.gapwalk.rt.call.ExecutionController", MainProgramRunner.class)
TreeMap mapTransfo = [:]
```

MainProgramRunner インターフェイスが使用可能になったら、runProgram メソッドを使用してプログラムを呼び出し、ターゲットプログラムの識別子をパラメータとして渡します。

REPROC.groovy

```
//*******************************************************************
//*                              STEPS                              *
//*******************************************************************
// STEP PRC001 - PGM - IDCAMS*****************************************
def stepPRC001(Object shell, Map params, Map programResults){
    shell.with {
        if (checkValidProgramResults(programResults)) {
            return execStep("PRC001", "IDCAMS", programResults, {
                mpr
                    .withFileConfigurations(new FileConfigurationUtils()
                        .systemOut("SYSPRINT")
                        .output("*")
                        .build()
                        .bluesam("FILEIN")
                        .dataset("NULLFILE")
                        .disposition("SHR")
                        .build()
                        .bluesam("FILEOUT")
                        .dataset("NULLFILE")
                        .disposition("SHR")
                        .build()
                        .fileSystem("SYSIN")
                        .path("&CNTLLIB(REPROCT)")
                        .disposition("SHR")
                        .build()
                        .getFileConfigurations(fcmap))
                    .withParameters(params)
                    .runProgram("IDCAMS")
                })
        }
    }
}
```

前の例では、ジョブステップが IDCAMS (ファイル処理ユーティリティプログラム) を呼び出し、実際のデータセット定義とその論理識別子の間のマッピングを提供します。

データセットを扱う場合、レガシープログラムではたいてい論理名を使用してデータセットを識別します。プログラムをスクリプトから呼び出す場合、スクリプトは論理名と実際の物理データセットとマッピングする必要があります。これらのデータセットは、ファイルシステムや Blusam ストレージに格納されている場合もあれば、インラインストリーム、複数のデータセットの連結、GDG の生成によって定義される場合もあります。

withFileConfiguration メソッドを使用して、データセットの論理マップと物理マップを構築し、呼び出したプログラムで使用できるようにします。

独自のプログラムの作成

スクリプト、またはその他のモダナイズされたプログラムを呼び出すために、独自のプログラムを作成するのはよくある作業です。通常、モダナイゼーションプロジェクトでは、実行可能なレガシープログラムがモダナイゼーションプロセスでサポートされていない言語で書かれていたり、ソースが失われた場合 (これは発生する場合があります)、またはプログラムがソースを利用できないユーティリティだったりする場合に、独自のプログラムを作成します。

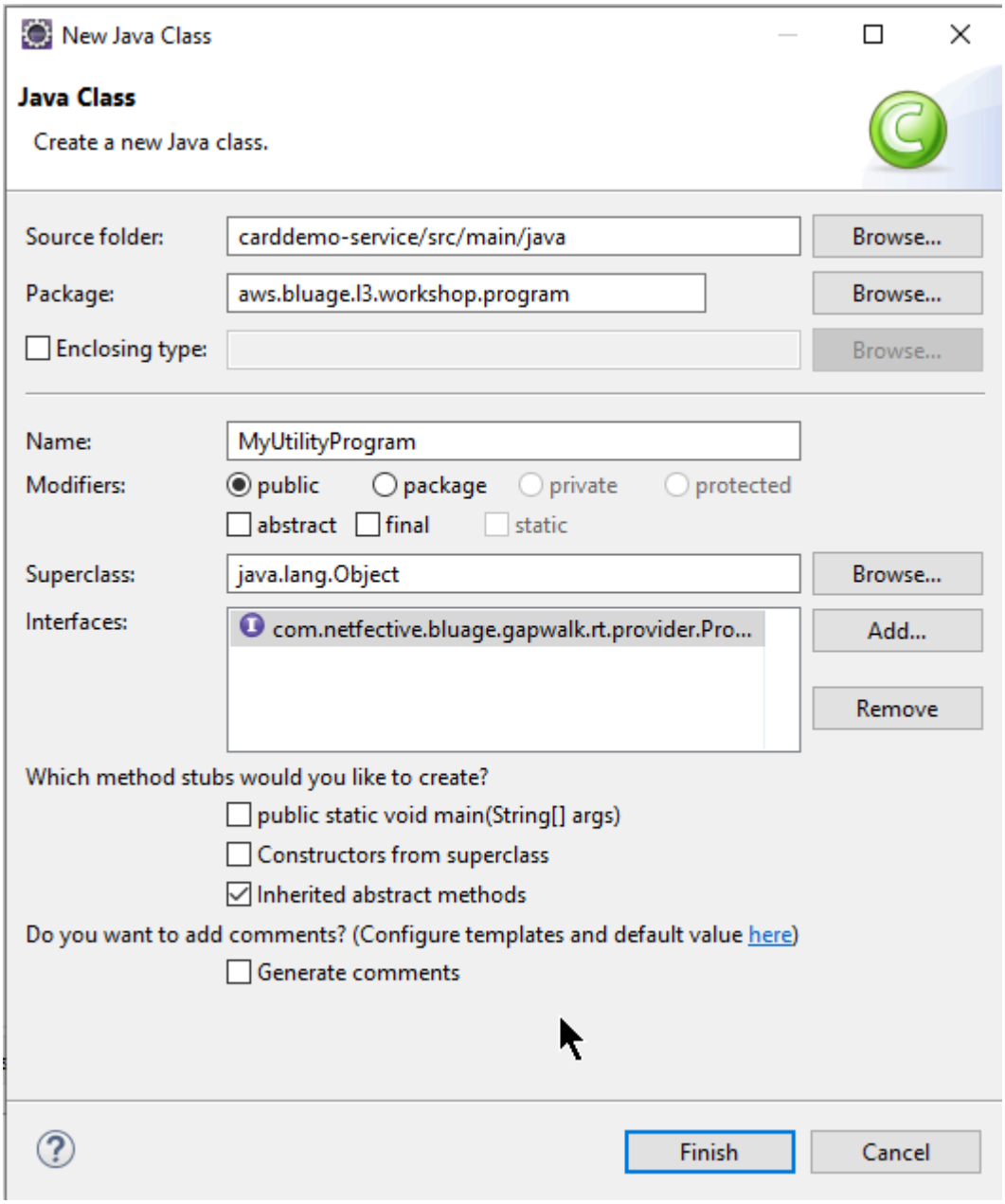
その場合、足りないプログラムを java で、自分で書かなければならない場合があります (プログラムの期待される動作がどうあるべきか、プログラムの引数のメモリレイアウト (ある場合) などについて十分な知識があることを前提としています)。java プログラムは、他のプログラムやスクリプトで実行できるように、このドキュメントで説明されているプログラムの仕組みに準拠している必要があります。

プログラムが使用に適していることを確認するには、次の 2 つの必須ステップを完了する必要があります。

- `Program` インターフェイスを正しく実装するクラスを作成して、登録および呼び出すことができるようにしてください。
- 他のプログラムやスクリプトから見えるようにするため、プログラムを正しく登録してください。

プログラム実装の記述

IDE を使用して `Program` インターフェイスを実装する新しい java クラスを作成します。



以下の画像は、実装する必須メソッドをすべて作成する Eclipse IDE を示しています。

MyUtilityProgram.java

```java
package aws.bluage.l3.workshop.program;

import java.util.Set;

public class MyUtilityProgram implements Program {

    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Set<String> getProgramIdentifiers() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Context getContext() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void run(ExecutionController ctrl) {
        // TODO Auto-generated method stub

    }

}
```

Spring との統合

まず、クラスを Spring コンポーネントとして宣言する必要があります。クラスに以下の `@Component` アノテーションを付けます。

```java
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.stereotype.Component;

import com.netfective.bluage.gapwalk.rt.call.ExecutionController;
import com.netfective.bluage.gapwalk.rt.context.Context;
import com.netfective.bluage.gapwalk.rt.provider.Program;

import aws.bluage.l3.workshop.SpringBootLauncher;

@Component
public class MyUtilityProgram implements Program {
```

次に、必要なメソッドを正しく実装します。このサンプルのコンテキストでは、モダナイズされたプログラムがすべて含まれているパッケージに `MyUtilityProgram` を追加しました。この配置によ

り、プログラムは既存の Springboot アプリケーションを使用して、 getSpringApplication メソッドの実装ConfigurableApplicationContextに必要な を提供できます。

```
public class MyUtilityProgram implements Program {

    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        return SpringBootLauncher.getCac();
    }
```

独自のプログラムでは別の場所を選択することもできます。例えば、特定のプログラムを別の専用サービスプロジェクトに配置する場合があります。特定のサービスプロジェクトに独自の Springboot アプリケーションがあることを確認します。これにより、 ApplicationContext (である必要がありますConfigurableApplicationContext) を取得できます。

プログラムに識別子を与える

他のプログラムやスクリプトから呼び出せるようにするには、そのプログラムに少なくとも 1 つの識別子を与える必要があります。その識別子は、システム内の既存の登録済みプログラムと衝突することはできません。識別子は既存のレガシープログラムの代わりとなるものが必要な場合に選択する可能性があります。その場合は、レガシープログラム全体で見られる呼び出し発生回数に応じて要求される識別子を使用する必要があります。レガシーシステムでは、ほとんどのプログラム識別子は 8 文字です。

その 1 つの方法はプログラム内で変更できない識別子のセットを作成することです。次の例は、「MYUTILPG」を単一識別子として選択する方法を示しています。

```
@Component
public class MyUtilityProgram implements Program {

    /**
     * Unique identifiers for the contained program.
     */
    private static final Set<String> programIdentifiers = Collections.unmodifiableSet(Stream.of("MYUTILPG").collect(Collectors.toSet()));


    public ConfigurableApplicationContext getSpringApplication() {

    @Override
    public Set<String> getProgramIdentifiers() {
        return programIdentifiers;
    }
```

プログラムをコンテキストに関連付ける

プログラムにはコンパニオン RuntimeContext インスタンスが必要です。モダナイズされたプログラムの場合、 AWS Blu Age はレガシープログラムの一部であるデータ構造を使用してコンパニオンコンテキストを自動的に生成します。

独自のプログラムを作成する場合は、コンパニオンコンテキストも作成する必要があります。

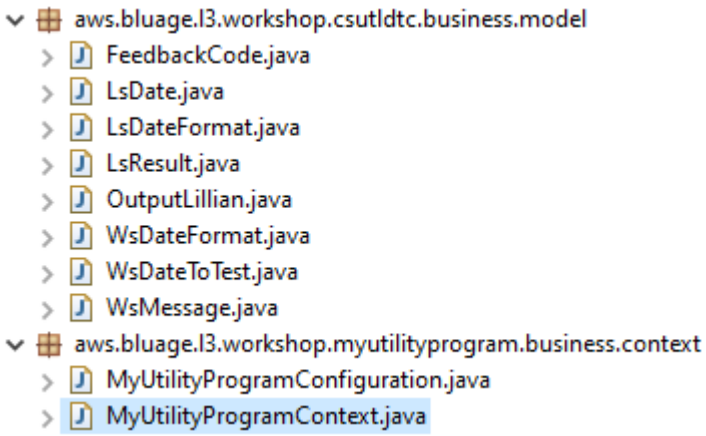
プログラム関連クラス を参照すると、プログラムには少なくとも以下の 2 つのコンパニオンクラスが必要であることがわかります。

- 設定クラス。
- 設定を使用するコンテキストクラス。

ユーティリティプログラムが追加のデータ構造を使用する場合は、そのデータ構造も記述してコンテキストで使用する必要があります。

これらのクラスは、コンテキストのコンポーネントと構成が Spring フレームワークによって処理されるようにするために、アプリケーションのスタートアップ時にスキャンされるパッケージ階層に含める必要があります。

エンティティプロジェクトで新しく作成した *base package*`.myutilityprogram.business.context` パッケージに、最小限の設定とコンテキストを記述してみましょう。



設定内容は次のとおりです。近くにある他の (モダナイズされた) プログラムと類似した設定ビルドを使っています。おそらく、特定のニーズに合わせてこれをカスタマイズする必要があるでしょう。

MyUtilityProgramConfiguration.java

```java
package aws.bluage.l3.workshop.myutilityprogram.business.context;

import java.nio.charset.Charset;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Lazy;

import com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration;
import com.netfective.bluage.gapwalk.datasimplifier.configuration.ConfigurationBuilder;

/**
 * Creates Datasimplifier configuration for the MyUtilityProgram context.
 */
@org.springframework.context.annotation.Configuration
@Lazy
public class MyUtilityProgramConfiguration {

    @Bean(name = "MyUtilityProgramContextConfiguration")
    public Configuration configuration() {
        return new ConfigurationBuilder()
                .encoding(Charset.forName("CP1047"))
                .humanReadableEncoding(Charset.forName("ISO-8859-15"))
                .initDefaultByte(0)
                .build();
    }
}

```

注記：

- 一般的な命名規則はProgramName設定です。
- @org.springframework.context.annotation.Configuration アノテーションと @Lazy アノテーションを使用する必要があります。
- 通常、ビーコン名はProgramNameContextConfiguration 規則に従いますが、必須ではありません。プロジェクト全体で bean 名が競合しないようにしてください。
- 実装する単一メソッドは `Configuration` オブジェクトを返す必要があります。オブジェクトの構築には `ConfigurationBuilder` fluent API が便利です。

関連するコンテキスト:

MyUtilityProgramContext.java

```java

import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

import com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration;
import com.netfective.bluage.gapwalk.rt.context.RuntimeContext;

@Component("aws.bluage.l3.workshop.mytuilityprogram.business.context.MyUtilityProgramContext")
@Lazy
@Scope("prototype")
public class MyUtilityProgramContext extends RuntimeContext{

    protected MyUtilityProgramContext(@Qualifier("MyUtilityProgramContextConfiguration") Configuration configuration) {
        super(configuration);
    }

    @Override
    public void cleanUp() {
        // TODO implement clean-up of associated data structures if any
    }

    @Override
    protected void doReset() {
        // TODO implement reset of associated data structures if any
    }

}

```

メモ

- コンテキストクラスは、既存の Context インターフェイス実装 (JICS 固有の項目で拡張された RuntimeContext である、RuntimeContextまたはJicsRuntimeContext) を拡張する必要があります。
- 一般的な命名規則は Context ProgramNameです。
- これをプロトタイプコンポーネントとして宣言し、@Lazy アノテーションを使用する必要があります。
- コンストラクタは関連付けされた設定を参照し、@Qualifier アノテーションを使って適切な設定クラスをターゲットにします。
- ユーティリティプログラムがいくつかの追加のデータ構造を使用する場合、以下のようにする必要があります。
  - *base package*.business.model パッケージに記述および追加されます。
  - コンテキストで参照します。他の既存のコンテキストクラスを見て、データ構造クラスを参照し必要に応じてコンテキストメソッド (コンストラクタ、クリーンアップ、リセット) を適応させる方法を確認してください。

専用のコンテキストが用意できたので、新しいプログラムでそれを使用しましょう。

MyUtilityProgram.java

```
import aws.bluage.l3.workshop.SpringBootLauncher;

@Component
@Import({
    aws.bluage.l3.workshop.myutilityprogram.business.context.MyUtilityProgramConfiguration.class,
    aws.bluage.l3.workshop.myutilityprogram.business.context.MyUtilityProgramContext.class
})
public class MyUtilityProgram implements Program {

    @Autowired
    BeanFactory beanFactory;

    /**
     * Unique identifiers for the contained program.
     */
    private static final Set<String> programIdentifiers = Collections.unmodifiableSet(Stream.of("MYUTILPG").collect(Collectors.toSet()));

    private static final String programIdentifier = "MYUTILPG";

    @Override
    public ConfigurableApplicationContext getSpringApplication() {
        return SpringBootLauncher.getCac();
    }

    @Override
    public Set<String> getProgramIdentifiers() {
        return programIdentifiers;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public String getProgramMainIdentifier() {
        return programIdentifier;
    }


    @Override
    public Context getContext() {
        return ProgramContextStore.getOrCreate(
        getProgramMainIdentifier(),
        aws.bluage.l3.workshop.myutilityprogram.business.context.MyUtilityProgramContext.class,
        beanFactory);
    }
```

注記：

- getContext メソッドは、 `ProgramContextStore`クラスの メソッドと自動接続 Spring の getOrCreate メソッドへの委任を使用して、示されているとおりに厳密に実装する必要がありますBeanFactory。プログラムコンテキストを `ProgramContextStore` に保存するには単一のプログラム識別子が使用されます。この識別子は「プログラムのメイン識別子」と呼ばれます。
- コンパニオン設定クラスとコンテキストクラスは `@Import` Spring アノテーションを使用して参照する必要があります。

ビジネスロジックの実装

プログラムスケルトンが完成したら、新しいユーティリティプログラムのビジネスロジックを実装します。

これをプログラムの run メソッドで行います。このメソッドは、別のプログラムまたはスクリプトによってプログラムが呼び出されるたびに実行されます。

コーディングおめでとう!

プログラム登録の処理

最後に、新しいプログラムが正しく ProgramRegistry に登録されていることを確認します。既に他のプログラムが含まれるパッケージに新しいプログラムを追加した場合は、何もする必要はありません。新しいプログラムは、アプリケーションのスタートアップ時にすべての隣接プログラムに取り込まれ、登録されます。

プログラム用に別の場所を選択した場合は、Tomcat のスタートアップ時にそのプログラムが正しく登録されていることを確認する必要があります。その方法に関するいくつかの問題については、サービスプロジェクト (s) で生成された SpringbootLauncher クラスの初期化方法を参照してください (「」を参照) サービスプロジェクトの内容。

Tomcat のスタートアップログを確認してください。すべてのプログラム登録が記録されます。プログラムが正常に登録されていると、一致するログエントリが見つかります。

プログラムが正しく登録されたことを確認したら、ビジネスロジックのコーディングの繰り返しを始めることができます。

完全修飾名マッピング

このセクションには、モダナイズされたアプリケーションで使用する AWS Blu Age およびサードパーティーの完全修飾名マッピングのリストが含まれています。

AWS Blu Age 完全修飾名マッピング

短縮名	完全修飾名
CallBuilder	com.netfective.bluage.gapwalk.runtime.statements.CallBuilder
Configuration	com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration

短縮名	完全修飾名
ConfigurationBuilder	com.netfective.bluage.gapwalk.datasimplifier.configuration.ConfigurationBuilder
ExecutionController	com.netfective.bluage.gapwalk.rt.call.ExecutionController
ExecutionControllerImpl	com.netfective.bluage.gapwalk.rt.call.internal.ExecutionControllerImpl
File	com.netfective.bluage.gapwalk.rt.io.File
MainProgramRunner	com.netfective.bluage.gapwalk.rt.call.MainProgramRunner
Program	com.netfective.bluage.gapwalk.rt.provider.Program
ProgramContextStore	com.netfective.bluage.gapwalk.rt.context.ProgramContextStore
ProgramRegistry	com.netfective.bluage.gapwalk.rt.provider.ProgramRegistry
Record	com.netfective.bluage.gapwalk.datasimplifier.data.Record
RecordEntity	com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity
RuntimeContext	com.netfective.bluage.gapwalk.rt.context.RuntimeContext

短縮名	完全修飾名
SimpleStateMachineController	com.netfective.bluage.gapwalk.rt.statemachine.SimpleStateMachineController
StateMachineController	com.netfective.bluage.gapwalk.rt.statemachine.StateMachineController
StateMachineRunner	com.netfective.bluage.gapwalk.rt.statemachine.StateMachineRunner

サードパーティーの完全修飾名マッピング

短縮名	完全修飾名
@Autowired	org.springframework.beans.factory.annotation.Autowired
@Bean	org.springframework.context.annotation.Bean
BeanFactory	org.springframework.beans.factory.BeanFactory
@Component	org.springframework.stereotype.Component
ConfigurableApplicationContext	org.springframework.context.ConfigurableApplicationContext
@Import	org.springframework.context.annotation.Import
@Lazy	org.springframework.context.annotation.Lazy

Data Simplifier

メインフレームシステムおよびミッドレンジシステム (以下のトピックでは「レガシー」システムと呼びます) では、COBOL、PL/I、RPG などのプログラミング言語が使用され、メモリへの低レベルアクセスが可能になります。このアクセスは、ゾーン型、パック型、英数字などのネイティブ型からアクセスされるメモリレイアウトに重点が置かれており、グループや配列を通じて集約されることもあります。

1 つのプログラムには、型付きフィールドによる特定のメモリへのアクセスと、バイト (raw メモリ) への直接アクセスの両方が混在しています。例えば、COBOL プログラムは、引数を連続したバイトセット (リンク) として呼び出し元に渡したり、ファイルからのデータの読み取り/書き込み (レコード) を同じ方法で実行したりします。同時に、コピーブックに整理された型付きフィールドを使用して、このようなメモリ範囲を解釈します。

メモリへの raw アクセスと構造化アクセスの組み合わせ、正確なバイトレベルのメモリレイアウトへの依存、ゾーン型やパック型などのレガシータイプなどの機能は、そのまま Java プログラミング環境で簡単に利用することはできません。

レガシープログラムを Java にモダナイズするための AWS Blu Age ソリューションの一環として、Data Simplifier ライブラリは、モダナイズされた Java プログラムにそのようなコンストラクトを提供し、Java デベロッパー (ゲッター/セッター、バイト配列、クラスベース) に可能な限りよく知られている方法でそれらを公開します。それは、このようなプログラムから生成されたモダナイズされた Java コードの主要な依存関係です。

わかりやすくするため以下の説明のほとんどは COBOL コンストラクトに基づいていますが、ほとんどの概念は似ているため、PL1 と RPG のデータレイアウトのモダナイズには同じ API を使用できます。

トピック

メインクラス

読みやすくするために、このドキュメントでは AWS Blu Age API インターフェイスとクラスの Java 短縮名を使用します。詳細については、「説明した Java タイプの FQN」を参照してください。

低レベルメモリ表現

最も低いレベルでは、メモリ (高速でランダムにアクセス可能な連続したバイト範囲) が Record インターフェイスによって表現されます。このインターフェイスは基本的に固定サイズのバイト配列を抽出したものです。そのため、基になるバイトにアクセスしたり変更したりできるセッターやゲッターが用意されています。

構造化されたデータ表現

COBOL DATA DIVISION にある「01 データ項目」や「01 コピーブック」などの構造化データを表すには、RecordEntity クラスのサブクラスが使用されます。これらは通常、手動で記述されるのではなく、対応するレガシーコンストラクトの AWS Blu Age モダナイゼーションツールによって生成されます。その主な構造と API を知っておくと、モダナイズされたプログラムのコードでどのように使われているかを理解できるようになるため、今でも役に立ちます。COBOL の場合、コードは PROCEDURE DIVISION から生成される Java です。

生成コードは RecordEntity サブクラスを持つ各「01 データ項目」で表現されます。それを構成する各基本フィールドまたは集合は、ツリーとして編成されたプライベート Java フィールドとして表現されます (ルート以外の各項目には親があります)。

説明のため、COBOL データ項目の例と、それに対応する AWS Blu Age が生成したモダナイズするコードが続きます。

```
01 TST2.
 02 FILLER PIC X(4).
 02 F1     PIC 9(2) VALUE 42.
 02 FILLER PIC X.
 02        PIC 9(3) VALUE 123.
 02 F2     PIC X VALUE 'A'.
```

```
public class Tst2 extends RecordEntity {

    private final Group root = new Group(getData()).named("TST2");
    private final Filler filler = new Filler(root,new AlphanumericType(4));
    private final Elementary f1 = new Elementary(root,new ZonedType(2, 0, false),new
 BigDecimal("42")).named("F1");
    private final Filler filler1 = new Filler(root,new AlphanumericType(1));
    private final Filler filler2 = new Filler(root,new ZonedType(3, 0, false),new
 BigDecimal("123"));
    private final Elementary f2 = new Elementary(root,new
 AlphanumericType(1),"A").named("F2");
```

```
    /**
     * Instantiate a new Tst2 with a default record.
     * @param configuration the configuration
     */
    public Tst2(Configuration configuration) {
        super(configuration);
        setupRoot(root);
    }
    /**
     * Instantiate a new Tst2 bound to the provided record.
     * @param configuration the configuration
     * @param record the existing record to bind
     */
    public Tst2(Configuration configuration, RecordAdaptable record) {
        super(configuration);
        setupRoot(root, record);
    }

    /**
     * Gets the reference for attribute f1.
     * @return the f1 attribute reference
     */
    public ElementaryRangeReference getF1Reference() {
        return f1.getReference();
    }

    /* *
     * Getter for f1 attribute.
     * @return f1 attribute
     */
    public int getF1() {
        return f1.getValue();
    }

    /**
     * Setter for f1 attribute.
     * @param f1 the new value of f1
     */
    public void setF1(int f1) {
        this.f1.setValue(f1);
    }
    /**
     * Gets the reference for attribute f2.
```

```
     * @return the f2 attribute reference
     */
    public ElementaryRangeReference getF2Reference() {
        return f2.getReference();
    }

    /**
     * Getter for f2 attribute.
     * @return f2 attribute
     */
    public String getF2() {
        return f2.getValue();
    }

    /**
     * Setter for f2 attribute.
     * @param f2 the new value of f2
     */
    public void setF2(String f2) {
        this.f2.setValue(f2);
    }
}
```

基本フィールド

クラスのフィールド Elementary (名前が付けられていない場合は Filler) はレガシーデータ構造の「リーフ」を表します。これらは基礎となるバイトの連続したスパン (「範囲」) に関連付けられており、通常、それらのバイトの解釈と変更の方法 (バイト配列に対して値を「デコード」および「エンコード」する) を表す型 (パラメータ化されている場合もあります) があります。

すべての基本型は RangeType のサブクラスです。一般的なタイプは次のとおりです。

COBOL タイプ	Data Simplifier タイプ
PIC X(n)	AlphanumericType
PIC 9(n)	ZonedType
PIC 9(n) COMP-3	PackedType
PIC 9(n) COMP-5	BinaryType

集計フィールド

集計フィールドは、その内容 (他の集計フィールドまたは基本フィールド) のメモリレイアウトを整理します。それら自体には基本型はありません。

Group フィールドはメモリ内の連続フィールドを表します。含まれている各フィールドはメモリ内で同じ順序で配置され、メモリ内のグループフィールド位置を基準にした最初のフィールドのオフセットは 0 で、2 番目のフィールドのオフセットは 0 + (size in bytes of first field) という具合です。それらは同じ格納フィールドの下にある COBOL フィールドのシーケンスを表すために使用されます。

Union フィールドは、同じメモリにアクセスする複数のフィールドを表します。格納されている各フィールドは、メモリ内のユニオンフィールド位置を基準にしたオフセットが 0 で配置されます。例えば、それらは COBOL の「REDEFINES」コンストラクト (最初のユニオンの子は再定義されたデータ項目、2 番目の子は最初の再定義など) を表すために使用されます。

配列フィールド (Repetition のサブクラス) は、メモリ内の子フィールド (集計自体であれ基本項目であれ) のレイアウトの繰り返しを表します。このような子レイアウトが指定された数だけメモリ内に配列され、それぞれのオフセットは index * (size in bytes of child) になります。これらは COBOL の「OCCURS」コンストラクトを表すために使われます。

プリミティブ型

モダナイゼーションケースによっては、「プリミティブ」を使用して独立した「ルート」データ項目を提示することもできます。これらは と非常によく似ていますRecordEntityが、そこから得られるものではなく、生成されたコードに基づくものでもありません。代わりに、インターフェイスのサブクラスとして AWS Blu Age ランタイムによって直接提供されますPrimitive。このような提供クラスの例には、Alphanumeric または ZonedDecimal があります。

データバインディングとアクセス

構造化データと基礎データを関連付ける方法は複数あります。

そのための重要なインターフェイスは RecordAdaptable で、これは RecordAdaptable の基礎データに「書き込み可能なビュー」を提供する Record を取得するために使用されます。以下の説明では、複数のクラスが RecordAdaptable を実装しています。逆に、 AWS Blu Age APIs と低レベルメモリ (プログラム引数、ファイル I/O レコード、CICS カンマエリア、割り当てられたメモリなど) を操作するコードでは、そのメモリの ハンドルRecordAdaptableとして が予期されることがよくあります。

COBOL のモダナイズの場合、ほとんどのデータ項目はメモリと関連付けられ、対応するプログラム実行の存続期間中は固定されます。この目的のために、RecordEntity サブクラスは生成された親オブジェクト (プログラムコンテキスト) で一度インスタンス化され、RecordEntity バイトサイズに基づいて、基礎となる Record のインスタンス化を行います。

COBOL では、LINKAGE 要素をプログラム引数に関連付けたり、SET ADDRESS OF コンストラクトをモダナイズしたりする場合などに、RecordEntity インスタンスを提供された RecordAdaptable に関連付ける必要があります。そのため、次の 2 つのメカニズムがあります。

- RecordEntity インスタンスが既に存在する場合は、(Bindable から継承した) RecordEntity.bind(RecordAdaptable) メソッドを使用して、このインスタンスにこの RecordAdaptable を「ポイント」させることができます。次に、RecordEntity で呼び出されるゲッターやセッターは、基になる RecordAdaptable バイトによってバックアップ (バイト読み取りまたは書き込み) されます。
- RecordEntity をインスタンス化する場合は、RecordAdaptable を受け付ける生成コンストラクタを使用できます。

逆に、現在構造化データにバインドされている Record にもアクセスできます。このため、RecordEntity は RecordAdaptable を実装し、getRecord( ) そのようなすべてのインスタンスから呼び出すことができます。

最後に、COBOL や CICS の動詞の多くは、読み取りまたは書き込みのために単一フィールドにアクセスする必要があります。RangeReference クラスはそのようなアクセスを表すために使用されます。そのインスタンスは、RecordEntity が生成した getXXXReference( ) メソッド (XXX はアクセスされたフィールド) から取得して、ランタイムメソッドに渡すことができます。RangeReference は通常は RecordEntity または Group 全体へのアクセスに使用され、サブクラス ElementaryRangeReference は Elementary フィールドへのアクセスを表します。

上記のほとんどの観測値は、(生成されたコードの代わりに) AWS Blu Age ランタイムによって提供されるRecordEntity場合と同様の動作の実装に重点を置いているため、Primitiveサブクラスに適用されることに注意してください。この目的のために、Primitive のすべてのサブクラスは RecordEntity サブクラスおよび基本フィールドの両方の代わりに使用できるように RecordAdaptable、ElementaryRangeReference、Bindable インターフェイスを実装しています。

説明した Java タイプの FQN

次の表は、このセクションで説明した Java タイプの完全修飾名を示しています。

短い名前	完全修飾名
Alphanumeric	com.netfective.bluage.gapwalk.datasimplifier.elementary.Alphanumeric
AlphanumericType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.AlphanumericType
BinaryType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.BinaryType
Bindable	com.netfective.bluage.gapwalk.datasimplifier.data.Bindable
Elementary	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Elementary
ElementaryRangeReference	com.netfective.bluage.gapwalk.datasimplifier.entity.ElementaryRangeReference
Filler	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Filler
Group	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Group
PackedType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.PackedType

短い名前	完全修飾名
Primitive	com.netfective.bluage.gapwalk.datasimplifier.elementary.Primitive
RangeReference	com.netfective.bluage.gapwalk.datasimplifier.entity.RangeReference
RangeType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.RangeType
Record	com.netfective.bluage.gapwalk.datasimplifier.data.Record
RecordAdaptable	com.netfective.bluage.gapwalk.datasimplifier.data.RecordAdaptable
RecordEntity	com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity
Repetition	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Repetition
Union	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Union
ZonedDecimal	com.netfective.bluage.gapwalk.datasimplifier.elementary.ZonedDecimal

短い名前	完全修飾名
ZonedType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.ZonedType

AWS Blu Age ランタイム設定および設定ファイル

Velocity フレームワークとクライアントコードは Spring Boot framework を使用するウェブアプリケーションです。Spring の機能を活用して、考えられる場所や優先順位のルールを複数指定して設定を行います。groovy スクリプト、SQL など、他の多くのファイルを指定する場合にも同様の優先順位ルールがあります。

Velocity フレームワークには、必要に応じてオプトインできるオプションのウェブアプリケーションも追加されています。

トピック

- アプリケーション設定の基本
- アプリケーションの優先順位
- データベース用 JNDI
- AWS シークレットの使用
- その他のファイル (groovy、SQL など)
- その他のウェブアプリケーション
- プロパティを有効にする
- Gapwalk アプリケーションの認証の設定

アプリケーション設定の基本

アプリケーション設定を処理するデフォルトの方法は、アプリケーションサーバーの configフォルダで提供される専用 YAML ファイルを使用することです。2 つの主要な YAML 設定ファイルがあります。

- application-main.yaml
- application-*profile*.yaml (*profile* 値はアプリケーション生成時に設定されます)。

最初のファイルはフレームワーク、つまり Gapwalk-application.war を設定し、2 番目のファイルはクライアントアプリケーション専用の追加オプション用です。これは Spring プロファイルを使用しても機能します。Gapwalk アプリケーションは main プロファイルを使用し、クライアントアプリケーションは *profile* プロファイルを使用します。

次の例は、代表的なメインの YAML ファイルを示しています。

```
########################################
##### JICS datasource configuration #####
########################################
datasource:
  jicsDs:
    driver-class-name : org.postgresql.Driver
    url: jdbc:postgresql://localhost/jics
    username: jics
    password: jics
    type : org.postgresql.ds.PGSimpleDataSource

#####################################################
##### Embedded Bluesam datasource configuration #####
#####################################################
  bluesamDs :
    driver-class-name : org.postgresql.Driver
    url : jdbc:postgresql://localhost/bluesam
    username : bluesam
    password : bluesam
    type : org.postgresql.ds.PGSimpleDataSource

########################################
##### Embedded Bluesam configuration #####
########################################
bluesam :
  remote : false
  cache : ehcache
  persistence : pgsql #pgsql, mssql, xodus...
  ehcache:
    resource-pool:
      size: 4GB
  write-behind:
```

次の例は、代表的なクライアント YAML ファイルを示しています。

```
# Logback context logger integration.
logging.config : classpath:logback-[redacted].xml
# Limits Spring logger output.
logging.level.org.springframework.beans.factory.support.DefaultListableBeanFactory : WARN
logging.level.org.springframework.statemachine : WARN
# If the datasource support mode is not static-xa, spring JTA transactions autoconfiguration must me disabled
spring.jta.enabled : false

spring:
  aws:
    client:
      datasources:
        names: primary
        primary:
          secret: arn:aws:secretsmanager:[redacted]

spring.jta.atomikos.datasource.primary.unique-resource-name: primary
spring.jta.atomikos.datasource.primary.xa-data-source-class-name: org.postgresql.xa.PGXADataSource
spring.jta.atomikos.datasource.primary.maxPoolSize: 20
spring.jta.atomikos.datasource.primary.autoCommit: false
```

YAML ファイルの内容については、「プロパティを有効にする」を参照してください。

アプリケーションの優先順位

これらの設定ファイルには、Spring の優先順位ルールが適用されます。特筆すべき点は以下のとおりです。

- application-main YAML ファイルはデフォルト値を持つ Gapwalk メインウォーファイルに表示され、configフォルダ内のファイルがそれよりも優先されます。
- クライアントアプリケーションの設定についても同じことを行う必要があります。
- サーバーの起動時に、コマンドラインで追加のパラメータを渡すことができます。これらは YAML を上書きします。

詳細については、「Spring Boot の公式ドキュメント」を参照してください。

データベース用 JNDI

データベース設定は Tomcat の context.xml ファイル内の JNDI で提供されている場合があります。このような設定は YAML の設定よりも優先されます。ただし、これを使用しても認証情報をシークレットマネージャーでラップできなくなることに注意してください (以下を参照)。

次の例は、JICS および BluSam データベースの設定例を示しています。

```
<Resource auth="Container" driverClassName="org.postgresql.Driver" initialSize="0"
 maxIdle="5"
```

```
    maxOpenPreparedStatements="-1" maxTotal="10" maxWaitMillis="-1" name="jdbc/jics"
    poolPreparedStatements="true" testOnBorrow="false" type="javax.sql.DataSource"
    url="jdbc:postgresql://XXXX.rds.amazonaws.com:5432/XXXX" username="XXXX"
  password="XXXX" />
```

jdbc/jics

JICS データベースjdbc/jicsの場合は 、blusam データベースの場合は jdbc/bluesam (「e」に注意) になります。

url="jdbc:postgresql://XXXX.rds.amazonaws.com:5432/XXXX" username="XXXX" password="XXXX"

データベース URL、ユーザー名、パスワード。

AWS シークレットの使用

認証情報を含むリソース設定の一部は、シー AWS クレットを使用してさらに保護できます。重要なデータを AWS シークレットに保存し、シークレットへの参照を YAML 設定に保持して、シークレットコンテンツが Tomcat の起動時にオンザフライで選択されるようにすることがアイデアです。

Aurora のシークレット

Aurora データベース設定 (jics、blusam、カスタマー db など) では、組み込みのデータベースシークレットを使用します。これにより、対応するデータベースからすべての関連フィールドが自動的に入力されます。

> **Note**
>
> dbname キーはオプションで、データベースの設定によっては、シークレットに入るか入らないかが決まります。手動で追加することも、YAML ファイルに名前を指定することもできます。

他のシークレット

その他のシークレットは、パスワードが 1 つしかないリソース (特にパスワードで保護された Redis キャッシュ) 用です。この場合、もう 1 つのタイプのシークレットを 1 つの password キーで使用する必要があります。

YAML によるシークレットへの参照

は、さまざまなリソースのシークレット ARN を参照application-main.yamlできます。最も重要なのは以下のとおりです。

- での JICS データベース認証情報 spring.aws.jics.db.secret
- で JICS TS Queues Redis 認証情報 spring.aws.client.jics.queues.ts.redis.secret
- spring.aws.client.bluesam.db.secret による Blusam データベース認証情報
- spring.aws.client.bluesam.redis.secret による Blusam キャッシュパスワード
- Blusam は spring.aws.client.bluesam.locks.redis.secret キャッシュパスワードをロックします

次の例は、これらのシークレットを YAML ファイルで宣言する方法を示しています。

```
spring:
  aws:
    client:
      bluesam:
        locks:
          redis:
            secret: arn:aws:secretsmanager:XXXX
        db:
          dbname: bluesam
          secret: arn:aws:secretsmanager:XXXX
        redis:
          secret: arn:aws:secretsmanager:XXXX
      jics:
         queues:
          ts:
           redis:
            secret: arn:aws:secretsmanager:XXXX
    jics:
      db:
        secret: arn:aws:secretsmanager:XXXX
```

データベース名: bluesam

> この例では、データベースの名前はシークレットには含まれておらず、代わりにここで指定されています。

クライアント application-*profile*.yaml は、クライアントデータベースのシークレット ARN を参照できます。これには、以下の例のように、データソースを一覧表示する追加のプロパティが必要です。

```
spring:
  aws:
    client:
      datasources:
        names: primary,host
        primary:
          secret: arn:aws:secretsmanager:XXXX
        host:

          secret: arn:aws:secretsmanager:XXXX
```

名前: プライマリ、ホスト

プライマリとホストという名前の 2 つのクライアントデータソースがあり、それぞれにデータベースと認証情報がある場合の例です。

データベース名: mydb

この例では、「ホスト」データベースの名前はシークレットにはなく、代わりにここで指定されています。一方、「プライマリ」データベースの名前はシークレットにあります。

XA がサポートするシークレットキーはありません

- エンジン (postgres/oracle/db2/mssql)
- port
- dbname
- currentSchema
- username
- password
- url

yml プロパティに加えて (disable/allow/prefer/require/verify-ca/verify-full) spring.aws.rds.ssl.cert-path で指定されたsslModeシークレットキーpostgresのみの場合、 は SSL で接続できます。

XA でサポートされるシークレットキー

クライアントデータベースが XA を使用している場合、サブ xa-properties はシークレット値を通じてサポートされます。

- ホスト
- port
- dbname
- currentSchema
- username
- password
- url
- sslConnection (true/false)

ただし、他の xa プロパティ (maxPoolSizeや などdriverType) `spring.jta.atomikos.datasource.XXXX.unique-resource-name`では、通常の YAML キーを引き続き指定する必要があります。

シークレット値は YAML プロパティを上書きします。

その他のファイル (groovy、SQL など)

カスタマープロジェクトが使用する他のファイルには、Spring 設定のものと同様の優先順位ルールが使われています。例:

- Groovy スクリプトは `scripts` フォルダまたはサブフォルダ内の `.groovy` ファイルです。
- SQL スクリプトは `sql` フォルダまたはサブフォルダ内の `.sql` ファイルです。
- デーモンスクリプトは `daemons` フォルダまたはサブフォルダ内の `.groovy` ファイルです。
- クエリデータベースマッピングファイルは、`sql` フォルダのサブフォルダにある `queries-database.mapping` ファイルという名前のファイルです。
- Jasper テンプレートは、`templates` フォルダまたはサブフォルダ内の `.jrxml` ファイルです。
- データセットカタログは、`catalog` フォルダ内の `.json` ファイルです。
- リンクファイルは、`lnk` フォルダ内の `.json` ファイルです。

これらの場所はすべて、システムプロパティまたはクライアント yml プロパティを使用して上書きできます。

- Groovy スクリプトの場合: `configuration.scripts`
- SQL スクリプトの場合: `configuration.sql`
- デーモンスクリプトの場合: `configuration.daemons`
- クエリデータベースマッピングファイルの場合: `configuration.databaseMapping`
- Jasper テンプレートの場合: `configuration.templates`
- データセットカタログの場合: `configuration.catalog`
- リンクファイルの場合: `configuration.lnk`

プロパティが見つからない場合、ファイルは上記のデフォルトの場所から取得されます。検索は、まず tomcat 作業ディレクトリをルートとして実行され、最後にアプリケーション war ファイルで実行されます。

その他のウェブアプリケーション

Velocity フレームワークの `webapps-extra` フォルダには、その他のウェブアプリケーションが含まれています。これらのアプリケーションは、Tomcat サーバーではデフォルトでは提供されません。

これらのウェブアプリケーションへのオプトインはモダナイゼーションプロジェクトによって異なり、必要な war ファイルを `webapps-extra` フォルダから `webapps` フォルダに移動して行います。それ以降は、次回の起動時に Tomcat サーバーが war を処理します。

プロジェクト固有の追加設定の中には、 ファイルで説明されているように、追加するウォーごとに YAML 設定`application-main.yml`ファイルで追加できるものもあります。追加の war は以下のとおりです。

- `gapwalk-utility-pgm.war`: ZOS ユーティリティプログラムのサポートを含み、その構成として `application-utility-pgm.yaml` を使用します。
- `gapwalk-cl-command.war`: AS/400 ユーティリティプログラムのサポートを含み、その構成として `application-cl-command.yaml` を使用します。
- `gapwalk-hierarchical-support.war`: IMS/MFS トランザクションサポートを含み、その構成として `application-jhdb.yaml` を使用します。

プロパティを有効にする

Spring Boot アプリケーションapplication-main.ymlでは、リスニングポート、データベース接続など、さまざまな種類のプロパティを定義する設定ファイルです。

トピック

YML 表記

次のドキュメントでは、 などのプロパティparent.child1.child2=trueは YAML 形式で次のように記述されています。

```
parent:
 child1:
   child2: true
```

クイックスタート/ユースケース

以下のユースケースは、適用可能なキーと値の例を示しています。

- デフォルトの application-main.yml

```
----
#### DEFAULT APPLICATION-MAIN.YML FILE      #####
#### SHOWING USEFUL CONFIGURATION ELEMENTS #####
#### SHOULD BE OVERRIDDEN AND EXTERNALIZED   #####

###################################
##### Logging configuration #####
###################################
logging:
  config: classpath:logback-main.xml
  level.org.springframework.beans.factory.support.DefaultListableBeanFactory : WARN

###################################
##### Spring configuration #####
```

```
################################
spring:
  quartz:
    auto-startup: false
    scheduler-name: Default
    properties:
      org.quartz.threadPool.threadCount: 1
  jta:
    enabled: false
    atomikos.properties.maxTimeout : 600000
    atomikos.properties.default-jta-timeout : 100000
  jpa:
# DISABLE OpenEntityManagerInViewInterceptor
    open-in-view: false
    # Fix Postgres JPA Error:
    # Method org.postgresql.jdbc.PgConnection.createClob() is not yet implemented.
    properties.hibernate.temp.use_jdbc_metadata_defaults : false
#####################################
##### Jics tables configuration #####
#####################################

    # The dialect should match the jics datasource choice
    database-platform : org.hibernate.dialect.PostgreSQLDialect #
 org.hibernate.dialect.PostgreSQLDialect, org.hibernate.dialect.SQLServerDialect

    # those properties can be used to create and initialize jics tables
 automatically.
#    properties:
#      hibernate:
#      globally_quoted_identifiers: true
#        hbm2ddl:
#           import_files_sql_extractor :
 org.hibernate.tool.hbm2ddl.MultipleLinesSqlCommandExtractor
#           import_files : file:./setup/initJics.sql
#           auto : create

#########################
###### Level 2 cache #####
#########################
#        cache:
#          use_second_level_cache: true
#          use_query_cache: true
#          region:
#            factory_class: org.hibernate.cache.ehcache.EhCacheRegionFactory
```

```
#      javax:
#        persistence:
#          sharedCache:
#            mode: ENABLE_SELECTIVE
##########################
###### Redis settings #####
##########################
  session:
    store-type: none #redis

########################################
##### JICS datasource configuration #####
########################################
datasource:
  jicsDs:
    driver-class-name : org.postgresql.Driver # org.postgresql.Driver,
 com.microsoft.sqlserver.jdbc.SQLServerDriver
    url: jdbc:postgresql://localhost/jics # jdbc:postgresql://localhost:5433/jics,
 jdbc:sqlserver://localhost\SQLEXPRESS:1434;databasename=jics;
    username: jics
    password: jics
    type : org.postgresql.ds.PGSimpleDataSource #
 org.postgresql.ds.PGSimpleDataSource,
 com.microsoft.sqlserver.jdbc.SQLServerDataSource

#####################################################
##### Embedded Bluesam datasource configuration #####
#####################################################
  bluesamDs :
    driver-class-name : org.postgresql.Driver # org.postgresql.Driver,
 com.microsoft.sqlserver.jdbc.SQLServerDriver
    url : jdbc:postgresql://localhost/bluesam # jdbc:postgresql://localhost:5433/
jics, jdbc:sqlserver://localhost\SQLEXPRESS:1434;databasename=jics;
    username : bluesam
    password : bluesam
    type : org.postgresql.ds.PGSimpleDataSource #
 org.postgresql.ds.PGSimpleDataSource,
 com.microsoft.sqlserver.jdbc.SQLServerDataSource

########################################
##### Embedded Bluesam configuration #####
########################################
bluesam :
  remote : false
```

```
  cache : ehcache
  persistence : pgsql #pgsql, mssql, xodus...
  ehcache:
    resource-pool:
      size: 4GB
  write-behind:
    enabled: true
  pgsql :
    dataSource : bluesamDs

#########################
##### Jics settings #####
#########################
rabbitmq.host: localhost
jics:
  cache: false #redis
  resource-definitions.store-type: jpa # default value: jpa, other possible value:
 redis
  redis.hostname: 127.0.0.1 # Redis server host.
  redis.password: redis # Login password of the redis server.
  redis.port: 6379 # Redis server port.
  redis.username: # Redis username
  redis.mode: standalone # Redis mode. Possible values: standalone, cluster
jics.disableSyncpoint : false
#jics.initList:
#jics.parameters.datform: DDMMYY
#jics.parameters.applid: VELOCITY
#jics.parameters.sysid: CICS
#jics.parameters.eibtrmid: TERM
#jics.parameters.userid: MYUSERID
#jics.parameters.username: MYUSERNAME
#jics.parameters.opid: XXX
#jics.parameters.cwa.length: 0
#jics.parameters.netname: MYNETNAME
#jics.parameters.jobname: MJOBNAME
#jics.parameters.sysname: SYSNAME

###############################################
##### Jics RunUnitLauncher pool settings #####
###############################################
#jics.runUnitLauncherPool.enable: false
#jics.runUnitLauncherPool.size: 20
#jics.runUnitLauncherPool.validationInterval: 1000
```

```
#########################
##### Jhdb settings #####
#########################
#jhdb.lterm: LTERMVAL
#jhdb.identificationCardData: SomeIDData

###################################
##### DateHelper configuration ####
###################################
#forcedDate: "2013-08-26T12:59:58+01:57"

#############################
##### Sort configuration ####
#############################
#externalSort.threshold: 256MB

###################################
##### Server timeout (10 min)  ####
###################################
spring.mvc.async.request-timeout: 600000

#############################
##### DATABASE STATISTICS  ####
#############################
databaseStatistics : false

######################
##### CALLS GRAPH ####
######################
callGraph : false

#############################
##### SQL SHIFT CODE POINT ####
#############################
# Code point 384 match unicode character \u0180
sqlCodePointShift : 384

#############################
##### LOCK TIMEOUT RECORD  ####
#############################
# Blu4IV record lock timeout
lockTimeout : 100

#############################
```

```
##### REPORTS OUTPUT PATH ####
##############################
reportOutputPath: reports

##############################
##### TASK EXECUTOR       ####
##############################
taskExecutor:
  corePoolSize: 5
  maxPoolSize: 10
  queueCapacity: 50
  allowCoreThreadTimeOut: false

##############################
##### PROGRAM NOT FOUND   ####
##############################
stopExecutionWhenProgNotFound: false

#########################################################
#####  DISP DEFAULT VALUE (to be removed one day) ####
#########################################################
defaultKeepExistingFiles: true

#####################################
#####  JOBQUEUE CONFIGURATION    ####
#####################################
jobqueue:
  api.enabled: false
  impl: none # possible values: quartz, none
  schedulers:  # list of schedulers
    -
      name: queue1
      threadCount: 5
    -
      name: queue2
      threadCount: 5

#################################################################################
##### QUERY BUILDING                                                          ##
# useConcatCondition : false by default
# if true, in the query, the where condition is build with key concatenation  ##
#################################################################################
# query.useConcatCondition: true
```

```
----
```

- LISTCAT コマンドで可変長ファイルを使用する

```
[**/*.*]
encoding=IBM930
reencoding=false

[global]
listcat.variablelengthpreprocessor.enabled=true
listcat.variablelengthpreprocessor.type=rdw
# use "rdw" if your .listcat file contains a set of records (RDW)
# use "bdw" if your .listcat file contains a set of blocks (bdw)
```

- LOAD/UNLOAD ユーティリティで Null バイトインジケータ値を指定します。

```
# Unload properties
# For date/time: if use database configuration is enabled, formats are ignored
# For nbi; use hexadecimal syntax to specify the byte value
# - When the value is null in database : the value dumped to the file is filled by
 low value characters and the NBI is
# equal to the byte 6F (the ? character)
# - When the value is not null in database and the column is nullable: the NBI is
 equal to the byte 00 (low value) and NOT
# equal to the byte 40 (space)
unload:
  sqlCodePointShift: 0
  nbi:
    whenNull: "6F"
    whenNotNull: "00"
  useDatabaseConfiguration: false
  format:
    date: MM/dd/yyyy
    time: HH.mm.ss
    timestamp: yyyy-MM-dd-HH.mm.ss.SSSSSS
```

メインのアプリケーションで使用可能なプロパティ

この表は、キー/バリューのパラメータを網羅的にまとめたものです。

キー	タイプ	デフォルト値	説明
`logging.config`	パス	クラスパス: logback-main.xml	logback 設定ファイルへの参照用の標準キー。他の標準ロギング記録キーも使用できます。
`spring.jta.enabled`	ブール値	false	標準キー。データソースサポートモードが static-xa でない場合は、spring JTA トランザクションの自動設定を無効にする必要があります。
`datasource.jicsDs + -driver-class-name + -url + -username + -password + -type`	サブキー付きの標準 spring データソース		Jics データベースの接続情報が含まれています。あるいは、xref:../configuration/configuration.adoc[Configuration] で説明されているように、AWS シークレットの使用を強くお勧めします。
`datasource.bluesamDs + -driver-class-name + -url + -username + -password + -type`	サブキー付きの標準 spring データソース		Blusam データベースの接続情報が含まれます。あるいは、xref:../configuration/configuration.adoc[Configuration] で説明されているよう

キー	タイプ	デフォルト値	説明
			に、AWS シークレットの使用を強くお勧めします。
bluesam.disabled	ブール値	false	blusam を完全に無効にするかどうか。
bluesam.cache	string		設定されていない場合、blusam キャッシュは使用されません。指定できる値 (キャッシュ実装) は ehcache と redis です。
forcedDate	string		指定された日付がある場合は、日付を強制的にその日付に合わせます。
frozenDate	ブール値	true	日付を固定するかどうかを指定します。forcedDate も設定されている場合にのみ適用されます。
externalSort.threshold	データサイズ (例: 12MB)		ソートのしきい値: 外部 (マージ) ソートに切り替えるタイミング。
jics.parameters.datform	string	MMDDYY	日付形式。

キー	タイプ	デフォルト値	説明
`jics.initList`	string		カンマ区切りの、jics 初期化リスト。存在する場合、Tomcat の起動時に CICS リストの中でアクティブ化するリストの名前をカンマで区切って定義します。値の例: `$UUU,DFH $IVPL,PEZ1` これはそれらのリストに含まれるグループとその基礎となるリソース定義にカスケードされ、ランタイムに表示されます。デフォルトでは空です。
`jics.parameters.applid`	string	VELOCITY	JICS でのアプリケーションの識別に適用されます (最低 4 文字、最大長なし)。
`jics.parameters.sysid`	string	CICS	システム識別子 (SYSID)。
`jics.parameters.eibtrmid`	string	TERM	ターミナル識別子 (最大 4 文字、最小 1 文字)。

キー	タイプ	デフォルト値	説明
jics.parameters.userid	string		ユーザー ID (最大 8 文字、最小文字数なし)。値が指定されていない場合 (デフォルトでは空白)、HTTP セッション ID がユーザー ID として使用されます。
jics.parameters.username	string	MYUSERNAME	ユーザー名 (最大 10 文字、最小 1 文字)。
jics.parameters.netname	string	MYNETNAME	ネットワーク名 (最大 8 文字、最小 1 文字)。
jics.parameters.opid	string	XXX	3 文字のオペレータ ID。
jics.parameters.jobname`	string	MJOBNAME	ジョブ名。
jics.parameters.sysname	string	SYSNAME	AS400 システム名 (sysname)。
jics.parameters.cwa.length	数値	0	共通ワークエリア (cwa) の長さ。
jics.parameters.charset	string	CP037	JICS がグローバルに使用する文字セット。

キー	タイプ	デフォルト値	説明
jics.parameters.tsqimpl	string	bluesam	JICS 一時ストレージキュー (TSQ) の実装 (指定できる値は bluesam/memory/redis)
jics.queues.ts.redis.hostname	string	127.0.0.1	jics キャッシュ redis サーバーのホスト名。
jics.queues.ts.redis.port	数値	6379	jics キャッシュ redis サーバーのポート。
jics.queues.ts.redis.password	string	redis	jics キャッシュ redis サーバーのパスワード。
jics.queues.ts.redis.username	string		jics キャッシュ redis サーバーのユーザー名。デフォルトは空白 (ユーザー名なし) です。
jics.queues.ts.redis.mode	string	スタンドアロン	jics キャッシュモード。想定される値は、standalone または cluster です。デフォルトは standalone です。
lockTimeout	数値	500	ロックタイムアウト (ミリ秒単位)。

キー	タイプ	デフォルト値	説明
`sqlCodePointShift`	数値		オプション。SQL コードポイントのシフト。レガシー rdbms データを最新の rdbms に移行する際に発生する可能性のある制御文字のコードポイントをシフトします。例えば、Unicode 文字 `\u0180` と一致するように `384` を指定できます。
`sqlIntegerOverflowAllowed`	ブール値	false	SQL 整数のオーバーフローを許可するかどうか、つまり、ホスト変数に大きな値を入れることを許可するかどうかを指定します。

キー	タイプ	デフォルト値	説明
database.cursor.overflow.allowed	ブール値	true	カーソルのオーバーフローを許可するかどうかを指定します。true に設定すると、カーソルの位置に関係なく、カーソル上で次の呼び出しが実行されます。false に設定すると、次のカーソル呼び出しを実行する前に、カーソルが最後の位置にあるかどうかを確認できます。カーソルが SCROLLABLE (SENSITIVE または INSENSITIVE) の場合にのみ有効になります。
reportOutputPath	string	/reports	レポート出力パス。
spring.session.store-type	string	なし	高可用性環境用のセッションキャッシュ。想定される値は、none または redis です。デフォルトは none です。

キー	タイプ	デフォルト値	説明
stopExecutionWhenProgNotFound	ブール値	true	プログラムが見つからない場合に実行を停止するかどうかを指定します。true に設定すると、プログラムが見つからなかった場合に実行が中断されます。
forceHR	ブール値	false	コンソール出力またはファイル出力のいずれかで、人間が読める形式の SYSPRINT を使用するかどうかを指定します。
rollbackOnRTE	ブール値	false	ランタイムの例外時に暗黙的な実行ユニットトランザクションをロールバックするかどうかを指定します。
sctThreadLimit	long	5	スクリプトをトリガーするスレッドの上限。

キー	タイプ	デフォルト値	説明
dataSimplifier.onInvalidNumericData	string	拒否	無効な数値データをデコードしたときの対処方法。許可される値は、reject/tolerates paces /tolerates paceslowvalues /toleratemost です。デフォルトは reject です。
filesDirectory	string		入出力ファイルをバッチ処理するディレクトリ。
ims.messages.extendedSize	ブール値	false	ims メッセージに extendedSize を設定するかどうかを指定します。
defaultKeepExistingFiles	ブール値	false	データセットのデフォルトの以前の値を設定するかどうかを指定します。
jics.db.ddlScriptLocation	string		Jics ddl スクリプトの場所。.sql スクリプトを使用して jics データベーススキーマを開始できます。デフォルトでは空白。例えば、./jics/sql/jics.sql などです。

キー	タイプ	デフォルト値	説明
jics.db.schemaTestQueryLocation	string		jics スキーマ内のオブジェクト数 (存在する場合) を返す一意のクエリを含む必要がある sql ファイルの場所。
jics.db.dataScriptLocation	string		メインフレームからの CSD エクスポートを解析してアナライザーが作成した initJics.sql スクリプトの場所。
jics.db.dataTestQueryLocation	string		オブジェクト数を返すと予想される 1 つの sql クエリを含む sql スクリプトの場所 (例: jics プログラムテーブル内のレコード数のカウント)。カウントが 0 の場合、データベースは jics.db.dataScript Location スクリプトを使用してロードされます。それ以外の場合は、データベースのロードはスキップされます。
jics.data.dataJson InitLocation	string		

キー	タイプ	デフォルト値	説明
jics.xa.agent.timeout	数値		
query.useConcatCondition	ブール値	false	キー条件がキー連結によって構築されるかどうかを指定します。
system.qdecfmt	string		
disposition.checkexistence	ブール値	false	DISP SHR または OLD のデータセットのファイル存在確認を解除するかどうかを指定します。
useControlMVariable	ブール値	false	変数置換に Control-M 仕様を使用するかどうかを指定します。
card.encoding	string	CP1145	カードのエンコーディング: useControlMVariable`.　で使用します。
mapTransfo.prefixes	string	&,@,%%	controlM 変数を変換するときに使用するプレフィックスのリスト。それぞれをカンマで区切ります。
checkinputfilesize	ブール値	false	ファイルサイズがレコードサイズの倍数である場合にチェックを解除するかどうかを指定します。

キー	タイプ	デフォルト値	説明
stepFailWhenAbend	ブール値	true	ステップが失敗した場合や実行が完了した場合に、異常終了を発生させるかどうかを指定します。
bluesam.fileLoading.commitInterval	数値	100000	bluesam のコミット間隔。
uppercaseUserInput	ブール値	true	ユーザー入力を大文字にする必要があるかどうかを指定します。
jhdb.lterm	string		IMS エミュレーションの場合に共通の論理ターミナル ID を強制的に使用することができます。設定されていない場合は、sessionId が使用されます。
jhdb.identificationCardData	string		一部の「オペレータ ID カードデータ」を CARD パラメータで指定された MID フィールドにハードコードするために使用されます。デフォルトでは空白、入力制限なし。

キー	タイプ	デフォルト値	説明
encoding	string	ASCII	プロジェクトで使われているエンコードです (groovy ファイルには使用されていません)。有効なエンコーディング CP1047、IBM930、ASCII、U などが必要です。
cl.configuration.context.encoding	string	CP297	CL ファイルのエンコーディング。有効なエンコーディング CP1047、IBM930、ASCII、U などが必要です。デフォルト値は CP297 です
cl.zonedMode	string	EBCDIC_STRICT	制御言語 (CL) コマンドをエンコードまたはデコードするためのモード。許可される値は、EBCDIC_STRICT /EBCDIC_MODIFIED /AS400 です。

キー	タイプ	デフォルト値	説明
ims.programs	string		使用する IMS プログラムのリスト。各パラメータはセミコロン (;) で、各トランザクションはカンマ (,) で区切ります。例: PCP008,PCT008;PCP054,PCT054;PCP066,PCT066;PCP068,PCT068;
jhdb.configuration.context.encoding	string	CP297	JHDB (Java 階層型データベース) のエンコーディング。有効なエンコーディング文字列 CP1047、IBM930、ASCII、U などが必要です。
jhdb.metadata.extrapath	string	file:./setup/	psbs フォルダと dbds フォルダ用に、ランタイム固有の追加のルートフォルダを指定する設定パラメータ。

キー	タイプ	デフォルト値	説明
jhdb.checkpointPersistence	string	なし	チェックポイント永続化モード。許可される値は、none/add/end です。新しいチェックポイントが作成されてレジストリに追加されたときに、チェックポイントを永続化するために add を使用します。サーバーのシャットダウン時にチェックポイントを永続化するために end を使用します。それ以外の値は永続化を無効にします。新しいチェックポイントがレジストリに追加されるたびに、既存のチェックポイントはすべてシリアル化され、ファイルは消去されることに注意してください。ファイル内の既存のデータへの追加ではありません。そのため、チェックポイントの数によっては、パフォーマンスに何らかの影響を与える可能性があります。

キー	タイプ	デフォルト値	説明
jhdb.checkpointPath	string	file:./setup/	jhdb.checkpointPersistence が none ではない場合、このパラメータを使用してチェックポイント永続化パス (checkpoint.dat ファイルの保存場所) を設定できます。レジストリに含まれるすべてのチェックポイントデータがシリアル化され、指定されたフォルダにあるファイル (checkpoint.dat) にバックアップされます。このバックアップの対象となるのはチェックポイントデータ (scriptId、stepId、データベース位置、チェックポイントエリア) のみであることに注意してください。
jhdb.navigation.cachenexts	数値	5000	RDBMS の階層ナビゲーションに使用されるキャッシュ時間 (ミリ秒単位)。

キー	タイプ	デフォルト値	説明
jhdb.use-db-prefix	ブール値	true	RDBMS の階層ナビゲーションでデータベースプレフィックスを有効にするかどうかを指定します。
jhdb.query.limitJoinUsage	ブール値	true	RDBMS グラフで結合使用制限パラメータを使用するかどうかを指定します。
taskExecutor.corePoolSize	数値	5	ターミナルのトランザクションが groovy スクリプトによって開始されると、新しいスレッドが作成されます。このパラメータは、コアプールサイズを設定するために使用します。
taskExecutor.maxPoolSize	数値	10	ターミナルのトランザクションが groovy スクリプトによって開始されると、新しいスレッドが作成されます。このパラメータを使用して、最大プールサイズ (並列スレッドの最大数) を設定します。

キー	タイプ	デフォルト値	説明
`taskExecutor.queueCapacity`	数値	50	ターミナルのトランザクションが groovy スクリプトによって開始されると、新しいスレッドが作成されます。このパラメータを使用してキューサイズ (= `taskExecutor.maxPoolSize` に到達したときの保留中のトランザクションの最大数) を設定します。
`taskExecutor.allowCoreThreadTimeOut`	ブール値	false	JCIS でコアスレッドのタイムアウトを許可するかどうかを指定します。これにより、0 以外のキューと組み合わせた場合でも動的な拡大と縮小が可能になります (最大プールサイズはキューがいっぱいになった場合のみ増加するため)。
`jics.runUnitLauncherPool.enable`	ブール値	false	JICS で実行ユニットランチャープールを有効にするかどうかを指定します。

キー	タイプ	デフォルト値	説明
`jics.runUnitLauncherPool.size`	数値	20	JICS 内の実行ユニットランチャープールサイズ。
`jics.runUnitLauncherPool.validationInterval`	数値	1,000	JICS 内の実行ユニットランチャープールの検証間隔 (ミリ秒単位)。
`spring.aws.application.credentials`	string	null	JICS の認証情報プロファイルファイルから AWS 認証情報をロードします。
`jics.queues.sqs.region`	string	eu-west-1	JICS で使用される AWS シンプルキューサービスの AWS リージョン。
`mq.queues.sqs.region`	string	eu-west-3	AWS SQS MQ サービスの AWS リージョン。
`quartz.scheduler.stand-by-if-error`	ブール値	false	ジョブスケジューラがスタンバイモードの場合にジョブの実行をトリガーするかどうかを指定します。「true」の場合、有効になってもジョブの実行はトリガーされません。

キー	タイプ	デフォルト値	説明
databaseStatistics	ブール値	false	SQL ビルダーに統計情報の収集と表示を許可するかどうかを指定します。
dbDateFormat	string	yyyy-MM-dd	db ターゲットの日付形式。
dbTimeFormat	string	HH:mm:ss	db ターゲット時間形式。
dbTimestampFormat	string	yyyy-MM-dd HH:mm:ss.SSSSSS	db ターゲットのタイムスタンプ形式。
dateTimeFormat	string	ISO	dateTimeFormat では、データベースの日付タイムスタンプタイプをデータ簡略化エンティティにスピルする方法について説明します。許可される値は、ISO/EUR/EUR/USA/LOCAL です。
localDateFormat	string		ローカル日付形式のリスト。各形式は \ で区切ります。
localTimeFormat	string		ローカルの時刻形式のリスト。各形式は \ で区切ります

キー	タイプ	デフォルト値	説明
localTimestampFormat	string		ローカルのタイムスタンプ形式のリスト。各形式は \ で区切ります。
pgmDateFormat	string	yyyy-MM-dd	日付時刻形式
pgmTimeFormat	string	HH.mm.ss	pgm (プログラム) の実行に使用される時刻形式。
pgmTimestampFormat	string	yyyy-MM-dd-HH.mm.ss.SSSSSS	タイムスタンプ形式。
cacheMetadata	ブール値	true	データベースメタデータをキャッシュするかどうかを指定します。
forceDisableSQLTrimStringType	ブール値	false	すべての sql 文字列パラメータのトリミングを無効にするかどうかを指定します。
fetchSize	数値		カーソルの fetchSize 値。ロード/アンロードユーティリティでチャンクを使用してデータを取得するときに使用します。
check-groovy-file	ブール値	true	登録する前に groovy ファイルの内容をチェックするかどうかを指定します。

キー	タイプ	デフォルト値	説明
`qtemp.uuid.length`	数値	9	QTEMP の固有の ID の長さ。
`qtemp.dblog`	ブール値	false	QTEMP データベースログ記録を有効にするかどうか。
`qtemp.cleanup.threshold.hours`	数値	0	`qtemp.dblog` をいつ有効にするかを指定します。DB パーティションの有効期間 (時間単位)。
`sort.function`	string		blu4iv データベースのソート関数名。

オプションのウェブアプリケーションで使用可能なプロパティ

モダナイズされたアプリケーションによっては、z/OS、AS/400、IMS/MFS などの依存関係をサポートするオプションのウェブアプリケーションを 1 つ以上設定する必要がある場合があります。以下の表には、各オプションのウェブアプリケーションの設定に使用できるキー/値パラメータの一覧が記載されています。

gapwalk-utility-pgm.war

このオプションのウェブアプリケーションには Z/OS ユーティリティプログラムのサポートが含まれています。

この表は、このアプリケーションのキー/値パラメータのすべてをまとめたものです。

キー	タイプ	デフォルト値	説明
`logging.config`	パス	classpath:logback-utility.xml	logback 設定ファイルへの参照用の標準キー。他の標準ログ

キー	タイプ	デフォルト値	説明
			記録キーも使用できます。
spring.jta.enabled	ブール値	false	標準キー。データソースサポートモードが static-xa でない場合は、spring JTA トランザクションの自動設定を無効にする必要があります。
spring.datasource.primary.jndi-name	string	jdbc/primary	JNDI を使用する場合の、プライマリデータソースの JNDI 名 (Java Naming And Directory Interface)。
primary.datasource + -driver-class-name + -url + -username + -password	サブキー付きの標準 spring データソース		JNDI を使用していない場合、アプリケーションデータベースの接続情報が含まれます。モダナイズされたアプリケーションの yml ファイルと同じ設定にする必要があります。 あるいは、xref:../configuration/configuration.adoc[Configuration] で説明されているように、AWS シークレットの使用を強くお勧めします。

キー	タイプ	デフォルト値	説明
encoding	string	ASCII	ユーティリティプログラムで使用されるエンコーディング。有効なエンコーディング CP1047、IBM930、ASCII、U などが必要です。
sysPunchEncoding	string	ASCII	syspunch エンコーディング文字セット。有効なエンコーディング CP1047、IBM930、ASCII、U などが必要です。
zonedMode	string	EBCDIC_STRICT	ゾーンデータ型をエンコードまたはデコードするためのモード。許可される値は、EBCDIC_STRICT /EBCDIC_MODIFIED /AS400 です。
unload.chunkSize	数値	0	アンロードユーティリティに使用されるチャンクサイズ。

キー	タイプ	デフォルト値	説明
unload.sqlCodePointShift	数値	0	アンロードユーティリティの SQL コードポイントシフト。文字シフト処理を実行します。DB2 のターゲットデータベースが PostgreSQL の場合に必要です。
unload.columnFiller	string	スペース	アンロードユーティリティの列フィラー。
unload.varCharIsNull	ブール値	false	INFTILB プログラムでこのパラメータを使用します。このパラメータを true に設定すると、空白 (スペース) の値を持つ NULL が許容されないフィールドはすべて空の文字列を返します。
unload.useDatabaseConfiguration	ブール値	false	アンロードユーティリティで application-main.yml の日付設定と時刻設定のどちらを使用するかを指定します。

キー	タイプ	デフォルト値	説明
unload.format.date	string	MM/dd/yyyy	unload.useDatabaseConfiguration が有効な場合、アンロードユーティリティで使用する日付形式です。
unload.format.time	string	HH.mm.ss	unload.useDatabaseConfiguration が有効な場合、アンロードユーティリティで使用する時間形式。
unload.format.timestamp	string	yyyy-MM-dd-HH.mm.ss.SSSSSS	unload.useDatabaseConfiguration が有効な場合、アンロードユーティリティで使用するタイムスタンプ形式。
unload.nbi.whenNull	16 進数	6F	データベースからの値が NULL の場合に追加される Null バイトインジケータ (nbi) 値。
unload.nbi.whenNotNull	16 進数	00	データベースの値が NULL でない場合に追加される NULL バイトインジケータ (nbi) 値。

キー	タイプ	デフォルト値	説明
unload.nbi.writeNullIndicator	ブール値	false	アンロード出力ファイルに NULL インジケータを書き出すかどうかを指定します。
unload.fetchSize	数値	0	アンロードユーティリティでカーソルを処理する際にフェッチサイズを調整できます。
treatLargeNumberAsInteger	ブール値	false	大きな数値を Integer として扱うかどうかを指定します。デフォルトでは、BigDecimal のように扱われます。
load.batchSize	数値	0	ロードユーティリティのバッチサイズ。
load.format.localDate	string	dd.MM.yyyy\dd/MM/yyyy\yyyy-MM-dd	使用するロードユーティリティのローカルの日付形式。
load.format.localTime	string	HH:mm:ss\HH.mm.ss	使用するロードユーティリティのローカルの時刻形式。
load.format.dbDate	string	yyyy-MM-dd	使用するロードユーティリティデータベース形式。

キー	タイプ	デフォルト値	説明
`load.format.dbTime`	string	HH:mm:ss	ロードユーティリティデータベースの使用時間。
`load.sqlCodePointShift`	数値	0s	ロードユーティリティの SQL コードポイントシフト。文字シフト処理を実行します。DB2 のターゲットデータベースが PostgreSQL の場合に必要です。
`forcedDate`	string		指定された日付がある場合は、日付を強制的にその日付に合わせます。
`frozenDate`	ブール値	true	日付を固定するかどうかを指定します。`forcedDate` も設定されている場合にのみ適用されます。
`jcl.type`	string	mvs	.jcl ファイルタイプ。許可される値は、`jcl/vse` です。IDCAMS ユーティリティの PRINT/REPRO コマンドは、vse jcl 以外のファイルが空の場合、4 を返します。

キー	タイプ	デフォルト値	説明
hasGraphic	ブール値	false	INFUTILB ユーティリティが GRAPHIC DB2 列を処理する必要があるかどうか。

gapwalk-cl-command.war

このオプションのウェブアプリケーションには AS/400 ユーティリティプログラムのサポートが含まれています。

この表は、このアプリケーションのキー/値パラメータのすべてをまとめたものです。

キー	タイプ	デフォルト値	説明
logging.config	パス	classpath:logback-utility.xml	logback 設定ファイルへの参照用の標準キー。他の標準ロギング記録キーも使用できます。
spring.jta.enabled	ブール値	false	標準キー。データソースサポートモードが static-xa でない場合は、spring JTA トランザクションの自動設定を無効にする必要があります。
spring.datasource.primary.jndi-name	string	jdbc/primary	JNDI を使用する場合の、プライマリデータソースの JNDI 名 (Java Naming And Directory Interface)。
primary.datasource	サブキー付きの標準 spring データソース		JNDI を使用していない場合、アプリケー

キー	タイプ	デフォルト値	説明
+ -driver-class-name + -url + -username + -password			ションデータベースの接続情報が含まれます。モダナイズされたアプリケーションの yml ファイルと同じ設定にする必要があります。 あるいは、xref:../configuration/configuration.adoc[Configuration] で説明されているように、AWS シークレットの使用を強くお勧めします。
encoding	string	ASCII	ユーティリティプログラムで使用されるエンコーディング。有効なエンコーディング CP1047、IBM930、ASCII、U などが必要です。
zonedMode	string	EBCDIC_STRICT	ゾーンデータ型をエンコードまたはデコードするためのモード。許可される値は、EBCDIC_STRICT /EBCDIC_MODIFIED /AS400 です。

キー	タイプ	デフォルト値	説明
commands-off	string		無効にするコマンドのリスト。カンマで区切られています。許可される値は、PGM_BASIC 、RCVMSG、SNDRCVF、CHGV です。既存のプログラムを無効化したり上書きしたりする場合に便利です。PGM_BASIC はデバッグ用に設計された特定の Velocity プログラムです。

gapwalk-hierarchical-support.war

このオプションのウェブアプリケーションには IMS/MFS トランザクションサポートが含まれています。

この表は、このアプリケーションのキー/値パラメータのすべてをまとめたものです。

キー	タイプ	デフォルト値	説明
logging.config	パス	classpath:logback-utility.xml	logback 設定ファイルへの参照用の標準キー。他の標準ログ記録キーも使用できます。
spring.jta.enabled	ブール値	false	標準キー。データソースサポートモードが static-xa でない場合は、spring JTA トランザクションの自

キー	タイプ	デフォルト値	説明
			動設定を無効にする必要があります。
jhdb.configuration.context.encoding	string		JHDB (Java 階層型データベース) のエンコーディング。有効なエンコーディング文字列 CP1047、IBM930、ASCII、U などが必要です。

キー	タイプ	デフォルト値	説明
jhdb.checkpointPersistence	string	なし	チェックポイント永続化モード。許可される値は、none/add/end です。新しいチェックポイントが作成されてレジストリに追加されたときに、チェックポイントを永続化するために add を使用します。サーバーのシャットダウン時にチェックポイントを永続化するために end を使用します。それ以外の値は永続化を無効にします。新しいチェックポイントがレジストリに追加されるたびに、既存のチェックポイントはすべてシリアル化され、ファイルは消去されることに注意してください。ファイル内の既存のデータへの追加ではありません。そのため、チェックポイントの数によっては、パフォーマンスに何らかの影響を与える可能性があります。

Gapwalk アプリケーションの認証の設定

このセクションでは、Cognito、Azure AD、Keycloak などの ID プロバイダー (IdP) を使用して Gapwalk アプリケーションの OAuth2 認証を設定する方法について説明します。

トピック

前提条件

このチュートリアルでは、IdP として Amazon Cognito を使用し、モダナイズされたプロジェクトとして planetDemo を使用します。

他の外部 ID プロバイダーも使用できます。 ClientRegistration 情報は IdP から取得する必要があり、gapwalk 認証に必要です。詳細については、IdP の公式ドキュメントを参照してください。

ClientRegistration 情報：

client-id

の ID は ClientRegistration、この例では「planetDemo」になります。

client-secret

クライアントシークレット。

認証エンドポイント

認証サーバーの認証エンドポイント URI。

トークンエンドポイント

認可サーバーのトークンエンドポイント URI。

jwks エンドポイント

認証サーバーによって発行された JSON ウェブ署名を検証するためのキーを含む JSON ウェブキー (JWK) の取得に使用される URI。

リダイレクト URI

アクセスが許可された場合に認証サーバーがエンドユーザーをリダイレクトする URI。

Amazon Cognito のセットアップ

まず、テストの目的でデプロイした gapwalk アプリケーションで使用する、Amazon Cognito ユーザープールとユーザーを作成して設定します。

> **Note**
>
> 別の IdP を使用している場合は、このステップをスキップできます。

ユーザープールの作成

1. の Amazon Cognito に移動 AWS Management Console し、 AWS 認証情報を使用して認証します。
2. [User Pools] (ユーザープール) を選択します。
3. [Create a user pool] を選択します。
4. [サインインエクスペリエンスを設定] では、[Cognito ユーザープール] のデフォルトプロバイダータイプをそのまま使用します。[Cognito ユーザープールのサインインオプション] を 1 つまたは複数選択できます。ここでは、[ユーザー名] を選択し、[次へ] を選択します。
5. セキュリティ要件の設定 で、MFA なしを選択し、次へ を選択して、デフォルトのままにして多要素認証を無効にします。
6. セキュリティ対策として [自己登録を有効化] を無効にして、[次へ] を選択します。
7. [Cognito で E メールを送信] を選択します。[次へ] を選択します。
8. [アプリケーションを統合] で、ユーザープールの名前を選択します。[ホストされた認証ページ] で、[Cognito でホストされた UI を使用] を選択します。
9. わかりやすくするために、[ドメイン] で [Cognito ドメインを使用する] を選択し、ドメインプレフィックス (例: `https://planetsdemo`) を入力します。デモアプリケーションはクライアントとして追加する必要があります。

    1. [最初のアプリケーションクライアント] で [秘密クライアント] を選択します。アプリケーションクライアント名 (`planetsdemo` など) を入力し、[クライアントのシークレットを生成する] を選択します。
    2. [許可されているコールバック URL] に、認証後にユーザーをリダイレクトする URL を入力します。一時的な http://localhost:8080/planetsdemo URL も使用できます。後で編集することもできます。

3. [高度なアプリケーションクライアントの設定] と [属性の読み込みおよび書き込みアクセス権限] セクションはデフォルト値のままにします。
4. [次へ] を選択します。

10. [確認して作成] で、選択を確認し、[ユーザープールを作成] を選択します。

詳細については、「ユーザープールの作成」を参照してください。

ユーザー作成

自己登録は無効になっているので、Amazon Cognito ユーザーを作成してください。 AWS Management Consoleの Amazon Cognito コンソールに移動します。作成したユーザープールを選択し、[ユーザー] で [ユーザーを作成] を選択します。

[ユーザー情報] で、[E メールで招待を送信] を選択し、ユーザー名とメールアドレスを入力して、[パスワードの生成] を選択します。[ユーザーの作成] を選択します。

Gapwalk アプリケーションでの Amazon Cognito の統合

Amazon Cognito ユーザープールとユーザーの準備ができたら、モダナイズされたアプリケーションの `main-application.yml` ファイルに移動し、次のコードを追加します。

```
spring:
  security:
    oauth2:
      client:
        registration:
          cognito:
            client-id: client-id
            client-name: client-name
            client-secret: client-secret
            provider: cognito
            authorization-grant-type: authorization_code
            scope: openid
            redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"
        provider:
          cognito:
            issuerUri: ${gapwalk-application.security.issuerUri}
            authorization-uri: ${gapwalk-application.security.domainName}/oauth2/
authorize
            jwk-set-uri: ${gapwalk-application.security.issuerUri}/.well-known/
jwks.json
```

```
            token-uri: ${gapwalk-application.security.domainName}/oauth2/token
            user-name-attribute: cognito:username
      resourceserver:
        jwt:
          jwk-set-uri: ${gapwalk-application.security.issuerUri}/.well-known/jwks.json

gapwalk-application.security: enabled
gapwalk-application.security.identity: oauth

gapwalk-application.security.issuerUri: https://cognito-idp.region.amazonaws.com/pool-
id
gapwalk-application.security.domainName: your-cognito-domain
```

説明に従って、次のプレースホルダーを置き換えます。

1. の Amazon Cognito に移動 AWS Management Console し、 AWS 認証情報を使用して認証します。
2. [ユーザープール] を選択して、作成したユーザープールを選択します。*### ID* は、[ユーザープール ID] にあります。
3. を検索できる アプリ統合 *your-cognito-domain*を選択し、アプリクライアントと分析に移動してアプリを選択します。
4. [アプリケーションクライアント: yourApp] では、*#######*、*###### ID*、*############* ([クライアントシークレットを表示]) を確認できます。
5. *##### ID* は、例えば Amazon Cognito ユーザーとユーザープールを作成したリージョン ID に対応します (例: `eu-west-3`)。
6. リダイレクト URI には、認証後にユーザーをリダイレクトする URI を入力します。

Gapwalk アプリケーションを今すぐデプロイし、以前に作成したユーザーを使用してアプリケーションにサインインできます。

> **Note**
>
> ログイン中に「属性に属性「cognito:username」がありません」というエラーが表示された場合は、次の行を削除します： user-name-attribute: cognito:username

AWS Blu Age ランタイム APIs

AWS Blu Age ランタイムは、複数のウェブアプリケーションを使用して REST エンドポイントを公開し、REST クライアントを使用してモダナイズされたアプリケーションを操作する方法 (スケジューラを使用してジョブを呼び出すなど) を提供します。

このドキュメントの目的は、利用可能な REST エンドポイントを一覧表示し、以下の詳細を記載することです。

- ロール
- 適切に使用する方法

エンドポイントのリストは、提供されるサービスの性質とエンドポイントを公開するウェブアプリケーションに応じて、カテゴリ別に整理されています。

POSTMAN、Thunder Client、CURL、ウェブブラウザなどの専用ツールを使用した REST エンドポイントの使用に関する基本的な知識は既にあることを前提としています。または、API コールを行うための独自のコードを書いてください。

トピック

URL の構築

以下の各ウェブアプリケーションは、すべてのエンドポイントで共有されるルートパスを定義しています。各エンドポイントは、独自の専用パスを追加します。パスを連結したものが、使用する URL になります。例えば、Gapwalk-Application の最初のエンドポイントを考えた場合、次のようになります。

- `/gapwalk-application` は、ウェブアプリケーションのルートパス用です。
- `/scripts` は、専用エンドポイントパス用です。

その結果、使用する URL は `http://server:port/gapwalk-application/scripts` になります。

server

サーバー名 (特定のウェブアプリケーションをホストしているサーバー) を指します。

port

サーバーが公開するポートです。

Gapwalk アプリケーション

Gapwalk ウェブアプリケーションのエンドポイントは、ルートパス `/gapwalk-application` を使用します。

トピック

- バッチジョブ (モダナイズされた JCL など) に関連付けられたエンドポイント
- エンドポイントのメトリクス
- その他のエンドポイント
- ジョブキューに関連付けられたエンドポイント

バッチジョブ (モダナイズされた JCL など) に関連付けられたエンドポイント

バッチジョブは、同期または非同期で実行できます (詳細は以下を参照)。バッチジョブは、レガシースクリプト (JCL) のモダナイゼーションの成果である groovy スクリプトを使用して実行されています。

トピック

- デプロイされたスクリプトの一覧表示
- スクリプトを同期で起動する
- スクリプトを非同期で起動する
- トリガーされたスクリプトの一覧表示
- ジョブ実行の詳細を取得する
- 非同期で起動されたスクリプトのうち、強制終了できるスクリプトの一覧表示
- 同期で起動されたスクリプトのうち、強制終了できるスクリプトの一覧表示

- 指定したジョブ実行を強制終了する
- 再開可能な既存のチェックポイントを一覧表示する
- ジョブの再開 (同期)
- ジョブの再開 (非同期)
- 非同期ジョブ実行のスレッド制限の設定

デプロイされたスクリプトの一覧表示

- サポートされているメソッド: GET
- パス: `/scripts`
- 引数: なし
- このエンドポイントは、サーバーにデプロイされている groovy スクリプトのリストを String として返します。このエンドポイントは、結果の String はアクティブリンク (起動可能なスクリプトごとのリンクで、以下の例を参照) 付きの HTML であるため、主にウェブブラウザから使用することを想定しています。

レスポンス例:

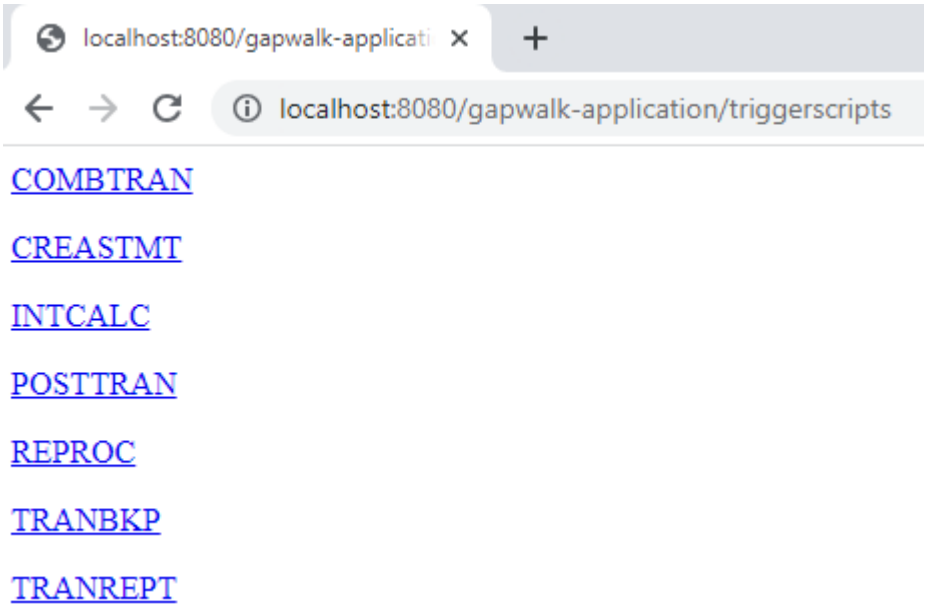
```
<p><a href=./script/COMBTRAN>COMBTRAN</a></p><p><a href=./script/CREASTMT>CREASTMT</
a></p><p><a href=./script/INTCALC>INTCALC</a></p><p><a href=./script/
POSTTRAN>POSTTRAN</a></p><p><a href=./script/REPROC>REPROC</a></p><p><a href=./script/
TRANBKP>TRANBKP</a></p><p><a href=./script/TRANREPT>TRANREPT</a></p><p><a href=./
script/functions>functions</a></p>
```

> **Note**
>
> このリンクは、リストされている各スクリプトを同期的に起動するために使用する URL を表します。

- サポートされているメソッド: GET
- パス: `/triggerscripts`
- 引数: なし
- このエンドポイントは、サーバーにデプロイされている groovy スクリプトのリストを String として返します。このエンドポイントは、結果の String はアクティブリンク (起動可能なスクリプトご

とのリンクで、以下の例を参照) 付きの HTML であるため、主にウェブブラウザから使用することを想定しています。

前のエンドポイントレスポンスとは対照的に、リンクは、リストされた各スクリプトを非同期的に起動するために使用する URL を表します。



スクリプトを同期で起動する

このエンドポイントには GET と POST 用の専用パスを持つ 2 つのバリアントがあります (以下を参照)。

- サポートされているメソッド: GET
- パス: `/script/{scriptId:.+}`
- サポートされているメソッド: POST
- パス: `/post/script/{scriptId:.+}`
- 引数:
  - 起動するスクリプトの識別子
  - オプション: リクエストパラメータ (「`Map<String,String>`」を参照) を使用してスクリプトに渡すパラメータ。指定したパラメータは、呼び出された groovy スクリプトのバインディングに自動的に追加されます。

- この呼び出しは、指定された識別子でスクリプトを起動します。パラメータが指定されている場合は追加のパラメータを使用し、スクリプトの実行が完了するのを待ってから、次のいずれかのメッセージ (String) を返します。
  - 「完了」 (ジョブの実行が順調に実行された場合)。
  - ジョブ実行中に発生した問題の詳細を含む JSON エラーメッセージ。サーバーのログから詳細情報をさらに取得して、ジョブの実行で何が問題だったのかを把握できます。

```
{
    "exitCode": -1,
    "stepName": "STEP15",
    "program": "CBACT04C",
    "status": "Error"
}
```

サーバーログを見ると、これがデプロイメントの問題であることがわかります (想定していたプログラムが正しくデプロイされていないため、それが見つからず、ジョブの実行が失敗しています)。

```
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.s.BatchWebController          - --> executing script INTCALC
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.s.BatchWebController          - Bound jobContext 419695287 - GDGEventsQueueHandler :907380469
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.s.ScriptControlTower          - Added jobExecutor [a65c2791-864f-43c9-972a-b5f2353389e6] to Sync Script Control Tower.
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.j.s.JobExecutor              - a65c2791-864f-43c9-972a-b5f2353389e6 - worker :Thread-26 [1547512424]
2023-06-09 10:27:28 default        INFO  - c.n.b.g.r.j.s.JobExecutor              - Triggered script: INTCALC - [a65c2791-864f-43c9-972a-b5f2353389e6] - jobContext [419695287]
2023-06-09_10-27-29-613 | [JOB] INTCALC - Started
2023-06-09_10-27-29-651 | [STEP] STEP15 - Started
2023-06-09 10:27:29 default        ERROR - c.n.b.g.r.c.i.ExecutionControllerImpl   - Could not find program "CBACT04C" in the program registry.
2023-06-09 10:27:29 default        ERROR - c.n.b.g.r.c.i.ExecutionControllerImpl   - Could not find program "CBACT04C" in the program registry.
2023-06-09_10-27-29-760 | Program not found => not executed !
2023-06-09_10-27-29-761 | [STEP] STEP15 - Ended
2023-06-09_10-27-29-772 | [JOB] INTCALC - Ended
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.j.s.DefaultJobContext         - Job [419695287] - starting final operation
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.j.s.DefaultJobContext         - End of job [419695287]
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.s.ScriptControlTower          - Removed jobExecutor [a65c2791-864f-43c9-972a-b5f2353389e6] from Script Control Tower.
2023-06-09 10:27:29 default        INFO  - c.n.b.g.r.s.ScriptControlTower          - Remaining jobExecutors:0
```

> **Note**
>
> 同期呼び出しは、実行時間が短いジョブ専用にしてください。長時間実行されるジョブは、非同期で起動する必要があります (以下の専用エンドポイントを参照)。

スクリプトを非同期で起動する

- サポートされているメソッド: GET / POST
- パス: /triggerscript/{scriptId:.+}
- 引数:
  - 起動するスクリプトの識別子

- オプション: リクエストパラメータ (「Map<String,String>」を参照) を使用してスクリプトに渡すパラメータ。指定したパラメータは、呼び出された groovy スクリプトの https://docs.groovy-lang.org/latest/html/api/groovy/lang/Binding.html[bindings] に自動的に追加されます。

- 上記の同期モードとは対照的に、エンドポイントはジョブの実行が終了するのを待たずにレスポンスを送信します。使用可能なスレッドが見つかると、直ちにジョブの実行を開始し、ジョブの実行を表す一意の識別子であるジョブ実行 ID を含むレスポンスが呼び出し元に送信されます。これを使用して、ジョブの実行状況を問い合わせたり、誤動作していると思われるジョブの実行を強制終了したりすることができます。レスポンスの形式は次のとおりです。

```
Triggered script <script identifier> [unique job execution id] @ <date and time>
```

- ジョブの非同期実行は、固定の限られた数のスレッドに依存しているため、使用可能なスレッドが見つからない場合、ジョブの実行が開始されない場合があります。この場合、返されるメッセージは次のようになります。

```
Script [<script identifier>] NOT triggered - Thread limit reached (<actual thread
 limit>) - Please retry later or increase thread limit.
```

  スレッド上限を引き上げる方法については、次の settriggerthreadlimit エンドポイントを参照してください。

レスポンス例:

```
Triggered script INTCALC [d43cbf46-4255-4ce2-aac2-79137573a8b4] @ 06-12-2023 16:26:15
```

一意のジョブ実行識別子により、必要に応じてサーバーログ内の関連するログエントリを迅速に取得できます。また、以下に詳述する、他のいくつかのエンドポイントで使用されています。

トリガーされたスクリプトの一覧表示

- サポートされているメソッド: GET
- パス: /triggeredscripts/{status:.+}、/triggeredscripts/{status:.+}/{namefilter}
- 引数:
  - ステータス (必須): 取得するトリガースクリプトのステータス。可能な値は以下のとおりです。
    - all: ジョブが実行中かどうかにかかわらず、ジョブ実行の詳細をすべて表示します。

  - running: 現在実行中のジョブの詳細のみを表示します。
  - done: 実行が終了したジョブの詳細のみを表示します。
  - killed: 専用エンドポイント (以下を参照) を使用して、実行が強制終了されたジョブの詳細のみを表示します。
  - triggered: トリガーされたがまだ起動していないジョブの詳細のみを表示します。
  - failed: 実行が失敗とマークされたジョブの詳細のみを表示します。
  - _namefilter (オプション): 指定したスクリプト識別子の実行のみを取得します。
- ジョブ実行の詳細のコレクションを JSON として返します。詳細については、「Job 実行詳細メッセージの構造」を参照してください。

レスポンス例:

```
[
    {
      "scriptId": "INTCALC",
      "caller": "127.0.0.1",
      "identifier": "d43cbf46-4255-4ce2-aac2-79137573a8b4",
      "startTime": "06-12-2023 16:26:15",
      "endTime": "06-12-2023 16:26:15",
      "status": "DONE",
      "executionResult": "{ \"exitCode\": -1, \"stepName\": \"STEP15\", \"program\":
 \"CBACT04C\", \"status\": \"Error\" }",
      "executionMode": "ASYNCHRONOUS"
    }
 ]
```

ジョブ実行の詳細を取得する

- サポートされているメソッド: GET
- パス: /getjobexecutioninfo/{jobexecutionid:.+}
- 引数:
  - jobexecutionid (必須): 対応するジョブ実行の詳細を取得するために使用する一意のジョブ実行識別子。
- 戻り値: 1 つのジョブ実行の詳細を表す JSON 文字列 (「Job 実行詳細メッセージの構造」を参照)、指定した ID のジョブ実行の詳細が見つからない場合は空のレスポンス。

非同期で起動されたスクリプトのうち、強制終了できるスクリプトの一覧表示

- サポートされているメソッド: GET
- パス: `/killablescripts`
- 非同期で起動され、現在も実行中で強制終了できるジョブのジョブ実行識別子のコレクションを返します (以下の `/kill` エンドポイントを参照)。

同期で起動されたスクリプトのうち、強制終了できるスクリプトの一覧表示

- サポートされているメソッド: GET
- パス: `/killablesyncscripts`
- 同期で起動され、現在も実行中で強制終了できるジョブのジョブ実行識別子のコレクションを返します (以下の `/kill` エンドポイントを参照)。

指定したジョブ実行を強制終了する

- サポートされているメソッド: GET
- パス: `/kill/{identifier:.+}`
- 引数: ジョブ実行識別子 (必須): 強制終了されるジョブ実行を示す、一意のジョブ実行識別子。
- 戻り値: メッセージには、ジョブ実行の強制終了試行の結果の詳細を記述したテキストメッセージとして、スクリプト識別子、一意のジョブ実行識別子、実行強制終了が発生した日時が含まれます。指定した識別子に対して実行中のジョブが見つからなかった場合は、代わりにエラーメッセージが返されます。

⚠ Warning

- ランタイムでは、対象のジョブ実行が確実に強制終了するように最善を尽くします。したがって、/kill エンドポイントからの応答は、 AWS Blu Age ランタイムがジョブを強制終了することによるビジネスへの影響を最小限に抑えようとするため、発信者に到達するまでに少し時間がかかる場合があります。
- ジョブの実行を強制終了することは、データの損失や破損の可能性など、ビジネスに直接的な影響をもたらす可能性があるため、安易に実行するものではありません。特定のジョブの実行に不具合があり、データ修復手段が明確に特定されている場合に限って使用する必要があります。

- ジョブを強制終了させた場合、詳細な調査 (事後分析) を行って何が問題であったかを特定し、適切な是正措置を講じる必要があります。
- どのような場合でも、実行中のジョブを強制終了しようとすると、警告レベルのメッセージとともにサーバーログに記録されます。

再開可能な既存のチェックポイントを一覧表示する

ジョブを再開できるかどうかは、スクリプトが CheckpointRegistry にチェックポイントを登録して、ジョブ実行の進行状況を追跡できるかどうかで決まります。ジョブの実行が正しく終了しない場合、再開チェックポイントが登録されていれば、最後に登録されたことが判明しているチェックポイントからジョブ実行を再開できます (チェックポイントより上のステップを実行する必要はありません)。

- サポートされているメソッド: GET
- パス: `/restarts`
- ジョブ実行が正常に終了しなかったジョブを再開するために使用できる既存のリスタートポイントのリストを HTML ページとして返します。どのスクリプトでもチェックポイントが登録されていない場合、ページの内容は「チェックポイントが登録されていません」になります。

ジョブの再開 (同期)

- サポートされているメソッド: GET
- パス: `/restart/{hashcode}`
- 引数: hashcode (整数 - 必須): 指定されたハッシュコードをチェックポイント値として使用して、以前に中止されたジョブの実行を再開します (有効なチェックポイント値を取得する方法については、上記の `/restarts` エンドポイントを参照してください)。
- 戻り値: 上記の `script` の戻り値の説明を参照してください。

ジョブの再開 (非同期)

- サポートされているメソッド: GET
- パス: `/triggerrestart/{hashcode}`

- 引数: hashcode (整数 - 必須): 指定されたハッシュコードをチェックポイント値として使用して、以前に中止されたジョブの実行を再開します (有効なチェックポイント値を取得する方法については、上記の /restarts エンドポイントを参照してください)。
- 戻り値: 上記の triggerscript の戻り値の説明を参照してください。

非同期ジョブ実行のスレッド制限の設定

ジョブの非同期実行は、JVM 内のスレッドの専用プールに依存します。そのプールには、使用可能なスレッド数に関する一定の制限があります。ユーザーは、ホストの機能 (CPU 数、使用可能なメモリなど) に応じて制限を調整できます。デフォルトでは、スレッド制限は 5 スレッドに設定されています。

- サポートされているメソッド: GET
- パス: /settriggerthreadlimit/{threadlimit:.+}
- 引数 (整数): 適用する新しいスレッド制限。厳密な正の整数である必要があります。
- 指定したスレッド制限値が有効でない (厳密な正の整数ではない) 場合は、新しいスレッド制限と直前のスレッド制限を示すメッセージ (String) を返すか、エラーメッセージを返します。

レスポンス例:

```
Set thread limit for Script Tower Control to 10 (previous value was 5)
```

現在実行中のトリガーされたジョブの実行回数のカウント

- サポートされているメソッド: GET
- パス: /countrunningtriggeredscripts
- 非同期で起動された実行中のジョブの数とスレッドの上限 (同時に実行できるトリガーされたジョブの最大数) を示すメッセージを返します。

レスポンス例:

```
0 triggered script(s) running (limit =10)
```

> **Note**
>
> これを使用して、ジョブを起動する前に、スレッドの上限に達していないかどうかを確認できます (ジョブが起動できなくなる)。

ジョブ実行情報を消去する

ジョブ実行情報は、サーバーが稼働している限り、サーバーのメモリに残ります。古くなった情報は関連性がないので、メモリから消去すると便利な場合があります。これが、このエンドポイントの目的です。

- サポートされているメソッド: GET
- パス: `/purgejobinformation/{age:.+}`
- 引数: 消去する情報の経過時間 (時間単位) を表す、厳密な正の整数値。
- 次の情報を含むメッセージを返します。
  - アーカイブ目的で消去されたジョブ実行情報が保存される消去ファイル名。
  - 消去されたジョブ実行情報の数。
  - メモに残っているジョブ実行情報の数

エンドポイントのメトリクス

JVM

このエンドポイントは、JVM に関連する利用可能なメトリクスを返します。

- サポートされているメソッド: GET
- パス: `/metrics/jvm`
- 引数: なし
- 次の情報を含むメッセージを返します。
  - threadActiveCount: アクティブなスレッドの数。
  - jvmMemoryUsed: Java 仮想マシンによってアクティブに使用されているメモリ。
  - jvmMemoryMax: Java 仮想マシンに許可される最大メモリ。
  - jvmMemoryFree: Java 仮想マシンで現在使用されていない使用可能なメモリ。

セッション

このエンドポイントは、現在開いている HTTP セッションに関連するメトリクスを返します。

- サポートされているメソッド: GET
- パス: `/metrics/session`
- 引数: なし
- 次の情報を含むメッセージを返します。
  - sessionCount: 現在サーバーが管理しているアクティブなユーザーセッションの数。

バッチ

- サポートされているメソッド: GET
- パス: `/metrics/batch`
- 引数:
  - startTimestamp (オプション、数値): データのフィルタリングの開始時のタイムスタンプ。
  - endTimestamp (オプション、数値): データのフィルタリングの終了時のタイムスタンプ。
  - page (オプション、番号): ページ分割用のページ番号。
  - pageSize (オプション、数値): ページ分割の 1 ページあたりの項目数。
- 次の情報を含むメッセージを返します。
  - content: バッチ実行メトリクスのリスト。
  - pageNumber: ページ分割の現在のページ番号。
  - pageSize: 1 ページあたりに表示されるアイテムの数。
  - totalPages: 利用可能なページの合計。
  - numberOfElements: 現在のページの項目数。
  - last: 最終ページのブール型フラグ。
  - first: 最初のページのブール値フラグ。

トランザクション

- サポートされているメソッド: GET
- パス: `/metrics/transaction`
- 引数:

  - startTimestamp (オプション、数値): データのフィルタリングの開始時のタイムスタンプ。
  - endTimestamp (オプション、数値): データのフィルタリングの終了時のタイムスタンプ。
  - page (オプション、番号): ページ分割用のページ番号。
  - pageSize (オプション、数値): ページ分割の 1 ページあたりの項目数。
- 次の情報を含むメッセージを返します。
  - content: 移行する実行メトリクスのリスト。
  - pageNumber: ページ分割の現在のページ番号。
  - pageSize: 1 ページあたりに表示されるアイテムの数。
  - totalPages: 利用可能なページの合計。
  - numberOfElements: 現在のページの項目数。
  - last: 最終ページのブール型フラグ。
  - first: 最初のページのブール値フラグ。

その他のエンドポイント

これらのエンドポイントを使用して、登録されているプログラムやサービスの一覧表示、ヘルスステータスの検出、JICS トランザクションの管理を行います。

トピック

- 登録済みプログラムの一覧表示
- 登録済みサービスの一覧表示
- [Health status] (ヘルスステータス)
- 使用可能な JICS トランザクションの一覧表示
- JICS トランザクションの起動
- JICS トランザクションの起動 (代替)

登録済みプログラムの一覧表示

- サポートされているメソッド: GET
- パス: `/programs`
- 登録されているプログラムのリストを HTML ページとして返します。各プログラムはメインプログラムの識別子によって指定されます。モダナイズされたレガシープログラムとユーティリティ

プログラム (IDCAMS、IEBGENER など) の両方がリストで返されます。利用可能なユーティリティプログラムは、tomcat サーバーにデプロイされているユーティリティウェブアプリケーションによって異なることに注意してください。例えば、モダナイズされた iSeries アセットには z/OS ユーティリティサポートプログラムは関連性がないため利用できない場合があります。

登録済みサービスの一覧表示

- サポートされているメソッド: GET
- パス: `/services`
- 登録されているランタイムサービスのリストを HTML ページとして返します。指定されたサービスは AWS Blu Age ランタイムによってユーティリティとして持ち込まれ、groovy スクリプトのインスタンスに使用できます。Blusam ロードサービス (レガシーデータセットから Blusam データセットを作成する) は、そのカテゴリに分類されます。

レスポンス例:

```
<p>BluesamESDSFileLoader</p><p>BluesamKSDSFileLoader</p><p>BluesamRRDSFileLoader</p>
```

[Health status] (ヘルスステータス)

- サポートされているメソッド: GET
- パス: `/`
- gapwalk-application が起動して実行中であることを示す簡単なメッセージを返します (`Jics application is running.`)

使用可能な JICS トランザクションの一覧表示

- サポートされているメソッド: GET
- パス: `/transactions`
- 使用可能なすべての JICS トランザクションを一覧表示する HTML ページを返します。これは、JICS 要素 (レガシー CICS 要素のモダナイゼーション) のある環境でのみ提供されます。

レスポンス例:

```
<p>INQ1</p><p>MENU</p><p>MNT2</p><p>ORD1</p><p>PRNT</p>
```

JICS トランザクションの起動

- サポートされているメソッド: GET、POST
- パス: `/jicstransrunner/{jtrans:.+}`
- 引数:
  - JICS トランザクション識別子 (文字列、必須): 起動する JICS トランザクションの識別子 (最大 8 文字)
  - 必須: トランザクションに Map<String,Object> として渡す追加の入力データ。このマップの内容は、JICS トランザクションによって消費される COMMAREA をフィードするために使用されます。トランザクションの実行に必要なデータがない場合は、マップを空にしても構いません。
  - オプション: 特定のトランザクションの実行環境をカスタマイズする HTTP ヘッダーエントリ。次のヘッダーキーがサポートされています。
    - `jics-channel`: このトランザクションの起動によって起動されるプログラムが使用する JICS CHANNEL の名前。
    - `jics-container`: この JICS トランザクションの起動に使用される JICS コンテナの名前。
    - `jics-startcode`: JICS トランザクションの開始時に使用する STARTCODE (文字列、最大 2 文字)。指定できる値については、「STARTCODE」を参照してください (ページの下を参照)。
    - `jicxa-xid`: 呼び出し元によって開始され、現在の JICS トランザクション起動が参加する「グローバルトランザクション」(XA) の XID (X/Open トランザクション識別子 XID 構造)。
- 戻り値: JICS トランザクション起動の結果を表す `com.netfective.bluage.gapwalk.rt.shared.web.TransactionResultBean` JSON のシリアル化。

構造の詳細については、「トランザクション起動結果の構造」を参照してください。

JICS トランザクションの起動 (代替)

- サポートされているメソッド: GET、POST
- パス: `/jicstransaction/{jtrans:.+}`
- 引数:

  JICS トランザクション ID (文字列、必須)

  起動する JICS トランザクションの識別子 (最大 8 文字)

必須: トランザクションに Map<String,Object> として渡す追加の入力データ

このマップの内容は、JICS トランザクションによって消費される COMMAREA をフィードするために使用されます。トランザクションの実行に必要なデータがない場合は、マップを空にしても構いません。

オプション: 特定のトランザクションの実行環境をカスタマイズする HTTP ヘッダーエントリ。

次のヘッダーキーがサポートされています。

- `jics-channel`: このトランザクションの起動によって起動されるプログラムが使用する JICS CHANNEL の名前。
- `jics-container`: この JICS トランザクションの起動に使用される JICS コンテナの名前。
- `jics-startcode`: JICS トランザクションの開始時に使用する STARTCODE (文字列、最大 2 文字)。指定できる値については、「STARTCODE」を参照してください (ページの下を参照してください)。
- `jicxa-xid`: 呼び出し元によって開始され、現在の JICS トランザクション起動が参加する「グローバルトランザクション」(XA) の XID (X/Open トランザクション識別子 XID 構造)。

- 戻り値: JICS トランザクション起動の結果を表す `com.netfective.bluage.gapwalk.rt.shared.web.RecordHolderBean` JSON のシリアル化。構造の詳細は、「トランザクション起動レコードの構造」を参照してください。

ジョブキューに関連付けられたエンドポイント

ジョブキューは、AS400 ジョブ送信メカニズムの AWS Blu Age サポートです。ジョブキューは、AS400 で特定のスレッドプールでジョブを実行するために使用されます。ジョブキューは、名前と、そのキューで同時に実行できるプログラムの最大数に対応する最大スレッド数によって定義されます。最大スレッド数よりも多くのジョブがキューに送信された場合、ジョブはスレッドが使用可能になるまで待機します。

キューのジョブのステータスを網羅したリストについては、「キューでジョブの可能性のあるステータス」を参照してください。

ジョブキューのオペレーションは、次の専用エンドポイントによって処理されます。これらのオペレーションは、次のルート URL を使用して Gapwalk アプリケーション URL から呼び出すことができます: `http://server:port/gapwalk-application/jobqueue`。

トピック

- 利用可能なキューを一覧表示する

- ジョブキューの開始または再開
- 送信してジョブを起動する
- スケジュールされたジョブをすべて一覧表示する
- 「保留中」になっているすべてのジョブを一覧表示する
- アクティブなすべてのジョブを一覧表示する
- 起動待ちのすべてのジョブを一覧表示します。
- 「保留中」になっているすべてのジョブをリリースする
- 特定のジョブ名で「保留中」になっているジョブをすべてリリースする
- 特定のジョブ番号のジョブをリリースする

利用可能なキューを一覧表示する

- サポートされているメソッド: GET
- パス: `list-queues`
- 利用可能なキューのリストを、そのステータスとともにキー値の JSON リストとして返します。

レスポンス例:

```
{"Default":"STAND_BY","queue1":"STARTED","queue2":"STARTED"}
```

可能性のあるジョブキューのステータスは次のとおりです。

STAND_BY

ジョブキューは開始を待っています。

開始

ジョブキューは実行中です。

UNKNOWN

ジョブキューのステータスを判断できません。

ジョブキューの開始または再開

- サポートされているメソッド: POST

- パス: `/restart/{name}`
- 引数: 開始/再開するキューの名前 (String) - 必須。
- エンドポイントは何も返さず、開始/再開オペレーションの結果を表示するのに http ステータスに依存しています。

  HTTP 200

  開始/再起動オペレーションは正常に実行されました。指定したジョブキューが開始されました。

  HTTP 404

  ジョブキューが存在しません。

  HTTP 503

  起動/再起動の試行中に例外が発生しました (サーバーログを調査して原因を特定する必要があります)。

送信してジョブを起動する

- サポートされているメソッド: POST
- 引数: リクエスト本文に `com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage` オブジェクトの JSON のシリアル化が必須。詳細については、「ジョブ入力の送信」を参照してください。
- 戻り値: オリジナルの `SubmitJobMessage` と、ジョブが送信されたかどうかを示すログを含む JSON。

スケジュールされたジョブをすべて一覧表示する

- サポートされているメソッド: GET
- パス: `list-jobs`
- 戻り値: スケジュールされたすべてのジョブのリスト (JSON 文字列)。レスポンス例については、「スケジューリングされているジョブのレスポンス一覧」を参照してください。

「保留中」になっているすべてのジョブを一覧表示する

- サポートされているメソッド: GET

- パス: `list-jobs-hold`
- 戻り値: スケジュールされたすべてのジョブのリスト (JSON 文字列)。レスポンス例については、「「保留中」のジョブのレスポンス一覧」を参照してください。

アクティブなすべてのジョブを一覧表示する

- サポートされているメソッド: GET
- パス: `list-jobs-active`
- 戻り値: ステータスが [ACTIVE] のすべてのジョブのリストを JSON 文字列として返します。レスポンスは、スケジューリングされているジョブのレスポンス一覧 からのものと構造が似ています。

起動待ちのすべてのジョブを一覧表示します。

- サポートされているメソッド: GET
- パス: `list-jobs-waiting`
- 戻り値: EXECUTION_WAIT (スレッドが起動可能状態になるまで待機しているジョブ) というステータスのすべてのジョブのリストを JSON 文字列で返します。レスポンスは、the section called "スケジューリングされているジョブのレスポンス一覧" からのものと構造が似ています。

「保留中」になっているすべてのジョブをリリースする

- サポートされているメソッド: POST
- パス: `release-all`
- 戻り値: リリース試行オペレーションの結果を示すメッセージ。次の 2 つのケースが考えられます。
  - HTTP 200 と「すべてのジョブが正常にリリースされました」というメッセージ すべてのジョブが正常にリリースされた場合。
  - HTTP 503 と「ジョブはリリースされていません」というメッセージ。不明なエラーが発生しました。リリース試行で問題が発生した場合、詳細については ログを参照してください。

特定のジョブ名で「保留中」になっているジョブをすべてリリースする

指定されたジョブ名に対して、異なるジョブ番号を持つ複数のジョブを送信できます (ジョブ実行の単一性は、<job name, job number> の組み合わせで付与されます)。エンドポイントは、「保留中」になっている、指定されたジョブ名を持つすべてのジョブ送信のリリースを試行します。

- サポートされているメソッド: POST
- パス: `/release/{name}`
- 引数: 検索するジョブ名 (文字列)。必須。
- 戻り値: リリース試行オペレーションの結果を示すメッセージ。次の 2 つのケースが考えられます。
  - HTTP 200 と「group <name> (<number of released jobs>) 内のジョブは正常にリリースされました」というメッセージ。ジョブは正常にリリースされました。
  - HTTP 503 と「group <name> のジョブはリリースされていません」というメッセージ。不明なエラーが発生しました。リリース試行で問題が発生した場合、詳細については口グを参照してください。

特定のジョブ番号のジョブをリリースする

エンドポイントは、指定された組み合わせ <job name, job number> に対し、一意のジョブ送信のリリースを試行します。

- サポートされているメソッド: POST
- パス: `/release/{name}/{number}`
- 引数:

  name

  検索するジョブ名 (文字列)。必須。

  数値

  検索するジョブ番号 (整数)。必須。

  が返す

  リリース試行オペレーションの結果を示すメッセージ。次の 2 つのケースが考えられます。

  - HTTP 200 と「Job <name/number> が正常にリリースされました」というメッセージ ジョブが正常にリリースされたかどうか。

- HTTP 503 と「Job <name/number> はリリースされていません」というメッセージ 不明なエラーが発生しました。リリース試行で問題が発生した場合、詳細についてはログを参照してください。

Blusam アプリケーションコンソール REST エンドポイント

Blusam アプリケーションコンソールは、モダナイズされた VSAM データセットの管理を簡素化するために設計された API です。Blusam ウェブアプリケーションのエンドポイントは、ルートパス /bac を使用します。

トピック

- エンドポイントに関連付けられたデータセット
- エンドポイントに関連付けられたバルクデータセット
- レコード
- マスク
- その他
- [ユーザー]

エンドポイントに関連付けられたデータセット

次のエンドポイントを使用して、特定のデータセットを作成または管理します。

トピック

- データセットの作成
- ファイルのアップロード
- データセットのロード
- Amazon S3 バケットからデータをロードする
- データセットを Amazon S3 バケットにエクスポートする
- データセットの消去
- データセットの削除
- データセットのレコード数のカウント

データセットの作成

データセットの作成エンドポイントでは、データセット定義を作成できますが、認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/createDataSet`
- 引数:

  name

  (必須、文字列): データセットの名前。

  type

  (必須、文字列): データセットタイプ。可能な値は、ESDS、KSDS、RRDS です。

  recordSize

  (オプション、文字列): データセットの各レコードの最大サイズ。

  fixedLength

  (オプション、ブール値): レコード長が固定かどうかを示します。

  compression

  (オプション、ブール値): データセットが圧縮されているかどうかを示します。

  cacheEnable

  (オプション、ブール値): データセットのキャッシュが有効になっているかどうかを示します。

  alternativeKeys

  (オプション、キーのリスト):
  - offset (必須、数値)
  - length (必須、数値)
  - name (必須、数値)
- 新規に作成されたデータセットを表す JSON ファイルを返します。

リクエスト例:

```
POST /api/services/rest/bluesamservice/createDataSet
{
```

```
  "name": "DATASET",
  "checked": false,
  "records": [],
  "primaryKey": {
    "name": "PK"
  },
  "alternativeKeys": [
    {
      "offset": 10,
      "length": 10,
      "name": "ALTK_0"
    }
  ],
  "type": "ESDS",
  "recordSize": 10,
  "compression": true,
  "cacheEnable": true
}
```

レスポンス例:

```
{
    "dataSet": {
      "name": "DATASET",
      "checked": false,
      "nbRecords": 0,
      "keyLength": -1,
      "recordSize": 10,
      "compression": false,
      "fixLength": true,
      "type": "ESDS",
      "cacheEnable": false,
      "cacheWarmup": false,
      "cacheEviction": "100ms",
      "creationDate": 1686744961234,
      "modificationDate": 1686744961234,
      "records": [],
      "primaryKey": {
        "name": "PK",
        "offset": null,
        "length": null,
        "columns": null,
        "unique": true
```

```
      },
      "alternativeKeys": [
        {
          "offset": 10,
          "length": 10,
          "name": "ALTK_0"
        }
      ],
      "readLimit": 0,
      "readEncoding": null,
      "initCharacter": null,
      "defaultCharacter": null,
      "blankCharacter": null,
      "strictZoned": null,
      "decimalSeparator": null,
      "currencySign": null,
      "pictureCurrencySign": null
    },
    "message": null,
    "result": true
  }
```

ファイルのアップロード

このエンドポイントにより、サーバーにファイルをアップロードできます。ファイルは、特定の各ユーザーに対応する一時フォルダに保存されます。このエンドポイントは、ファイルのアップロードが必要になるたびに使用する必要があります。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/upload`
- 引数:

  file

  (必須、multipart/form-data): アップロードするファイル。
- アップロードのステータスを反映するプール値を返します。

データセットのロード

前述のデータセット作成エンドポイントを使用してデータセット定義を作成すると、アップロードされたファイルに関連するレコードを特定のデータセットにロードできます。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/loadDataSet`
- 引数:

  name

  (必須、文字列): データセットの名前。
- 戻り値: リクエストのステータスと、ロードされたデータセット。

Amazon S3 バケットからデータをロードする

listcat ファイルを使用して、Amazon S3 バケットからデータセットをロードします。

- サポートされているメソッド: GET
- パス: `` `/api/services/rest/bluesamservice/loadDataSetFromS3` ``
- 引数:

  listcatFileS3Location

  (必須、文字列): listcat ファイルの Amazon S3 の場所。

  datasetFileS3Location

  (必須、文字列): データセットファイルの Amazon S3 の場所。

  region

  (必須、文字列): ファイルが保存されている Amazon S3 AWS リージョン 。
- 新規に作成されたデータセットを返します

リクエスト例:

```
/BAC/api/services/rest/bluesamservice/loadDataSetFromS3?region=us-
east-1&listcatFileS3Location=s3://bucket-name/listcat.json&datasetFileS3Location=s3://
bucket-name/dataset.DAT
```

データセットを Amazon S3 バケットにエクスポートする

データセットを指定した Amazon S3 バケットにエクスポートします。

- サポートされているメソッド: GET

- パス: /api/services/rest/bluesamservice/exportDataSetToS3
- 引数:

  S3 の場所

  (必須、文字列): データセットをエクスポートする Amazon S3 の場所。

  datasetName

  (必須、文字列): エクスポートするデータセットの名前。

  region

  (必須、文字列): Amazon S3 バケット AWS リージョン の 。 Amazon S3
- エクスポートされたデータセットを返します。

リクエスト例:

```
/BAC/api/services/rest/bluesamservice/exportDataSetToS3?region=eu-
west-1&s3Location=s3://bucket-name/dump&datasetName=dataset
```

データセットの消去

データセットからすべてのレコードを消去します。認証が必要です。

- サポートされているメソッド: POST
- パス: /api/services/rest/bluesamservice/clearDataSet
- 引数:

  name

  (必須、文字列): 消去するデータセットの名前。
- リクエストのステータスを返します。

データセットの削除

データセット定義とレコードを削除します。認証が必要です。

- サポートされているメソッド: POST
- パス: /api/services/rest/bluesamservice/deleteDataSet
- 引数:

name

(必須、文字列): 削除するデータセットの名前。

- 戻り値: リクエストのステータスと、削除されたデータセット。

データセットのレコード数のカウント

このエンドポイントは、データセットに関連付けられたレコード数を返します。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/countRecords`
- 引数:

name

(必須、文字列): データセットの名前。

- 戻り値: レコード数

エンドポイントに関連付けられたバルクデータセット

次のエンドポイントを使用して、一度に複数のデータセットを作成または管理します。

トピック

- データセットのエクスポート
- 複数のデータセットの作成
- すべてのデータセットのリスト
- すべてのデータセットの直接の一覧表示
- すべてのデータセットの削除
- listcat ファイルからデータセット定義を取得する
- アップロードした listcat ファイルからデータセット定義を取得する
- JSON ファイルから listcat をロードする

データセットのエクスポート

- サポートされているメソッド: GET
- パス: `/api/services/rest/bluesamservice/exportDataSet`

- 引数:

  name

  (必須、文字列): 削除するデータセットの名前。

  datasetOutputFile

  (オプション、文字列): エクスポートされたデータセットをサーバーに保存するパス

  rdw

  (オプション、ブール値): RDW フィールドをエクスポートする必要がある場合。
- リクエストのステータスと、エクスポートされたデータセットを含むファイルを返します。

複数のデータセットの作成

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/createAllDataSets`
- 引数:
  - データセットのリスト

    name

    (必須、文字列): データセットの名前。

    type

    (必須、文字列): データセットタイプ。可能な値は、ESDS、KSDS、RRDS です。

    recordSize

    (オプション、文字列): データセットの各レコードの最大サイズ。

    fixedLength

    (オプション、ブール値): レコード長が固定かどうかを示します。

    compression

    (オプション、ブール値): データセットが圧縮されているかどうかを示します。

    cacheEnable

    (オプション、ブール値): データセットのキャッシュが有効になっているかどうかを示します。

- 戻り値: リクエストのステータスと、新規に作成されたデータセット。

すべてのデータセットのリスト

- サポートされているメソッド: GET
- パス: `/api/services/rest/bluesamservice/listDataSet`
- 引数: なし
- 戻り値: リクエストのステータスと、データセットのリスト。

すべてのデータセットの直接の一覧表示

- サポートされているメソッド: GET
- パス: `/api/services/rest/bluesamservice/directListDataSet`
- 引数: なし
- 戻り値: リクエストのステータスと、データセットのリスト。

すべてのデータセットの削除

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/removeAll`
- 引数: なし
- 戻り値: リクエストのステータスを表すブール値。

listcat ファイルからデータセット定義を取得する

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/getDataSetsDefinitionFromListcat`
- 引数:

  paramFilePath

  (必須、文字列): listcat ファイルへのパス。
- 戻り値: データセットのリスト

アップロードした listcat ファイルからデータセット定義を取得する

- サポートされているメソッド: POST
- パス: `/api/services/rest/bluesamservice/getDataSetsDefinitionFromUploadedListcat`
- 引数: なし
- 戻り値: データセットのリスト

JSON ファイルから listcat をロードする

- サポートされているメソッド: GET
- パス: `/api/services/rest/bluesamservice/loadListcatFromJsonFile`
- 引数:

  filePath

  (必須、文字列): listcat ファイルへのパス。
- 戻り値: データセットのリスト

レコード

次のエンドポイントを使用して、データセット内のレコードを作成または管理します。

トピック

- レコードを作成する
- データセットの読み込み
- レコードを削除する
- レコードの更新
- レコードの保存

レコードを作成する

このエンドポイントでは、新しいレコードを作成できます。認証が必要です。

- サポートされているメソッド: POST

- パス: `/api/services/rest/crud/createRecord`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト

  マスク

  (必須、マスク): マスクオブジェクト。
- 戻り値: リクエストのステータスと、作成されたレコード。

データセットの読み込み

このエンドポイントは、データセットを読み込むことができます。

- サポートされているメソッド: GET
- パス: `/api/services/rest/crud/readDataSet`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト。
- 戻り値: リクエストのステータスと、レコードを含むデータセット。

レコードを削除する

このエンドポイントは、データセットからレコードを削除できます。認証が必要です。

- サポートされているメソッド: DELETE
- パス: `/api/services/rest/crud/deleteRecord`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト

  record

  (必須、レコード): 削除するレコード
- 削除のステータスを返します。

レコードの更新

このエンドポイントでは、データセットに関連付けられたレコードを更新できます。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/crud/updateRecord`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト

  record

  (必須、レコード): 更新するレコード
- 戻り値: リクエストのステータスと、レコードを含むデータセット。

レコードの保存

このエンドポイントでは、マスクを使用してレコードをデータセットに保存できます。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/crud/saveRecord`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト

  record

  (必須、レコード): 保存するレコード
- 戻り値: リクエストのステータスと、レコードを含むデータセット。

マスク

次の以下のエンドポイントを使用して、データセットにマスクをロードまたは適用します。

トピック

- マスクのロード
- マスクの適用
- マスクフィルタの適用

マスクのロード

このエンドポイントは、特定のデータセットに関連するすべてのマスクを取得できます。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/crud/loadMasks`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト
- 戻り値: リクエストのステータスと、マスクのリスト。

マスクの適用

このエンドポイントでは、特定のデータセットにマスクを適用できます。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/crud/applyMask`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト

  マスク

  (必須、マスク): データセットオブジェクト
- 戻り値: リクエストのステータスと、適用したマスクを含むデータセット。

マスクフィルタの適用

このエンドポイントでは、特定のデータセットにマスクとフィルタを適用できます。認証が必要です。

- サポートされているメソッド: POST
- パス: `/api/services/rest/crud/applyMaskFilter`
- 引数:

  データセット

  (必須 DataSet): データセットオブジェクト

  マスク

  (必須、マスク): データセットオブジェクト
- 戻り値: リクエストのステータスと、適用したマスクとフィルタを含むデータセット。

その他

次のエンドポイントを使用して、データセットのキャッシュの管理や、データセットの特性の確認を行います。

トピック

ウォームアップキャッシュの確認

特定のデータセットに対して、ウォームアップキャッシュが有効になっているかどうかを確認します。認証が必要です。

- サポートされているメソッド: POST
- パス: `` `/api/services/rest/bluesamservice/warmupCache` ``
- 引数:

name

(必須、文字列): データセットの名前。

- 戻り値: ウォームアップキャッシュが有効になっている場合は「true」、それ以外の場合は「false」。

チェックキャッシュの有効化

特定のデータセットに対して、キャッシュが有効になっているかどうかを確認します。認証が必要です。

- サポートされているメソッド: GET
- パス: `/api/services/rest/bluesamservice/isEnableCache`
- 引数: なし
- キャッシュが有効になっている場合は「true」を返します。

キャッシュの有効化

- サポートされているメソッド: GET
- パス: `/api/services/rest/bluesamservice/enableDisableCache/{enable}`
- 引数:

  有効化

  (必須、プール値):「true」に設定すると、キャッシュが有効になります。
- 戻り値: なし

割り当てられた RAM キャッシュの確認

このエンドポイントは、割り当てられた RAM キャッシュメモリを取得できます。認証が必要です。

- サポートされているメソッド: GET
- パス: `/api/services/rest/bluesamservice/allocatedRamCache`
- 引数: なし
- 戻り値: メモリのサイズ (文字列)

永続性の確認

- サポートされているメソッド: GET
- パス: /api/services/rest/bluesamservice/persistence
- 引数: なし
- 戻り値: 文字列として使用される永続性

サポートされているデータセットタイプの確認

- パス: /api/services/rest/bluesamservice/getDataSetTypes
- 引数: なし
- 戻り値: サポートされているデータセットタイプのリスト (文字列のリスト)。

サーバーヘルスの確認

- サポートされているメソッド: GET
- パス: /api/services/rest/bluesamservice/serverIsUp
- 引数: なし
- 戻り値: なし

[ユーザー]

次のエンドポイントを使用して、ユーザーインタラクションを管理します。

トピック

- 「ログイン」
- ユーザーアカウントの確認
- サインオン
- すべてのユーザーを一覧表示する
- Logout

「ログイン」

- サポートされているメソッド: POST

- パス: /api/services/security/servicelogin/login
- 引数:

  username

  (必須、文字列)。

  password

  (必須、文字列)。
- ログイン中のユーザーのユーザー名とロールを返します。

レスポンス例

```
{"login":"some-user","roles":[{"id":0,"roleName":"ROLE_ADMIN"}]}
```

ユーザーアカウントの確認

- サポートされているメソッド: POST
- パス: /api/services/security/servicelogin/hasAccount
- 引数: なし
- 戻り値: ユーザーが既にログインしている場合は「true」

サインオン

- サポートされているメソッド: POST
- パス: /api/services/security/servicelogin/recorduser
- 引数: なし
- 戻り値: ユーザーが既にログインしている場合は「true」

すべてのユーザーを一覧表示する

- サポートされているメソッド: GET
- パス: /api/services/security/servicelogin/listusers
- 引数: なし
- 戻り値: すべてのユーザーのリスト

Logout

- サポートされているメソッド: POST
- パス: `/api/services/security/servicelogout/logout`
- 引数: なし
- 戻り値: ユーザーが正常にログアウトした場合は「true」。

JICS アプリケーションコンソール

JICS コンポーネントは、従来の CICS リソースのモダナイゼーションに対する AWS Blu Age サポートです。JICS アプリケーションコンソールウェブアプリケーションは、JICS リソースの管理専用です。次のエンドポイントを使用すると、JAC ユーザーインターフェイスを操作しなくても管理タスクを実行できます。エンドポイントが認証を要求する場合は、必ず、リクエストには認証の詳細 (通常は基本的な認証で要求されるユーザー名/パスワード) を含める必要があります。JICS アプリケーションコンソールウェブアプリケーションのエンドポイントは、ルートパス を使用します`/jac/`。

トピック

- JICS リソースの管理
- JAC ユーザー管理エンドポイント

JICS リソースの管理

次のエンドポイントは、すべて JICS リソース管理に関連するものであり、JICS 管理者は日常的にリソースを扱うことができます。

トピック

- JICS LISTS と GROUPS の一覧表示
- JICS GROUPS の一覧表示
- 特定の LIST の JICS GROUPS の一覧表示
- 特定の GROUP の JICS リソースの一覧表示
- 指定された GROUP の LIST JICS (名前の使用による代替)
- 複数の LISTS の所有 GROUPS の編集
- LIST の削除
- GROUP の削除

- TRANSACTION の削除
- PROGRAM の削除
- FILE の削除
- TDQUEUE の削除
- TSMODEL の削除
- LIST の作成
- GROUP の作成
- RESOURCES 作成に関する一般的な考慮事項
- TRANSACTION の作成
- PROGRAM の作成
- FILE の作成
- TDQUEUE の作成
- TSMODEL の作成
- LIST の更新
- GROUP の更新
- RESOURCES の更新に関する一般的な考慮事項
- TRANSACTION の更新
- PROGRAM の更新
- FILE の更新
- TDQUEUE の更新
- TSMODEL の更新

JICS LISTS と GROUPS の一覧表示

LIST と GROUPS は、JICS コンポーネント内の主要な所有コンテナリソースです。すべての JICS リソースは GROUP に属している必要があります。グループは LISTS に帰属できますが、必須ではありません。LISTS は特定の JICS 環境には存在しない場合もありますが、ほとんどの場合、LISTS はリソースを整理するためのものです。CICS リソース組織についての詳細は、「CICS リソース」を参照してください。

- サポートされているメソッド: GET

- 認証が必要
- パス: `/api/services/rest/jicsservice/listJicsListsAndGroups`
- 戻り値: LISTS と GROUPS の両方のシリアル化された JicsContainer オブジェクトのリストを JSON として返します。

レスポンス例:

```
[
    {
      "name": "Resources",
      "children": [
        {
          "jacType": "JACList",
          "name": "MURACHS",
          "isActive": true,
          "children": [
            {
              "jacType": "JACGroup",
              "name": "MURACHS",
              "isActive": true,
              "children": []
            }
          ]
        },
        {
          "jacType": "JACGroup",
          "name": "TEST",
          "isActive": true,
          "children": []
        }
      ],
      "isExpanded": true
    }
  ]
```

JICS GROUPS の一覧表示

- サポートされているメソッド: GET
- 認証が必要
- パス: `/api/services/rest/jicsservice/listJicsGroups`

- : シリアル化された JicsContainer オブジェクト (GROUPS) のリストを JSON として返します。GROUPS は所有 LIST 情報なしで返されます。

レスポンス例:

```
[
    {
      "jacType": "JACGroup",
      "name": "MURACHS",
      "isActive": true,
      "children": []
    },
    {
      "jacType": "JACGroup",
      "name": "TEST",
      "isActive": true,
      "children": []
    }
  ]
```

特定の LIST の JICS GROUPS の一覧表示

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/listGroupsForList`
- 引数: JSON ペイロードで、探している GROUPS を含む JICS LIST を表します。これは、`com.netfective.bluage.jac.entities.JACList` オブジェクトの JSON のシリアル化です。

  リクエスト例:

  ```
  {
      "jacType":"JACList",
      "name":"MURACHS",
      "isActive":true
    }
  ```

- 戻り値: 指定された LIST にアタッチされた JSON 形式のシリアル化された JicsContainer オブジェクト (GROUPS) のリスト。GROUPS は所有 LIST 情報なしで返されます。

レスポンス例:

```
[
    {
      "jacType": "JACGroup",
      "name": "MURACHS",
      "isActive": true,
      "children": []
    }
  ]
```

特定の GROUP の JICS リソースの一覧表示

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/listResourcesForGroup`
- 引数: JSON ペイロードで、探しているリソースを含む JICS GROUP を表します。これは、`com.netfective.bluage.jac.entities.JACGroup` オブジェクトの JSON のシリアル化です。GROUP のすべてのフィールドを指定する必要はありませんが、名前は必須です。

  リクエスト例:

```
{
    "jacType":"JACGroup",
    "name":"MURACHS",
    "isActive":true
  }
```

- 戻り値: 指定された GROUP が所有するシリアル化された JicsResource オブジェクトのリスト。オブジェクトは順不同で返されます。また、さまざまなタイプ (PROGRAM、TRANSACTION、FILE など) があります。

指定された GROUP の LIST JICS (名前の使用による代替)

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/listResourcesForGroupName`

- 引数: 探しているリソースを所有する GROUP の名前。
- 戻り値: 指定された GROUP が所有するシリアル化された JicsResource オブジェクトのリスト。オブジェクトは順不同で返されます。また、さまざまなタイプ (PROGRAM、TRANSACTION、FILE など) があります。

複数の LISTS の所有 GROUPS の編集

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/editGroupsList`
- 引数: 子 GROUPS を含む LISTS のコレクションを JSON で表現したもの。

  リクエスト例:

```
[
  {
        "jacType": "JACList",
        "name": "MURACHS",
        "isActive": true,
        "children": [
          {
            "jacType": "JACGroup",
            "name": "MURACHS",
            "isActive": true,
            "children": []
          },
          {
            "jacType": "JACGroup",
            "name": "TEST",
            "isActive": true,
            "children": []
          }
        ]
  }
]
```

  これを編集する前には、「MURACHS」という名前のグループだけが「MURACHS」という名前の LIST に属していました。これを編集することにより、「TEST」という名前のグループを「MURACHS」という名前の LIST に「追加」しました。

- ブール値を返します。値が「true」の場合、LISTS の変更は基盤となる JICS ストレージ内で適切に永続化されています。

LIST の削除

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/deleteList`
- 引数: 削除する JICS LIST を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACList` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、LIST の削除は JICS ストレージ内で適切に行われています。

GROUP の削除

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/deleteGroup`
- 引数: 削除する JICS GROUP を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACGroup` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、GROUP の削除は JICS ストレージ内で適切に行われています。

TRANSACTION の削除

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/deleteTransaction`
- 引数: 削除する JICS Transaction を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACTransaction` オブジェクトの JSON のシリアル化です。

- ブール値を返します。値が「true」の場合、TRANSACTION の削除は JICS ストレージ内で適切に行われています。

PROGRAM の削除

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/deleteProgram`
- 引数: 削除する JICS Program を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACProgram` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、PROGRAM の削除は JICS ストレージ内で適切に行われています。

FILE の削除

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/deleteFile`
- 引数: 削除する JICS File を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACFile` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、FILE の削除は JICS ストレージ内で適切に行われています。

TDQUEUE の削除

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/deleteTDQueue`
- 引数: 削除する JICS TDQUEUE を表す JSON ペイロード。これは「com.netfective.bluage.jac.entities.JACTDQueue」オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TDQUEUE の削除は JICS ストレージ内で適切に行われています。

TSMODEL の削除

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/deleteTSModel`
- 引数: 削除する JICS TSMODEL を表す JSON ペイロード。これは「com.netfective.bluage.jac.entities.JACTSModel」オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TSMODEL の削除は JICS ストレージ内で適切に行われています。

LIST の作成

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/createList`
- 引数: 作成する JICS LIST を表す JSON ペイロード。これは「com.netfective.bluage.jac.entities.JACList」オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、LIST は JICS ストレージ内で適切に作成されています。

> **Note**
>
> LIST は、常に空の状態で作成されます。GROUPS を LIST にアタッチするには、別のオペレーションが必要です。

GROUP の作成

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/createGroup`
- 引数: 作成する JICS GROUP を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACGroup` オブジェクトの JSON のシリアル化です。

- ブール値を返します。値が「true」の場合、GROUP は JICS ストレージ内で適切に作成されています。

> **Note**
>
> GROUP は常に空の状態で作成されます。RESOURCES を GROUP にアタッチするには、追加のオペレーションが必要です (リソースを作成すると、リソースは特定の GROUP に自動的にアタッチされます)。

RESOURCES 作成に関する一般的な考慮事項

次のすべてのエンドポイントは JICS RESOURCES の作成に関連しており、共通する制約がいくつかあります。エンドポイントに送信されるリクエストペイロードでは、`groupName` フィールドに値を設定する必要があります。

GROUP の所有権の制約:

リソースは、既存のグループにアタッチしないと作成できません。エンドポイントは、groupName を使用して、このリソースをアタッチするグループを取得します。`groupName` は、既存のグループ名を指している必要があります。`groupName` が基盤となる JICS ストレージの既存のグループを指していない場合、HTTP STATUS 400 のエラーメッセージが送信されます。

GROUP 内の単一性の制約:

指定の名前のリソースは、指定のグループ内で一意である必要があります。単一性のチェックは、各リソースの作成エンドポイントによって実行されます。指定したペイロードが単一性の制約を優先しない場合、エンドポイントは HTTP STATUS 400 (BAD REQUEST) のレスポンスを送信します。詳細については、次のレスポンス例を参照してください。

ペイロードの例:「TEST」グループにトランザクション「ARIT」を作成しようとしましたが、このグループに、既にその名前のトランザクションが存在しています。

```
{
    "jacType":"JACTransaction",
    "name":"ARIT",
    "groupName":"TEST",
    "isActive":true
  }
```

次のようなエラーレスポンスが返されました。

```
{
    "timestamp": 1686759054510,
    "status": 400,
    "error": "Bad Request",
    "path": "/jac/api/services/rest/jicsservice/createTransaction"
  }
```

サーバーのログを調べることにより、問題の原因が確認できます。

```
2023-06-14 18:10:54 default         TRACE - o.s.w.m.HandlerMethod
        - Arguments: [java.lang.IllegalArgumentException: Transaction already
 present in the group, org.springframework.security.web.header.HeaderWriterFilter
$HeaderWriterResponse@e34f6b8]
2023-06-14 18:10:54 default         ERROR - c.n.b.j.a.WebConfig                    -
 400
java.lang.IllegalArgumentException: Transaction already present in the group
 at
 com.netfective.bluage.jac.server.services.rest.impl.JicsServiceImpl.createElement(JicsServiceI
```

TRANSACTION の作成

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/createTransaction`
- 引数: 作成する JICS TRANSACTION を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACTransaction` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TRANSACTION は JICS ストレージ内で適切に作成されています。

PROGRAM の作成

- サポートされているメソッド: POST
- 認証が必要

- パス: `/api/services/rest/jicsservice/createProgram`
- 引数: 作成する JICS PROGRAM を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACProgram` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、PROGRAM は JICS ストレージ内で適切に作成されています。

FILE の作成

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/createFile`
- 引数: 作成する JICS FILE を表す JSON ペイロード。これは「com.netfective.bluage.jac.entities.JACFile」オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、FILE は JICS ストレージ内で適切に作成されています。

TDQUEUE の作成

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/createTDQueue`
- 引数: 作成する JICS TDQUEUE を表す JSON ペイロード。これは「com.netfective.bluage.jac.entities.JACTDQueue」オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TDQUEUE は JICS ストレージ内で適切に作成されています。

TSMODEL の作成

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/createTSModel`

- 引数: 作成する JICS TSMODEL を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACTSModel` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TSMODEL は JICS ストレージ内で適切に作成されています。

LIST の更新

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/updateList`
- 引数: 更新する JICS LIST を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACList` オブジェクトの JSON のシリアル化です。その LIST の子を指定する必要はありません。LIST の更新メカニズムではこのことは考慮されません。
- ブール値を返します。値が「true」の場合、LIST は JICS ストレージ内で適切に更新されています。

LIST の「isActive」フラグを更新すると、LIST のすべての所有要素、つまり LIST が所有するすべての GROUPS と、それらの GROUPS が所有するすべての RESOURCES に反映されます。これは、複数の GROUPS にまたがる大量のリソースを 1 回のオペレーションで無効にする場合に便利な方法です。

GROUP の更新

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/updateGroup`
- 引数: 更新する JICS GROUP を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACGroup` オブジェクトの JSON のシリアル化です。その GROUP の子を指定する必要はありません。GROUP の更新メカニズムではこのことは考慮されません。
- ブール値を返します。値が「true」の場合、GROUP は JICS ストレージ内で適切に更新されています。

> **Note**
>
> GROUP の「isActive」フラグを更新すると、GROUP のすべての所有要素、つまり GROUP が所有するすべての RESOURCES に反映されます。これは、特定の GROUP 内で、大量のリソースを 1 回のオペレーションで無効にする場合に便利な方法です。

RESOURCES の更新に関する一般的な考慮事項

次のエンドポイントは、すべて JICS RESOURCES の更新に関するものです。`groupName` フィールドを使用して、JICS RESOURCE の所有 GROUP を変更できます。ただし、フィールド値が基盤 JICS ストレージの存在する GROUP を指している場合に限ります。それ以外の場合は、エンドポイントから BAD REQUEST レスポンス (HTTP STATUS 400) が返されます。

TRANSACTION の更新

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/updateTransaction`
- 引数: 更新する JICS TRANSACTION を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACTransaction` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TRANSACTION は JICS ストレージ内で適切に更新されています。

PROGRAM の更新

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/updateProgram`
- 引数: 更新する JICS PROGRAM を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACProgram` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、PROGRAM は JICS ストレージ内で適切に更新されています。

FILE の更新

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/updateFile`
- 引数: 更新する JICS FILE を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACFile` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、FILE は JICS ストレージ内で適切に更新されています。

TDQUEUE の更新

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/updateTDQueue`
- 引数: 更新する JICS TDQUEUE を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACTDQueue` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TDQueue は JICS ストレージ内で適切に更新されています。

TSMODEL の更新

- サポートされているメソッド: POST
- 認証が必要
- パス: `/api/services/rest/jicsservice/updateTSModel`
- 引数: 更新する JICS TSMODEL を表す JSON ペイロード。これは、`com.netfective.bluage.jac.entities.JACTSModel` オブジェクトの JSON のシリアル化です。
- ブール値を返します。値が「true」の場合、TSMODEL は JICS ストレージ内で適切に更新されています。

JAC ユーザー管理エンドポイント

トピック

ユーザーの削除

- サポートされているメソッド: POST
- 認証と管理者権限が必要です。
- パス: `/api/services/security/servicelogin/deleteuser`
- 引数: ストレージから削除するユーザーを表す `com.netfective.bluage.jac.entities.SignOn` オブジェクトの JSON のシリアル化。
- ブール値を返します。値が「true」の場合、ユーザーは JICS システムから正常に削除されています。

> **Note**
>
> このオペレーションは元に戻すことはできません。削除されたユーザーは、JAC アプリケーションに再び接続できません。注意して使用してください。

ユーザーのログ記録

- サポートされているメソッド: POST
- 認証と管理者権限が必要です。
- パス: `/api/services/security/servicelogin/login`

- 現在のリクエストで入力された認証情報を持つユーザーを表す、com.netfective.bluage.jac.entities.SignOn オブジェクトの JSON のシリアル化を返します。パスワードは、返されたオブジェクトの中では表示されません。ユーザーに割り当てられたロールが一覧表示されます。

レスポンス例:

```
{
    "login": "sadmin",
    "password": null,
    "roles": [
      {
        "id": 0,
        "roleName": "ROLE_SUPER_ADMIN"
      }
    ]
  }
```

システムに少なくとも 1 人のユーザーが存在するかどうかをテストする

- サポートされているメソッド: GET
- パス: /api/services/security/servicelogin/hasAccount
- JICS システムで少なくとも 1 人のユーザー (デフォルトのスーパー管理者ユーザー以外) が作成されている場合、その値が「true」となるブール値を返します。

新規ユーザーの記録

- サポートされているメソッド: POST
- 認証と管理者権限が必要です。
- パス: /api/services/security/servicelogin/recorduser
- 引数: ストレージに追加されるユーザーを表す com.netfective.bluage.jac.entities.SignOn オブジェクトの JSON のシリアル化。ユーザーのロールを定義する必要があります。定義されていないと、ユーザーは JAC 機能やエンドポイントを使用できない可能性があります。

リクエスト例:

```
{
    "login": "simpleuser",
    "password": "simpleuser",
    "roles": [
      {
        "id": 2,
        "roleName": "ROLE_USER"
      }
    ]
  }
```

新規ユーザーを記録するときに使用できるのは、次のロールのみです。

- ROLE_ADMIN: JICS リソースとユーザーを管理できます。
- ROLE_USER: JICS リソースは管理できますが、ユーザーは管理できません。

ユーザーの一覧表示

- サポートされているメソッド: GET
- 認証と管理者権限が必要です。
- パス: `/api/services/security/servicelogin/listusers`
- JSON としてシリアル化された `com.netfective.bluage.jac.entities.SignOn` の一覧を返します。

ユーザーの一覧表示

- サポートされているメソッド: GET
- 認証と管理者権限が必要です。
- パス: `/api/services/security/servicelogin/listusers`
- JSON としてシリアル化された `com.netfective.bluage.jac.entities.SignOn` の一覧を返します。

現在のユーザーをログアウトする

- サポートされているメソッド: GET
- パス: `/api/services/security/servicelogout/logout`

- 現在のユーザーのログアウトが成功した場合、「{"success":true}」の JSON メッセージを返します (関連する HTTP セッションは無効になります)。

JICS サーバーのヘルスステータス

- サポートされているメソッド: GET
- パス: `/api/services/rest/jicsserver/serverIsUp`
- HTTP STATUS 200 のレスポンスが返された場合、JICS サーバーが稼働中であることを示します。

データ構造

このセクションでは、さまざまなデータ構造の詳細について説明します。

トピック

Job 実行詳細メッセージの構造

各ジョブの実行の詳細には、次のフィールドが含まれます。

scriptId

呼び出されたスクリプトの識別子。

caller

発信者の IP アドレス。

識別子

一意のジョブの実行の識別子。

startTime

ジョブの実行を開始した日時。

endTime

ジョブの実行が終了した日時。

status

ジョブの実行のステータス。指定できる値は 1 つです。

- DONE: ジョブの実行が正常に終了しました。
- TRIGGERED: ジョブの実行がトリガーされたが、まだ開始されていません。
- RUNNING: ジョブの実行中です。
- KILLED: ジョブの実行は強制終了されました。
- FAILED: ジョブの実行が失敗しました。

executionResult

ジョブ実行の結果を要約するメッセージ。このメッセージは、ジョブ実行がまだ終了していない場合の簡単なメッセージか、次のフィールドを持つ JSON 構造のいずれかになります。

- ExitCode: 数値の終了コードで、負の値は障害の状態を示します。
- program: ジョブによって起動された最新のプログラムを示します。
- status: 指定できる値は次のうちの 1 つです。
  - Error: exitCode = -1 の場合。これはジョブ実行中に発生した (技術的な) エラーに対応します。
  - Failed: exitCode = -2 の場合、これはサービスプログラムの実行中に発生した障害 (異常終了の状態など) に対応します。
  - Succeeded: exitCode >= 0 の場合:
- stepName: ジョブで実行された最新のステップの名前。

executionMode

ジョブの起動方法によって、SYNCHRONOUS または ASYNCHRONOUS のいずれかになります。

サンプル出力:

```
{
```

```
    "scriptId": "INTCALC",
    "caller": "127.0.0.1",
    "identifier": "97d410be-efa7-4bd3-b7b9-d080e5769771",
    "startTime": "06-09-2023 11:42:41",
    "endTime": "06-09-2023 11:42:42",
    "status": "DONE",
    "executionResult": "{ \"exitCode\": -1, \"stepName\": \"STEP15\", \"program\":
\"CBACT04C\", \"status\": \"Error\" }",
    "executionMode": "ASYNCHRONOUS"
 }
```

トランザクション起動結果の構造

構造には、次のフィールドが含まれる場合があります。

outCome

トランザクションの実行結果を表す文字列。可能な値は以下のとおりです。

- Success: トランザクションの実行が正しく終了しました。
- Failure: トランザクションの実行が正しく終了しませんでした。いくつかの問題が発生しました。

commarea

COMMAREA の最終値を byte64 でエンコードしたバイト配列として表す文字列。空の文字列の場合もあります。

containerRecord

(オプション) コンテナのレコード内容を byte64 でエンコードしたバイト配列として表す文字列。

serverDescription

(デバッグ目的で) リクエストを処理したサーバーに関する情報が含まれる場合があります。空の文字列の場合もあります。

abendCode

(オプション) 起動されたトランザクションによって参照されたプログラムが異常終了した場合、異常終了コード値がこのフィールドに文字列として返されます。

レスポンス例:

成功

```
{
    "outCome": "Success",
    "commarea": "",
    "serverDescription": ""
  }
```

失敗

```
{
    "outCome": "Failure",
    "commarea": "",
    "serverDescription": "",
    "abendCode": "AEIA"
  }
```

トランザクション起動レコードの構造

構造には、次のフィールドが含まれる場合があります。

recordContent

COMMAREA のレコード内容を byte64 でエンコードしたバイト配列として表す文字列。

containerRecord

コンテナのレコード内容を byte64 でエンコードしたバイト配列として表す文字列。

serverDescription

(デバッグ目的で) リクエストを処理したサーバーに関する情報が含まれる場合があります。空の文字列の場合もあります。

レスポンス例:

成功

```
{
    "recordContent": "",
    "serverDescription": ""
}
```

キューでジョブの可能性のあるステータス

キューでは、ジョブのステータスを持つことができます。

ACTIVE

ジョブは現在キューで実行中です。

EXECUTION_WAIT

ジョブはスレッドが使用可能になるのを待っています。

SCHEDULED

ジョブは特定の日付と時刻に実行されるようにスケジュールされています。

HOLD

ジョブは実行前にリリースされるのを待っています。

COMPLETED

ジョブは正常に実行されました。

FAILED

ジョブの実行が失敗しました。

UNKNOWN

ステータスは不明です。

ジョブ入力の送信

送信するジョブ入力は、`com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage` オブジェクトの JSON のシリアル化です。以下のサンプル入力は、このような Bean のすべてのフィールドを示しています。

入力例:

```
{
    "messageQueueName":null,
    "scheduleDate":null,
```

```
    "scheduleTime":null,
    "programName":"PTA0044",
    "programParams":
     {"wmind":"B"},
    "localDataAreaValue":"",
    "userName":"USER1",
    "jobName":"PTA0044",
    "jobNumber":9,
    "jobPriority":5,
    "executionDate":"20181231",
    "jobQueue":"queue1",
    "jobOnHold":false
}
```

jobNumber

jobNumber が 0 の場合、ジョブ番号はジョブ番号シーケンスの次の番号を使用して自動的に生成されます。この値は 0 に設定する必要があります (テスト目的を除く)。

jobPriority

AS400 のデフォルトのジョブ優先度は 5 です。有効範囲は 0～9 で、0 が最優先です。

jobOnHold

保留でジョブが送信された場合、そのジョブはすぐには実行されず、誰かが「リリース」したときに初めて実行されます。ジョブは REST API (/release または /release-all) を使用してリリースできます。

scheduleDate および scheduleTime

これらの値が null でない場合、ジョブは指定された日時に実行されます。

日付

MMddyy または ddMMyyyy の形式で指定できます (入力のサイズによって、使用する形式が決まります)。

時間

HHmm または HHmmss の形式で指定できます (入力のサイズによって、使用する形式が決まります)。

programParams

これは、マップとしてプログラムに渡されます。

スケジューリングされているジョブのレスポンス一覧

これは、list-jobs ジョブキューエンドポイントの構造です。ジョブの送信に使用されたジョブ送信メッセージは、応答の一部であることに注意してください。これは、トラッキング、テスト、再送信の目的で使用できます。ジョブが完了すると、開始日と終了日も入力されます。

```
[
    {
      "qrtzJobGroup": "PTA0044-PTA0044",
      "qrtzJobName": "PTA0044-168156109957800",
      "status": "HOLD",
      "qrtzDelay": 0,
      "startDate": null,
      "endDate": null,
      "jobName": "PTA0044",
      "userName": "USER1",
      "jobNumber": 9,
      "jobPriority": 5,
      "jobQueue": "queue1",
      "message": {
        "messageQueueName": null,
        "scheduleDate": null,
        "scheduleTime": null,
        "programName": "PTA0044",
        "programParams": {
          "wmind": "B"
        },
        "localDataAreaValue": "",
        "userName": "USER1",
        "jobName": "PTA0044",
        "jobNumber": 9,
        "jobPriority": 5,
        "executionDate": "20181231",
        "jobQueue": "queue1",
        "jobOnHold": true
      }
    },
    {
      "qrtzJobGroup": "PTA0044-PTA0044",
      "qrtzJobName": "PTA0044-166196954877600",
      "status": "COMPLETED",
      "qrtzDelay": 0,
      "startDate": "2022-10-13T22:48:34.025+00:00",
```

```
      "endDate": "2022-10-13T22:52:54.475+00:00",
      "jobName": "PTA0044",
      "userName": "USER1",
      "jobNumber": 9,
      "jobPriority": 5,
      "jobQueue": "queue1",
      "message": {
        "messageQueueName": null,
        "scheduleDate": null,
        "scheduleTime": null,
        "programName": "PTA0044",
        "programParams": {
          "wmind": "B"
        },
        "localDataAreaValue": "",
        "userName": "USER1",
        "jobName": "PTA0044",
        "jobNumber": 9,
        "jobPriority": 5,
        "executionDate": "20181231",
        "jobQueue": "queue1",
        "jobOnHold": true
      }
    }
  ]
```

「保留中」のジョブのレスポンス一覧

返されたジョブがすべて「保留」になることを除けば、以前のジョブと構造は同じです。

```
[
    {
      "qrtzJobGroup": "PTA0044-PTA0044",
      "qrtzJobName": "PTA0044-168156109957800",
      "status": "HOLD",
      "qrtzDelay": 0,
      "startDate": null,
      "endDate": null,
      "jobName": "PTA0044",
      "userName": "USER1",
      "jobNumber": 9,
      "jobPriority": 5,
```

```
      "jobQueue": "queue1",
      "message": {
        "messageQueueName": null,
        "scheduleDate": null,
        "scheduleTime": null,
        "programName": "PTA0044",
        "programParams": {
          "wmind": "B"
        },
        "localDataAreaValue": "",
        "userName": "USER1",
        "jobName": "PTA0044",
        "jobNumber": 9,
        "jobPriority": 5,
        "executionDate": "20181231",
        "jobQueue": "queue1",
        "jobOnHold": true
      }
    }
  ]
```

AWS Blu Age ランタイム (非マネージド) セットアップ

このセクションでは、 AWS インフラストラクチャで AWS Blu Age ランタイム (非マネージド) を設定する手順について説明します。

トピック

- AWS Blu Age ランタイムの前提条件
- AWS Blu Age ランタイムオンボーディング
- AWS Blu Age ランタイムのインフラストラクチャセットアップ要件 (非マネージド)
- AWS によって管理される Amazon ECS での Blu Age ランタイムデプロイ AWS Fargate
- AWS Amazon EC2 での Blu Age ランタイムデプロイ

AWS Blu Age ランタイムの前提条件

AWS Blu Age ランタイム (非マネージド) は、いくつかのリリースバージョン で使用できます。進行中のモダナイゼーションプロジェクトがある場合は、実装とテストの目的でランタイムの増分バージョンが必要になる場合があります。ニーズを定義するには、 AWS Blu Age デリバリーマネージャーにお問い合わせください。

AWS Blu Age ランタイム (非マネージド) オンボーディングプロセスを開始する前に、次の操作を行います。

- AWS アカウントがあることを確認します。
- AWS Blu Age でモダナイズされたアプリケーションがリファクタリングされていることを確認します。
- AWS リージョンとコンピューティング (の Amazon EC2 または Amazon ECS) を選択します AWS Fargate。
- 使用する AWS Blu Age ランタイムバージョンを選択します。
- AWS Blu Age ランタイム (非マネージド) の実行に必要な追加コンポーネントを確認してthe section called “インフラストラクチャセットアップ要件”検証します。

> **Note**
>
> AWS Blu Age ランタイム (非マネージド) の機能をテストする場合はPlanets Demo、デモアプリケーション を使用できます。これは PlanetsDemo-v1.zip からダウンロードできます。

AWS Blu Age ランタイムオンボーディング

開始するには、 AWS Support ケースを作成して AWS Blu Age ランタイムにアクセスするためのオンボーディングをリクエストします。ID AWS アカウント 、使用する AWS リージョン、コンピューティングの選択とランタイムバージョンをリクエストに含めます。必要なバージョンが不明な場合は、 AWS Blu Age 配信マネージャーにお問い合わせください。

AWS Amazon EC2 での Blu Age ランタイム (非マネージド)

AWS Blu Age ランタイム (非マネージド) アーティファクトは、リージョン別およびコンピューティングの選択別に異なる Amazon S3 バケットに保存されます。Amazon EC2 で AWS リージョン for AWS Blu Age ランタイム (非マネージド) のバケットにアクセスするには、次の表に示す名前を使用します。

AWS リージョン	バケットリリース	バケットビルド
米国東部 (オハイオ)	aws-bluage-runtime-artifacts-055777665268-us-east-2	aws-bluage-runtime-artifacts-dev-055777665268-us-east-2

AWS リージョン	バケットリリース	バケットビルド
米国東部 (バージニア北部)	aws-bluage-runtime-artifacts-139023371234-us-east-1	aws-bluage-runtime-artifacts-dev-139023371234-us-east-1
米国西部 (北カリフォルニア)	aws-bluage-runtime-artifacts-788454048782-us-west-1	aws-bluage-runtime-artifacts-dev-788454048782-us-west-1
米国西部 (オレゴン)	aws-bluage-runtime-artifacts-836771190483-us-west-2	aws-bluage-runtime-artifacts-dev-836771190483-us-west-2
欧州 (アイルランド)	aws-bluage-runtime-artifacts-925278190477-eu-west-1	aws-bluage-runtime-artifacts-dev-925278190477-eu-west-1
欧州 (パリ)	aws-bluage-runtime-artifacts-673009995881-eu-west-3	aws-bluage-runtime-artifacts-dev-673009995881-eu-west-3
欧州 (フランクフルト)	aws-bluage-runtime-artifacts-485196800481-eu-central-1	aws-bluage-runtime-artifacts-dev-485196800481-eu-central-1
南米 (サンパウロ)	aws-bluage-runtime-artifacts-737536804457-sa-east-1	aws-bluage-runtime-artifacts-dev-737536804457-sa-east-1
アジアパシフィック (東京)	aws-bluage-runtime-artifacts-445578176276-ap-northeast-1	aws-bluage-runtime-artifacts-dev-445578176276-ap-northeast-1
アジアパシフィック (シドニー)	aws-bluage-runtime-artifacts-726160321909-ap-southeast-2	aws-bluage-runtime-artifacts-dev-726160321909-ap-southeast-2

AWS Fargate が管理する Amazon ECS の Blu Age ランタイム (非マネージド)

AWS Blu Age ランタイム (非マネージド) アーティファクトは、リージョン別およびコンピューティングの選択別に異なる Amazon S3 バケットに保存されます。Fargate によって管理される Amazon ECS の AWS リージョン for AWS Blu Age ランタイム (非マネージド) のバケットにアクセスするには、次の表に示す名前を使用します。

AWS リージョン	バケットリリース	バケットビルド
米国東部 (オハイオ)	aws-bluage-runtime-fargate-rel-483416914331-us-east-2	aws-bluage-runtime-fargate-dev-483416914331-us-east-2
米国東部 (バージニア北部)	aws-bluage-runtime-fargate-rel-308472162679-us-east-1	aws-bluage-runtime-fargate-dev-308472162679-us-east-1
米国西部 (北カリフォルニア)	aws-bluage-runtime-fargate-rel-343763094578-us-west-1	aws-bluage-runtime-fargate-dev-343763094578-us-west-1
米国西部 (オレゴン)	aws-bluage-runtime-fargate-rel-688933007849-us-west-2	aws-bluage-runtime-fargate-dev-688933007849-us-west-2
欧州 (アイルランド)	aws-bluage-runtime-fargate-rel-140138033705-eu-west-1	aws-bluage-runtime-fargate-dev-140138033705-eu-west-1
欧州 (パリ)	aws-bluage-runtime-fargate-rel-339712948211-eu-west-3	aws-bluage-runtime-fargate-dev-339712948211-eu-west-3
欧州 (フランクフルト)	aws-bluage-runtime-fargate-rel-339712918892-eu-central-1	aws-bluage-runtime-fargate-dev-339712918892-eu-central-1
南米 (サンパウロ)	aws-bluage-runtime-fargate-rel-767397998881-sa-east-1	aws-bluage-runtime-fargate-dev-767397998881-sa-east-1
アジアパシフィック (東京)	aws-bluage-runtime-fargate-rel-891377400849-ap-northeast-1	aws-bluage-runtime-fargate-dev-891377400849-ap-northeast-1
アジアパシフィック (シドニー)	aws-bluage-runtime-fargate-rel-533267435478-ap-southeast-2	aws-bluage-runtime-fargate-dev-533267435478-ap-southeast-2

コマンドラインを使用してバケットの内容をリスト表示する

オンボードした後、ターミナルで以下のコマンドを実行するとバケットの内容を一覧表示できます。

```
aws s3 ls bucket-name
```

を、前の表 AWS リージョン の のバケット名bucket-nameに置き換えます。

このコマンドは、 などの AWS Blu Age ランタイム (非マネージド) ランタイムの異なるバージョンに対応するフォルダのリストを返します。

```
PRE 3.3.0.1/
PRE 3.3.0.2/
```

利用可能な最新バージョンを使用することをお勧めします。それが不可能な場合は、アプリケーションのリファクタリングフェーズで検証されたランタイムバージョンを使用します。特定のバージョンで利用可能なフレームワークをリスト表示するには、次のコマンドを実行します。

```
aws s3 ls s3://bucket-name/version/Framework/
```

を のバケットの名前bucket-nameに置き換え AWS リージョン 、 を必要なバージョンversionに置き換えます。次に例を示します。

```
aws s3 ls s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/
Framework/
```

このコマンドは、次のようなフレームワークのリストを返します。

```
2023-06-05 10:26:52   97960225 aws-bluage-runtime-3.4.0.tar.gz
2023-06-05 10:27:12         45 aws-bluage-runtime-3.4.0.tar.gz.checksumSHA256
2023-06-05 10:27:14  138497123 aws-bluage-webapps-3.4.0.tar.gz
2023-06-05 10:27:44         45 aws-bluage-webapps-3.4.0.tar.gz.checksumSHA256
```

フレームワークをダウンロードする

例えば、フレームワークをダウンロードして、既存の Amazon EC2 インスタンスの Velocity ランタイムバージョンをアップグレードできます。

```
aws s3 cp s3://bucket-name/version/Framework/ folder-of-your-choice --
recursive
```

コードの説明は以下のとおりです。

folder-of-your-choice

フレームワークをダウンロードするフォルダパス。

例：`aws s3 cp s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/Framework/ . --recursive`

このコマンドによって以下の出力が生成されます。

```
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/
Framework/aws-bluage-runtime-3.4.0.tar.gz.checksumSHA256 to ./aws-bluage-
runtime-3.4.0.tar.gz.checksumSHA256
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/
Framework/aws-bluage-webapps-3.4.0.tar.gz.checksumSHA256 to ./aws-bluage-
webapps-3.4.0.tar.gz.checksumSHA256
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/Framework/aws-
bluage-webapps-3.4.0.tar.gz to ./aws-bluage-webapps-3.4.0.tar.gz
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/Framework/aws-
bluage-runtime-3.4.0.tar.gz to ./aws-bluage-runtime-3.4.0.tar.gz
```

フレームワークファイルは次のように一覧表示できます。

```
ls -l
```

このコマンドによって以下の出力が生成されます。

```
total 230928
-rw-rw-r-- 1 cloudshell-user cloudshell-user  97960225 Jun  5 10:26 aws-bluage-
runtime-3.4.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user        45 Jun  5 10:27 aws-bluage-
runtime-3.4.0.tar.gz.checksumSHA256
-rw-rw-r-- 1 cloudshell-user cloudshell-user 138497123 Jun  5 10:27 aws-bluage-
webapps-3.4.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user        45 Jun  5 10:27 aws-bluage-
webapps-3.4.0.tar.gz.checksumSHA256
```

AWS Blu Age ランタイムのインフラストラクチャセットアップ要件 (非マネージド)

このトピックでは、 AWS Blu Age ランタイム (非マネージド) を実行するために必要な最小限のインフラストラクチャ設定について説明します。次の手順では、選択したコンピューティングに AWS Blu Age ランタイム (非マネージド) をセットアップして、 AWS Blu Age ランタイムにモダナイズされたアプリケーションをデプロイする方法について説明します。作成するリソースは、アプリケーションドメイン専用のサブネットを持つ Amazon VPC 内に存在する必要があります。

セキュリティグループの作成

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. ナビゲーションペインの [セキュリティ] で、[セキュリティグループ] を選択します。
3. 中央のペインで、[セキュリティグループの作成] を選択します。
4. [セキュリティグループ名] フィールドに「**M2BluagePrivateLink-SG**」と入力します。
5. [インバウンドルール] セクションで、[ルールの追加] を選択します。
6. [タイプ] で、[HTTPS] を選択します。
7. [ソース] に VPC CIDR を入力します。
8. [アウトバウンドのルール] セクションで、[ルールの追加] を選択します。
9. [タイプ] で、[HTTPS] を選択します。
10. [送信先] に「**0.0.0.0/0**」と入力します。
11. [Create Security Group] を選択します。

Amazon VPC エンドポイントを作成する

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. 左のナビゲーションペインの [仮想プライベートクラウド] で、[エンドポイント] を選択します。
3. 中央のペインで、[エンドポイントの作成] を選択します。
4. 「サービス」セクション**SQS**で、検索フィールドに「」と入力し、リージョンに対応する Amazon SQS サービスを選択します。
5. [VPC] セクションでは、前のステップで作成した VPC を選択します。
6. [サブネット] セクションで、アプリケーションドメイン用に作成したサブネットを選択します。
7. セキュリティグループ セクションで、M2BluagePrivateLink-SG を選択します。
8. [エンドポイントの作成] を選択します。

IAM ポリシーを作成する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインの [アクセス管理] で、[ポリシー] を選択します。
3. 中央のペインで、[ポリシーの作成] を選択します。
4. [ポリシーエディター] セクションで、[JSON] を選択します。

5. エディタに表示されるすべての JSON を以下の JSON に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sqs:GetQueueUrl",
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ポリシーをカスタマイズするためにさらに詳細が必要な場合は、 AWS Blu Age 配信マネージャーまたはアカウントマネージャーにお問い合わせください。

6. [次へ] を選択します。
7. ポリシーの名前を入力し、[ポリシーの作成] を選択します。

IAM ロールを作成する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインの [アクセス管理] で、[ロール] を選択します。
3. 中央のペインで、[ロールを作成] を選択します。
4. ユースケースセクションで、コンピューティングの選択に応じて EC2 または Elastic Container Service (次に Elastic Container Service タスク) を選択します。
5. [次へ] を選択します。
6. 作成したポリシーの名前を検索ボックスに入力します。
7. ポリシーの左側にあるチェックボックスをオンにします。
8. [次へ] をクリックします。

9. ロールの名前を入力し、[ロールの作成] を選択します。

Amazon EC2 での AWS Blu Age ランタイムの実行

Amazon EC2 インスタンスを作成する

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. [インスタンスを起動] を選択します。
3. インスタンスタイプ で、 に記載されているタイプのいずれかを選択しますthe section called “AWS Blu Age ランタイムの Amazon EC2 インスタンスタイプ (Amazon EC2)”。
4. [キーペア] セクションで、既存のキーペアを選択するか、新規にキーペアを作成します。
5. [ネットワーク設定] セクションで、[既存のセキュリティグループを選択する] を選択します。
6. 共通セキュリティグループ で、M2BluagePrivateLink-SG を選択します。
7. [高度な詳細] セクションを展開します。
8. [IAM インスタンスプロファイル] で、先ほど作成した IAM ロールを選択します。
9. [インスタンスを起動] を選択します。

Amazon EC2 インスタンスの状態が [実行中] に変わったら、インスタンスに接続し、次のソフトウェアコンポーネントをインストールします。

- Java ランタイム環境 (JRE) 11
- Apache Tomcat 9
- AWS Blu Age ランタイム (Amazon EC2)。Apache Tomcat インストールフォルダのルートに AWS Blu Age ランタイムをインストールします (一部のファイルは追加され、他のファイルは上書きされます) 。

他のウェブアプリケーションをインストールする場合は、別のフォルダにインストールします。この場合は、Apache Tomcat をインストールしてから、その上にアーカイブをインストールするプロセスを繰り返してください。

によって管理される Amazon ECS での AWS Blu Age ランタイムの実行 AWS Fargate

後で タスクを起動するときに使用する Amazon ECS クラスターとタスクロールを作成します AWS Fargate 。タスクロールには、前に作成したポリシーを含めます。シナリオによっては、タスクロールに追加の IAM ポリシーをアタッチする必要がある場合があります。

AWS Blu Age ランタイムの Amazon EC2 インスタンスタイプ (Amazon EC2)

AWS Blu Age ランタイム (Amazon EC2) に使用できる Amazon EC2 インスタンスタイプのリストを次に示します。

```
t3.xlarge
t3.small
t3.large
t2.small
t2.large
r6i.xlarge
r6i.large
r6i.4xlarge
r6i.2xlarge
r5b.xlarge
r5b.large
r5b.2xlarge
r3.xlarge
m6i.xlarge
m6i.large
m6i.8xlarge
m6i.4xlarge
m6i.2xlarge
m6i.16xlarge
m5zn.xlarge
m5zn.large
m5zn.3xlarge
m5zn.2xlarge
m5.xlarge
m5.large
m5.8xlarge
m5.4xlarge
m5.2xlarge
m5.16xlarge
m5.12xlarge
c6i.xlarge
c6i.large
c6i.8xlarge
c6i.4xlarge
c6i.2xlarge
c6i.16xlarge
c5.xlarge
c5.large
```

```
c5.9xlarge
c5.4xlarge
c5.2xlarge
c5.18xlarge
c5.12xlarge
```

AWS によって管理される Amazon ECS での Blu Age ランタイムデプロイ AWS Fargate

このセクションのトピックでは、 によって管理される Amazon ECS で AWS Blu Age ランタイムを設定する方法 AWS Fargate、ランタイムバージョンを更新する方法、Amazon CloudWatch アラームを使用してデプロイをモニタリングする方法、およびライセンスされた依存関係を追加する方法について説明します。

トピック

- によって管理される Amazon ECS での AWS Blu Age ランタイムのセットアップ AWS Fargate
- によって管理される Amazon ECS での AWS Blu Age ランタイムのアップグレード AWS Fargate
- によって管理される Amazon ECS での AWS Blu Age ランタイムの Amazon CloudWatch アラーム AWS Fargate
- が管理する Amazon ECS の AWS Blu Age ランタイムでのライセンス付き依存関係の設定 AWS Fargate

によって管理される Amazon ECS での AWS Blu Age ランタイムのセットアップ AWS Fargate

このトピックでは、 が管理する Amazon ECS で AWS Blu Age ランタイムを使用して PlanetsDemo サンプルアプリケーションをセットアップしてデプロイする方法について説明します AWS Fargate。

AWS によって管理される Amazon ECS の Blu Age ランタイム AWS Fargate は Linux/X86 で使用できます。

トピック

- 前提条件
- 設定
- PlanetsDemo アプリケーションをテストする

前提条件

開始する前に、以下の前提条件を満たしていることを確認してください。

- AWS CLI 「AWS CLI の設定」の手順に従って を設定します。
- 「the section called “AWS Blu Age ランタイムの前提条件”」および「the section called “AWS Blu Age ランタイムオンボーディング”」を完了させます。
- AWS Fargate バイナリによって管理される Amazon ECS に AWS Blu Age ランタイムをダウンロードします。手順については、「the section called “AWS Blu Age ランタイムオンボーディング”」を参照してください。
- Apache Tomcat 9 バイナリをダウンロードします。
- PlanetsDemo アプリケーションアーカイブ をダウンロードします。
- JICS 用の Amazon Aurora PostgreSQL データベースを作成し、そこで`PlanetsDemo-v1/jics/sql/initJics.sql`クエリを実行します。Amazon Aurora PostgreSQL データベースを作成する方法については、「Aurora PostgreSQL DB クラスターの作成と接続」を参照してください。

設定

PlanetsDemo サンプルアプリケーションをセットアップするには、次のステップを実行します。

1. Apache Tomcat バイナリをダウンロードしたら、内容を抽出して `conf`フォルダに移動します。編集する`catalina.properties`ファイルを開き、 で始まる行を次の行`common.loader`に置き換えます。

   ```
   common.loader="${catalina.base}/lib","${catalina.base}/lib/
   *.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar","${catalina.home}/
   shared","${catalina.home}/shared/*.jar","${catalina.home}/extra","${catalina.home}/
   extra/*.jar"
   ```

2. tar コマンドを使用して「tar.gz」アーカイブを構築し、Apache Tomcat フォルダを圧縮します。
3. 提供されたランタイムバイナリと Apache Tomcat サーバーバイナリに基づいてカスタムイメージを構築するための Dockerfile を準備します。次の Dockerfile の例を参照してください。目的は、Apache Tomcat 9 をインストールし、その後に Apache Tomcat 9 インストールディレクトリのルートで抽出された AWS Blu Age ランタイム (によって管理される Amazon ECS 用 AWS Fargate) をインストールし、 という名前のモダナイズされたサンプルをインストールすることです PlanetsDemo。

> **Note**
>
> この例の Dockerfile で使用される install-gapwalk.sh および install-app.sh スクリプトの内容は、Dockerfile の後に一覧表示されます。

```
FROM --platform=linux/x86_64 amazonlinux:2

RUN mkdir -p /workdir/apps
WORKDIR /workdir
COPY install-gapwalk.sh .
COPY install-app.sh .
RUN chmod +x install-gapwalk.sh
RUN chmod +x install-app.sh

# Install Java and AWS CLI v2-y
RUN yum install sudo java-17-amazon-corretto unzip tar -y
RUN sudo yum remove awscli -y
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
 "awscliv2.zip"
RUN sudo unzip awscliv2.zip
RUN sudo ./aws/install

#·Installation dir
RUN mkdir -p /usr/local/velocity/installation/gapwalk
# Copy PlanetsDemo archive to a dedicated apps dir
COPY PlanetsDemo-v1.zip /workdir/apps/

# Copy resources (tomcat, blu age runtime) to installation dir
COPY tomcat.tar.gz /usr/local/velocity/installation/tomcat.tar.gz
COPY aws-bluage-on-fargate-runtime-3.10.0.15.tar.gz /usr/local/velocity/
installation/gapwalk/gapwalk-bluage-on-fargate.tar.gz

# run relevant installation scripts
RUN ./install-gapwalk.sh
RUN ./install-app.sh

EXPOSE 8080
EXPOSE 8081
# ...
```

```
# Run Command to start Tomcat server
CMD ["sh", "-c", "sudo /bluage-on-fargate/tomcat.gapwalk/velocity/startup.sh
 $ECS_CONTAINER_METADATA_URI_V4 $AWS_CONTAINER_CREDENTIALS_RELATIVE_URI"]
```

install-gapwalk.sh の内容は次のとおりです。

```
#!/bin/sh

# Vars
TEMP_DIR=/bluage-on-fargate/tomcat.gapwalk/temp

# Install
echo "Installing Gapwalk and Tomcat"
sudo rm -rf /bluage-on-fargate
mkdir -p ${TEMP_DIR}
# Copy Blu Age runtime and tomcat archives to temporary extraction dir
sudo cp /usr/local/velocity/installation/gapwalk/gapwalk-bluage-on-fargate.tar.gz
 ${TEMP_DIR}
sudo cp /usr/local/velocity/installation/tomcat.tar.gz ${TEMP_DIR}
#·Create velocity dir
mkdir -p /bluage-on-fargate/tomcat.gapwalk/velocity
#·Extract tomcat files
tar -xvf ${TEMP_DIR}/tomcat.tar.gz -C ${TEMP_DIR}
# Copy all tomcat files to velocity dir
cp -fr ${TEMP_DIR}/apache-tomcat-9.x.x/* /bluage-on-fargate/tomcat.gapwalk/velocity
# Remove default webapps of Tomcat
rm -f /bluage-on-fargate/tomcat.gapwalk/velocity/webapps/*
# Extract Blu Age runtime at velocity dir
tar -xvf ${TEMP_DIR}/gapwalk-bluage-on-fargate.tar.gz -C /bluage-on-fargate/
tomcat.gapwalk
# Remove temporary extraction dir
sudo rm -rf ${TEMP_DIR}
```

install-app.sh の内容は次のとおりです。

```
#!/bin/sh

APP_DIR=/workdir/apps
TOMCAT_GAPWALK_DIR=/bluage-on-fargate/tomcat.gapwalk

unzip ${APP_DIR}/PlanetsDemo-v1.zip -d ${APP_DIR}
cp -r ${APP_DIR}/webapps/* ${TOMCAT_GAPWALK_DIR}/velocity/webapps/
```

```
cp -r ${APP_DIR}/config/* ${TOMCAT_GAPWALK_DIR}/velocity/config/
```

4. {TOMCAT_GAPWALK_DIR}/config フォルダにある application-main.ymlファイルの次のスニペットの前提条件の一部として作成したデータベースの接続情報を指定します。詳細については、「Aurora PostgreSQL DB クラスターの作成と接続」を参照してください。

```
datasource:
   jicsDs:
     driver-class-name :
     url:
     username:
     password:
     type :
```

5. イメージをビルドして Amazon ECR リポジトリにプッシュします。手順については、Amazon Elastic Container Registry ユーザーガイドの「Docker イメージのプッシュ」を参照してください。
6. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
7. 左側のナビゲーションペインで、[タスク定義] をクリックします。
8. 起動タイプ で、 を選択します AWS Fargate。
9. の一部として作成したタスクロールを選択しますthe section called “インフラストラクチャセットアップ要件”。
10. イメージをコンテナにアタッチします。
11. フォームへの入力を完了し、 の作成を選択します。
12. 左側のナビゲーションペインで、クラスター を選択し、リストからクラスターを選択します。
13. クラスターの詳細ページの「サービス」タブで、「 の作成」を選択します。
14. タスク定義を選択します。
15. ネットワークセクションを展開し、 の一部として作成した VPC、サブネット、セキュリティグループを設定しますthe section called “インフラストラクチャセットアップ要件”。
16. Amazon ECS サービスをデプロイします。

デプロイが失敗した場合は、ログを確認します。それらを見つけるには、 によって管理される Amazon ECS のタスクページに移動し AWS Fargate、ログタブを選択します。C で始まり、その後に CTAK などの数字が続くエラーコードが見つかった場合は、エラーメッセージを書き留めます。例えば、エラーコード C5102 は、インフラストラクチャ設定が正しくないことを示す一般的なエラーです。実行中のタスク内を移動し、 AWS Blu Age ランタイム (Amazon EC2) に似たいくつか

のコマンドを実行することもできます。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS Exec を使用したデバッグ」を参照してください。

インタラクティブシェルを開くには、ローカルマシンから次のコマンドを実行します。

```
aws ecs execute-command --cluster your_cluster_name --container your_container_name --
task task_id --interactive --command /bin/sh
```

PlanetsDemo アプリケーションをテストする

デプロイされた PlanetsDemo アプリケーションのステータスを確認するには、load-balancer-DNS-name、、 をセットアップに適したlistener-portweb-binary-name値に置き換えた後に、次のコマンドを実行します。

```
curl http://load-balancer-DNS-name:listener-port/gapwalk-application/
```

アプリケーションが実行されている場合は、 の出力メッセージが表示されますJics application is running。

次に、次のコマンドを実行します。

```
curl http://load-balancer-DNS-name:listener-port/jac/api/services/rest/jicsservice/
```

アプリケーションが実行されている場合は、次の出力メッセージが表示されます: Jics application is running。

```
Jics application is running
```

Blusam を設定している場合、次のコマンドを実行すると空のレスポンスが期待できます。

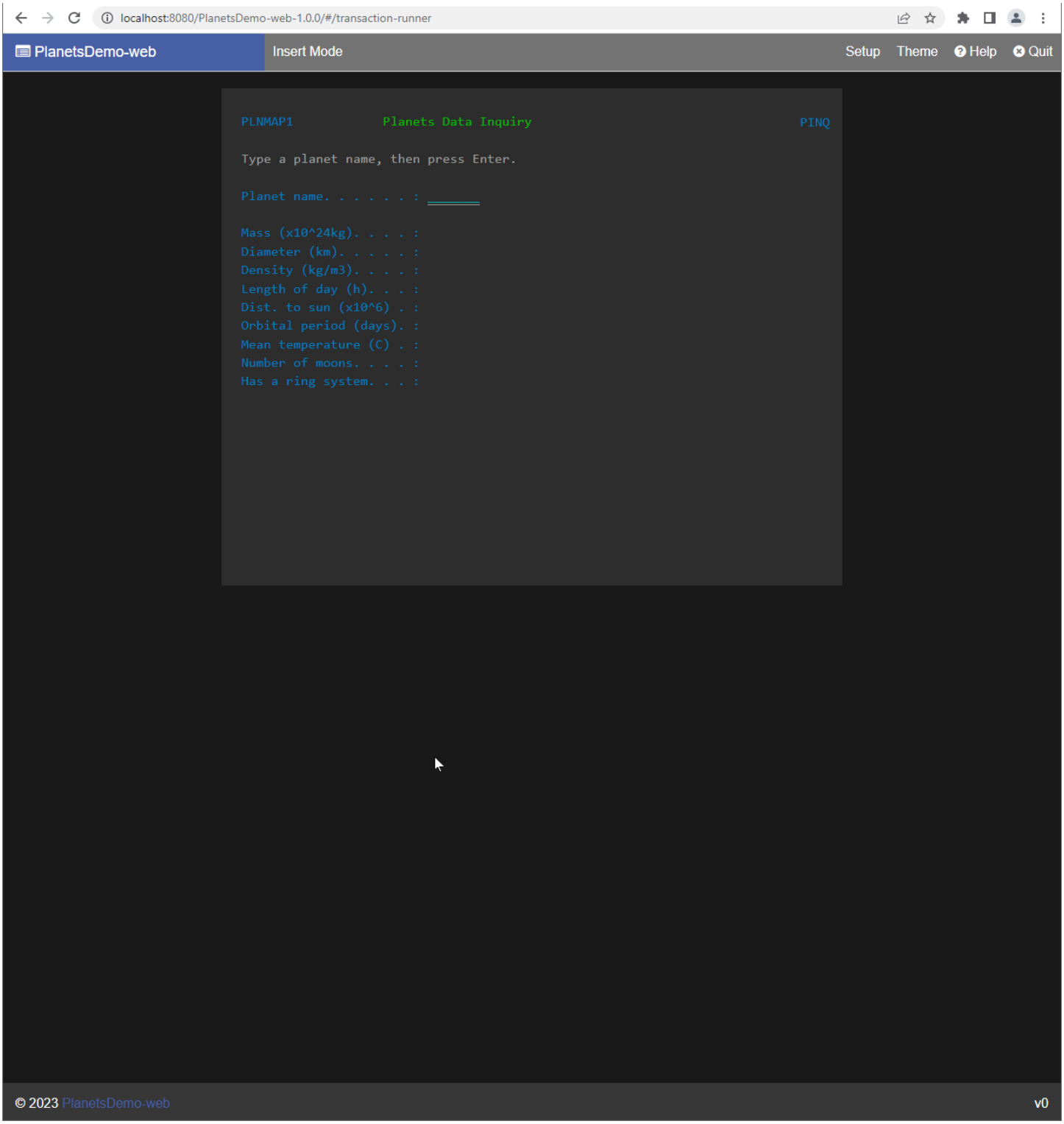
```
curl http://load-balancer-DNS-name:listener-port/bac/api/services/rest/bluesamserver/
serverIsUp
```

ウェブバイナリの名前を書き留めます (PlanetsDemo-web-1.0.0、変更がない場合は)。アプリケーションにアクセスするには PlanetsDemo、次の形式の URL を使用します。

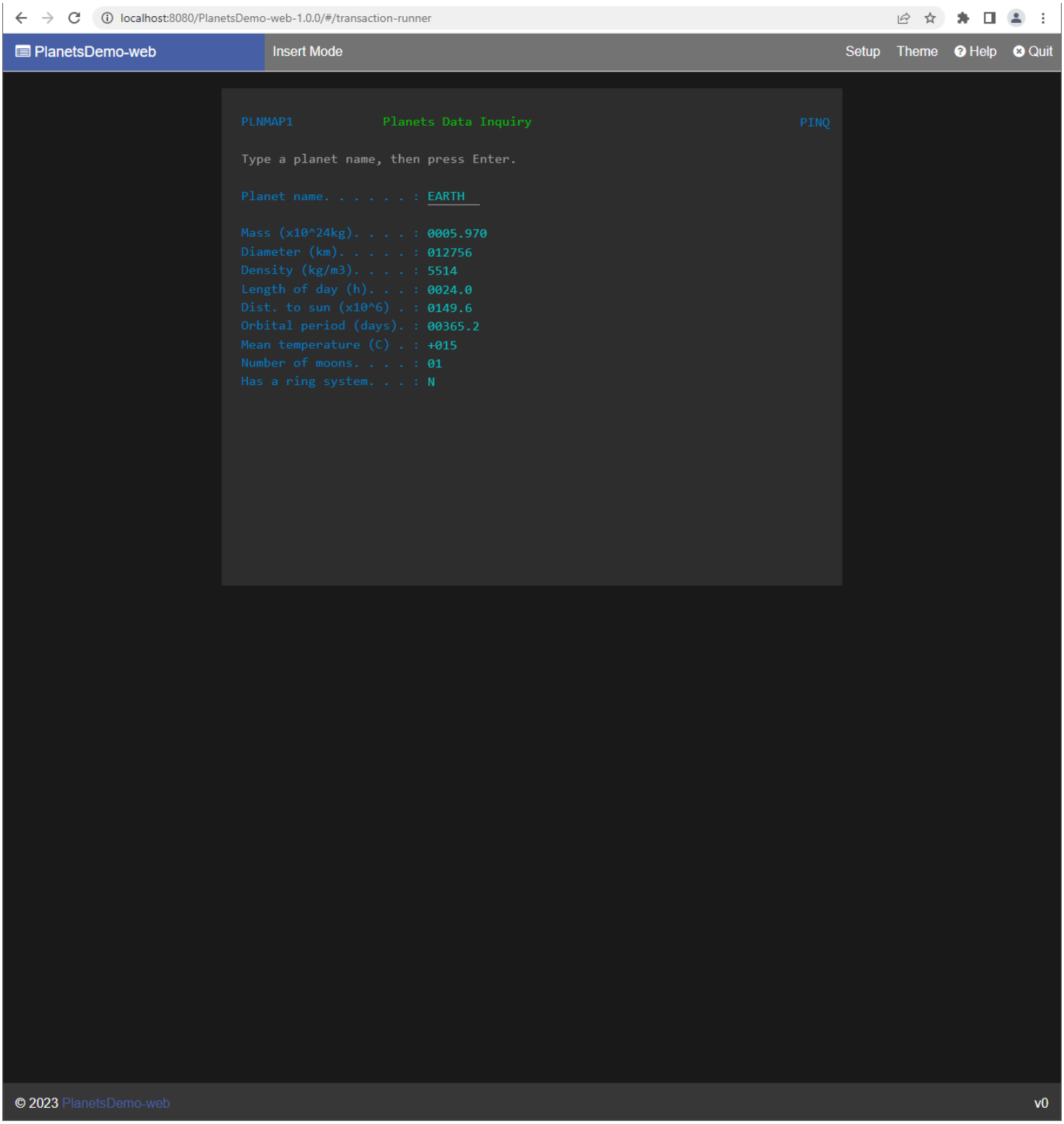
```
https://load-balancer-DNS-name:listener-port/web-binary-name
```

PlanetsDemo アプリケーションが起動すると、ホームページが表示されます。

テキストボックスで、「PINQ」と入力して、Enter キーを押します。データ問い合わせページが表示されます。



例えば、 PlanetsDemo 名前フィールドに EARTH と入力し、Enter キーを押します。入力した惑星のページが表示されます。


localhost:8080/PlanetsDemo-web-1.0.0/#/transaction-runner
PlanetsDemo-web
Insert Mode
Setup
Theme
Help
Quit
PLNMAP1 Planets Data Inquiry PINQ
Type a planet name, then press Enter.
Planet name. . . . . : EARTH
Mass (x10^24kg). . . . : 0005.970
Diameter (km). . . . . : 012756
Density (kg/m3). . . . : 5514
Length of day (h). . . : 0024.0
Dist. to sun (x10^6) . : 0149.6
Orbital period (days). : 00365.2
Mean temperature (C) . : +015
Number of moons. . . . : 01
Has a ring system. . . : N
© 2023 PlanetsDemo-web
v0

によって管理される Amazon ECS での AWS Blu Age ランタイムのアップグレード AWS Fargate

このガイドでは、 が管理する Amazon ECS で AWS Blu Age ランタイムをアップグレードする方法について説明します AWS Fargate。

トピック

前提条件

開始する前に、以下の前提条件を満たしていることを確認してください。

- 「the section called “AWS Blu Age ランタイムの前提条件”」および「the section called “AWS Blu Age ランタイムオンボーディング”」を完了させます。
- アップグレードする AWS Blu Age ランタイムのバージョンをダウンロードします。詳細については、「the section called “AWS Blu Age ランタイムオンボーディング”」を参照してください。フレームワークは、 `aws-bluage-runtime-x.x.x.x.tar.gz`と の 2 つのバイナリファイルで構成されます`aws-bluage-webapps-x.x.x.x.tar.gz`。

Velocity ランタイムをアップグレードする

Velocity ランタイムをアップグレードするには、以下のステップを完了します。

1. 目的の AWS Blu Age ランタイムバージョンを使用して Docker イメージを再構築します。手順については、「the section called “によって管理される Amazon ECS での AWS Blu Age ランタイムのセットアップ AWS Fargate”」を参照してください。
2. Docker イメージを Amazon ECR リポジトリにプッシュします。
3. Amazon ECS サービスを停止して再起動します。
4. ログを検証します。

AWS Blu Age ランタイムは正常にアップグレードされました。

によって管理される Amazon ECS での AWS Blu Age ランタイムの Amazon CloudWatch アラーム AWS Fargate

デプロイされたアプリケーションで例外が発生するたびに通知をより明確にするには、アプリケーションログを受信する CloudWatch ように を設定し、エラーの可能性を警告するアラームを追加します。

アラームの設定

CloudWatch ログを使用すると、アプリケーションやニーズに応じて、任意の数のメトリクスとアラームを設定できます。

具体的には、Amazon ECS クラスターの作成中に使用状況アラートのプロアクティブアラームを直接設定して、エラーが発生したときに通知を受け取ることができます。 AWS Blu Age 管理システムへの接続のエラーを強調表示するには、ログの文字列「エラー C」に関するメトリクスを追加します。その後、このメトリクスに反応するアラームを定義できます。

が管理する Amazon ECS の AWS Blu Age ランタイムでのライセンス付き依存関係の設定 AWS Fargate

このトピックでは、 が管理する Amazon ECS で AWS Blu Age ランタイムで使用できる追加のライセンス依存関係を設定する方法について説明します AWS Fargate。

トピック

前提条件

開始する前に、以下の前提条件を満たしていることを確認してください。

- 「the section called “AWS Blu Age ランタイムの前提条件”」および「the section called “AWS Blu Age ランタイムオンボーディング”」を完了させます。
- 以下の依存関係をソースから取得します。

Oracle データベース

Oracle データベースドライバーを用意してください。例えば、ojdbc8-19.8.0.0.jar などです。

IBM MQ の接続

IBM MQ クライアントを用意してください。例えば、com.ibm.mq.allclient-9.3.0.15.jar です。

この依存関係バージョンでは、以下の推移的な依存関係も指定してください。

- javax.jms-api-2.0.1.jar
- json-20080701.jar

DDS プリンターファイル

Jasper レポートライブラリを用意してください。例えば、jasperreports-6.16.0.jar などですが、より新しいバージョンには互換性がある場合があります。

この依存関係バージョンでは、以下の推移的な依存関係も指定してください。

- castor-core-1.4.1.jar
- castor-xml-1.4.1.jar
- commons-digester-2.1.jar
- ecj-3.21.0.jar
- itext-2.1.7.js8.jar
- javax.inject-1.jar
- jcommon-1.0.23.jar
- jfreechart-1.0.19.jar

概要

依存関係をインストールして、以下のステップを完了します。

1. 必要に応じて、上記の依存関係のいずれかを Docker イメージビルドフォルダにコピーします。
2. JICS または Blusam データベースが Oracle でホストされている場合は、 で Oracle データベースドライバーを指定します*your-tomcat-path*/extra。
3. Dockerfile で、これらの依存関係を にコピーします*your-tomcat-path*/extra。
4. Docker イメージをビルドし、Amazon ECR にプッシュします。
5. Amazon ECS サービスを停止して再起動します。

6. ログの確認

AWS Amazon EC2 での Blu Age ランタイムデプロイ

このセクションのトピックでは、Amazon EC2 で AWS Blu Age ランタイム (非マネージド) を設定する方法、ランタイムバージョンを更新する方法、Amazon CloudWatch アラームを使用してデプロイをモニタリングする方法、およびライセンスされた依存関係を追加する方法について説明します。

トピック

- Amazon EC2 での AWS Blu Age ランタイム (非マネージド) のセットアップ
- Amazon EC2 での AWS Blu Age ランタイムのアップグレード
- AWS Blu Age ランタイム (Amazon EC2) Amazon CloudWatch アラーム
- Amazon EC2 の AWS Blu Age ランタイムでのライセンス依存関係のセットアップ

Amazon EC2 での AWS Blu Age ランタイム (非マネージド) のセットアップ

このトピックでは、Amazon EC2 で AWS Blu Age ランタイム (非マネージド) を使用して PlanetsDemo サンプルアプリケーションをセットアップしてデプロイする方法について説明します。

トピック

- 前提条件
- 設定
- PlanetsDemo アプリケーションをテストする

前提条件

開始する前に、以下の前提条件を満たしていることを確認してください。

- AWS CLI 「AWS CLI の設定」の手順に従って を設定します。
- 「the section called “AWS Blu Age ランタイムの前提条件”」および「the section called “AWS Blu Age ランタイムオンボーディング”」を完了させます。
- 最新の AWS Blu Age ランタイム (Amazon EC2 上) を含む Amazon EC2 インスタンスを作成します。詳細については、「Amazon EC2 Linux インスタンスの開始方法」を参照してください。
- SSM を使用するなどして、Amazon EC2 インスタンスに正常に接続できることを確認します。

- で AWS Blu Age ランタイム (Amazon EC2 上) をダウンロードして抽出します*your-tomcat-path*/*。手順については、「the section called “AWS Blu Age ランタイムオンボーディング”」を参照してください。
- PlanetsDemo アプリケーションアーカイブ をダウンロードします。
- アーカイブを解凍し、選択した Amazon S3 バケットにアプリケーションをアップロードします。
- JICS 用の Amazon Aurora PostgreSQL データベースを作成し、そこでPlanetsDemo-v1/jics/sql/initJics.sqlクエリを実行します。Amazon Aurora PostgreSQL データベースの作成方法については、「Aurora PostgreSQL DB クラスターの作成と接続」を参照してください。

設定

PlanetsDemo サンプルアプリケーションをセットアップするには、次のステップを実行します。

1. Amazon EC2 インスタンスに接続し、Apache Tomcat 9 インストールconfフォルダの下の フォルダに移動します。編集するcatalina.propertiesファイルを開き、 で始まる行を次の行common.loaderに置き換えます。

```
common.loader="${catalina.base}/lib","${catalina.base}/lib/
*.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar","${catalina.home}/
shared","${catalina.home}/shared/*.jar","${catalina.home}/extra","${catalina.home}/
extra/*.jar"
```

2. *<your-tomcat-path>*/webapps フォルダに移動します。
3. 次のコマンドを使用して、Amazon S3 バケットから PlanetsDemo-v1/webapps/ フォルダにある PlanetsDemo バイナリをコピーします。

```
aws s3 cp s3://path-to-demo-app-webapps/ . --recursive
```

> **Note**
>
> を、以前に PlanetsDemo アーカイブを解凍したバケットの正しい Amazon S3 URI path-to-demo-app-webappsに置き換えます。

4. PlanetsDemo-v1/config/ フォルダの内容を *<your-tomcat-path>*/config/ にコピーします。

5. ファイル内の次のスニペットの前提条件の一部として作成したデータベースの接続情報を指定しますapplication-main.yml。詳細については、「Aurora PostgreSQL DB クラスターの作成と接続」を参照してください。

```
datasource:
   jicsDs:
     driver-class-name :
     url:
     username:
     password:
     type :
```

6. Apache Tomcat サーバーを起動し、ログを確認します。

```
your-tomcat-path/startup.sh

tail -f your-tomcat-path/logs/catalina.log
```

C で始まり、その後に CTAK などの数字が続くエラーコードが見つかった場合は、エラーメッセージを書き留めます。例えば、エラーコード C5102 は、インフラストラクチャ設定が正しくないことを示す一般的なエラーです。

PlanetsDemo アプリケーションをテストする

デプロイされた PlanetsDemo アプリケーションのステータスを確認するには、load-balancer-DNS-name、、 をセットアップに適したlistener-portweb-binary-name値に置き換えた後に、次のコマンドを実行します。

```
curl http://load-balancer-DNS-name:listener-port/gapwalk-application/
```

アプリケーションが実行されている場合は、 の出力メッセージが表示されますJics application is running。

次に、次のコマンドを実行します。

```
curl http://load-balancer-DNS-name:listener-port/jac/api/services/rest/jicsservice/
```

アプリケーションが実行されている場合は、 の出力メッセージが表示されますJics application is running。

```
Jics application is running
```

Blusam を設定している場合、次のコマンドを実行すると空のレスポンスが期待できます。

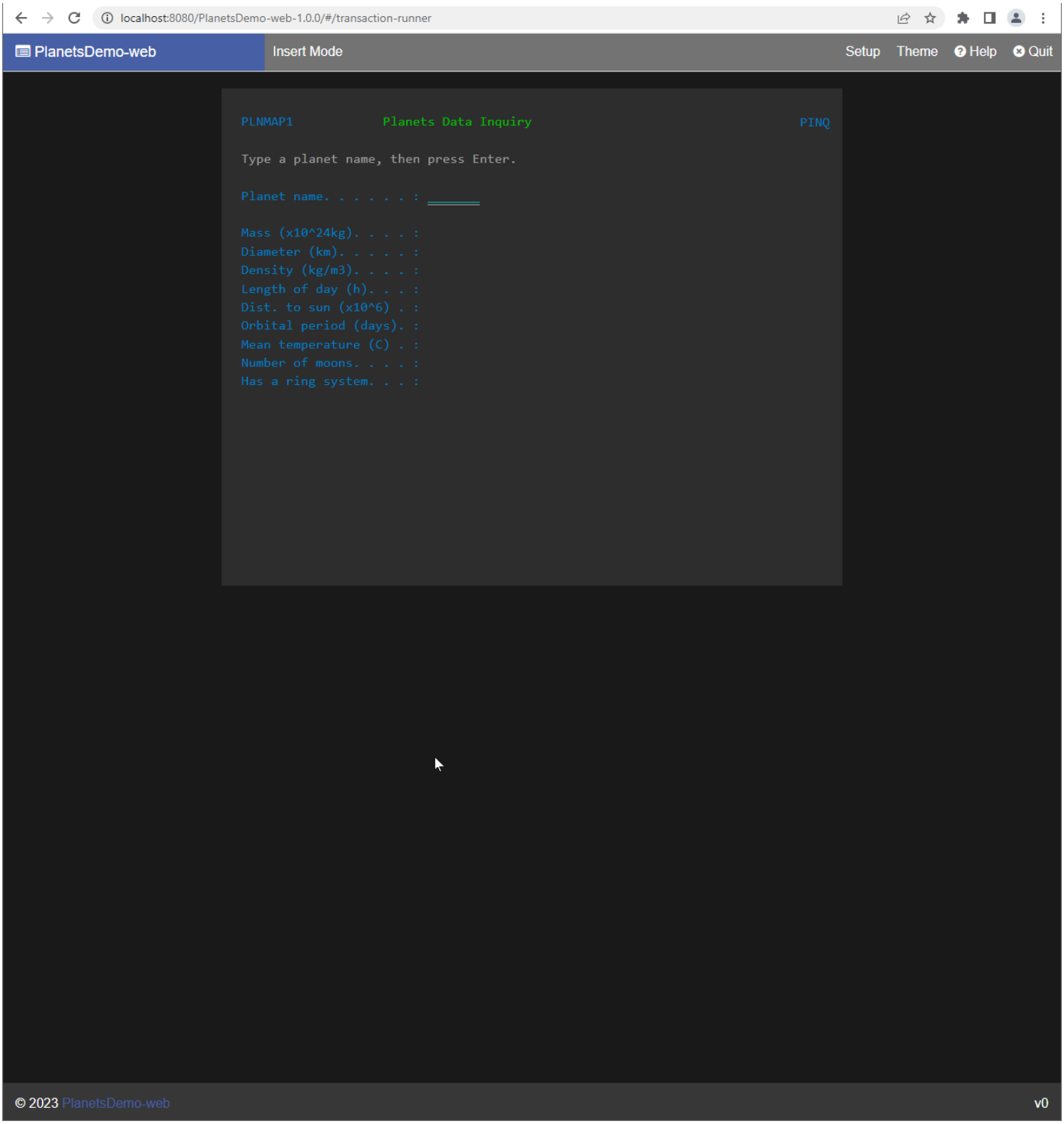
```
curl http://load-balancer-DNS-name:listener-port/bac/api/services/rest/bluesamserver/
serverIsUp
```

ウェブバイナリの名前を書き留めます (PlanetsDemo-web-1.0.0、変更しない場合は)。
PlanetsDemo アプリケーションにアクセスするには、次の形式の URL を使用します。

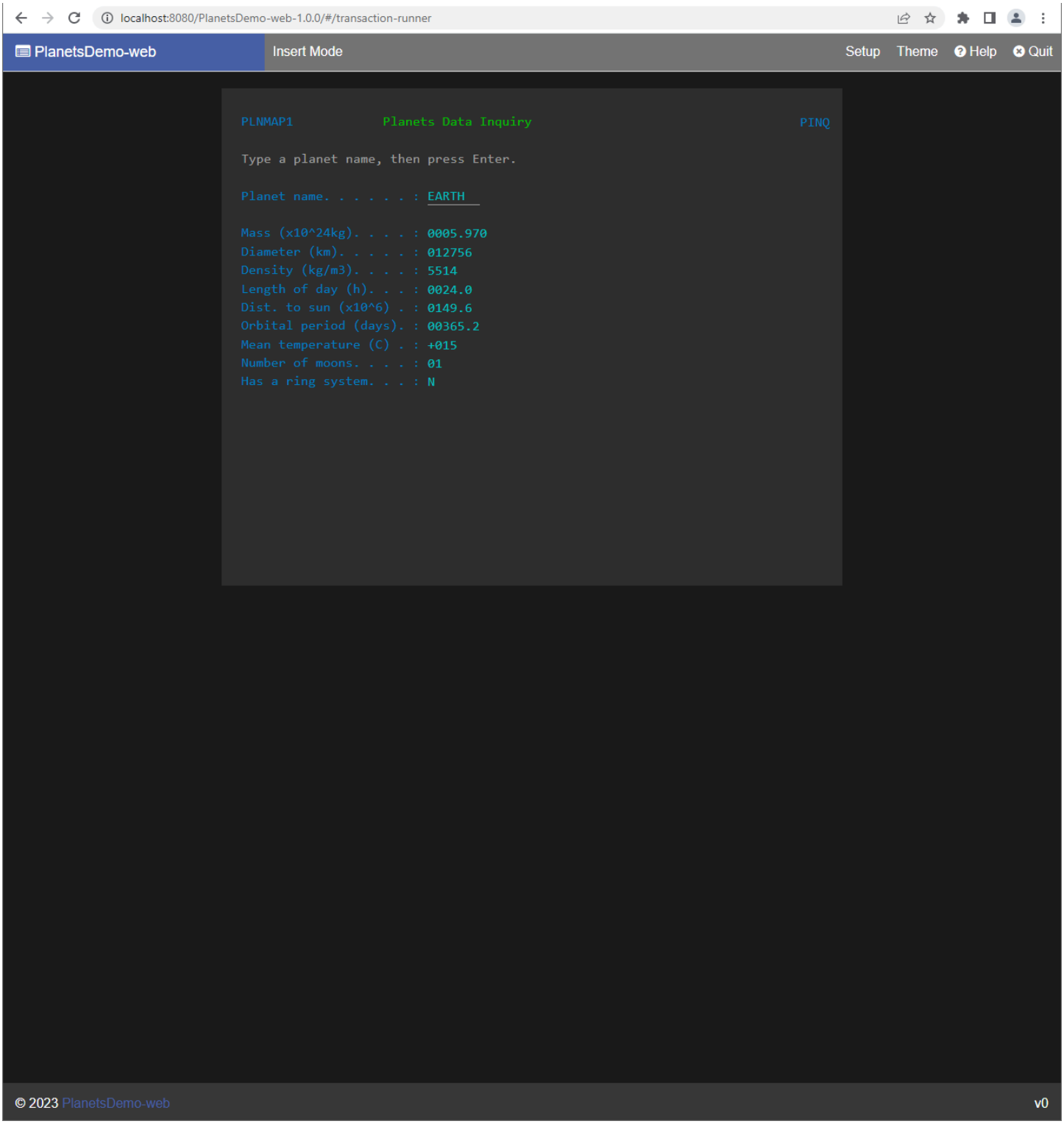
```
https://load-balancer-DNS-name:listener-port/web-binary-name
```

PlanetsDemo アプリケーションが起動すると、ホームページが表示されます。

テキストボックスで、「PINQ」と入力して、Enter キーを押します。データ問い合わせページが表示されます。



例えば、 PlanetsDemo 名前フィールドに EARTH と入力し、Enter キーを押します。入力した惑星のページが表示されます。



Amazon EC2 での AWS Blu Age ランタイムのアップグレード

このガイドでは、Amazon EC2 で AWS Blu Age ランタイムをアップグレードする方法について説明します。

トピック

前提条件

開始する前に、以下の前提条件を満たしていることを確認してください。

- 「the section called “AWS Blu Age ランタイムの前提条件”」および「the section called “AWS Blu Age ランタイムオンボーディング”」を完了させます。
- 最新の AWS Blu Age ランタイムを含む Amazon EC2 インスタンスがあることを確認します。詳細については、「Amazon EC2 Linux インスタンスの開始方法」を参照してください。
- SSM を使用するなどして、Amazon EC2 インスタンスに正常に接続できることを確認します。
- アップグレードする AWS Blu Age ランタイム (Amazon EC2 上) のバージョンをダウンロードします。詳細については、「the section called “AWS Blu Age ランタイム (非マネージド) セットアップ”」を参照してください。フレームワークは 2 つのバイナリファイルで構成されます (`aws-bluage-runtime-x.x.x.x.tar.gz` および `aws-bluage-webapps-x.x.x.x.tar.gz`)。

概要

Velocity ランタイムをアップグレードするには、以下のステップを完了します。

1. 次のコマンドを実行して、Amazon EC2 インスタンスに接続し、ユーザーを su に変更します。

   ```
   sudo su
   ```

   このチュートリアルのコマンドを実行するには、スーパーユーザー権限が必要です。
2. バイナリファイルごとに 1 つずつ、合計 2 つのフォルダを作成します。
3. 各フォルダに、バイナリファイルと同じ名前を付けます。
4. 各バイナリファイルを対応するフォルダにコピーします。

   > **Warning**
   >
   > 各バイナリを抽出すると、同じ名前のフォルダが生成されます。したがって、両方のバイナリファイルを同じ場所で次々に抽出すると、コンテンツが上書きされます。

5. バイナリを抽出するには、次のコマンドを使用します。各フォルダでコマンドを実行します。

```
tar xvf aws-bluage-runtime-x.x.x.x.tar.gz
tar xvf aws-bluage-webapps-x.x.x.x.tar.gz
```

6. 次のコマンドを使用して、Apache Tomcat サービスを停止します。

```
systemctl stop tomcat.service
systemctl stop tomcat-webapps.service
```

7. <your-tomcat-path>/shared/ の内容を aws-bluage-runtime-x.x.x.x/velocity/shared/ の内容に置き換えます。
8. <your-tomcat-path>/webapps/gapwalk-application.war を aws-bluage-runtime-x.x.x.x/velocity/webapps/gapwalk-application.war に置き換えます。
9. <your-tomcat-path>/webapps/ の war ファイル、つまり bac.war と jac.war を aws-bluage-webapps-x.x.x.x/velocity/webapps/ の同じファイルに置き換えます。
10. 次のコマンドを実行して、Apache Tomcat サービスを開始します。

```
systemctl start tomcat.service
systemctl start tomcat-webapps.service
```

11. ログの確認

デプロイされたアプリケーションの状態を確認するには、次のコマンドを実行します。

```
curl http://localhost:8080/gapwalk-application/
```

以下のメッセージが表示されます。

```
Jics application is running
```

```
curl http://localhost:8181/jac/api/services/rest/jicsservice/
```

以下のメッセージが表示されます。

```
Jics application is running
```

```
curl http://localhost:8181/bac/api/services/rest/bluesamserver/serverIsUp
```

レスポンスは空である必要があります。

AWS Blu Age ランタイムは正常にアップグレードされました。

AWS Blu Age ランタイム (Amazon EC2) Amazon CloudWatch アラーム

デプロイされたアプリケーションに猶予期間内のアプリケーションを配置する例外が発生した場合に、通知をより明確に表示するには、 を設定 CloudWatchしてアプリケーションログを受信し、エラーの可能性を警告するアラームを追加します。

CloudWatch ログ記録のデプロイ

デフォルトでは、`application-main.yml` ファイルには `logback-cloudwatch.yml` いう名前の別のログ記録設定ファイルへの参照が含まれます。

```
logging:
  config: classpath:logback-cloudwatch.xml
```

次のセクションで説明するように、どちらのファイルも config フォルダにあり、これは CloudWatch ログ記録の設定方法です。

CloudWatch ログ記録の設定

デフォルトの `logback-cloudwatch.xml` ファイルの内容は次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration>
<configuration>

    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS,UTC}  %level --- [%thread{15}]
 %logger{40} : %msg%n%xThrowable</pattern>
        </encoder>
    </appender>

    <appender name="cloudwatch"
 class="com.netfective.bluage.runtime.cloudwatchlogger.CloudWatchAppender">
        <logGroup>BluAgeRuntimeOnEC2-Logs</logGroup>
```

```
        <logStream>%date{yyyy-MM-dd,UTC}.%instanceId.%uuid</logStream>
        <layout>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS,UTC}  %level --- [%thread{15}]
 %logger{40} : %msg%n%xThrowable</pattern>
        </layout>
        <appender-ref ref="console" />
    </appender>

    <root level="INFO">
        <appender-ref ref="cloudwatch" />
    </root>
</configuration>
```

<appender name="cloudwatch"/> 要素の外部は、すべて標準的なログバックの設定です。このファイルには、コンソールにログを送信するコンソールアペンダーと、 にログを送信する CloudWatch アペンダーの 2 つのアペンダーがあります CloudWatch。

`root` 要素の `level` 属性は、アプリケーション全体のログ記録レベルを指定します。

タグ <appender name="cloudwatch"TAK 内の必須値は次のとおりです。

- <logGroup TAK: のロググループの名前を設定します CloudWatch。値が指定されない場合、デフォルトは `BluAgeRuntimeOnEC2-Logs` になります。ロググループが存在しない場合は、自動的に作成されます。この動作は、次で説明する設定によって変更できます。
- <logStream TAK: の logStream の名前 (ロググループ内) を設定します CloudWatch。

オプションの値：

- <region/>: ログストリームが書き込まれるリージョンを無効にします。デフォルトでは、ログは EC2 インスタンスと同じリージョンに送られます。
- <layout/>: ログメッセージが使用するパターン。
- <maxbatchsizeTAK: オペレーション CloudWatch ごとに に送信するログメッセージの最大数。
- <maxbatchtimemillisTAK: CloudWatch ログの書き込みを許可するミリ秒単位の時間。
- <maxqueuewaittimemillis/>: 内部ログキューにリクエストの挿入を試行する時間 (ミリ秒単位)。
- <internalqueuesize/>: 内部キューの最大サイズ。
- <createlogdests/>: 存在しない場合はロググループとログストリームを作成します。
- <initialwaittimemillis/>: スタートアップ時にスレッドを休止させる時間。この最初の待機時間により、最初にログが蓄積されます。

- <maxeventmessagesize/>: ログイベントの最大サイズ。このサイズを超えるログは送信されません。
- <truncateeventmessages/>: 長すぎるメッセージは切り捨てられます。
- <printrejectedevents/>: 緊急アペンダーを有効にします。

CloudWatch セットアップ

上記の設定でログを に正しくプッシュするには CloudWatch、Amazon EC2 IAM インスタンスプロファイルロールを更新して、「BluAgeRuntimeOnEC2-Logs」ロググループとそのログストリームに追加のアクセス許可を付与します。

- `logs:CreateLogStream`
- `logs:DescribeLogStreams`
- `logs:CreateLogGroup`
- `logs:PutLogEvents`
- `logs:DescribeLogGroups`

アラームの設定

CloudWatch ログにより、アプリケーションやニーズに応じてさまざまなメトリクスとアラームを設定できます。具体的には、使用状況の警告に関するプロアクティブアラームを設定して、アプリケーションが猶予期間に入る (そして最終的にはアプリケーションが動作しなくなる) ようなエラーが発生した場合に警告することができます。これを実現するには、ログに「Error C5001」文字列に関するメトリクスを追加できます。これにより、 AWS Blu Age 管理システムへの接続でエラーが強調表示されます。その後、このメトリクスに反応するアラームを定義できます。

Amazon EC2 の AWS Blu Age ランタイムでのライセンス依存関係のセットアップ

このガイドでは、Amazon EC2 の AWS Blu Age ランタイムで使用できる追加のライセンス依存関係を設定する方法について説明します。

トピック

前提条件

開始する前に、以下の前提条件を満たしていることを確認してください。

- 「the section called “AWS Blu Age ランタイムの前提条件”」および「the section called “AWS Blu Age ランタイムオンボーディング”」を完了させます。
- 最新の AWS Blu Age ランタイム (Amazon EC2) を含む Amazon EC2 インスタンスがあることを確認します。詳細については、「Amazon EC2 Linux インスタンスの開始方法」を参照してください。
- SSM を使用するなどして、Amazon EC2 インスタンスに正常に接続できることを確認します。
- ソースから次の依存関係を取得します。

Oracle データベース

Oracle データベースドライバーを用意してください。 AWS Blu Age Runtime (Amazon EC2) の機能をバージョン ojdbc8-19.8.0.0.jar でテストしましたが、より新しいバージョンには互換性がある可能性があります。

IBM MQ の接続

IBM MQ クライアントを用意してください。 AWS Blu Age ランタイム (Amazon EC2) の機能をバージョン com.ibm.mq.allclient-9.3.0.15.jar でテストしましたが、より新しいバージョンには互換性がある可能性があります。

この依存関係バージョンでは、以下の推移的な依存関係も指定してください。

- javax.jms-api-2.0.1.jar
- json-20080701.jar

DDS プリンターファイル

Jasper レポートライブラリを用意してください。jasperreports-6.16.0.jar を使用して AWS Blu Age ランタイム (Amazon EC2) 機能をテストしましたが、より新しいバージョンには互換性がある可能性があります。

この依存関係バージョンでは、以下の推移的な依存関係も指定してください。

- castor-core-1.4.1.jar
- castor-xml-1.4.1.jar

- commons-digester-2.1.jar
- ecj-3.21.0.jar
- itext-2.1.7.js8.jar
- javax.inject-1.jar
- jcommon-1.0.23.jar
- jfreechart-1.0.19.jar

概要

依存関係をインストールして、以下のステップを完了します。

1. 次のコマンドを実行して、Amazon EC2 インスタンスに接続し、ユーザーを su に変更します。

   ```
   sudo su
   ```

   このチュートリアルのコマンドを実行するには、スーパーユーザー権限が必要です。
2. <your-tomcat-path>/extra/ フォルダに移動します。

   ```
   cd <your-tomcat-path>/extra/
   ```
3. 上記の依存関係を必要に応じてこのフォルダにコピーします。
4. 次のコマンドを実行して、tomcat.service を停止して開始します。

   ```
   systemctl stop tomcat.service
   ```

   ```
   systemctl start tomcat.service
   ```
5. サービスのステータスをチェックして、それが実行中であることを確認します。

   ```
   systemctl status tomcat.service
   ```
6. ログを検証します。

JAC と BAC のウェブアプリケーションの依存関係を設定する

1. JICS または Blusam データベースが Oracle でホストされている場合は、<your-tomcat-path>/extra に Oracle データベースドライバーを提供する必要があります。

2. フォルダがまだ存在しない場合は作成してください。
3. Apache Tomcat サーバーを停止して再起動します。
4. ログを検証します。

Blu Age デベロッパー IDE でソースコードを修正する

AWSマネージド AWS Blu Age ランタイムエンジンを使用している場合は、 Blu Age デベロッパーを使用して生成されたソースコードを変更できます。これは、何らかの理由でモダナイズされたコードを更新する必要がある場合や、レガシーソースコードの一部をモダナイズできなかった場合に行います。Amazon AppStream 2.0 から Blu Age Developer にアクセスします。このセクションでは、 AppStream 2.0 で Blu Age デベロッパーを設定する方法について説明します。また、 Blu Age Developer を使用して、サンプルアプリケーション を使用してソースコードを更新する方法についても説明します PlanetsDemo。

トピック

- チュートリアル: AWS Blu Age デベロッパー IDE 用に AppStream 2.0 をセットアップする
- チュートリアル: AppStream 2.0 で AWS Blu Age デベロッパーを使用する

チュートリアル: AWS Blu Age デベロッパー IDE 用に AppStream 2.0 をセットアップする

AWS Mainframe Modernization は、アプリケーションを書き換えることなくユーザーにデスクトップアプリケーションをストリーミングできる、完全マネージド型の安全なアプリケーションストリーミングサービス AppStream AppStream です。 AppStream 2.0 を使用すると、ユーザーは必要なアプリケーションにすばやくアクセスでき、選択したデバイスで応答的でスムーズなユーザーエクスペリエンスを実現できます。 AppStream 2.0 を使用してランタイムエンジン固有のツールをホストすると、お客様のアプリケーションチームはウェブブラウザから直接ツールを使用でき、Amazon S3 バケットまたは CodeCommitリポジトリに保存されているアプリケーションファイルを操作できます。

AppStream 2.0 でのブラウザのサポートについては、「Amazon 2.0 管理ガイド」の「システム要件と機能のサポート (ウェブブラウザ)」を参照してください。 AppStream AppStream 2.0 の使用中に問題が発生した場合は、「Amazon AppStream 2.0 管理ガイド」の「2.0 ユーザーの問題のトラブルシューティング AppStream」を参照してください。

このドキュメントでは、 AppStream 2.0 フリートで AWS Blu Age デベロッパー IDE を設定する方法について説明します。

トピック

前提条件

Blu Age デベロッパー IDE AWS を AppStream 2.0 未満で設定するために必要なアーティファクトを含むアーカイブファイルをダウンロードします。

> **Note**
>
> これは大きなファイルです。オペレーションのタイムアウトに問題がある場合は、Amazon EC2 インスタンスを使用してアップロードとダウンロードのパフォーマンスを向上させることをお勧めします。

ステップ 1: Amazon S3 バケットを作成する

作成する AppStream 2.0 フリート AWS リージョン と同じ に Amazon S3 バケットを作成します。このバケットには、このチュートリアルを完了するために必要なアーティファクトが含まれています。

ステップ 2: S3 バケットにポリシーをアタッチする

このチュートリアル用に作成したバケットに次のポリシーをアタッチします。必ず、MYBUCKET を作成するバケットの実際の名前に置き換えてください。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "AllowAppStream2.0ToRetrieveObjects",
        "Effect": "Allow",
        "Principal": {
            "Service": "appstream.amazonaws.com"
        },
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::MYBUCKET/*"
    }]
}
```

ステップ 3: Amazon S3 バケットにファイルをアップロードする

前提条件でダウンロードしたファイルを解凍し、appstream フォルダをバケットにアップロードします。このフォルダをアップロードすると、バケットに正しい構造が作成されます。詳細については、「Amazon S3 ユーザーガイド」の「オブジェクトのアップロード」を参照してください。

ステップ 4: AWS CloudFormation テンプレートをダウンロードする

次の AWS CloudFormation テンプレートをダウンロードします。 AppStream 2.0 フリートを作成してデータを入力するには、これらのテンプレートが必要です。

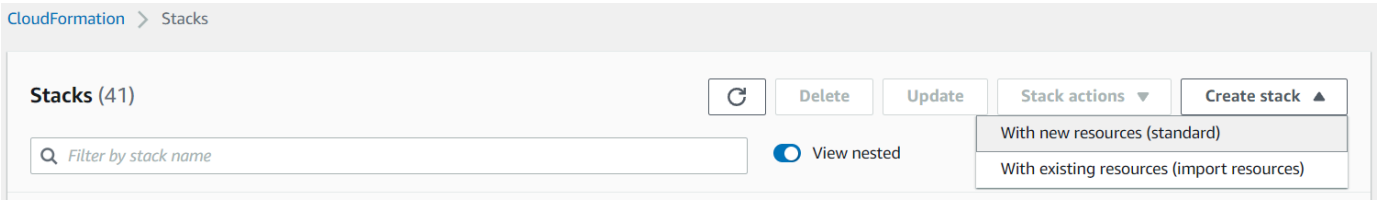
- cfn-m2-appstream-elastic-fleet-linux.yaml
- cfn-m2appstream-bluage-dev-tools--linux.yaml
- cfn-m2-appstream-bluage-shared-linux.yaml
- cfn-m2-appstream-chrome-linux.yaml
- cfn-m2-appstream-eclipse-jee-linux.yaml
- cfn-m2-appstream-pgadmin-linux.yaml

ステップ 5: を使用してフリートを作成する AWS CloudFormation

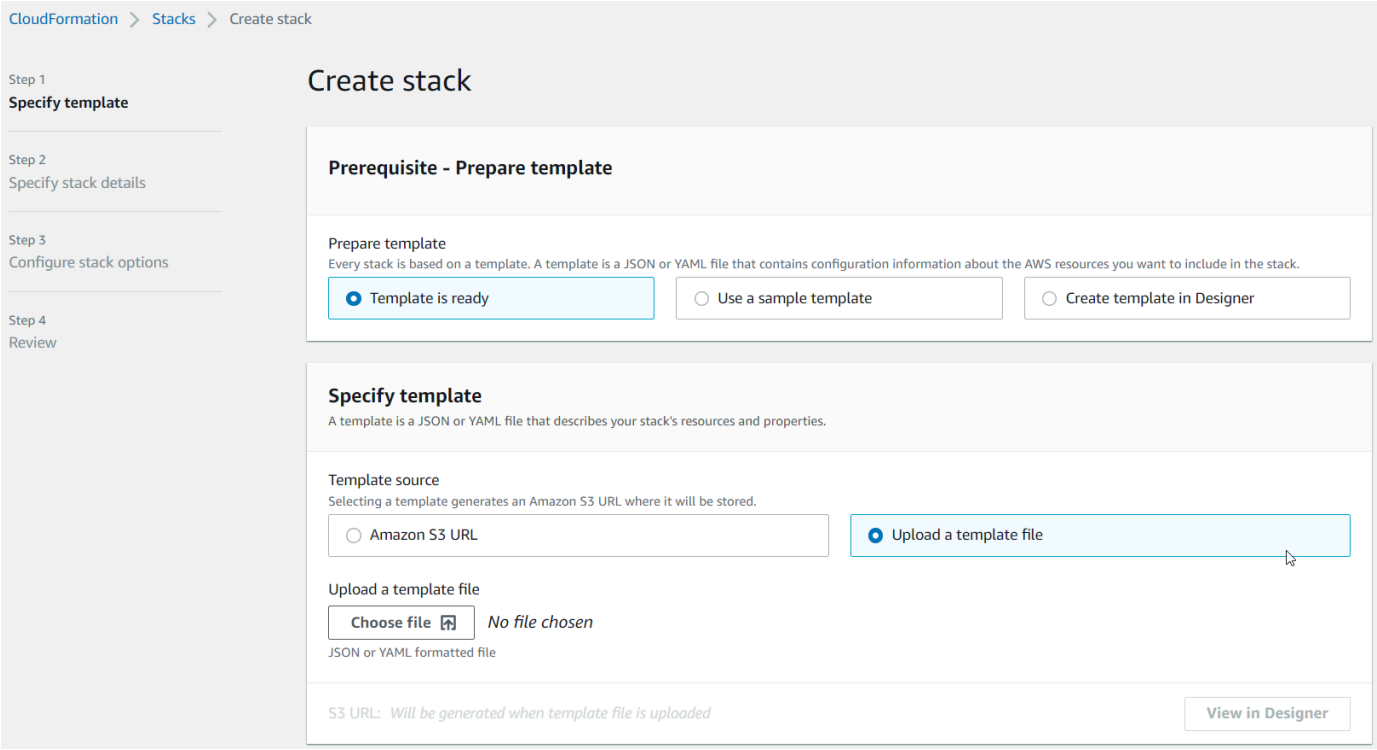
このステップでは、 cfn-m2-appstream-elastic-fleet-linux.yaml AWS CloudFormation テンプレートを使用して AppStream 2.0 フリートを作成し、スタックを使用して AWS Blu Age デベロッパー IDE をホストします。フリートとスタックを作成したら、前のステップでダウンロードした他の AWS CloudFormation テンプレートを実行して、デベロッパー IDE やその他の必要なツールをインストールします。

1. AWS マネジメントコンソール AWS CloudFormation で に移動し、スタック を選択します。

2. [スタック] で、[スタックを作成]、[新しいリソースを使用 (標準)] の順に選択します。



3. [スタックを作成] で、[テンプレートの準備完了] と [テンプレートファイルのアップロード] を選択します。



4. [ファイルを選択] を選択して、ファイル `cfn-m2-appstream-elastic-fleet-linux.yaml` に移動します。[次へ] を選択します。
5. [スタックの詳細を指定] で、次の情報を入力します。

   - スタックの名前です。
   - デフォルトのセキュリティグループと、そのセキュリティグループの 2 つのサブネット。

> **Note**
>
> セキュリティグループの 2 つのサブネットは、異なるアベイラビリティーゾーンにある必要があります。

6. [次へ] を選択し、もう一度 [次へ] を選択します。

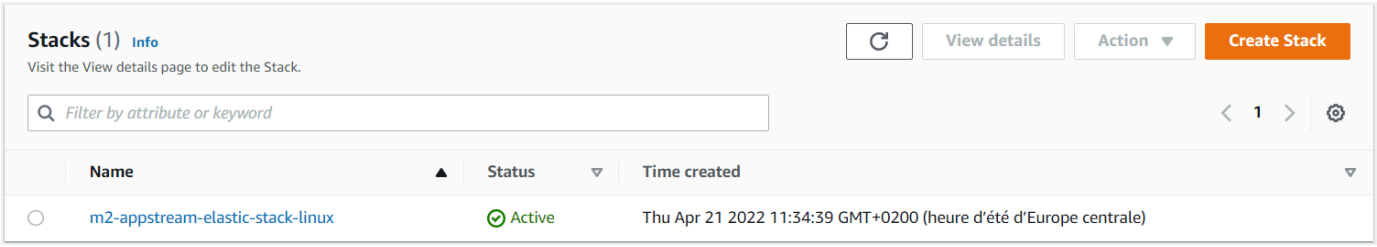
7. がカスタム名で IAM リソースを作成する AWS CloudFormation 場合があることを承認する を選択し、送信 を選択します。
8. フリートを作成したら、ダウンロードした他のテンプレートを使用して CloudFormation スタックを作成し、アプリケーションのセットアップを完了します。正しい S3 バケットを指すようにBucketName毎回更新してください。コンソールBucketNameで CloudFormation を編集できます。または、テンプレートファイルを直接編集し、`S3Bucket` プロパティを更新することもできます。

> **Note**
>
> ダウンロードしたテンプレートは、`appstream/bluage/developer-ide/` というフォルダ構造の S3 バケット内のアセットを検索することを想定しています。バケットは、作成したフリート AWS リージョン と同じ にある必要があります。

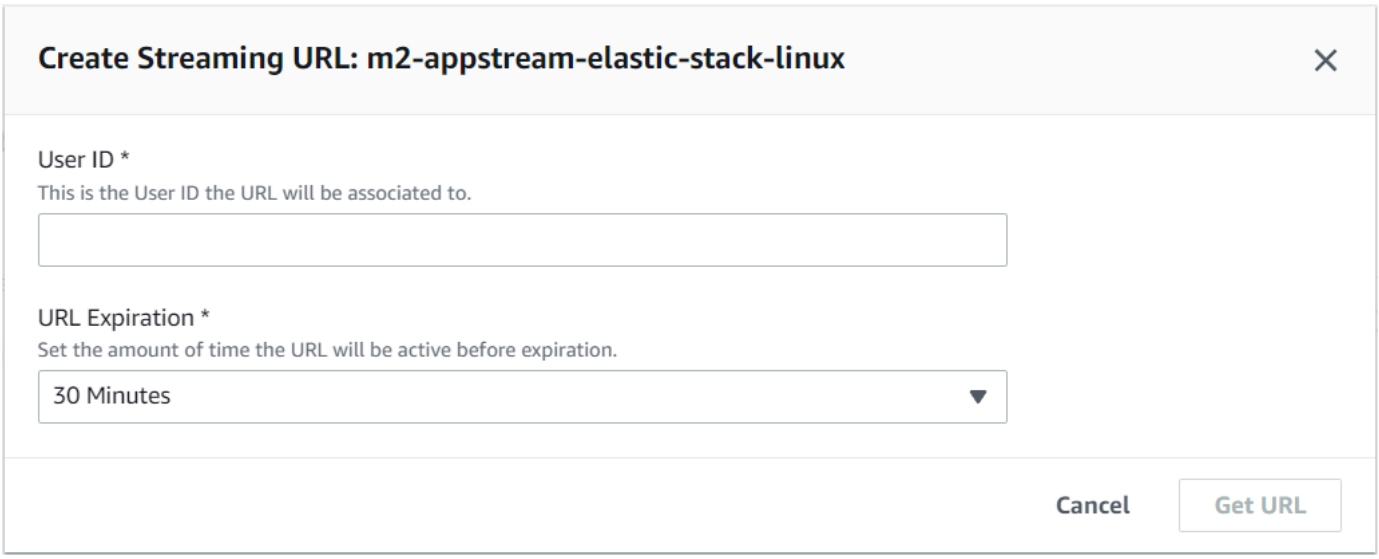
ステップ 6: インスタンスにアクセスする

フリートを作成して開始したら、ネイティブクライアント経由でフリートにアクセスするための一時的なリンクを作成できます。

1. で AppStream 2.0 に移動 AWS Management Console し、以前に作成したスタックを選択します。



2. [スタックの詳細] ページで、[アクション] を選択してから、[ストリーミング URL の作成] を選択します。



3. [ストリーミング URL の作成] で、任意のユーザー ID と URL の有効期限を入力し、[URL を取得] を選択します。ブラウザまたはネイティブクライアントにストリーミングするために使用できる URL を取得します。ネイティブクライアントにストリーミングすることをお勧めします。

リソースをクリーンアップする

作成されたスタックとフリートをクリーンアップする手順については、AppStream 「2.0 フリートとスタックを作成する」を参照してください。

AppStream 2.0 オブジェクトを削除すると、ユーザーまたはアカウント管理者は、アプリケーション設定とホームフォルダの S3 バケットをクリーンアップすることもできます。

> **Note**
>
> 特定のユーザーのホームフォルダはすべてのフリートで一意であるため、他の AppStream 2.0 スタックが同じアカウントでアクティブである場合は、保持する必要がある場合があります。

AppStream 2.0 コンソールを使用してユーザーを削除することはできません。代わりに、 AWS CLI でサービス API を使用する必要があります。詳細については、「Amazon 2.0 管理ガイド」の「ユーザープール管理」を参照してください。 AppStream

チュートリアル: AppStream 2.0 で AWS Blu Age デベロッパーを使用する

このチュートリアルでは、 AWS Blu Age Developer on AppStream 2.0 にアクセスしてサンプルアプリケーションで使用する方法を示します。これにより、機能を試すことができます。このチュートリアルを終了すると、同じ手順を独自のアプリケーションで使用できます。

トピック

ステップ 1: データベースを作成する

このステップでは、Amazon RDS を使用して、デモアプリケーションが設定情報を保存するために使用するマネージド PostgreSQL データベースを作成します。

1. Amazon RDS コンソールを開きます。
2. [データベース] > [データベースの作成] を選択します。
3. [標準作成] > [PostgreSQL] を選択し、デフォルトバージョンのままにして、[無料利用枠] を選択します。
4. DB インスタンス識別子を選択します。
5. [認証情報の設定] で、[AWS Secrets Manager でのマスター資格情報の管理] を選択します。詳細については、「Amazon RDS ユーザーガイド」の「Amazon RDS および AWS Secrets Manager でのパスワード管理」を参照してください。
6. VPC が AppStream 2.0 インスタンスに使用する VPC と同じであることを確認します。この値は管理者に問い合わせることができます。

7. [VPC セキュリティグループ] で、[新規作成] を選択します。
8. [パブリックアクセス] を [はい] に設定します。
9. 他はすべてデフォルト値のままにしておきます。これらの値を確認してください。
10. [データベースの作成] を選択します。

インスタンスからデータベースサーバーにアクセスできるようにするには、Amazon RDS のデータベースサーバーを選択します。[接続とセキュリティ] で、データベースサーバーの VPC セキュリティグループを選択します。このセキュリティグループは以前に作成したもので、RDS マネジメントコンソールで作成の説明と類似の説明が付いているはずです。[アクション] > [インバウンドルールの編集] を選択し、[ルールを追加] を選択して、PostgreSQL タイプのルールを作成します。ルールソースには、セキュリティグループのデフォルトを使用します。最初に [ソース] フィールドにソース名を入力し、提示された ID を受け入れます。最後に、[ルールを保存] を選択します。

ステップ 2: 環境にアクセスする

このステップでは、 AppStream 2.0 の AWS Blu Age 開発環境にアクセスします。

1. AppStream 2.0 インスタンスへの適切なアクセス方法については、管理者にお問い合わせください。可能なクライアントと設定の一般的な情報については、AppStream 「Amazon 2.0 管理ガイド」の「2.0 アクセス方法とクライアント」を参照してください。 AppStream 最適なエクスペリエンスを得るには、ネイティブクライアントの使用を検討してください。
2. AppStream 2.0 では、デスクトップ を選択します。

ステップ 3: ランタイムを設定する

このステップでは、AWS Blu Age ランタイムを設定します。初回起動時にランタイムを設定し、ランタイムのアップグレードを通知された場合は再度設定する必要があります。このステップにより `.m2` フォルダにデータが入力されます。

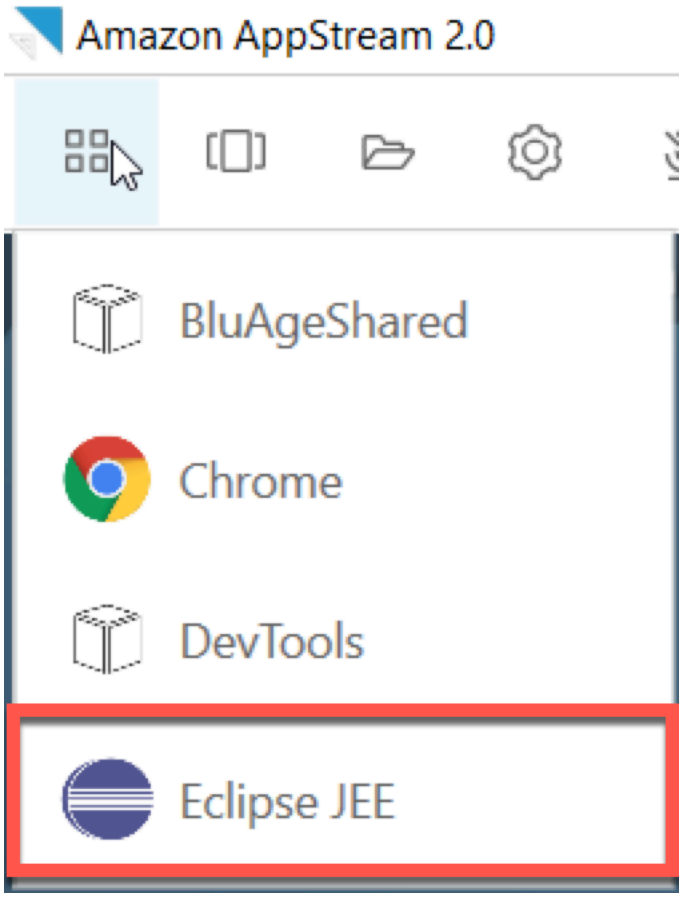
1. メニューバーから [アプリケーション] を選択し、[ターミナル] を選択します。
2. 次のコマンドを入力します。

```
~/_install-velocity-runtime.sh
```

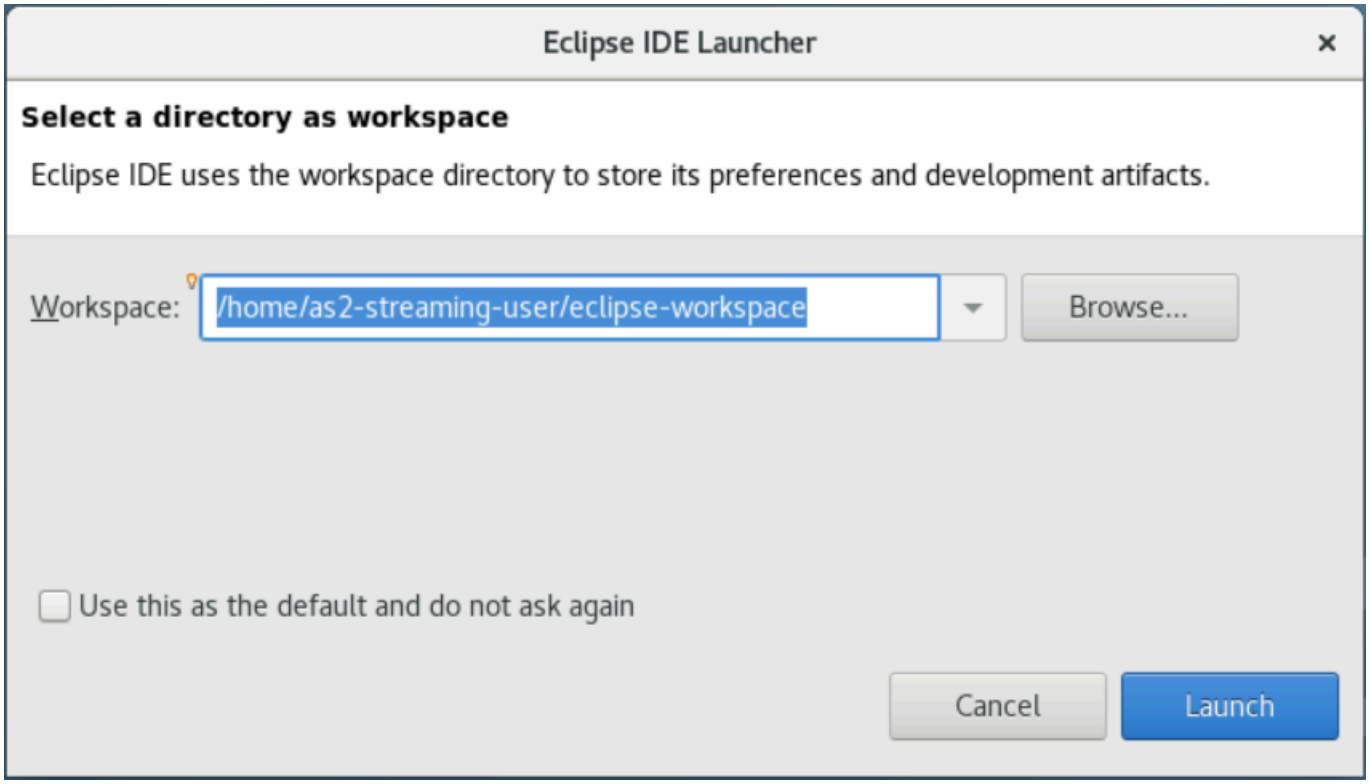
ステップ 4: Eclipse IDE を起動する

このステップでは、Eclipse IDE を起動し、ワークスペースを作成する場所を選択します。

1. AppStream 2.0 では、ツールバーのアプリケーション起動アイコンを選択し、次に Eclipse JEE を選択します。



2. ランチャーが開いたら、ワークスペースを作成する場所を入力し、[起動する] を選択します。



オプションで、以下のように Eclipse をコマンドラインから起動できます。

```
~/eclipse &
```

ステップ 5: Maven プロジェクトをセットアップする

このステップでは、Planets デモアプリケーション用の Maven プロジェクトをインポートします。

1. PlanetsDemo-pom.zip をホームフォルダにアップロードします。これには、ネイティブクライアントの「My Files」機能を使用できます。
2. `unzip` コマンドラインツールを使用してファイルを抽出します。
3. 解凍したフォルダ内に移動し、テキストエディタでプロジェクトのルート `pom.xml` を開きます。
4. インストールされている AWS Blu Age ランタイムと一致するように `gapwalk.version` プロパティを編集します。

   インストールされているバージョンがわからない場合は、ターミナルで次のコマンドを実行します。

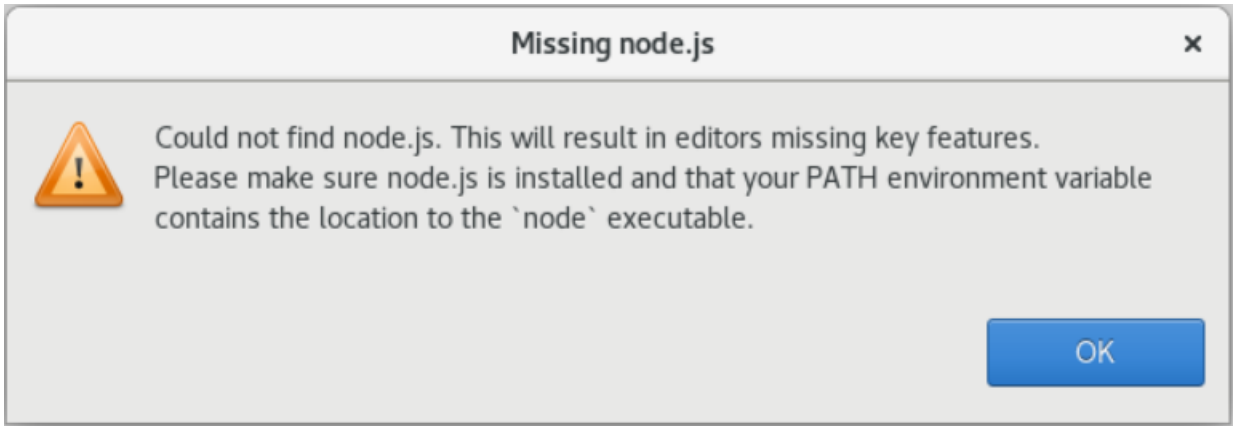
```
cat ~/runtime-version.txt
```

このコマンドは、現在利用可能なランタイムバージョン (例: `3.1.0-b3257-dev`) を出力します。

> **Note**
>
> `gapwalk.version` に `-dev` サフィックスを含めないでください。例えば、有効な値は `<gapwalk.version>3.1.0-b3257</gapwalk.version>` となります。

5. Eclipse で、[ファイル]、[インポート] の順に選択します。[インポート] ダイアログウィンドウで Maven を展開し、[既存の Maven プロジェクト] を選択します。[次へ] をクリックします。
6. [Maven プロジェクトのインポート] で、抽出したファイルの場所を指定し、[完了] を選択します。

   次のポップアップは無視しても問題ありません。Maven は `node.js` のローカルコピーをダウンロードして、プロジェクトの Angular (*-web) 部分をビルドします。

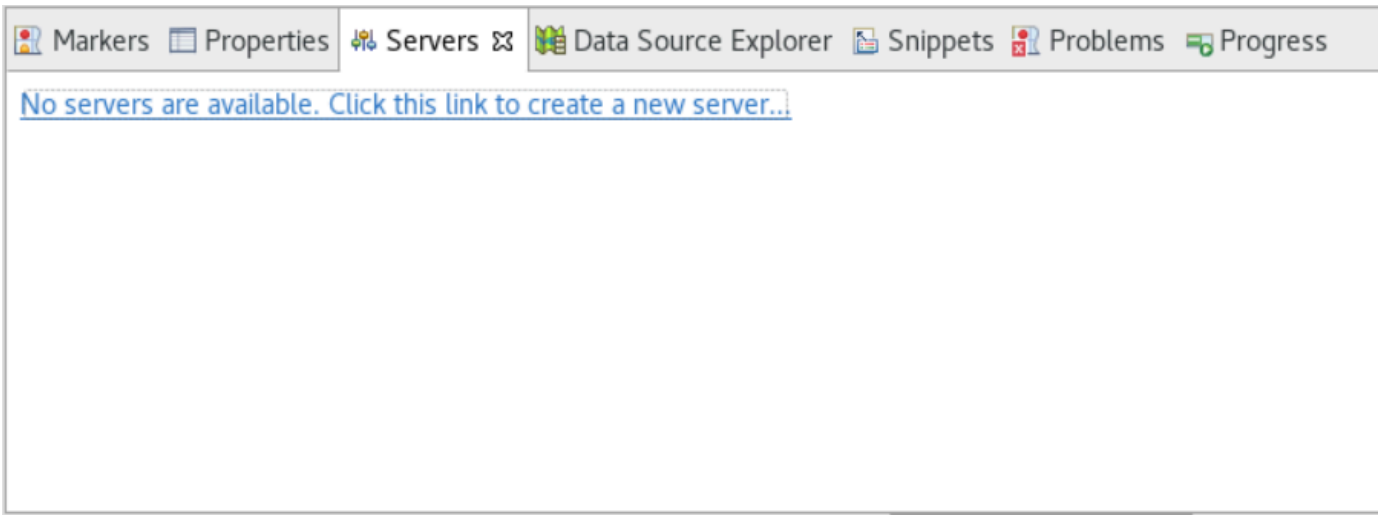


   ビルドが終了するまでお待ちください。[進行状況] ビューでビルドをフォローできます。
7. Eclipse では、プロジェクトを選択し、[として実行] を選択します。次に、[Maven インストール] を選択します。Maven のインストールが成功すると、`PlanetsDemoPom/PlanetsDemo-web/target/PlanetsDemo-web-1.0.0.war` の下に `war` ファイルが作成されます。

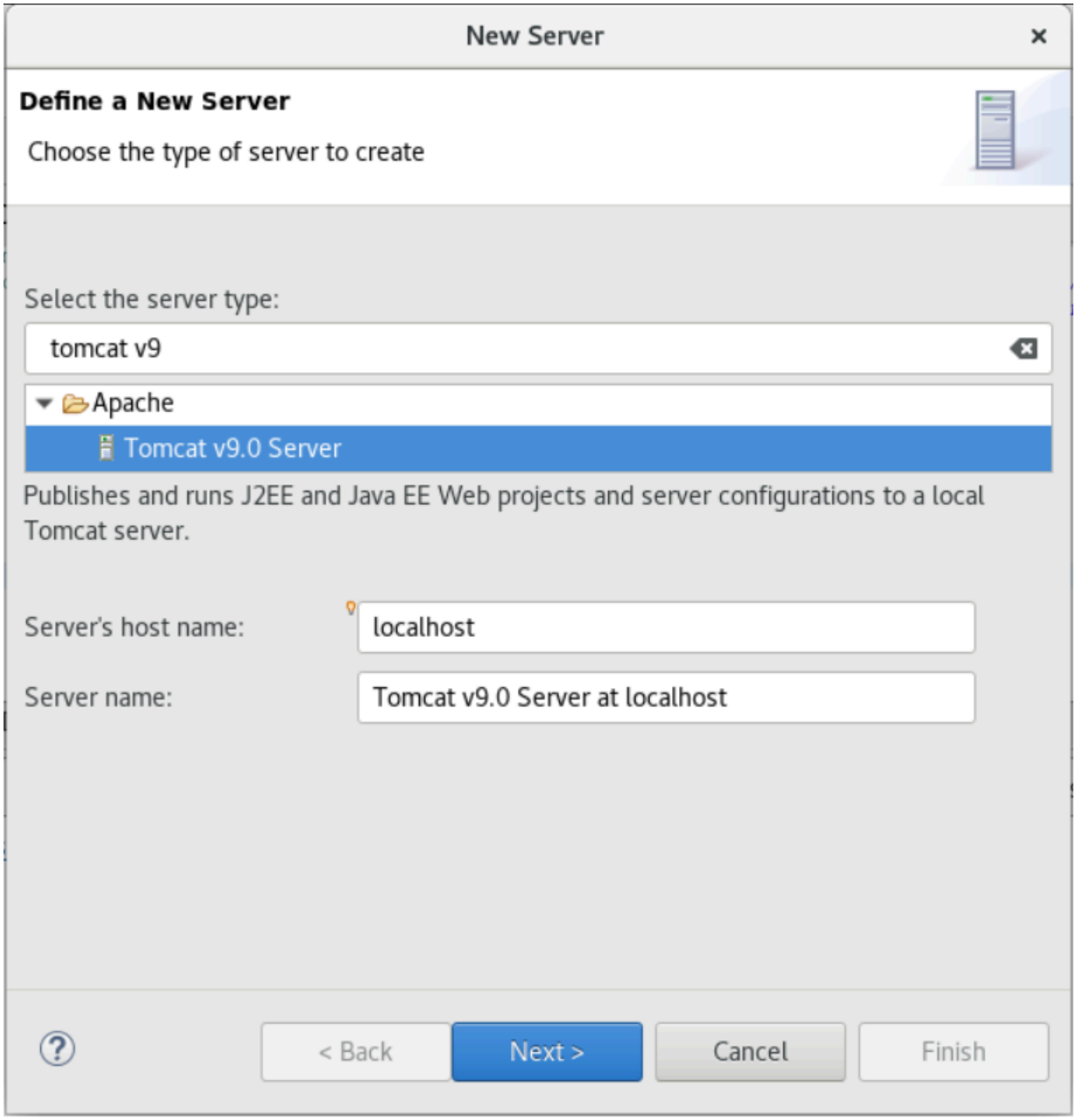
ステップ 6: Tomcat サーバーを設定する

このステップでは、コンパイルしたアプリケーションをデプロイして起動する Tomcat サーバーを設定します。

1. Eclipse では、[ウィンドウ] > [ビューを表示] > [サーバー] を選択して [サーバー] ビューを表示します。

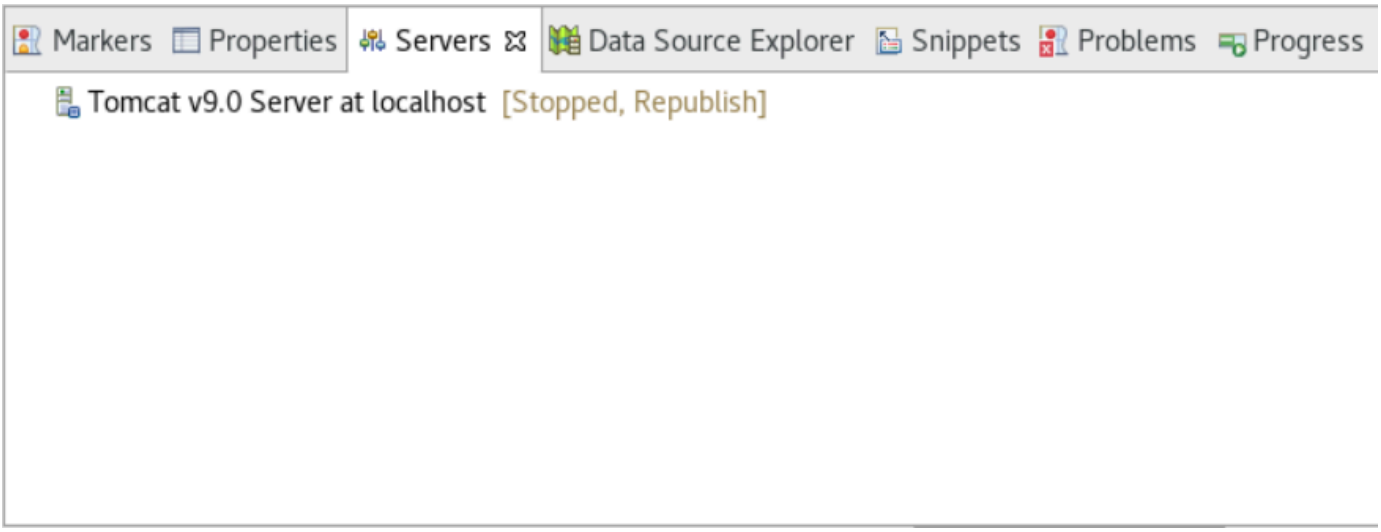


2. [利用可能なサーバーはありません] を選択します。このリンクをクリックして新しいサーバーを作成します...。新しいサーバー ウィザードが表示されます。ウィザードの [サーバーの種類の選択] フィールドに「tomcat v9」と入力し、[Tomcat v9.0 サーバー] を選択します。次いで、[次へ] を選択します。



3. [閲覧] を選択し、ホームフォルダのルートにある [tomcat] フォルダを選択します。JRE はデフォルト値のままにして、[完了] を選択します。

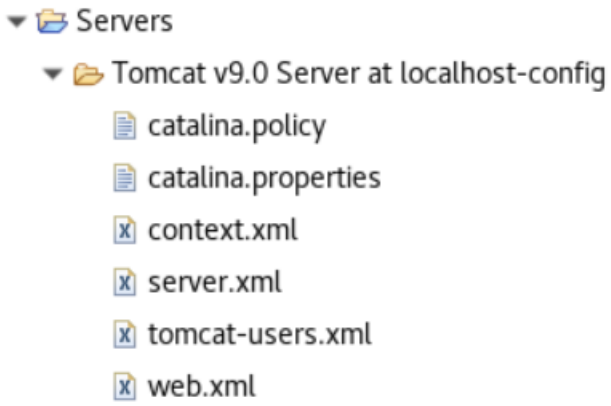
   Servers プロジェクトがワークスペースに作成され、Tomcat v9.0 サーバーが [サーバー] ビューに表示されるようになりました。コンパイルされたアプリケーションはここでデプロイされ、起動されます。



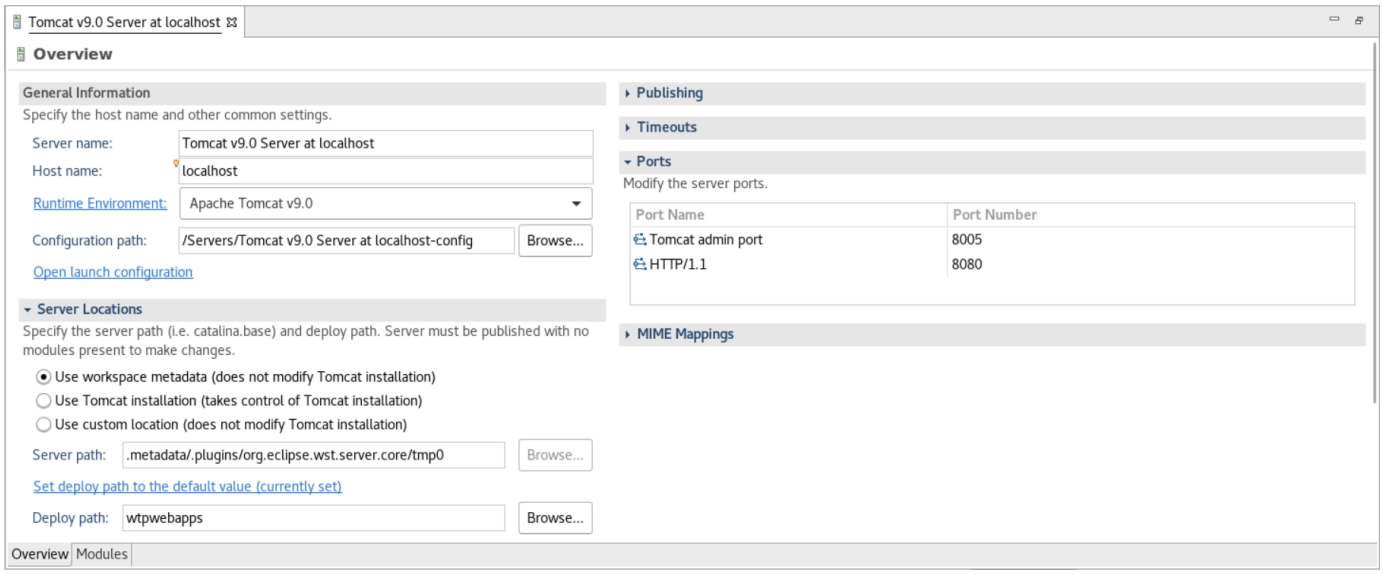
ステップ 7: Tomcat にデプロイする

このステップでは、Planets デモアプリケーションを Tomcat サーバーにデプロイして、アプリケーションを実行できるようにします。

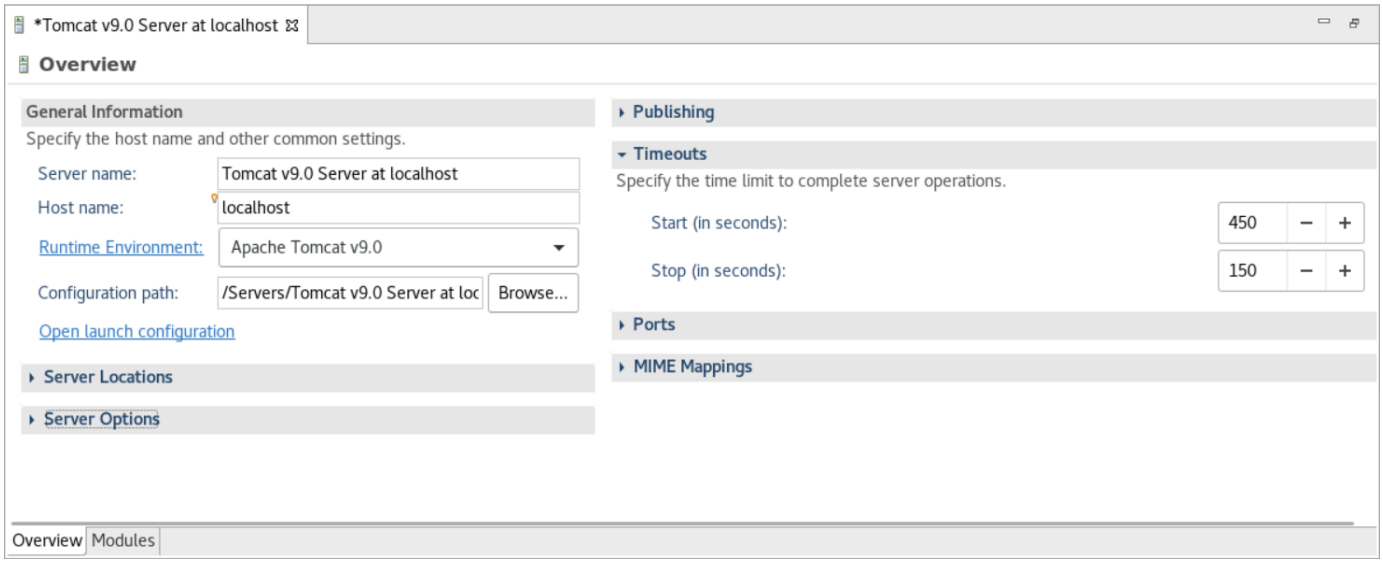
1. `PlanetsDemo-web` ファイルを選択し、[として実行] > [Maven インストール] を選択します。`PlanetsDemo-web` をもう一度選択して [更新] を選択し、npm でコンパイルされたフロントエンドが .war に正しくコンパイルされ、Eclipse に認識されていることを確認します。
2. PlanetsDemo-runtime.zip をインスタンスにアップロードし、アクセス可能な場所でファイルを解凍します。これにより、デモアプリケーションは必要な設定フォルダとファイルにアクセスできるようになります。
3. `PlanetsDemo-runtime/tomcat-config` の内容を Tomcat サーバー用に作成した `Servers/Tomcat v9.0...` サブフォルダにコピーします。



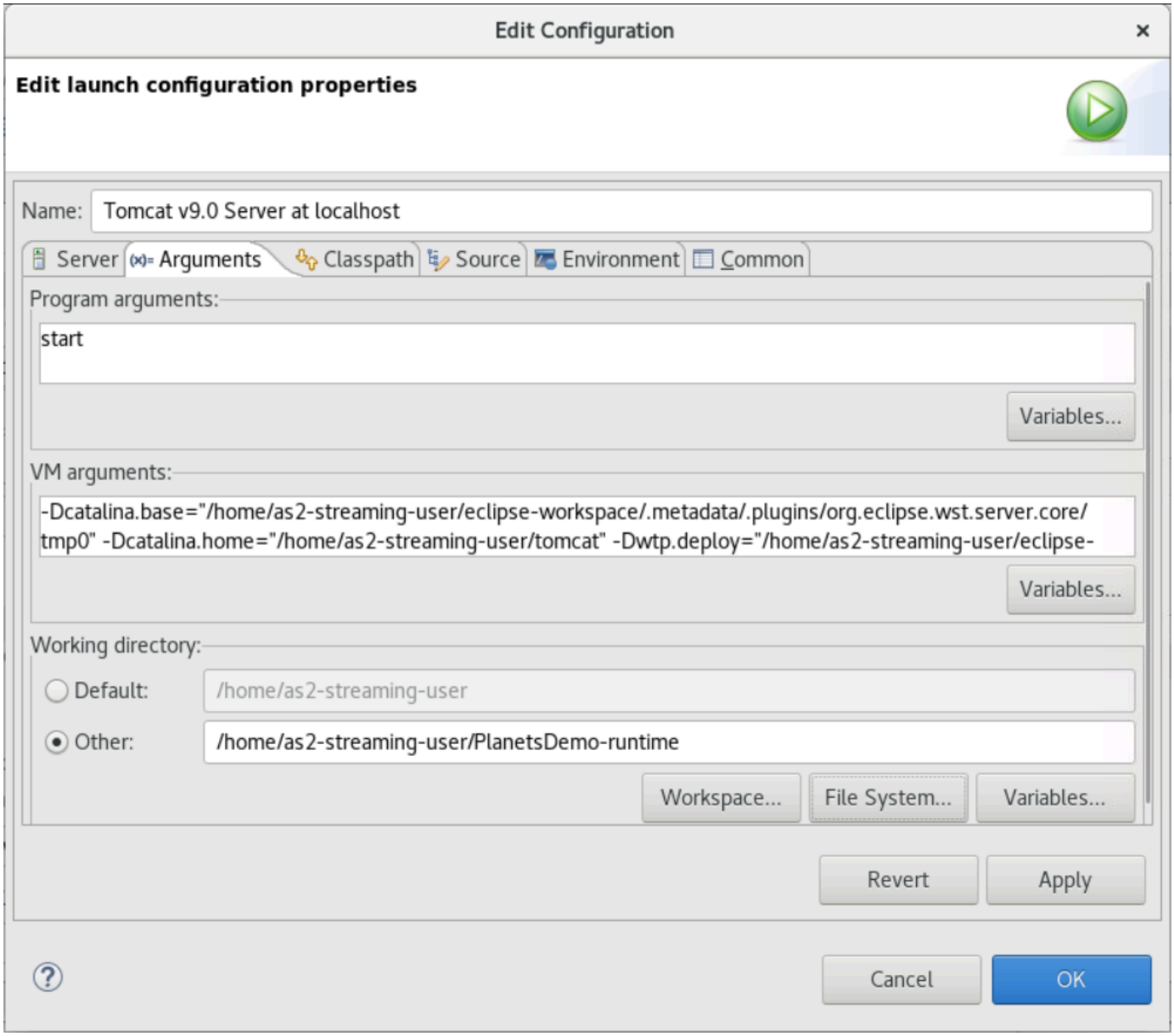
4. [サーバー] ビューで tomcat v9.0 サーバーエントリを開きます。サーバープロパティエディタが表示されます。



5. 次に示すように、[概要] タブで、[開始] の [タイムアウト] の値を 450 秒、[停止] を 150 秒に増やします。



6. [起動設定を開く] を選択します。ウィザードが表示されます。ウィザードで [引数] フォルダに移動し、[作業ディレクトリ] に [その他] を選択します。[ファイルシステム] を選択し、先ほど解凍した PlanetsDemo-runtime フォルダに移動します。このフォルダには、config という直接のサブフォルダが含まれています。



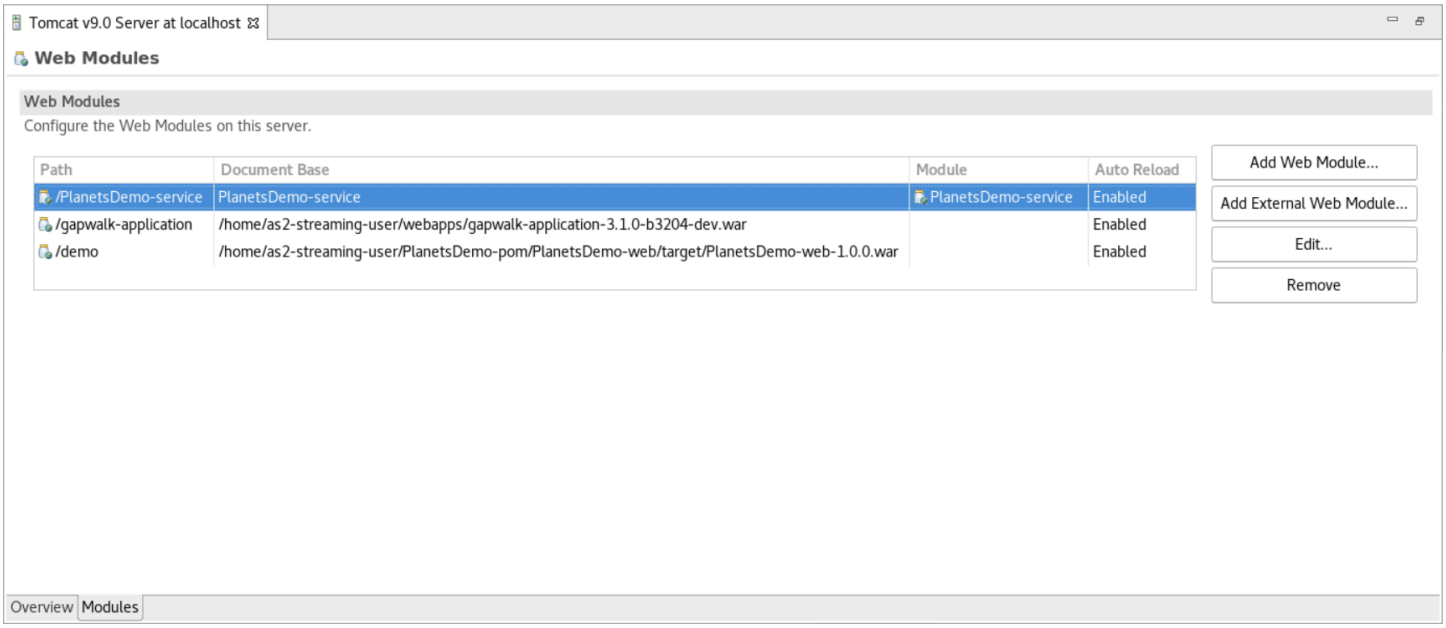
7. サーバープロパティエディタの [モジュール] タブを選択し、以下の変更を行います。

   - [ウェブモジュールを追加] を選択し、`PlanetsDemo-service` を追加します。
   - [外部ウェブモジュールを追加] を選択します。[ウェブモジュールを追加] ダイアログウィンドウが表示されます。以下の変更を加えます。
     - [ドキュメントベース] で、[閲覧] を選択し、`~/webapps/gapwalk-application...war` に移動します。
     - [パス] に、`/gapwalk-application` を入力します。
   - [OK] をクリックします。
   - もう一度 [外部ウェブモジュールを追加] を選択し、次の変更を行います。

- [ドキュメントベース] には、フロントエンドの (`PlanetsDemo-web/target` にある) .war へのパスを入力します。
- [パス] には、`/demo` を入力します。

- [OK] をクリックします。
- エディタの変更を保存します (Ctrl + S)。

これで、エディタの内容は、以下のようになります。



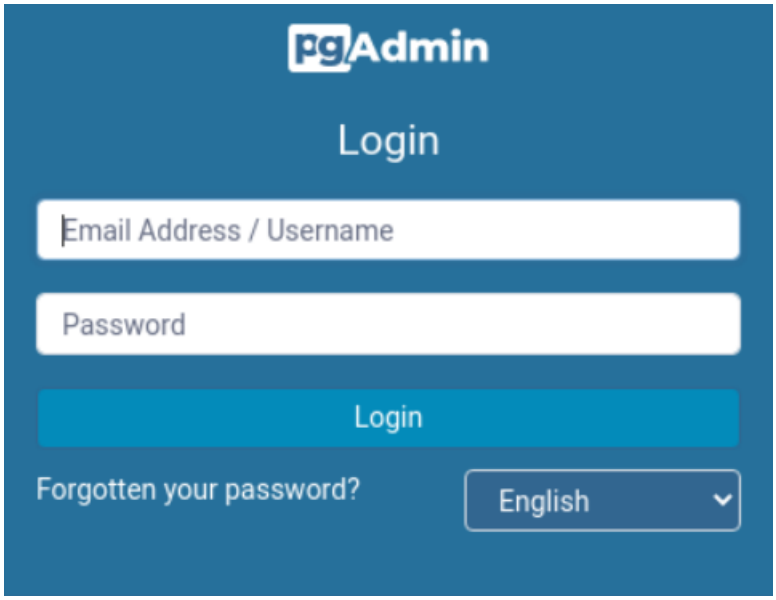
ステップ 8: JICS データベースを作成する

このステップでは、ステップ 1: データベースを作成する で作成したデータベースに接続します。

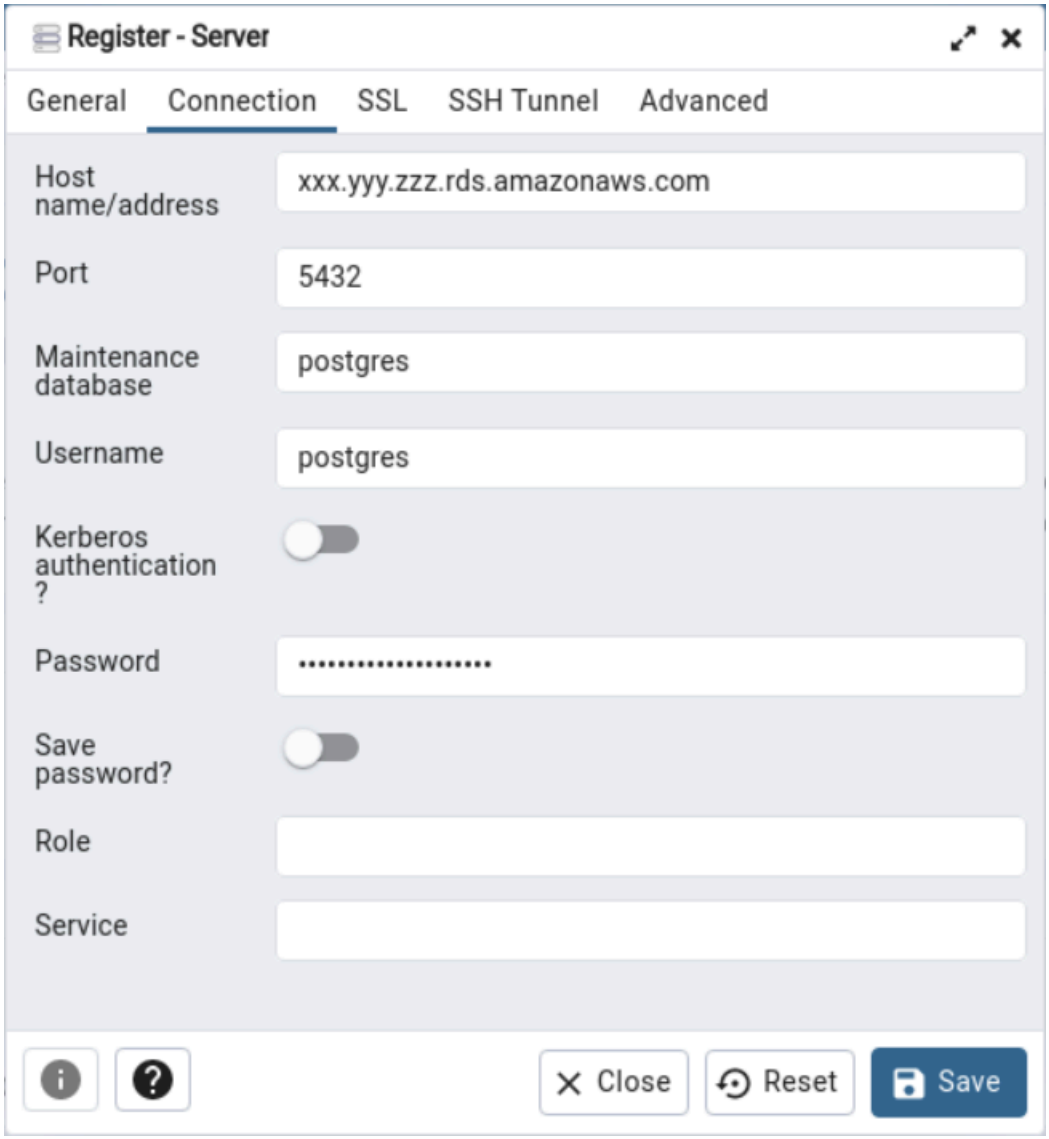
1. AppStream 2.0 インスタンスから、ターミナルで次のコマンドを実行して を起動しますpgAdmin。

```
./pgadmin-start.sh
```

2. ログイン識別子として email アドレスとパスワードを選択します。提供された URL (通常は http://127.0.0.1:5050) をメモしておきます。インスタンスで Google Chrome を起動し、URL をコピーしてブラウザに貼り付け、ユーザーの ID を使用してログインします。



3. ログインしたら、[新しいサーバーの追加] を選択し、以前に作成したデータベースへの接続情報を次のように入力します。

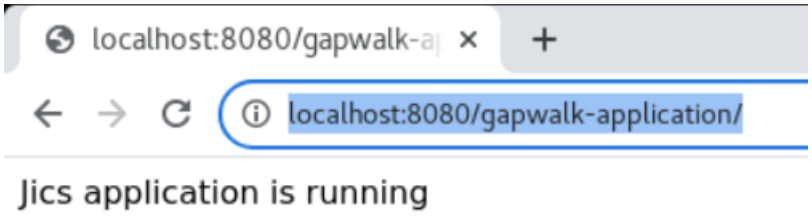


4. データベースサーバーに接続したら、[オブジェクト] > [作成] > [データベース] を使用して、jics という名前の新しいデータベースを作成します。
5. デモアプリが使用したデータベース接続情報を編集します。この情報は `PlanetsDemo-runtime/config/application-main.yml` で定義されています。`jicsDs` エントリを検索します。`username` および `password` の値を取得するには、Amazon RDS コンソールでデータベースに移動します。[設定] タブの [マスター認証情報 ARN] で、[Secrets Manager の管理] を選択します。次に、Secrets Manager コンソールのシークレットで、[シークレットの値を取得する] を選択します。

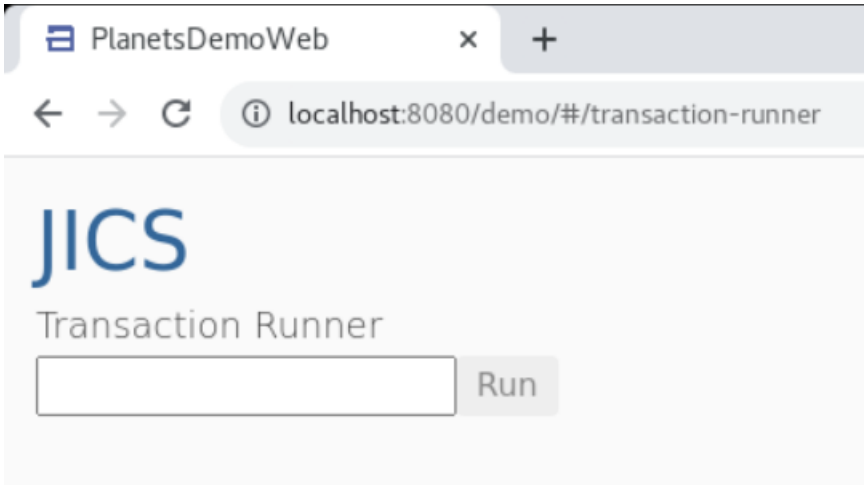
ステップ 9: アプリケーションの起動とテスト

このステップでは、Tomcat サーバーとデモアプリケーションを起動してテストできるようにします。

1. Tomcat サーバーと以前にデプロイしたアプリケーションを起動するには、「サーバー」ビューでサーバーエントリを選択し、[開始] を選択します。起動ログを表示するコンソールが表示されます。
2. [サーバー] ビューでサーバーの状態を確認するか、コンソールに「[xxx] ミリ秒でサーバーが起動します」というメッセージが表示されるまで待ちます。サーバーが起動したら、gapwalk-application が正しくデプロイされていることを確認します。そのためには、Google Chrome ブラウザで http://localhost:8080/gapwalk-application URL にアクセスします。次のように表示されます。



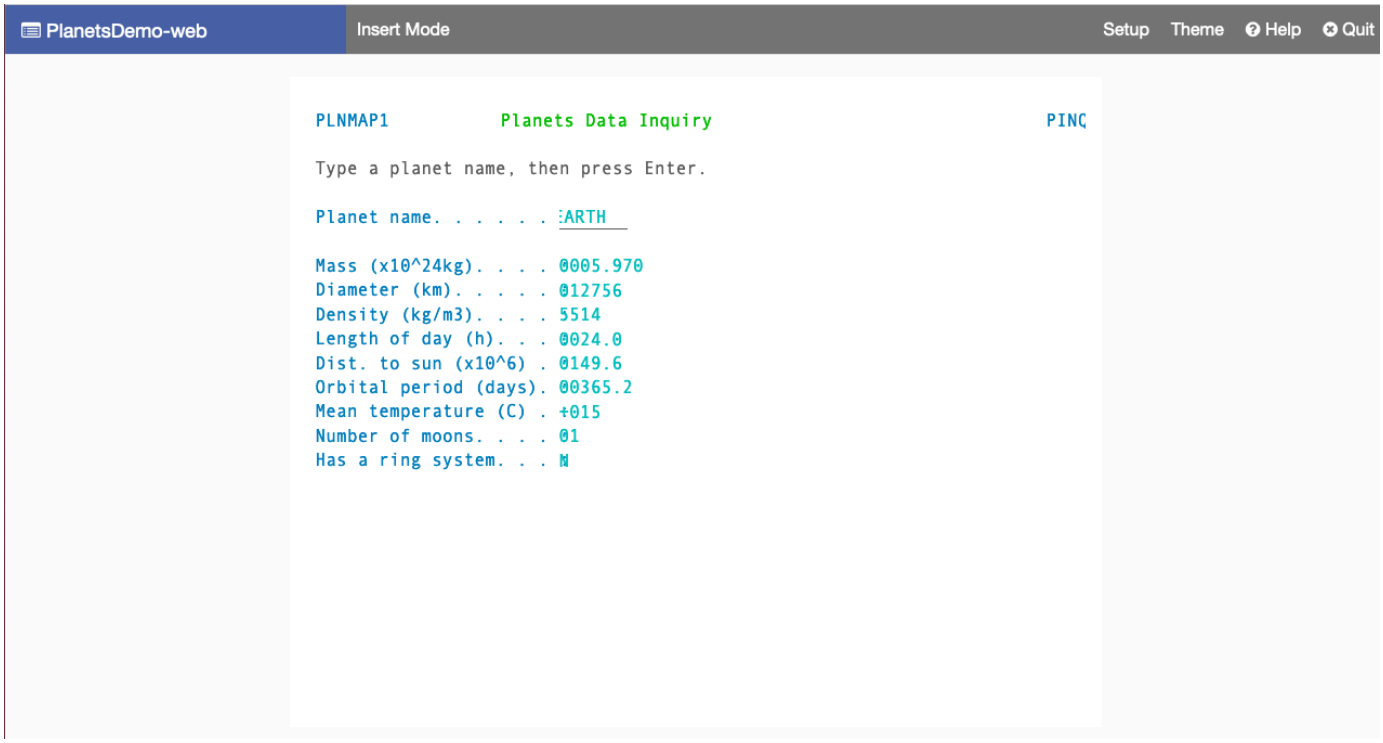
3. デプロイされたアプリケーションのフロントエンドに Google Chrome の http://localhost:8080/demo からアクセスします。次の [トランザクションランチャー] ページが表示されます。



4. アプリケーショントランザクションを開始するには、入力フィールドに `PINQ` を入力し、[実行] を選択します (または Enter キーを押します)。

   デモアプリ画面が表示されます。

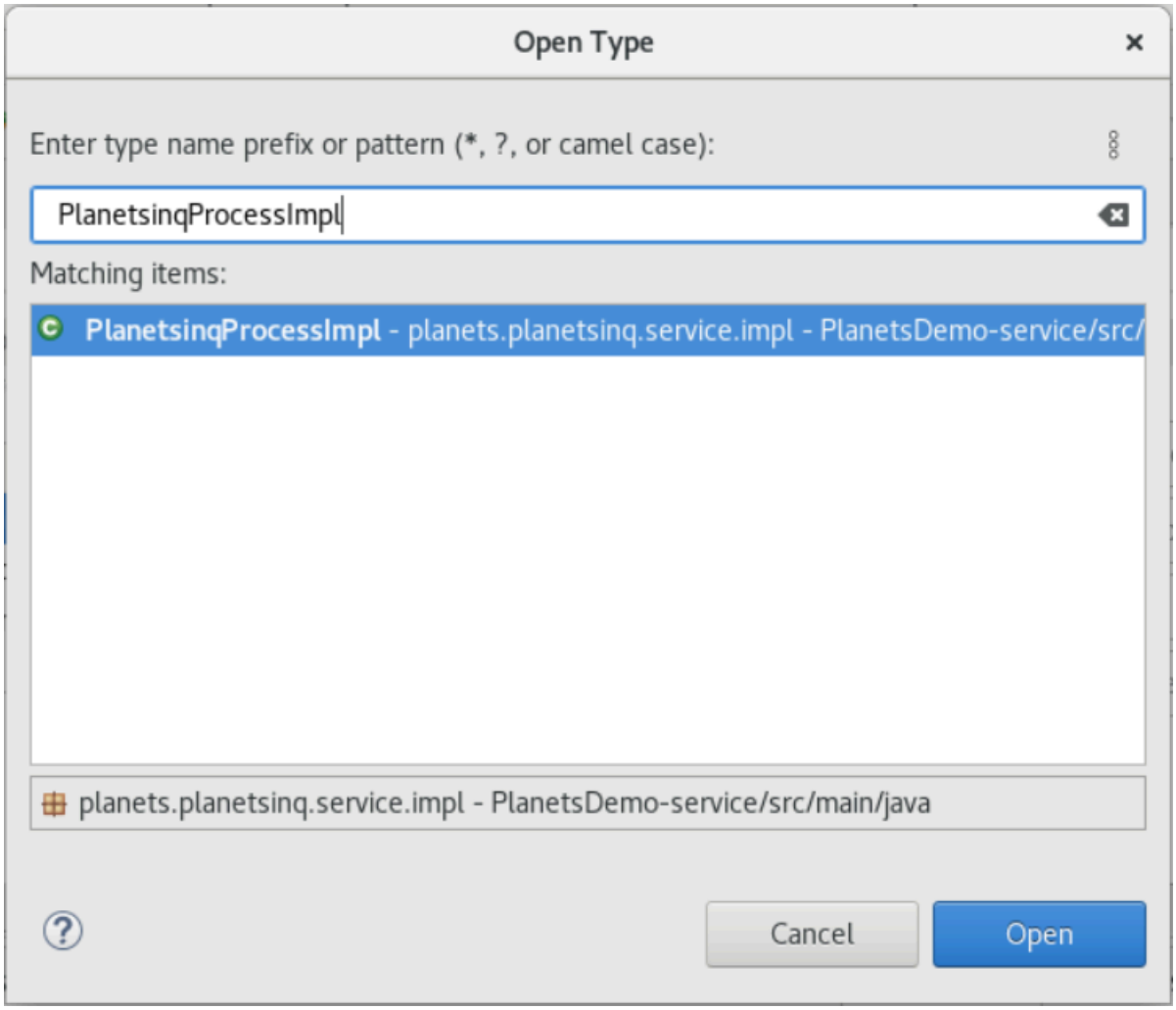
5. 対応するフィールドに惑星名を入力し、Enter キーを押します。



ステップ 10: アプリケーションをデバッグする

このステップでは、Eclipse の標準デバッグ機能を使用してテストします。これらの機能は、最新のアプリケーションを開発しているときに利用できます。

1. メインサービスクラスを開くには、Ctrl + Shift + T キーを押して `PlanetsinqProcessImpl` を入力します。



2. `searchPlanet` メソッドに移動し、そこにブレークポイントを設定します。
3. サーバー名を選択し、[デバッグで再起動] を選択します。
4. 前の手順を繰り返します。つまり、アプリケーションにアクセスし、惑星名を入力して Enter キーを押します。

   Eclipse は `searchPlanet` メソッドのアプリケーションを停止します。これで調べることができます。

リソースをクリーンアップする

このチュートリアル用に作成したリソースが不要になった場合は、追加料金の発生を避けるため、削除してください。以下のステップを実行します。

- Planets アプリケーションがまだ実行中の場合は、終了させます。
- ステップ 1: データベースを作成する で作成したデータベースを削除します。詳細については、「DB インスタンスの削除」を参照してください。

Micro Focus によるアプリケーションのリプラットフォーム

このセクションでは、リプラットフォームプロセスの各ステップについて説明します。すべてのタスクについて説明し、Amazon EC2 での AWS Mainframe Modernization ランタイムの設定と運用に関する情報が含まれています。

トピック

Micro Focus ランタイム (Amazon EC2) の設定

AWS Mainframe Modernization には、Micro Focus ライセンス製品を含むいくつかの Amazon マシンイメージ (AMIs) が用意されています。これらの AMI を使用すると、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスをすばやくプロビジョニングして、お客様が制御および管理する Micro Focus 環境をサポートできるようになります。このトピックでは、これらの AMI にアクセスして起動するために必要な手順について説明します。これらの AMI の使用は完全に任意であり、このユーザーガイドのチュートリアルを完了するために必須ではありません。

トピック

- AWS Identity and Access Management ロールの作成
- License Manager への必要なアクセス許可の付与
- Amazon マシンイメージのサブスクライブ
- AWS Mainframe Modernization Micro Focus インスタンスを起動する
- インターネットにアクセスできないサブネットまたは VPC
- ライセンスの問題のトラブルシューティング

前提条件

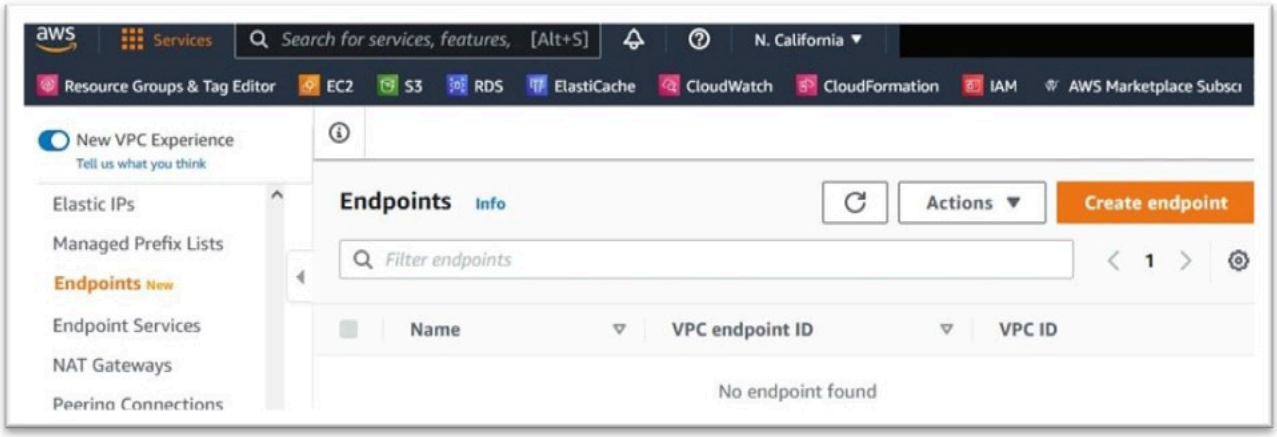
次の前提条件を満たしていることを確認します。

- Amazon EC2 インスタンスを作成するアカウントへの管理者アクセス。
- AWS リージョン Amazon EC2 インスタンスが作成される を特定し、 AWS Mainframe Modernization サービスが利用可能であることを確認します。「AWS サービス (リージョン別)」を参照してください。サービスを利用できるリージョンを必ず選択してください。
- Amazon EC2 インスタンスを作成する Amazon Virtual Private Cloud (Amazon VPC) を特定します。

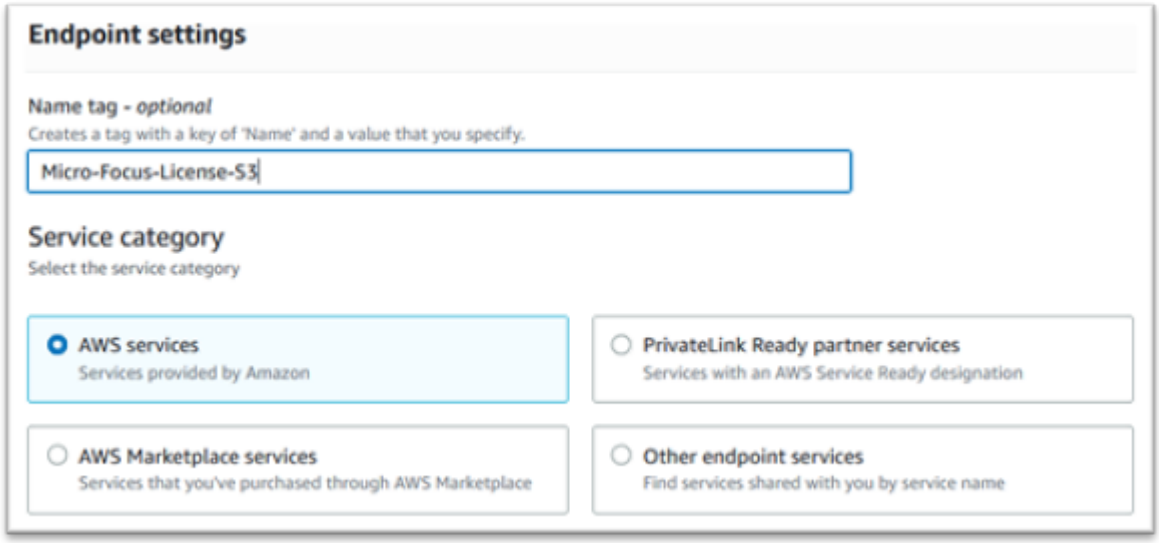
Amazon S3 用の Amazon VPC エンドポイントの作成

このセクションでは、Amazon S3 で使用する Amazon VPC エンドポイントを作成します。

1. AWS Management Consoleの Amazon VPC に移動します。
2. ナビゲーションペインで、[エンドポイント] を選択します。
3. [エンドポイントの作成] を選択します。



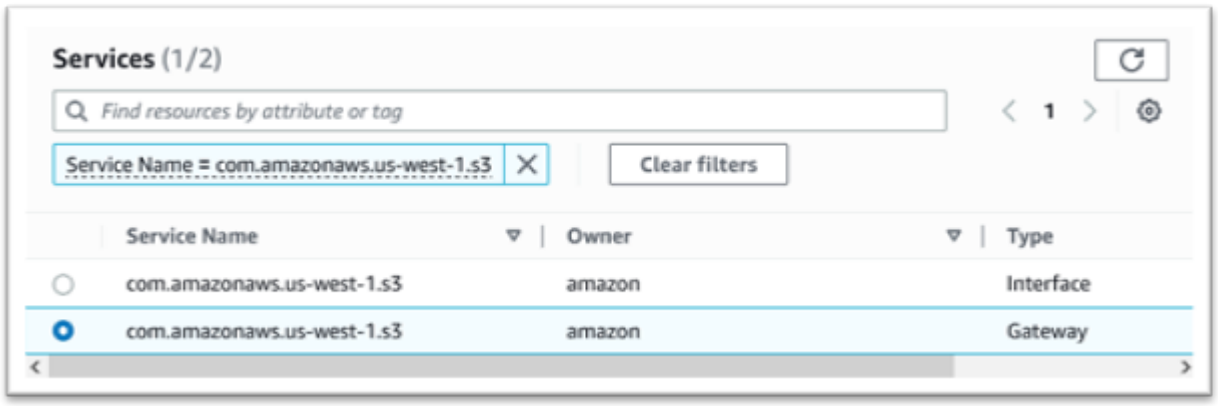
4. わかりやすい名前タグを入力します (例:「Micro-Focus-License-S3」)。
5. サービスカテゴリで、[AWS サービス] を選択します。



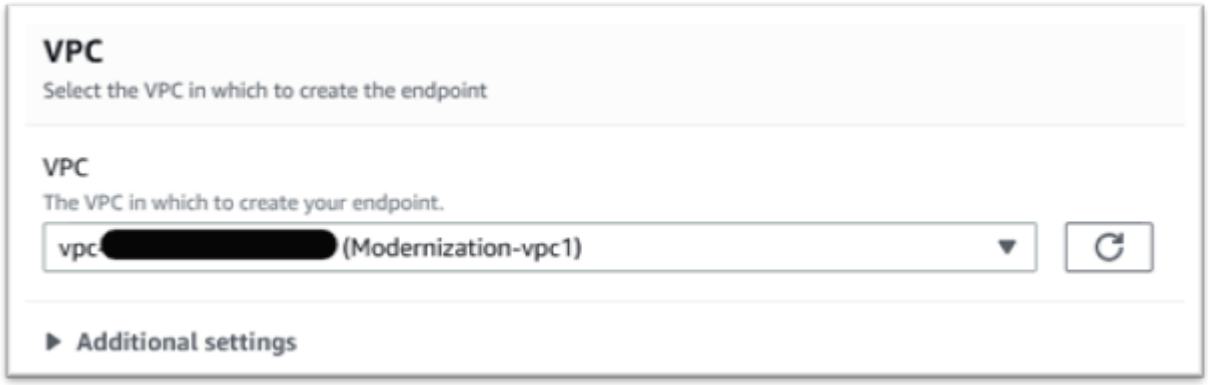
6. [サービス] で Amazon S3 ゲートウェイサービス com.amazonaws.[region].s3 を検索します。

   us-west-1 の場合は、com.amazonaws.us-west-1.s3 となります。

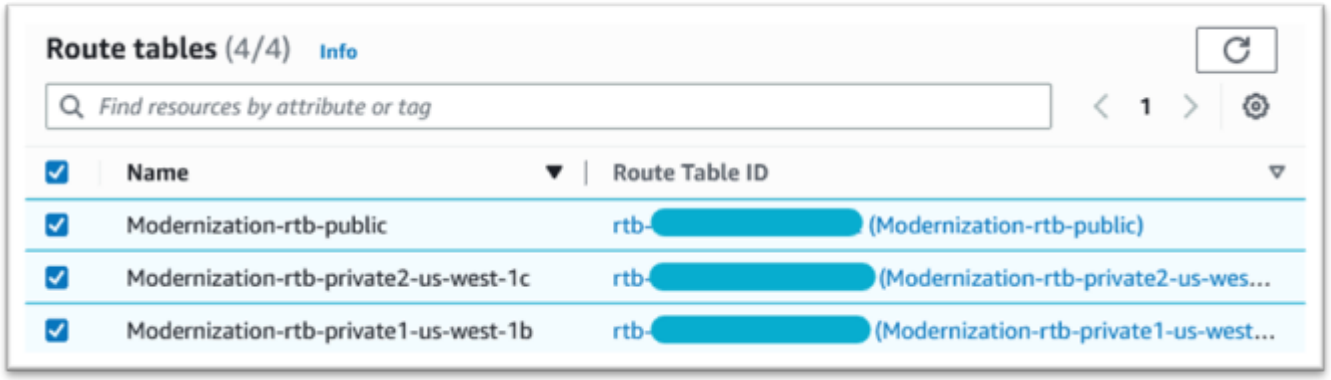
7. [ゲートウェイ] サービスを選択してください。



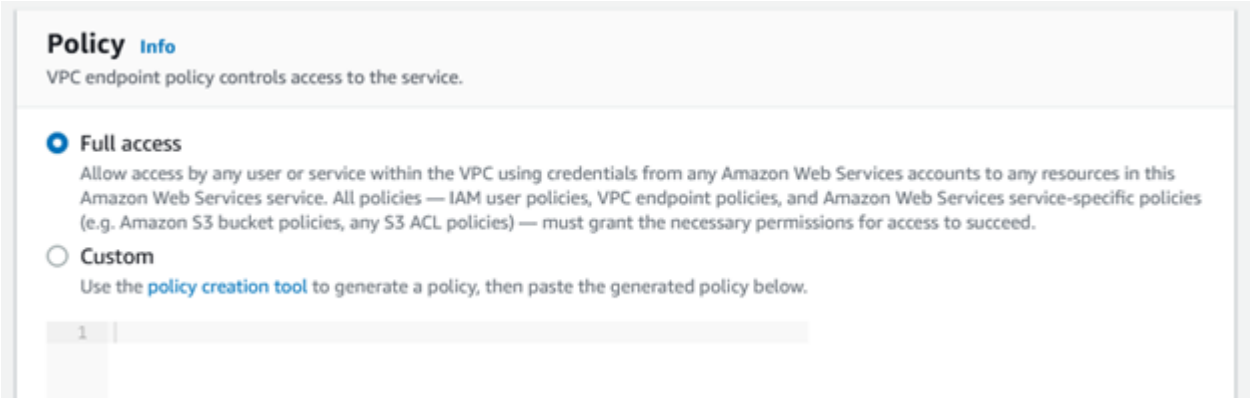
8. VPC の場合は、使用する VPC を選択します。



9. VPC のすべてのルートテーブルを選択します。



10. [ポリシー] で [フルアクセス] を選択します。



11. [エンドポイントの作成] を選択します。

アカウントの許可リスト更新のリクエスト

AWS 担当者と協力して、 Mainframe Modernization AWS AMIs のアカウント許可リストに登録してください。以下の情報を指定します。

- AWS アカウント ID。

- Amazon VPC エンドポイント AWS リージョン が作成された 。
- 「Amazon S3 用の Amazon VPC エンドポイントの作成」で作成した Amazon VPC Amazon S3 エンドポイント ID。これは com.amazonaws.[region].s3 Gateway エンドポイントの `vpce-xxxxxxxxxxxxxxxxx` ID です。
- すべての Micro Focus Enterprise Suite AMI Amazon EC2 インスタンスに必要なライセンス数。

  CPU コアごと (ほとんどの Amazon EC2 インスタンスでは 2 つの vCPU ごと) に 1 つのライセンスが必要です。

  詳細については、「CPU オプションの最適化」を参照してください。

  リクエストされた番号は、後で によって調整できます AWS。

> **Note**
>
> AWS 担当者は、許可リストリクエストのサポートチケットを開く必要があります。直接リクエストすることはできず、リクエストが完了するまでに数日かかる場合があります。

AWS Identity and Access Management ロールの作成

AWS Mainframe Modernization Amazon EC2 インスタンスで使用する AWS Identity and Access Management ポリシーとロールを作成します。IAM コンソールでロールを作成すると、同じ名前の関連するインスタンスプロファイルが作成されます。このインスタンスプロファイルを Amazon EC2 インスタンスに割り当てると、Micro Focus ライセンスを割り当てられます。インスタンスプロファイルの詳細については、「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」を参照してしてください。

IAM ポリシーの作成

IAM ポリシーが最初に作成され、次にロールにアタッチされます。

1. AWS Identity and Access Management の に移動します AWS Management Console。
2. [ポリシー] を選択して、[ポリシーの作成] を選択します。

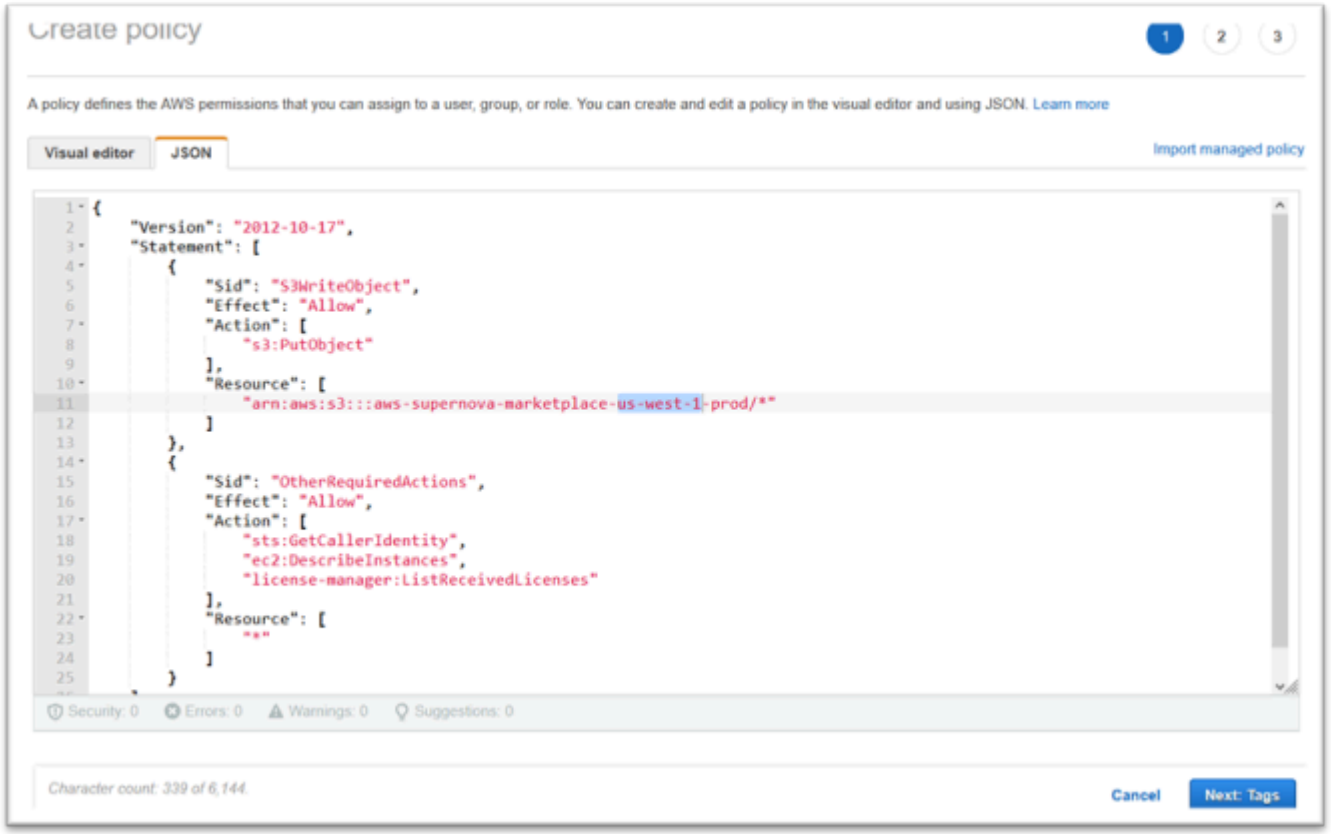
3. [JSON] タブを選択します。

4. 次の JSON us-west-1の を Amazon S3 エンドポイント AWS リージョン が定義された に置き換え、JSON をコピーしてポリシーエディタに貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3WriteObject",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::aws-supernova-marketplace-us-west-1-prod/*"
            ]
        },
        {
            "Sid": "OtherRequiredActions",
```

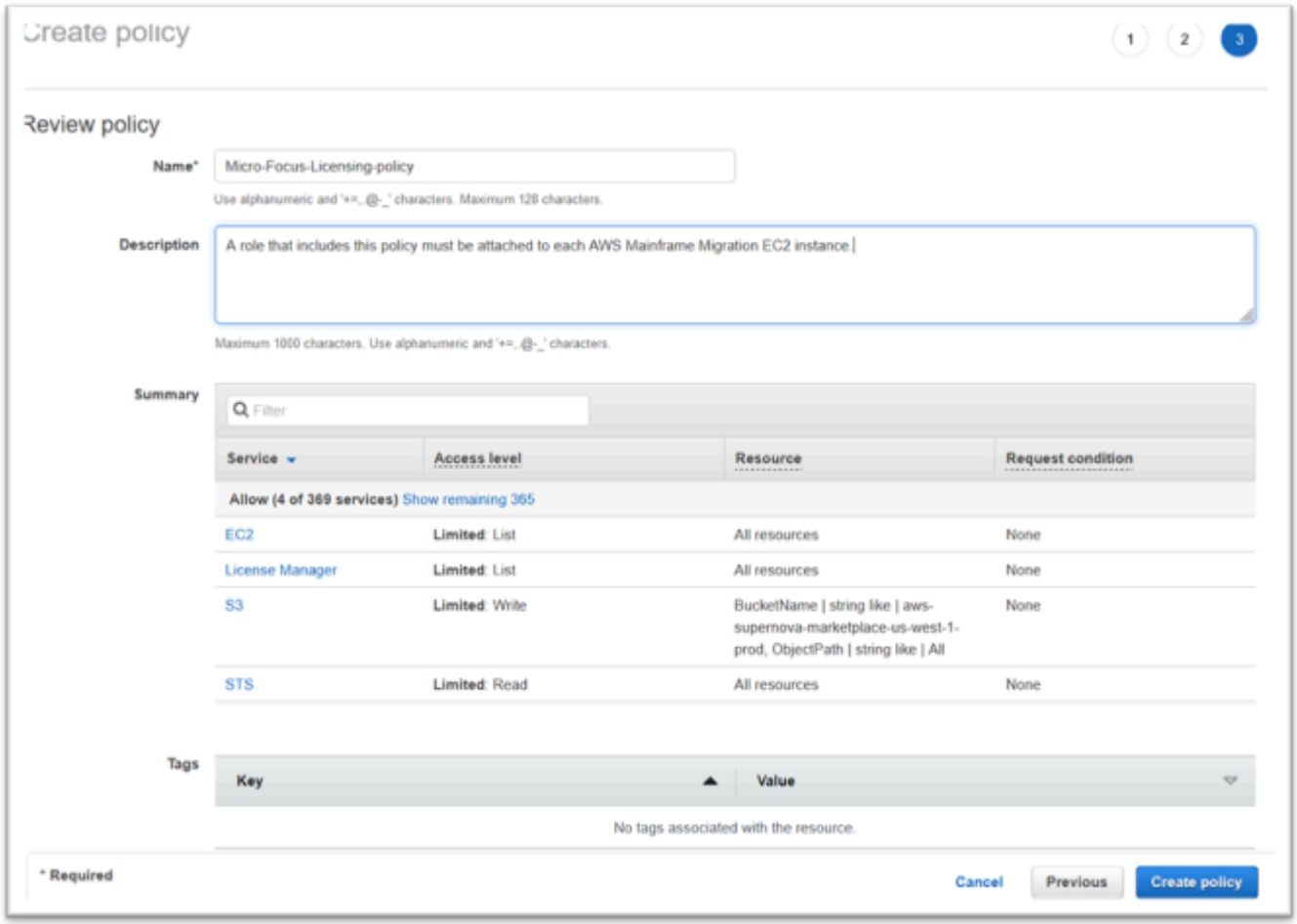
```
            "Effect": "Allow",
            "Action": [
                "sts:GetCallerIdentity",
                "ec2:DescribeInstances",
                "license-manager:ListReceivedLicenses"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

> **Note**
>
> Sid OtherRequiredActions の下のアクションは、リソースレベルのアクセス許可をサポートしないため、リソース要素で * を指定する必要があります。



5. [次へ: タグ] を選択します。

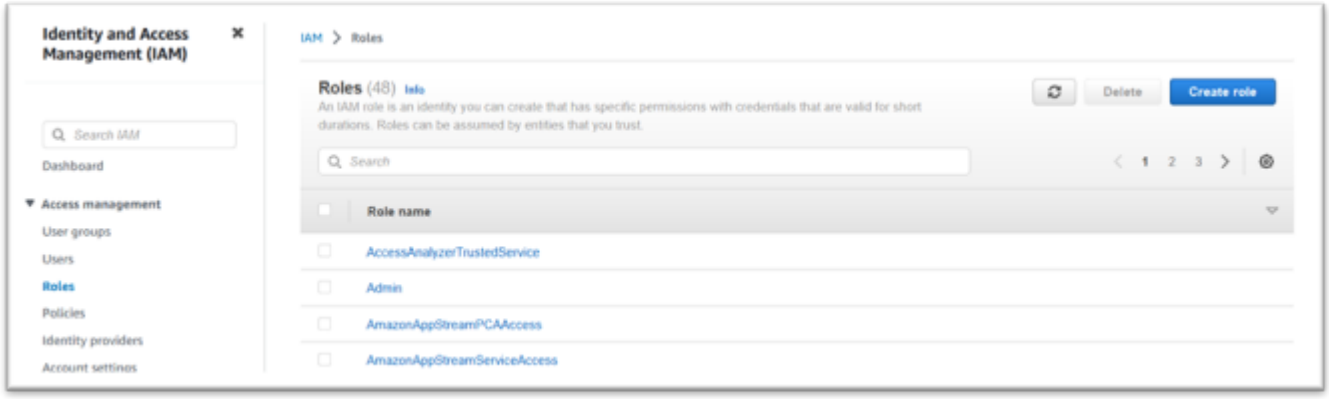
6. 必要に応じてタグを入力し、[次へ: 確認] を選択します。
7. ポリシーの名前を入力します (例: 「Micro-Focus-Licensing-policy」)。オプションで、「このポリシーを含むロールは、各 AWS Mainframe Modernization Amazon EC2 インスタンスにアタッチする必要があります」などの説明を入力します。



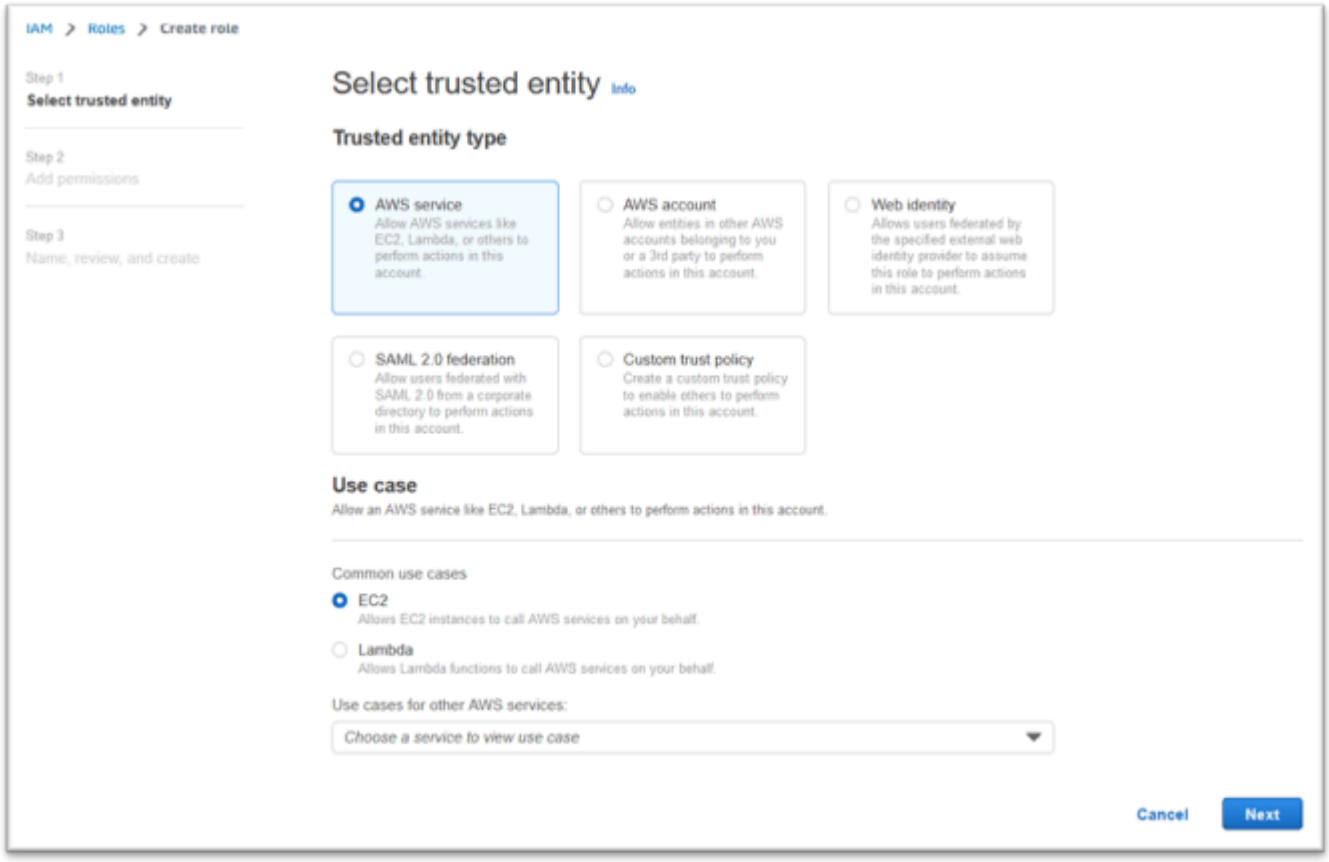
8. [ポリシーの作成] を選択します。

IAM ロールの作成

1. AWS Management Consoleで IAM に移動します。
2. [ロール] を選択してから [ロールの作成] を選びます。

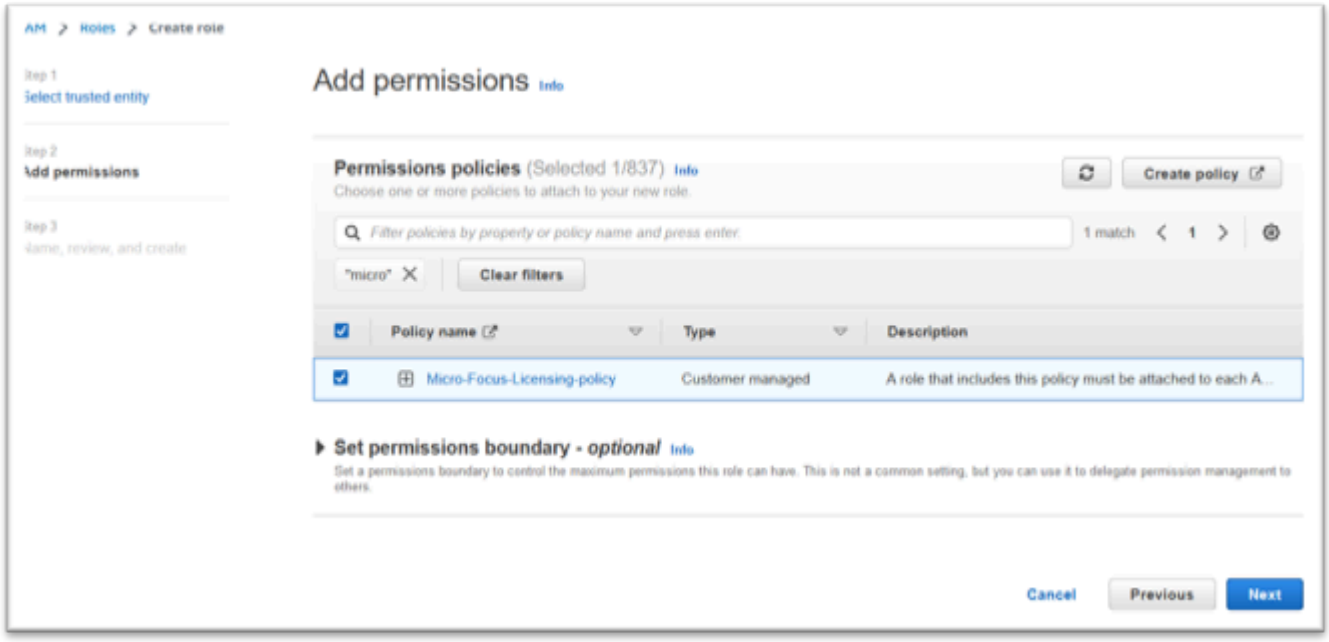


3. [信頼されたエンティティタイプ] は [AWS のサービス] のままにしておき、一般的なユースケースは [EC2] を選択します。

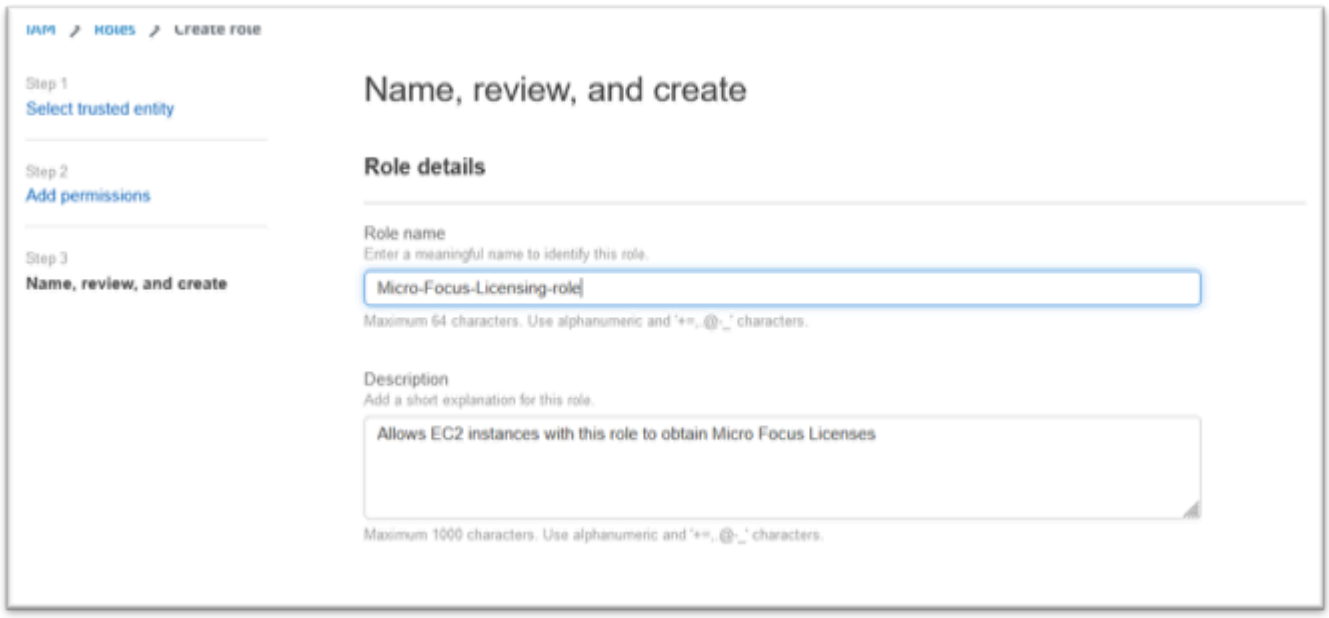


4. [次へ] を選択します。
5. フィルタに「Micro」と入力し、Enter キーを押してフィルタを適用します。

6. 作成したばかりのポリシーを選択します (例: 「Micro-Focus-Licensing-policy」)。
7. [次へ] を選択します。



8. ロール名 (例: 「Micro-Focus-Licensing-role」) を入力します。
9. 説明を独自の説明に置き換えてください (例: 「このロールの Amazon EC2 インスタンスに Micro Focus ライセンスの取得を許可します」)。

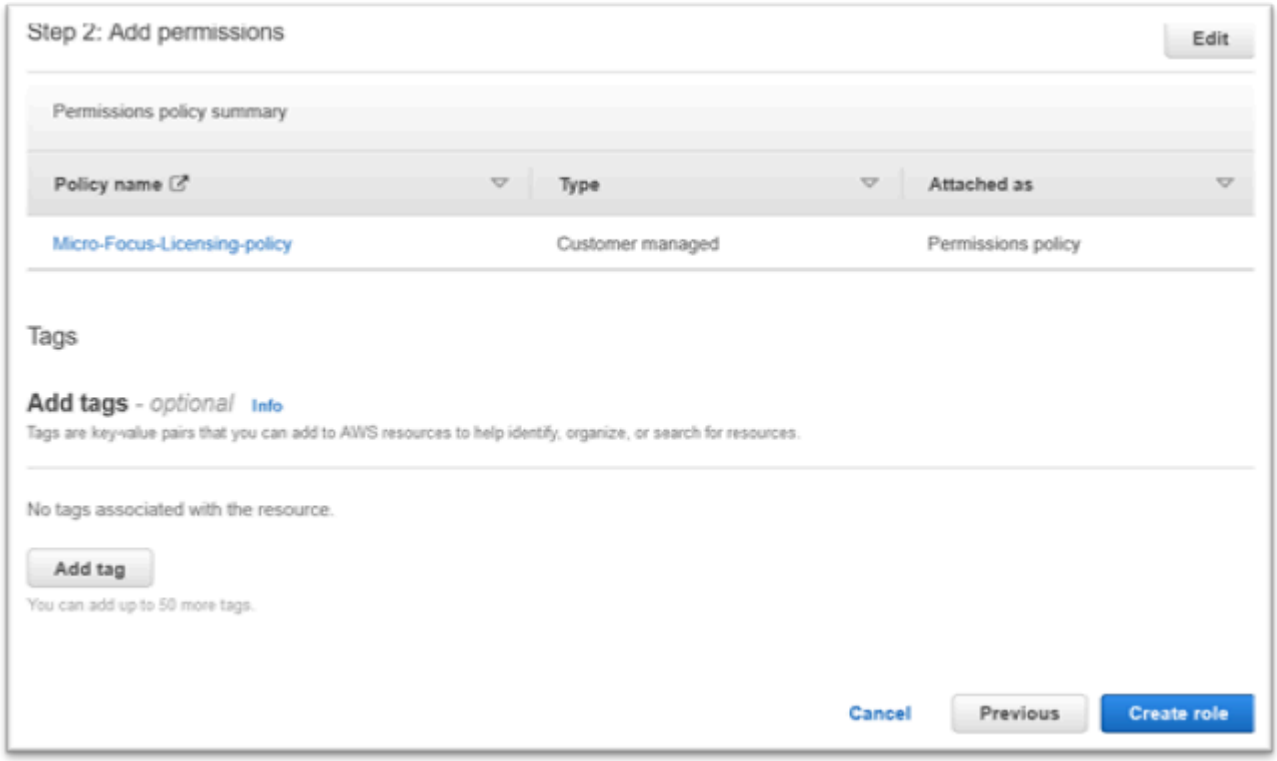


10. [ステップ 1: 信頼されたエンティティを選択] で JSON を確認し、次の値になっていることを確認します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

> **Note**
>
> Effect、Action、Principal の順序は重要ではありません。

11. [ステップ 2: アクセス許可の追加] にライセンスポリシーが表示されていることを確認します。

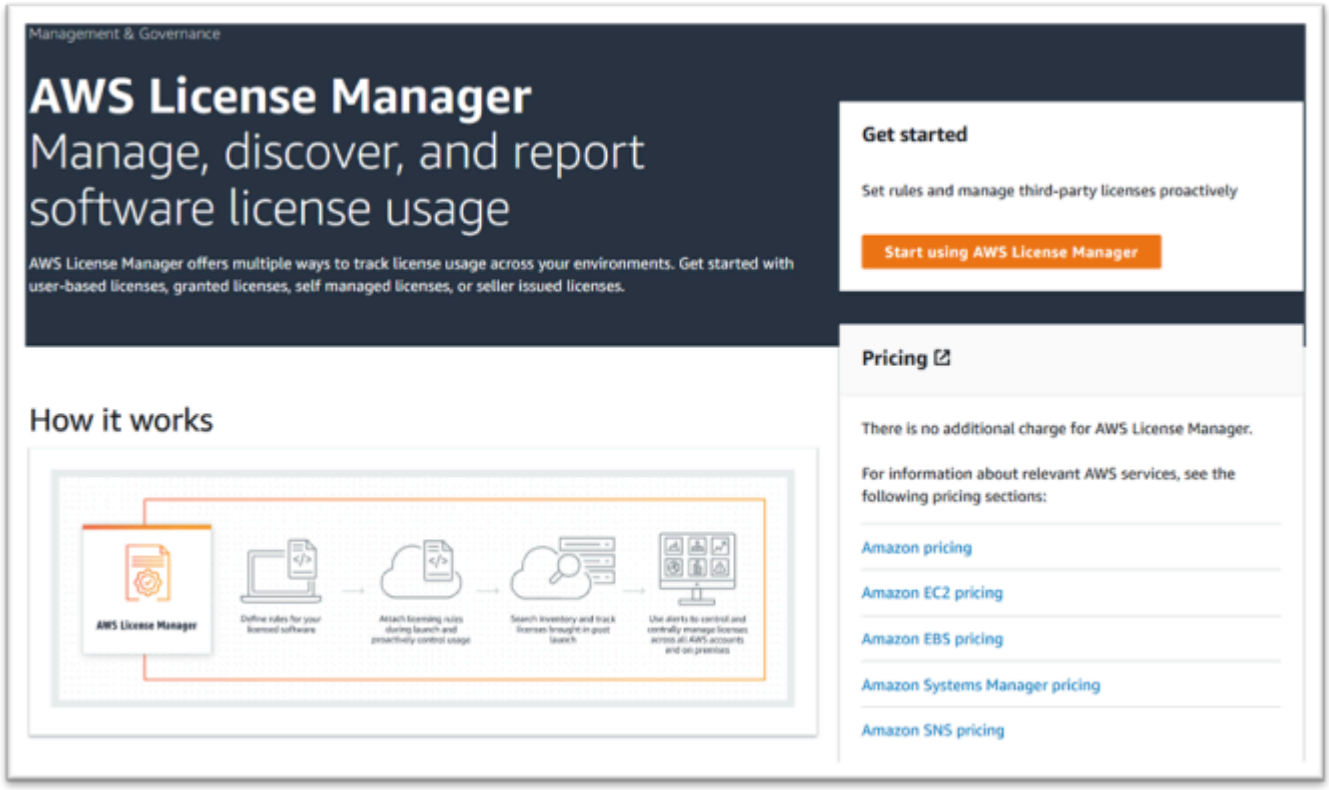


12. [ロールの作成] を選択します。

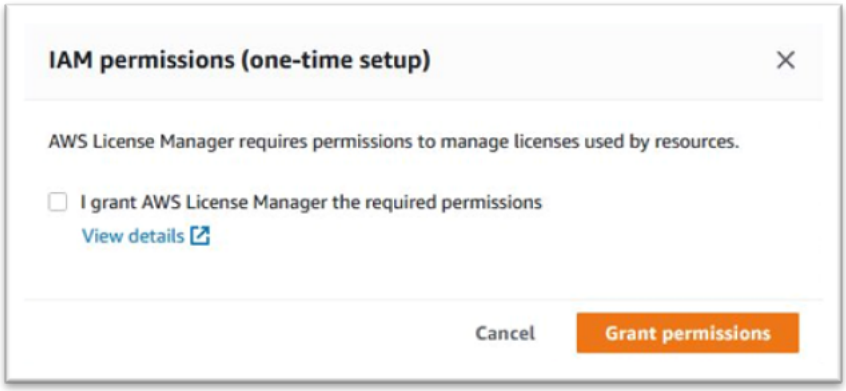
許可リストのリクエストが完了したら、次のステップに進みます。

License Manager への必要なアクセス許可の付与

1. AWS License Manager の に移動します AWS Management Console。



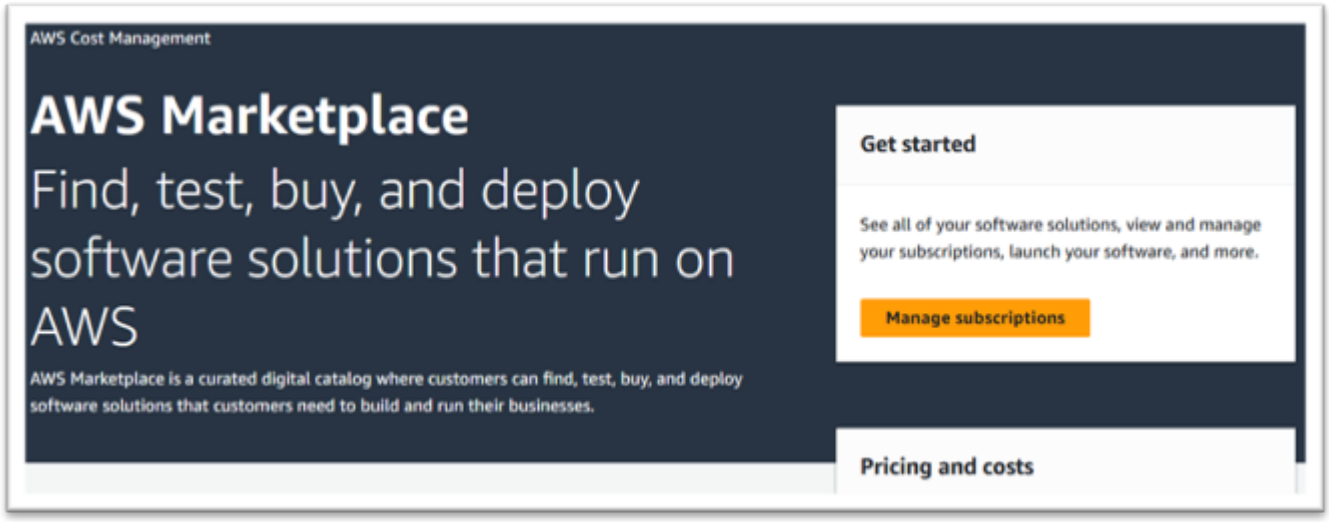
2. [AWS License Manager の使用を開始する] を選択します。
3. 次のポップアップが表示されたら、詳細を確認してチェックボックスを選択し、[アクセス許可を付与] を選択します。



Amazon マシンイメージのサブスクライブ

AWS Marketplace 製品をサブスクライブしたら、製品の AMI からインスタンスを起動できます。

1. の AWS Marketplace サブスクリプションに移動します AWS Management Console。
2. [サブスクリプションを管理する] を選択します。



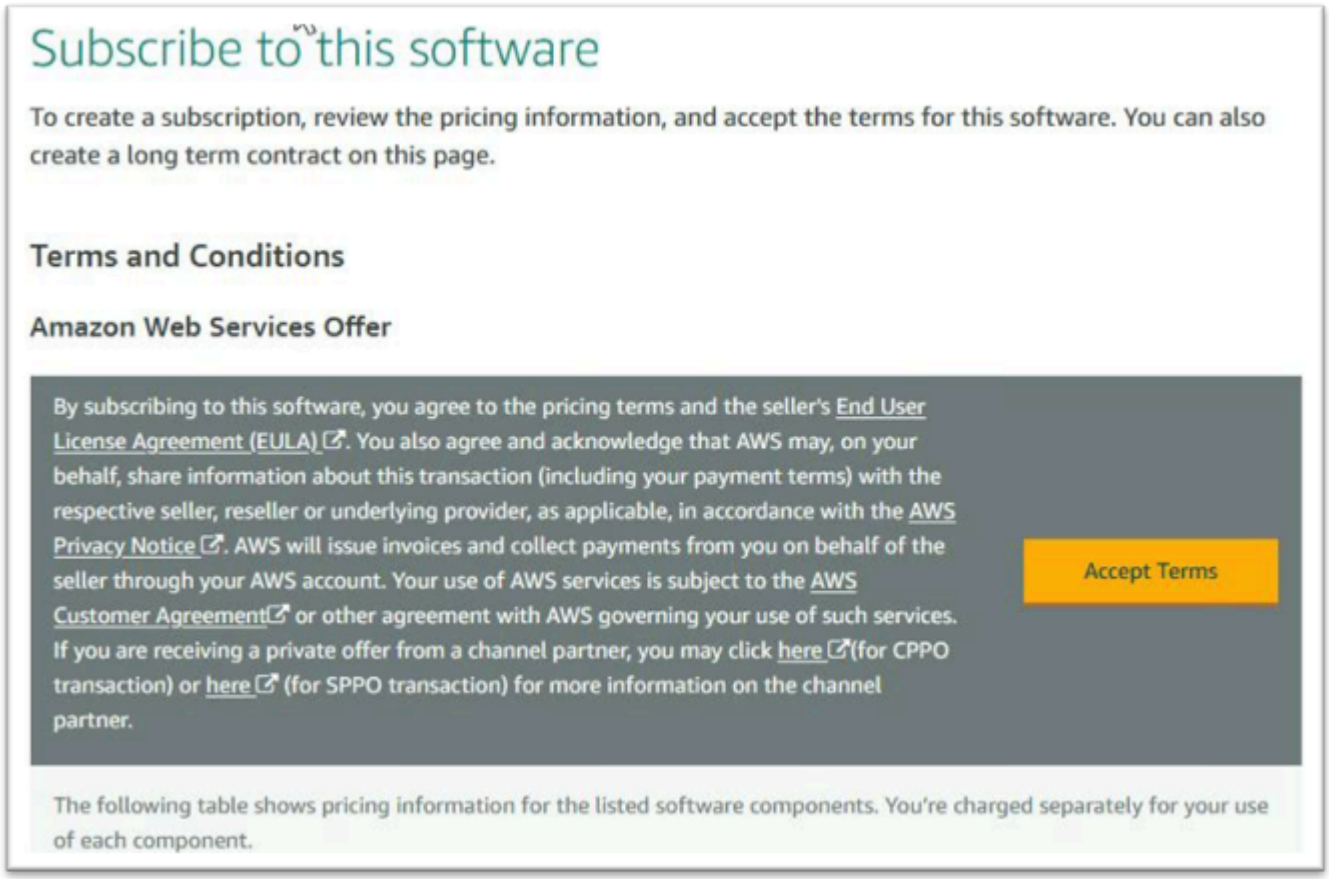
3. 以下のリンクをコピーしてブラウザのアドレスバーに貼り付けます。

> **Note**
>
> 使用が許可されている製品の 1 つへのリンクのみを選択してください。

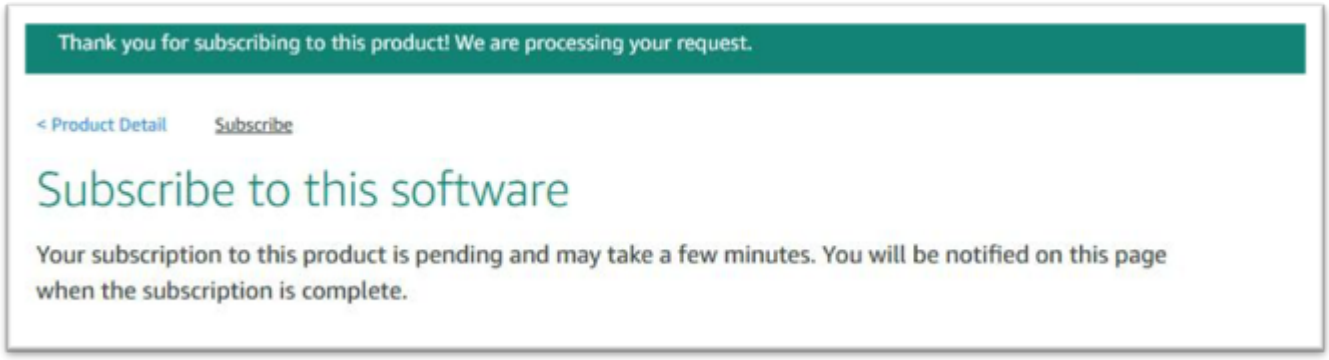
- Enterprise Server: https://aws.amazon.com/marketplace/pp/prodview-g5emev63l7blc
- Enterprise Server (Windows 用): https://aws.amazon.com/marketplace/pp/prodview-lwybsiyikbhc2
- Enterprise Developer: https://aws.amazon.com/marketplace/pp/prodview-77qmpr42yzxwk
- Enterprise Developer (Visual Studio 2022): https://aws.amazon.com/marketplace/pp/prodview-m4l3lqiszo6cm
- Enterprise Analyzer: https://aws.amazon.com/marketplace/pp/prodview-tttheylcmcihm
- Enterprise Build ツール (Windows 用): https://aws.amazon.com/marketplace/pp/prodview-2rw35bbt6uozi
- Enterprise ストアドプロシージャ: https://aws.amazon.com/marketplace/pp/prodview-zoeyqnsdsj6ha
- Enterprise ストアドプロシージャ (SQL Server 2019): https://aws.amazon.com/marketplace/pp/prodview-ynfklquwubnz4

4. [サブスクリプションを続行する] を選択します。

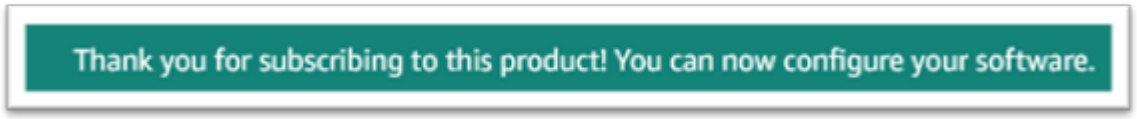
5. 利用規約に同意できる場合は、[規約の受諾] を選択します。



6. このサブスクリプションの処理には数分かかる場合があります。

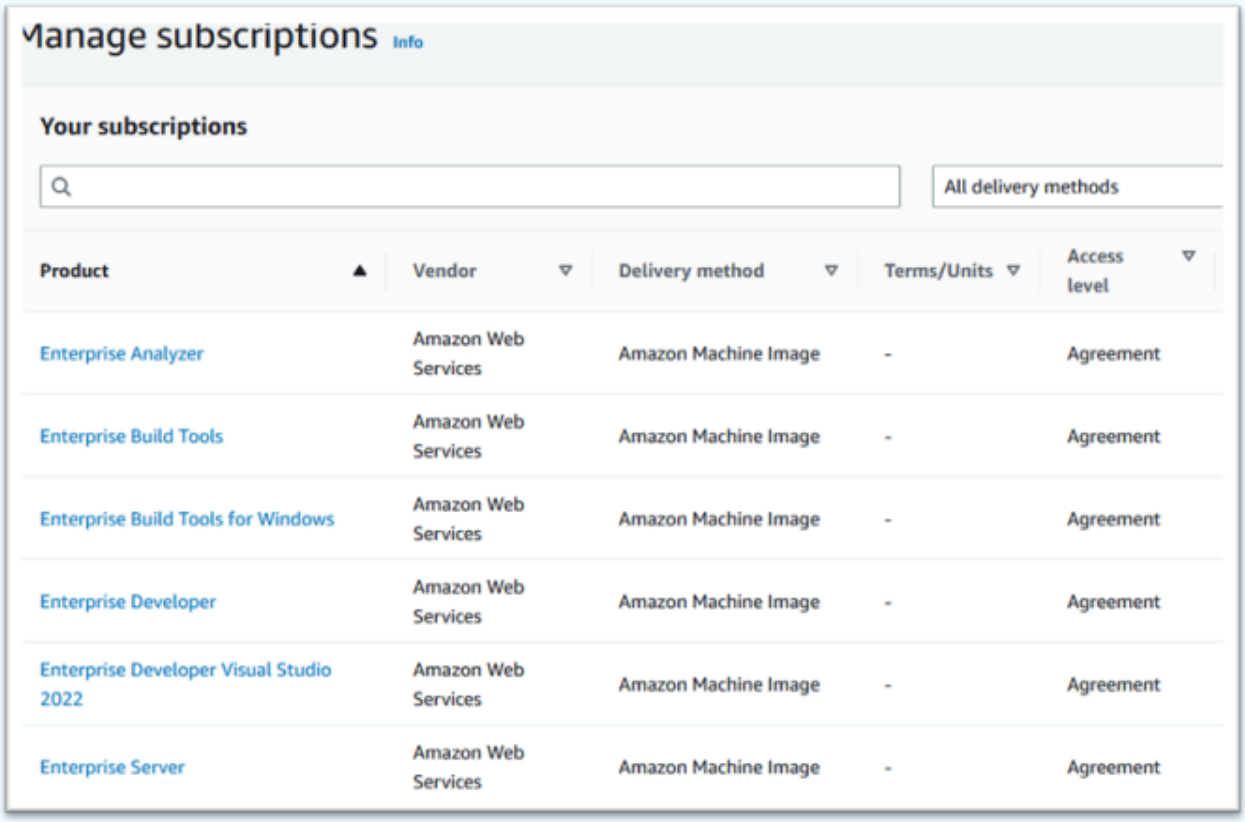


7. お礼のメッセージが表示されたら、ステップ 3 の次のリンクをコピーして貼り付け、サブスクリプションの追加を続行します。



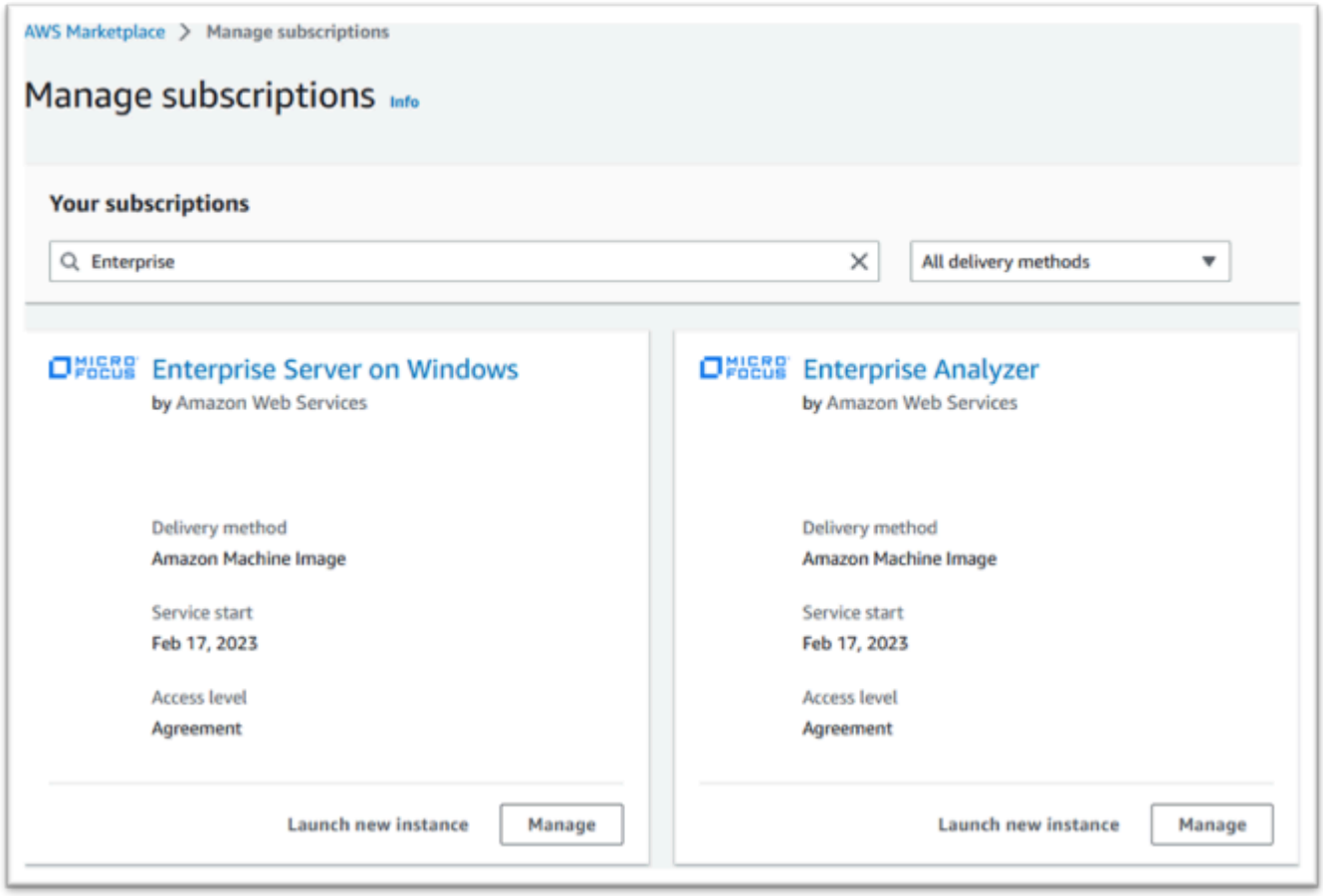
8. サブスクライブしたすべての AMI が [サブスクリプションを管理する] に表示されたら停止します。

> **Note**
>
> パネルの詳細設定 (歯車アイコン) は、テーブルとして表示するように設定されています。



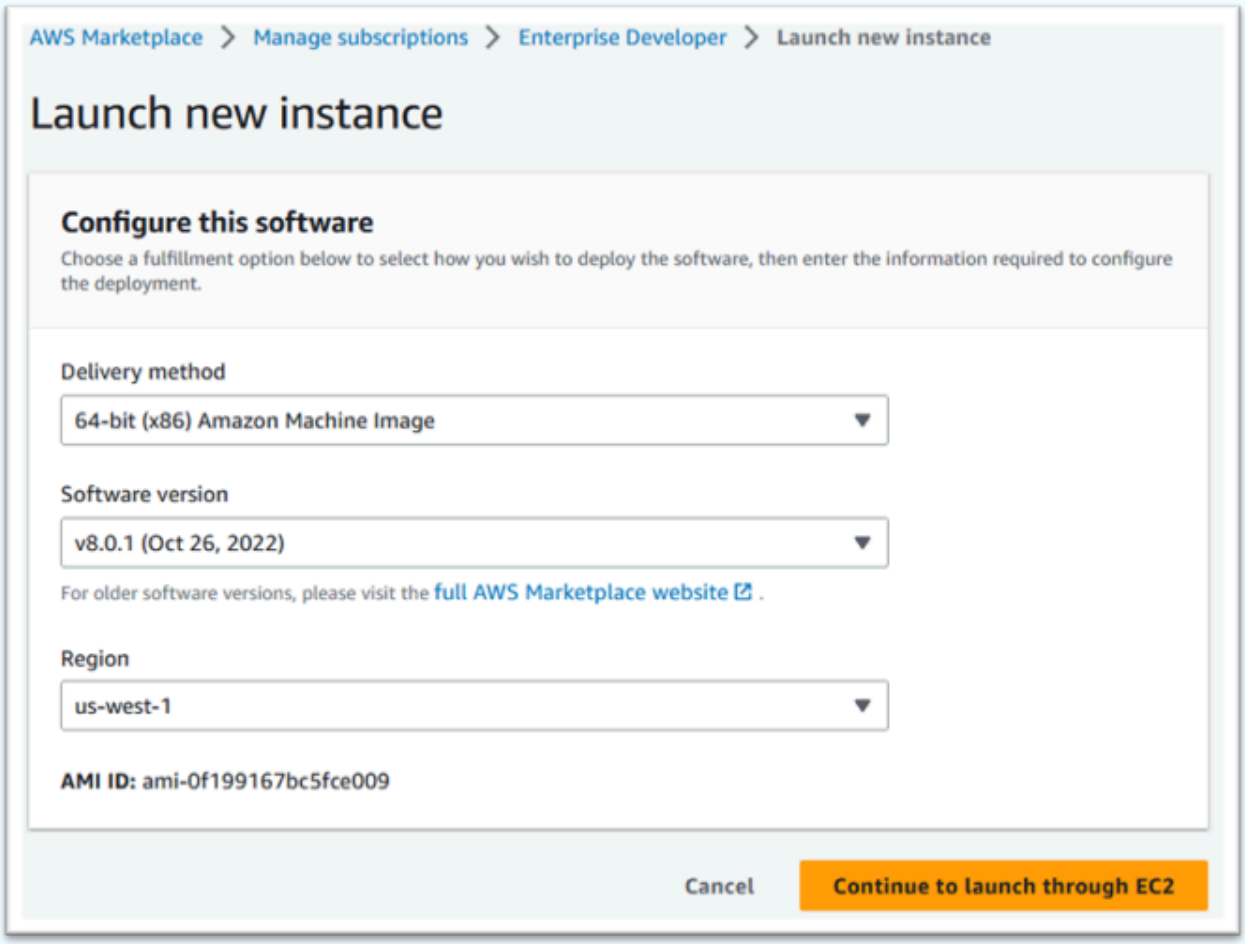
AWS Mainframe Modernization Micro Focus インスタンスを起動する

1. の AWS Marketplace サブスクリプションに移動します AWS Management Console。
2. 起動する AMI を見つけて、[新規インスタンスの起動] を選択します。



3. [新規インスタンスの起動] ダイアログで、許可リストに登録されているリージョンが選択されていることを確認します。
4. [EC2 経由で起動を続行する] を選択します。

> **Note**
>
> 次の例は、Enterprise Developer AMI の起動を示していますが、プロセスはすべての AWS Mainframe Modernization AMIs。

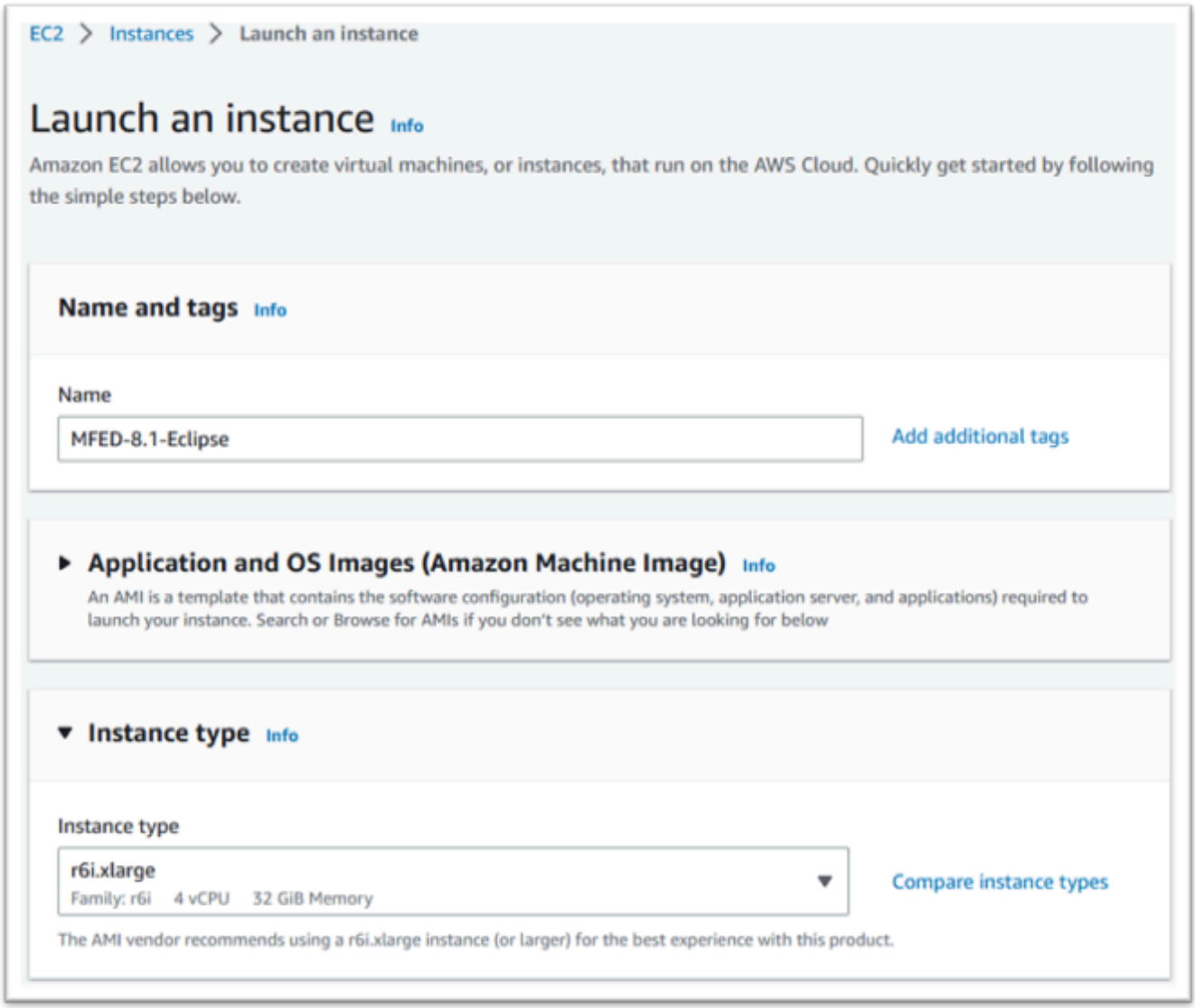


5. サーバーの名前を入力します。
6. インスタンスタイプを選択します。

   選択するインスタンスタイプは、プロジェクトのパフォーマンスとコスト要件によって決定する必要があります。出発点としては以下のものが推奨されます。

   - Enterprise Analyzer の場合、r6i.xlarge
   - Enterprise Developer の場合、r6i.large
   - Enterprise Server のスタンドアロンインスタンスの場合、r6i.xlarge
   - スケールアウト機能付き Micro Focus Performance Availability Cluster (PAC) の場合、r6i.large

> **Note**
>
> スクリーンショットでは、[アプリケーションと OS イメージ] セクションは折りたたまれています。



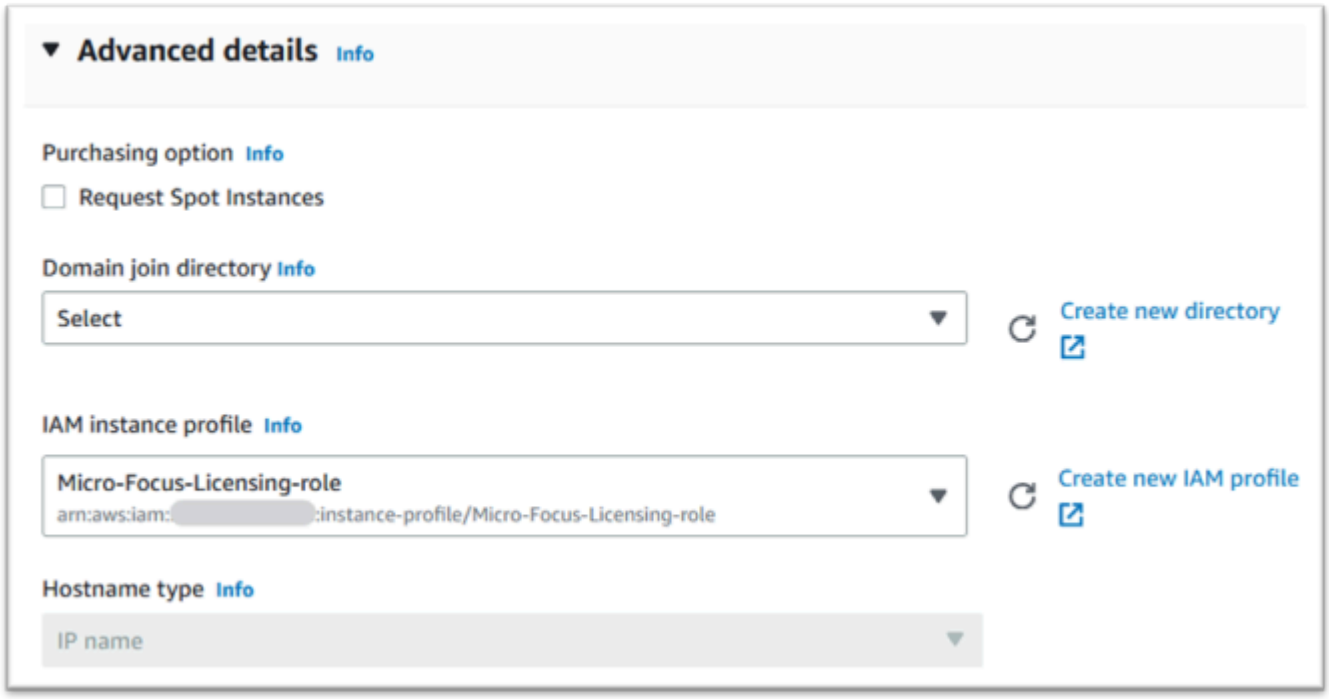
7. キーペアを選択または作成 (および保存) します (ここには表示されていません)。

   キーペアと Linux インスタンスの詳細については、の「Amazon EC2 のキーペアと Linux インスタンス」を参照してください。

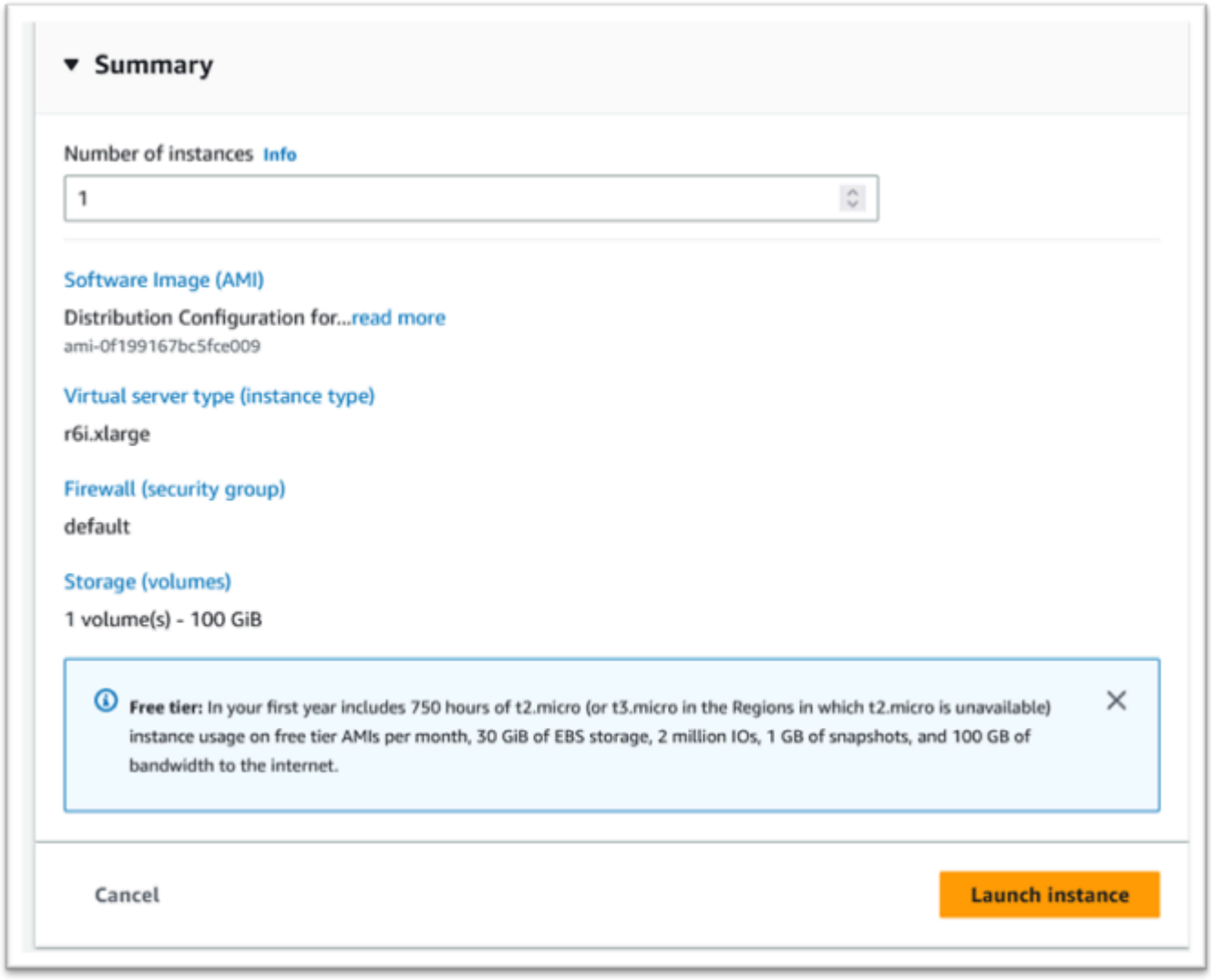
キーペアと Windows インスタンスの詳細については、「Amazon EC2 のキーペアと Windows インスタンス」を参照してください。

8. ネットワーク設定を編集し、許可リストに登録された VPC と適切なサブネットを選択します。
9. セキュリティグループを選択または作成します。Enterprise Server EC2 インスタンスの場合は、通常、ポート 86 と 10086 への TCP トラフィックを許可して Micro Focus の設定を管理します。
10. Amazon EC2 インスタンスのストレージをオプションで設定します。
11. 重要 - [高度な詳細] を展開し、[IAM インスタンスプロファイル] で、先ほど作成したライセンスロール (「Micro-Focus-Licensing-Role」など) を選択します。

> **Note**
>
> このステップを見逃した場合は、インスタンスの作成後に EC2 インスタンスの [アクション] メニューの [セキュリティ] オプションから IAM ロールを変更できます。

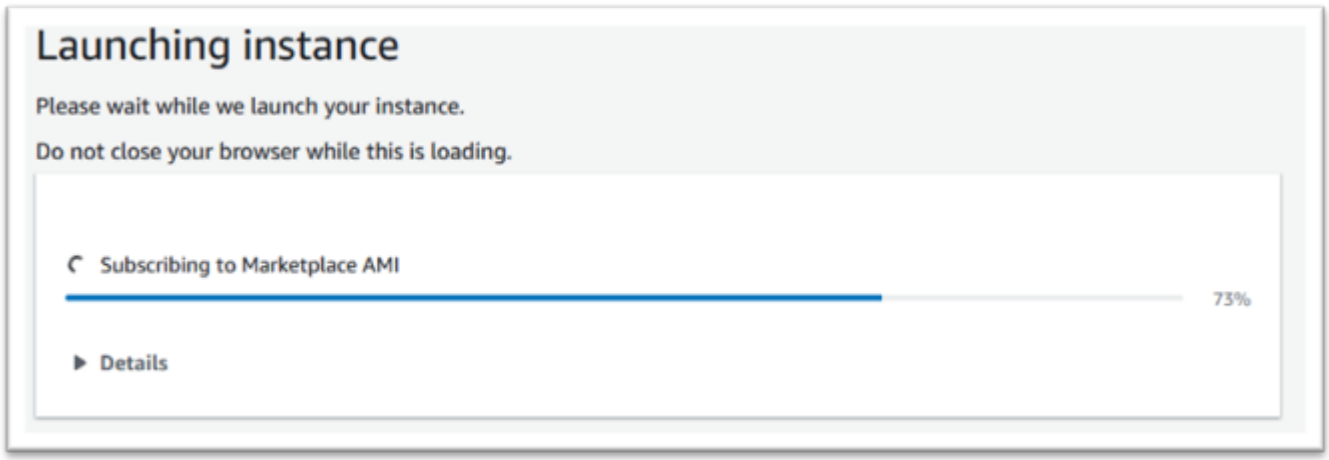


12. 概要を確認し、[インスタンスを起動] を選択します。

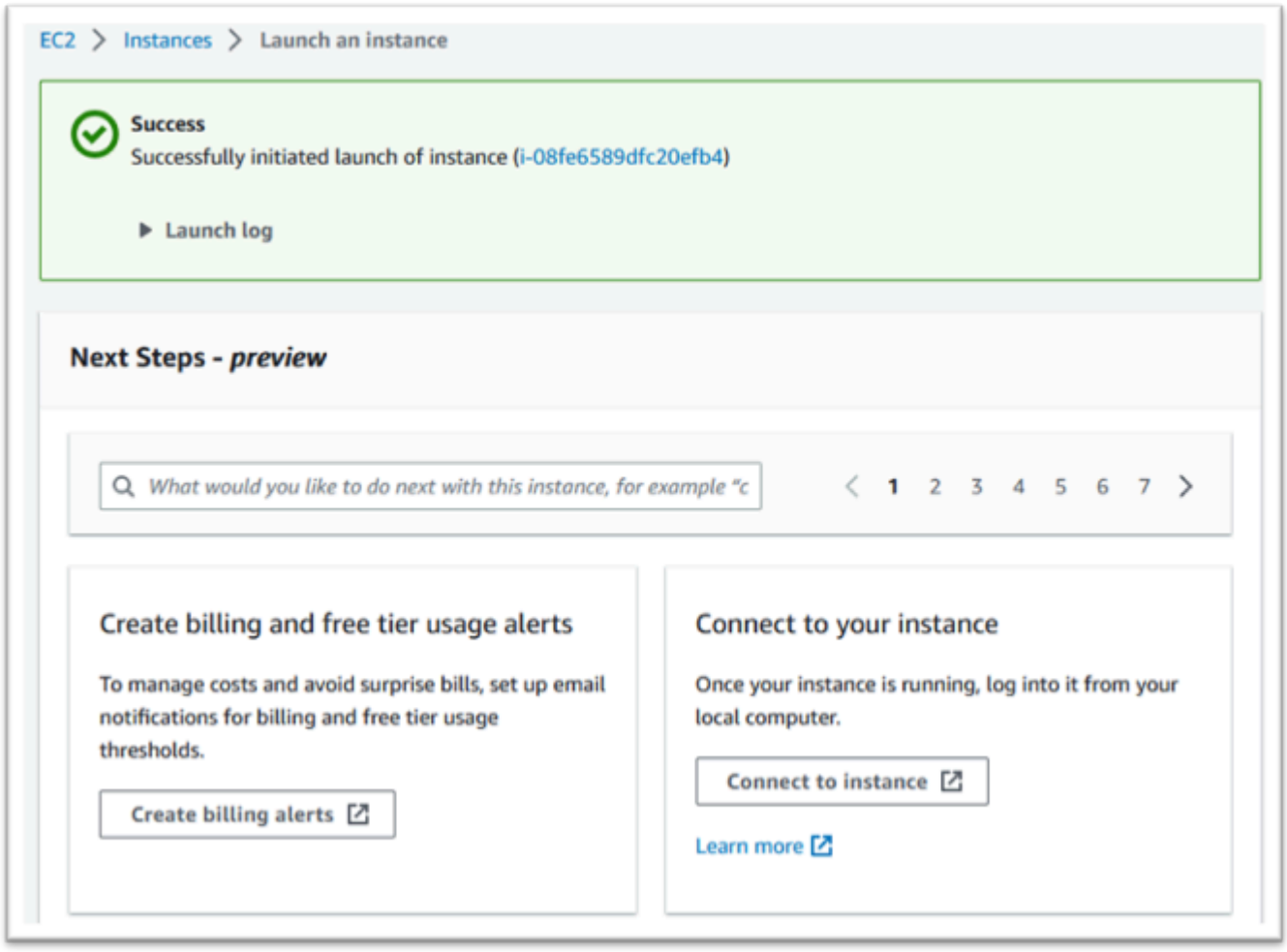


13. 無効な仮想サーバータイプを選択した場合、インスタンスの起動は失敗します。

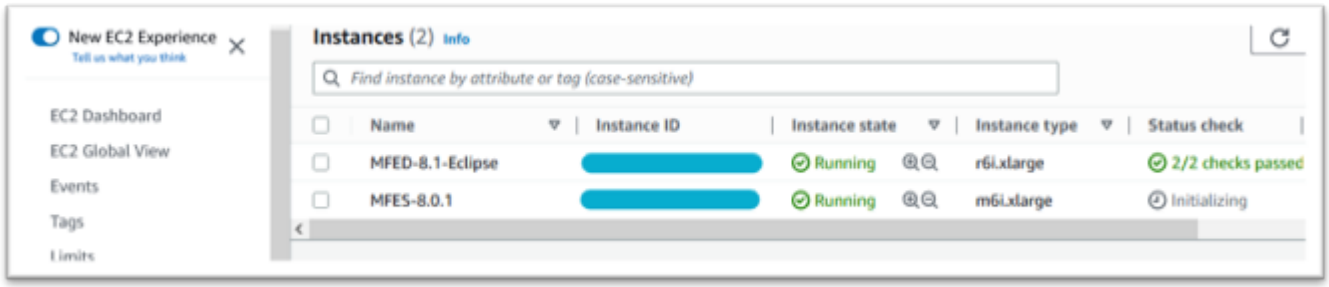
    その場合は、[インスタンス設定を編集] を選択し、インスタンスタイプを変更してください。



14. 「成功」メッセージが表示されたら、[インスタンスに接続] を選択して接続の詳細を取得します。



15. または、 AWS Management Consoleの EC2 に移動します。
16. [インスタンス] を選択すると、新しいインスタンスの状態が表示されます。



インターネットにアクセスできないサブネットまたは VPC

サブネットまたは VPC にアウトバウンドのインターネットアクセスがない場合は、これらの追加変更を行います。

ライセンスマネージャーは、次の AWS サービスにアクセスする必要があります。

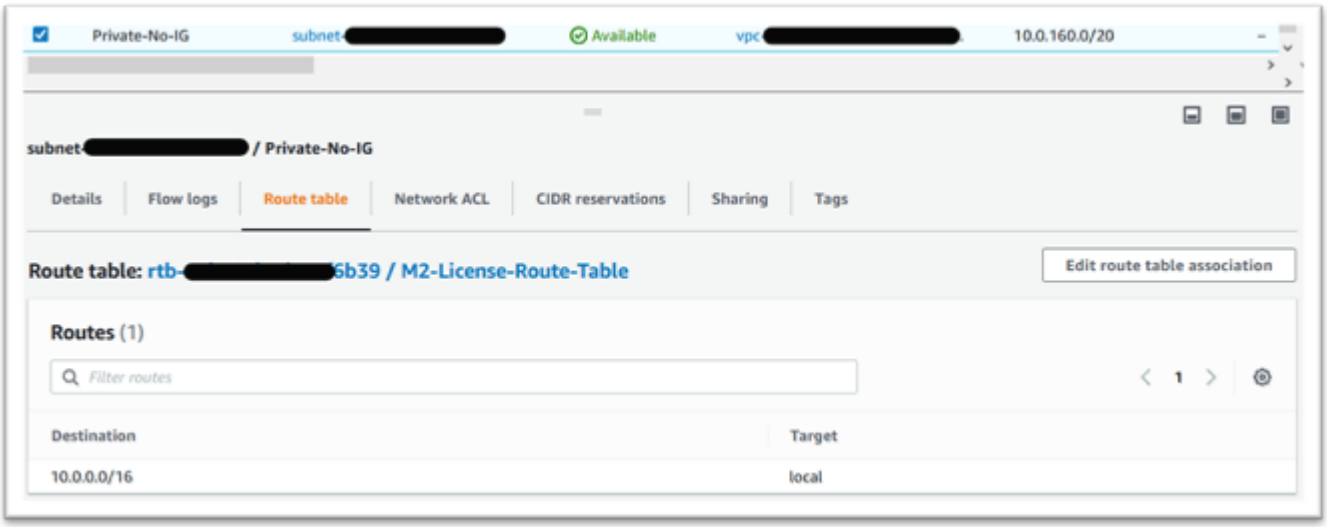
- com.amazonaws.*region*.s3
- com.amazonaws.*region*.ec2
- com.amazonaws.*region*.license-manager
- com.amazonaws.*region*.sts

前のステップでは com.amazonaws.*region*.s3 サービスをゲートウェイエンドポイントとして定義しました。このエンドポイントには、インターネットにアクセスできないサブネットのルートテーブルエントリが必要です。

追加の 3 つのサービスはインターフェイスエンドポイントとして定義されます。

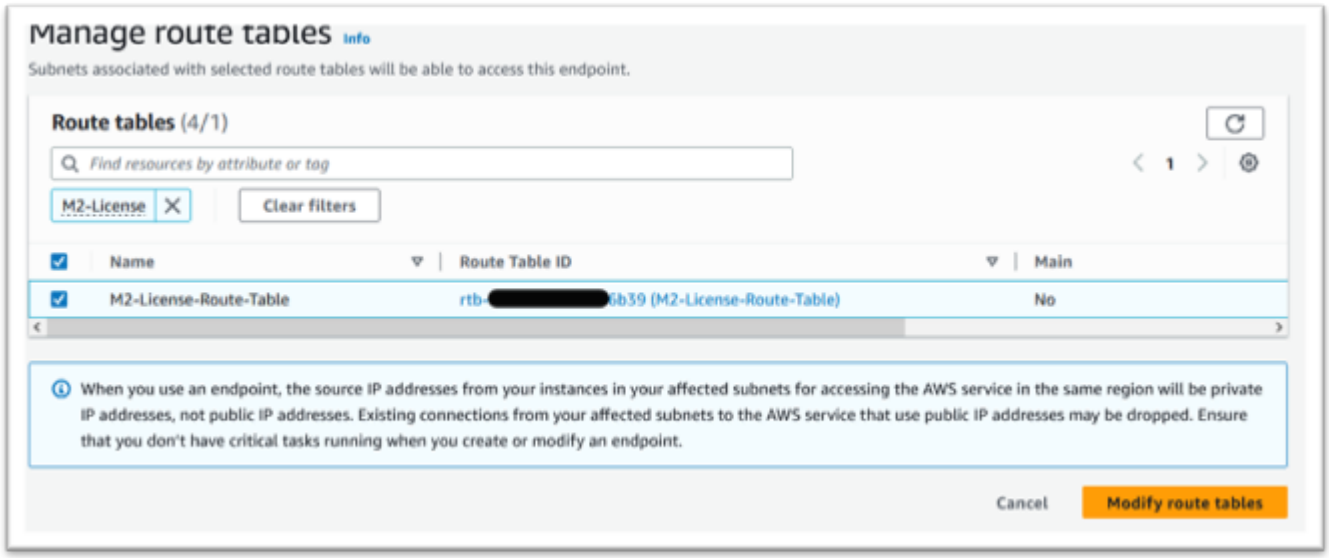
Amazon S3 エンドポイントのルートテーブルエントリの追加

1. で VPC に移動 AWS Management Console し、サブネット を選択します。
2. Amazon EC2 インスタンスを作成するサブネットを選択し、[ルートテーブル] タブを選択します。
3. ルートテーブル ID の末尾の数桁に注意してください。例えば、下の画像の 6b39 です。



4. ナビゲーションペインで、[エンドポイント] を選択します。

5. 先に作成したエンドポイントを選択し、エンドポイントの [ルートテーブル] タブまたは [アクション] ドロップダウンから [ルートテーブルを管理] を選択します。
6. 先ほど確認した数字を使用してルートテーブルを選択し、[ルートテーブルの変更] を選択します。

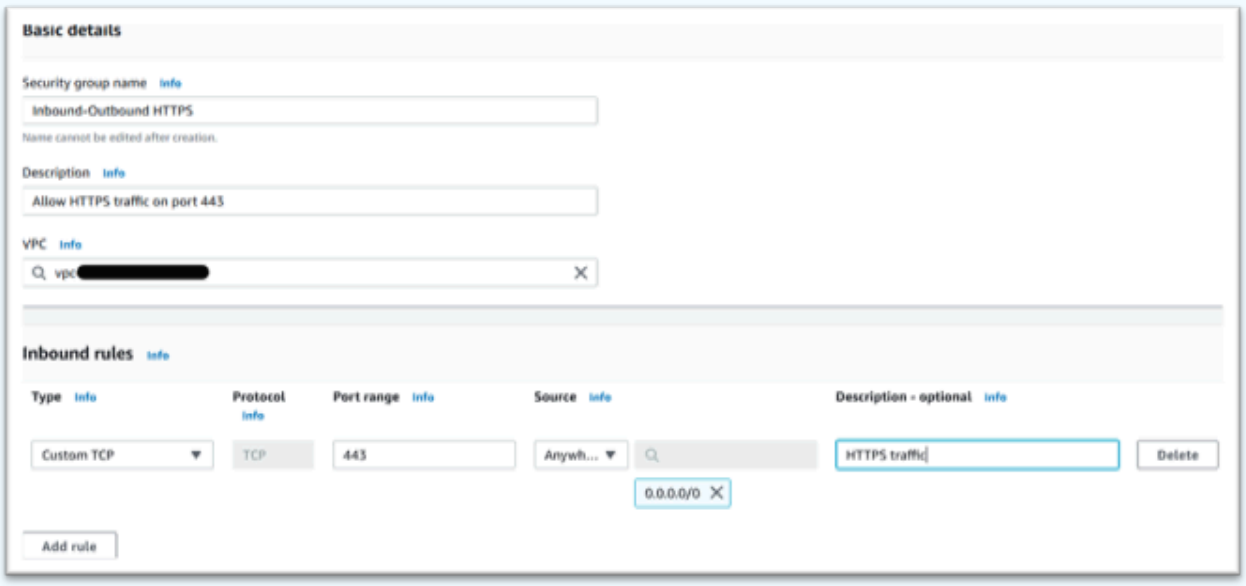


必要なセキュリティグループの定義

Amazon EC2 および License Manager サービスは AWS STS、ポート 443 経由で HTTPS 経由で通信します。この通信は双方向であり、インスタンスがサービスと通信できるようにするにはインバウンドルールとアウトバウンドルールが必要です。

1. AWS Management Consoleの Amazon VPC に移動します。
2. ナビゲーションバーで [セキュリティグループ] を選択して、[セキュリティグループの作成] を選択します。
3. セキュリティグループ名と説明 (「Inbound-Outbound HTTPS」など) を入力します。
4. VPC 選択エリアで [X] を選択してデフォルト VPC を削除し、S3 エンドポイントを含む VPC を選択します。
5. 任意の場所からのポート 443 の TCP トラフィックを許可するインバウンドルールを追加します。

> **Note**
>
> 送信元を制限すると、インバウンド (およびアウトバウンドルール) をさらに制限できます。詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループを使用して AWS リソースへのトラフィックを制御する」を参照してください。

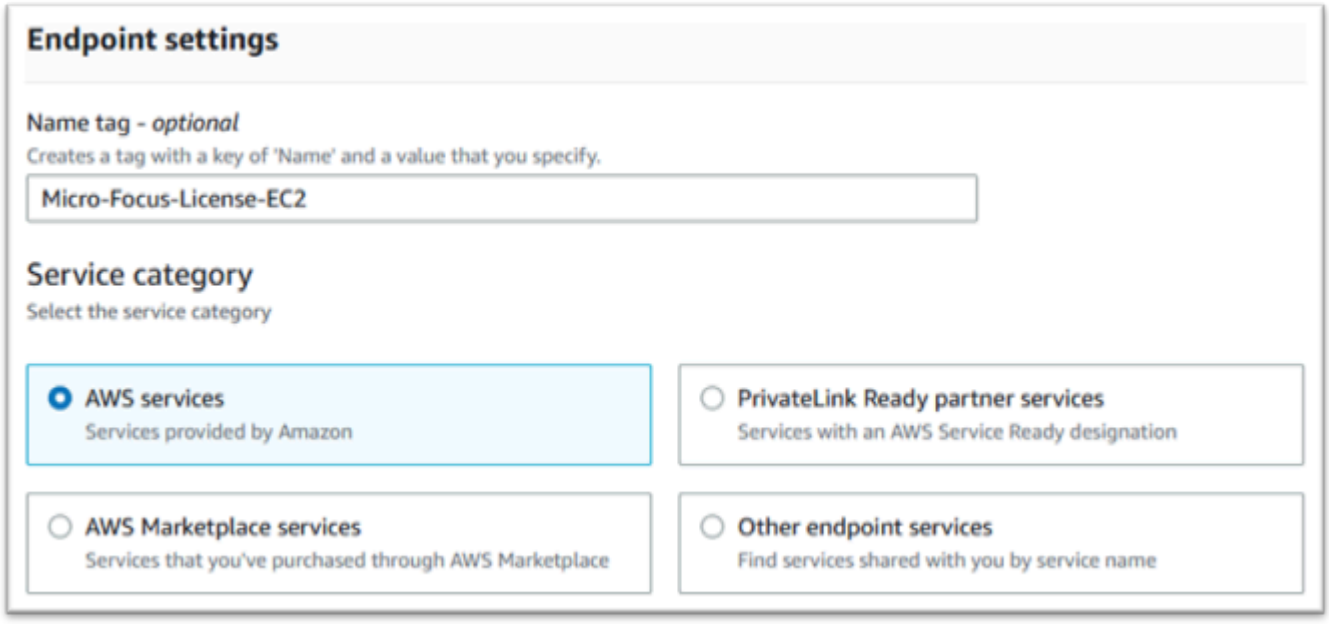


6. [セキュリティグループの作成] を選択します。

サービスエンドポイントの作成

このプロセスをサービスごとに 1 回ずつ、合計 3 回繰り返します。

1. で Amazon VPC に移動 AWS Management Console し、エンドポイント を選択します。
2. [エンドポイントの作成] を選択します。
3. 「Micro-Focus-License-EC2」、「Micro-Focus-License-STS」、「Micro-Focus-License-Manager」などの名前を入力します。
4. [AWS サービス] サービスカテゴリを選択します。



5. [サービス] で、以下のいずれかと一致するインターフェイスサービスを検索します。

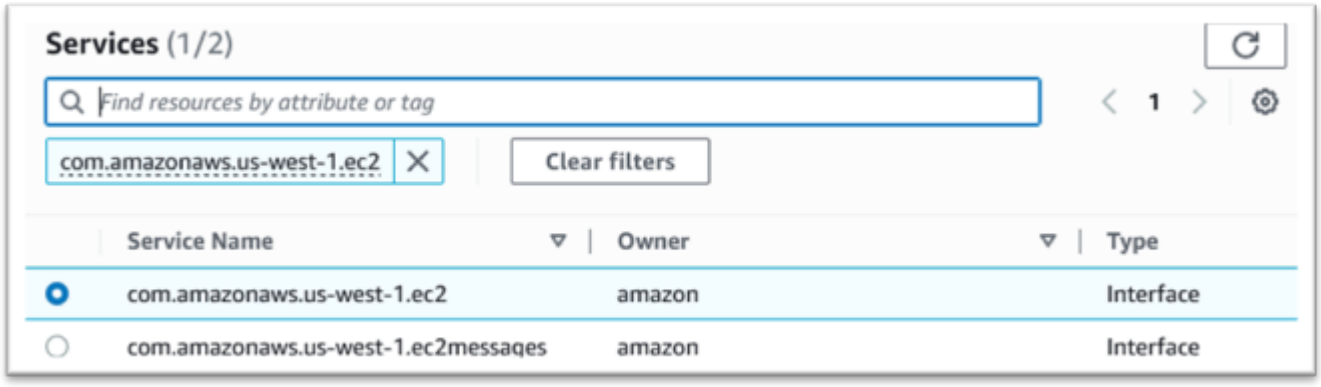
   - com.amazonaws.*region*.ec2
   - com.amazonaws.*region*.sts
   - com.amazonaws.*region*.license-manager

   例:

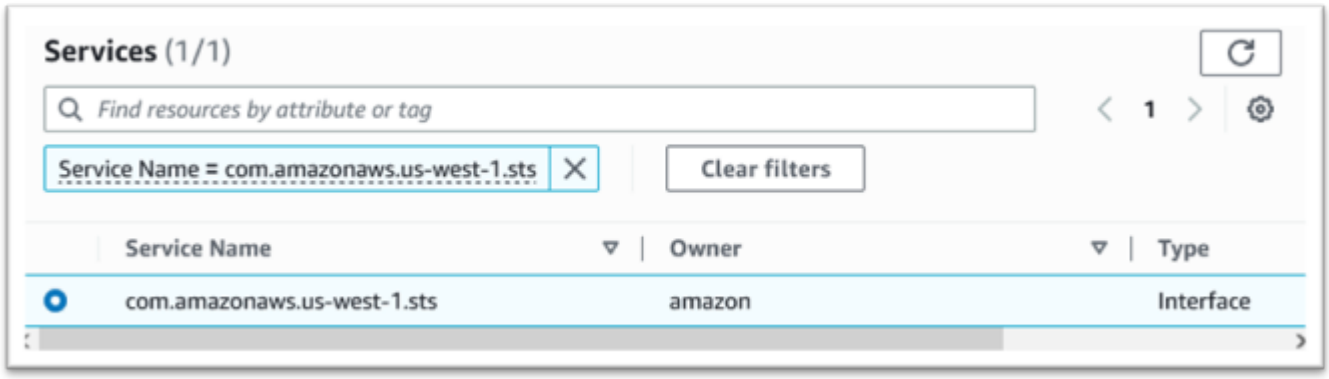
   - com.amazonaws.us-west-1.ec2
   - com.amazonaws.us-west-1.sts
   - com.amazonaws.us-west-1.license-manager

6. 一致するインターフェイスサービスを選択します。

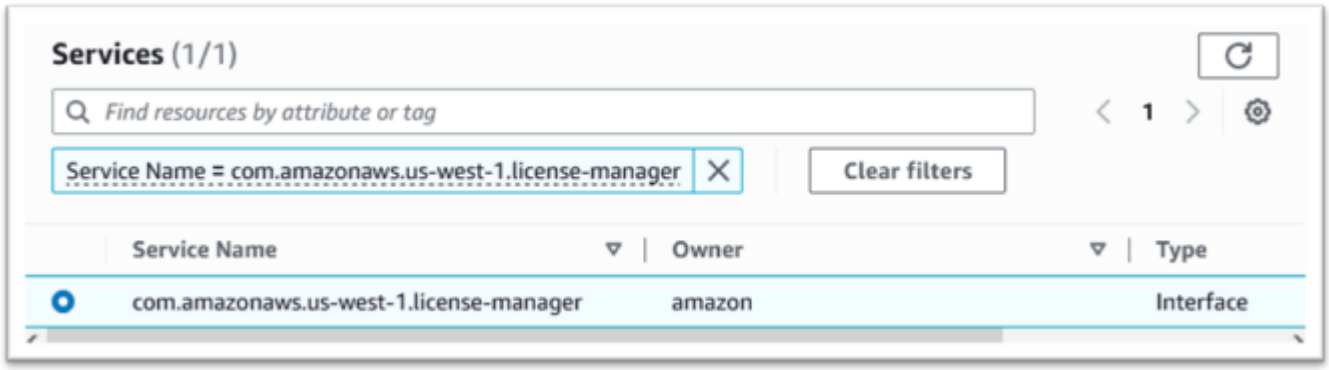
   com.amazonaws.***region***.ec2:



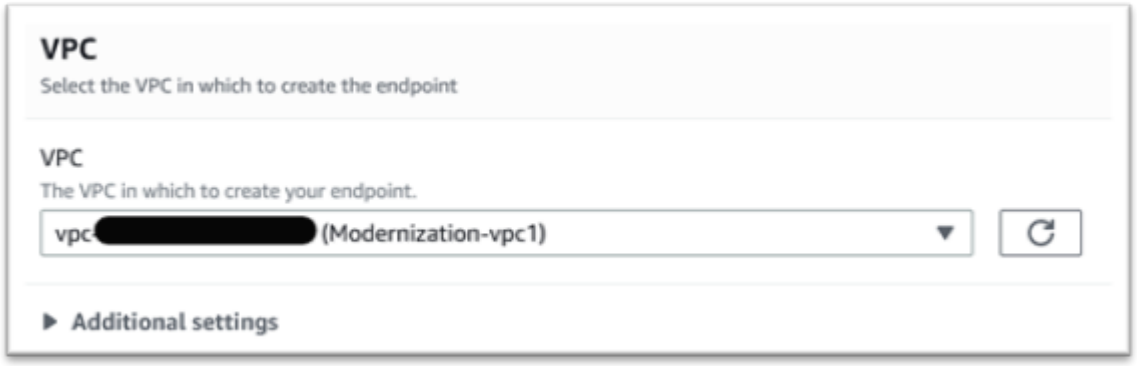
com.amazonaws.***region***.sts:



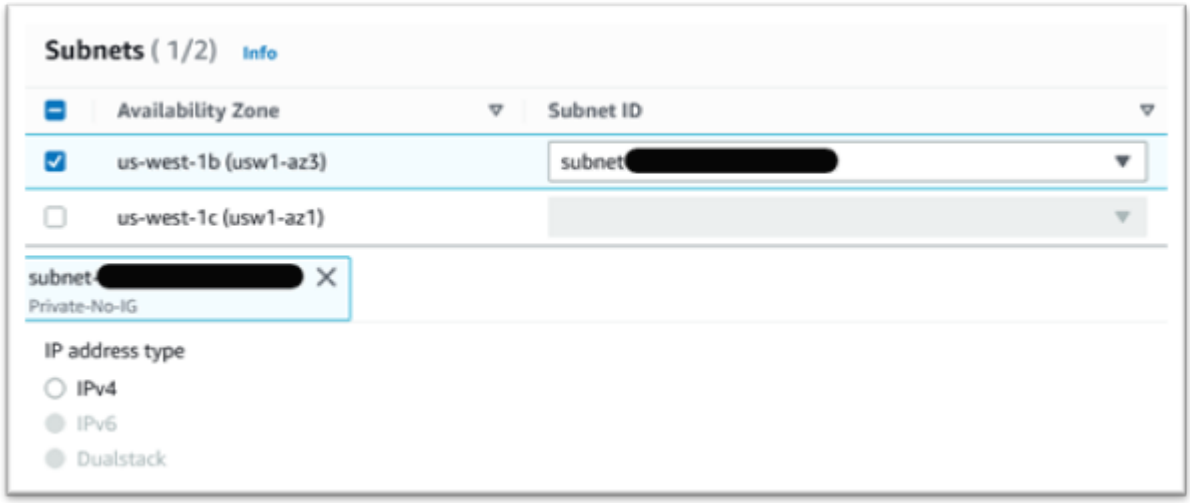
com.amazonaws.***region***.license-manager:



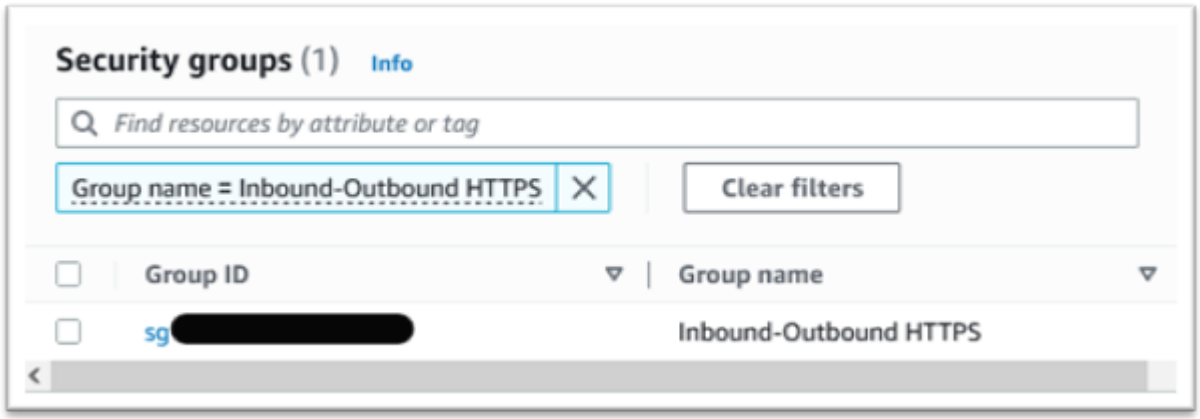
7. [VPC] で、インスタンスの VPC を選択します。



8. VPC の [アベイラビリティーゾーン] と [サブネット] を選択します。



9. 前に作成したセキュリティグループを選択します。



10. [ポリシー] で [フルアクセス] を選択します。

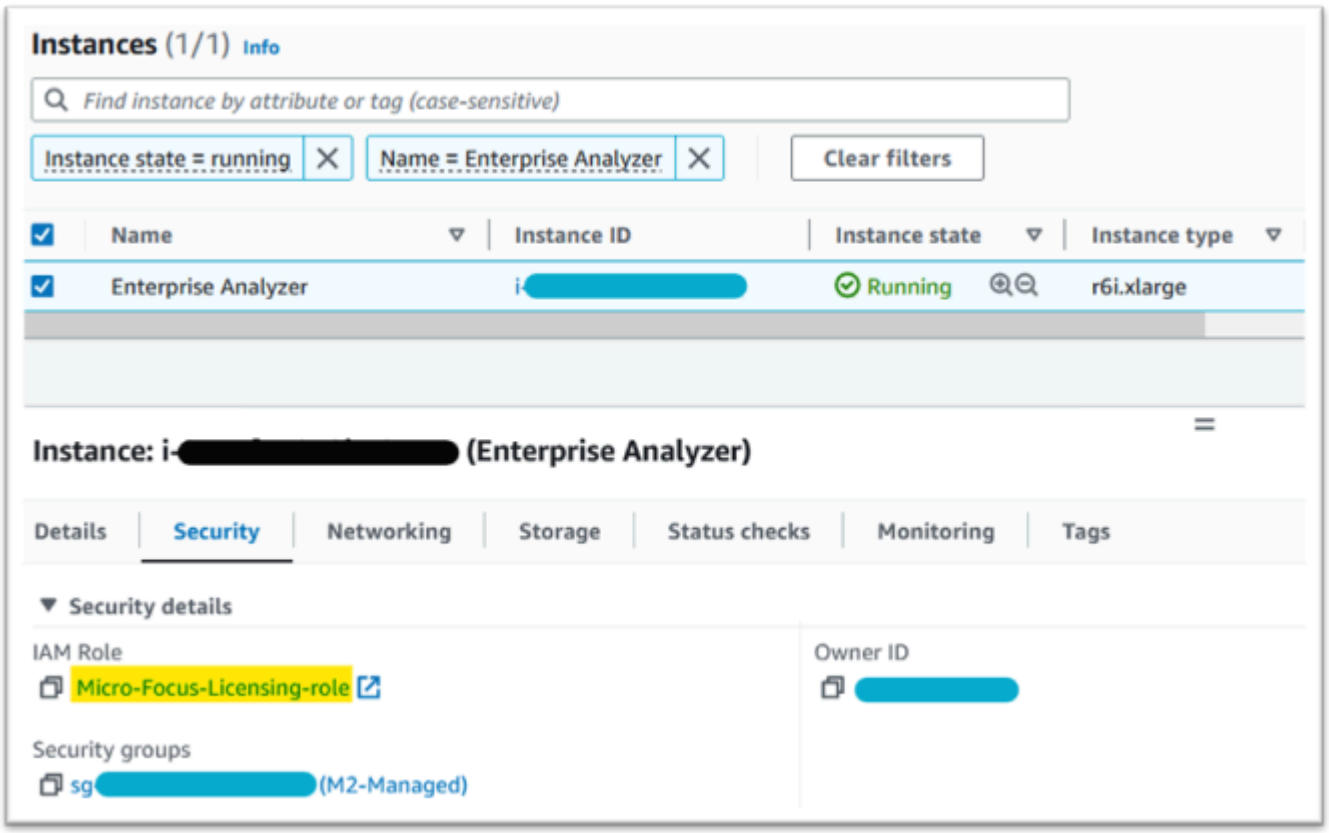
11. [エンドポイントの作成] を選択します。
12. 残りのインターフェイスでこのプロセスを繰り返します。

ライセンスの問題のトラブルシューティング

AMI へのアクセスや使用に問題がある場合は、次の情報が役に立つことがあります。

Amazon EC2 インスタンスに IAM ライセンスロールがあることを確認する

これは Amazon EC2 インスタンス詳細の [セキュリティ] タブで確認できます。これは [アクション] ドロップダウンメニューの [セキュリティオプション] を使用して変更できます。



Reachability Analyzer を使用する

AWS Network Manager コンソールページで Reachability Analyzer を見つけます。

AMI から作成された Amazon EC2 インスタンスと Amazon S3 VPC エンドポイント間のパスを作成して分析します。

Amazon EC2 インスタンスがインターネットにアクセスできない場合は、4 つのエンドポイントすべてへのパス分析を繰り返します。

Reachability Analyzer の詳細については、「Reachability Analyzer ガイド」の「Getting started with Reachability Analyzer」を参照してください。

ライセンスデーモンを実行する

Windows Enterprise Developer では、コマンドプロンプトから以下のコマンドを使用します。

```
“C:\Program Files (x86)\Micro Focus\Enterprise Developer\AdoptOpenJDK\bin\java” -jar
 "C:\Program Files (x86)\Micro Focus\Licensing\aws-license-daemon.jar"
```

出力を確認します。SLF4J のメッセージは無視して、最初の例外を探してください。

Enterprise Analyzer では、コマンドプロンプトから以下のコマンドを使用します。

```
"C:\Program Files (x86)\Micro Focus\AdoptOpenJDK\bin\java" -jar "C:\Program Files
 (x86)\Micro Focus\Licensing\aws-license-daemon.jar"
```

出力を確認します。SLF4J のメッセージは無視して、最初の例外を探してください。

Linux では、以下を実行します。

```
java -jar /var/microfocuslicensing/bin/aws-license-daemon.jar
```

SLF4J のメッセージは無視して、最初の例外を探してください。

例えば、Amazon S3 リソースが利用できない場合、例外は次のとおりです。

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Exception in thread "main" software.amazon.awssdk.services.s3.model.S3Exception: Access
 Denied (Service: S3, Status Code: 403, Request ID: P6
```

例外メッセージには、どのリソースが使用できないかが示されます。設定値をこのトピックに示されている値と比較してください。

チュートリアル: Micro Focus Enterprise Analyzer と Micro Focus Enterprise Developer で使用するための AppStream 2.0 のセットアップ

AWS Mainframe Modernization では、Amazon AppStream 2.0 を通じて複数のツールが用意されています。AppStream 2.0 は、アプリケーションを書き換えることなく、ユーザーにデスクトップアプリケーションをストリーミングするための、完全マネージド型のセキュアなアプリケーションストリーミングサービスです。AppStream 2.0 を使用すると、ユーザーは必要なアプリケーションに即座にアクセスできます。ユーザーが選択したデバイス上で応答性が高く、流動的なユーザーエクスペリエンスを提供します。AppStream 2.0 を使用してランタイムエンジン固有のツールをホストすると、顧客のアプリケーションチームはウェブブラウザから直接ツールを使用し、Amazon S3 バケットまたは CodeCommit リポジトリに保存されているアプリケーションファイルを操作できます。

AppStream 2.0 でのブラウザのサポートについては、「Amazon AppStream 2.0 管理ガイド」の「システム要件と機能サポート (ウェブブラウザ)」を参照してください。AppStream 2.0 の使用中に問題が発生した場合は、「Amazon AppStream 2.0 管理ガイド」の「AppStream 2.0 ユーザー問題のトラブルシューティング」を参照してください。

このドキュメントは、カスタマーオペレーションチームのメンバーを対象としています。AWS Mainframe Modernization で使用される Micro Focus Enterprise Analyzer と Micro Focus Enterprise Developer ツールをホストするために Amazon AppStream 2.0 フリートとスタックを設定する方法について説明します。AWS Micro Focus Enterprise Analyzer は通常、Mainframe Modernization アプローチの評価フェーズで使用され、Micro Focus Enterprise Developer は通常、移行とモダナイズのフェーズで使用されます。Enterprise Analyzer と Enterprise Developer の両方を使用する予定がある場合は、ツールごとに別々のフリートとスタックを作成する必要があります。ライセンス条件が異なるため、ツールごとに独自のフリートとスタックが必要です。

> **Important**
>
> このチュートリアルの手順は、ダウンロード可能な AWS CloudFormation テンプレート cfn-m2-appstream-fleet-ea-ed.yml に基づいています。

トピック

前提条件

- テンプレート cfn-m2-appstream-fleet-ea-ed.yml をダウンロードしてください。
- デフォルト VPC とセキュリティグループの ID を取得します。詳細については、「Amazon VPC ユーザーガイド」の「デフォルト VPC とデフォルトサブネット」を参照してください。デフォル

トのセキュリティグループの詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「デフォルトセキュリティグループとカスタムセキュリティグループ」を参照してください。

- 次のアクセス許可があることを確認します。
  - AppStream 2.0 でスタック、フリート、およびユーザーを作成します。
  - テンプレートを使用して AWS CloudFormation にスタックを作成します。
  - バケットを作成し、Amazon S3 のバケットにファイルをアップロードします。
  - IAM から認証情報 (`access_key_id`と`secret_access_key`) をダウンロードします。

ステップ 1: AppStream 2.0 イメージを取得する

このステップでは、Enterprise Analyzer と Enterprise Developer 用の AppStream 2.0 イメージを AWS アカウントと共有します。

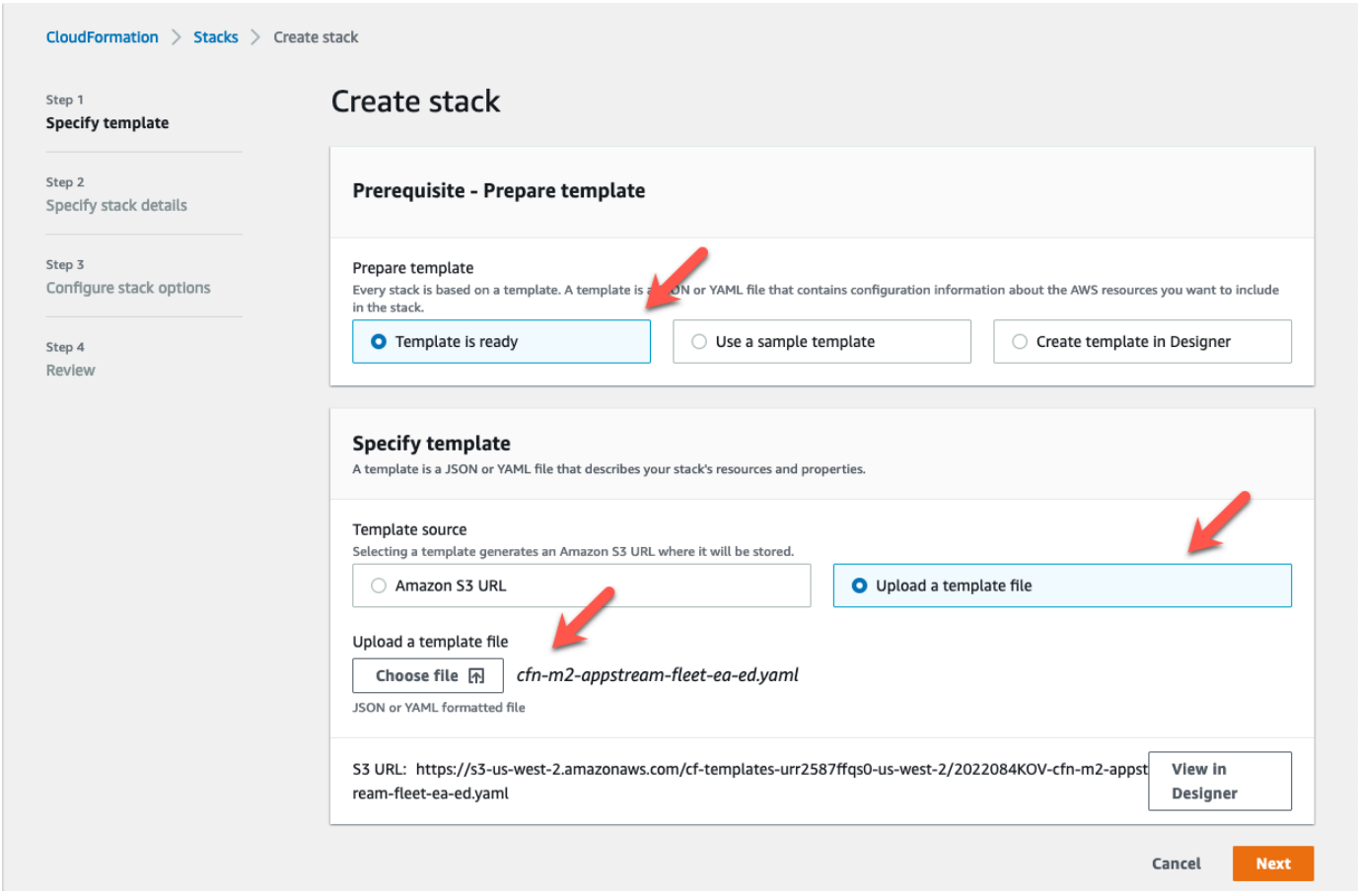
1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. 左側のナビゲーションで、[ツール] を選択します。
3. [分析、開発、およびアセットの構築] で、[AWS アカウントとアセットを共有する] を選択します。

ステップ 2: AWS CloudFormation テンプレートを使用してスタックを作成する

このステップでは、ダウンロードした AWS CloudFormation テンプレートを使用して、Micro Focus Enterprise Analyzer を実行するための AppStream 2.0 スタックとフリートを作成します。後でこの手順を繰り返して、Micro Focus Enterprise Developer を実行するための別の AppStream 2.0 スタックとフリートを作成できます。これは、AppStream 2.0 では各ツールに独自のフリートとスタックが必要だからです。AWS CloudFormation スタックの詳細については、「AWS CloudFormation ユーザーガイド」の「スタックの操作」を参照してください。

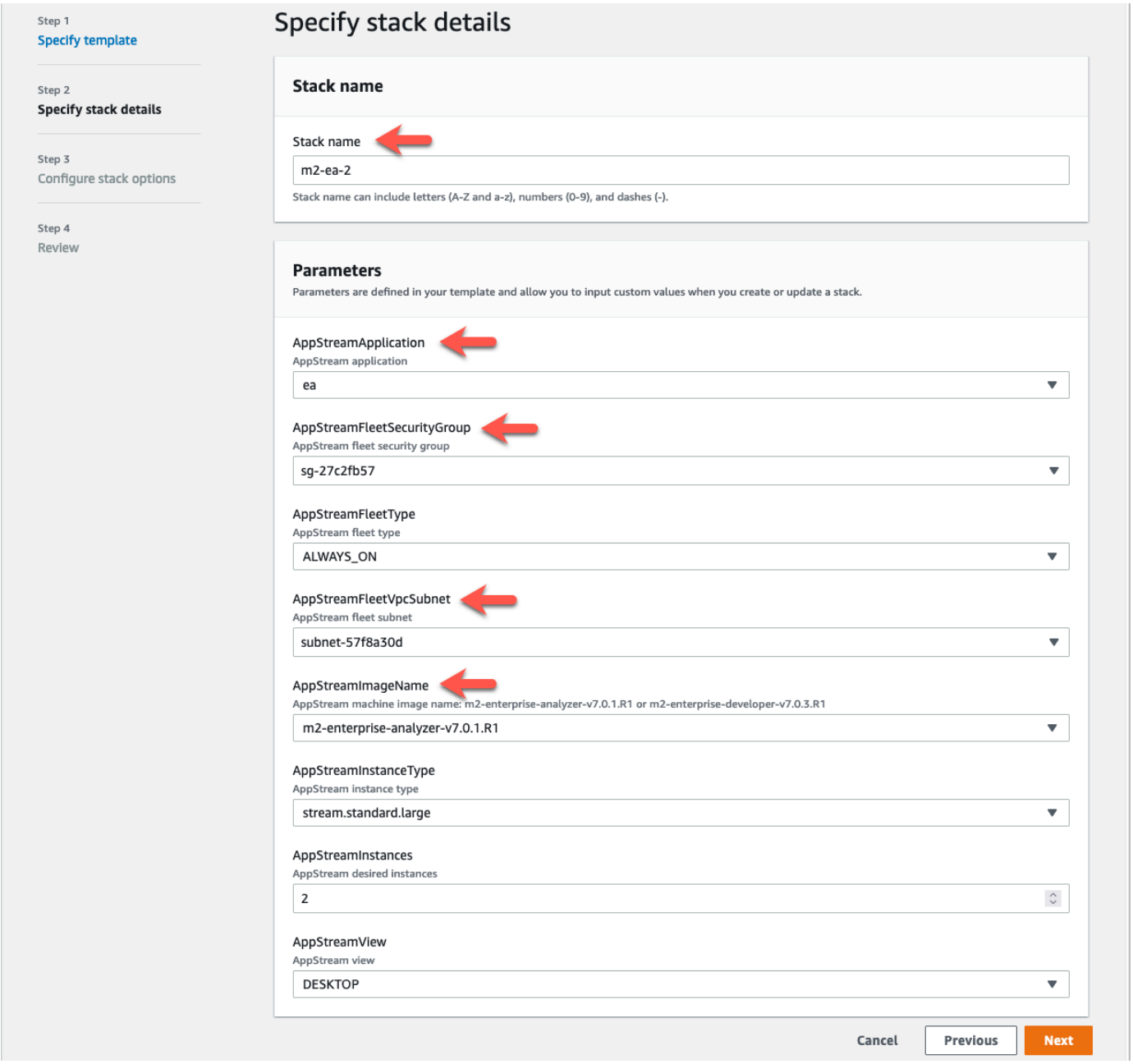
> **Note**
>
> AWS Mainframe Modernization では、Enterprise Analyzer と Enterprise Developer の使用に関する AppStream 2.0 の標準料金に追加料金が加算されます。詳細については、「AWS Mainframe Modernization pricing」を参照してください。

1. 必要に応じて、cfn-m2-appstream-fleet-ea-ed.yml テンプレートをダウンロードしてください。
2. AWS CloudFormation コンソールを開き、[スタックの作成] と [新しいリソースを使用 (標準)] を選択します。
3. [前提条件 - テンプレートの準備] で、[テンプレートの準備完了] を選択します。
4. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択します。
5. [テンプレートファイルのアップロード] で [ファイルを選択] を選択し、cfn-m2-appstream-fleet-ea-ed.yml テンプレートをアップロードします。
6. [次へ] をクリックします。



7. [スタックの詳細を指定] で、次の情報を入力します。

- [スタック名] で、希望する名前を入力します。例えば、**m2-ea** です。
- [AppStreamApplication] で、[ea] を選択します。
- [AppStreamFleetSecurityGroup] で、デフォルトの VPC のデフォルトのセキュリティグループを選択します。
- [AppStreamFleetVpcSubnet] で、デフォルトの VPC 内のサブネットを選択します。
- [AppStreamImageName] で、`m2-enterprise-analyzer` で始まるイメージを選択します。このイメージには、現在サポートされているバージョンの Micro Focus Enterprise Analyzer ツールが含まれています。
- 他のフィールドはデフォルトのままにし、[次へ] を選択します。



8. すべてをデフォルトのままにし、[次へ] を選択します。
9. [レビュー] で、すべてのパラメータが意図したとおりであることを確認します。
10. 一番下までスクロールし、[AWS CloudFormation によって IAM リソースが作成される場合があることを承認します] を選択し、[スタックの作成] を選択します。

スタックとフリートが作成されるまで 20～30 分かかります。[最新表示] を選択すると、発生した AWS CloudFormation イベントを確認できます。

ステップ 3: AppStream 2.0 でユーザーを作成する

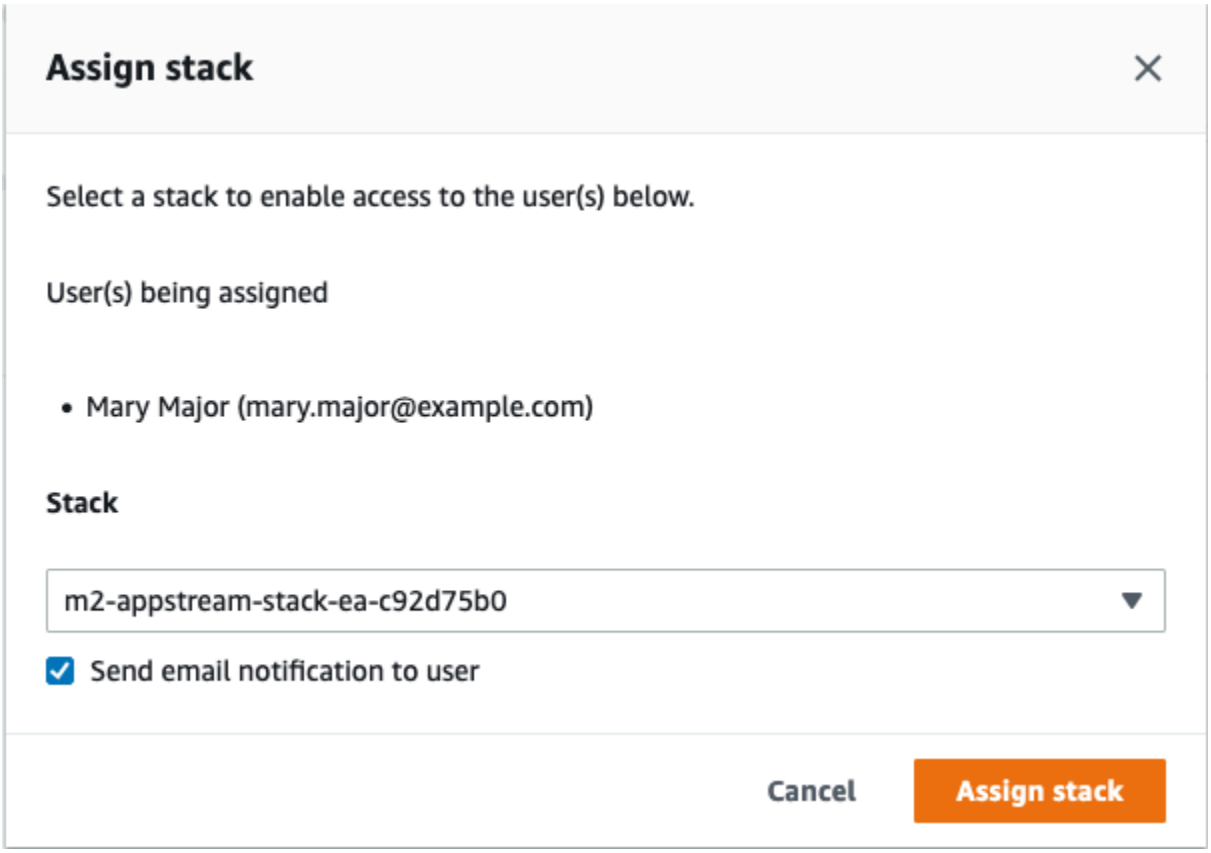
AWS CloudFormation がスタックの作成を完了するのを待っている間に、AppStream 2.0 で 1 人以上のユーザーを作成できます。これらのユーザーは、AppStream 2.0 で Enterprise Analyzer を使用するユーザーです。各ユーザーの E メールアドレスを指定し、各ユーザーが Amazon S3 でバケットを作成し、バケットにファイルをアップロードし、バケットにリンクして内容をマッピングするための十分なアクセス許可を持っていることを確認する必要があります。

1. AppStream 2.0 コンソールを開きます。
2. 左側のナビゲーションで、[ユーザープール] を選択します。
3. [Create user] (ユーザーの作成) を選択します。
4. ユーザーが AppStream 2.0 を使用する招待メールを受信できるメールアドレス、姓名を入力し、[ユーザーの作成] を選択します。
5. 必要に応じて同じ手順を繰り返して、さらにユーザーを作成します。各メールアドレスは異なるメールアドレスでなければなりません。

AppStream 2.0 ユーザーの作成の詳細については、「Amazon AppStream 2.0 管理ガイド」の「AppStream 2.0 ユーザープール」を参照してください。

AWS CloudFormation がスタックの作成を終了したら、次のように作成したユーザーをスタックに割り当てることができます。

1. AppStream 2.0 コンソールを開きます。
2. ユーザー名を選択します。
3. [アクション]、[スタックを割り当てる] の順に選択します。
4. [スタックを割り当てる] で、`m2-appstream-stack-ea` で始まるスタックを選択します。
5. [Assign stack] を選択します。



スタックにユーザーを割り当てると、AppStream 2.0 は指定したアドレスのユーザーにメールを送信します。このメールには、AppStream 2.0 ログインページへのリンクが含まれています。

ステップ 4: AppStream 2.0 にログインする

このステップでは、AppStream 2.0 から ステップ 3: AppStream 2.0 でユーザーを作成する で作成したユーザーに送信された E メール内のリンクを使用して、AppStream 2.0 にログインします。

1. AppStream 2.0 から送信された E メールに記載されているリンクを使用して、AppStream 2.0 にログインします。
2. プロンプトが表示されたら、パスワードを変更します。表示される AppStream 2.0 画面は、次のようになります。

3. [デスクトップ] を選択します。
4. タスクバーで [検索] を選択し、**D:** を入力してホームフォルダに移動します。

> **Note**
>
> このステップを省略すると、ホームフォルダにアクセスしようとしたときに `Device not ready` エラーが発生することがあります。

AppStream 2.0 へのサインインに問題がある場合は、いつでも AppStream 2.0 フリートを再起動して、次のステップを使用してサインインを再試行できます。

1. AppStream 2.0 コンソールを開きます。
2. 左側のナビゲーションペインの [フリート] を選択します。
3. 使用するフリートを選択してください。
4. [アクション] を選択し、[停止] を選択します。
5. フリートが停止するのを待ちます。
6. [アクション] を選択してから、[開始] を選択します。

このプロセスには 10 分ほどかかる場合があります。

ステップ 5: Amazon S3 のバケットを確認する (オプション)

スタックの作成に使用した AWS CloudFormation テンプレートによって完了したタスクの 1 つは、Amazon S3 に 2 つのバケットを作成することでした。これらのバケットは、複数の作業セッションにわたってユーザーデータとアプリケーション設定を保存および復元するために必要です。これらのバケットは以下のとおりです。

- 名前は `appstream2-` で始まります。このバケットは AppStream 2.0 (`D:\PhotonUser\My Files\Home Folder`) のホームフォルダにデータをマッピングします。

  > **Note**
  >
  > ホームフォルダは特定の E メールアドレスに固有のもので、特定の AWS アカウントのすべてのフリートとスタックで共有されます。ホームフォルダの名前は、ユーザーのメールアドレスの SHA256 ハッシュで、そのハッシュに基づくパスに保存されます。

- 名前は `appstream-app-settings-` で始まります。このバケットには AppStream 2.0 のユーザーセッション情報が含まれており、ブラウザのお気に入り、IDE とアプリケーションの接続プロファイル、UI のカスタマイズなどの設定が含まれています。詳細については、「Amazon AppStream 2.0 管理ガイド」の「アプリケーション設定の永続化の仕組み」を参照してください。

バケットが作成されたことを確認するには、次のステップに従います。

1. Amazon S3 コンソールを開きます。
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. [名前でバケットを検索] に、「**appstream**」と入力してリストをフィルタリングします。

バケットが表示されている場合は、これ以外のアクションは必要ありません。バケットが存在することだけは覚えておいてください。バケットが表示されない場合は、AWS CloudFormation テンプレートの実行が終了していないか、エラーが発生しています。AWS CloudFormation コンソールに移動し、スタック作成メッセージを確認します。

次のステップ

AppStream 2.0 インフラストラクチャがセットアップされたので、Enterprise Analyzer をセットアップして使用を開始できます。詳細については、「チュートリアル: AppStream 2.0 での Enterprise Analyzer のセットアップ」を参照してください。Enterprise Developer をセットアップすることもで

きます。詳細については、「チュートリアル: AppStream 2.0 で Micro Focus Enterprise Developer をセットアップする」を参照してください。

リソースをクリーンアップする

作成したスタックとフリートをクリーンアップする手順については、「AppStream 2.0 フリートとスタックを作成する」で説明されています。

AppStream 2.0 オブジェクトが削除されると、アカウント管理者は必要に応じて、アプリケーション設定とホームフォルダの Amazon S3 バケットをクリーンアップすることもできます。

> **Note**
>
> 特定のユーザーのホームフォルダはすべてのフリートで一意であるため、同じアカウントで他の AppStream 2.0 スタックがアクティブになっている場合は、そのホームフォルダを保持する必要がある場合があります。

最後に、AppStream 2.0 では、コンソールを使用してユーザーを削除することはできません。代わりに、CLI でサービス API を使用する必要があります。詳細については、「Amazon AppStream 2.0 管理ガイド」の「ユーザープール管理」を参照してください。

チュートリアル: AppStream 2.0 での Enterprise Analyzer のセットアップ

このチュートリアルでは、Micro Focus Enterprise Analyzer をセットアップして 1 つ以上のメインフレームアプリケーションを分析する方法について説明します。Enterprise Analyzer ツールは、アプリケーションのソースコードとシステム定義の分析に基づいて複数のレポートを提供します。

この設定は、チームのコラボレーションを促進するように設計されています。インストールでは Amazon S3 バケットを使用して、仮想ディスクとソースコードを共有します。これを行うには、Windows マシンの Rclone) を利用します。PostgreSQL を実行する共通の Amazon RDS インスタンスでは、チームのどのメンバーもリクエストされたすべてのレポートにアクセスできます。

チームメンバーは、Amazon S3 でバックアップされた仮想ディスクを自分のパーソナルマシンにマウントし、ワークステーションからソースバケットを更新することもできます。他のオンプレミスの内部システムに接続していれば、自分のマシンでスクリプトやその他の自動化手段を使用できる可能性があります。

セットアップは、AWS Mainframe Modernization がお客様と共有する AppStream 2.0 Windows イメージに基づいています。セットアップは、チュートリアル: Micro Focus Enterprise Analyzer と Micro Focus Enterprise Developer で使用するための AppStream 2.0 のセットアップ で説明されているように AppStream 2.0 のフリートとスタックの作成にも基づいています。

> **Important**
>
> このチュートリアルのステップは、ダウンロード可能な AWS CloudFormation テンプレート cfn-m2-appstream-fleet-ea-ed.yml を使用して AppStream 2.0 をセットアップすることを前提としています。詳細については、「チュートリアル: Micro Focus Enterprise Analyzer と Micro Focus Enterprise Developer で使用するための AppStream 2.0 のセットアップ」を参照してください。
>
> このチュートリアルのステップを実行するには、Enterprise Analyzer フリートとスタックを設定済みで、それらが実行中である必要があります。

Enterprise Analyzer の機能と成果物の詳細については、Micro Focus ウェブサイトの「Enterprise Analyzer Documentation」を参照してください。

イメージのコンテンツ

Enterprise Analyzer アプリケーション自体に加えて、イメージには次のツールとライブラリが含まれています。

サードパーティー製ツール

- Python
- Rclone
- pgAdmin
- git-scm
- PostgreSQL ODBC ドライバー

`C:\Users\Public` 内のライブラリ

- Enterprise Developer 向けの BankDemo ソースコードとプロジェクト定義: `m2-bankdemo-template.zip`。

- メインフレーム用の MFA インストールパッケージ: `mfa.zip`。詳細については、「Micro Focus Enterprise Developer ドキュメント」の「Mainframe Access Overview」を参照してください。
- Rclone のコマンドファイルおよび設定ファイル (使用方法はチュートリアルに記載されています): `m2-rclone.cmd` と `m2-rclone.conf`。

トピック

前提条件

- 分析する顧客アプリケーションのソースコードとシステム定義を S3 バケットにアップロードします。システム定義には、CICS CSD、DB2 オブジェクト定義などが含まれます。バケット内には、アプリケーションのアーティファクトをどのように整理するかに合わせたフォルダ構造を作成できます。例えば、BankDemo サンプルを解凍すると、次の構造になります。

```
demo
     |--> jcl
     |--> RDEF
     |--> transaction
     |--> xa
```

- PostgreSQL を実行している Amazon RDS インスタンスを作成して開始する このインスタンスには、Enterprise Analyzer によって生成されたデータと結果が保存されます。このインスタンスは、アプリケーションチームのすべてのメンバーと共有できます。また、`m2_ea` (またはその他の適切な名前) という名前の空のスキーマをデータベースに作成します。権限のあるユーザーがこのスキーマの項目を作成、挿入、更新、削除できるようにする認証情報を定義します。データベース名、サーバーエンドポイント URL、TCP ポートは Amazon RDS コンソールまたはアカウント管理者から取得できます。

- AWS アカウント へのプログラムによるアクセスが設定されていることを確認してください。詳細については、「Amazon Web Services 全般のリファレンス」の「プログラム的なアクセス」を参照してください。

ステップ 1: セットアップ

1. AppStream 2.0 からのウェルカムメールメッセージで受信した URL を使用して AppStream 2.0 のセッションを開始します。
2. E メールをユーザー ID として使用し、永久パスワードを定義します。
3. Enterprise Analyzer スタックを選択します。
4. AppStream 2.0 メニューページで [デスクトップ] を選択し、フリートがストリーミングしている Windows デスクトップにアクセスします。

ステップ 2: Windows 用の Amazon S3 ベースの仮想フォルダを作成する

> **Note**
>
> AWS Mainframe Modernization プレビュー中に既に Rclone を使用していた場合は、`m2-rclone.cmd` を `C:\Users\Public` にある新しいバージョンに更新する必要があります。

1. `C:\Users\Public` で提供されている `m2-rclone.conf` ファイルと `m2-rclone.cmd` ファイルを、ファイルエクスプローラーを使用してホームフォルダ `C:\Users\PhotonUser\My Files\Home Folder` にコピーします。
2. AWS アクセスキーと対応するシークレット、および AWS リージョン を使用して `m2-rclone.conf` 設定パラメータを更新します。

```
[m2-s3]
type = s3
provider = AWS
access_key_id = YOUR-ACCESS-KEY
secret_access_key = YOUR-SECRET-KEY
region = YOUR-REGION
acl = private
server_side_encryption = AES256
```

3. `m2-rclone.cmd` で、以下の変更を加えます。

- your-s3-bucket を Amazon S3 バケットの名前に変更します。例えば、m2-s3-mybucket です。
- your-s3-folder-key を Amazon S3 バケットキーに変更します。例えば、myProject です。
- your-local-folder-path を、アプリケーションファイルを含む Amazon S3 バケットから同期するディレクトリのパスに変更します。例えば、D:\PhotonUser\My Files\Home Folder\m2-new です。AppStream 2.0 がセッションの開始時と終了時に適切にバックアップおよび復元するには、この同期されたディレクトリをホームフォルダのサブディレクトリにする必要があります。

```
:loop
timeout /T 10
"C:\Program Files\rclone\rclone.exe" sync m2-s3:your-s3-bucket/your-s3-folder-
key "D:\PhotonUser\My Files\Home Folder\your-local-folder-path" --config "D:
\PhotonUser\My Files\Home Folder\m2-rclone.conf"
goto :loop
```

4. Windows コマンドプロンプトを開き、必要に応じて cd コマンドで C:\Users\PhotonUser\My Files\Home Folder に変更し、m2-rclone.cmd を実行します。このコマンドスクリプトは連続ループを実行し、Amazon S3 バケットとキーを 10 秒ごとにローカルフォルダに同期します。タイムアウトは、必要に応じて調整できます。Windows ファイルエクスプローラーの Amazon S3 バケットにあるアプリケーションのソースコードが表示されます。

作業中のセットに新しいファイルを追加したり、既存のファイルを更新したりするには、ファイルを Amazon S3 バケットにアップロードします。そうすれば、m2-rclone.cmd で定義した次のイテレーションでディレクトリに同期されます。同様に、ファイルを削除する場合は Amazon S3 バケットから削除します。次の同期オペレーションでは、ローカルディレクトリから削除されます。

ステップ 3: Amazon RDS インスタンス用の ODBC ソースを作成する

1. EA_Admin ツールを開始するには、ブラウザウィンドウの左上隅にあるアプリケーションセレクタメニューに移動し、[MF EA_Admin] を選択します。
2. [管理] メニューから、[ODBC データソース] を選択し、[ユーザー DSN] タブから [追加] を選択します。
3. [新しいデータソースを作成] ダイアログボックスで、[PostgreSQL Unicode] ドライバーを選択し、[終了] を選択します。

4. [PostgreSQL Unicode ODBC ドライバー (psqlODBC) のセットアップ] ダイアログボックスで、目的のデータソース名を定義し、書き留めておきます。以前に作成した RDS インスタンスの値を使用して、次のパラメータを入力します。

   説明

   このデータベース接続をすばやく識別するのに役立つ説明 (オプション)。

   データベース

   以前に作成した Amazon RDS データベース。

   サーバー

   Amazon RDS エンドポイント。

   ポート

   Amazon RDS ポート。

   ユーザー名

   Amazon RDS インスタンスで定義されているとおり。

   パスワード

   Amazon RDS インスタンスで定義されているとおり。

5. [テスト] を選択して Amazon RDS への接続が成功したことを確認し、[保存] を選択して新しいユーザー DSN を保存します。
6. 適切なワークスペースが作成されたことを確認するメッセージが表示されるまで待ってから、[OK] を選択して ODBC データソースを終了し、EA_Admin ツールを閉じます。
7. アプリケーションセレクタメニューに再度移動し、[Enterprise Analyzer] を選択してツールを開始します。[新規作成] を選択します。
8. ワークスペース設定ウィンドウで、ワークスペース名を入力し、その場所を定義します。ワークスペースは、この設定で作業する場合は Amazon S3 ベースのディスクにすることも、必要に応じてホームフォルダにすることもできます。
9. [他のデータベースを選択] を選択して、Amazon RDS インスタンスに接続します。
10. オプションから [Postgre] アイコンを選択し、[OK] を選択します。
11. Windows 設定の [オプション - 接続パラメータの定義] に、作成したデータソースの名前を入力します。データベース名、スキーマ名、ユーザー名、パスワードも入力します。[OK] をクリックします。

12. Enterprise Analyzer が結果を保存するために必要なテーブル、インデックスなどをすべて作成するまでお待ちください。このステップには数分かかる場合があります。Enterprise Analyzer は、データベースとワークスペースが使用できる状態になったことを確認します。
13. アプリケーションセレクタメニューに再度移動し、[Enterprise Analyzer] を選択してツールを開始します。
14. Enterprise Analyzer のスタートアップウィンドウが、新しく選択したワークスペースの場所に表示されます。[OK] を選択します。
15. 左ペインのリポジトリに移動し、リポジトリ名を選択し、[ファイル/フォルダをワークスペースに追加] を選択します。アプリケーションコードが保存されているフォルダを選択して、ワークスペースに追加します。必要であれば、以前の BankDemo サンプルコードを使用できます。Enterprise Analyzer からこれらのファイルの検証を求めるメッセージが表示されたら、[検証] を選択して最初の Enterprise Analyzer 検証レポートを開始します。アプリケーションのサイズによっては、完了するまで数分かかる場合があります。
16. ワークスペースを展開すると、ワークスペースに追加したファイルとフォルダが表示されます。オブジェクトタイプと循環的複雑度レポートは、[チャートビューアー] ペインの上部にも表示されます。

これで、必要なすべてのタスクに Enterprise Analyzer を使用できるようになりました。

以降のセッション

1. AppStream 2.0 からのウェルカムメールメッセージで受信した URL を使用して AppStream 2.0 のセッションを開始します。
2. E メールと永久パスワードを使用してログインします。
3. Enterprise Analyzer スタックを選択します。
4. このオプションを使用してワークスペースファイルを共有する場合は、`Rclone` を起動して Amazon S3 バックアップディスクに接続します。
5. Enterprise Analyzer を起動してタスクを行います。

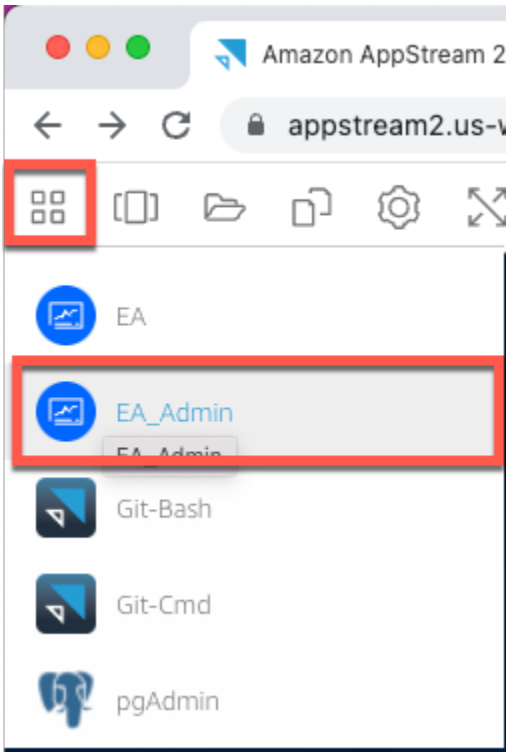
ワークスペース接続のトラブルシューティング

Enterprise Analyzer ワークスペースに再接続しようとすると、次のようなエラーが表示される場合があります。

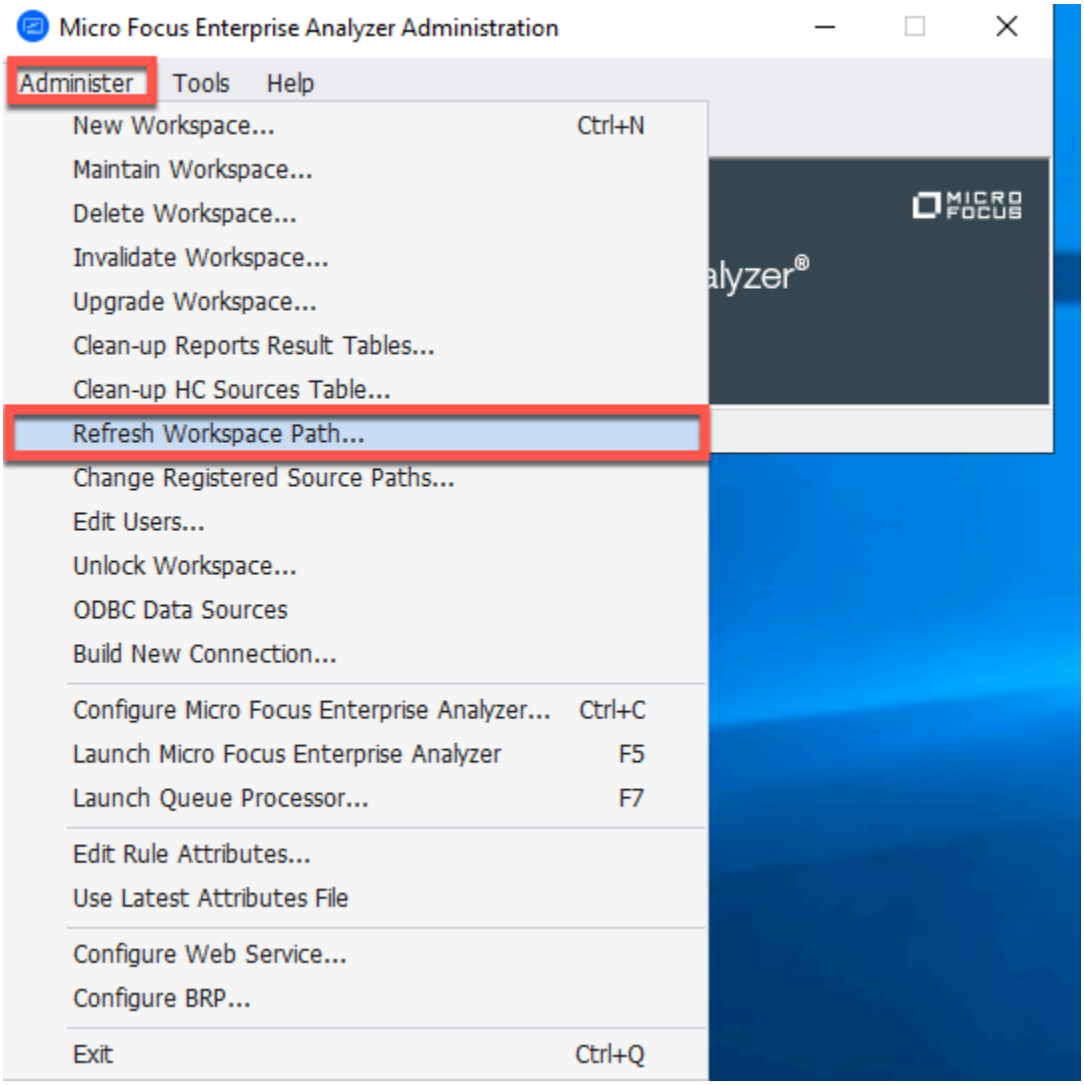
```
Cannot access the workspace directory D:\PhotonUser\My Files\Home Folder\EA_BankDemo.
 The workspace has been created on a non-shared disk of the EC2AMAZ-E6LC33H computer.
 Would you like to correct the workspace directory location?
```

この問題を解決するには、[OK] を選択してメッセージをクリアし、次の手順を実行します。

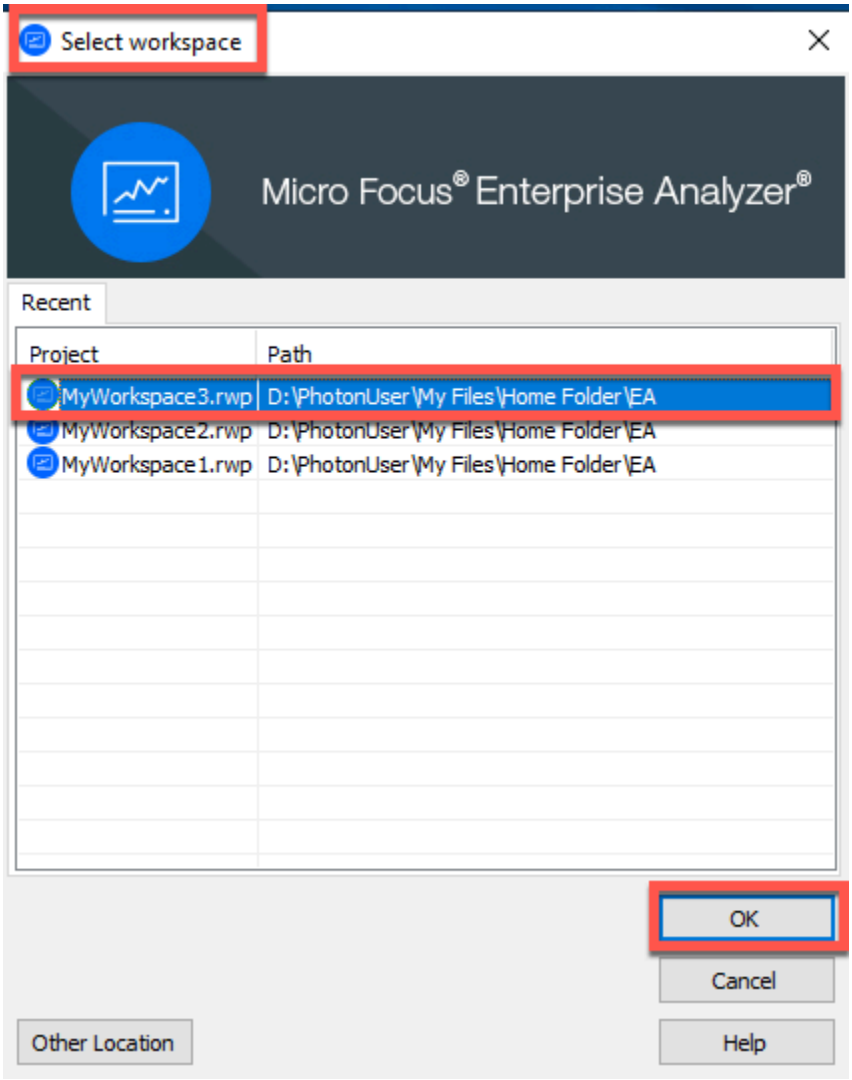
1. AppStream 2.0 で、ツールバーの [アプリケーションを起動] アイコンを選択し、次に [EA_Admin] を選択して Micro Focus Enterprise Analyzer 管理ツールを開始します。



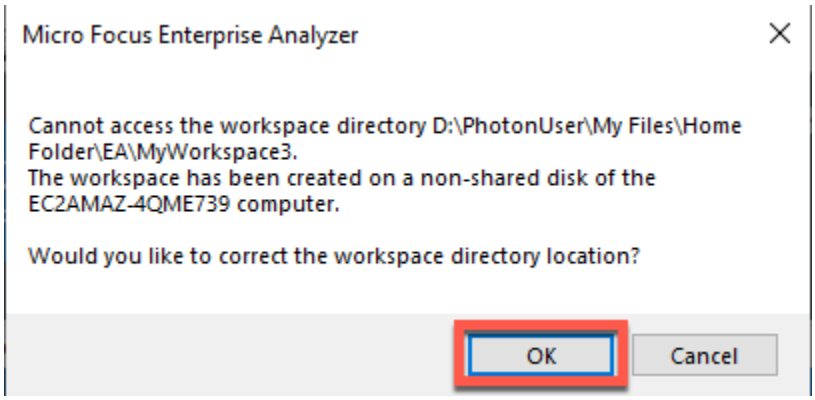
2. [管理] メニューから [ワークスペースパスを更新...] を選択します。



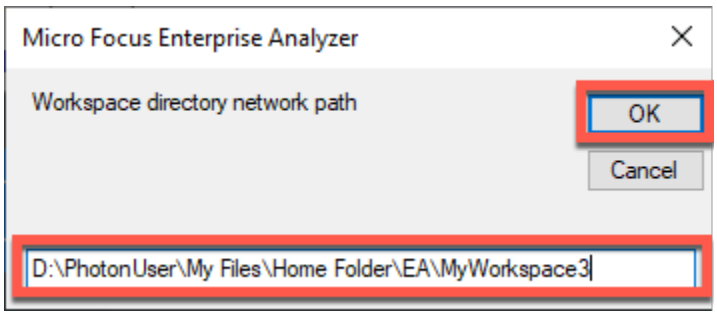
3. [ワークスペースを選択] で、目的のワークスペースを選択し、[OK] を選択します。



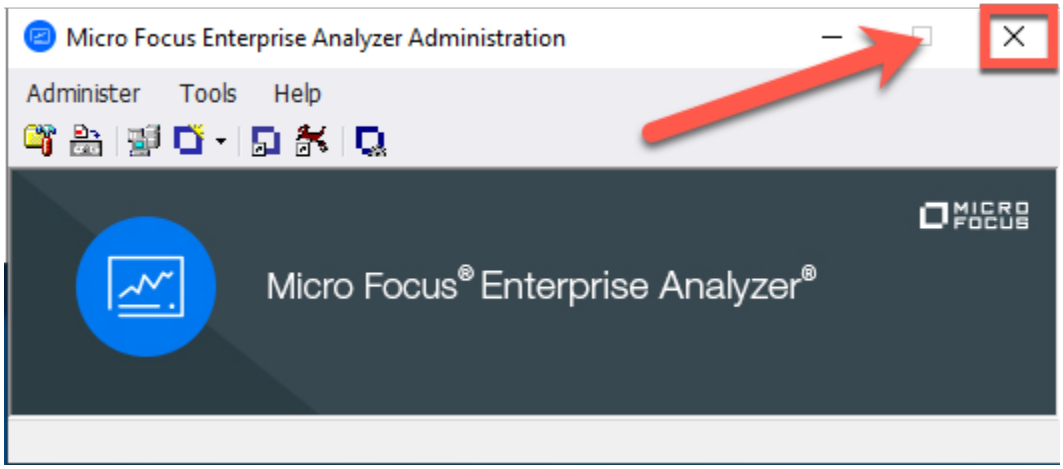
4. [OK] を選択してエラーメッセージを確認します。



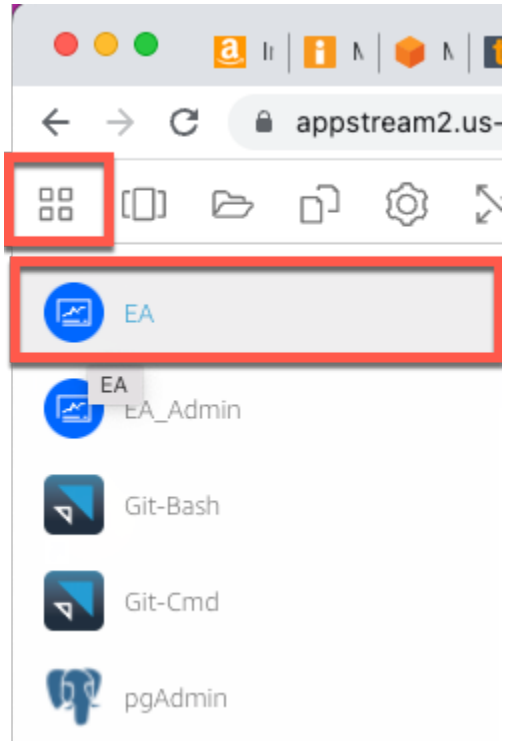
5. [ワークスペースディレクトリのネットワークパス] に、ワークスペースへの正しいパス (例: `D:\PhotonUser\My Files\Home Folder\EA\MyWorkspace3`) を入力します。



6. Micro Focus Enterprise Analyzer 管理ツールを閉じます。



7. AppStream 2.0 で、ツールバーの [アプリケーションを起動] アイコンを選択し、[EA] を選択して Micro Focus Enterprise Analyzer を開始します。



8. ステップ 3～5 を繰り返します。

これで、Micro Focus Enterprise Analyzer が既存のワークスペースで開かれるはずです。

リソースをクリーンアップする

このチュートリアル用に作成したリソースが不要になった場合は、追加料金の発生を避けるため、それらを削除してください。以下のステップを実行します。

- [EA_Admin] ツールを使用してワークスペースを削除します。
- このチュートリアル用に作成した S3 バケットを削除します。詳細については、「Amazon S3 ユーザーガイド」の「バケットの削除」を参照してください。
- このチュートリアル用に作成したデータベースを削除します。詳細については、「DB インスタンスの削除」を参照してください。

チュートリアル: AppStream 2.0 で Micro Focus Enterprise Developer をセットアップする

このチュートリアルでは、1 つ以上のメインフレームアプリケーションに Micro Focus Enterprise Developer をセットアップし、Enterprise Developer 機能を使用してそれらのアプリケーションを保守、コンパイル、テストする方法について説明します。セットアップは、AWS Mainframe Modernization がお客様と共有する AppStream 2.0 Windows イメージと、チュートリアル: Micro Focus Enterprise Analyzer と Micro Focus Enterprise Developer で使用するための AppStream 2.0 のセットアップ で説明されている AppStream 2.0 のフリートとスタックの作成に基づいています。

> **Important**
>
> このチュートリアルのステップは、ダウンロード可能な AWS CloudFormation テンプレート cfn-m2-appstream-fleet-ea-ed.yaml を使用して AppStream 2.0 をセットアップすることを前提としています。詳細については、「チュートリアル: Micro Focus Enterprise Analyzer と Micro Focus Enterprise Developer で使用するための AppStream 2.0 のセットアップ」を参照してください。
>
> このセットアップのステップは、Enterprise Developer フリートとスタックが稼働しているときに実行する必要があります。

Enterprise Developer v7 の機能と成果物の詳細については、Micro Focus サイトにある最新のオンラインドキュメント (v7.0) を参照してください。

イメージのコンテンツ

イメージには、Enterprise Developer 自体に加えて、Rumba (TN3270 エミュレータ) を含むイメージが含まれています。次のツールとライブラリも含まれています。

サードパーティー製ツール

- Python
- Rclone
- pgAdmin
- git-scm
- PostgreSQL ODBC ドライバー

`C:\Users\Public` 内のライブラリ

- Enterprise Developer 向けの BankDemo ソースコードとプロジェクト定義: `m2-bankdemo-template.zip`。
- メインフレーム用の MFA インストールパッケージ: `mfa.zip`。詳細については、「Micro Focus Enterprise Developer ドキュメント」の「Mainframe Access Overview」を参照してください。
- Rclone のコマンドファイルおよび設定ファイル (使用方法はチュートリアルに記載されています): `m2-rclone.cmd` と `m2-rclone.conf`。

CodeCommit リポジトリにまだロードされていないが、Amazon S3 バケットにあるソースコードにアクセスする必要がある場合 (例えば、ソースコードを Git に最初にロードする場合など) は、チュートリアル: AppStream 2.0 での Enterprise Analyzer のセットアップ で説明されている手順に従って仮想 Windows ディスクを作成します。

トピック

- 前提条件
- ステップ 1: 個々の Enterprise Developer ユーザーによる設定
- ステップ 2: Windows で Amazon S3 ベースの仮想フォルダを作成する (オプション)
- ステップ 3: リポジトリのクローンを作成する

- 以降のセッション
- リソースをクリーンアップする

前提条件

- 保守対象のアプリケーションのソースコードがロードされた 1 つ以上の CodeCommit リポジトリ。両方のツールを組み合わせて相乗効果を生み出すには、リポジトリの設定が上記の CI/CD パイプラインの要件と一致している必要があります。
- 各ユーザーは、「AWS CodeCommit の認証とアクセスコントロール」の情報に従って、アカウント管理者が定義した CodeCommit リポジトリまたはリポジトリへの認証情報を持っている必要があります。これらの認証情報の構造については、「AWS CodeCommit の認証とアクセスコントロール」を参照してください。CodeCommit の IAM 承認に関する完全なリファレンスは、CodeCommit アクセス許可リファレンスにあります。管理者は、各リポジトリのロールに固有の認証情報を持ち、ユーザーの承認を特定のリポジトリで実行する必要がある特定のタスクセットに限定して、個別のロールに対して個別の IAM ポリシーを定義できます。そのため、アカウント管理者は CodeCommit リポジトリ管理者ごとにプライマリユーザーを生成し、適切な IAM ポリシーまたは CodeCommit アクセス用のポリシーを選択して、必要なリポジトリにアクセスするアクセス許可をこのユーザーに付与します。

ステップ 1: 個々の Enterprise Developer ユーザーによる設定

1. IAM 認証情報の取得:
    1. AWS コンソール (https://console.aws.amazon.com/iam/) に接続します。
    2. AWS CodeCommit ユーザーガイドの「Git 認証情報を使用する HTTPS ユーザーのセットアップ」のステップ 3 で説明されている手順に従います。
    3. IAM が生成した CodeCommit 固有のサインイン認証情報をコピーするには、この情報を表示して、ローカルコンピュータ上の安全なファイルにコピーアンドペーストするか、[認証情報のダウンロード] を選択して .CSV ファイルとしてこの情報をダウンロードします。CodeCommit に接続するには、この情報が必要です。
2. ウェルカムメールで受信した URL に基づいて、AppStream 2.0 でセッションを開始します。E メールをユーザー名として使用し、パスワードを作成します。
3. Enterprise Developer スタックを選択します。
4. メニューページで、左側のナビゲーションで [デスクトップ] を選択すると、フリートによってストリーミングされている Windows デスクトップに移動します。

ステップ 2: Windows で Amazon S3 ベースの仮想フォルダを作成する (オプション)

Rclone (上を参照) が必要な場合は、Windows に Amazon S3 ベースの仮想フォルダを作成します (すべてのアプリケーションアーティファクトが CodeCommit アクセスからのみ取得される場合はオプション)。

> **Note**
>
> AWS Mainframe Modernization プレビュー中に既に Rclone を使用していた場合は、`m2-rclone.cmd` を `C:\Users\Public` にある新しいバージョンに更新する必要があります。

1. `C:\Users\Public` で提供されている `m2-rclone.conf` ファイルと `m2-rclone.cmd` ファイルを、ファイルエクスプローラーを使用してホームフォルダ `C:\Users\PhotonUser\My Files\Home Folder` にコピーします。
2. AWS アクセスキーと対応するシークレット、および AWS リージョン を使用して `m2-rclone.conf` 設定パラメータを更新します。

```
[m2-s3]
type = s3
provider = AWS
access_key_id = YOUR-ACCESS-KEY
secret_access_key = YOUR-SECRET-KEY
region = YOUR-REGION
acl = private
server_side_encryption = AES256
```

3. `m2-rclone.cmd` で、以下の変更を加えます。

   - `your-s3-bucket` を Amazon S3 バケットの名前に変更します。例えば、`m2-s3-mybucket` です。
   - `your-s3-folder-key` を Amazon S3 バケットキーに変更します。例えば、`myProject` です。
   - `your-local-folder-path` を、アプリケーションファイルを含む Amazon S3 バケットから同期するディレクトリのパスに変更します。例えば、`D:\PhotonUser\My Files\Home Folder\m2-new` です。AppStream 2.0 がセッションの開始時と終了時に適切にバックアッ

プおよび復元するには、この同期されたディレクトリをホームフォルダのサブディレクトリにする必要があります。

```
:loop
timeout /T 10
"C:\Program Files\rclone\rclone.exe" sync m2-s3:your-s3-bucket/your-s3-folder-
key "D:\PhotonUser\My Files\Home Folder\your-local-folder-path" --config "D:
\PhotonUser\My Files\Home Folder\m2-rclone.conf"
goto :loop
```

4. Windows コマンドプロンプトを開き、必要に応じて cd コマンドで `C:\Users\PhotonUser\My Files\Home Folder` に変更し、`m2-rclone.cmd` を実行します。このコマンドスクリプトは連続ループを実行し、Amazon S3 バケットとキーを 10 秒ごとにローカルフォルダに同期します。タイムアウトは、必要に応じて調整できます。Windows ファイルエクスプローラーの Amazon S3 バケットにあるアプリケーションのソースコードが表示されます。

作業中のセットに新しいファイルを追加したり、既存のファイルを更新したりするには、ファイルを Amazon S3 バケットにアップロードします。そうすれば、`m2-rclone.cmd` で定義した次のイテレーションでディレクトリに同期されます。同様に、ファイルを削除する場合は Amazon S3 バケットから削除します。次の同期オペレーションでは、ローカルディレクトリから削除されます。

ステップ 3: リポジトリのクローンを作成する

1. ブラウザウィンドウの左上隅にあるアプリケーションセレクタメニューに移動し、[Enterprise Developer] を選択します。
2. ワークスペースの場所として `C:\Users\PhotonUser\My Files\Home Folder` (別名`D:\PhotonUser\My Files\Home Folder`) を選択して、Enterprise Developer が必要とするワークスペースの作成をホームフォルダで完了します。
3. Enterprise Developer で、プロジェクトエクスプローラーに移動し、右クリックして [インポート]、[インポート...]、[Git]、[Git] [クローン URI] からの [プロジェクト] を選択して CodeCommit リポジトリのクローンを作成します。次に、CodeCommit 固有のサインイン認証情報を入力し、Eclipse ダイアログに入力してコードをインポートします。

これで、CodeCommit の git リポジトリがローカルワークスペースに複製されました。

これで、Enterprise Developer ワークスペースでアプリケーションのメンテナンス作業を開始する準備ができました。特に、Enterprise Developer と統合された Microfocus Enterprise Server (ES) の

ローカルインスタンスを使用して、アプリケーションをインタラクティブにデバッグおよび実行し、変更をローカルで検証できます。

> **Note**
>
> ローカルのエンタープライズサーバーインスタンスを含むローカルの Enterprise Developer 環境は Windows で実行され、AWS Mainframe Modernization は Linux で実行されます。新しいアプリケーションを CodeCommit にコミットしてこのターゲット用に再構築した後、新しいアプリケーションを本番環境にロールアウトする前に、AWS Mainframe Modernization が提供する Linux 環境で補完的なテストを実行することをお勧めします。

以降のセッション

CodeCommit リポジトリのクローニング用のホームフォルダなど、AppStream 2.0 の管理下にあるフォルダを選択すると、そのフォルダはセッション間で透過的に保存および復元されます。次回、アプリケーションを操作する必要が生じたときには、次のステップを実行してください。

1. ウェルカムメールで受信した URL に基づいて、AppStream 2.0 でセッションを開始します。
2. E メールと永久パスワードを使用してログインします。
3. Enterprise Developer スタックを選択します。
4. このオプションを使用してワークスペースファイルを共有する場合は、`Rclone` を開始して Amazon S3 ベースのディスクに接続します (上を参照)。
5. Enterprise Developer を起動して作業を行います。

リソースをクリーンアップする

このチュートリアルで作成したリソースが不要になった場合は、追加料金の発生を避けるため、それらを削除してください。以下のステップを実行します。

- このチュートリアル用に作成した CodeCommit リポジトリを削除します。詳細については、「AWS CodeCommit ユーザーガイド」の「CodeCommit リポジトリを削除する」を参照してください。
- このチュートリアル用に作成したデータベースを削除します。詳細については、「DB インスタンスの削除」を参照してください。

Micro Focus Enterprise Analyzer と Micro Focus Enterprise Developer ストリーミングセッションのオートメーションをセットアップする

セッションの開始時と終了時にスクリプトを自動的に実行し、お客様の状況に合わせた自動化を実現できます。この AppStream 2.0 機能の詳細については、「Amazon AppStream 2.0 管理ガイド」の「セッションスクリプトを使用して AppStream 2.0 ユーザーのストリーミングエクスペリエンスを管理する」を参照してください。

この機能には、少なくとも以下のバージョンの Enterprise Analyzer イメージと Enterprise Developer イメージが必要です。

- `m2-enterprise-analyzer-v8.0.4.R1`
- `m2-enterprise-developer-v8.0.4.R1`

トピック

セッション開始時にオートメーションをセットアップする

ユーザーが AppStream 2.0 に接続したときにオートメーションスクリプトを実行する場合は、スクリプトを作成して `m2-user-setup.cmd` という名前を付けます。ユーザーの AppStream 2.0 ホームフォルダにスクリプトを保存します。AWS Mainframe Modernization が提供する AppStream 2.0 イメージは、その場所でその名前のスクリプトを探し、存在する場合はそれを実行します。

> **Note**
>
> スクリプトの実行時間は、AppStream 2.0 で設定されている制限 (現在は 60 秒) を超えることはできません。詳細については、「Amazon AppStream 2.0 管理ガイド」の「ストリーミングセッションの開始前にスクリプトを実行する」を参照してください。

セッション終了時にオートメーションをセットアップする

ユーザーが AppStream 2.0 から切断したときにオートメーションスクリプトを実行する場合は、スクリプトを作成して `m2-user-teardown.cmd` という名前を付けます。ユーザーの AppStream 2.0 ホームフォルダにスクリプトを保存します。AWS Mainframe Modernization が提供する AppStream 2.0 イメージは、その場所でその名前のスクリプトを探し、存在する場合はそれを実行します。

> **Note**
>
> スクリプトの実行時間は、AppStream 2.0 で設定されている制限 (現在は 60 秒) を超えることはできません。詳細については、「Amazon AppStream 2.0 管理ガイド」の「ストリーミングセッションの終了後にスクリプトを実行する」を参照してください。

Enterprise Developer でデータセットをテーブルと列として表示する

Micro Focus ランタイムを使用して、AWS Mainframe Modernization にデプロイされたメインフレームデータセットにアクセスできます。移行したデータセットは、Micro Focus Enterprise Developer インスタンスからテーブルや列として表示できます。この方法でデータセットを表示すると、次のことが可能になります。

- 移行したデータファイルに対して SQL SELECT オペレーションを実行します。
- アプリケーションを変更せずに、移行したメインフレームアプリケーションの外部にデータを公開します。
- データを簡単にフィルタリングし、CSV またはその他のファイル形式で保存できます。

> **Note**
>
> ステップ 1 と 2 は 1 回限りのアクティビティです。データセットごとにステップ 3 と 4 を繰り返して、データベースビューを作成します。

トピック

- 前提条件

前提条件

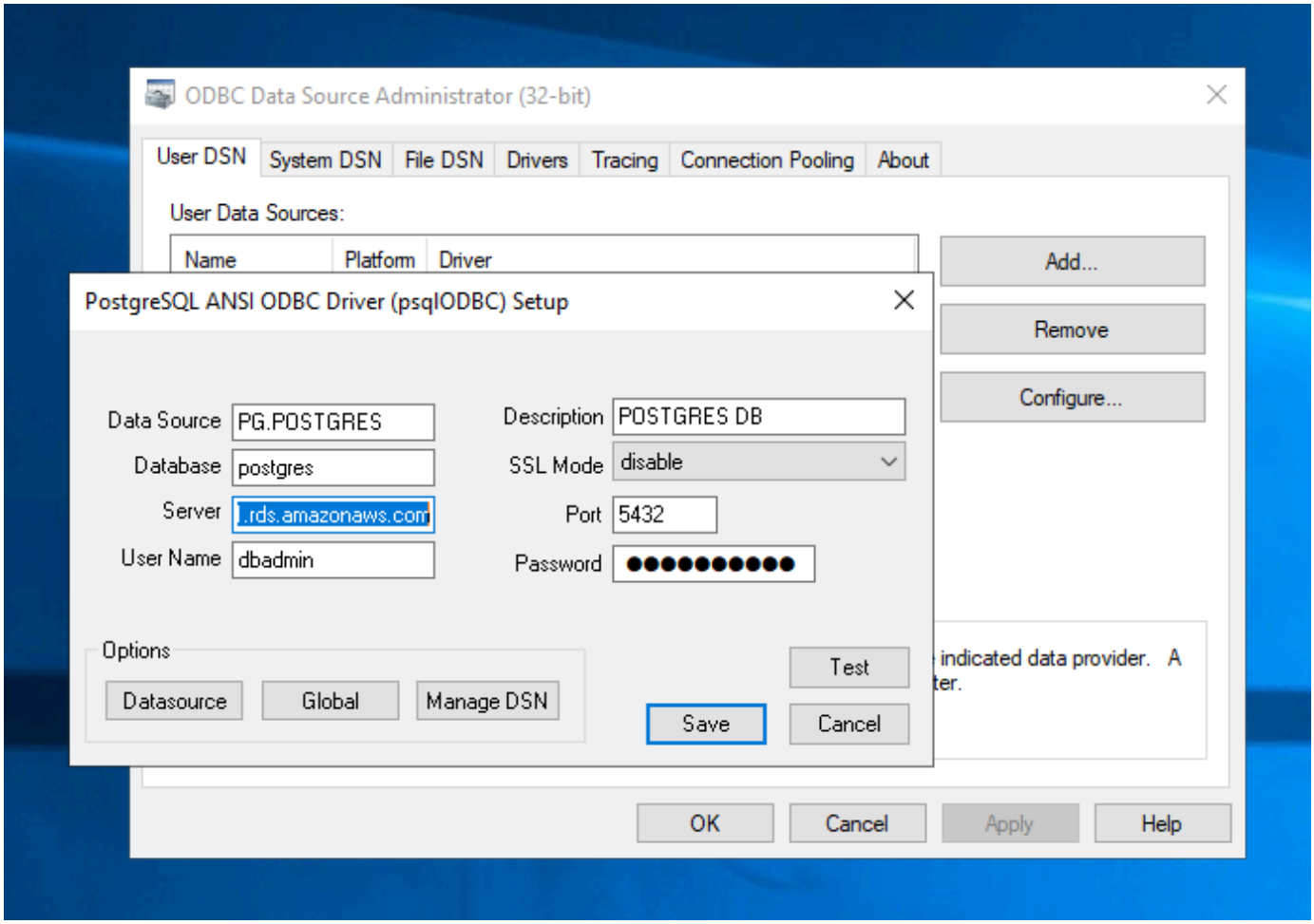
- AppStream 2.0 経由で Micro Focus Enterprise Developer Desktop にアクセスできる必要があります。
- Micro Focus ランタイムエンジンを使用して、AWS Mainframe Modernization でアプリケーションをデプロイし、実行する必要があります。
- Aurora PostgreSQL 互換エディションにアプリケーションデータを保存しています。

ステップ 1: Micro Focus データストア (Amazon RDS データベース) への ODBC 接続をセットアップする

このステップでは、表示するデータをテーブルや列として格納しているデータベースへの ODBC 接続をセットアップします。これは 1 回限りのステップです。

1. AppStream 2.0 ストリーミング URL を使用して、Micro Focus Enterprise Developer Desktop にログインします。
2. [ODBC データソースの管理者] を開き、[ユーザー DSN] を選択し、[追加] を選択します。
3. [新しいデータソースを作成] で [PostgreSQL ANSI] を選択し、[完了] を選択します。
4. 以下のように必要なデータベース情報を指定して、`PG.POSTGRES` のデータソースを作成します。

```
Data Source : PG.POSTGRES
Database    : postgres
Server      : rds_endpoint.rds.amazonaws.com
Port        : 5432
User Name   : user_name
Password    : user_password
```



5. [テスト] を選択し、接続が機能することを確認します。テストが成功すると、メッセージ `Connection successful` が表示されます。

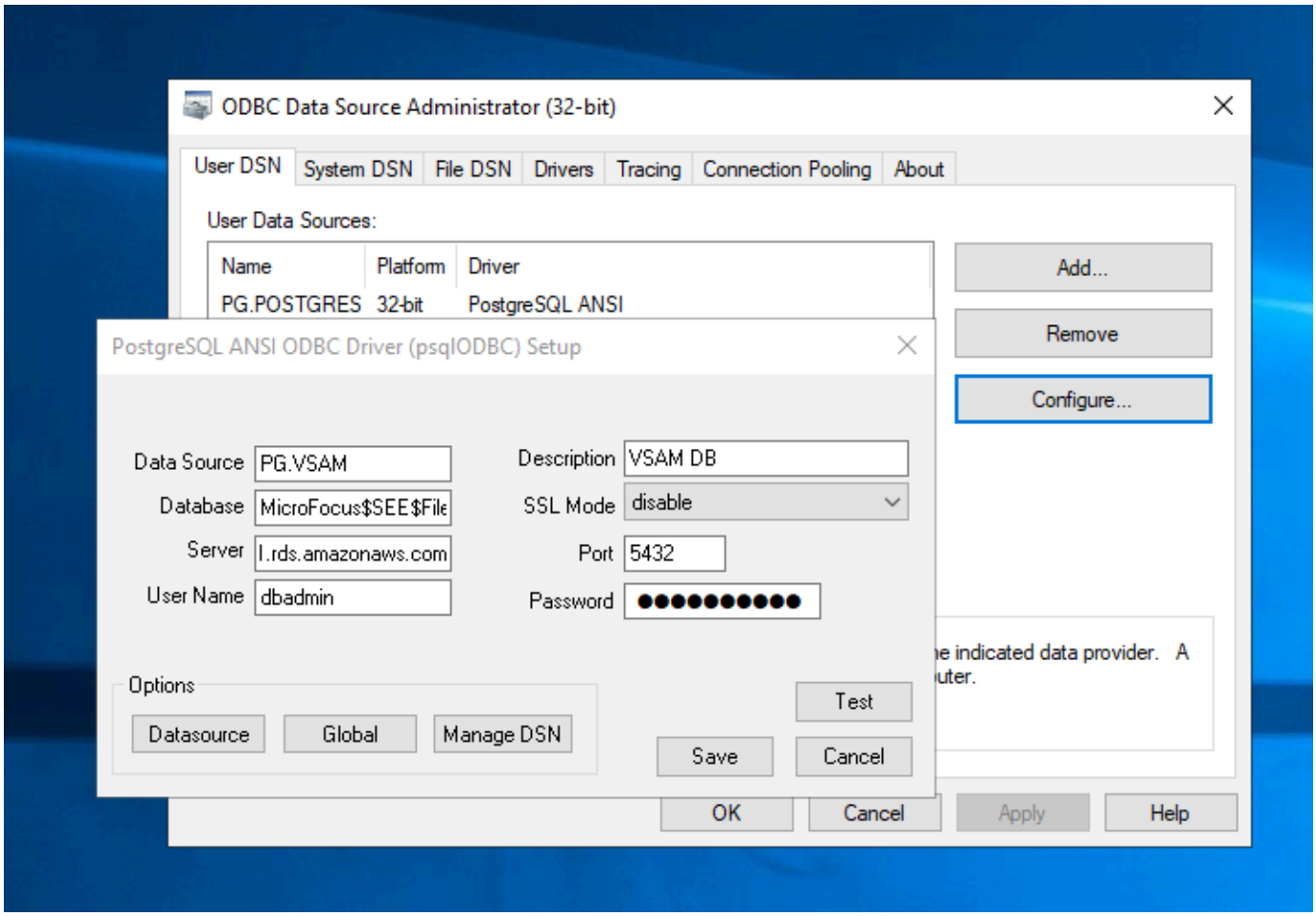
   テストが成功しなかった場合は、次の情報を確認してください。

   - Amazon RDS のトラブルシューティング
   - Amazon RDS DB インスタンスに接続するときの問題を解決するにはどうすればよいですか?

6. データソースを保存します。
7. `PG.VSAM` のデータソースを作成し、接続をテストして、データソースを保存します。必要に応じて、次のデータベース情報を入力します:

```
Data Source : PG.VSAM
Database    : MicroFocus$SEE$Files$VSAM
Server      : rds_endpoint.rds.amazonaws.com
Port        : 5432
User Name   : user_name
```

```
Password     : user_password
```



ステップ 2: MFDBFH.cfg ファイルを作成する

このステップでは、Micro Focus データストアを記述する設定ファイルを作成します。これは 1 回限りの設定ステップです。

1. 例えば、D:\PhotonUser\My Files\Home Folder\MFED\cfg\MFDBFH.cfg のホームフォルダに、次の内容の MFDBFH.cfg ファイルを作成します。

```
<datastores>
       <server name="ESPACDatabase" type="postgresql" access="odbc">
        <dsn name="PG.POSTGRES" type="database" dbname="postgres"/>
        <dsn name="PG.VSAM" type="datastore" dsname="VSAM"/>
       </server>
      </datastores>
```

2. 以下のコマンドを実行して Micro Focus データストアをクエリし、MFDBFH の設定を確認します。

```
*###*
*## Test the connection by running the following commands*
*###*

set MFDBFH_CONFIG="D:\PhotonUser\My Files\Home Folder\MFED\cfg\MFDBFH.cfg"

dbfhdeploy list sql://ESPACDatabase/VSAM?folder=/DATA
```

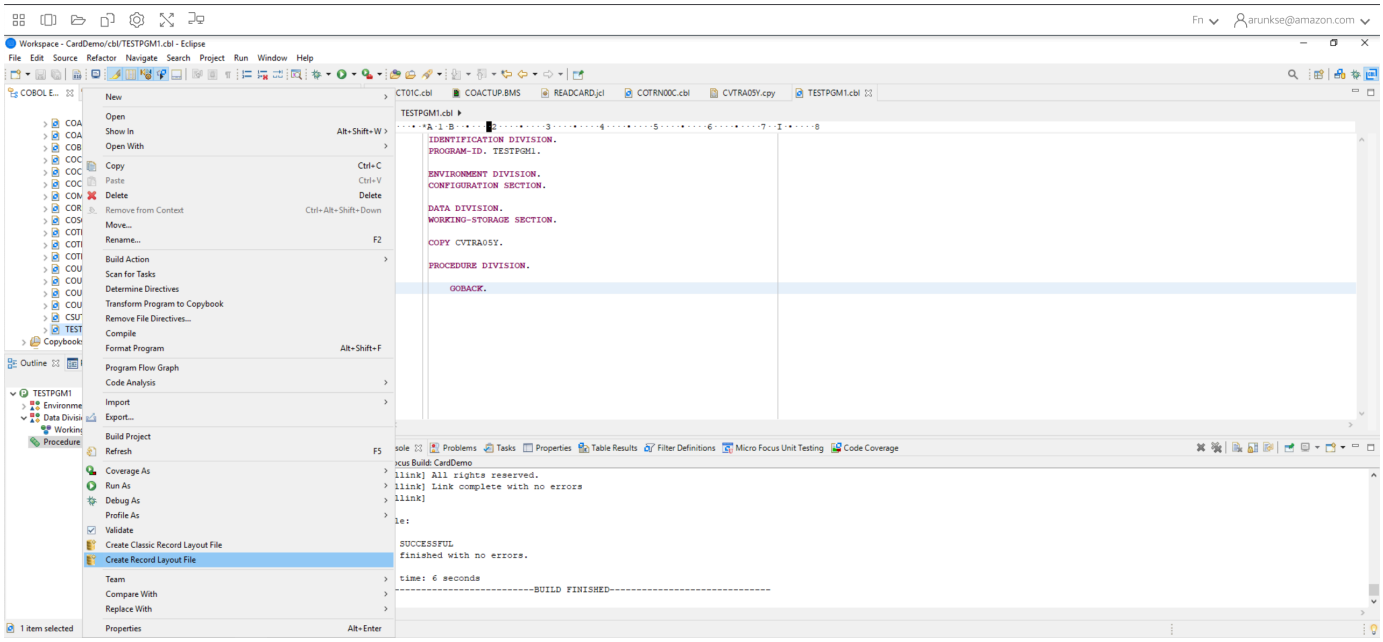
ステップ 3: コピーブックレイアウト用の構造 (STR) ファイルを作成する

このステップでは、コピーブックレイアウト用の構造ファイルを作成し、後でデータセットからデータベースビューを作成する際に使用できるようにします。

1. コピーブックに関連するプログラムをコンパイルします。コピーブックを使用するプログラムがない場合は、コピーブック用の COPY ステートメントを使用して、次のような簡単なプログラムを作成してコンパイルします。

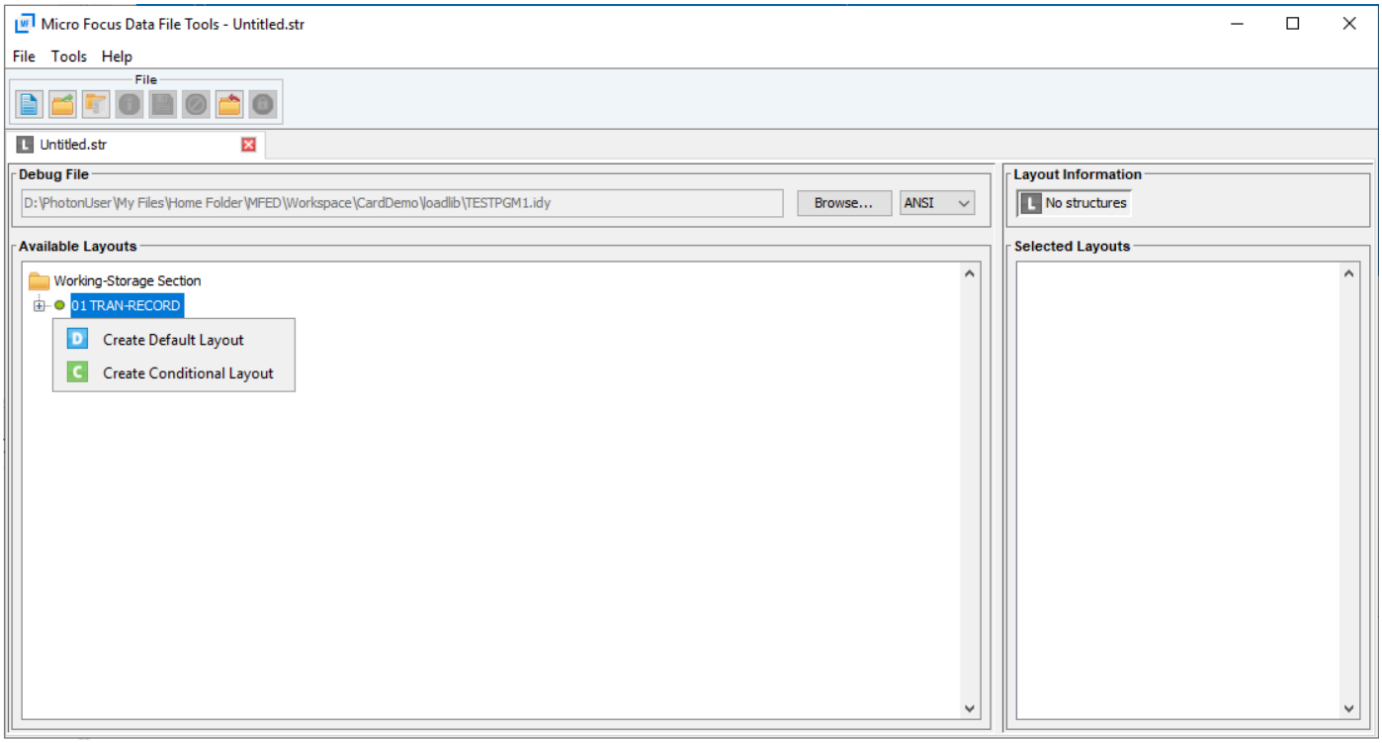
```
IDENTIFICATION DIVISION.
      PROGRAM-ID. TESTPGM1.

      ENVIRONMENT DIVISION.
      CONFIGURATION SECTION.

      DATA DIVISION.
      WORKING-STORAGE SECTION.

      COPY CVTRA05Y.

      PROCEDURE DIVISION.

      GOBACK.
```

2. コンパイルが成功したら、プログラムを右クリックし、[レコードレイアウトファイルの作成] を選択します。これにより、コンパイル中に生成された.idy ファイルを使用して Micro Focus データファイルツールが開きます。

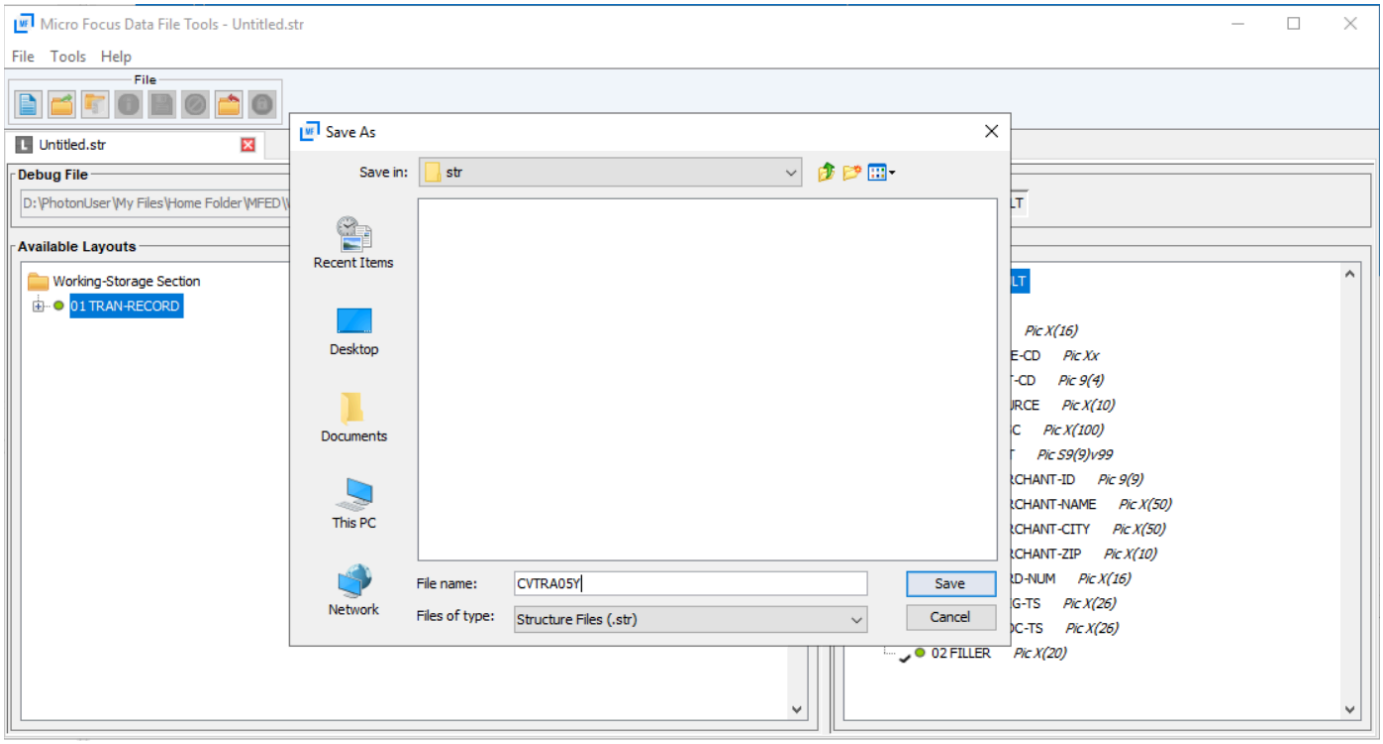


3. レコード構造を右クリックし、レイアウトに応じて [デフォルトレイアウトを作成] (単一構造) または [条件付きレイアウトを作成] (複数構造) を選択します。

   詳細については、Micro Focus ドキュメンテーションの「構造ファイルとレイアウトの作成」を参照してください。



4. レイアウトを作成したら、メニューから [ファイル] を選択し、[名前を付けて保存] を選択します。ホームフォルダにあるファイルを参照し、コピーブックと同じ名前で保存します。str という名前のフォルダを作成し、そこにすべての構造ファイルを保存することができます。



ステップ 4: 構造 (STR) ファイルを使用してデータベースビューを作成する

このステップでは、以前に作成した構造ファイルを使用して、データセットのデータベースビューを作成します。

- 以下の例に示すように、dbfhview コマンドを使用して Micro Focus データストアに既に存在するデータセットのデータベースビューを作成します。

```
##
      ## The below command creates database view for VSAM file
 AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
      ## using the STR file CVTRA05Y.str
      ##

      dbfhview -create -struct:"D:\PhotonUser\My Files\Home Folder\MFED\str
\CVTRA05Y.str" -name:V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT -file:sql://
ESPACDatabase/VSAM/AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT?folder=/DATA

      ##
```

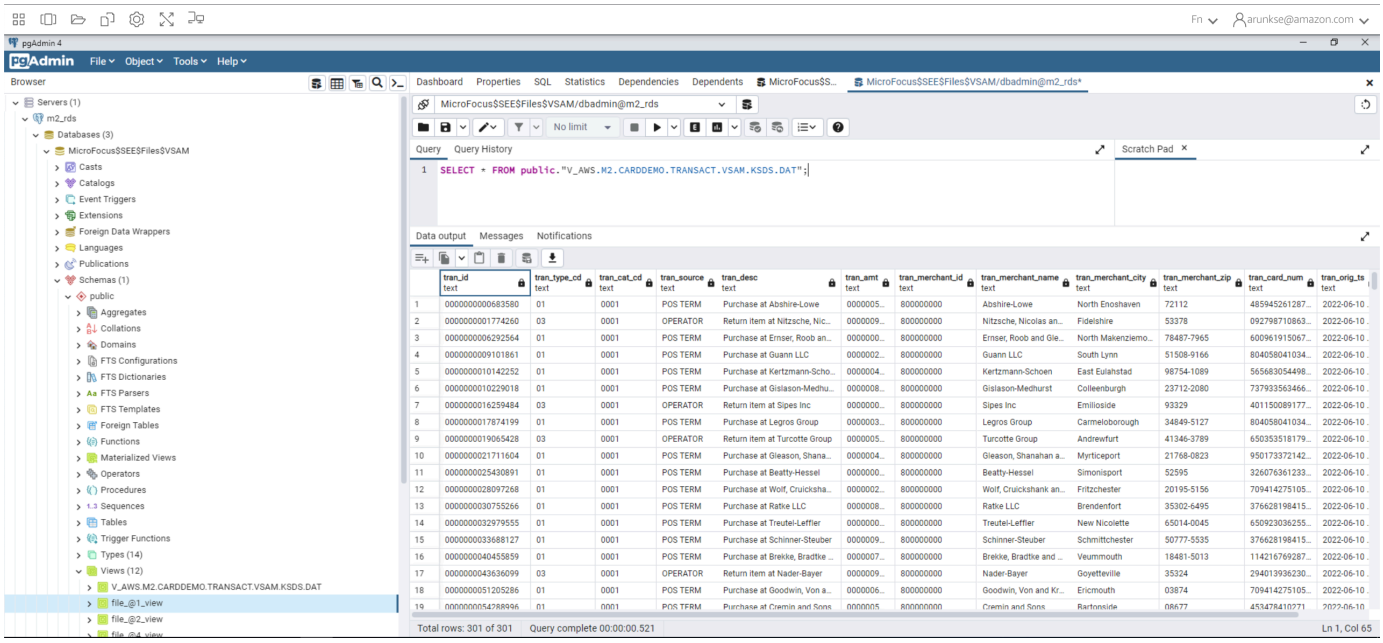
```
      ## Output:
      ##

      Micro Focus Database File Handler - View Generation Tool Version 8.0.00
      Copyright (C) 1984-2022 Micro Focus. All rights reserved.

      VGN0017I Using structure definition 'TRAN-RECORD-DEFAULT'
      VGN0022I View 'V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT' installed in
datastore 'sql://espacdatabase/VSAM'
      VGN0002I The operation completed successfully
```

ステップ 5: Micro Focus データセットをテーブルと列として表示する

このステップでは、pgAdmin を使用してデータベースに接続し、クエリを実行してテーブルや列などのデータセットを表示できます。

- pgAdmin を使用してデータベース MicroFocusSEEFiles$VSAM に接続し、ステップ 4 で作成したデータベースビューをクエリします。

```
SELECT * FROM public."V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT";
```



チュートリアル: Micro Focus Enterprise Developer でのテンプレートの使用

このチュートリアルでは、Micro Focus Enterprise Developer でテンプレートと事前定義済みプロジェクトを使用する方法について説明します。3 つのユースケースについて説明します。すべてのユースケースでは、BankDemo サンプルで提供されているサンプルコードを使用しています。サンプルをダウンロードするには、`bankdemo.zip` を選択します。

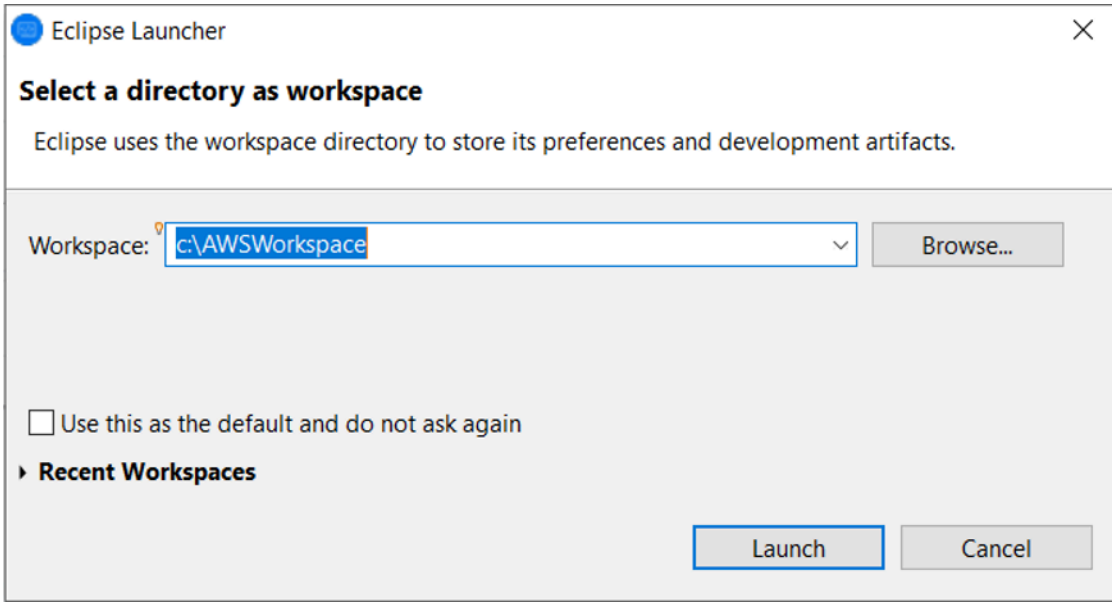
> **Important**
>
> Enterprise Developer for Windows を使用している場合、コンパイラによって生成されたバイナリは、Enterprise Developer が提供する Enterprise Server でのみ実行できます。Linux ベースの AWS Mainframe Modernization ランタイムでは実行できません。

トピック

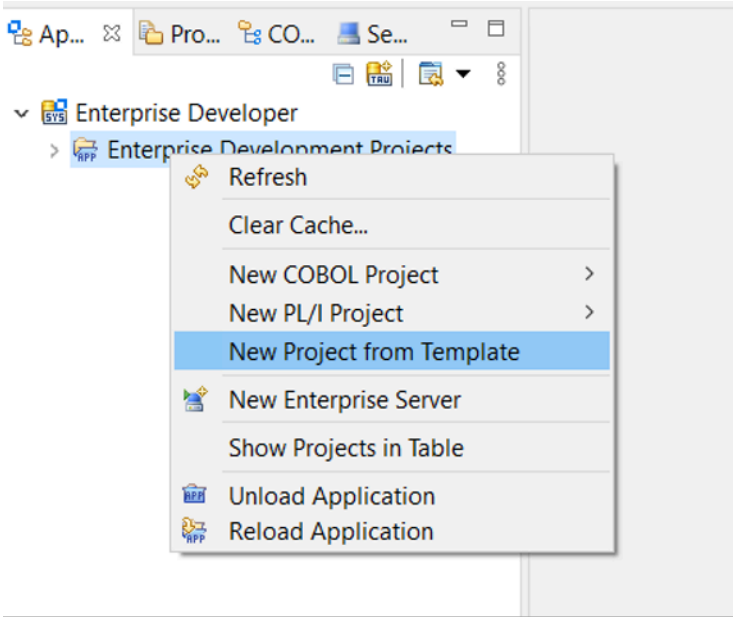
ユースケース 1 - ソースコンポーネントを含む COBOL プロジェクトテンプレートの使用

このユースケースでは、デモの事前セットアップ手順の一環として、ソースコンポーネントをテンプレートのディレクトリ構造にコピーする必要があります。`bankdemo.zip` では、ソースのコピーを 2 つ必要としないように、元の配布物 `AWSTemplates.zip` から変更されている箇所があります。

1. Enterprise Developer を起動し、選択済みのワークスペースを指定します。

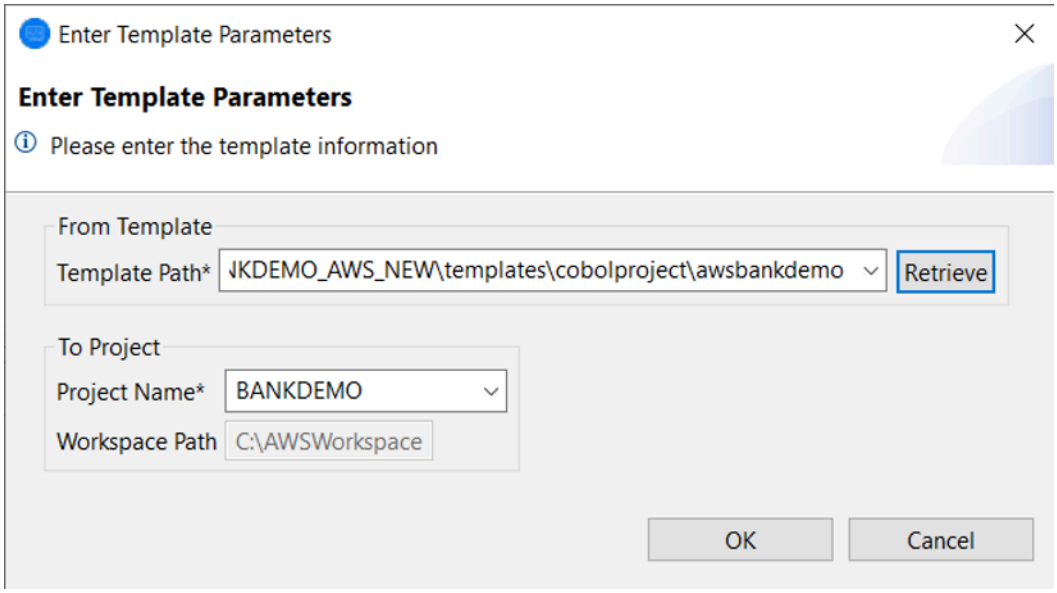


2. アプリケーションエクスプローラービュー の [Enterprise Development プロジェクト] ツリービュー項目で、コンテキストメニューから [テンプレートから新規プロジェクト] を選択します。

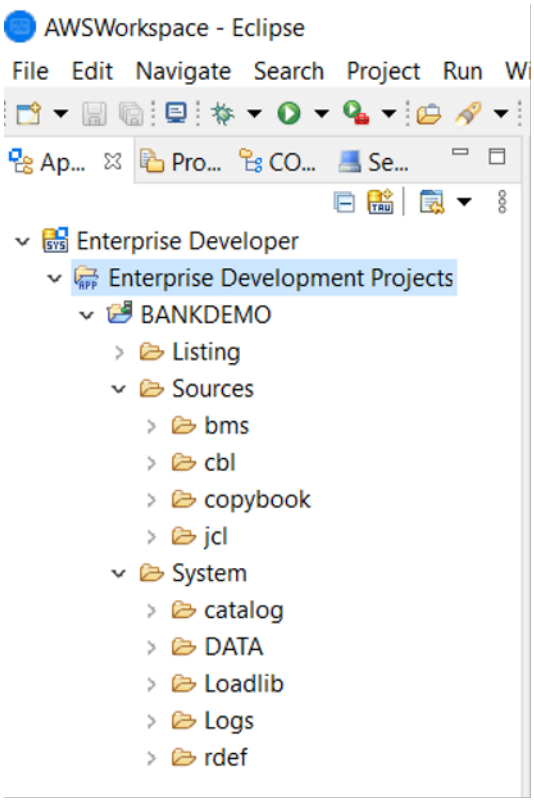


3. 表示されているとおりに、テンプレートのパラメータを入力してください。

> **Note**
>
> テンプレートパスは ZIP が抽出された場所を参照します。



4. [OK] を選択すると、提供されたテンプレートに基づき、ソースと実行環境構造が完全な、ローカル開発用 Eclipse プロジェクトが作成されます。



この System 構造には、BANKDEMO に必要なエントリを含む完全なリソース定義ファイル、エントリが追加された必須カタログ、対応する ASCII データファイルが含まれています。

ソーステンプレート構造にはすべてのソース項目が含まれているため、これらのファイルはローカルプロジェクトにコピーされ、Enterprise Developer で自動的に構築されます。

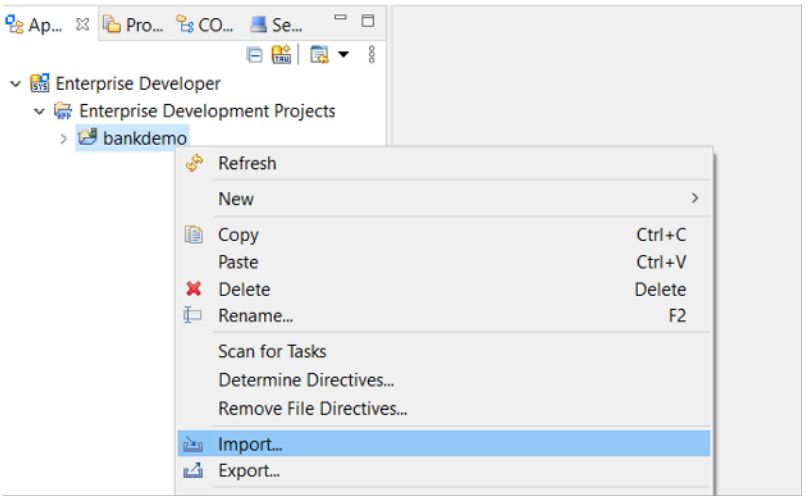
ユースケース 2 - ソースコンポーネントなしで COBOL プロジェクトテンプレートを使用

ステップ 1 から 3 は ユースケース 1 - ソースコンポーネントを含む COBOL プロジェクトテンプレートの使用 と同様です。

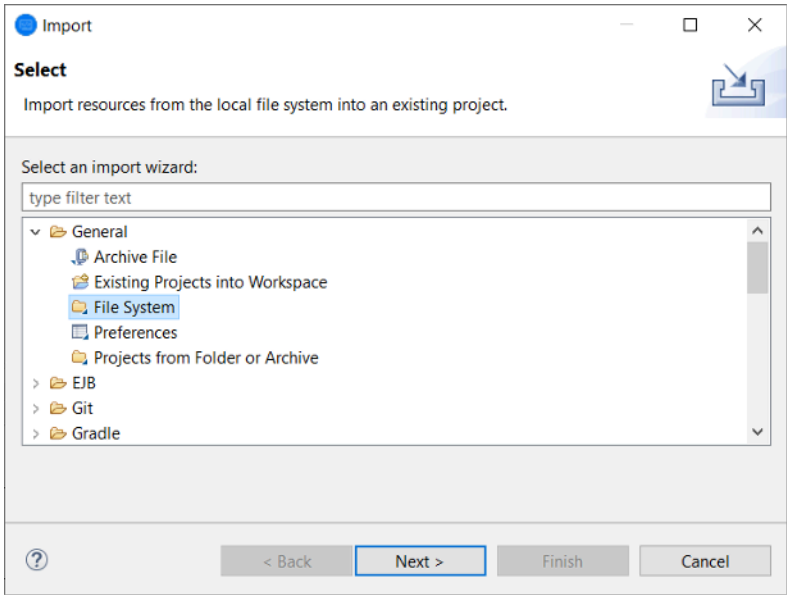
このユースケースにおける System 構造にも、BankDemo に必要なエントリを含む完全なリソース定義ファイル、エントリが追加された必須カタログ、対応する ASCII データファイルが含まれています。

ただし、テンプレートソース構造にはコンポーネントは含まれていません。使用しているソースリポジトリからこれらのコンポーネントをプロジェクトにインポートする必要があります。

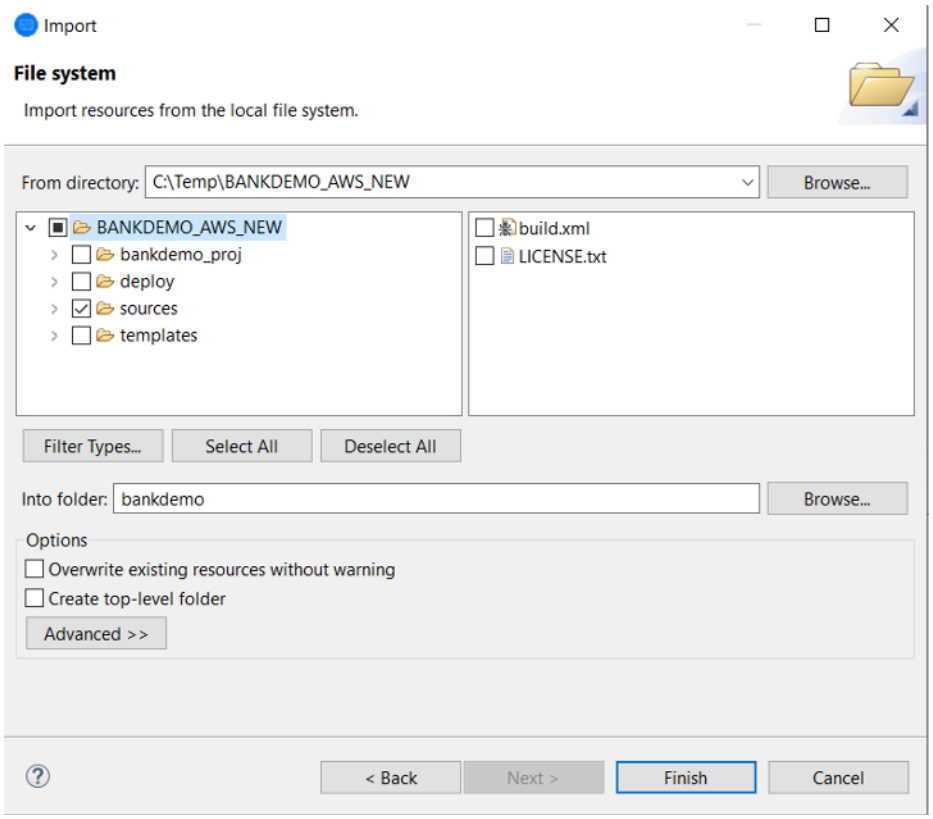
1. プロジェクトの名前を選択します。関連するコンテキストメニューから [インポート] を選択します。



2. ダイアログが表示されたら、[一般] セクションで [ファイルシステム] を選択し、[次へ] を選択します。



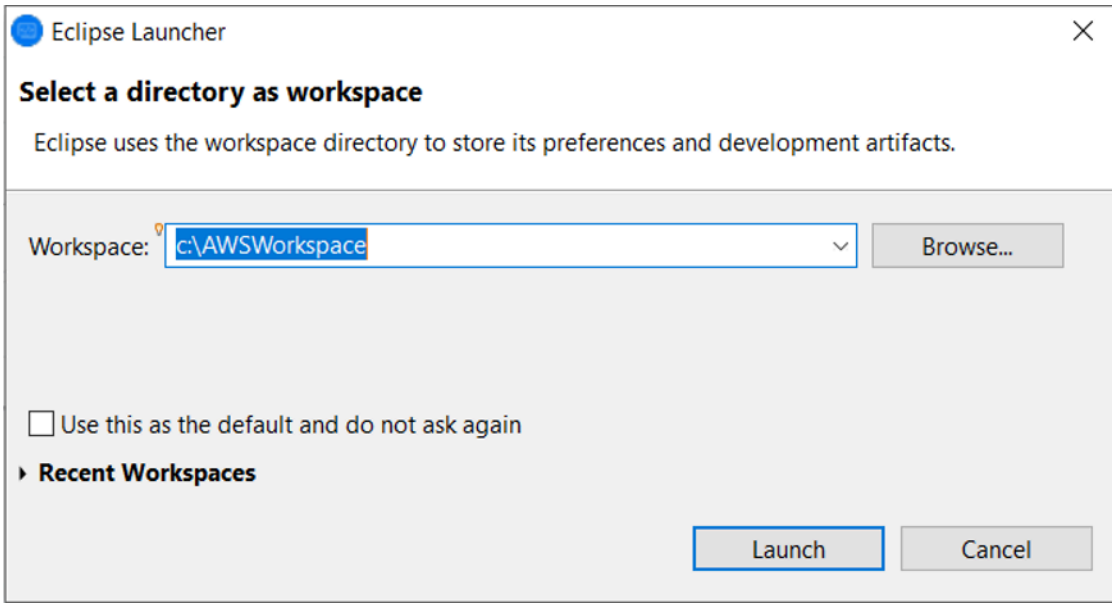
3. ファイルシステムを参照した状態でリポジトリフォルダを指定して、[差出人ディレクトリ] フィールドに入力します。sources などのインポート先フォルダをすべて選択します。Into folder フィールドは、事前入力されています。[終了] を選択します。



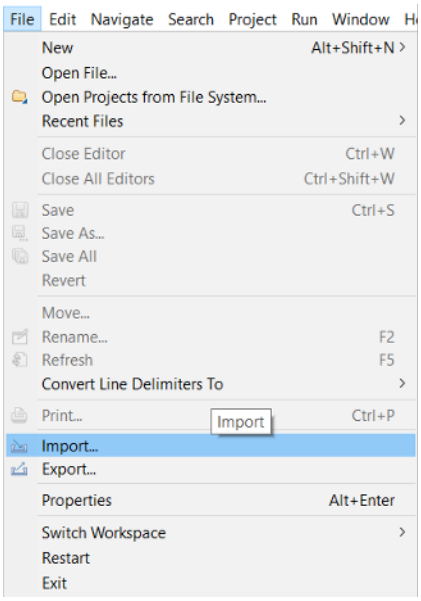
ソーステンプレート構造にすべてのソーステンプレートが含まれた後、これらの項目は Enterprise Developer で自動的に構築されます。

ユースケース 3 - ソースフォルダにリンクする事前定義済みの COBOL プロジェクトを使用

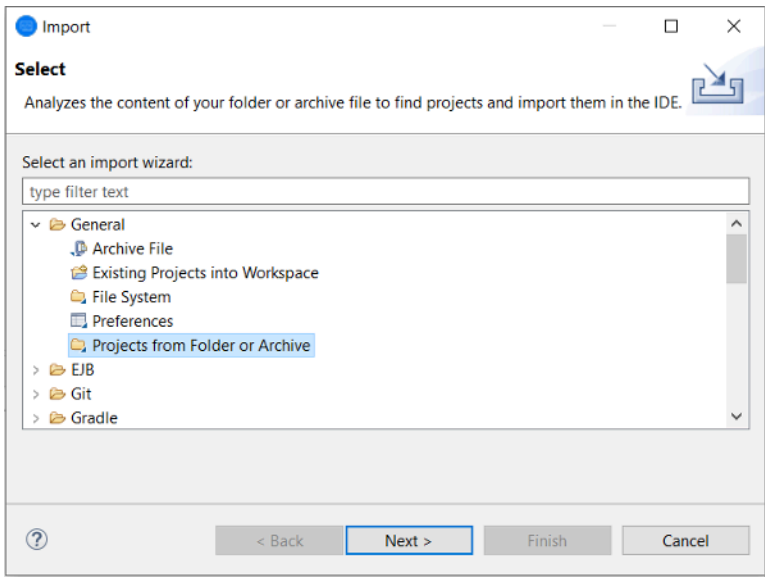
1. Enterprise Developer を起動し、選択済みのワークスペースを指定します。



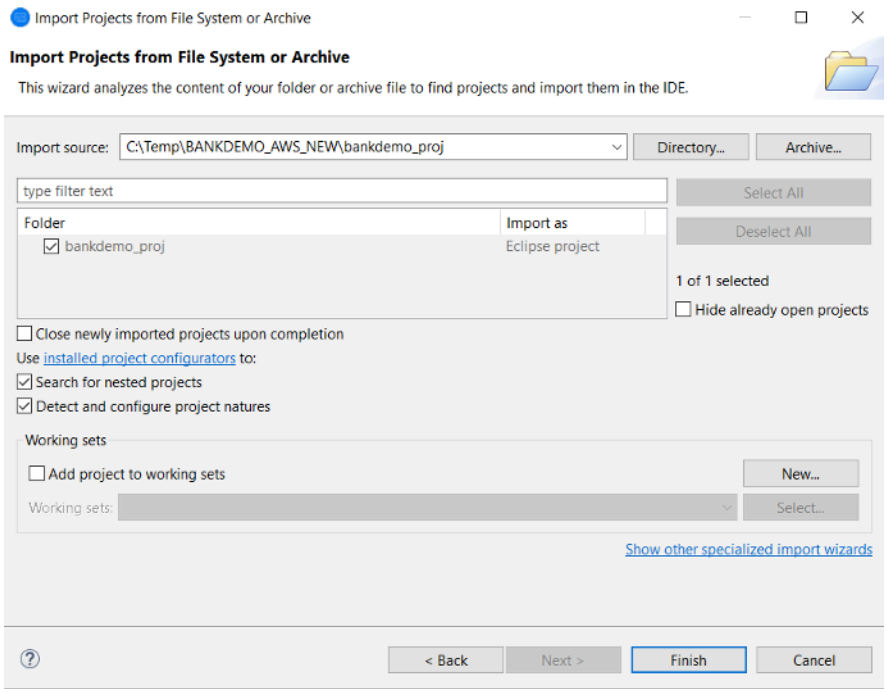
2. [ファイル] メニューから [インポート] を選択します。



3. 表示されたダイアログの [一般] で、[フォルダまたはアーカイブからのプロジェクト] を選択し、[次へ] を選択します。



4. [ソースをインポート] を選択し、[ディレクトリ] を選択し、ファイルシステムを参照して事前定義済みのプロジェクトフォルダを選択します。この中に含まれるプロジェクトには、同じリポジトリ内のソースフォルダへのリンクがあります。

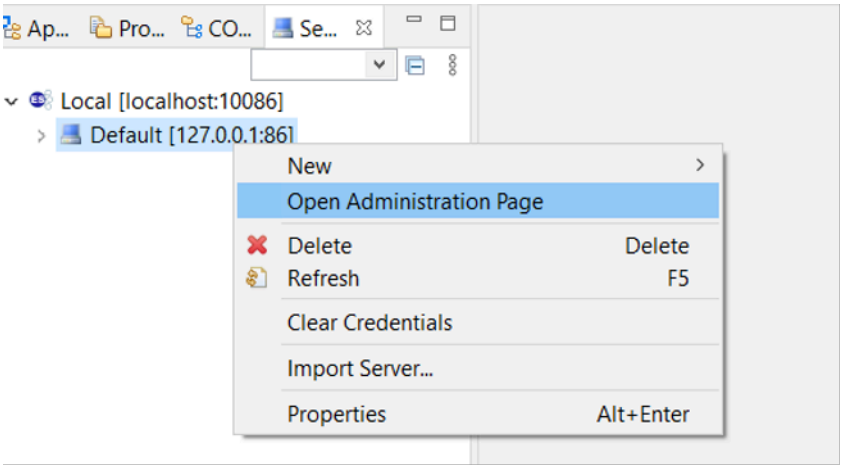


[終了] を選択します。

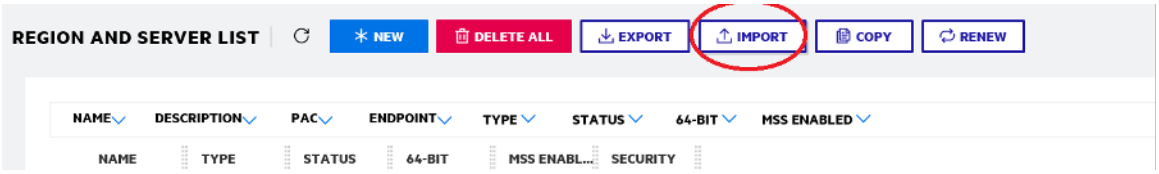
プロジェクトにはソースフォルダへのリンクが入力されるため、コードは自動的に構築されます。

リージョン定義 JSON テンプレートを使用する

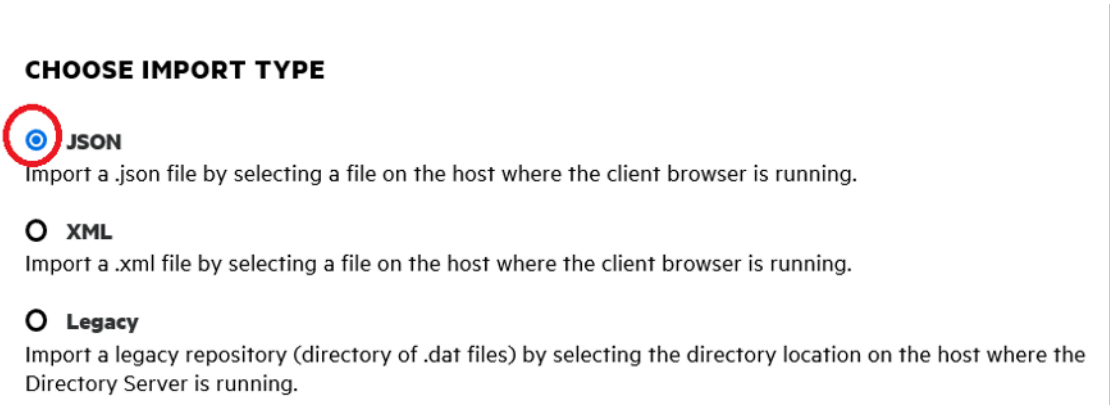
1. Server Explorer ビューに切り替えます。関連するコンテキストメニューから [管理ページを開く] を選択すると、デフォルトのブラウザが起動します。



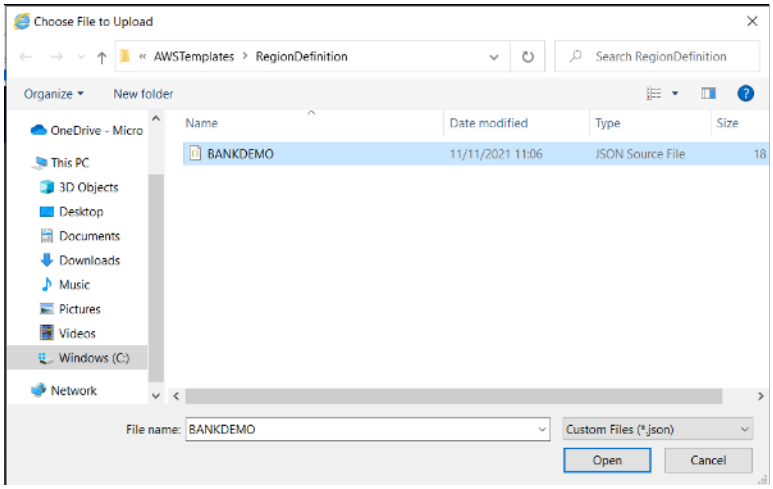
2. 表示された Server Explorer 共通 Web 管理 (ESCWA) 画面から、[インポート] を選択します。



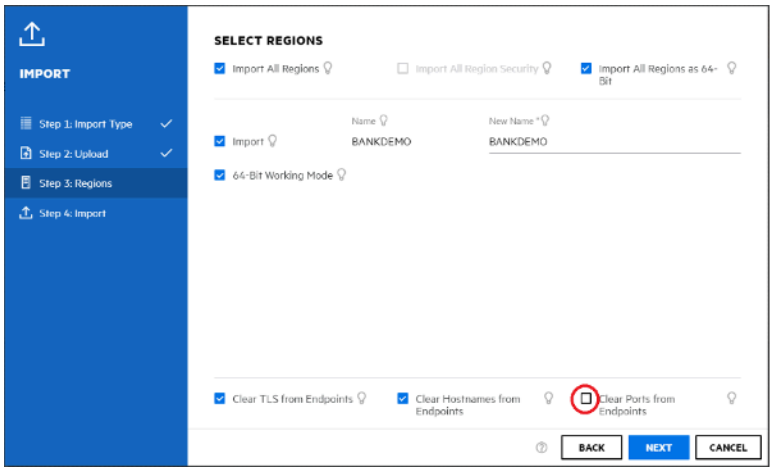
3. JSON インポートタイプを選択し、[次へ] を選択します。



4. 指定した `BANKDEMO.JSON` ファイルをアップロードします。

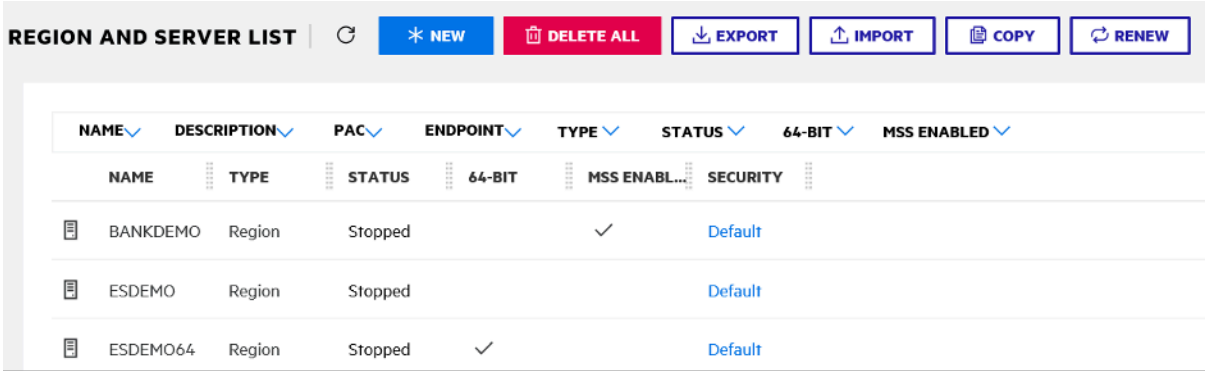


選択したら、[次へ] を選択します。

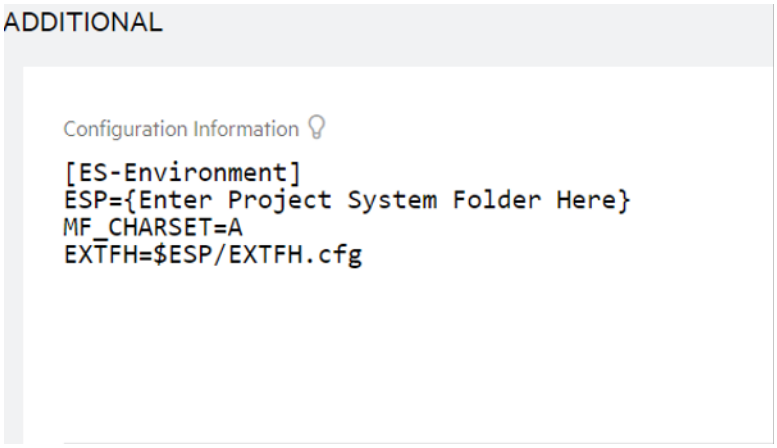


[リージョンの選択] パネルで、[エンドポイントからポートをクリア] オプションが未選択であることを確認してから、[インポートの実行] パネルが表示されるまで各パネルで [次へ] を選択します。次に、[インポート] を選択します。

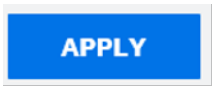
最後に [完了] をクリックします。この流れで、BANKDEMO リージョンがサーバーリストに追加されます。



5. BANKDEMO リージョンの [一般プロパティ] に移動します。
6. [設定] セクションにスクロールします。
7. 前のステップで作成した Eclipse プロジェクトに関連する `System` フォルダに ESP 環境変数を設定する必要があります。これは `workspacefolder/projectname/System` に設定されている必要があります。

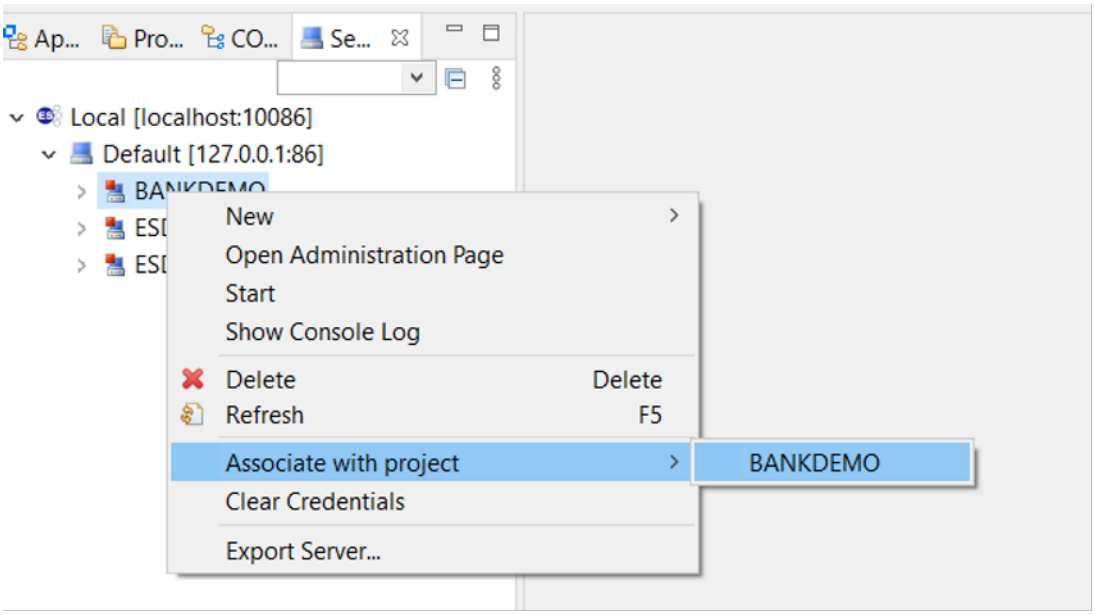


8. [適用] をクリックします。



これで、このリージョンが Eclipse COBOL プロジェクトと連携して動作するように設定が完了しました。

9. 最後に、Enterprise Developer に戻り、インポートしたリージョンをプロジェクトに関連付けます。



これで、Enterprise Developer 環境が使用可能になり、BankDemo の完全版が完成しました。設定済みのリージョンにおいてコードを編集、コンパイル、デバッグできます。

> **Important**
>
> Enterprise Developer for Windows を使用している場合、コンパイラによって生成されたバイナリは、Enterprise Developer が提供する Enterprise Server でのみ実行できます。Linux ベースの AWS Mainframe Modernization ランタイムでは実行できません。

チュートリアル: BankDemo サンプルアプリケーション用の Micro Focus ビルドのセットアップ

AWS Mainframe Modernization では、移行したアプリケーションのビルドと継続的インテグレーション/継続的デリバリー (CI/CD) パイプラインを設定できます。これらのビルドとパイプラインは、AWS CodeBuild、AWS CodeCommit、AWS CodePipeline を使用してこれらの機能を提供します。CodeBuild は完全マネージド型の構築サービスです。ソースコードのコンパイル、ユニットテストの実行、すぐにデプロイできるアーティファクトの生成を行います。CodeCommit は、AWS クラウドでの Git リポジトリのプライベートな保存と管理を有効にするバージョン制御サービスです。CodePipeline は、ソフトウェアをリリースするために必要な手順のモデル化、視覚化、および自動化ができる継続的なデリバリーサービスです。

このチュートリアルでは、AWS CodeBuild を使用して BankDemo サンプルアプリケーションのソースコードを Amazon S3 からコンパイルし、コンパイルしたコードを Amazon S3 にエクスポートする方法を示します。

AWS CodeBuild はフルマネージド型の継続的統合サービスであり、このサービスにより、ソースコードをコンパイルし、テストを実行し、デプロイ可能なソフトウェアパッケージを作成できます。CodeBuild では、パッケージ済みのビルド環境を使用するか、ご自分のビルドツールを使用するカスタムビルド環境を作成することができます。このデモシナリオでは 2 つ目のオプションを使用します。あらかじめパッケージ化された Docker イメージを使用する CodeBuild ビルド環境で構成されています。

> **Important**
>
> メインフレームモダナイゼーションプロジェクトを開始する前に、AWS メインフレーム向け Migration Acceleration Program (MAP) について確認するか、メインフレームアプリケーションのモダナイズに必要な手順について AWS メインフレームのスペシャリストに問い合わせることをお勧めします。

トピック

前提条件

このチュートリアルを開始する前に、次の前提条件を完了してください。

- BankDemo サンプルアプリケーションをダウンロードし、フォルダに解凍します。ソースフォルダには、COBOL プログラム、コピーブック、CICS BMS 定義が含まれています。JCL をビルドする必要はありませんが、参照用の JCL フォルダも含まれています。このフォルダには、ビルドに必要なメタファイルも含まれています。
- AWS メインフレームモダナイゼーションコンソールで [ツール] を選択します。[分析、開発、およびアセットの構築] で、[アカウントとアセットを共有する] を選択します。

ステップ 1: Amazon S3 バケットを作成する

このステップでは、2 つの Amazon S3 バケットを作成します。1 つ目はソースコードを保持する入力バケットで、もう 1 つはビルド出力を保持する出力バケットです。詳細については、「Amazon S3 ユーザーガイド」の「Amazon S3 バケットの作成、設定、操作」を参照してください。

1. 入力バケットを作成するには、Amazon S3 コンソールにログインし、[バケットの作成] を選択します。
2. [一般的な設定] では、バケット名を入力し、バケットを作成する AWS リージョン を指定します。名前の例は `codebuild-regionId-accountId-input-bucket` で、`regionId` はバケットの AWS リージョン で、`accountId` はユーザーの AWS アカウント ID です。

   > **Note**
   >
   > 米国東部 (バージニア北部) とは異なる AWS リージョン でバケットを作成する場合は、`LocationConstraint` パラメータを指定します。詳細については、「Amazon Simple Storage Service API リファレンス」の「バケットを作成」を参照してください。

3. その他の設定はすべて保持し、[バケットを作成] を選択します。
4. ステップ 1～3 を繰り返し、出力バケットを作成します。名前の例は `codebuild-regionId-accountId-output-bucket` で、`regionId` はバケットの AWS リージョン で、`accountId` はユーザーの AWS アカウント ID です。

   これらのバケットにどの名前を選択する場合でも、このチュートリアル全体で、その名前を使用してください。

ステップ 2: ビルド仕様ファイルを作成する

このステップでは、ビルド仕様ファイルを作成します。このファイルには、CodeBuild がビルドを実行するためのビルドコマンドと関連設定が含まれます。詳細については、「AWS CodeBuild ユーザーガイド」の「CodeBuild のビルド仕様に関するリファレンス」を参照してください。

1. 前提条件として解凍したディレクトリに、buildspec.yml という名前のファイルを作成します。
2. ファイルに以下の内容を追加して保存します。このファイルでは変更は必要ありません。

```
version: 0.2
env:
  exported-variables:
    - CODEBUILD_BUILD_ID
    - CODEBUILD_BUILD_ARN
phases:
  install:
    runtime-versions:
      python: 3.7
  pre_build:
    commands:
      - echo Installing source dependencies...
      - ls -lR $CODEBUILD_SRC_DIR/source
  build:
    commands:
      - echo Build started on `date`
      - /start-build.sh -Dbasedir=$CODEBUILD_SRC_DIR/source -Dloaddir=
$CODEBUILD_SRC_DIR/target
  post_build:
    commands:
      - ls -lR $CODEBUILD_SRC_DIR/target
      - echo Build completed on `date`
artifacts:
  files:
    - $CODEBUILD_SRC_DIR/target/**
```

ここでは、CODEBUILD_BUILD_ID、CODEBUILD_BUILD_ARN、$CODEBUILD_SRC_DIR/source、$CODEBUILD_SRC_DIR/target は CodeBuild 内で使用できる環境変数を示します。詳細については、「ビルド環境の環境変数」を参照してください。

この時点で、ディレクトリは次のようになります。

```
(root directory name)
    |-- build.xml
    |-- buildspec.yml
    |-- LICENSE.txt
    |-- source
         |... etc.
```

3. フォルダの内容を BankDemo.zip という名前のファイルに圧縮します。このチュートリアルでは、フォルダを圧縮することはできません。代わりに、フォルダの内容を BankDemo.zip ファイルに圧縮します。

ステップ 3: ソースファイルをアップロードする

このステップでは、BankDemo サンプルアプリケーションのソースコードを Amazon S3 入力バケットにアップロードします。

1. Amazon S3 コンソールにログインし、ナビゲーションペインで、[バケット] を選択します。次に、以前に作成した入力バケットを選択します。
2. [オブジェクト] で [アップロード] を選択します。
3. [ファイルとフォルダ] セクションで、[ファイルを追加] を選択します。
4. BankDemo.zip ファイルに移動して、選択します。
5. [Upload (アップロード)] を選択します。

ステップ 4: IAM ポリシーを作成する

このステップでは、2 つの IAM ポリシーを作成します。1 つのポリシーは、Micro Focus ビルドツールを含む Docker イメージにアクセスして使用する権限を AWS メインフレームモダナイゼーションに付与します。このポリシーはお客様向けにカスタマイズされていません。もう 1 つのポリシーは、AWS メインフレームモダナイゼーションが入力バケットと出力バケット、および CodeBuild が生成する Amazon CloudWatch ログを操作するためのアクセス許可を付与します。

IAM ポリシーの作成については、「IAM ユーザーガイド」の「IAM ポリシーの編集」を参照してください。

Docker イメージにアクセスするためのポリシーを作成するには

1. IAM コンソールで、次のポリシー文書をコピーしてポリシーエディタに貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:*:673918848628:repository/m2-enterprise-build-
tools"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::aws-m2-repo-*/*"
        }
    ]
}
```

2. ポリシーの名前を指定します (m2CodeBuildPolicy など)。

AWS メインフレームモダナイゼーションがバケットやログを操作できるようにするポリシーを作成するには

1. IAM コンソールで、次のポリシー文書をコピーしてポリシーエディタに貼り付けます。必ず regionId を AWS リージョン に accountId を AWS アカウント に更新してください。

```
{
      "Version": "2012-10-17",
      "Statement": [
```

```
            {
                "Action": [
                    "logs:CreateLogGroup",
                    "logs:CreateLogStream",
                    "logs:PutLogEvents"
                ],
                "Resource": [
                    "arn:aws:logs:regionId:accountId:log-group:/aws/codebuild/
codebuild-bankdemo-project",
                    "arn:aws:logs:regionId:accountId:log-group:/aws/codebuild/
codebuild-bankdemo-project:*"
                ],
                "Effect": "Allow"
            },
            {
                "Action": [
                    "s3:PutObject",
                    "s3:GetObject",
                    "s3:GetObjectVersion",
                    "s3:GetBucketAcl",
                    "s3:GetBucketLocation",
                    "s3:List*"
                ],
                "Resource": [
                    "arn:aws:s3:::codebuild-regionId-accountId-input-bucket",
                    "arn:aws:s3:::codebuild-regionId-accountId-input-bucket/*",
                    "arn:aws:s3:::codebuild-regionId-accountId-output-bucket",
                    "arn:aws:s3:::codebuild-regionId-accountId-output-bucket/*"
                ],
                "Effect": "Allow"
            }
        ]
    }
```

2. ポリシーの名前を指定します (BankdemoCodeBuildRolePolicy など)。

ステップ 5: IAM ロールを作成する

このステップでは、以前に作成した IAM ポリシーをこの新しい IAM ロールに関連付けた後に、CodeBuild が AWS リソースとやり取りできるようにする新しい IAM ロールを作成します。

サービスロールの作成について詳しくは、「IAM ユーザーガイド」の「AWS サービスにアクセス許可を委任するロールの作成」を参照してください。

1. IAM コンソールにログインし、左のナビゲーションペインで、[ロール] を選択します。
2. [Create role] (ロールの作成) を選択します。
3. [信頼されたエンティティタイプ] から、[AWS サービス] を選択します。
4. [他の AWS サービスのユースケース] で [CodeBuild] を選択し、もう一度 [CodeBuild] を選択します。
5. [Next] (次へ) をクリックします。
6. [アクセス許可を追加] ページで [次へ] を選択します。後でロールにポリシーを割り当てます。
7. [ロールの詳細] に、ロールの名前 (例: BankdemoCodeBuildServiceRole) を入力します。
8. [信頼されたエンティティを選択] で、ポリシードキュメントが以下のようになっていることを確認します。

```
{
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "codebuild.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        }
```

9. [Create role] (ロールの作成) を選択します。

ステップ 6: IAM ポリシーを IAM ロールにアタッチする

このステップでは、前に作成した IAM ポリシーを BankdemoCodeBuildServiceRole IAM ロールにアタッチします。

1. IAM コンソールにログインし、左のナビゲーションペインで、[ロール] を選択します。
2. [ロール] で、前に作成したロールを選択します (BankdemoCodeBuildServiceRole など)。

3. [アクセス許可ポリシー] で、[アクセス許可を追加]、[ポリシーをアタッチします] の順に選択します。
4. [その他の許可ポリシー] で、以前に作成したポリシー (例: `m2CodeBuildPolicy` と `BankdemoCodeBuildRolePolicy`) を選択します。
5. [ポリシーのアタッチ] を選択します。

ステップ 7: CodeBuild プロジェクトを作成する

このステップでは、CodeBuild プロジェクトを作成します。

1. CodeBuild コンソールにログインして [ビルドプロジェクトを作成する] を選択します。
2. [プロジェクトの設定] セクションで、プロジェクトの名前 (例: `codebuild-bankdemo-project`) を入力します。
3. [ソース] セクションの [ソースプロバイダー] で [Amazon S3] を選択し、以前に作成した入力バケット (例: `codebuild-regionId-accountId-input-bucket`) を選択します。
4. [S3 オブジェクトキーまたは S3 フォルダ] フィールドに S3 バケットにアップロードした zip ファイルの名前を入力します。この場合、ファイル名は `bankdemo.zip` です。
5. [環境] セクションで、[カスタムイメージ] を選択します。
6. [環境タイプ] フィールドで [Linux] を選択します。
7. [イメージレジストリ] で、[その他のレジストリ] を選択します。
8. [外部レジストリの URL] フィールドに、`673918848628.dkr.ecr.us-west-2.amazonaws.com/m2-enterprise-build-tools:latest` を入力します。
9. [サービスロール] で [既存のサービスロール] を選択し、[ロール ARN] フィールドで、以前に作成したサービスロール (例: `BankdemoCodeBuildServiceRole`) を選択します。
10. [Buildspec] セクションで [buildspec ファイルを使用する] を選択します。
11. [アーティファクト] セクションの [タイプ] で [Amazon S3] を選択し、次に出力バケット (例: `codebuild-regionId-accountId-output-bucket`) を選択します。
12. [名前] フィールドに、ビルド出力アーティファクトを格納するバケット内のフォルダの名前 (例: `bankdemo-output.zip`) を入力します。
13. [アーティファクトのパッケージ化] では、[Zip] を選択します。
14. Create build project (ビルドプロジェクトの作成)を選択します。

ステップ 8: ビルドを開始する

このステップでは、ビルドを開始します。

1. CodeBuild コンソールにログインします。
2. ナビゲーションペインで、[ビルドプロジェクト] を選択します。
3. 以前に作成したビルドプロジェクト (例: `codebuild-bankdemo-project`) を選択します。
4. [Start build] を選択します。

このコマンドはビルドを開始します。ビルドは非同期で実行されます。コマンドの出力は、属性 ID を含む JSON です。この属性 ID は、開始したばかりのビルドの CodeBuild ビルド ID への参照です。コンソールで、ビルドのステータスを表示することができます。ビルド実行に関する詳細なログをコンソールで確認することもできます。詳細については、「AWS CodeBuild ユーザーガイド」の「詳細なビルド情報を表示する」を参照してください。

現在のフェーズが完了すると、ビルドが正常に終了し、コンパイルされたアーティファクトが Amazon S3 で準備できたことを意味します。

ステップ 9: 出力アーティファクトをダウンロードする

このステップでは、Amazon S3 から出力アーティファクトをダウンロードします。Micro Focus ビルドツールでは、複数の異なる種類の実行ファイルを作成できます。このチュートリアルでは、共有オブジェクトを生成します。

1. Amazon S3 コンソールにログインします。
2. [バケット] セクションで、出力バケットの名前を選択します (例えば、`codebuild-regionId-accountId-output-bucket`)。
3. [ダウンロード] を選択します。
4. ダウンロードした ファイルを解凍します。ターゲットフォルダに移動して、ビルドアーティファクトを確認します。これには `.so` Linux 共有オブジェクトが含まれます。

リソースをクリーンアップする

このチュートリアルのために作成したリソースが不要になった場合は、追加料金が発生しないように削除します。そのためには、以下のステップを実行します。

- このチュートリアル用に作成した S3 バケットを削除します。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの削除」を参照してください。
- このチュートリアル用に作成した IAM ポリシーを削除します。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの削除」を参照してください。
- このチュートリアル用に作成した IAM ロールを削除します。詳細については、「IAM ユーザーガイド」の「ロールまたはインスタンスプロファイルを削除する」を参照してください。
- このチュートリアル用に作成した CodeBuild プロジェクトを削除します。詳細については、「AWS CodeBuild ユーザーガイド」の「Delete a build project in CodeBuild」を参照してください。

チュートリアル: Micro Focus エンタープライズデベロッパーで使用するための CI/CD パイプラインの設定

このチュートリアルでは、BankDemo サンプルアプリケーションを Micro Focus Enterprise Developer にインポート、編集、コンパイル、実行し、変更をコミットして CI/CD パイプラインをトリガーする方法を示します。

目次

前提条件

次のファイルをダウンロードします。

- `basic-infra.yaml`
  - 欧州 (フランクフルト) リージョンからダウンロードします。
  - 米国東部 (バージニア北部) リージョンからダウンロードします。
- `pipeline.yaml`
  - 欧州 (フランクフルト) リージョンからダウンロードします。
  - 米国東部 (バージニア北部) リージョンからダウンロードします。
- `m2-code-sync-function.zip`
  - 欧州 (フランクフルト) リージョンからダウンロードします。
  - 米国東部 (バージニア北部) リージョンからダウンロードします。
- `config_git.yml`
  - 欧州 (フランクフルト) リージョンからダウンロードします。
  - 米国東部 (バージニア北部) リージョンからダウンロードします。
- `BANKDEMO-source.zip`
  - 欧州 (フランクフルト) リージョンからダウンロードします。
  - 米国東部 (バージニア北部) リージョンからダウンロードします。
- `BANKDEMO-exercise.zip`

- 欧州 (フランクフルト) リージョンからダウンロードします。
- 米国東部 (バージニア北部) リージョンからダウンロードします。

各ファイルの目的は次のとおりです。

`basic-infra.yaml`

この AWS CloudFormation テンプレートは、CI/CD パイプラインに必要な基本インフラストラクチャ (VPC、Amazon S3 バケットなど) を作成します。

`pipeline.yaml`

この AWS CloudFormation テンプレートは、Lambda 関数がパイプラインスタックを起動するために使用されます。このテンプレートが、パブリックにアクセス可能な Amazon S3 バケットにあることを確認してください。このバケットへのリンクを `basic-infra.yaml` テンプレートの `PipelineTemplateURL` パラメータのデフォルト値として追加します。

`m2-code-sync-function.zip`

この Lambda 関数は、CodeCommit リポジトリ、`config_git.yaml` に基づくディレクトリ構造を作成し、`pipeline.yaml` を使用してパイプラインスタックを起動します。この zip ファイルが、AWS メインフレームの近代化がサポートされているすべての AWS リージョン で、一般にアクセス可能な Amazon S3 バケットで利用できることを確認してください。ファイルを 1 つの AWS リージョン にある 1 つのバケットに保存し、すべての AWS リージョン でバケットに複製することをお勧めします。特定のものを識別するサフィックスを付けたバケットの命名規則 AWS リージョン (例:`m2-cicd-deployment-source-eu-west-1`) を使用し、そのプレフィックス `m2-cicd-deployment-source` をパラメータ `DeploymentSourceBucket` のデフォルト値として追加し、リソース `SourceSyncLambdaFunction` の `basic-infra.yaml` テンプレートでそのバケットを参照しながら AWS CloudFormation 代替関数 `!Sub {DeploymentSourceBucket}-${AWS::Region}` を使用して完全なバケットを形成します。

`config_git.yml`

CodeCommit ディレクトリ構造定義。詳細については、「サンプル YAML トリガーファイル config_git.yml」を参照してください。

`BANKDEMO-source.zip`.

CodeCommit リポジトリから作成された BankDemo ソースコードと設定ファイル。

`BANKDEMO-exercise.zip`.

CodeCommit リポジトリから作成されたチュートリアル演習用の BankDemo ソース。

CI/CD パイプラインの基本インフラストラクチャの作成

AWS CloudFormation テンプレート `basic-infra.yaml` を使用して、AWS CloudFormation コンソールから CI/CD パイプラインの基本インフラストラクチャスタックを作成します。このスタックは、アプリケーションコードとデータをアップロードする Amazon S3 バケットと、AWS CodeCommit リポジトリや AWS CodePipeline パイプラインなどの他の必要なリソースを作成するためのサポート AWS Lambda 関数を作成します。

> **Note**
>
> このスタックを起動するには、IAM、Amazon S3、Lambda、AWS CloudFormation を管理するためのアクセス許可と、AWS KMS を使用する権限が必要です。

1. AWS Management Console にサインインし、AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. 次のオプションのうち 1 つを使用して新しいスタックを作成します。
   - [Create Stack] (スタックの作成) を選択します。現在実行中のスタックがある場合は、これが唯一のオプションです。
   - [スタック] ページで [スタックを作成] を選択します。このオプションは、実行中のスタックがない場合にのみ表示されます。
3. [テンプレートを指定] ページで:
   - [テンプレートの準備] の [テンプレートの準備完了] を選択します。
   - [テンプレートを指定] で、テンプレートソースとして Amazon S3 URL を選択し、AWS リージョン に応じて次の URL のいずれかを入力します。
     - `https://m2-us-east-1.s3.us-east-1.amazonaws.com/cicd/mf/basic-infra.yaml`
     - `https://m2-eu-central-1.s3.eu-central-1.amazonaws.com/cicd/mf/basic-infra.yaml`
   - 設定を受け入れるには、[次へ] を選択します。

   [スタックの作成] ページが開きます。

Specify stack details

Stack name

Stack name

m2-cicd-pipeline-foundation

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

Networking Configuration

Do you want to use an existing VPC in your account?

If you select 'Yes', then you must provide the VPC ID and the Subnet IDs.

No (Create one)

Which VPC ID should be used?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Which private subnet ID should be used?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Which private subnet ID in a different AZ should be used for HA?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Enter the CIDR block that should be used for the new VPC

If you selected 'No (Create one)' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

11.11.0.0/16

CIDR bits for creating subnets. Choose 5 for /27, 6 for /26, 7 for /25, 8 for /24 range

If you selected 'No (Create one)' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

5

Deployment Configuration

Name of the S3 bucket which contains the source files for this stack deployment

Don't change unless you know what you are doing.

m2

Name of the source package file for the infrastructure Lambda function

Don't change unless you know what you are doing.

cicd/mf/m2-code-sync-function.zip

Full URL of the pipeline CloudFormation template file

Don't change unless you know what you are doing.

https://m2-us-east-1.s3.us-east-1.amazonaws.com/cicd/mf/pipeline.yaml

What name prefix to use for the new S3 buckets?

A name prefix for the S3 buckets that will be created by this stack.

mf

Cancel Previous Next

以下の変更を加えます。

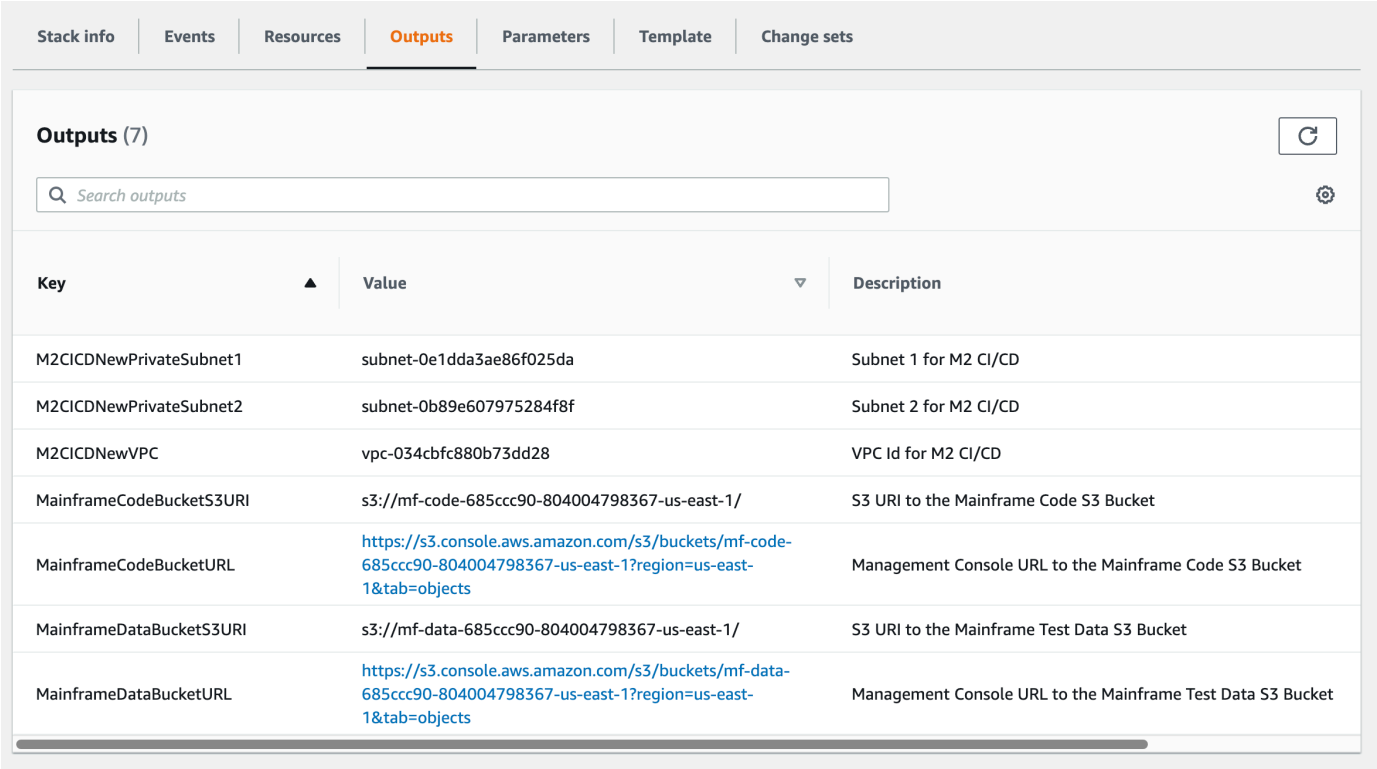
- [スタック名] には適切な値を、[ネットワーキング設定] にはパラメータを指定します。
- デプロイ設定のほとんどのパラメータは適切にあらかじめ入力されているので、変更する必要はありません。AWS リージョン に応じて、パイプライン AWS CloudFormation テンプレートを次の Amazon S3 URL のいずれかに変更します。
  - `https://m2-us-east-1.s3.amazonaws.com/cicd/mf/pipeline.yaml`
  - `https://m2-eu-central-1.s3.eu-central-1.amazonaws.com/cicd/mf/pipeline.yaml`
- [Next] (次へ) をクリックします。

> **Note**
>
> AWS CloudFormation テンプレートを自分で変更していない限り、デフォルトのパラメータ値を変更しないでください。

4. [スタックオプションの設定] で、[次へ] を選択します。
5. [機能] で、[IAM AWS CloudFormation リソースを作成する可能性がある] を選択して、AWS CloudFormation にユーザーに代わって IAM ロールを作成する権限を付与します。[スタックの作成] を選択します。

> **Note**
>
> スタックがプロビジョニングされるまで、最大 3~5 分かかる場合があります。

6. スタックが正常に作成されたら、新しくプロビジョニングされたスタックの [出力] セクションに移動します。そこには、メインフレームコードと依存ファイルをアップロードする必要がある Amazon S3 バケットがあります。



AWS CodeCommit リポジトリと CI/CD パイプラインの作成

このステップでは、CodeCommit リポジトリを作成し、パイプラインスタックを作成する AWS CloudFormation を呼び出す Lambda 関数を呼び出して CI/CD パイプラインスタックをプロビジョニングします。

1. BankDemo サンプルアプリケーションをローカルマシンにダウンロードします。
2. ローカルマシンから CI/CD パイプラインの基本インフラストラクチャの作成 で作成した Amazon S3 `bankdemo.zip` バケットにアップロードします。
3. `config_git.yml` をダウンロードします。
4. 必要に応じて `config_git.yml` を次のように変更します。

   - 独自のターゲットリポジトリ名、ターゲットブランチ、コミットメッセージを追加します。

   ```
   repository-config:
     target-repository: bankdemo-repo
     target-branch: main
     commit-message: Initial commit for bankdemo-repo main branch
   ```

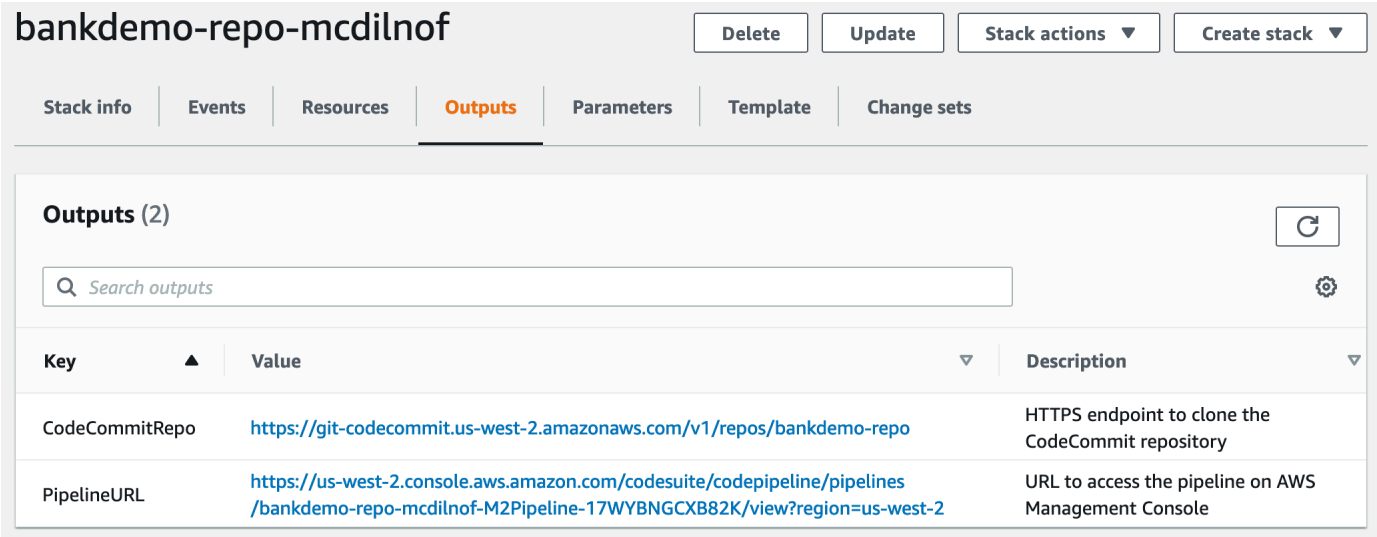
   - 通知を受信する E メールアドレスを入力します。

```
pipeline-config:
  # Send pipeline failure notifications to these email addresses
  alert-notifications:
    - myname@mycompany.com
  # Send notifications for manual approval before production deployment to these
 email addresses
  approval-notifications:
    - myname@mycompany.com
```

5. CodeCommit リポジトリのフォルダ構造の定義を含む `config_git.yml` ファイルを、CI/CD パイプラインの基本インフラストラクチャの作成 で作成した Amazon S3 バケットにアップロードします。これにより、リポジトリとパイプラインを自動的にプロビジョニングする Lambda 関数が呼び出されます。

   これにより、`config_git.yml` ファイルに定義されている `target-repository` で指定する名前で CodeCommit リポジトリが作成されます (例: `bankdemo-repo`)。

   また、Lambda 関数は AWS CloudFormation 経由で CI/CD パイプラインスタックを作成します。AWS CloudFormation スタックには、指定した `target-repository` の名前と同じプレフィックスが付けられ、その後にランダムな文字列が続きます (例:`bankdemo-repo-`*`01234567`*)。CodeCommit リポジトリの URL と、作成したパイプラインにアクセスするための URL は AWS 管理コンソールで確認できます。



6. CodeCommit リポジトリの作成が完了すると、CI/CD パイプラインが直ちにトリガーされ、完全な CI/CD が実行されます。

7. ファイルがプッシュされると、パイプラインが自動的にトリガーされます。パイプラインは、ビルド、ステージングへのデプロイ、テストの実行、手動承認を待ってから本番環境にデプロイされます。

サンプル YAML トリガーファイル config_git.yml

```
repository-config:
  target-repository: bankdemo-repo
  target-branch: main
  commit-message: Initial commit for bankdemo-repo main branch
  directory-structure:
    - '/':
        files:
        - build.xml
        - '*.yaml'
        - '*.yml'
        - '*.xml'
        - 'LICENSE.txt'
        readme: |
          # Root Folder
          - 'build.xml' : Build configuration for the application
    - tests:
        files:
        - '*.py'
        readme: |
          # Test Folder
          - '*.py' : Test scripts
    - config:
        files:
        - 'BANKDEMO.csd'
        - 'BANKDEMO.json'
        - 'BANKDEMO_ED.json'
        - 'dfhdrdat'
        - 'ESPGSQLXA.dll'
        - 'ESPGSQLXA64.so'
        - 'ESPGSQLXA64_S.so'
        - 'EXTFH.cfg'
        - 'm2-2021-04-28.normal.json'
        - 'MFDBFH.cfg'
        - 'application-definition-template-config.json'
        readme: |
          # Config Folder
```

```
          This folder contains the application configuration files.
          - 'BANKDEMO.csd'     : CICS Resource definitions export file
          - 'BANKDEMO.json'    : Enterprise Server configuration
          - 'BANKDEMO_ED.json' : Enterprise Server configuration for ED
          - 'dfhdrdat'         : CICS resource definition file
          - 'ESPGSQLXA.dll'    : XA switch module Windows
          - 'ESPGSQLXA64.so'   : XA switch module Linux
          - 'ESPGSQLXA64_S.so' : XA switch module Linux
          - 'EXTFH.cfg'        : Micro Focus File Handler configuration
          - 'm2-2021-04-28.normal.json' : M2 request document
          - 'MFDBFH.cfg'       : Micro Focus Database File Handler
          - 'application-definition-template-config.json' : Application definition for
 M2
    - source:
        subdirs:
        - .settings:
            files:
              - '.bms.mfdirset'
              - '.cbl.mfdirset'
        - copybook:
            files:
              - '*.cpy'
              - '*.inc'
            readme: |
              # Copy folder
              This folder contains the source for COBOL copy books, PLI includes, ...
              - .cpy COBOL copybooks
              - .inc PLI includes
#        - ctlcards:
#            files:
#              - '*.ctl'
#              - 'KBNKSRT1.txt'
#            readme: |
#              # Control Card folder
#              This folder contains the source for Batch Control Cards
#              - .ctl Control Cards
        - ims:
            files:
              - '*.dbd'
              - '*.psb'
            readme: |
              # ims folder
              This folder contains the IMS DB source files with the extensions
              - .dbd for IMS DBD source
```

```
              - .psb for IMS PSB source
        - jcl:
            files:
              - '*.jcl'
              - '*.ctl'
              - 'KBNKSRT1.txt'
              - '*.prc'
            readme: |
              # jcl folder
              This folder contains the JCL source files with the extensions
              - .jcl
#        - proclib:
#            files:
#              - '*.prc'
#            readme: |
#              # proclib folder
#              This folder contains the JCL procedures referenced via PROCLIB
 statements in the JCL with extensions
#              - .prc
        - rdbms:
            files:
              - '*.sql'
            readme: |
              # rdbms folder
              This folder contains any DB2 related source files with extensions
              - .sql for any kind of SQL source
        - screens:
            files:
            - '*.bms'
            - '*.mfs'
            readme: |
              # screens folder
              This folder contains the screens source files with the extensions
              - .bms for CICS BMS screens
              - .mfs for IMS MFS screens
            subdirs:
            - .settings:
                files:
                    - '*.bms.mfdirset'
        - cobol:
            files:
              - '*.cbl'
              - '*.pli'
            readme: |
```

```
              # source folder
              This folder contains the program source files with the extensions
              - .cbl for COBOL source
              - .pli for PLI source
            subdirs:
            - .settings:
                files:
                   - '*.cbl.mfdirset'
    - tests:
        files:
        - 'test_script.py'
        readme: |
          # tests Folder
          This folder contains the application test scripts
pipeline-config:
  alert-notifications:
    - myname@mycompany.com
  approval-notifications:
    - myname@mycompany.com
```

Enterprise Developer AppStream 2.0 の作成

AppStream 2.0 で Micro Focus Enterprise Developer をセットアップするには、「チュートリアル: AppStream 2.0 で Micro Focus Enterprise Developer をセットアップする」を参照してください。

CodeCommit リポジトリをエンタープライズデベロッパーに接続するには、サンプル YAML トリガーファイル config_git.yml の `target-repository`で指定された名前を使用します。

Enterprise Developer のセットアップとテスト

トピック

- Enterprise Developer の BankDemo CodeCommit リポジトリをクローンする
- BankDemo メインフレーム COBOL プロジェクトを作成し、アプリケーションをビルドする
- テスト用のローカル BankDemo CICS とバッチ環境を作成する
- Enterprise Developer から BANKDEMO サーバーを起動する
- Rumba 3270 ターミナルを起動する
- BankDemo トランザクションを実行する
- Enterprise Developer から BANKDEMO サーバーを停止します。

Enterprise Developer AppStream 2.0 の作成 で作成した Enterprise Developer AppStream 2.0 インスタンスに接続します。

1. Windows スタートから Enterprise Developer を起動します。Micro Focus Enterprise Developer を選択し、次に Enterprise Developer for Eclipse を選択します。初めて起動する場合は、時間がかかる場合があります。
2. Eclipse ランチャーの [ワークスペース] に「`C:\Users\<username>\workspace`」と入力し、[起動] を選択します。

   > **Note**
   >
   > AppStream 2.0 インスタンスに再接続した後に、必ず同じ場所を選択してください。ワークスペースの選択は永続的ではありません。

3. [ようこそ] で、[COBOL Perspective を開く] を選択します。これは新しいワークスペースで最初のみ表示されます。

Enterprise Developer の BankDemo CodeCommit リポジトリをクローンする

1. [ウィンドウ]/[Perspective]/[Perspective を開く]/[その他...] /[Git] を選択します。
2. [Git リポジトリのクローンを作成] を選択します。
3. [Git リポジトリをクローン] で、次の情報を入力します。

   - [ロケーション URI] に、CodeCommit リポジトリの HTTPS URL を入力します。

     > **Note**
     >
     > AWS 管理コンソールの CodeCommit リポジトリのクローン URL HTTPS をコピーし、ここに貼り付けます。URI は [ホスト] パスと [リポジトリ] パスに分割されます。

   - ユーザー CodeCommit リポジトリ認証情報は [認証ユーザー] と [パスワード] にあり、[セキュアストア] で [保存] を選択します。
4. [ブランチ選択] で [メイン] ブランチを選択し、[次へ] を選択します。
5. [ローカルの宛先] の [ディレクトリ] に `C:\Users\<username>\workspace` と入力して [完了] を選択します。

   [Git リポジトリ] ビューに `BANKDEMO [main]` が表示されたら、クローン処理は完了です。

BankDemo メインフレーム COBOL プロジェクトを作成し、アプリケーションをビルドする

1. [COBOL Perspective] に変更します。
2. [プロジェクト] で [自動的にビルド] を無効にします。
3. [ファイル] で [新規] を選択し、[メインフレーム COBOL プロジェクト] を選択します。
4. [新規メインフレーム COBOL プロジェクト] に、以下の情報を入力します。
   - [プロジェクト名] に `BankDemo` を入力します。
   - [Micro Focus テンプレート [64 ビット]] を選択します。
   - [終了] を選択します。
5. [COBOL Explorer] で、新しい BankDemo プロジェクトを展開します。

   > **Note**
   >
   > 角かっこ内の `[BANKDEMO main]` は、プロジェクトがローカルの BankDemo CodeCommit リポジトリに接続されていることを示します。

6. ツリービューに COBOL プログラム、コピーブック、BMS ソース、JCL ファイルのエントリが表示されない場合は、BankDemo プロジェクトのコンテキストメニューから [更新] を選択します。
7. BankDemo コンテキストメニューから [プロパティ]/[Micro Focus]/[プロジェクト設定]/[COBOL] を選択します。
   - [文字セット- ASCII] を選択します。
   - [適用]、[閉じる] の順に選択します。
8. BMS と COBOL ソースのビルドがすぐに開始されない場合は、[プロジェクト] メニューで [自動的にビルド] オプションが有効になっているか確認してください。

   ビルド出力は [コンソール] ビューに表示され、数分後にメッセージ `BUILD SUCCESSFUL` と `Build finished with no errors` とともに完了します。

   これで BankDemo アプリケーションがコンパイルされ、ローカルで実行できる状態になっているはずです。

テスト用のローカル BankDemo CICS とバッチ環境を作成する

1. [COBOL Explorer] で BANKDEMO / config を展開します。
2. エディタで、BANKDEMO_ED.json を開きます。
3. 文字列 ED_Home= を検索し、D:\\<username>\\workspace\\BANKDEMO のように、Enterprise Developer プロジェクトを指すようにパスを変更します。パス定義では二重スラッシュ (\\) が使用されていることに注意してください。
4. ファイルを保存して閉じます。
5. [Server Explorer] を選択します。
6. [デフォルト] コンテキストメニューから [管理ページを開く] を選択します。Micro Focus Enterprise Server [管理] ページがデフォルトのブラウザで開きます。
7. AppStream 2.0 セッションのみ、ローカルの Enterprise Server リージョンをローカルテスト用に保持できるように、次の変更を行います。

   - [ディレクトリサーバー]/[デフォルト] で、[プロパティ]/[設定] を選択します。
   - [リポジトリの場所] を D:\<username>\My Files\Home Folder\MFDS に置き換えます。

   > **Note**
   >
   > AppStream 2.0 インスタンスに新たに接続するたびに、ステップ 5～8 を完了する必要があります。

8. [ディレクトリサーバー]/[デフォルト] で [インポート] を選択し、次の手順を実行します。

   - 「ステップ 1: インポートの種類」で、[JSON] を選択し、[次へ] を選択します。
   - [ステップ 2: アップロード] で、青い四角形のファイルをクリックしてアップロードします。
   - [アップロードするファイルを選択] に
     - ファイル名: D:\<username>\workspace\BANKDEMO\config\BANKDEMO_ED.json を入力します。
     - 開く をクリックします。
   - [Next] (次へ) をクリックします。
   - [ステップ 3: リージョン] で、[エンドポイントからポートをクリア] をクリアします。
   - [Next] (次へ) をクリックします。
   - [ステップ 4: インポート] で、[インポート] を選択します。

- [終了] を選択します。

これで、リストに新しいサーバー名 BANKDEMO が表示されます。

Enterprise Developer から BANKDEMO サーバーを起動する

1. [Enterprise Developer] を選択します。
2. [Server Explorer] で [デフォルト] を選択し、コンテキストメニューから [更新] を選択します。

   これで、サーバーリストにも BANKDEMO が表示されるはずです。
3. [BANKDEMO] を選択します。
4. コンテキストメニューから [プロジェクトに関連付ける] を選択し、[BANKDEMO] を選択します。
5. コンテキストメニューから [開始] を選択します。

   [コンソール] ビューにはサーバースタートアップのログが表示されるはずです。

   メッセージ `BANKDEMO CASSI5030I PLTPI Phase 2 List(PI) Processing Completed` が表示されれば、サーバーは CICS BANKDEMO アプリケーションをテストする準備ができています。

Rumba 3270 ターミナルを起動する

1. Windows スタートから、Micro Focus Rumba+ Desktop/Rumba+ Desktop を起動します。
2. [ようこそ] で、[新規セッションを作成]/[メインフレームディスプレイ] を選択します。
3. [メインフレームディスプレイ] で、[接続]/[設定] を選択します。
4. [セッション設定] で、[接続]/[TN3270] を選択します。
5. [ホスト名]/[アドレス] に [挿入] を選択し、IP アドレス `127.0.0.1` を入力します。
6. [Telnet ポート] にポート `6000` を入力します。
7. [Apply] (適用) を選択します。
8. [接続]を選択します。

   CICS のウェルカム画面には、行 1 のメッセージ `This is the Micro Focus MFE CICS region BANKDEMO` を含む画面が表示されます。

9. Ctrl + Shift + Z キーを押して画面をクリアします。

BankDemo トランザクションを実行する

1. 空白の画面で、BANK と入力します。
2. [BANK10] 画面の [ユーザー ID...:] の入力フィールドに guest と入力して Enter キーを押します。
3. [BANK20] 画面の [ローンの費用を計算] の前の入力フィールドに / (スラッシュ) を入力して Enter キーを押します。
4. [BANK70] で次のように操作します。
   - [借りたい金額...:] で 10000 と入力します。
   - [金利...:] で 5.0 と入力します。
   - [借入期間 (月)...:] で 10 と入力します。
   - [Enter] キーを押します。

   次の結果が表示されます。

   ```
   Resulting monthly payment.............:   $1023.06
   ```

   これで、Enterprise Developer の BANKDEMO アプリケーションのセットアップは完了です。

Enterprise Developer から BANKDEMO サーバーを停止します。

1. [Server Explorer] で [デフォルト] を選択し、コンテキストメニューから [更新] を選択します。
2. [BANKDEMO] を選択します。
3. コンテキストメニューから [停止] を選択します。

   [コンソール] ビューにはサーバー停止のログが表示されるはずです。

   メッセージ Server: BANKDEMO stopped successfully が表示されれば、サーバーは正常にシャットダウンされています。

演習 1: BANKDEMO アプリケーションでのローン計算の強化

トピック

このシナリオでは、コードにサンプル変更を加え、デプロイし、テストするプロセスを順を追って説明します。

ローン部門は、[ローン計算] 画面の BANK70 に新しい項目を追加して、ローン総額を表示することを希望しています。そのためには、BMS 画面の MBANK70.CBL を変更し、新しいフィールドと、対応する画面処理プログラム SBANK70P.CBL と関連するコピーブックを追加する必要があります。さらに、BBANK70P.CBL のローン計算ルーチンを追加の計算式で拡張する必要があります。

この演習を完了するには、以下の前提条件を完了してください。

- BANKDEMO-exercise.zip を `D:\PhotonUser\My Files\Home Folder` にダウンロードしてください。
- .zip ファイルを `D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise` に展開します。
- `D:\PhotonUser\My Files\Home Folder\AnalysisRules` フォルダを作成します。
- ルールファイル `Loan+Calculation+Update.General-1.xml` を `BANKDEMO-exercise` フォルダから `D:\PhotonUser\My Files\Home Folder\AnalysisRules` にコピーします。

> **Note**
>
> この演習では、*.CBL と*.CPY のコード変更は、列 1 ~ 6 に EXER01 とマークされています。

Enterprise Developer コード分析にローン分析ルールを追加します。

Micro Focus Enterprise Analyzer で定義された分析ルールを Enterprise Analyzer からエクスポートして Enterprise Developer にインポートすることによって、Enterprise Developer プロジェクトのソース全体で同じ分析ルールを実行できます。

1. `Window/Preferences/Micro Focus/COBOL/Code Analysis/Rules` を開きます。
2. [編集...] を選択し、ルールファイル `Loan+Calculation+Update.General-1.xml` を含むフォルダ名 `D:\PhotonUser\My Files\Home Folder\AnalysisRules` を入力します。
3. [終了] を選択します。
4. [適用]、[閉じる] の順に選択します。
5. BANKDEMO プロジェクトのコンテキストメニューから [コード分析] を選択します。

   [ローン計算更新] のエントリが表示されます。

ステップ 1: ローン計算のコード分析を実行する

新しい分析ルールでは、検索パターン*PAYMENT*、*LOAN*、式の *RATE*、ステートメント、変数に一致する COBOL プログラムとコード行を特定することを希望しています。これにより、コードをナビゲートし、必要なコード変更を特定しやすくなります。

1. BANKDEMO プロジェクトのコンテキストメニューから、[コード分析/ローン計算更新] を選択します。

   これにより、検索ルールが実行され、[コード分析] という新しいタブに結果が一覧表示されます。右下の緑色の進行状況バーが消えたら、分析の実行は完了です。

   [コード分析] タブには、`BBANK20P.CBL`、`BBANK70P.CBL`、`SBANK70P.CBL` の拡張リストが表示され、各リストには検索パターンに一致するステートメント、式、変数が一覧表示されます。

   `BBANK20P.CBL` の結果を見ると、検索パターンと一致するリテラルだけが移動されていることがわかります。したがって、このプログラムは無視できます。
2. タブメニューバーで [-] アイコンを選択すると、すべて折りたたまれます。
3. ダブルクリックで `SBANK70P.CBL` を展開して任意の行を任意の順序で選択すると、ソースが開き、ソースコードで選択した行が強調表示されます。また、識別されたソース行がすべてマークされていることもわかります。

ステップ 2: CICS BMS マップと COBOL プログラムの修正とテスト

まず、BMS マップ MBANK70.BMS と画面処理プログラム SBANK70P.CBL およびコピーブック CBANKDAT.CPY を変更して、新しいフィールドを表示します。この演習では不必要なコーディングを避けるため、変更したソースモジュールを D:\PhotonUser\My Files\Home Folder \BANKDEMO-exercise\Exercise01 フォルダ内に用意しています。通常、開発者はコード分析の結果を使用してソースをナビゲートしたり変更したりします。時間に余裕があり、手動で変更を希望する場合は、*MBANK70.BMS と SBANK70P.CBL の手動変更 (オプション) * に記載されている情報を参考にしてください。

すばやく変更するには、次のファイルをコピーしてください。

1. ..\BANKDEMO-exercise\Exercis01\screens\MBANK70.BMS を D:\PhotonUser \workspace\bankdemo\source\screens へ
2. .\BANKDEMO-exercise\Exercis01\cobol\SBANK70P.CBL を D:\PhotonUser \workspace\bankdemo\source\cobol へ
3. ..\BANKDEMO-exercise\Exercis01\copybook\CBANKDAT.CPY を D:\PhotonUser \workspace\bankdemo\source\copybook へ
4. 変更の影響を受けるすべてのプログラムが確実にコンパイルされるようにするには、[プロジェクト/クリーン.../すべてのプロジェクトをクリーンアップ] を選択してください。

MBANK70.BMS と SBANK70P.CBL に手動で変更するには、以下の手順を実行してください。

- BMS MBANK70.BMS ソースを手動で変更する場合は、PAYMENT フィールドの後に追加してください。
  - TXT08 と同じ属性で、初期値が「ローン総額」の TXT09
  - 支払いと同じ属性の合計

変更のテスト

変更をテストするには、以下のセクションの手順を繰り返します。

1. Enterprise Developer から BANKDEMO サーバーを起動する
2. Rumba 3270 ターミナルを起動する
3. BankDemo トランザクションを実行する

さらに、テキスト Total Loan Amount.....: も表示されるはずです。

4. Enterprise Developer から BANKDEMO サーバーを停止します。

ステップ 3: COBOL プログラムに合計金額計算を追加

2 番目のステップでは、BBANK70P.CBL を変更してローン総額の計算を追加します。必要な変更を加えた準備済みのソースは D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise\Exercise01 フォルダにあります。時間に余裕があり、手動で変更を希望する場合は、*BBANK70P.CBL の手動変更 (オプション) * に記載されている情報を参考にしてください。

すばやく変更するには、次のファイルをコピーしてください。

- ..\BANKDEMO-exercise\Exercis01\source\cobol\BBANK70P.CBL を D:\PhotonUser\workspace\bankdemo\source\cobol へ

BBANK70P.CBL に手動で変更するには、以下の手順を実行します。

- コード分析結果を使用して、必要な変更を特定します。

変更のテスト

変更をテストするには、以下のセクションの手順を繰り返します。

1. Enterprise Developer から BANKDEMO サーバーを起動する
2. Rumba 3270 ターミナルを起動する
3. BankDemo トランザクションを実行する

   さらに、テキスト Total Loan Amount.....: $10230.60 も表示されるはずです。
4. Enterprise Developer から BANKDEMO サーバーを停止します。

ステップ 4: 変更をコミットして CI/CD パイプラインを実行する

変更を中央の CodeCommit リポジトリにコミットし、CI/CD パイプラインをトリガーして変更をビルド、テスト、デプロイします。

1. BANKDEMO プロジェクトから、コンテキストメニューで [チーム/コミット] を選択します。

2. [Git Staging] タブで、コミットメッセージ `Added Total Amount Calculation` を入力します。
3. [コミットしてプッシュする...] を選択します。
4. CodePipeline コンソールを開き、パイプライン実行の状態を確認します。

> **Note**
>
> Enterprise Developer またはチーム機能のコミットまたはプッシュで問題が発生した場合は、Git Bash コマンドラインインターフェイスを使用してください。

演習 2: BankDemo アプリケーションでのローン計算の抽出

トピック

次の演習では、別のサンプル変更リクエストを処理します。このシナリオでは、ローン部門はローン計算ルーチンをスタンドアロンの WebService として再利用することを考えています。ルーチンは COBOL に残し、既存の CICS COBOL プログラム `BBANK70P.CBL` からも呼び出せるはずです。

ステップ 1: ローン計算ルーチンを COBOL セクションにリファクタリングする

最初のステップでは、ローン計算ルーチンを COBOL セクションに抽出します。このステップは、次のステップでコードをスタンドアロンの COBOL プログラムに抽出するために必要です。

1. COBOL エディタで `BBANK70P.CBL` を開きます。
2. エディタで、コンテキストメニューから [コード分析]/[ローン計算更新] を選択します。これにより、現在のソースのみをスキャンして、分析ルールで定義されているパターンを見つけます。
3. [コード分析] タブの結果から、最初の算術ステートメント `DIVIDE WS-LOAN-INTEREST BY 12` を探します。
4. ステートメントをダブルクリックして、エディタのソース行に移動します。これはローン計算ルーチンの最初のステートメントです。
5. ローン計算ルーチンをセクションに抽出するための次のコードブロックをマークします。

```
DIVIDE WS-LOAN-INTEREST BY 12
             GIVING WS-LOAN-INTEREST ROUNDED.
           COMPUTE WS-LOAN-MONTHLY-PAYMENT ROUNDED =
             ((WS-LOAN-INTEREST * ((1 + WS-LOAN-INTEREST)
                 ** WS-LOAN-TERM)) /
             (((1 + WS-LOAN-INTEREST) * WS-LOAN-TERM) - 1 ))
                 * WS-LOAN-PRINCIPAL.
EXER01     COMPUTE WS-LOAN-TOTAL-PAYMENT =
EXER01               (WS-LOAN-MONTHLY-PAYMENT * WS-LOAN-TERM).
```

6. エディタのコンテキストメニューから [リファクタリング/セクションに抽出...] を選択します。
7. 新しいセクション名: LOAN-CALCULATION を入力します。
8. [OK] を選択します。

   マークされたコードブロックが新しい LOAN-CALCULATION セクションに抽出され、コードブロックが PERFROM LOAN-CALCULATION ステートメントに置き換えられています。

変更のテスト

変更をテストするには、次のセクションで説明する手順を繰り返します。

1. Enterprise Developer から BANKDEMO サーバーを起動する
2. Rumba 3270 ターミナルを起動する
3. BankDemo トランザクションを実行する

   さらに、テキスト Total Loan Amount.....: $10230.60 も表示されるはずです。

4. Enterprise Developer から BANKDEMO サーバーを停止します。

> **Note**
>
> 上記の手順でコードブロックをセクションに抽出しない場合は、ステップ 1 で変更したソースを ..\BANKDEMO-exercise\Exercis02\Step1\cobol\BBANK70P.CBL から D:\PhotonUser\workspace\bankdemo\source\cobol にコピーできます。

ステップ 2: ローン計算ルーチンをスタンドアロンの COBOL プログラムに抽出する

ステップ 2 では、LOAN-CALCULATION セクション内のコードブロックがスタンドアロンプログラムに抽出され、元のコードが新しいサブプログラムを呼び出すコードに置き換えられます。

1. BBANK70P.CBL を エディタで開き、ステップ 1 で作成した新しい PERFORM LOAN-CALCULATION ステートメントを見つけます。
2. セクション名の中にカーソルを置きます。灰色でマークされます。
3. コンテキストメニューから [リファクタリング] → [セクション/段落をプログラムに抽出...] を選択します。
4. [セクション/段落をプログラムに抽出] に、新しいファイル名: LOANCALC.CBL を入力します。
5. [OK] をクリックします。

   新しい LOANCALC.CBL プログラムがエディタで開きます。
6. 下にスクロールして、コールインターフェイス用に抽出および生成されているコードを確認します。
7. BBANK70P.CBL でエディタを選択し、LOAN-CALCULATION SECTION に進みます。生成中のコードを確認して、新しいサブプログラム LOANCALC.CBL を呼び出します。

> **Note**
>
> CALL ステートメントは CICS 制御ブロックとの LOANCALC の呼び出しに DFHEIBLK および DFHCOMMAREA を使用しています。新しい LOANCALC.CBL サブプログラムを CICS 以外のプログラムとして呼び出すので、コメントアウトまたは削除して呼び出しから DFHEIBLK と DFHCOMMAREA を削除する必要があります。

変更のテスト

変更をテストするには、次のセクションで説明する手順を繰り返します。

1. Enterprise Developer から BANKDEMO サーバーを起動する
2. Rumba 3270 ターミナルを起動する
3. BankDemo トランザクションを実行する

   さらに、テキスト Total Loan Amount.....: $10230.60 も表示されるはずです。

4. Enterprise Developer から BANKDEMO サーバーを停止します。

> **Note**
>
> 上記の手順でコードブロックをセクションに抽出しない場合は、ステップ 1 で変更したソースを `..\BANKDEMO-exercise\Exercis02\Step2\cobol\BBANK70P.CBL` と `LOANCALC.CBL` から `D:\PhotonUser\workspace\bankdemo\source\cobol` にコピーできます。

ステップ 3: 変更をコミットして CI/CD パイプラインを実行する

変更を中央の CodeCommit リポジトリにコミットし、CI/CD パイプラインをトリガーして変更をビルド、テスト、デプロイします。

1. BANKDEMO プロジェクトから、コンテキストメニューで [チーム/コミット] を選択します。
2. [Git Staging] タブ
   - Unstaged Stages LOANCALC.CBL と LOANCALC.CBL.mfdirset で追加します。
   - コミットメッセージ `Added Total Amount Calculation` を入力してください。
3. [コミットしてプッシュする...] を選択します。
4. CodePipeline コンソールを開き、パイプライン実行の状態を確認します。

> **Note**
>
> Enterprise Developer またはチーム機能のコミットまたはプッシュで問題が発生した場合は、Git Bash コマンドラインインターフェイスを使用してください。

リソースをクリーンアップする

このチュートリアルで作成したリソースが不要になった場合は、追加料金の発生を避けるため、それらを削除してください。以下のステップを実行します。

- CodePipeline パイプラインを削除します。詳細については、「AWS CodePipeline ユーザーガイド」の「Delete a pipeline in CodePipeline」を参照してください。

- CodeCommit リポジトリを削除します。詳細については、「AWS CodeCommit ユーザーガイド」の「CodeCommit リポジトリを削除する」を参照してください。
- S3; バケットを削除します。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの削除」を参照してください。
- AWS CloudFormation スタックを削除します。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation コンソールのスタックを削除」を参照してください。

AWS Mainframe Modernization のバッチユーティリティ

メインフレームアプリケーションでは、データのソート、FTP によるファイルの転送、DB2 などのデータベースへのデータのロード、データベースからのデータのアンロードなど、特定の機能を実行するためにバッチユーティリティプログラムを使用することがよくあります。

アプリケーションを AWS Mainframe Modernization に移行する場合、メインフレームで使用したものと同じタスクを実行できる機能的に同等の代替ユーティリティが必要です。これらのユーティリティの一部は Mainframe Modernization AWS ランタイムエンジンの一部として既に利用できる場合がありますが、以下の代替ユーティリティを提供しています。

- M2SFTP - SFTP プロトコルを使用して安全なファイル転送を可能にします。
- M2WAIT - 指定した時間待機してから、バッチジョブの次のステップに進みます。
- TXT2PDF - テキストファイルを PDF 形式に変換します。
- M2DFUTIL - メインフレームの ADRDSSU ユーティリティが提供するサポートと同様の、データセットをバックアップ、復元、削除、コピーする機能を提供します。
- M2RUNCMD - Micro Focus のコマンド、スクリプト、システムコールを JCL から直接実行できます。

これらのバッチユーティリティは、お客様からのフィードバックに基づいて開発され、メインフレームユーティリティと同じ機能を提供するように設計されました。目標は、メインフレームから AWS Mainframe Modernization への移行を可能な限りスムーズにすることです。

トピック

- バイナリロケーション
- M2SFTP バッチユーティリティ
- M2WAIT バッチユーティリティ

- TXT2PDF Batch ユーティリティ
- M2DFUTIL バッチユーティリティ
- M2RUNCMD バッチユーティリティ

バイナリロケーション

これらのユーティリティは Micro Focus Enterprise Developer (ED) および Micro Focus Enterprise Server (ES) 製品にあらかじめインストールされています。ED および ES のすべてのバリアントについて、次の場所で入手できます。

- Linux: `/opt/aws/m2/microfocus/utilities/64bit`
- Windows (32 ビット): `C:\AWS\M2\MicroFocus\Utilities\32bit`
- Windows (64 ビット): `C:\AWS\M2\MicroFocus\Utilities\64bit`

M2SFTP バッチユーティリティ

M2SFTP は、Secure File Transfer Protocol (SFTP) を使用してシステム間で安全なファイル転送を実行するように設計された JCL ユーティリティプログラムです。このプログラムは PuTTY SFTP クライアント (`psftp`) を使用して実際のファイル転送を実行します。このプログラムはメインフレームの FTP ユーティリティプログラムと同様に機能し、ユーザー認証とパスワード認証を使用します。

> **Note**
>
> パブリックキー認証はサポートされていません。

メインフレームの FTP JCL を SFTP を使用するように変換するには、`PGM=FTP` を `PGM=M2SFTP` に変更してください。

トピック

- サポートされているプラットフォーム
- 依存関係をインストールする
- AWS Mainframe Modernization Managed 用に M2SFTP を設定する
- Amazon EC2 で Mainframe Modernization ランタイム用に M2SFTP を設定する (AppStream 2.0 を含む) AWS

- サンプル JCL
- PuTTY SFTP (PSFTP) クライアントコマンドリファレンス
- 次のステップ

サポートされているプラットフォーム

M2SFTP は、以下のどのプラットフォームでも使用できます。

- AWS Mainframe Modernization Micro Focus Managed
- Micro Focus ランタイム (Amazon EC2)
- Micro Focus Enterprise Developer (ED) および Micro Focus Enterprise Server (ES) 製品の全バリエーション。

依存関係をインストールする

PuTTY SFTP クライアントを Windows にインストールするには

- PuTTY SFTP クライアントをダウンロードしてインストールします。

PuTTY SFTP クライアントを Linux にインストールするには

- PuTTY SFTP クライアントをインストールするには、以下のコマンドを実行します。

```
sudo yum -y install putty
```

AWS Mainframe Modernization Managed 用に M2SFTP を設定する

移行したアプリケーションが AWS Mainframe Modernization Managed で実行されている場合は、次のように M2SFTP を設定する必要があります。

- MFFTP に適切な Micro Focus Enterprise Server 環境変数を設定します。ここにいくつか例を挙げます。
  - `MFFTP_TEMP_DIR`
  - `MFFTP_SENDEOL`

- MFFTP_TIME
- MFFTP_ABEND

これらの変数は必要な数だけ設定することも、多く設定することもできます。これらの変数は、ENVAR DD ステートメントを使用して JCL で設定できます。これらの変数の詳細については、「Micro Focus ドキュメント」の「MFFTP Control Variables」を参照してください。

設定をテストするには、「サンプル JCL」を参照してください。

Amazon EC2 で Mainframe Modernization ランタイム用に M2SFTP を設定する (AppStream 2.0 を含む) AWS

移行したアプリケーションが Amazon EC2 の AWS Mainframe Modernization ランタイムで実行されている場合は、次のように M2SFTP を設定します。

1. Micro Focus JES プログラムパスをバッチユーティリティのバイナリロケーションが含まれるように変更します。複数のパスを指定する必要がある場合、Linux ではコロン (:) を使用してパスを区切り、Windows ではセミコロン (;) を使用してパスを区切ります。

   - Linux: /opt/aws/m2/microfocus/utilities/64bit
   - Windows (32 ビット): C:\AWS\M2\MicroFocus\Utilities\32bit
   - Windows (64 ビット): C:\AWS\M2\MicroFocus\Utilities\64bit

2. MFFTP に適切な Micro Focus Enterprise Server 環境変数を設定します。ここにいくつか例を挙げます。

   - MFFTP_TEMP_DIR
   - MFFTP_SENDEOL
   - MFFTP_TIME
   - MFFTP_ABEND

   これらの変数は必要な数だけ設定することも、多く設定することもできます。これらの変数は、ENVAR DD ステートメントを使用して JCL で設定できます。これらの変数の詳細については、「Micro Focus ドキュメント」の「MFFTP Control Variables」を参照してください。

設定をテストするには、「サンプル JCL」を参照してください。

サンプル JCL

インストールのテストには、以下のサンプル JCL ファイルのいずれかを使用します。

M2SFTP1.jcl

この JCL は、M2SFTP を呼び出してリモート SFTP サーバーにファイルを送信する方法を示しています。ENVVAR DD ステートメントに設定されている環境変数に注目してください。

```
//M2SFTP1 JOB 'M2SFTP1',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*--------------------------------------------------------------------**
//* Sample SFTP JCL step to send a file to SFTP server*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2SFTP,
//            PARM='127.0.0.1 (EXIT=99 TIMEOUT 300'
//*
//SYSFTPD  DD  *
RECFM FB
LRECL 80
SBSENDEOL CRLF
MBSENDEOL CRLF
TRAILINGBLANKS FALSE
/*
//NETRC    DD  *
machine 127.0.0.1 login sftpuser password sftppass
/*
//SYSPRINT DD  SYSOUT=*
//OUTPUT   DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//INPUT    DD  *
type a
locsite notrailingblanks
cd files
put 'AWS.M2.TXT2PDF1.PDF' AWS.M2.TXT2PDF1.pdf
put 'AWS.M2.CARDDEMO.CARDDATA.PS' AWS.M2.CARDDEMO.CARDDATA.PS1.txt
quit
/*
//ENVVAR   DD *
```

```
MFFTP_VERBOSE_OUTPUT=ON
MFFTP_KEEP=N
/*
//*
//
```

M2SFTP2.jcl

この JCL は、M2SFTP を呼び出してリモート SFTP サーバーからファイルを受信する方法を示しています。ENVVAR DD ステートメントに設定されている環境変数に注目してください。

```
//M2SFTP2 JOB 'M2SFTP2',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*--------------------------------------------------------------------**
//* Sample SFTP JCL step to receive a file from SFTP server*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2SFTP
//*
//SYSPRINT DD  SYSOUT=*
//OUTPUT   DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//INPUT    DD  *
open 127.0.0.1
sftpuser
sftppass
cd files
locsite recfm=fb lrecl=150
get AWS.M2.CARDDEMO.CARDDATA.PS.txt +
'AWS.M2.CARDDEMO.CARDDATA.PS2' (replace
quit
/*
//ENVVAR   DD *
MFFTP_VERBOSE_OUTPUT=ON
MFFTP_KEEP=N
/*
//*
//
```

> **Note**
>
> FTP 認証情報を NETRC ファイルに保存し、アクセスを許可されたユーザーのみに制限することを強くお勧めします。

PuTTY SFTP (PSFTP) クライアントコマンドリファレンス

PSFTP クライアントはすべての FTP コマンドをサポートしているわけではありません。次のリストは、PSFTP がサポートするすべてのコマンドを示しています。

コマンド	説明
!	ローカルコマンドを実行します
bye	SFTP セッションを終了します
cd	リモート作業ディレクトリを変更します
chmod	ファイル権限とモードを変更します
close	SFTP セッションを終了しますが、PSFTP は終了しません
del	リモートサーバーのファイルを削除します
dir	リモートファイルを一覧表示します
exit	SFTP セッションを終了します
get	サーバーのファイルをローカルマシンにダウンロードします
help	ヘルプを表示します
lcd	ローカル作業ディレクトリを変更します
lpwd	ローカル作業ディレクトリを表示します
ls	リモートファイルを一覧表示します

コマンド	説明
mget	複数のファイルを一度にダウンロードします
mkdir	リモートサーバーにディレクトリを作成します
mput	複数のファイルを一度にアップロードします
mv	リモートサーバーのファイルを移動または名前変更します
open	ホストに接続します
put	ローカルマシンのファイルをサーバーにアップロードします
pwd	リモート作業ディレクトリを表示します
quit	SFTP セッションを終了します
reget	ファイルのダウンロードを続けます
ren	リモートサーバーのファイルを移動または名前変更します
reput	ファイルのアップロードを続けます
rm	リモートサーバーのファイルを削除します
rmdir	リモートサーバーのディレクトリを削除します

次のステップ

SFTP を使用して Amazon Simple Storage Service にファイルをアップロードおよびダウンロードするには、次のブログ記事で説明されているように AWS Transfer Family、 と組み合わせて M2SFTP を使用できます。

- AWS SFTP 論理ディレクトリを使用して単純なデータ分散サービスを構築する
- AWS Transfer for SFTP を使用して のパスワード認証を有効にする AWS Secrets Manager

M2WAIT バッチユーティリティ

M2WAIT は、秒単位、分単位、または時間単位で時間を指定して JCL スクリプトに待機時間を導入できるメインフレームユーティリティプログラムです。待機する時間を入力パラメータとして渡すことにより、JCL から直接 M2WAIT を呼び出せます。内部では、M2WAIT プログラムは Micro Focus 提供のモジュール C$SLEEP を呼び出して、指定された時間待機します。

> **Note**
>
> Micro Focus エイリアスを使用して JCL スクリプトにあるものを置き換えられます。詳細については、「Micro Focus ドキュメント」の「JES Alias」を参照してください。

トピック

サポートされているプラットフォーム

M2WAIT は以下のどのプラットフォームでも使用できます。

- AWS Mainframe Modernization Micro Focus Managed
- Micro Focus ランタイム (Amazon EC2)
- Micro Focus Enterprise Developer (ED) および Micro Focus Enterprise Server (ES) 製品の全バリエーション。

AWS Mainframe Modernization Managed 用に M2WAIT を設定する

移行したアプリケーションが AWS Mainframe Modernization Managed で実行されている場合は、次のように M2WAIT を設定する必要があります。

- 「サンプル JCL」に示すように、入力パラメータを渡して JCL で M2WAIT プログラムを使用します。

Amazon EC2 で Mainframe Modernization ランタイム用に M2WAIT を設定する (AppStream 2.0 を含む) AWS

移行したアプリケーションが Amazon EC2 の AWS Mainframe Modernization ランタイムで実行されている場合は、次のように M2WAIT を設定します。

1. Micro Focus JES プログラムパスをバッチユーティリティのバイナリロケーションが含まれるように変更します。複数のパスを指定する必要がある場合、Linux ではコロン (:) を使用してパスを区切り、Windows ではセミコロン (;) を使用してパスを区切ります。

   - Linux: `/opt/aws/m2/microfocus/utilities/64bit`
   - Windows (32 ビット): `C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows (64 ビット): `C:\AWS\M2\MicroFocus\Utilities\64bit`

2. 「サンプル JCL」に示すように、入力パラメータを渡して JCL で M2WAIT プログラムを使用します。

サンプル JCL

インストールをテストするには、`M2WAIT1.jcl` プログラムを使用できます。

このサンプル JCL は、M2WAIT を呼び出して複数の異なる期間を渡す方法を示しています。

```
//M2WAIT1 JOB 'M2WAIT',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*-------------------------------------------------------------------**
//* Wait for 12 Seconds*
//*-------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2WAIT,PARM='S012'
//SYSOUT DD SYSOUT=*
//*
//*-------------------------------------------------------------------**
//* Wait for 0 Seconds (defaulted to 10 Seconds)*
//*-------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=M2WAIT,PARM='S000'
//SYSOUT DD SYSOUT=*
```

```
//*
//*-------------------------------------------------------------------**
//* Wait for 1 Minute*
//*-------------------------------------------------------------------**
//*
//STEP03 EXEC PGM=M2WAIT,PARM='M001'
//SYSOUT DD SYSOUT=*
//*
//
```

TXT2PDF Batch ユーティリティ

TXT2PDF は、テキストファイルを PDF ファイルに変換するために一般的に使用されるメインフレームユーティリティプログラムです。このユーティリティには TXT2PDF (z/OS フリーウェア) と同じソースコードを使用します。 AWS Mainframe Modernization Micro Focus ランタイム環境で実行するように変更しました。

トピック

- サポートされているプラットフォーム
- AWS Mainframe Modernization Managed 用に TXT2PDF を設定する
- Amazon EC2 で Mainframe Modernization ランタイムの TXT2PDF を設定する (AppStream 2.0 を含む) AWS
- サンプル JCL
- 変更
- リファレンス

サポートされているプラットフォーム

以下のどのプラットフォームでも TXT2PDF を使用できます。

- AWS Mainframe Modernization Micro Focus Managed
- Micro Focus ランタイム (Amazon EC2)
- Micro Focus Enterprise Developer (ED) および Micro Focus Enterprise Server (ES) 製品の全バリエーション。

AWS Mainframe Modernization Managed 用に TXT2PDF を設定する

移行したアプリケーションが AWS Mainframe Modernization Managed で実行されている場合は、TXT2PDF を次のように設定します。

- `AWS.M2.REXX.EXEC` という名前の REXX EXEC ライブラリを作成します。これらの REXX モジュールをダウンロードし、ライブラリにコピーします。
  - `TXT2PDF.rex` - TXT2PDF z/OS フリーウェア (修正済み)
  - `TXT2PDFD.rex` - TXT2PDF z/OS フリーウェア (未修正)
  - `TXT2PDFX.rex` - TXT2PDF z/OS フリーウェア (修正済み)
  - `M2GETOS.rex` - OS の種類の確認や (Windows または Linux)

設定をテストするには、「サンプル JCL」を参照してください。

Amazon EC2 で Mainframe Modernization ランタイムの TXT2PDF を設定する (AppStream 2.0 を含む) AWS

移行したアプリケーションが Amazon EC2 の AWS Mainframe Modernization ランタイムで実行されている場合は、TXT2PDF を次のように設定します。

1. Micro Focus 環境変数 `MFREXX_CHARSET` を適切な値 (ASCII データの場合は「`A`」など) に設定します。

   > **Important**
   >
   > 正しくない値を入力すると、データ変換 (EBCDIC から ASCII) の問題が発生し、生成される PDF が読めなくなる可能性や、操作できなくなる可能性があります。`MFREXX_CHARSET` と `MF_CHARSET` が一致するように設定することをお勧めします。

2. Micro Focus JES プログラムパスをバッチユーティリティのバイナリロケーションが含まれるように変更します。複数のパスを指定する必要がある場合、Linux ではコロン (`:`) を使用してパスを区切り、Windows ではセミコロン (`;`) を使用してパスを区切ります。
   - Linux: `/opt/aws/m2/microfocus/utilities/64bit`
   - Windows (32 ビット): `C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows (64 ビット): `C:\AWS\M2\MicroFocus\Utilities\64bit`

3. AWS.M2.REXX.EXEC` という名前の REXX EXEC ライブラリを作成します。これらの REXX モジュールをダウンロードし、ライブラリにコピーします。

   - TXT2PDF.rex - TXT2PDF z/OS フリーウェア (修正済み)
   - TXT2PDFD.rex - TXT2PDF z/OS フリーウェア (未修正)
   - TXT2PDFX.rex - TXT2PDF z/OS フリーウェア (修正済み)
   - M2GETOS.rex - OS の種類の確認や (Windows または Linux)

設定をテストするには、「サンプル JCL」を参照してください。

サンプル JCL

インストールのテストには、以下のサンプル JCL ファイルのいずれかを使用します。

TXT2PDF1.jcl

このサンプル JCL ファイルでは、TXT2PDF の変換に DD 名を使用しています。

```
//TXT2PDF1 JOB 'TXT2PDF1',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*--------------------------------------------------------------------**
//* PRE DELETE*
//*--------------------------------------------------------------------**
//*
//PREDEL  EXEC PGM=IEFBR14
//*
//DD01     DD DSN=AWS.M2.TXT2PDF1.PDF.VB,
//            DISP=(MOD,DELETE,DELETE)
//*
//DD02     DD DSN=AWS.M2.TXT2PDF1.PDF,
//            DISP=(MOD,DELETE,DELETE)
//*
//*--------------------------------------------------------------------**
//* CALL TXT2PDF TO CONVERT FROM TEXT TO PDF (VB)*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=IKJEFT1B
//*
//SYSEXEC  DD DISP=SHR,DSN=AWS.M2.REXX.EXEC
```

```
//*
//INDD     DD *
1THIS IS THE FIRST LINE ON THE PAGE 1
0THIS IS THE THIRD LINE ON THE PAGE 1
-THIS IS THE   6TH LINE ON THE PAGE 1
THIS IS THE   7TH LINE ON THE PAGE 1
+_____________________________________ - OVERSTRIKE 7TH LINE
1THIS IS THE FIRST LINE ON THE PAGE 2
0THIS IS THE THIRD LINE ON THE PAGE 2
-THIS IS THE   6TH LINE ON THE PAGE 2
THIS IS THE   7TH LINE ON THE PAGE 2
+_____________________________________ - OVERSTRIKE 7TH LINE
/*
//*
//OUTDD    DD DSN=AWS.M2.TXT2PDF1.PDF.VB,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=VB,BLKSIZE=0)
//*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=SYSIN
//*
//SYSIN    DD *
%TXT2PDF BROWSE Y IN DD:INDD +
OUT DD:OUTDD +
CC YES
/*
//*
//*--------------------------------------------------------------------**
//* CONVERT PDF (VB) TO PDF (LSEQ - BYTE STREAM)*
//*--------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=VB2LSEQ
//*
//INFILE   DD DSN=AWS.M2.TXT2PDF1.PDF.VB,DISP=SHR
//*
//OUTFILE  DD DSN=AWS.M2.TXT2PDF1.PDF,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=LSEQ,BLKSIZE=0)
//*
//SYSOUT   DD SYSOUT=*
//*
//
```

TXT2PDF2.jcl

このサンプル JCL では、TXT2PDF の変換に DSN 名を使用しています。

```
//TXT2PDF2 JOB 'TXT2PDF2',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*--------------------------------------------------------------------**
//* PRE DELETE*
//*--------------------------------------------------------------------**
//*
//PREDEL  EXEC PGM=IEFBR14
//*
//DD01     DD DSN=AWS.M2.TXT2PDF2.PDF.VB,
//            DISP=(MOD,DELETE,DELETE)
//*
//DD02     DD DSN=AWS.M2.TXT2PDF2.PDF,
//            DISP=(MOD,DELETE,DELETE)
//*
//*--------------------------------------------------------------------**
//* CALL TXT2PDF TO CONVERT FROM TEXT TO PDF (VB)*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=IKJEFT1B
//*
//SYSEXEC  DD DISP=SHR,DSN=AWS.M2.REXX.EXEC
//*
//INDD     DD *
1THIS IS THE FIRST LINE ON THE PAGE 1
0THIS IS THE THIRD LINE ON THE PAGE 1
-THIS IS THE   6TH LINE ON THE PAGE 1
THIS IS THE   7TH LINE ON THE PAGE 1
+_____________________________________ - OVERSTRIKE 7TH LINE
1THIS IS THE FIRST LINE ON THE PAGE 2
0THIS IS THE THIRD LINE ON THE PAGE 2
-THIS IS THE   6TH LINE ON THE PAGE 2
THIS IS THE   7TH LINE ON THE PAGE 2
+_____________________________________ - OVERSTRIKE 7TH LINE
/*
//*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=SYSIN
```

```
//*
//SYSIN    DD *
%TXT2PDF BROWSE Y IN DD:INDD +
OUT 'AWS.M2.TXT2PDF2.PDF.VB' +
CC YES
/*
//*
//*-------------------------------------------------------------------**
//* CONVERT PDF (VB) TO PDF (LSEQ - BYTE STREAM)*
//*-------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=VB2LSEQ
//*
//INFILE   DD DSN=AWS.M2.TXT2PDF2.PDF.VB,DISP=SHR
//*
//OUTFILE  DD DSN=AWS.M2.TXT2PDF2.PDF,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=LSEQ,BLKSIZE=0)
//*
//SYSOUT   DD SYSOUT=*
//*
//
```

変更

TXT2PDF プログラムを AWS Mainframe Modernization Micro Focus ランタイム環境で実行するために、次の変更を加えました。

- Micro Focus REXX ランタイムとの互換性を確保するためのソースコードの変更。
- Windows と Linux 両方のオペレーティングシステムでプログラムを実行できるようにするための変更
- EBCDIC と ASCII ランタイム両方をサポートするための変更

リファレンス

TXT2PDF リファレンスとソースコード:

- テキストから PDF へのコンバータ
- z/OS フリーウェア、TCP/IP、メールツール
- TXT2PDF ユーザーリファレンスガイド

M2DFUTIL バッチユーティリティ

M2DFUTIL は、メインフレーム ADRDSSU ユーティリティが提供するサポートと同様に、データセットのバックアップ、復元、削除、コピー機能を提供する JCL ユーティリティプログラムです。このプログラムは ADRDSSU の SYSIN パラメータの多くを保持するため、この新しいユーティリティへの移行プロセスが効率化されます。

トピック

サポートされているプラットフォーム

M2DFUTIL は、以下のどのプラットフォームでも使用できます。

- Windows (64 ビットと 32 ビット) 上の Micro Focus ES
- Linux (64 ビット) 上の Micro Focus ES

プラットフォームの要件

M2DFUTIL は、正規表現テストを実行するスクリプトの呼び出しに依存します。Windows では、このスクリプトを実行するには Windows Services for Linux (WSL) をインストールする必要があります。

今後のサポートの予定

以下を含む機能は、現在メインフレームの ADRDSSU ユーティリティでは使用できませんが、今後サポートを予定しています。

- M2 Managed

- VSAM
- ファイルの名前変更での COPY のサポート
- RESTORE での RENAME のサポート
- 複数の INCLUDE と EXCLUDE
- DSORG、CREDT、EXPDT による副選択のための BY 句
- エンキュー失敗を再試行するための MWAIT 句
- DUMP/RESTORE での S3 ストレージのサポート

アセットの場所

このユーティリティのロードモジュールは、Linux では `M2DFUTIL.dll`、Windows では `M2DFUTIL.so` と呼ばれます。このロードモジュールは、次のいずれかの場所にあります。

- Linux: `/opt/aws/m2/microfocus/utilities/64bit`
- Windows (32 ビット): `C:\AWS\M2\MicroFocus\Utilities\32bit`
- Windows (64 ビット): `C:\AWS\M2\MicroFocus\Utilities\64bit`

正規表現のテストに使用されるスクリプトは `compare.sh` と呼ばれます。このスクリプトは、次のいずれかの場所にあります。

- Linux: `/opt/aws/m2/microfocus/utilities/scripts`
- Windows (32 ビット): `C:\AWS\M2\MicroFocus\Utilities\scripts`

Amazon EC2 で M2DFUTIL または AWS Mainframe Modernization ランタイムを設定する (AppStream 2.0 を含む)

Enterprise Server のリージョンを次のように設定します。

- [ES 環境] に以下の変数を追加します。
  - `M2DFUTILS_BASE_LOC` - DUMP 出力のデフォルト場所
  - `M2DFUTILS_SCRIPTPATH` - 「アセットの場所」に記載されている `compare.sh` スクリプトの場所
  - `M2DFUTILS_VERBOSE` - [冗長または標準]。これは `SYSPRINT`　出力の詳細レベルを制御します。

- ロードモジュールパスが JES\Configuration\JES Program Path 設定に追加されていることを確認します。
- ユーティリティディレクトリ内のスクリプトに実行権限があることを確認します。Linux 環境では、chmod + x <script name> コマンドを使用して実行権限を追加できます。

一般的な構文

DUMP

現在のカタログの場所からバックアップの場所にファイルをコピーできます。この場所は、現時点でファイルシステムである必要があります。

プロセス

DUMP は以下を実行します。

1. ターゲットロケーションディレクトリを作成する。
2. ターゲットロケーションディレクトリを PDS メンバーとしてカタログ化する。
3. INCLUDE パラメータを処理して、インクルードするファイルを特定する。
4. EXCLUDE パラメータを処理して、インクルードされたファイルの選択を解除する。
5. ダンプ中のファイルを削除するかどうかを判断する。
6. 処理するファイルをエンキューする。
7. ファイルをコピーする。
8. 今後の RESTORE オペレーションのため、コピーしたファイルのカタログ化された DCB 情報をターゲットロケーションのサイドファイルにエクスポートする。

Syntax

```
DUMP
TARGET ( TARGET LOCATION  )    -
INCLUDE ( DSN. )
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ DELETE ]
```

必須パラメータ

以下は DUMP の必須パラメータです。

- SYSPRINT DD NAME - 追加の口ギング情報を格納するため
- TARGET - ターゲット口ケーションョン 以下のいずれかになります。
  - ダンプロケーションのフルパス
  - M2DFUTILS_BASE_LOC 変数で定義された場所に作成されたサブディレクトリ名
- INCLUDE - 単名の DSNAME または有効なメインフレーム DSN 検索文字列のいずれか
- EXCLUDE - 単名の DSNAME または有効なメインフレーム DSN 検索文字列のいずれか

任意指定のパラメータ

- CANCEL - エラーが発生した場合はキャンセルします。処理済みのファイルは保持されます。
- (デフォルト) IGNORE - すべてのエラーを無視し、最後まで処理します
- DELETE - ENQ エラーが発生しない場合、ファイルは削除され、カタログ化が解除されます。

DELETE

ファイルの一括削除およびカタログ化の解除ができます。ファイルはバックアップされません。

プロセス

DELETE は以下を実行します。

1. INCLUDE パラメータを処理して、インクルードするファイルを特定する。
2. EXCLUDE パラメータを処理して、インクルードされたファイルの選択を解除する。
3. 処理するファイルをエンキューする。処理を OLD、DELETE、KEEP に設定する。

Syntax

```
DELETE
INCLUDE ( DSN )
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ DELETE ]
```

必須パラメータ

以下は DELETE の必須パラメータです。

- SYSPRINT DD NAME - 追加の口ギング情報を格納するため
- INCLUDE - 単名の DSNAME または有効なメインフレーム DSN 検索文字列のいずれか
- EXCLUDE - 単名の DSNAME または有効なメインフレーム DSN 検索文字列のいずれか

任意指定のパラメータ

- CANCEL - エラーが発生した場合はキャンセルします。処理済みのファイルは保持されます。
- (デフォルト) IGNORE - すべてのエラーを無視し、最後まで処理します

RESTORE

DUMP を使用して以前にバックアップしたファイルを復元できます。RENAME を使用して復元済みの DSNAME を変更した場合を除き、ファイルはカタログ化された元の場所に復元されます。

プロセス

RESTORE は以下を実行します。

1. ソースロケーションディレクトリを検証する。
2. カタログエクスポートファイルを処理して、インクルードするファイルを特定する。
3. EXCLUDE パラメータを処理して、インクルードされたファイルの選択を解除する。
4. 処理するファイルをエンキューする。
5. エクスポート情報に基づいてカタログ化されていないファイルをカタログ化する。
6. ファイルが既にカタログ化されていて、エクスポートカタログ情報が同じ場合、REPLACE オプションが設定されている場合は、RESTORE はカタログ化されたデータセットを置き換えます。

Syntax

```
RESTORE
SOURCE ( TARGET LOCATION )
INCLUDE ( DSN )
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ REPLACE]
```

必須パラメータ

以下は RESTORE の必須パラメータです。

- SYSPRINT DD NAME - 追加のロギング情報を格納するため
- SOURCE - ソースロケーション。以下のいずれかになります。
  - ダンプロケーションのフルパス
  - M2DFUTILS_BASE_LOC 変数で定義された場所に作成されたサブディレクトリ名
- INCLUDE - 単名の DSNAME または有効なメインフレーム DSN 検索文字列のいずれか
- EXCLUDE - 単名の DSNAME または有効なメインフレーム DSN 検索文字列のいずれか

任意指定のパラメータ

- CANCEL - エラーが発生した場合はキャンセルします。処理済みのファイルは保持されます。
- (デフォルト) IGNORE - すべてのエラーを無視し、最後まで処理します
- REPLACE - ファイルが既にカタログ化されていて、カタログレコードが同じ場合は、カタログ化されたファイルを置き換えます。

サンプル JCL

DUMP ジョブ

このジョブは TESTDUMP というサブディレクトリを作成します。これは M2DFUTILS_BASE_LOC 変数によって指定されるデフォルトのバックアップ場所です。このバックアップ用に、M2DFUTILS.TESTDUMP という PDS ライブラリが作成されます。エクスポートされたカタログデータは、CATDUMP.DAT というバックアップディレクトリの行順ファイルに保存されます。選択したすべてのファイルが、このバックアップディレクトリにコピーされます。

```
//M2DFDMP JOB 'M2DFDMP',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
//SYSPRINT DD DSN=TESTDUMP.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSIN    DD *
DUMP TARGET(TESTDUMP)               -
     INCLUDE(TEST.FB.FILE*.ABC)     -
 CANCEL
```

```
/*
//
```

DELETE ジョブ

このジョブは、INCLUDE パラメータに一致するすべてのファイルをカタログから削除します。

```
/M2DFDEL JOB 'M2DFDEL',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
//SYSPRINT DD DSN=TESTDEL.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
  DELETE                                -
     INCLUDE(TEST.FB.FILE*.ABC)         -
 CANCEL
/*
 //
```

RESTORE ジョブ

このジョブは、INCLUDE パラメータに一致するファイルを TESTDUMP バックアップ場所から復元します。カタログ化されたファイルが CATDUMP エクスポートのファイルと同じで、REPLACE オプションが指定されている場合、カタログ化されたファイルは置き換えられます。

```
//M2DFREST JOB 'M2DFREST',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
////SYSPRINT DD DSN=TESTREST.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
RESTORE SOURCE(TESTDUMP)                -
     INCLUDE(TEST.FB.FILE*.ABC)         -
 IGNORE
 REPLACE
/*
//
```

M2RUNCMD バッチユーティリティ

バッチユーティリティプログラムである M2RUNCMD を使用すると、Micro Focus のコマンド、スクリプト、システム呼び出しを、ターミナルやコマンドプロンプトから実行する代わりに、JCL から直接実行できます。コマンドからの出力は、バッチジョブのスプールログに記録されます。

トピック

- サポートされているプラットフォーム
- Amazon EC2 で Mainframe Modernization ランタイム用に M2RUNCMD を設定する (AppStream 2.0 を含む) AWS
- サンプル JCL

サポートされているプラットフォーム

M2RUNCMD は、以下のどのプラットフォームでも使用できます。

- Micro Focus ランタイム (Amazon EC2)
- Micro Focus Enterprise Developer (ED) および Micro Focus Enterprise Server (ES) 製品の全バリエーション。

Amazon EC2 で Mainframe Modernization ランタイム用に M2RUNCMD を設定する (AppStream 2.0 を含む) AWS

移行したアプリケーションが Amazon EC2 の AWS Mainframe Modernization ランタイムで実行されている場合は、次のように M2RUNCMD を設定します。

- Micro Focus JES プログラムパスをバッチユーティリティのバイナリロケーションが含まれるように変更します。複数のパスを指定する必要がある場合、Linux ではコロン (:) を使用してパスを区切り、Windows ではセミコロン (;) を使用してパスを区切ります。
  - Linux: `/opt/aws/m2/microfocus/utilities/64bit`
  - Windows (32 ビット): `C:\AWS\M2\MicroFocus\Utilities\32bit`
  - Windows (64 ビット): `C:\AWS\M2\MicroFocus\Utilities\64bit`

サンプル JCL

インストールのテストには、以下のサンプル JCL のいずれかを使用できます。

RUNSCRL1.jcl

このサンプル JCL はスクリプトを作成して実行します。最初のステップでは、SYSUT1 インストリームデータからのコンテンツを使用して、/tmp/TEST_SCRIPT.sh というスクリプトを作成します。2 番目のステップでは、実行権限を設定し、最初のステップで作成したスクリプトを実行します。また、2 番目のステップだけを実行して、既存の Micro Focus コマンドやシステムコマンドを実行することもできます。

```
//RUNSCRL1 JOB 'RUN SCRIPT',CLASS=A,MSGCLASS=X,TIME=1440
//*
//*
//*--------------------------------------------------------------------*
//*  CREATE SCRIPT (LINUX)
//*--------------------------------------------------------------------*
//*
//STEP0010 EXEC PGM=IEBGENER
//*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//*
//SYSUT1   DD *
#!/bin/bash

set -x

## ECHO PATH ENVIRONMNET VARIABLE
echo $PATH

## CLOSE/DISABLE VSAM FILE
casfile -r$ES_SERVER -oc  -ed -dACCTFIL

## OPEN/ENABLE VSAM FILE
casfile -r$ES_SERVER -ooi -ee -dACCTFIL

exit $?
/*
//SYSUT2   DD DSN=&&TEMP,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(RECFM=LSEQ,LRECL=300,DSORG=PS,BLKSIZE=0)
//*MFE: %PCDSN='/tmp/TEST_SCRIPT.sh'
//*
//*--------------------------------------------------------------------*
//*   RUN SCRIPT (LINUX)                                               *
```

```
//*-------------------------------------------------------------------*
//*
//STEP0020 EXEC PGM=RUNCMD
//*
//SYSOUT  DD  SYSOUT=*
//*
//SYSIN   DD *
*RUN SCRIPT
 sh /tmp/TEST_SCRIPT.sh
/*
//
```

SYSOUT

実行されたコマンドまたはスクリプトからの出力は SYSOUT ログに書き込まれます。実行されたコマンドごとに、コマンド、出力、リターンコードが表示されます。

```
************ CMD Start ************

CMD_STR: sh /tmp/TEST_SCRIPT.sh

CMD_OUT:


+ echo /opt/microfocus/EnterpriseServer/bin:/sbin:/bin:/usr/sbin:/usr/bin
/opt/microfocus/EnterpriseServer/bin:/sbin:/bin:/usr/sbin:/usr/bin
+ casfile -rMYDEV -oc -ed -dACCTFIL

-Return Code:   0

Highest return code:    0

+ casfile -rMYDEV -ooi -ee -dACCTFIL

-Return Code:   8

Highest return code:    8

+ exit 8


CMD_RC=8
```

```
************  CMD End  ************
```

RUNCMDL1.jcl

このサンプル JCL は RUNCMD を使用して複数のコマンドを実行します。

```
//RUNCMDL1 JOB 'RUN CMD',CLASS=A,MSGCLASS=X,TIME=1440
//*
//*
//*--------------------------------------------------------------------*
//*   RUN SYSTEM COMMANDS                                              *
//*--------------------------------------------------------------------*
//*
//STEP0001 EXEC PGM=RUNCMD
//*
//SYSOUT  DD  SYSOUT=*
//*
//SYSIN   DD *
*LIST DIRECTORY
 ls
*ECHO PATH ENVIRONMNET VARIABLE
 echo $PATH
/*
//
```

AWS Precisely による Mainframe Modernization データレプリケーション

AWS Mainframe Modernization には、さまざまな Amazon マシンイメージ (AMIs。これらの AMIs、Amazon EC2 インスタンスの迅速なプロビジョニングを容易にし、メインフレームシステムから Precisely AWS を使用するまでのデータレプリケーションのためのカスタマイズされた環境を作成します。このガイドでは、これらの AMI にアクセスして使用するために必要な手順について説明します。

前提条件

- Amazon EC2 インスタンスを作成できる AWS アカウントへの管理者アクセス権があることを確認します。
- AWS Mainframe Modernization サービスが、Amazon EC2 インスタンスを作成する予定のリージョンで利用可能であることを確認します。「リージョン別に利用可能な AWS サービスのリスト」を参照してください。
- Amazon EC2 インスタンスを作成する Amazon Virtual Private Cloud (Amazon VPC) を特定します。
- Amazon VPC に Amazon EC2 インスタンスを作成するときは、関連するルートテーブルにインターネットゲートウェイまたは NAT ゲートウェイがあることを確認してください。

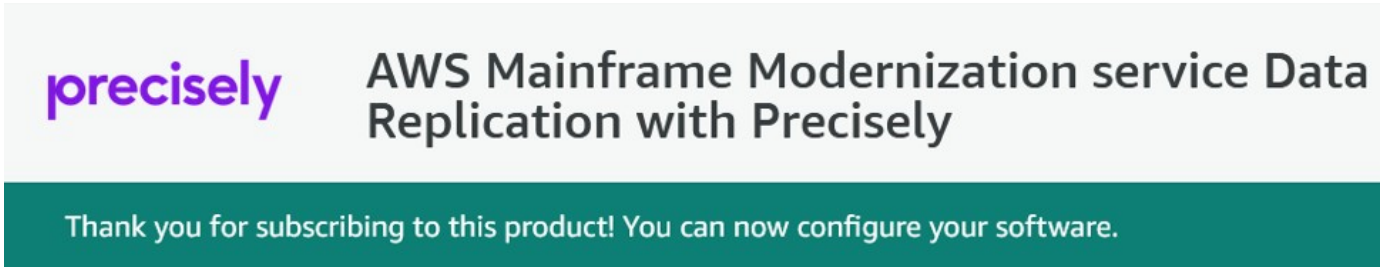
> **Note**
>
> 正常にデータレプリケーションが行われるには、AWS EC2 インスタンスが AWS Marketplace に通信できる必要があります。AWS Marketplace との接続に問題がある場合、レプリケーションプロセスは失敗します。

Amazon マシンイメージのサブスクライブ

AWS Marketplace 製品をサブスクライブすると、その製品の AMI からインスタンスを起動できるようになります。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/marketplace で AWS Marketplace コンソールを開きます。

2. [サブスクリプションを管理する] を選択します。
3. 次のリンクをコピーしてブラウザのアドレスバーに貼り付けます: https://aws.amazon.com/marketplace/pp/prodview-en3xrbgzbs3dk
4. [サブスクリプションを続行する] を選択します。
5. 利用規約に同意できる場合は、[規約の受諾] を選択します。このサブスクリプションの処理には数分かかる場合があります。
6. 以下のような、お礼のメッセージが表示されるまで待ちます。このメッセージは、製品へのサブスクリプションが完了したことを確認するものです。



7. 左のナビゲーションペインで、[サブスクリプションの管理] をクリックします。このビューには、サブクライブしているサブスクリプションがすべて表示されます。

Precisely で AWS Mainframe Modernization データレプリケーションを起動する

1. https://console.aws.amazon.com/marketplace で AWS Marketplace コンソールを開きます。
2. 左のナビゲーションペインで、[サブスクリプションの管理] をクリックします。
3. 起動する AMI を見つけて、[新規インスタンスを起動] をクリックします。
4. [リージョン] で、許可リストに登録されているリージョンを選択します。
5. [EC2 経由で起動を続行する] を選択します。これにより、Amazon EC2 コンソールに移動します。
6. サーバーの名前を入力します。
7. プロジェクトのパフォーマンスとコスト要件に合ったインスタンスタイプを選択します。インスタンスサイズの推奨開始点は `c5.2xLarge` です。
8. 既存のキーペアを選択するか、新しいキーペアを作成して保存します。キーペアの詳細については、「Amazon EC2 Linux インスタンス用ユーザーガイド」の「Amazon EC2 のキーペアと Linux インスタンス」を参照してください。
9. ネットワーク設定を編集し、許可リストに登録された VPC と適切なサブネットを選択します。

10. 既存のセキュリティグループを選択するか、新しいセキュリティグループを作成します。Precisely サーバーの EC2 インスタンスを使用したデータレプリケーションの場合、SSH アクセス (デフォルトではポート 22) を許可するほか、通常はデフォルトポート 2626 への TCP トラフィックを許可します。
11. Amazon EC2 インスタンスのストレージを設定します。
12. 概要を確認し、[インスタンスを起動] をクリックします。起動を成功させるには、インスタンスタイプが有効である必要があります。起動に失敗した場合は、[インスタンス設定を編集] をクリックし、別のインスタンスタイプを選択します。
13. 成功メッセージが表示されたら、[インスタンスに接続] をクリックします。
14. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
15. 左側のナビゲーションペインの [インスタンス] メニューで、[インスタンス] をクリックします。
16. メインペインで、インスタンスのステータスをチェックします。

IAM ポリシーを作成する

AWS Marketplace リストに従ってデプロイされた AWS Mainframe Modernization EC2 インスタンスを正常に運用するには、IAM ロールとポリシーを設定する必要があります。この特別にカスタマイズされた IAM セットアップはオプションではなく、Amazon EC2 インスタンスが AWS Marketplace サービスとやり取りすることを許可します。IAM ロールとポリシーにより、 AWS Mainframe Modernization は正確な請求に不可欠な使用状況データを正確に記録できます。この構成を実装しないと、データレプリケーションの試行に失敗し、運用が中断される可能性があります。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。
3. [ポリシー] を初めて選択する場合は、[管理ポリシーにようこそ] ページが表示されます。[今すぐ始める] を選択します。
4. ページの上部で、[ポリシーを作成] を選択します。
5. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
6. 以下の JSON ポリシーを入力します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": ["aws-marketplace:MeterUsage"],
```

```
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

IAM ロールを作成する

1. https://console.aws.amazon.com/iam/IAMコンソールを開きます。
2. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。
3. [信頼するエンティティのタイプ] で [AWS サービス] を選択します。
4. [ユースケース] セクションの [サービスまたはユースケース] で、[Amazon EC2] を選択します。
5. [次へ] をクリックします。
6. ポリシーのリストで、[タイプでフィルタリング] ドロップダウンから [カスタマー管理] を選択し、作成したポリシーの名前を入力します。ポリシー名の横にあるチェックボックスをオンにします。
7. [次へ] を選択します。
8. ロールの名前を入力し、オプションで説明を入力します。
9. 信頼ポリシーとアクセス許可を確認し、[ロールを作成] を選択します。

Amazon EC2 インスタンスに IAM ロールをアタッチする

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. Amazon EC2 インスタンスを選択します。
4. [アクション] メニューで、[セキュリティ]、[IAM ロールを変更] の順に選択します。
5. インスタンスにアタッチする IAM ロールを選択して、[IAM ロールの更新] をクリックします。

Charon の統合

Charon-SSP の概要

1987 年、Sun Microsystems は 32 ビット RISC プロセッサである SPARC V7 プロセッサをリリースしました。その後、1990 年に SPARC V8 がリリースされました。これは、元の SPARC V7 を改訂したもので、最も大きく変わった点は、ハードウェアによる除算と乗算の命令が組み込まれたことでした。SPARC V8 プロセッサは、SPARCstation 5、10、20 などの多くのサーバーやワークステーションの基盤となりました。1993 年に SPARC V8 プロセッサに続いて登場したのは、64 ビットの SPARC V9 プロセッサです。やはりこれも、Enterprise 250 や 450 などの多くのサーバーやワークステーションの基盤となりました。

ハードウェアが旧モデルとなり、スペア部品や再生部品が欠品しているため、古い SPARC ベースのワークステーションやサーバー向けに開発されたソフトウェアやシステムは保守が難しくなっています。特定の end-of-life SPARC ベースのシステムの継続的なニーズを満たすために、Stromasys S.A. は SPARC エミュレータ製品のCharron-SSP ラインを開発しました。以下の製品は、特定のネイティブハードウェアの SPARC システムに代わるソフトウェアベースの仮想マシンです。エミュレートされたハードウェアファミリーの一般的な概要は次のとおりです。

Charon-SSP/4M は以下の SPARC ハードウェアをエミュレートします。

- Sun-4m ファミリー (Sun SPARCstation 20 など): 当初はマルチプロセッサ Sun-4 バリアントで、SPARCServer 600MP シリーズで導入された MBus プロセッサモジュールバスをベースにしていました。その後、Sun-4m アーキテクチャには、SPARC V8 アーキテクチャプロセッサを利用した SPARCstation 5 などの MBus 以外の単一プロセッサシステムも含まれるようになりました。SunOS 4.1.2 以降と、Solaris 2.1 から Solaris 9 までサポートされています。SPARCServer 600MP のサポートは Solaris 2.5.1 以降では廃止されています。

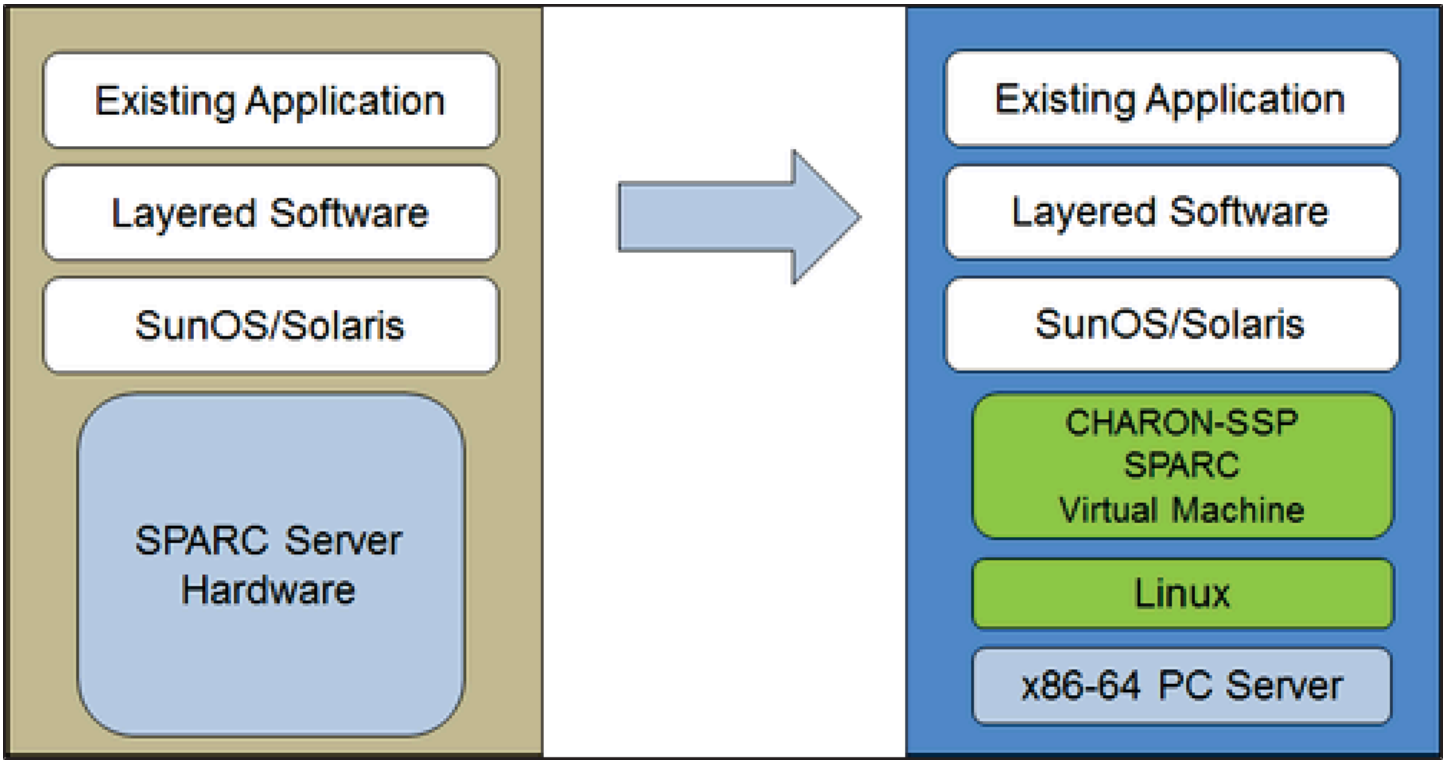
Charon-SSP/4U(+) は以下の SPARC ハードウェアをエミュレートします。

- Sun-4u ファミリー (Sun Enterprise 450 など): (U は UltraSPARC を表す) - このバリアントでは、Sun Ultra シリーズで初めて使用された 64 ビット SPARC V9 プロセッサアーキテクチャと UPA プロセッサインターコネクトが導入されました。バージョン 2.5.1 以降の Solaris の 32 ビットバージョンでサポートされます。Sun-4u 向けの最初の 64 ビット Solaris リリースは Solaris 7 です。UltraSPARC I のサポートは Solaris 9 以降廃止されました。Solaris 10 は、UltraSPARC II から UltraSPARC IV までの Sun-4u 実装をサポートしています。

Charon-SSP/4V(+) は以下の SPARC ハードウェアをエミュレートします。

- Sun-4v ファミリー (SPARC T2 および T4 など): このバリエーションでは、Ultra SPARC T1 マルチコアプロセッサで導入された Sun-4u にハイパーバイザープロセッサの仮想化が追加されました。一部のハードウェアは、リリース 3/05 HW2 以降の Solaris バージョン 10 でサポートされていました (Charon-SSP でエミュレートされたハードウェアを含め、ほとんどのモデルには新しいバージョンの Solaris 10 が必要です)。Solaris 11 のいくつかのバージョンもサポートされています。

次の図は、物理ハードウェアをエミュレータに移行する基本概念を示しています。



Sun と Oracle SPARC ベースのコンピューターのユーザーは、Charon-SSP 仮想マシンを使用して、元のシステム構成をほとんど変更する必要なくネイティブハードウェアを置き換えることができます。別のプラットフォームへの切り替えや移植を必要とせずに、アプリケーションとデータを引き続き実行できる、ということです。Charon-SSP ソフトウェアは市販の Intel 64 ビットシステムで動作するため、投資の継続的保護が保証されます。

Charon-SSP/4U+ は、Charon-SSP/4U と同じ仮想 SPARC プラットフォームをサポートし、Charon-SSP/4V+ は Charon-SSP/4V と同じ仮想 SPARC プラットフォームをサポートしています。ただし、4U+ バージョンと 4V+ バージョンでは、最新の CPU でインテルの VTx/EPT と AMD の AMD-v/NPT ハードウェア仮想化支援テクノロジーを活用して、仮想 CPU のパフォーマンスを向

上させています。Charon-SSP/4U+ と Charon-SSP/4V+ は、VT-x/EPT または AMD-V/NPT をサポートする CPU を必要としており、専用のホストシステムにインストールする必要があります。これらの製品バリアントを VM (VMware など) で実行することはサポートされていません。

> **Note**
>
> Charon-SSP/4U+ または 4V+ をクラウド環境で実行する場合は、Stromasys または Stromasys VAR にお問い合わせいただき、ご要望について相談してください。

サポートされるゲストオペレーティングシステム

Charon-SSP/4M 仮想マシンは、以下のゲストオペレーティングシステムのリリースをサポートしています。

- SunOS 4.1.3 および 4.1.4
- Solaris 2.3 から Solaris 9 まで

Charon-SSP/4U(+) 仮想マシンは、以下のゲストオペレーティングシステムのリリースをサポートしています。

- Solaris 2.5.1 から Solaris 10 まで

Charon-SSP/4V(+) 仮想マシンは、以下のゲストオペレーティングシステムのリリースをサポートしています。

- Solaris 10 (アップデート 4 の 08/07 以降) と Solaris 11.1 から Solaris 11.4 まで

Charon-SSP/4V (+) については、以下の点に注意してください。

- エミュレートされた SPARC T4 でサポートされている Solaris 10 バージョンは、Oracle Solaris 10 1/13、Oracle Solaris 10 8/11、Solaris 10 9/10、または Oracle Solaris 10 8/11 パッチセットを適用した Solaris 10 10/09 です。
- エミュレートされた SPARC T4 モデルは、エミュレータで Solaris 11.4 を実行するための前提条件です。
- Solaris カーネルゾーンはサポートされていません。

Charon-SSP クラウドインスタンスの前提条件

インスタンスのタイプまたは形状を選択すると、クラウド内の Charon-SSP ホストインスタンスに使用される仮想ハードウェアが決まります。したがって、インスタンスのタイプや形状を選択することで、Charon-SSP 仮想ホストハードウェアのハードウェア特性 (仮想 Charon ホストシステムに搭載される CPU コア数やメモリの量など) が決まります。

> **Note**
>
> Charon-SSP マーケットプレイスイメージを使用してインスタンスを起動すると、Linux ホストのオペレーティングシステムの要件はすべて満たされます。

ハードウェアの最小要件は次のとおりです。

サイジングガイドラインに関する重要点:

- 以下のサイジングガイドライン (特にホストの CPU コア数とホストメモリ数に関するガイドライン) は、最小要件を示しています。すべてのデプロイ状況を見直し、必要に応じて実際のホストサイズを調整する必要があります。例えば、ゲストアプリケーションの I/O 負荷が高い場合は、I/O に使用できる CPU コア数を増やす必要があります。また、エミュレートされた CPU の数が多いシステムでは一般的に I/O 負荷が高くなる可能性があるため、I/O に使用できる CPU コア数を増やす必要がある場合があります。ハイパースレッディング環境では、最高のパフォーマンスを引き出すにはアクティブなエミュレータの CPU 要件を満たすのに十分な CPU (つまり、実際の CPU / 物理 CPU) コア数が必要です。これにより、作業負荷の高いスレッドが 1 つの物理 CPU コアを共有することは避けられます。
- エミュレートされた CPU と I/O 処理用の CPU コアの CPU コア割り当ては、構成によって決まります。これに関する詳細と、I/O 処理用 CPU コアのデフォルト割り当てについては、一般的な「Charon-SSP ユーザーガイド」の「CPU 設定」を参照してください。

> **⚠ 重要な一般情報**
>
> - あるクラウドインスタンスから別のクラウドインスタンスへのエミュレータデータの高速転送をしやすくするために、関連するすべてのエミュレータデータを古いインスタンスから簡単に切り離して新しいインスタンスに接続できる別のディスクボリュームに保存することを強くお勧めします。

- インスタンスのディメンションは最初から正しく設定するようにしてください (以下の最小要件を確認してください)。Charon-SSP AL の Charon-SSP ライセンスは、インスタンスを最初に起動したときに作成されます。後で別のインスタンスのサイズやタイプに変更し、CPU コア数を変更すると、ライセンスが無効になり、Charon インスタンスを起動できなくなります (新しいインスタンスが必要です)。Charon-SSP AL インスタンスを AutoVE モードで使用する場合は、最初に起動する前に必ず AutoVE サーバの情報を含めるようにしてください。含めない場合、パブリックライセンスサーバーが使用されます。Charon-SSP VE のライセンスは、ライセンスサーバーで取得したフィンガープリントに基づいて作成されます。ライセンスサーバーがエミュレータホスト上で直接実行され、後になってエミュレータホストで CPU コア数の変更が必要になった場合、ライセンスは無効になります (新しいライセンスと、場合によっては新しいインスタンスが必要になります)。

インスタンスの前提条件

一般的な CPU 要件: Charon-SSP は、最新の x86-64 アーキテクチャプロセッサベースの Amazon EC2 インスタンスをサポートします。

Charon-SSP の最小要件:

- ホストシステムの CPU コアの最小数:
  - ホストオペレーティングシステム用に 1 つ以上の CPU コア、および
  - エミュレートされた SPARC システムごとに:
    - インスタンスのエミュレートされた CPU ごとに CPU コア 1 つ、および
    - I/O 処理用に 1 つ以上の追加 CPU コア (サーバーの JIT 最適化を使用する場合は少なくとも 2 つ)。設定オプションについては、上記の「CPU 設定」セクションを参照してください。デフォルトでは、Charon は Charon ホストから認識される CPU 数の 1/3 (最低 1 つ、切り捨て) を I/O 処理に割り当てます。
- メモリの最小要件:
  - Linux ホストオペレーティングシステム用に 4 GB 以上の RAM。Linux ホストで実行されるエミュレータ以外のサービスの要件によっては、実際の要件はもっと容量が大きくなる場合があります。Linux ホストに 2 GB 以上の RAM を搭載するという以前の推奨事項は、多くのシステムで引き続き有効ですが、Linux オペレーティングシステムとアプリケーションの要件が増加しているため、新規インストールの推奨事項が更新されました。ならびに、
  - エミュレートされた SPARC システムごとに:

    - エミュレートされたインスタンスの設定済みメモリ、および、
    - DIT 最適化、エミュレータ要件、ランタイムバッファ、SMP、グラフィックエミュレーションを可能にする 2 GB の RAM (サーバー JIT を使用する場合は 6 GB の RAM)
- 最新の x86-64 CPU でハイパースレッディングが有効になっていれば、物理 CPU コア 1 つで 2 つのスレッドを実行でき、ホストオペレーティングシステムに論理 CPU を 2 つ提供できます。可能であれば、Charon-SSP ホストのハイパースレッディングを無効にしてください。ただし、VMware やクラウド環境では無効にできない場合や、ハイパースレッディングが使用されているかどうか不明なことがよくあります。Charon-SSP のハイパースレッディングオプションを使用すると、Charon-SSP がこういった環境に適応できるようになります。詳しい設定情報については、前述の一般的な Charon-SSP ユーザーガイドの「CPU 設定」セクションを参照してください。注意: 最高のパフォーマンスを得るためには、Charon-SSP スレッドが物理 CPU コアを共有しないようにしてください。設定したエミュレータの要件を満たすのに十分な物理コアがホストシステム上に確保されている必要があります。
- 1 つ以上のネットワークインターフェース (顧客の要件による)
- Charon-SSP/4U+ と Charon-SSP/4V+ は、Intel VT-x/EPT または AMD-V/NPT (ベアメタルインスタンス) をサポートする物理ハードウェア上で動作する必要があるため、すべてのクラウド環境で実行できるわけではありません。このようなハードウェアが利用できるかどうかについては、クラウドプロバイダーのドキュメントを確認してください。以下の点にも注意してください。
    - Charon-SSP/4U+ と Charon-SSP/4V+ は、Stromasys がサポートする Linux カーネルを使用している場合にのみ利用できます。
    - この種のエミュレートされた SPARC ハードウェアが必要な場合は、Stromasys または Stromasys VAR に要件の詳細をお問い合わせください。

Ceron 用の AWS クラウドインスタンスの作成と設定 (新しい GUI)

このセクションには、2022 AWS Management Console 年の が反映されています。古いコンソールを引き続き使用する場合は、Charron-SSP AWS 入門ガイドの付録を参照してください。

一般的な前提条件

この説明は、 AWSでの Linux インスタンスの基本設定を示しています。特定の前提条件は記載されていません。ただし、ユースケースに応じて、以下の前提条件を検討してください。

- Amazon アカウントと AWS Marketplace サブスクリプション

  - で Linux インスタンスを設定するには AWS、管理者アクセス権を持つ AWS アカウントが必要です。
  - インスタンスを起動する AWS リージョンを特定します。使用する AWS のサービスがそのリージョンで利用可能であることを確認します。「AWS サービス (リージョン別)」を参照してください。
  - インスタンスを起動する VPC とサブネットを指定します。
  - インスタンスにインターネットアクセスが必要な場合は、VPC に関連付けられているルートテーブルにインターネットゲートウェイがあることを確認してください。インスタンスがオンプレミスネットワークへの VPN アクセスを必要とする場合は、VPN ゲートウェイが使用可能であることを確認してください。VPC とそのサブネットの正確な設定は、ネットワーク設計とアプリケーションの要件によって異なります。
  - 特定の AWS Marketplace サービスをサブスクライブするには、 で AWS Marketplace サブスクリプションを選択し AWS Management Console 、サブスクリプションの管理 を選択します。
  - 使用するサービスを検索してサブスクライブします。サブスクリプションが完了すると、[サブスクリプションの管理] セクションにサブスクリプションが表示されます。そこから、新しいインスタンスを直接起動できます。
- インスタンスのハードウェアとソフトウェアの前提条件は、インスタンスの使用計画によって異なります。
  - オプション 1: インスタンスを Charon エミュレータホストシステムとして使用する。
    - Linux インスタンスが満たす必要のあるハードウェアとソフトウェアの前提条件を正確に知るには、Charon 製品のユーザーガイドや入門ガイドのハードウェアとソフトウェアの前提条件のセクションを参照してください。インスタンスの起動に使用するイメージと選択したインスタンスタイプによって、クラウドインスタンスのソフトウェアとハードウェアが決まります。
    - エミュレートされたレガシーシステムを実行するには、Charon 製品ライセンスが必要です。詳細については、Charon 製品のマニュアルに記載されているライセンス情報を参照するか、Stromasys の担当者または Stromasys VAR にお問い合わせください。
  - オプション 2: インスタンスを専用の VE ライセンスサーバーとして使用する。
    - 詳しい前提条件については、「VE ライセンスサーバガイド」を参照してください。
- Charon エミュレータ製品から提供されるエミュレートシステムで実行できるレガシーオペレーティングシステムによっては、そのオペレーティングシステムの元のベンダーのライセンスが必要となります。レガシーオペレーティングシステムに関連するすべてのライセンス義務はユーザーが負担し、適切なライセンスを提示する必要があります。

を使用して新しいインスタンス AWS Management Console を起動する

新しいインスタンスを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. [インスタンスの起動] を選択します。
3. インスタンスの名前を入力します。
4. AMI を選択します。AMI は、クラウドインスタンスの起動に使用される、事前にパッケージされたイメージです。これには、オペレーティングシステムと、適切なアプリケーションソフトウェアが含まれます。どの AMI を選択するかは、インスタンスをどのように使用するかによります。

   - インスタンスを Charon エミュレータホストシステムとして使用する場合は、複数の AMI を選択できます。
     - 事前にパッケージされた Charon マーケットプレイスイメージからの Charon ホストシステムのインストール: Charon ホストシステムには、基盤となるオペレーティングシステムとプレインストールされた Charon ソフトウェアが含まれます。
       - ご利用のクラウドプロバイダーのマーケットプレイスで現在利用できるオプションについては、Stromasys の担当者にご確認ください。
       - クラウドプロバイダーと Stromasys 製品のリリースプランによって、次の 2 つのバリエーションがあります。
         - Stromasys が運営するパブリックライセンスサーバー、または顧客が運営するプライベートの AutoVE ライセンスサーバーで使用する自動ライセンス (AL)
         - 顧客が運営するプライベート VE ライセンスサーバで使用する仮想環境 (VE)
     - Linux 用の Charon エミュレータインストール RPM パッケージを使用した、従来の Charon エミュレータインストールによる Charon ホストシステムのインストール
       - 選択した Charon 製品とバージョンでサポートされているディストリビューションの Linux AMI を選んでください。Stromasys ドキュメントサイトにある製品のユーザーガイドを参照してください。
   - インスタンスを専用の VE ライセンスサーバーとして使用する場合は、ライセンスドキュメントの「VE License Server Guide」を参照して Linux インスタンスの要件を確認してください。

必要な AMI を決めたら、対応する Linux または Charon 製品の AMI を選択します。必要な AMI が表示されない場合は、[その他の AMI を閲覧する] を選択します。インスタンスの使用方法に合った Linux AMI を選択します。次のいずれかを指定できます。

- 事前にパッケージされた Charon VE マーケットプレイスイメージ。AMI の名前に「ve」という文字が含まれます。
- 自動ライセンスまたは AutoVE 用に事前にパッケージされた Charon AL マーケットプレイスイメージ。
- RPM 製品のインストールがサポートされている Linux バージョン。
- VE ライセンスサーバーでサポートされている Linux バージョン。

5. インスタンスタイプの選択 Amazon EC2 では、CPU、メモリ、ストレージ、ネットワーキングキャパシティをさまざまな組み合わせで用いたインスタンスタイプが提供されています。使用する Charon 製品の要件に対応するインスタンスタイプを選択します。マーケットプレイスイメージの中には、インスタンスタイプの選択が制限されているものがあります。
6. 既存のキーペアを選択するか、新しいキーペアを作成して保存します。既存のキーペアを選択する場合は、対応するプライベートキーがあることを確認してください。ない場合は、インスタンスに接続できなくなります。

> **Note**
>
> 管理システムが RHEL 9.x、Rocky Linux 9.x、Oracle Linux 9.x をサポートしている場合、SSH キータイプ ECDSA または ED25519 を使用してください。これらのタイプでは、Charon ホストのデフォルトの暗号ポリシー設定を安全性の低い設定に変更しなくても、SSH トンネルを使用して Charon ホストの Linux システムに接続できます。例えば、これは Charon-SSP Manager にとって重要です。Red Hat ドキュメントの「システム全体の暗号化ポリシーの使用」を参照してください。

7. [ネットワーク設定] セクションで、[編集] をクリックします。ご使用の環境に対応する設定を選択します。

- VPC を指定します。
- 既存のサブネットを指定するか、新しいサブネットを作成します。
- プライマリインターフェイスへのパブリック IP アドレスの自動割り当てを有効または無効にします。自動割り当ては、インスタンスにネットワークインターフェイスが 1 つしかない場合にのみ可能です。

- 既存または新規のカスタムセキュリティグループを割り当てます。セキュリティグループは、少なくとも SSH によるインスタンスへのアクセスを許可する必要があります。インスタンス上で実行するアプリケーションに必要なポートもすべて許可されている必要があります。インスタンスを作成した後は、いつでもセキュリティグループを変更できます。

8. [ストレージ]セクションのルートボリューム (システムディスク) で、環境に適したサイズを選択します。Linux システムの推奨最小システムディスクサイズは 30 GiB です。仮想ディスクコンテナやその他のストレージ要件に対応するスペースを確保するために、今すぐかインスタンスを起動した後にストレージを追加できます。ただし、システムディスクサイズは、インストールするアプリケーションやユーティリティを含め、Linux のシステム要件を満たす必要があります。

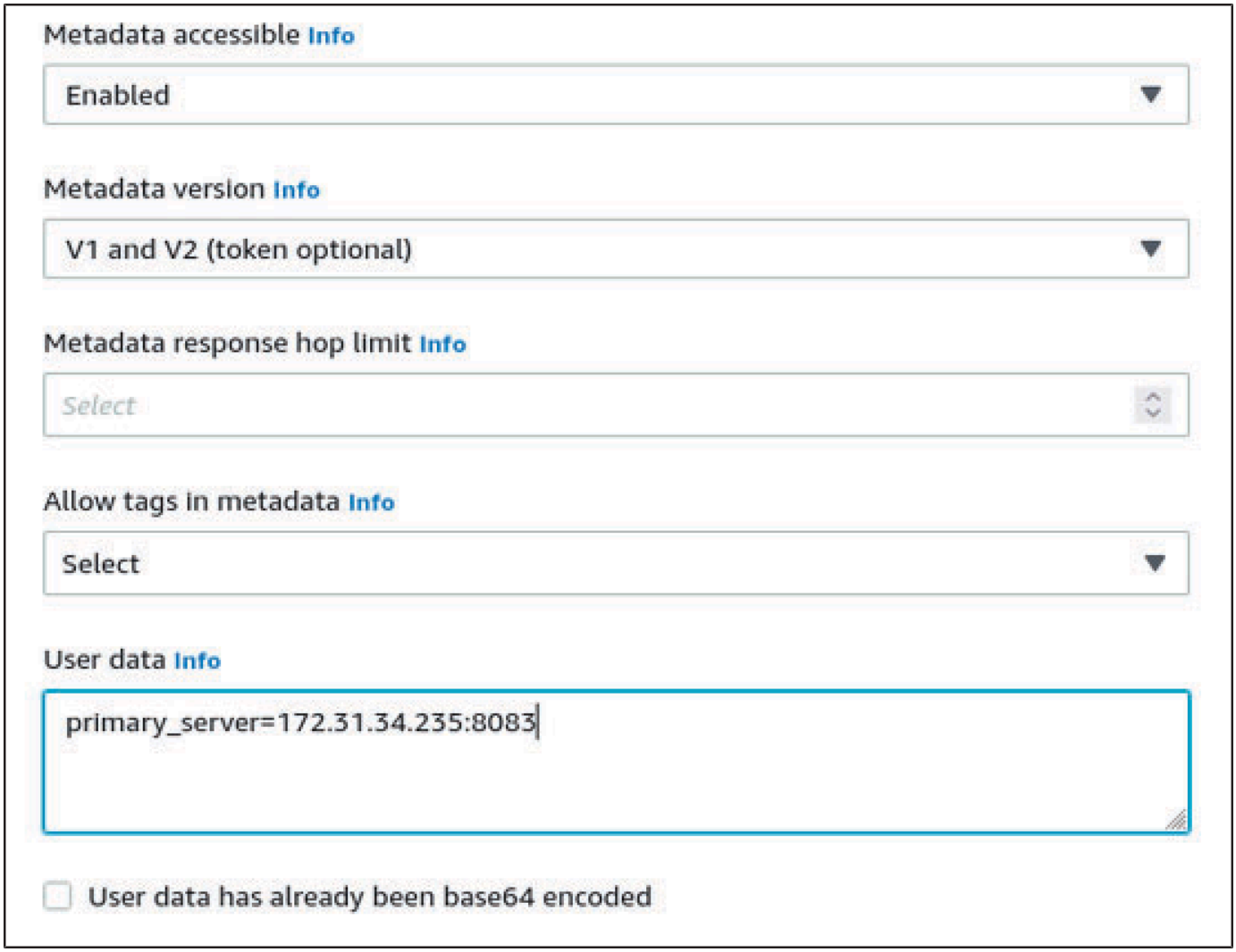
> **Note**
>
> Charon アプリケーションデータ (ディスクイメージなど) 用に別のストレージボリュームを作成することをお勧めします。必要ならば、後でそのボリュームを別のインスタンスに移行できます。

9. [詳細設定] セクションを展開し、下にスクロールして [CPU オプションの指定] を選択します。以下の図は、Charon エミュレータ環境に役立つと思われる 3 つを例として示しています。

10. 1.1.23 より前のバージョンの VE ライセンスサーバシステムでは、必要な IAM ロールをインスタンスに割り当てる必要があります。`ListUsers` アクションを許可するロールでなければなりません。ロールを割り当てるには、展開した [詳細設定] セクションで、[IAM インスタンスプロファイル] でロールを選択するか、[新しい IAM プロファイルの作成] を選択します。詳細については、「Amazon EC2 の IAM ロール」を参照してください。

11. インスタンスがチャロン AL AWS Marketplace イメージに基づいていて、Stromasys が推奨するパブリックライセンスサーバーを使用する予定がある場合は、インスタンスを起動する前に、対応する情報をインスタンス設定に追加する必要があります。

    次の画像に示すように、AutoVE ライセンスサーバーの情報を入力します。



    有効なユーザーデータ設定オプションは次のとおりです。

    - **primary_server=**<*ip-address*>[**:**<*port*>]
    - **backup_server=**<*ip-address*>[**:**<*port*>]

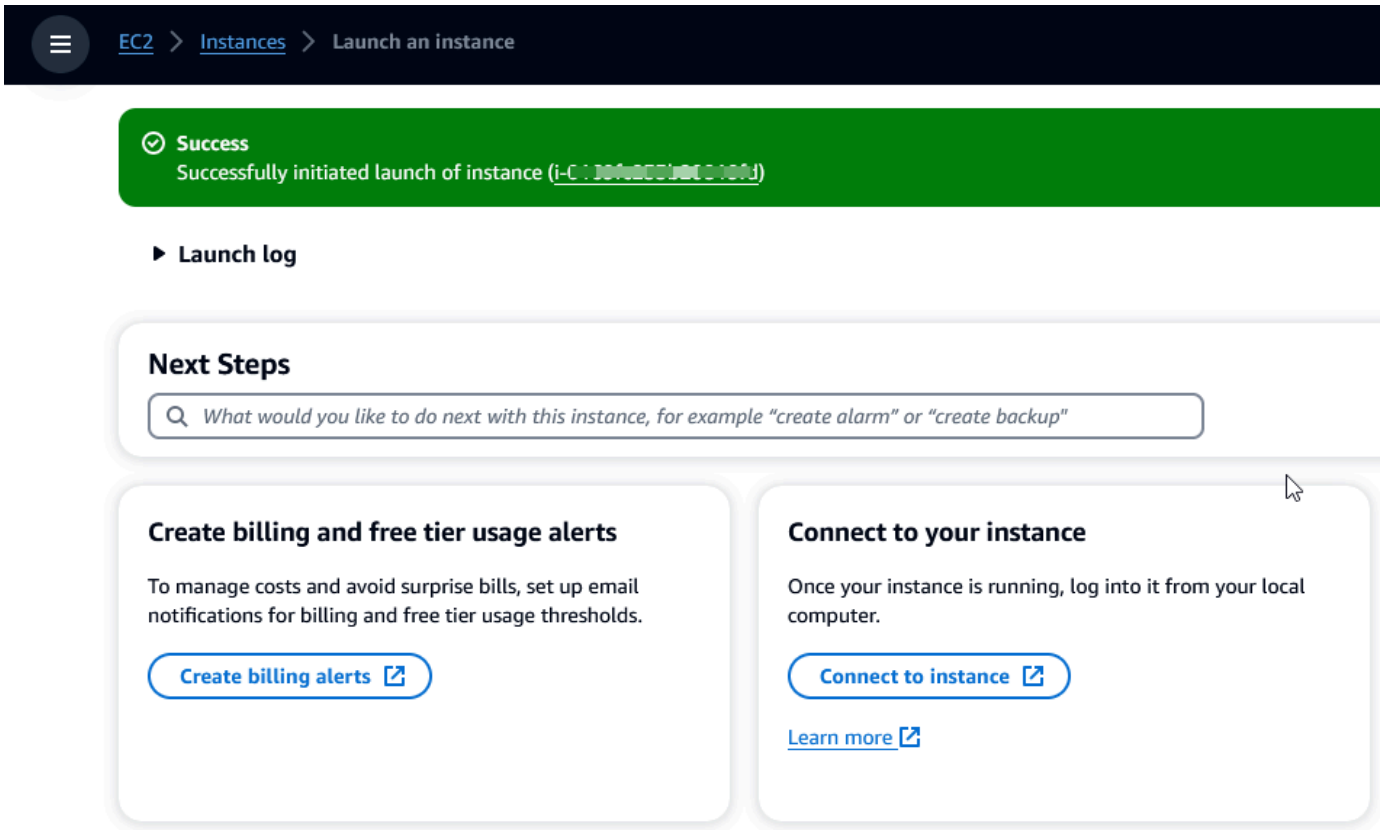
    各パラメータの意味は次のとおりです。

    - <ip-address> はプライマリサーバーとバックアップサーバー (該当する場合) の IP アドレスを表します。

- <port> はライセンスサーバーとの通信に使用されるデフォルト以外の TCP ポート (デフォルト: TCP/8083) を表します。

> **Note**
>
> AutoVE モードを有効にするには、初回起動時に少なくとも 1 つのライセンスサーバーを設定する必要があります。設定しないと、インスタンスは Stromasys が運営するパブリックライセンスサーバーの 1 つにバインドされます。

12. [サマリー] セクションで、[インスタンスの起動] を選択します。しばらくすると、以下の成功メッセージが表示されます。



13. 画面右下にある [すべてのインスタンスの表示] を選択します。
14. インスタンスの詳細を表示するには、[インスタンス] テーブル内のインスタンスを表す行の左側にあるチェックボックスを選択します。インスタンスの詳細が画面の下半分に表示されます。インスタンスへの接続方法の詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「接続」を参照してください。

AWS NTT DATA による Mainframe Modernization の再プラットフォーム

AWS Mainframe Modernization には、さまざまな Amazon マシンイメージ (AMIs。これらの AMIs、Amazon EC2 インスタンスの迅速なプロビジョニングを容易に AWS し、NTT データを使用してメインフレームアプリケーションを でリホストおよびリプラットフォームするためのカスタマイズされた環境を作成します。このガイドでは、これらの AMI にアクセスして使用するために必要な手順について説明します。

前提条件

- Amazon EC2 インスタンスを作成できる AWS アカウントへの管理者アクセス権があることを確認します。
- AWS Mainframe Modernization サービスが、Amazon EC2 インスタンスを作成する予定のリージョンで利用可能であることを確認します。「リージョン別に利用可能な AWS サービスのリスト」を参照してください。
- Amazon EC2 インスタンスを作成する Amazon VPC を特定します。

Amazon マシンイメージのサブスクライブ

AWS Marketplace 製品をサブスクライブすると、その製品の AMI からインスタンスを起動できるようになります。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/marketplace で AWS Marketplace コンソールを開きます。
2. [サブスクリプションを管理する] を選択します。
3. 次のリンクをコピーしてブラウザのアドレスバーに貼り付けます。https://aws.amazon.com/marketplace/pp/prodview-eg227ymldsnx2
4. [サブスクリプションを続行する] を選択します。
5. 利用規約に同意できる場合は、[規約の受諾] を選択します。このサブスクリプションの処理には数分かかる場合があります。
6. お礼のメッセージが表示されるまで待ちます。このメッセージは、製品へのサブスクリプションが完了したことを確認するものです。

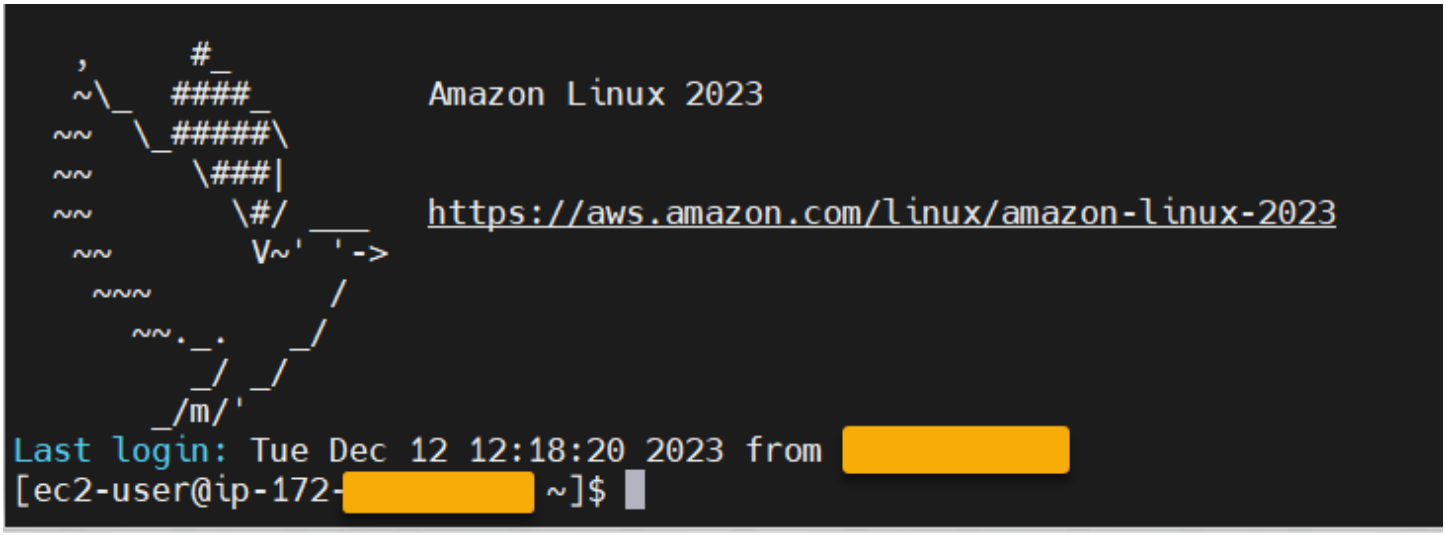
7. 左のナビゲーションペインで、[サブスクリプションの管理] をクリックします。このビューには、すべてのサブスクリプションが表示されます。

NTT DATA インスタンスで AWS Mainframe Modernization リプラットフォームを起動する

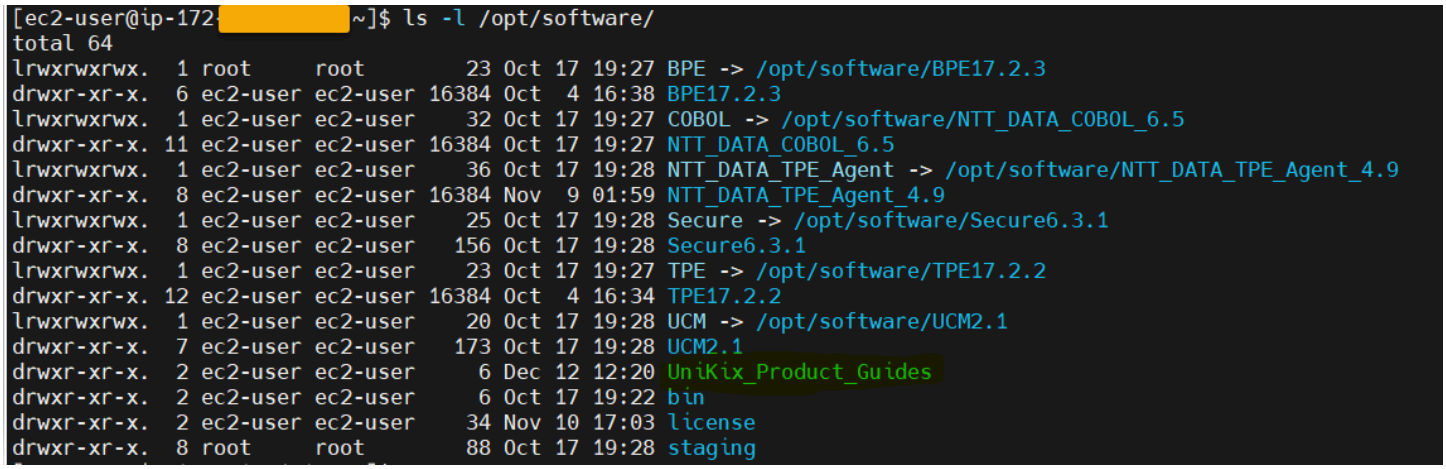
1. https://console.aws.amazon.com/marketplace で AWS Marketplace コンソールを開きます。
2. 左のナビゲーションペインで、[サブスクリプションの管理] をクリックします。
3. 起動する AMI を見つけて、[新規インスタンスを起動] をクリックします。
4. [リージョン] で、許可リストに登録されているリージョンを選択します。
5. [EC2 経由で起動を続行する] を選択します。これにより、Amazon EC2 コンソールに移動します。
6. サーバーの名前を入力します。
7. プロジェクトのパフォーマンスとコスト要件に合ったインスタンスタイプを選択します。インスタンスサイズの推奨開始点は `c5.2xLarge` です。
8. 既存のキーペアを選択するか、新しいキーペアを作成して保存します。キーペアの詳細については、「Amazon EC2 Linux インスタンス用ユーザーガイド」の「Amazon EC2 のキーペアと Linux インスタンス」を参照してください。
9. ネットワーク設定を編集し、許可リストに登録された VPC と適切なサブネットを選択します。
10. 既存のセキュリティグループを選択するか、新しいセキュリティグループを作成します。Enterprise Server Amazon EC2 インスタンスの場合は、通常、ポート 86 と 10086 への TCP トラフィックを許可して Micro Focus の設定を管理します。
11. Amazon EC2 インスタンスのストレージを設定します。
12. 概要を確認し、[インスタンスを起動] をクリックします。起動を成功させるには、インスタンスタイプが有効である必要があります。起動に失敗した場合は、[インスタンス設定を編集] をクリックし、別のインスタンスタイプを選択します。
13. 成功メッセージが表示されたら、[インスタンスに接続] をクリックします。
14. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
15. 左側のナビゲーションペインの [インスタンス] メニューで、[インスタンス] をクリックします。
16. メインペインで、インスタンスのステータスをチェックします。

NTT Data の使用開始

Amazon EC2 インスタンスをプロビジョニングしたら、ユーザー名 ec2-user を使用してそのインスタンスに SSH 接続します。画面は次の画像のようになります。



以下の画像に示すように、この /opt/software/ フォルダの下には UniKix_Product_Guides という名前のフォルダがあります。



UniKix_Product_Guides フォルダには、この Amazon EC2 インスタンスにインストールされている以下のコンポーネントのドキュメントが含まれています。

- NTT DATA TPE
- NTT DATA BPE
- NTT DATA Enterprise COBOL
- NTT データ UniKix セキュア

- NTT DATA UniKix Central Manager

前の画像に表示されている `software` フォルダーには、上記のコンポーネントのバイナリが格納されています。

Amazon EC2 インスタンスが正常に検証されたら、NTT データドキュメントに従って NTT DATA で AWS Mainframe Modernization リプラットフォームの使用を開始します。

AWS Mainframe Modernization におけるアプリケーション

AWS Mainframe Modernization を初めて使用する場合は、次のトピックを参照して使用を開始してください。

- AWS Mainframe Modernization とは
- AWS Mainframe Modernization のセットアップ
- チュートリアル: AWS Blu Age のマネージドランタイム
- チュートリアル: Micro Focus のマネージドランタイム

AWS Mainframe Modernization アプリケーションには、移行したメインフレームのワークロードが含まれています。このアプリケーションは、メインフレームのワークロードに似ており、ランタイム環境に関連しています。バッチファイルやデータセットをアプリケーションに追加して、実行中のアプリケーションを監視できます。移行するワークロードごとに AWS Mainframe Modernization アプリケーションを作成します。AWS Mainframe Modernization アプリケーションの作成時に、アプリケーションを実行するエンジンを指定します。自動リファクタリングパターンを使用している場合は AWS Blu Age を選択し、リプラットフォームパターンを使用している場合は Micro Focus を選択します。

トピック

- AWS Mainframe Modernization アプリケーションの作成
- AWS Mainframe Modernization アプリケーションのデプロイ
- AWS Mainframe Modernization アプリケーションの更新
- 環境からの AWS Mainframe Modernization アプリケーションの削除
- AWS Mainframe Modernization アプリケーションの削除
- AWS Mainframe Modernization アプリケーションのバッチジョブの送信
- AWS Mainframe Modernization アプリケーションのデータセットのインポート
- AWS Mainframe Modernization アプリケーションのトランザクションの管理
- 移行済みアプリケーションの AWS リソースの作成
- マネージドアプリケーションの設定
- AWS Mainframe Modernization アプリケーション定義リファレンス
- AWS Mainframe Modernization データセット定義リファレンス

AWS Mainframe Modernization アプリケーションの作成

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization アプリケーションを作成します。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

アプリケーションの作成

アプリケーションを作成するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、アプリケーションを作成するリージョンを選択します。
3. [Applications] ページで、[Create application] を選択します。
4. [基本情報を指定] ページの [名前と説明] セクションに、アプリケーションの名前を入力します。
5. (オプション) [アプリケーションの説明] フィールドにアプリケーションの説明を入力します。この説明は、お客様と他のユーザーがアプリケーションの用途を識別するのに役立ちます。
6. [エンジンタイプ] セクションで、自動リファクタリングには [Blu Age] を、リプラットフォームには [Micro Focus] を選択します。
7. [KMS キー] セクションで、カスタマーマネージド AWS KMS キーを使用する場合は、[暗号化の設定をカスタマイズ] を選択します。詳細については、「AWS Mainframe Modernization サービスの保管時のデータ暗号化」を参照してください。

   > **Note**
   >
   > デフォルトでは、AWS Mainframe Modernization は、AWS Mainframe Modernization が所有および管理する AWS KMS キーを使用してデータを暗号化します。ただし、カスタマーマネージド AWS KMS キーの使用を選択できます。

8. (オプション) 名前または Amazon リソースネーム (ARN) で AWS KMS キーを選択するか、[AWS KMS キーを作成] を選択して AWS KMS コンソールに移動し、新しい AWS KMS キーを作成します。
9. (オプション) [タグ] セクションで [新しいタグを追加] を選択し、アプリケーションタグをアプリケーションに追加します。アプリケーションタグとは、AWS のリソースの整理や管理に役立つカスタム属性ラベルです。

10. [次へ] をクリックします。
11. [リソースと設定] セクションで、インラインエディタを使用してアプリケーション定義を入力します。または、[Amazon S3 バケット内のアプリケーション定義 JSON ファイルを使用する] を選択し、使用するアプリケーション定義の場所を指定します。詳細については、AWS Blu Age アプリケーション定義のサンプルまたはMicro Focus アプリケーション定義を参照してください。
12. [次へ] をクリックします。
13. [確認と作成] ページで入力した情報を確認してから、[アプリケーションを作成] を選択します。

AWS Mainframe Modernization アプリケーションのデプロイ

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization アプリケーションをデプロイします。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

アプリケーションのデプロイ

AWS Mainframe Modernization アプリケーションを実行するには、ランタイム環境にデプロイする必要があります。1 つのアプリケーションには、1 つまたは複数のバージョンを設定できます。アプリケーションの各バージョンには、独自のアプリケーション定義があります。アプリケーションをデプロイするには、デプロイするバージョンを指定する必要があります。

指定したアプリケーションのバージョンは、一度に 1 つのバージョンしかデプロイできません。あるバージョンのアプリケーションをデプロイし、代わりに別のバージョンをデプロイすることにした場合、まず、実行中のアプリケーションを停止する必要があります。

アプリケーションをデプロイするには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、アプリケーションを作成するリージョンを選択します。
3. [アプリケーション] ページで、デプロイするアプリケーションを選択します。
4. [アプリケーションのデプロイ] を選択します。
5. [利用可能なバージョン] セクションで、デプロイするバージョンを選択します。

6. [環境] セクションで、アプリケーションを実行するランタイム環境を選択します。
7. デプロイを選択します。

デプロイしたアプリケーションの別のバージョンをデプロイするには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、アプリケーションを作成するリージョンを選択します。
3. [アプリケーション] ページで、デプロイするアプリケーションを選択します。
4. [アクション] メニューから [アプリケーションを停止] を選択します。
5. アプリケーションが停止したら、[アプリケーションをデプロイ] を選択します。
6. [利用可能なバージョン] セクションで、デプロイするバージョンを選択します。[環境] セクションでは、アプリケーションが既にデプロイされている環境があらかじめ選択されています。
7. デプロイを選択します。

AWS Mainframe Modernization アプリケーションの更新

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization アプリケーションを更新します。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

アプリケーションを更新する

AWS Mainframe Modernization アプリケーションには複数のバージョンがあり、それぞれに独自のアプリケーション定義があります。アプリケーションを更新するには、新しいアプリケーション定義を指定します。この定義により、アプリケーションの新しいバージョンを作成します。

アプリケーションを更新するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、作成されたアプリケーションを更新するリージョンを選択します。

3. [アプリケーション] ページで、更新するアプリケーションを選択します。
4. [アプリケーションの詳細] ページの [現在の定義] セクションで [編集] を選択し、現在のアプリケーション定義を更新します。
5. [アプリケーションを更新] ページで、インラインエディタを使用して現在のアプリケーション定義を更新します。

   または、[Amazon S3 バケット内のアプリケーション定義 JSON ファイルを使用する] を選択し、使用するアプリケーション定義の場所を指定します。詳細については、AWS Blu Age アプリケーション定義のサンプルまたはMicro Focus アプリケーション定義を参照してください。
6. アプリケーション定義の更新が完了したら、[更新] を選択します。

> **Note**
>
> アプリケーションを更新したら、再度デプロイする必要があります。詳細については、「AWS Mainframe Modernization アプリケーションのデプロイ」を参照してください。

環境からの AWS Mainframe Modernization アプリケーションの削除

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization アプリケーションを環境から削除できます。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

環境からのアプリケーションの削除

実行中の AWS Mainframe Modernization アプリケーションを削除する必要がある場合は、まず、そのアプリケーションを停止します。アプリケーションのステータスは、[アプリケーション] ページで確認できます。

環境からアプリケーションを削除するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。

2. AWS リージョン セレクタで、作成されたアプリケーションを環境から削除するリージョンを選択します。
3. [アプリケーション] ページで、環境から削除するアプリケーションを選択し、[アクション] を選択します。
4. (オプション) アプリケーションのステータスが Running の場合、[アプリケーションを停止] を選択します。
5. [環境から削除] を選択します。

   すぐに削除が開始されます。

AWS Mainframe Modernization アプリケーションの削除

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization アプリケーションを削除します。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

アプリケーションの削除

実行中の AWS Mainframe Modernization アプリケーションを削除する必要がある場合は、まず、そのアプリケーションを停止します。アプリケーションのステータスは、[アプリケーション] ページで確認できます。

アプリケーションを削除するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、作成されたアプリケーションを削除するリージョンを選択します。
3. [アプリケーション] ページで、削除するアプリケーションを選択し、[アクション] を選択します。
4. (オプション) アプリケーションのステータスが Running の場合、[アプリケーションを停止] を選択します。
5. [アプリケーションを削除] を選択します。
6. [アプリケーションを削除] ウィンドウで、delete を入力して削除するアプリケーションを確認し、[削除] を選択します。

AWS Mainframe Modernization アプリケーションのバッチジョブの送信

AWS Mainframe Modernization では、アプリケーションのバッチジョブを送信できます。バッチジョブの送信やキャンセル、バッチジョブの実行に関する詳細を確認できます。バッチジョブを送信するたびに、AWS Mainframe Modernization は個別のバッチジョブの実行を作成します。このジョブの実行をモニタリングできます。バッチジョブを名前で検索し、バッチジョブに JCL またはスクリプトファイルを提供できます。

> **Important**
>
> バッチジョブをキャンセルしても、ジョブは削除されません。特定のバッチジョブの実行はキャンセルされます。バッチジョブのレコードは、バッチジョブの実行の詳細で引き続き確認できます。

バッチジョブが 1 つ以上のデータセットにアクセスする必要がある場合は、AWS Mainframe Modernization コンソールまたは AWS Command Line Interface (AWS CLI) を使用してデータセットをインポートします。詳細については、「AWS Mainframe Modernization アプリケーションのデータセットのインポート」を参照してください。

これらの手順では、AWS Mainframe Modernization のセットアップ および AWS Mainframe Modernization アプリケーションの作成 のステップを完了していることを前提としています。

バッチジョブの送信

バッチジョブを送信するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、作成されたバッチジョブを送信するリージョンを選択します。
3. [アプリケーション] ページで、バッチジョブを送信するアプリケーションを選択します。

> **Note**
>
> アプリケーションにバッチジョブを送信する前に、アプリケーションを正常にデプロイする必要があります。

4. [アプリケーションの詳細] ページで、[バッチジョブ] を選択します。
5. [Submit job] (ジョブの送信) を選択します。
6. [スクリプトの選択] セクションで、スクリプトを選択します。名前で必要なスクリプトを検索できます。
7. [Submit job] (ジョブの送信) を選択します。

AWS Mainframe Modernization アプリケーションのデータセットのインポート

AWS Mainframe Modernization では、データセットをインポートしてアプリケーションで使用できます。Amazon S3 バケットに保存された JSON ファイルでデータセットを指定できます。また、データセットの設定値を個別に指定できます。データセットをインポートした後、インポートタスクの詳細を確認して、必要なデータセットがインポートされたことを確認できます。アプリケーションのすべてのカタログ化されたデータセットは、コンソールにまとめて一覧表示されます。

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization アプリケーションのデータセットをインポートします。

これらの手順では、AWS Mainframe Modernization のセットアップ および AWS Mainframe Modernization アプリケーションの作成 のステップを完了していることを前提としています。

データセットのインポート

データをインポートするには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、データセットをインポートするアプリケーションが作成されたリージョンを選択します。
3. [アプリケーション] ページで、データセットをインポートするアプリケーションを選択します。
4. [アプリケーションの詳細] ページで、[データセット] を選択します。
5. Import (インポート) を選択します。
6. 次のいずれかを行います。
   - [Amazon S3 バケットのデータセット設定 JSON ファイルを使用する] を選択し、データセット設定の場所を指定します。

- ガイド付き設定を使用して [データセットの設定値を個別に指定する] を選択します。具体的な定義の詳細については、「the section called “データセット定義リファレンス”」を参照してください。

  名前、データセット組織 (VSAM、GDG、PO、PS)、場所、外部の Amazon S3 の場所、各データセット設定値のパラメータ設定を入力します。また、ガイド付き設定では、[JSON を生成] を選択して、入力から JSON 設定を確認できます。

7. 送信 を選択します。

AWS Mainframe Modernization アプリケーションのトランザクションの管理

AWS Mainframe Modernization を使用すると、同一のファイルやプログラムを使用して、同じアプリケーションを実行するリクエストを送信する他の多くのユーザーと同時に、リクエストに応じてアプリケーションを実行できます。1 つのトランザクションは、必要な処理を実行する 1 つ以上のアプリケーションプログラムで構成されます。

これらの手順では、AWS Mainframe Modernization のセットアップ および AWS Mainframe Modernization アプリケーションの作成 のステップを完了していることを前提としています。

アプリケーションのトランザクションの管理

アプリケーションのトランザクションを表示および編集できます。

アプリケーションのトランザクションを管理するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、作成されたアプリケーションを実行する AWS リージョン を選択します。
3. [アプリケーション] ページで、トランザクションを管理するアプリケーションを選択します。
4. [トランザクション] タブの [トランザクションリソース] で、ドロップダウンリストからリソースの表示方法を選択します。トランザクションリソース、グループ、リスト、または SIT に従ってリソースを表示できます。

   - [トランザクションリソース] では、ファイル定義、トランザクション定義、プログラム定義、または一時データキュー定義に従ってリソースタイプを選択できます。

> **Note**
>
> AWS Mainframe Modernization Managed トランザクションでは、これらよりも多くのリソースタイプをサポートしていますが、ここで直接編集することはできません。他のリソースは外部で編集する必要があり、更新済みのリソースエントリを使用してアプリケーションを再作成する必要があります。

- [グループ] は、トランザクションリソースの集合です。また、トランザクションリソースに関連付けるグループを選択できます。
- [リスト] は、順序付けられたグループの集合です。すべてのトランザクションリソースとグループをリストビューで確認できます。[スタートアップリスト] によって、サーバーの初期化時にどのリソースが読み込まれるかを決定します。
  - Blu Age リファクタリングエンジンでは、スタートアップ時に含めるリストを指定します。リストの数に制限はありません。
  - Micro Focus リプラットフォームエンジンでは、1 つの SIT で最大 4 つのリストを指定できます。
- [システム初期化テーブル (SIT)] には、使用可能なすべてのトランザクション構成が表示されます。プロパティ (名前、説明、スタートアップリスト) に従って SIT を検索できます。リストを選択することにより、選択した SIT に関連付けることもできます。

> **Note**
>
> SIT は、Micro Focus リプラットフォームエンジンにのみ適用されます。

5. トランザクションリソースを選択すると、すべてのリソース情報が表示されます。また、トランザクションリソースに関連付けられたすべての属性の表示や、その他の属性の表示または編集を行うことができます。
6. 現在のトランザクションリソースに関連する情報を編集する場合は、[編集] を選択します。

   a. [編集] ページでは、リソースの説明を追加または変更できます。

      1. リストに表示されるトランザクションリソースについては、必要に応じて、リストを追加、削除、または並べ替えることもできます。
      2. 特定のリソースタイプ (ファイル定義、トランザクション定義、プログラム定義、一時データキュー定義) については、個々のリソースプロパティを編集できます。これらのプロパティは、エンジンとリソースタイプによって異なります。

b. 必要な変更を加えたら、[変更の保存] を選択します。

リソースが正常に更新されると、メッセージが表示されます。

移行済みアプリケーションの AWS リソースの作成

AWS で移行済みのアプリケーションを実行するには、他の AWS のサービス リソースと一緒に、いくつかの AWS リソースを作成する必要があります。作成する必要があるリソースには、次のものが含まれます。

- アプリケーションコード、設定、データファイル、その他の必要なアーティファクトを格納する S3 バケット。
- アプリケーションが必要なデータを保持する Amazon RDS または Amazon Aurora データベース。
- AWS Secrets Manager によってシークレットを作成し、保存するために必要な AWS KMS key。
- データベースの認証情報を保持する Secrets Manager のシークレット。

> **Note**
>
> 移行するアプリケーションには、それぞれ独自のリソースセットが必要です。これは、最低限のセットです。また、アプリケーションでは、Amazon Cognito Secrets や MQ キューなどの追加のリソースが必要になる場合があります。

必要なアクセス許可

次のアクセス許可があることを確認します。

- `s3:CreateBucket`, `s3:PutObject`
- `rds:CreateDBInstance`
- `kms:CreateKey`
- `secretsmanager:CreateSecret`

Amazon S3 バケット

リファクタリングされたアプリケーションとリプラットフォームされたアプリケーションのどちらにも、次のように設定された S3 バケットが必要です。

```
bucket-name/root-folder-name/application-name
```

bucket-name

Amazon S3 の命名の制約内の任意の名前。AWS リージョン 名をバケット名の一部に含めることを推奨します。移行済みのアプリケーションをデプロイするリージョンと同じリージョンにバケットを作成してください。

root-folder-name

AWS Mainframe Modernization アプリケーションの一部として作成するアプリケーション定義の制約を満たすために必要な名前。`root-folder-name` を使用すると、V1 や V2 など、アプリケーションのさまざまなバージョンを区別できます。

application-name

移行したアプリケーションの名前。例えば、 PlanetsDemo BankDemo。

データベース

リファクタリングされたアプリケーションとリプラットフォームされたアプリケーションのどちらにも、データベースが必要な場合があります。各ランタイムエンジンの固有の要件に従って、データベースを作成、設定、管理する必要があります。AWSMainframe Modernization は、このデータベースでの転送中の暗号化をサポートします。データベースで SSL を有効にする場合は、データベースの接続情報とともにデータベースシークレットに `sslMode` を必ず指定してください。詳細については、「AWS Secrets Manager シークレット」を参照してください。

AWS Blu Age リファクタリングパターンを使用しており、 BluSam データベースが必要な場合、AWS Blu Age ランタイムエンジンは Amazon Aurora PostgreSQL データベースを想定しており、このデータベースを作成、設定、管理する必要があります。 BluSam データベースはオプションです。このデータベースは、アプリケーションで必要な場合にのみ作成してください。データベースを作成するには、「Amazon Aurora ユーザーガイド」の「Amazon Aurora DB クラスターの作成」のステップに従ってください。

Micro Focus のリプラットフォームパターンを使用している場合は、Amazon RDS または Amazon Aurora PostgreSQL データベースのいずれかを作成できます。データベースを作成するには、「Amazon RDS ユーザーガイド」の「Amazon Aurora DB インスタンスの作成」または「Amazon Aurora ユーザーガイド」の「Amazon Aurora DB クラスターの作成」のステップに従ってください。

どちらのランタイムエンジンでも、データベース認証情報を暗号化するために AWS KMS key を使用して AWS Secrets Manager に保存する必要があります。

AWS Key Management Service キー

アプリケーションデータベースの認証情報は、AWS Secrets Manager に安全に保存する必要があります。Secrets Manager でシークレットを作成するには、AWS KMS key を作成する必要があります。KMS キーを作成するには、「AWS Key Management Service デベロッパーガイド」の「キーの作成」のステップに従います。

キーを作成したら、キーポリシーを更新して AWS Mainframe Modernization の復号の権限を付与する必要があります。次のポリシーステートメントを追加します。

```
{
   "Effect" : "Allow",
   "Principal" : {
   "Service" : "m2.amazonaws.com"
   },
   "Action" : "kms:Decrypt",
   "Resource" : "*"
   }
```

AWS Secrets Manager シークレット

アプリケーションデータベースの認証情報は、AWS Secrets Manager に安全に保存する必要があります。シークレットを作成するには、「AWS Secrets Manager ユーザーガイド」の「データベースシークレットを作成する」のステップに従います。

AWS Mainframe Modernization は、このデータベースでの転送中の暗号化をサポートします。データベースで SSL を有効にする場合は、データベースの接続情報とともにデータベースシークレットに `sslMode` を必ず指定してください。`sslMode` には、`verify-full`、`verify-ca`、または `disable` のいずれかの値を指定できます。

キーの作成中に、[リソースのアクセス許可 - オプション] を選択し、[アクセス許可の編集] を選択します。暗号化されたフィールドの内容を取得するために、ポリシーエディタで次のようなリソースベースのポリシーを追加します。

```
{
   "Effect" : "Allow",
   "Principal" : {
   "Service" : "m2.amazonaws.com"
   },
   "Action" : "secretsmanager:GetSecretValue",
   "Resource" : "*"
   }
```

マネージドアプリケーションの設定

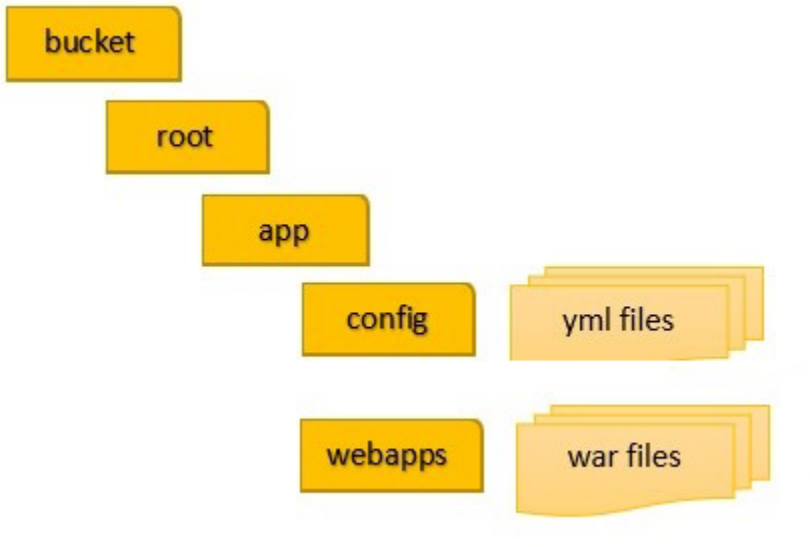
レガシーユーティリティへのアクセスを含めるようにアプリケーションを設定できます。その他のプロパティもカスタマイズできます。設定できる内容と場所を理解するために、 AWS Blu Age のモダナイズされたアプリケーションの全体的な構造を理解するのに役立ちます。

トピック

- AWS Blu Age マネージドアプリケーションの構造
- マネージドアプリケーションのユーティリティへのアクセスの設定
- AWS Blu Age エンジンの設定プロパティを追加する

AWS Blu Age マネージドアプリケーションの構造

AWS Blu Age リファクタリングパターンを使用する場合、 AWS Blu Age ランタイムエンジンは S3 バケットの `application-name`フォルダ内の次の構造を想定します。



config

プロジェクトの YAML ファイルが含まれます。これらはアプリケーションに固有の YAML ファイルapplication-planetsdemo.yamlで、通常は AWS Mainframe Modernization が提供して自動的に設定するapplication-main.yamlファイルではなく、 のような名前が付けられます。

webapps

アプリケーションの war ファイルが含まれます。これらのファイルは、モダナイゼーションプロセスの出力です。

アプリケーションには、次のオプションフォルダを含めることもできます。

jics/sql

アプリケーションの JICS データベースを初期化する initJics.sql スクリプトが含まれています。

スクリプティング

アプリケーションスクリプトが含まれており、war ファイル内で直接指定することもできます。

sql

アプリケーション SQL ファイルが含まれており、war ファイル内で直接指定することもできます。

lnk

アプリケーション LNK ファイルが含まれており、war ファイル内で直接指定することもできます。

アプリケーションの Java メモリ消費量の管理

アプリケーションの Java メモリ消費量を管理するには、tomcat.properties という名前のプロパティファイルを application-name フォルダに追加します。このファイルには 2 つのプロパティを設定でき、xms は、Java メモリの最小消費量を指定し、xmx は、Java メモリの最大消費量を指定します。次に示すのは、有効な tomcat.properties ファイルのコンテンツの例です。

```
xms=512M
xmx=1G
```

この 2 つのプロパティに指定する値は、次のいずれかの単位になります。

- バイト: 単位は指定しません。
- キロバイト: 値に K を追加します。
- メガバイト: 値に M を追加します。
- ギガバイト: 値に G を追加します。

マネージドアプリケーションのユーティリティへのアクセスの設定

AWS Blu Age でメインフレームアプリケーションをリファクタリングする場合、アプリケーションが IDCAMS、INFUTILB、SORT などのレガシープラットフォームユーティリティプログラムに依存している場合は、サポートを提供する必要がある場合があります。 AWS Blu Age リファクタリングは、モダナイズされたアプリケーションとともにデプロイされる専用のウェブアプリケーションでこのアクセスを提供します。このウェブアプリケーションには、ユーザーが提供する設定ファイルの application-utility-pgm.yml が必要です。この設定ファイルを指定しない場合、ウェブアプリケーションはユーザーのアプリケーションと一緒にデプロイできず、使用できなくなります。

トピック

- 設定プロパティ

このトピックでは、application-utility-pgm.yml 設定ファイルで指定できるすべてのプロパティとそのデフォルトについて説明します。このトピックでは、必須プロパティとオプションプロパ

ティについて説明します。次の例で示しているのは、すべての設定ファイルです。推奨する順序でプロパティを一覧表示しています。この例は、独自の設定ファイルの開始点として使用できます。

```
# If the datasource support mode is not static-xa, spring JTA transactions
 autoconfiguration must be disabled
 spring.jta.enabled: false
 logging.config: 'classpath:logback-utility.xml'

 # Encoding
 encoding: cp1047

 # Encoding to be used by INFUTILB and DSNUTILB to generate and read SYSPUNCH files
 sysPunchEncoding: cp1047

 # Utility database access
 spring.aws.client.datasources.primary.secret: `arn:aws:secretsmanager:us-
west-2:111122223333:secret:business-FfmXLG`

 treatLargeNumberAsInteger: false

 # Zoned mode : valid values = EBCDIC_STRICT, EBCDIC_MODIFIED, AS400
 zonedMode: EBCDIC_STRICT

 jcl.type: mvs

 # Unload properties
 # For date/time: if use database configuration is enabled, formats are ignored
 # For nbi; use hexadecimal syntaxe to specify the byte value
 unload:
  sqlCodePointShift: 384
  nbi:
    whenNull: "6F"
    whenNotNull: "00"
  useDatabaseConfiguration: false
  format:
    date: MM/dd/yyyy
    time: HH.mm.ss
    timestamp: yyyy-MM-dd-HH.mm.ss.SSSSSS
  chunkSize:500
  fetchSize: 500
  varCharIsNull: false
  columnFiller: space
```

```
# Load properties
# Batch size for DSNUTILB Load Task
load:
 sqlCodePointShift: 384
 batchSize: 500
 format:
   localDate: dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd
   dbDate: yyyy-MM-dd
   localTime: 'HH:mm:ss|HH.mm.ss'
   dbTime: 'HH:mm:ss'

table-mappings:
 TABLE_1_NAME : LEGACY_TABLE_1_NAME
 TABLE_2_NAME : LEGACY_TABLE_2_NAME
```

設定プロパティ

設定ファイルに次のプロパティを指定できます。

spring.jta.enabled

オプション。JTA サポートを有効化するかどうか制御します。ユーティリティでは、この値を `false` に設定することをお勧めします。

```
spring.jta.enabled : false
```

logging.config

必須。専用ロガー設定ファイルのパスを指定します。`logback-utility.xml` の名前を使用し、このファイルを最新のアプリケーションの一部として提供することを推奨します。これらのファイルを整理する一般的な方法は、すべてのロガー設定ファイルを同じ場所に置くことです。通常、`/config` は yaml 設定ファイルを含むフォルダのサブフォルダ `/config/logback` に置きます。詳細については、「Logback ドキュメント」の「第 3 章: logback の設定」を参照してください。

```
logging.config : classpath:logback-utility.xml
```

encoding

必須。ユーティリティプログラムが使用する文字セットを指定します。z/OS プラットフォームから移行する際、ほとんどの場合、この文字セットは EBCDIC バリアントであり、モダナイズされ

たアプリケーション用に構成されている文字セットと一致する必要があります。設定しない場合は、デフォルトの ASCII が使用されます。

```
encoding : cp1047
```

sysPunchEncoding

オプション。INFUTILB と DSNUTILB が SYSPUNCH ファイルの生成と読み取りに使用する文字セットを指定します。レガシープラットフォームの SYSPUNCH ファイルをそのまま使用する場合、この値は EBCDIC バリアントである必要があります。設定しない場合は、デフォルトの ASCII が使用されます。

```
sysPunchEncoding : cp1047
```

プライマリデータソースの設定

LOAD や UNLOAD などの一部のデータベース関連のユーティリティでは、データソースを介してターゲットデータベースにアクセスする必要があります。 AWS Mainframe Modernization 内の他のデータソース定義と同様に、このアクセスには を使用する必要があります AWS Secrets Manager。次のものは、Secrets Manager の適切なシークレットを指すプロパティです。

spring.aws.client.datasources.primary.secret

オプション。データソースプロパティを含む Secrets Manager のシークレットを指定します。

```
spring.aws.client.datasources.primary.secret: datasource-secret-ARN
```

spring.aws.client.datasources.primary.dbname

オプション。データベース名がデータベースシークレットに直接指定されていない場合は、dbname プロパティでターゲットデータベース名を指定します。

```
spring.aws.client.datasources.primary.dbname: target-database-name
```

treatLargeNumberAsInteger

オプション。Oracle データベースエンジンの仕様と DSNTEP2/DSNTEP4 ユーティリティの使用に関連しています。このフラグを「true」に設定すると、Oracle データベースからの大きな数値 (NUMBER (38,0)) は整数として扱われます。デフォルト: false

```
treatLargeNumberAsInteger : false
```

zonedMode

オプション。ゾーン化されたデータ型をエンコードまたはデコードするためのゾーンモードを設定します。この設定は、符号桁の表現方法に影響します。以下の値が有効です。

- EBCDIC_STRICT がデフォルトになります。符号の処理には厳密な定義を使用してください。文字セットが EBCDIC か ASCII かに応じて、符号桁表現には次の文字を使用します。
  - バイト (Cn+Dn) に対応する EBCDIC 文字で、正と負の桁範囲 (+0 ~ +9、-0 ~ -9) を表します。文字は、{、A ~ I、}、J ~ R のように表示されます
  - バイト (3n+7n) に対応する ASCII 文字で、正と負の桁範囲 (+0 ~ +9、-0 ~ -9) を表します。文字は、0 ~ 9、p ~ y のように表示されます
- EBCDIC_MODIFIED: 符号処理には変更された定義を使用します。EBDIC と ASCII のどちらでも、符号数字を表す文字のリストは同じです。つまり、+0 ~ +9 は { + A ~ I に、-0 ~ -9 は } + J ~ R にマッピングされます。
- AS400: iSeries (AS400) プラットフォームから提供されるモダナイズされたレガシーアセットに使用します。

```
zonedMode:EBCDIC_STRICT
```

jcl.type

オプション。モダナイズされた JCL スクリプトのレガシータイプを示します。IDCAMS ユーティリティは、この設定を使用して、呼び出し元の JCL がタイプ vse である場合、リターンコードを調整します。有効な値は次のとおりです。

- mvs (デフォルト)
- vse

```
jcl.type : mvs
```

データベースアンロードのユーティリティに関連するプロパティ

これらのプロパティを使用して、データベーステーブルをデータセットにアンロードするユーティリティを設定します。次のプロパティは、すべてオプションです。

この例は、使用可能なすべてのアンロードプロパティを示しています。

```
# Unload properties
 # For date/time: if use database configuration is enabled, formats are ignored
 # For nbi; use hexadecimal syntaxe to specify the byte value
 unload:
 sqlCodePointShift: 0
 nbi:
 whenNull: "6F"
 whenNotNull: "00"
 useDatabaseConfiguration: false
 format:
 date: MM/dd/yyyy
 time: HH.mm.ss
 timestamp: yyyy-MM-dd-HH.mm.ss.SSSSSS
 chunkSize: 0
 fetchSize: 0
 varCharIsNull: false
 columnFiller: space
```

sqlCodePointシフト

オプション。データに使用される SQL コードポイントのシフトを表す整数値を指定します。デフォルトは 0 です。つまり、コードポイントのシフトは行われません。この設定を、モダナイズされたアプリケーションで使用されている SQL コードポイントのシフトパラメータに合わせます。コードポイントのシフトを使用している場合、このパラメータの最も一般的な値は 384 です。

```
unload.sqlCodePointShift: 0
```

nbi

オプション。null インジケータバイトを指定します。これは、データ値の右に (文字列として) 16 進数値を加えたものです。指定できる値は次の 2 つです。

- whenNull: データ値が null の場合に 16 進数値を追加します。デフォルトは 6` です。代わりに、大きい値 FF が使われることもあります。

  ```
  unload.nbi.whenNull: "6F"
  ```

- whenNotNull: データ値が null ではないが、列が null の場合は、16 進値を追加します。デフォルトは 00 (小さい値) です。

```
unload.nbi.whenNotNull: "00"
```

useDatabaseConfiguration

オプション。日付と時刻形式を決めるプロパティを指定します。UNLOAD クエリの日付/時刻オブジェクトを処理するために使用されます。デフォルトは false です。

- true に設定すると、メインの設定ファイル (application-main.yml) の pgmDateFormat、pgmTimeFormat、および pgmTimestampFormat プロパティを使用します。
- false に設定すると、次の日付と時刻の形式を指定するプロパティを使用します。
  - unload.format.date: 日付形式のパターンを指定します。デフォルトは MM/dd/yyyy です。
  - unload.format.time: 時刻形式のパターンを指定します。デフォルトは HH.mm.ss です。
  - unload.format.timestamp: タイムスタンプ形式のパターンを指定します。デフォルトは yyyy-MM-dd-HH.mm.ss.SSSSSS です。

chunkSize

オプション。SYSREC データセットの作成に使用するデータチャンクのサイズを指定します。これらのデータセットは、並列オペレーションを伴う、データセットのアンロードオペレーションの対象となります。デフォルトは 0 (チャンクなし) です。

```
unload.chunkSize:0
```

fetchSize

オプション。データフェッチサイズを指定します。この値は、データチャンク方式を使用する場合に、一度に取得するレコードの数です。デフォルト: 0。

```
unload.fetchSize:0
```

varCharIsNull

オプション。null を許容しない varchar 列の内容が空白の場合の処理方法を指定します。デフォルトは false です。

この値を true に設定すると、アンロードの目的から、列の内容はスペース文字ではなく、空の文字列として扱われます。Oracle データベースエンジンの場合に限り、このフラグを true に設定してください。

```
unload.varCharIsNull: false
```

columnFiller

オプション。アンロードされた列を varchar 列のパディングに使用する値を指定します。指定できる値は、スペースまたは低い値です。デフォルトはスペースです。

```
unload.columnFiller: space
```

データベースのロード関連のプロパティ

これらのプロパティを使用して、データセットレコードをターゲットデータベースにロードするユーティリティ (DSNUTILB など) を設定します。次のプロパティは、すべてオプションです。

この例は、使用可能なすべてのロードプロパティを示しています。

```
# Load properties
 # Batch size for DSNUTILB Load Task
 load:
 sqlCodePointShift: 384
 batchSize: 500
 format:
 localDate: dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd
 dbDate: yyyy-MM-dd
 localTime: HH:mm:ss|HH.mm.ss
 dbTime: HH:mm:ss

 table-mappings:
 TABLE_1_NAME : LEGACY_TABLE_1_NAME
 TABLE_2_NAME : LEGACY_TABLE_2_NAME
```

sqlCodePointシフト

オプション。データに使用される SQL コードポイントのシフトを表す整数値を指定します。デフォルトは 0 です。つまり、アプリケーションはコードポイントシフトを行いません。この設定を、モダナイズされたアプリケーションで使用されている SQL コードポイントのシフトパラ

メータに合わせます。コードポイントのシフトを使用する場合、このパラメータの最も一般的な値は 384 です。

```
load.sqlCodePointShift : 384
```

batchSize

オプション。実際のバッチステートメントをデータベースに送信する前に、処理するレコード数を表す整数値を指定します。デフォルトは 0 です。

```
load.batchSize: 500
```

format

オプション。データベースロードオペレーション中の日付/時刻変換に使用する日付と時刻の形式パターンを指定します。

- `load.format.localDate`: ローカルの日付形式のパターン。デフォルトは `dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd` です。
- `load.format.dbDate`: データベースの日付形式のパターン。デフォルトは `yyyy-MM-dd` です。
- `load.format.localTime`: ローカルの時刻形式のパターン。デフォルトは `HH:mm:ss|HH.mm.ss` です。
- `load.format.dbTime`: データベースの時刻形式のパターン。デフォルトは `HH:mm:ss` です。

table-mappings

オプション。レガシーテーブル名とモダナイズされたテーブル名の間の顧客提供のマッピングコレクションを指定します。DSNUTILB ユーティリティプログラムでは、これらのマッピングを使用します。

値は、MODERN_TABLE_NAME : LEGACY_TABLE_NAME の形式で指定します。

以下がその例です。

```
table-mappings:
 TABLE_1_NAME : LEGACY_TABLE_1_NAME
 TABLE_2_NAME : LEGACY_TABLE_2_NAME
 ...
 TABLE_*N*_NAME : LEGACY_TABLE_*N*_NAME
```

> **Note**
>
> ユーティリティアプリケーションが起動すると、提供されたすべてのマッピングが明示的にログに記録されます。

AWS Blu Age エンジンの設定プロパティを追加する

AWS Blu Age ランタイムエンジンの新機能にアクセスできるリファクタリングされたアプリケーションの configフォルダにファイルを追加できます。このファイル user-properties.yml には、名前を付ける必要があります。このファイルは、アプリケーション定義を置き換えるものではなく、拡張するものです。このトピックでは、user-properties.yml ファイルに含めることができるプロパティについて説明します。

> **Note**
>
> AWS Mainframe Modernization またはアプリケーション定義によって制御されるため、一部のパラメータを変更することはできません。アプリケーションのアプリケーション定義で定義されているすべてのパラメータは、user-properties.yml で指定するパラメータよりも優先されます。

リファクタリングしたアプリケーションの構造の詳細については、「AWS Blu Age マネージドアプリケーションの構造」を参照してください。

次の図は、 AWS Blu Age サンプルアプリケーションの構造内のuser-properties.ymlファイルの場所を示しています PlanetsDemo。

```
PlanetsDemo-v1/
   ## config/
   #  ## application-PlanetsDemo.yml
   #  ## user-properties.yml
   ## jics/
   ## webapps/
```

構成プロパティのリファレンス

これは、利用可能なプロパティのリストです。すべてのパラメータは省略可能です。

トピック

Gapwalk アプリケーションプロパティ

bluesam.fileLoading.commitInterval

オプション。bluesam のコミット間隔。

タイプ: 数値

デフォルト: 100000

card.encoding

オプション。カードのエンコーディング: `useControlMVariable` で使用します。

タイプ: 文字列

デフォルト: CP1145

checkinputfilesize

オプション。ファイルサイズがレコードサイズの倍数である場合にチェックを解除するかどうかを指定します。

タイプ：ブール値

デフォルト: false

database.cursor.overflow.allowed

オプション。カーソルのオーバーフローを許可するかどうかを指定します。`true` に設定すると、カーソルの位置に関係なく、カーソル上で次の呼び出しが実行されます。`false` に設定すると、次のカーソル呼び出しを実行する前に、カーソルが最後の位置にあるかどうかを確認できます。カーソルが SCROLLABLE (SENSITIVE または INSENSITIVE) の場合にのみ有効になります。

タイプ：ブール値

デフォルト: true

dataSimplifieronInvalidNumeric.Data

オプション。無効な数値データをデコードしたときの対処方法。許可される値は、`reject`、`toleratespaces`、`toleratespaceslowvalues`、`toleratemost` です。

タイプ: 文字列

デフォルト: reject

defaultKeepExistingファイル

オプション。データセットのデフォルトの以前の値を設定するかどうかを指定します。

タイプ：ブール値

デフォルト: false

disposition.checkexistence

オプション。DISP SHR または OLD のデータセットのファイル存在確認を解除するかどうかを指定します。

タイプ：ブール値

デフォルト: false

externalSort.threshold

オプション。ソートのしきい値: 外部 (マージ) ソートに切り替えるタイミング。

タイプ: 文字列

デフォルト: null

```
externalSort.threshold: 12MB
```

forceHR

オプション。コンソール出力またはファイル出力のいずれかで、人間が読める形式の SYSPRINT を使用するかどうかを指定します。

タイプ：ブール値

デフォルト: false

forcedDate

オプション。データベース内の特定の日付と時刻を強制的に適用します。開発中およびテスト中にのみ使用してください。

デフォルト: null

```
forcedDate: 2022-08-26T12:59:58.123456+01:57
```

frozenDate

オプション。データベースの日付と時刻を固定します。開発中およびテスト中にのみ使用してください。

デフォルト: false

```
frozenDate: false
```

ims.messages.extendedSize

オプション。ims メッセージに extendedSize を設定するかどうかを指定します。

タイプ：ブール値

デフォルト: false

lockTimeout

オプション。指定した時間内にロックを取得できなかった場合のトランザクションのタイムアウト (ミリ秒単位)。

タイプ: 数値

デフォルト: 500

mapTransfo.prefixes

オプション。controlM 変数を変換するときに使用するプレフィックスのリスト。それぞれをカンマで区切ります。

タイプ: 文字列

デフォルト: &,@,%%

クエリ。useConcatCondition

オプション。キー条件がキー連結によって構築されるかどうかを指定します。

タイプ：ブール値

デフォルト: false

rollbackOnRTE

オプション。ランタイムの例外時に暗黙的な実行ユニットトランザクションをロールバックするかどうかを指定します。

タイプ：ブール値

デフォルト: false

sctThreadLimit

オプション。スクリプトをトリガーするスレッドの上限。

タイプ: 数値

デフォルト: 5

sqlCodePointシフト

オプション。SQL コードポイントのシフト。レガシー rdbms データを最新の rdbms に移行する際に発生する可能性のある制御文字のコードポイントをシフトします。例えば、Unicode 文字 `\u0180` と一致するように `384` を指定できます。

タイプ: 数値

デフォルト: 0

sqlIntegerOverflow許可

オプション。SQL 整数のオーバーフローを許可するかどうか、つまり、ホスト変数に大きな値を入れることを許可するかどうかを指定します。

タイプ：ブール値

デフォルト: false

stepFailWhen中止

オプション。ステップが失敗した場合や実行が完了した場合に、異常終了を発生させるかどうかを指定します。

タイプ：ブール値

デフォルト: true

stopExecutionWhenProgNotFound

オプション。プログラムが見つからない場合に実行を停止するかどうかを指定します。`true` に設定すると、プログラムが見つからなかった場合に実行が中断されます。

タイプ：ブール値

デフォルト: true

uppercaseUserInput

オプション。ユーザー入力を大文字にする必要があるかどうかを指定します。

タイプ：ブール値

デフォルト: true

useControlMVariable

オプション。変数置換に Control-M 仕様を使用するかどうかを指定します。

タイプ：ブール値

デフォルト: false

Gapwalk バッチスクリプトのプロパティ

encoding

オプション。バッチスクリプトプロジェクトで使用されるエンコーディング (groovy では使用されません)。有効なエンコーディング `CP1047`、`IBM930`、`ASCII`、`UTF-8` などが必要です。

タイプ: 文字列

デフォルト: ASCII

Gapwalk Blugen プロパティ

managers.trancode

オプション。ダイアログマネージャーのトランスコードマッピング。JICS トランザクションコードをダイアログマネージャーにマッピングできます。必要な形式は `trancode1:dialogManager1;trancode2:dialogManager2;` です。

タイプ: 文字列

デフォルト: null

```
managers.trancode: OR12:MYDIALOG1
```

Gapwalk CL コマンドプロパティ

commands-off

オプション。無効にするコマンドのリスト。カンマで区切られています。許可される値は、PGM_BASIC、RCVMSG、SNDRCVF、CHGVAR、QCLRDTAQ、RTVJOBA、ADDLFM、ADDPFM、RCVF、
です。既存のプログラムを無効化したり上書きしたりする場合に便利です。 PGM_BASIC はデバッグ用に設計された特定の Velocity プログラムです。

タイプ: 文字列

デフォルト: null

spring.datasource.primary.jndi-name

オプション。プライマリ JNDI (Java Naming And Directory Interface) データソース。

タイプ: 文字列

デフォルト: jdbc/primary

zonedMode

オプション。ゾーンデータ型をエンコードまたはデコードするためのモード。許可される値は、EBCDIC_STRICT/EBCDIC_MODIFIED/AS400 です。

タイプ: 文字列

デフォルト: EBCDIC_STRICT

Gapwalk CL ランナープロパティ

cl.configuration.context.encoding

オプション。CL ファイルのエンコーディング。有効なエンコーディング CP1047、IBM930、ASCII、UTF-8 などが必要です。

タイプ: 文字列

デフォルト: CP297

cl.zonedMode

オプション。制御言語 (CL) コマンドをエンコードまたはデコードするためのモード。許可される値は、EBCDIC_STRICT/EBCDIC_MODIFIED/AS400 です。

タイプ: 文字列

デフォルト: EBCDIC_STRICT

Gapwalk JHDB プロパティ

ims.programs

オプション。使用する IMS プログラムのリスト。各パラメータはセミコロン (;) で、各トランザクションはカンマ (,) で区切ります。例: ims.programs: PCP008,PCT008;PCP054,PCT054;PCP066,PCT066;PCP068,PCT068;

タイプ: 文字列

デフォルト: null

jhdb.checkpointPath

オプション。jhdb.checkpointPersistence が none ではない場合、このパラメータを使用してチェックポイント永続化パス (checkpoint.dat ファイルの保存場所) を設定できます。レジストリに含まれるすべてのチェックポイントデータがシリアル化され、指定されたフォルダにあるファイル (checkpoint.dat) にバックアップされます。このバックアップの対象となるのはチェックポイントデータ (scriptId、stepId、データベース位置、チェックポイントエリア) のみであることに注意してください。

タイプ: 文字列

デフォルト: file:./setup/

jhdb.checkpointPersistence

オプション。チェックポイント永続化モード。許可される値は、none/add/end です。新しいチェックポイントが作成されてレジストリに追加されたときに、チェックポイントを永続化するために add を使用します。サーバーのシャットダウン時にチェックポイントを永続化するために end を使用します。それ以外の値は永続化を無効にします。新しいチェックポイントがレジストリに追加されるたびに、既存のチェックポイントはすべてシリアル化され、ファイルは消去されることに注意してください。ファイル内の既存のデータへの追加ではありません。そのため、チェックポイントの数によっては、パフォーマンスに何らかの影響を与える可能性があります。

タイプ: 文字列

デフォルト: なし

jhdb.configuration.context.encoding

オプション。JHDB (Java 階層型データベース) のエンコーディング。有効なエンコーディング文字列 CP1047、IBM930、ASCII、UTF-8 などが必要です。

タイプ: 文字列

デフォルト: CP297

jhdb。identificationCardData

オプション。一部の「オペレータ ID カードデータ」を CARD パラメータで指定された MID フィールドにハードコードするために使用されます。

タイプ: 文字列

デフォルト: ""

jhdb.lterm

オプション。IMS エミュレーションの場合に共通の論理ターミナル ID を強制的に使用できます。設定されていない場合は、sessionId が使用されます。

タイプ: 文字列

デフォルト: null

jhdb.metadata.extrapath

psbs フォルダと dbds フォルダ用に、ランタイム固有の追加のルートフォルダを指定する設定パラメータ。

タイプ: 文字列

デフォルト: file:./setup/

> **Note**
>
> 現在、デプロイの制約のため、dbds ディレクトリと psbs ディレクトリをアプリケーションの config ディレクトリまたは config ディレクトリのサブディレクトリ (例: config/setup) にコピーする必要があります。
>
> ```
> config
> |- setup
>    |- dbds
>    |- psbs
> ```
>
> また、application-jhdb.yml で設定します。
> `jhdb.metadata.extrapath: file: ./config/setup/`

jhdb.navigation.cachenexts

オプション。RDBMS の階層ナビゲーションに使用されるキャッシュ時間 (ミリ秒単位)。

タイプ: 数値

デフォルト: 5000

jhdb.query。limitJoinUsage

オプション。RDBMS グラフで結合使用制限パラメータを使用するかどうかを指定します。

タイプ：ブール値

デフォルト: true

jhdb。use-db-prefix

オプション。RDBMS の階層ナビゲーションでデータベースプレフィックスを有効にするかどうかを指定します。

タイプ：ブール値

デフォルト: true

Gapwalk JICS プロパティ

jics.data.dataJsonInitLocation

オプション。アナライザーが CSD を解析して作成し、jics データベースの初期化に使用した JSON ファイルの場所

タイプ: 文字列

デフォルト: ""

jics.db。dataScriptLocation

オプション。メインフレームからの CSD エクスポートを解析してアナライザーが作成した initJics.sql スクリプトの場所。

タイプ: 文字列

デフォルト: ""

jics.db.dataTestQueryLocation

オプション。オブジェクト数を返すと予想される 1 つの sql クエリを含む sql スクリプトの場所 (例: jics プログラムテーブル内のレコード数のカウント)。カウントが 0 の場合、データベースは `jics.db.dataScriptLocation` スクリプトを使用してロードされます。それ以外の場合は、データベースのロードはスキップされます。

タイプ: 文字列

デフォルト: ""

jics.db。ddlScriptLocation

オプション。Jics ddl スクリプトの場所。.sql スクリプトを使用して jics データベーススキーマを開始できます。

タイプ: 文字列

デフォルト: ""

```
jics.db.ddlScriptLocation: ./jics/sql/jics.sql
```

jics.db.schemaTestQueryLocation

オプション。jics スキーマ内のオブジェクト数 (存在する場合) を返す一意のクエリを含む必要がある sql ファイルの場所。

タイプ: 文字列

デフォルト: ""

jics.runUnitLauncherPool.enable

オプション。JICS で実行ユニットランチャープールを有効にするかどうかを指定します。

タイプ：ブール値

デフォルト: false

jics.runUnitLauncherPool.size

オプション。JICS 内の実行ユニットランチャープールサイズ。

タイプ: 数値

デフォルト: 20

jics.runUnitLauncherPool.validationInterval

オプション: JICS 内の実行ユニットランチャープールの検証間隔 (ミリ秒単位)。

タイプ: 数値

デフォルト: 1000

jics.queues.sqs.region

オプション。JICS で使用される Amazon SQS AWS リージョン の 。パフォーマンス向上のため、デプロイしたアプリケーションと同じリージョンに設定することが推奨されますが、必須ではありません。

タイプ: 文字列

デフォルト: eu-west-1

jics.xa.agent.timeout

オプション。分散トランザクションの管理を担当する xa エージェントによって、オペレーションが完了するまでの最大時間を定義します。

タイプ: 数値

デフォルト: null

mq.queues.sqs.region

オプション。 AWS リージョン Amazon SQS MQ サービスの 。

タイプ: 文字列

デフォルト: eu-west-3

taskExecutorallowCoreThreadTimeOut。

オプション。JCIS でコアスレッドのタイムアウトを許可するかどうかを指定します。これにより、0 以外のキューと組み合わせた場合でも動的な拡大と縮小が可能になります (最大プールサイズはキューがいっぱいになった場合のみ増加するため)。

タイプ : ブール値

デフォルト: false

taskExecutorcorePoolSize。

オプション。ターミナルのトランザクションが groovy スクリプトによって開始されると、新しいスレッドが作成されます。このパラメータは、コアプールサイズを設定するために使用します。

タイプ: 数値

デフォルト: 5

taskExecutormaxPoolSize。

オプション。ターミナルのトランザクションが groovy スクリプトによって開始されると、新しいスレッドが作成されます。このパラメータを使用して、最大プールサイズ (並列スレッドの最大数) を設定します。

タイプ: 数値

デフォルト: 10

taskExecutor.queueCapacity

オプション。ターミナルのトランザクションが groovy スクリプトによって開始されると、新しいスレッドが作成されます。このパラメータを使用してキューサイズ (=

taskExecutor.maxPoolSize に到達したときの保留中のトランザクションの最大数) を設定します。

タイプ: 数値

デフォルト: 50

Gapwalk ランタイムプロパティ

cacheMetadata

オプション。データベースメタデータをキャッシュするかどうかを指定します。

タイプ：ブール値

デフォルト: true

check-groovy-file

オプション。登録する前に groovy ファイルの内容をチェックするかどうかを指定します。

タイプ：ブール値

デフォルト: true

databaseStatistics

オプション。SQL ビルダーに統計情報の収集と表示を許可するかどうかを指定します。

タイプ：ブール値

デフォルト: false

dateTimeFormat

オプション。 dateTimeFormat では、データベースの日付タイムスタンプタイプをデータ簡略化エンティティにスピルする方法について説明します。許可される値は、ISO/EUR/USA/LOCAL です。

タイプ: 文字列

デフォルト: ISO

dbDateFormat

オプション。データベースターゲットの日付形式。

タイプ: 文字列

デフォルト: yyyy-MM-dd

dbTimeFormat

オプション。データベースターゲットの時間形式。

タイプ: 文字列

デフォルト: HH:mm:ss

dbTimestampFormat

オプション。データベースターゲットのタイムスタンプ形式。

タイプ: 文字列

デフォルト: yyyy-MM-dd HH:mm:ss.SSSSSS

fetchSize

オプション。カーソルの fetchSize 値。ロード/アンロードユーティリティでチャンクを使用してデータを取得するときに使用します。

タイプ: 数値

デフォルト: 10

forceDisableSQLTrimStringType

オプション。すべての sql 文字列パラメータのトリミングを無効にするかどうかを指定します。

タイプ : ブール値

デフォルト: false

localDateFormat

オプション。ローカルの日付形式のリスト。各形式は | で区切ります。

タイプ: 文字列

localTimeFormat

オプション。ローカルの時刻形式のリスト。各形式は | で区切ります。

タイプ: 文字列

localTimestampFormat

オプション。ローカルのタイムスタンプ形式のリスト。各形式は | で区切ります。

タイプ: 文字列

デフォルト:
pgmDateFormat

オプション。プログラムで使用される日付/時刻の形式。

タイプ: 文字列

デフォルト: yyyy-MM-dd
pgmTimeFormat

オプション。pgm (プログラム) の実行に使用される時刻形式。

タイプ: 文字列

デフォルト: HH.mm.ss
pgmTimestampFormat

オプション。タイムスタンプ形式。

タイプ: 文字列

デフォルト: yyyy-MM-dd-HH.mm.ss.SSSSSS

Gapwalk ユーティリティプログラムのプロパティ

jcl.type

オプション: `.jcl` ファイルタイプ。許可される値は、`jcl/vse` です。IDCAMS ユーティリティの PRINT/REPRO コマンドは、vse jcl 以外のファイルが空の場合、4 を返します。

タイプ: 文字列

デフォルト: mvs
listcat.variablelengthpreprocessor.enabled

オプション。LISTCAT コマンドの可変長プリプロセッサを有効にするかどうかを指定します。

タイプ : ブール値

デフォルト: false

listcat.variablelengthpreprocessor.type

オプション。listcat ファイルに含まれるオブジェクトのタイプ (`listcat.variablelengthpreprocessor.enabled` が有効の場合)。許可される値は、`rdw/bdw` です。

タイプ: 文字列

デフォルト: rdw

load.batchSize

オプション。ロードユーティリティのバッチサイズ。

タイプ: 数値

デフォルト: 0

load.format.dbDate

オプション。使用するロードユーティリティデータベース形式。

タイプ: 文字列

デフォルト: yyyy-MM-dd

load.format.dbTime

オプション。ロードユーティリティデータベースの使用時間。

タイプ: 文字列

デフォルト: HH:mm:ss

load.format.localDate

オプション。使用するロードユーティリティのローカルの日付形式。

タイプ: 文字列

デフォルト: dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd

load.format.localTime

オプション。使用するロードユーティリティのローカルの時刻形式。

タイプ: 文字列

デフォルト: HH:mm:ss|HH.mm.ss

load.sqlCodePointShift

オプション。ロードユーティリティの SQL コードポイントシフト。文字シフト処理を実行します。DB2 のターゲットデータベースが PostgreSQL の場合に必要です。

タイプ: 数値

デフォルト: 0

sysPunchEncoding

オプション。syspunch エンコーディング文字セット。サポートされている値は、`Cp1047` / `ASCII` です。

タイプ: 文字列

デフォルト: ASCII

treatLargeNumberAsInteger

オプション。大きな数値を `Integer` として扱うかどうかを指定します。デフォルトでは、`BigDecimal` のように扱われます。

タイプ : ブール値

デフォルト: false

unload.chunkSize

オプション。アンロードユーティリティに使用されるチャンクサイズ。

タイプ: 数値

デフォルト: 0

unload.columnFiller

オプション。アンロードユーティリティの列フィラー。

タイプ: 文字列

デフォルト: スペース

unload.fetchSize

オプション。アンロードユーティリティでカーソルを処理する際にフェッチサイズを調整できます。

タイプ: 数値

デフォルト: 0

unload.format.date

オプション。unload.useDatabaseConfiguration が有効な場合、アンロードユーティリティで使用する日付形式です。詳細については、「unload.format.date」を参照してください。

タイプ: 文字列

デフォルト: MM/dd/yyyy

unload.format.time

オプション。unload.useDatabaseConfiguration が有効な場合、アンロードユーティリティで使用する時間形式。

タイプ: 文字列

デフォルト: HH.mm.ss

unload.format.timestamp

オプション。unload.useDatabaseConfiguration が有効な場合、アンロードユーティリティで使用するタイムスタンプ形式。

タイプ: 文字列

デフォルト: yyyy-MM-dd-HH.mm.ss.SSSSSS

unload.nbi。whenNotNull

オプション。データベースの値が NULL でない場合に追加される NULL バイトインジケータ (nbi) 値。

タイプ: 16 進数

デフォルト: 00

unload.nbi.whenNull

オプション。データベースからの値が NULL の場合に追加される Null バイトインジケータ (nbi) 値。

タイプ: 16 進数

デフォルト: 6F

unload.nbi。writeNullIndicator

オプション。アンロード出力ファイルに NULL インジケータを書き出すかどうかを指定します。

タイプ：ブール値

デフォルト: false

unload.sqlCodePointShift

オプション。アンロードユーティリティの SQL コードポイントシフト。文字シフト処理を実行します。DB2 のターゲットデータベースが PostgreSQL の場合に必要です。

タイプ: 数値

デフォルト: 0

アンロード。useDatabaseConfiguration

オプション。アンロードユーティリティで application-main.yml の日付設定と時刻設定のどちらを使用するかを指定します。

タイプ：ブール値

デフォルト: false

unload.varCharIsNull

オプション。INFTILB プログラムでこのパラメータを使用します。このパラメータを `true` に設定すると、空白 (スペース) の値を持つ null が許容されないフィールドはすべて空の文字列を返します。

タイプ：ブール値

デフォルト: false

その他のプロパティ

qtemp.cleanup.threshold.hours

オプション。qtemp.dblog をいつ有効にするかを指定します。DB パーティションの有効期間 (時間単位)。

タイプ: 数値

デフォルト: 0

qtemp.dblog

オプション。QTEMP データベースログ記録を有効にするかどうか。

タイプ : ブール値

デフォルト: false

qtemp.uuid.length

オプション。QTEMP の固有の ID の長さ。

タイプ: 数値

デフォルト: 9

スケジューラ。stand-by-if-error

オプション。ジョブスケジューラがスタンバイモードの場合にジョブの実行をトリガーするかどうかを指定します。「true」の場合、有効になってもジョブの実行はトリガーされません。

タイプ : ブール値

デフォルト: false

warmUpCache

オプション。サーバーの起動時に、すべての datacom テーブルデータをウォームアップキャッシュにロードするかどうかを指定します。

タイプ : ブール値

デフォルト: false

AWS Mainframe Modernization アプリケーション定義リファレンス

AWS Mainframe Modernization では、選択したランタイムエンジンに固有のアプリケーション定義 JSON ファイルで移行されたメインフレームアプリケーションを設定します。アプリケーション定義には、一般的な情報とエンジン固有の情報の両方が含まれます。このトピックでは、 AWS Blu Age と Micro Focus の両方のアプリケーション定義について説明し、必要な要素とオプション要素をすべて特定します。

トピック

一般的なヘッダーセクション

各アプリケーション定義は、テンプレートのバージョンとソースの場所に関する一般的な情報から始まります。現在のアプリケーション定義のバージョンは 2.0 です。バージョン 1 は引き続き動作しますが、廃止される予定です。アプリケーションを作成または更新する場合は、バージョン 2 を使用することを推奨します。

テンプレートのバージョンとソースの場所を指定するには、次の構造を使用します。

```
"template-version": "2.0",
    "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket-aaa",
                "s3-key-prefix": "v1"
```

```
            }
        }
    ]
```

> **Note**
>
> S3 ARN を s3-bucket として入力する場合は、次の構文を使用できます。
>
> ```
> "template-version": "2.0",
>   "source-locations": [
>       {
>           "source-id": "s3-source",
>           "source-type": "s3",
>           "properties": {
>               "s3-bucket": "arn:aws:s3:::mainframe-deployment-bucket-aaa",
>               "s3-key-prefix": "v1"
>           }
>       }
>   ]
> ```

template-version

必須。アプリケーション定義ファイルのバージョンを指定します。この値を変更しないでください。現在、許容されている値は 2.0 のみです。文字列で `template-version` を指定します。

source-locations

実行時にアプリケーションが必要とするファイルやその他のリソースの場所を指定します。

プロパティ

ソースの場所の詳細が表示されます。各プロパティは文字列で指定します。

- `s3-bucket` - 必須。ファイルを保存する Amazon S3 バケットの名前を指定します。
- `s3-key-prefix` - 必須。ファイルを保存する Amazon S3 バケット内のフォルダ名を指定します。

> **Note**
>
> バケット ARN ではなく、必ず Amazon S3 バケット名を指定してください。バケット内のリソースへの絶対パスは指定しないでください。

定義セクションの概要

アプリケーションの実行に必要なサービス、設定、データ、その他の一般的なリソースのリソース定義を指定します。アプリケーション定義を更新すると、 AWS Mainframe Modernization は、アプリケーション定義 JSON ファイルの以前のバージョンと現在のバージョンの両方で、`source-locations` と `definition` リストを比較して変更を検出します。

定義セクションはエンジン固有のものであり、変更される可能性があります。次のセクションでは、両方のエンジンに固有のアプリケーション定義の例を示します。

AWS Blu Age アプリケーション定義のサンプル

```
{
 "template-version": "2.0",
 "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket-aaa",
                "s3-key-prefix": "v1"
            }
        }
    ],
    "definition" : {
        "listeners": [{
            "port": 8194,
            "type": "http"
        }],
        "ba-application": {
            "app-location": "${s3-source}/murachs-v6/"
        },
        "blusam": {
            "db": {
                "nb-threads": 8,
                "batch-size": 10000,
                "name": "blusam",
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:blusam-FfmXLG"
            },
            "redis": {
                "hostname": "blusam.c3geul.ng.0001.usw2.cache.amazonaws.com",
```

```
                "port": 6379,
                "useSsl": true,
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:bluesamredis-nioefm"
            }
        }
    }
}
```

AWS Blu Age 定義の詳細

リスナー - 必須

AWS Mainframe Modernization で作成された Elastic Load Balancing を使用してアプリケーションにアクセスするために使用するポートを指定します。次の構造を使用します。

```
"listeners": [{
            "port": 8194,
            "type": "http"
        }],
```

port

必須。0 ~ 1023 のウェルノウンポートを除き、使用可能なポートであればどれでも使用できます。8192 ~ 8199 の範囲の値を使用することを推奨します。このポートで他のリスナーやアプリケーションが動作していないことを確認してください。

type

必須。現在は、http のみがサポートされます。

AWS Blu Age アプリケーション - 必須

次の構造を使用してエンジンがアプリケーションイメージファイルを取得する場所を指定します。

```
"ba-application": {
            "app-location": "${s3-source}/murachs-v6/",
            "files-directory": "/m2/mount/myfolder",
            "enable-jics": <true|false>,
            "shared-app-location": "${s3-source}/shared/"
        },
```

app-location

アプリケーションイメージファイルが保存される Amazon S3 の特定の場所。

files-directory

オプション。バッチの入出力ファイルの場所。環境レベルで設定された Amazon EFS または Amazon FSx マウントポイントのサブフォルダである必要があります。

enable-jics

オプション。JICS を有効にするかどうかを指定します。デフォルトは true です。「false」に設定すると、JICS データベースが生成されなくなります。

shared-app-location

オプション。共有アプリケーション要素が保存される Amazon S3 のその他の場所。app-location と同じ種類のアプリケーション構造を保存できます。

BluSAM - オプション

次の構造を使用して、BluSAM データベースと Redis キャッシュを指定します。

```
"blusam": {
            "db": {
                "nb-threads": 8,
                "batch-size": 10000,
                "name": "blusam",
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:blusam-FfmXLG"
            },
            "redis": {
                "hostname": "blusam.c3geul.ng.0001.usw2.cache.amazonaws.com",
                "port": 6379,
                "useSsl": true,
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:bluesamredis-nioefm"
            }
        }
```

db

アプリケーションで使用するデータベースのプロパティを指定します。データベースは Aurora PostgreSQL データベースである必要があります。次のプロパティを指定できます。

- nb-threads - オプション。blusam エンジンが依存するライトビハインドメカニズムに使用する専用スレッドの数を指定します。デフォルトは 8 です。
- batch-size - オプション。ライトビハインドメカニズムがバッチストレージオペレーションを開始するために使用するしきい値を指定します。このしきい値は、変更されたレコードを確実に保持するために、バッチストレージオペレーションを開始する変更レコード数を表します。トリガー自体は、バッチサイズと 1 秒の経過時間のいずれか早い方の組み合わせに基づいています。デフォルトは 10,000 です。
- name - オプション。データベースの名前を指定します。
- secret-manager-arn - データベース認証情報を含むシークレットの Amazon リソースネーム (ARN) を指定します。詳細については、「ステップ 4: AWS Secrets Manager データベースシークレットを作成して設定する」を参照してください。

redis

パフォーマンス向上のため、アプリケーションが必要な一時データを一元的に保存するために使用する Redis キャッシュのプロパティを指定します。Redis キャッシュの暗号化とパスワード保護の両方を実施することを推奨します。

- hostname - Redis キャッシュの場所を指定します。
- port - Redis キャッシュの通信の送受信ポート (通常は 6379) を指定します。
- useSsl - Redis キャッシュを暗号化するかどうかを指定します。キャッシュが暗号化されていない場合は、useSsl を「false」に設定します。
- secret-manager-arn - Redis キャッシュパスワードを含む、シークレットの Amazon リソースネーム (ARN) を指定します。Redis キャッシュがパスワードで保護されていない場合は、secret-manager-arn を指定しないでください。詳細については、「ステップ 4: AWS Secrets Manager データベースシークレットを作成して設定する」を参照してください。

AWS Blu Age メッセージキュー - オプション

AWS Blu Age アプリケーションの JMS-MQ 接続の詳細を指定します。

```
"message-queues": [
  {
      "product-type": "JMS-MQ",
      "queue-manager": "QMgr1",
      "channel": "mqChannel1",
      "hostname": "mqserver-host1",
      "port": 1414,
```

```
      "user-id": "app-user1",
      "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:sample/mq/test-279PTa"
  },
  {
      "product-type": "JMS-MQ",
      "queue-manager": "QMgr2",
      "channel": "mqChannel2",
      "hostname": "mqserver-host2",
      "port": 1412,
      "user-id": "app-user2",
      "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:sample/mq/test-279PTa"
  }
]
```

product-type

必須。製品タイプを指定します。現在、これは AWS Blu Age アプリケーションでは「JMS-MQ」にしかできません。

queue-manager

必須。キューマネージャーの名前を指定します。

チャンネル

必須。サーバー接続チャネルの名前を指定します。

hostname

必須。メッセージキューサーバーのホスト名を指定します。

port

必須。サーバーがリッスンするリスナーのポート番号を指定します。

user-id

オプション。指定したチャネルでメッセージキューオペレーションを実行できるユーザーアカウント ID を指定します。

secret-manager-arn

オプション。指定されたユーザーのパスワードを提供する Secrets Manager の Amazon リソースネーム (ARN) を指定します。

Micro Focus アプリケーション定義

次のサンプル定義セクションは Micro Focus ランタイムエンジン用で、必須要素とオプション要素の両方が含まれています。

```
{
 "template-version": "2.0",
 "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket-aaa",
                "s3-key-prefix": "v1"
            }
        }
    ],
    "definition" : {
        "listeners": [{
            "port": 5101,
            "type": "tn3270"
        }],
        "dataset-location": {
            "db-locations": [{
                "name": "Database1",
                "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456"
            }]
        },
        "cognito-auth-handler": {
            "user-pool-id": "cognito-idp.us-west-2.amazonaws.com/us-west-2_rvYFnQIxL",
            "client-id": "58k05jb8grukjjsudm5hhn1v87",
            "identity-pool-id": "us-west-2:64464b12-0bfb-4dea-ab35-5c22c6c245f6"
        },
        "ldap-ad-auth-handler": {
            "ldap-ad-connection-secrets": [LIST OF AD-SECRETS]
        },
        "batch-settings": {
            "initiators": [{
                "classes": ["A", "B"],
                "description": "initiator...."
            }],
            "jcl-file-location": "${s3-source}/batch/jcl"
        },
```

```
        "cics-settings": {
            "binary-file-location": "${s3-source}/cics/binaries",
            "csd-file-location": "${s3-source}/cics/def",
            "system-initialization-table": "BNKCICV"
        },
        "xa-resources" : [{
            "name": "XASQL",
            "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456",
            "module": "${s3-source}/xa/ESPGSQLXA64.so"
        }]
    }
}
```

Micro Focus 定義の詳細

Micro Focus アプリケーション定義ファイルの定義セクションの内容は、移行するメインフレームアプリケーションが実行時に必要とするリソースによって異なります。

リスナー - 必須

次の構造を使用して、リスナーを指定します。

```
"listeners": [{
    "port": 5101,
    "type": "tn3270"
}],
```

port

tn3270 の場合、デフォルトは 5101 です。他のタイプのサービスリスナーの場合、ポートは異なります。0 ~ 1023 のウェルノウンポートを除き、使用可能なポートであればどれでも使用できます。各リスナーには、固有のポートが必要です。リスナーは、ポートを共有できません。詳細については、「Micro Focus Enterprise Server ドキュメント」の「Listener Control」を参照してください。

type

サービスリスナーのタイプを指定します。詳細については、「Micro Focus Enterprise Server ドキュメント」の「Listeners」を参照してください。

データセットの場所 - 必須

次の構造を使用して、データセットの場所を指定します。

```
"dataset-location": {
            "db-locations": [{
                "name": "Database1",
                "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456"
            }],
 }
```

db-locations

移行したアプリケーションが作成するデータセットの場所を指定します。現在、 AWS Mainframe Modernization は単一の VSAM データベースからのデータセットのみをサポートしています。

- `name` - 移行したアプリケーションが作成するデータセットを含むデータベースインスタンスの名前を指定します。
- `secret-manager-arn` - データベース認証情報を含むシークレットの Amazon リソースネーム (ARN) を指定します。

Amazon Cognito 認証および承認ハンドラー - オプション

AWS Mainframe Modernization は、移行されたアプリケーションの認証と認可に Amazon Cognito を使用します。次の構造を使用して、Amazon Cognito 認証ハンドラーを指定します。

```
"cognito-auth-handler": {
            "user-pool-id": "cognito-idp.Region.amazonaws.com/Region_rvYFnQIxL",
            "client-id": "58k05jb8grukjjsudm5hhn1v87",
            "identity-pool-id": "Region:64464b12-0bfb-4dea-ab35-5c22c6c245f6"
}
```

user-pool-id

AWS Mainframe Modernization が移行されたアプリケーションのユーザーを認証するために使用する Amazon Cognito ユーザープールを指定します。ユーザープール AWS リージョン の は、AWS Mainframe Modernization アプリケーションの AWS リージョン と一致する必要があります。

client-id

認証されたユーザーがアクセスできる移行済みのアプリケーションを指定します。

identity-pool-id

認証されたユーザーがユーザープールトークンを Mainframe Modernization へのアクセスを許可する認証情報と交換する Amazon Cognito AWS ID プールを指定します。ID プール AWS リージョン の は、 AWS Mainframe Modernization アプリケーションの AWS リージョン と一致する必要があります。

LDAP ハンドラーと Active Directory ハンドラー - オプション

アプリケーションを Active Directory (AD) または任意の種類の LDAP サーバーと統合して、アプリケーションのユーザーが LDAP/AD 認証情報を承認と認証に使用できるようにすることができます。

アプリケーションを AD に統合するには

1. Micro Focus Enterprise Server ドキュメントの「Configuring Active Directory for Enterprise Server Security」で説明されている手順に従ってください。
2. アプリケーションで使用する AD/LDAP サーバーごとに、AD/LDAP の詳細を含む AWS Secrets Manager シークレットを作成します。シークレットの作成方法については、「 ユーザーガイド」の「AWS Secrets Manager シークレットを作成する」を参照してください。 AWS Secrets Manager シークレットタイプには、[その他のシークレットのタイプ] を選択し、以下のキーと値のペアを含めます。

```
{
    "connectionPath"       : "<HOST-ADDRESS>:<PORT>",
    "authorizedId"         : "<USER-FULL-DN>",
    "password"             : "<PASSWORD>",
    "baseDn"               : "<BASE-FULL-DN>",
    "userClassDn"          : "<USER-TYPE>",
    "userContainerDn"      : "<USER-CONTAINER-DN>",
    "groupContainerDn"     : "<GROUP-CONTAINER-DN>",
    "resourceContainerDn"  : "<RESOURCE-CONTAINER-DN>"
}
```

⚠ セキュリティに関する推奨事項

- の場合connectionPath、 AWS Mainframe Modernization は LDAP および LDAP over SSL (LDAPS) プロトコルをサポートします。LDAPS を使用することをお勧めします。LDAPS の方が安全性が高く、ネットワーク伝送に認証情報が表示されないためです。
- authorizedId および password には、アプリケーションの実行に必要な、最も制限の厳しい読み取り専用権限と検証権限以外は持たないユーザーの認証情報を指定することをお勧めします。
- 定期的に AD/LDAP 認証情報を更新することをお勧めします。
- awsuser または mfuser というユーザー名で AD ユーザーを作成しないでください。この 2 つのユーザー名は AWS 専用に予約されています。

次に例を示します。

```
{
    "connectionPath" : "ldaps://msad4.m2.example.people.aws.dev:636",
    "authorizedId" :
 "CN=LDAPUser,OU=Users,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "password" : "ADPassword",
    "userContainerDn" : "CN=Enterprise Server Users,CN=Micro Focus,CN=Program
 Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "groupContainerDn" : "CN=Enterprise Server Groups,CN=Micro Focus,CN=Program
 Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "resourceContainerDn" : "CN=Enterprise Server Resources,CN=Micro
 Focus,CN=Program Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev"
}
```

カスタマーマネージド KMS キーを使用してシークレットを作成します。 AWS Mainframe Modernization にシークレットに対する GetSecretValueおよび アクセスDescribeSecret許可、Decryptおよび KMS キーに対する アクセスDescribeKey許可を付与する必要があります。詳細については、「 AWS Secrets Manager ユーザーガイド」の「KMS キーのアクセス許可」を参照してください。

3. アプリケーション定義に以下を追加します。

```
"ldap-ad-auth-handler": {
```

```
    "ldap-ad-connection-secrets": [LIST OF AD/LDAP SECRETS]
}
```

次に例を示します。

```
"ldap-ad-auth-handler": {
    "ldap-ad-connection-secrets": ["arn:aws:secrets:1234:us-east-1:secret:123456"]
}
```

LDAP/AD 認証ハンドラーは Micro Focus 8.0.11 以降のバージョンで使用できます。

Batch 設定 - 必須

次の構造を使用して、アプリケーションの一部として実行されるバッチジョブに必要な詳細を指定します。

```
"batch-settings": {
            "initiators": [{
                "classes": ["A","B"],
                "description": "initiator...."
            }],
            "jcl-file-location": "${s3-source}/batch/jcls"
 }
```

initiators

移行したアプリケーションが正常に起動したときに起動し、アプリケーションが停止するまで実行を継続するバッチイニシエータを指定します。イニシエータごとに 1 つまたは複数のクラスを定義できます。複数のイニシエータを定義することもできます。例:

```
"batch-settings": {
            "initiators": [
                {
                  "classes": ["A", "B"],
                  "description": "initiator...."
                },
                {
                  "classes": ["C", "D"],
                  "description": "initiator...."
                }
```

```
            ],
            "jcl-file-location": "${s3-source}/batch/jcls"
}
```

詳細については、「Micro Focus Enterprise Server ドキュメント」の「バッチ・イニシエーターまたはプリンター SEP を定義するには」を参照してください。

- `classes` - イニシエータが実行可能なジョブクラスを指定します。最大 36 文字を使用できます。A ~ Z または 0 ~ 9 の文字を使用できます。
- `description` - イニシエータの目的を説明します。
- `jcl-file-location` - 移行したアプリケーションが実行するバッチジョブに必要な JCL ファイルの場所を指定します。

CICS 設定 - 必須

次の構造を使用して、アプリケーションの一部として実行される CICS トランザクションに必要な詳細を指定します。

```
"cics-settings": {
            "binary-file-location": "${s3-source}/cics/binaries",
            "csd-file-location": "${s3-source}/cics/def",
            "system-initialization-table": "BNKCICV"
        }
```

binary-file-location

CICS トランザクションプログラムファイルの場所を指定します。

csd-file-location

このアプリケーションの CICS リソース定義 (CSD) ファイルの場所を指定します。詳細については、「Micro Focus Enterprise Server ドキュメント」の「CICS リソース定義」を参照してください。

system-initialization-table

移行するアプリケーションが使用するシステム初期化テーブル (SIT) を指定します。SIT テーブルの名前は最大 8 文字です。A ~ Z、0 ~ 9、$、@、# を使用できます。詳細については、「Micro Focus Enterprise Server ドキュメント」の「CICS リソース定義」を参照してください。

XA リソース - 必須

次の構造を使用して、アプリケーションに必要な XA リソースの詳細を指定します。

```
"xa-resources" : [{
            "name": "XASQL",
            "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456",
            "module": "${s3-source}/xa/ESPGSQLXA64.so"
        }]
```

name

必須。XA リソースの名前を指定します。

secret-manager-arn

データベースに接続するための認証情報を含むシークレットの Amazon リソースネーム (ARN) を指定します。

module

RM スイッチモジュールの実行ファイルの場所を指定します。詳細については、「Micro Focus Enterprise Server ドキュメント」の「XAR の計画および設計」を参照してください。

AWS Mainframe Modernization データセット定義リファレンス

アプリケーションの処理に複数のデータセットが必要な場合、データセットを AWS Mainframe Modernization コンソールに 1 つずつ入力するのは効率の悪い方法です。代わりに、JSON ファイルを作成して各データセットを指定することをお勧めします。JSON ではデータセットの種類によって指定方法が異なりますが、多くのパラメータは共通です。このドキュメントでは、さまざまなタイプのデータセットをインポートするために必要な JSON の詳細について説明します。

> **Note**
>
> データセットをインポートする前に、データセットをメインフレームからに AWS に転送する必要があります。次に、データセットがメインフレーム形式から AWS で使用できる形式に変換されていることを確認する必要があります。必要に応じて、データを変換し、変換したデータセットを Amazon S3 に保存します。データセット定義 JSON ファイルにバケットとフォルダの名前を指定します。
> Micro Focus ランタイムエンジンを使用している場合は、DFCONV ユーティリティを使用してデータセットを変換することができます。このユーティリティは Micro Focus

Enterprise Developer イメージと Enterprise Server イメージに含まれています。詳細については、Micro Focus Enterprise Developer ドキュメントの「DFCONV Batch File Conversion」を参照してください。

トピック

一般的なプロパティ

いくつかのパラメータは、すべてのデータセットに共通しています。これらのパラメータは、以下の領域をカバーします。

- データセットに関する情報 (datasetName、datasetOrg、recordLength、encoding)
- インポート元の場所に関する情報。つまりデータセットのソースの場所に関する情報。これはメインフレーム上の場所ではありません。データセット (externalLocation) をアップロードした Amazon S3 の場所へのパスです。
- インポート先の場所に関する情報。つまりデータセットのターゲットの場所に関する情報。この場所は、ランタイムエンジンによってデータベースかファイルシステムのどちらかになります (storageTypeとrelativePath)。
- データセットタイプ (特定のデータセットタイプ、形式、エンコードなど) に関する情報。

各データセット定義は同じ JSON 構造になっています。次の JSON の例は、これらすべての共通パラメータを示しています。

```
{
    "dataSet": {
        "storageType": "Database",
        "datasetName": "MFI01V.MFIDEMO.BNKACC",
        "relativePath": "DATA",
        "datasetOrg": {
            "type": {
```

```
                type-specific properties
                    ...
            },
        },
    },
}
```

以下のプロパティは、すべてのデータセットに共通です。

storageType

必須。ターゲットの場所に適用されます。データセットをデータベースに保存するのか、ファイルシステムに保存するのかを指定します。想定される値は、`Database` または `FileSystem` です。

- AWS Blu Age ランタイムエンジン: ファイルシステムはサポートされていません。データベースを使用する必要があります。
- Micro Focus ランタイムエンジン: データベースとファイルシステムの両方がサポートされています。データベースには Amazon Relational Database Service または Amazon Aurora のいずれかを使用することができます。ファイルシステムには Amazon Elastic File System または Amazon FSx for Lustre を使用することができます。

datasetName

必須。メインフレームに表示されるデータセットの完全修飾名を指定します。

relativePath

必須。ターゲットの場所に適用されます。データベースまたはファイルシステム内のデータセットの相対位置を指定します。

datasetOrg

必須。データセットのタイプを指定します。可能な値は `vsam`、`gdg`、`ps`、`po`、または `unknown` です。

- AWS Blu Age ランタイムエンジン: VSAM タイプのデータセットのみがサポートされています。
- Micro Focus ランタイムエンジン: VSAM、GDG、PS、PO、Unknown のタイプのデータセットがサポートされています。

> **Note**
>
> アプリケーションが COBOL データファイルではなく PDF またはその他のバイナリファイルを必要とする場合は、次のように指定できます。
>
> ```
> "datasetOrg": {
>             "type": PS {
>                 "format": U
>             },
> ```

VSAM のデータセットリクエストフォーマットの例

- AWS Blu Age ランタイムエンジン: サポートされています。
- Micro Focus ランタイムエンジン: サポートされています。

VSAM データセットをインポートする場合は、`datasetOrg` として `vsam` を指定します。JSON は次の例のようになります。

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.VSAM.KSDS",
    "relativePath": "DATA",
    "datasetOrg": {
        "vsam": {

            "encoding": "A",
            "format": "KS",
            "primaryKey": {
                "length": 11,
                "offset": 0
            }
        }
    },
    "recordLength": {
        "min": 300,
        "max": 300
    }
```

```
},
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/catalog/data/AWS.M2.VSAM.KSDS.DAT"
}
```

VSAM データセットでは次のプロパティがサポートされています。

encoding

必須。データセットの文字セットエンコードを指定します。指定できる値は ASCII (A)、EBCDIC (E)、Unknown (?) です。

format

必須。VSAM データセットタイプとレコード形式を指定します。

- AWS Blu Age ランタイムエンジン: 指定できる値は ESDS (ES)、KSDS (KS)、RRDS (RR) です。レコード形式は固定長でも可変長でも構いません。
- Micro Focus ランタイムエンジン: 指定できる値は ESDS (ES)、KSDS (KS)、RRDS (RR) です。VSAM の定義には、レコードの形式が含まれているため、別途指定する必要はありません。

primaryKey

VSAM KSDS データセットにのみ適用されます。プライマリキーを指定します。プライマリキー名、キーオフセット、キー長で構成されます。`name` はオプションで、`offset` と `length` は必須です。

recordLength

必須。レコードの長さを指定します。固定長レコード形式では、これらの値が一致する必要があります。

- AWS Blu Age エンジン: VSAM ESDS、KSDS、RRDS では `min` がオプションで、`max` が必須です。
- Micro Focus ランタイムエンジン: `min` と `max` が必須です。

externalLocation

必須。ソースの場所、つまりデータセットをアップロードした Amazon S3 バケットを指定します。

Blue Age エンジン固有のプロパティ

AWS Blu Age ランタイムエンジンは VSAM データセットの圧縮をサポートしています。以下の例は、JSON でこのプロパティを指定する方法を示しています。

```
{
    common properties
        ...
        "datasetOrg": {
            "vsam": {
                common properties
                ...
                "compressed": boolean,
                common properties
                ...
            }
        }
}
```

圧縮プロパティを次のように指定します。

compression

オプション。このデータセットのインデックスを圧縮値として保存するかどうかを指定します。データセットが大きい場合 (通常は 100 Mb 以上)、このフラグを `true` に設定することを検討してください。

GDG Base のデータセットリクエストフォーマットの例

- AWS Blu Age ランタイムエンジン: サポートされていません。
- Micro Focus ランタイムエンジン: サポートされています。

GDG ベースデータセットをインポートする場合は、`datasetOrg` として `gdg` を指定します。JSON は次の例のようになります。

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.GDG",
    "relativePath": "DATA",
```

```
    "datasetOrg": {
        "gdg": {
            "limit": "3",
            "rollDisposition": "Scratch and No Empty"
        }
    }
}
```

GDG ベースデータセットでは以下のプロパティがサポートされています。

limit

必須。アクティブな世代の数またはバイアスを指定します。GDG ベースクラスターの最大値は 255 です。

rollDisposition

オプション。最大値に達した場合や最大値を超えた場合の世代データセットの処理方法を指定します。可能な値は、`No Scratch and No Empty`、`Scratch and No Empty`、`Scratch and Empty` または `No Scratch and Empty` です。デフォルトは `Scratch and No Empty` です。

PS または GDG 世代別データセットリクエストフォーマットの例

- AWS Blu Age ランタイムエンジン: サポートされていません。
- Micro Focus ランタイムエンジン: サポートされています。

PS または GDG 世代別データセットをインポートする場合は、`datasetOrg` として `ps` を指定します。JSON は次の例のようになります。

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.PS.FB",
    "relativePath": "DATA",
    "datasetOrg": {
        "ps": {
            "format": "FB",
            "encoding": "A"
        }
    },
    "recordLength": {
```

```
        "min": 300,
        "max": 300
    }
},
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/catalog/data/AWS.M2.PS.LSEQ"
}
}
```

PS または GDG 世代別データセットでは、次のプロパティがサポートされています。

format

必須。データセットレコードの形式を指定します。指定できる値は、F、FA、FB、FBA、FBM、FBS、FM、FS、LSEQ、U、V、VA、VB、VBA、VBM、VBS、VM、VS です。

encoding

必須。データセットの文字セットエンコードを指定します。指定できる値は ASCII (A)、EBCDIC (E)、Unknown (?) です。

recordLength

必須。レコードの長さを指定します。レコードの最小長 (min) と最大長 (max) の両方を指定する必要があります。固定長レコード形式では、これらの値が一致する必要があります。

externalLocation

必須。ソースの場所、つまりデータセットをアップロードした Amazon S3 バケットを指定します。

PO のデータセットリクエストフォーマットの例

PO データセットをインポートする場合は、datasetOrg として po を指定します。JSON は次の例のようになります。

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.PO.PROC",
    "relativePath": "DATA",
    "datasetOrg": {
```

```
        "po": {
            "format": "LSEQ",
            "encoding": "A",
            "memberFileExtensions": ["PRC"]
        }
    },
    "recordLength": {
        "min": 80,
        "max": 80
    }
},
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/source/proc/"
}
}
```

PO データセットでは以下のプロパティがサポートされています。

format

必須。データセットレコードの形式を指定します。指定できる値は、F、FA、FB、FBA、FBM、FBS、FM、FS、LSEQ、U、V、VA、VB、VBA、VBM、VBS、VM、VS です。

encoding

必須。データセットの文字セットエンコードを指定します。指定できる値は ASCII (A)、EBCDIC (E)、Unknown (?) です。

memberFileExtensions

必須。1 つ以上のファイル名拡張子を含む配列を指定すると、PDS メンバーとして含めるファイルを指定することができます。

recordLength

オプション。レコードの長さを指定します。レコードの最小長 (min) と最大長 (max) はどちらもオプションです。固定長レコード形式では、これらの値が一致する必要があります。

externalLocation

必須。ソースの場所、つまりデータセットをアップロードした Amazon S3 バケットを指定します。

> **Note**
>
> Micro Focus ランタイムエンジンの現在の実装では、PDS エントリが動的データセットとして追加されます。

AWS Mainframe Modernization におけるマネージドランタイム環境

AWS Mainframe Modernization を初めて使用する場合は、次のトピックを参照して使用を開始してください。

- AWS Mainframe Modernization とは
- AWS Mainframe Modernization のセットアップ
- AWS Mainframe Modernization の開始方法
- チュートリアル: AWS Blu Age のマネージドランタイム
- チュートリアル: Micro Focus のマネージドランタイム

AWS Mainframe Modernization のランタイム環境は、AWS コンピューティングリソース、ランタイムエンジン、およびユーザーが指定する構成の詳細を組み合わせて名前を付けたものです。ランタイム環境は 1 つ以上のアプリケーションをホストします。AWS Mainframe Modernization のアプリケーションには、移行したメインフレームのワークロードが含まれています。作成する環境に適したランタイムエンジンを選択することができます。自動リファクタリングパターンを使用している場合は AWS Blu Age を選択し、リプラットフォームパターンを使用している場合は Micro Focus を選択します。また、アプリケーションに適したコンピューティングリソースの量を選択し、オプションでストレージをランタイム環境に接続することもできます。AWSMainframe Modernization では、Amazon CloudWatch メトリクスと ログ記録が有効になっているため、ランタイム環境をモニタリングできます。

トピック

- AWS Mainframe Modernization ランタイム環境を作成する
- AWS Mainframe Modernization ランタイム環境を更新する
- AWS Mainframe Modernization ランタイム環境を停止する
- AWS Mainframe Modernization ランタイム環境を再起動する
- AWS Mainframe Modernization ランタイム環境を削除する

AWS Mainframe Modernization ランタイム環境を作成する

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization 環境を作成します。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

ランタイム環境を作成する

ランタイム環境を作成するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、環境を作成するリージョンを選択します。
3. [環境] ページで、[環境を作成] を選択します。
4. [基本情報を指定] ページで、以下の情報を入力します。

    a. [名前と説明] セクションに環境の名前を入力します。

    b. (オプション) [環境の説明] フィールドに環境の説明を入力します。この説明は、お客様と他のユーザーがランタイム環境の用途を識別するのに役立ちます。

    c. [エンジンオプション] セクションで、自動リファクタリングには [Blu Age] を、リプラットフォームには [Micro Focus] を選択します。

    d. 選択したエンジンのバージョンを選択します。

    e. (オプション) [タグ] セクションで [新しいタグを追加] を選択し、環境タグを環境に追加します。環境タグとは、AWS のリソースの整理や管理に役立つカスタム属性ラベルのことです。

    f. [次へ] をクリックします。
5. [設定を指定] ページで、次の情報を指定します。

    a. [アベイラビリティ] セクションで、[スタンドアロンランタイム環境] または [高可用性クラスター] を選択します。

    可用性パターンによって、実行時のアプリケーションの可用性が決まります。スタンドアロンは開発目的に適しています。高可用性は、常に使用可能でなければならないアプリケーション向けです。

    b. [リソース] で、インスタンスタイプと必要な容量を選択します。

これらのリソースは、ランタイム環境をホストする AWS Mainframe Modernization が管理する Amazon EC2 インスタンスです。スタンドアロンランタイム環境では、インスタンスタイプに 2 つの選択肢があり、許可されるインスタンスは 1 つだけです。高可用性ランタイム環境では、インスタンスタイプに 2 つの選択肢があり、最大 2 つのインスタンスが許可されます。

詳細については、「Amazon EC2 インスタンスタイプ」を参照してください。また、ガイダンスについては AWS メインフレームのスペシャリストに問い合わせてください。

6. [セキュリティとネットワーク] セクションで、以下の操作を行います。

   a. アプリケーションをパブリックアクセス可能にする場合は、[Allow applications deployed to this environment to be publicly accessible] を選択します。

   b. Virtual Private Cloud (VPC) を選択します。

   c. 高可用性パターンを使用している場合は、2 つ以上のサブネットを選択します。AWS Blu Age エンジンでスタンドアロンパターンを使用している場合は、2 つ以上のサブネットを選択します。Micro Focus エンジンでスタンドアロンパターンを使用している場合は、サブネットを 1 つ指定することができます。

   d. 選択した VPC のセキュリティグループを選択します。

   > **Note**
   >
   > AWS Mainframe Modernization は、ランタイム環境への接続を分散するための Network Load Balancer を作成します。セキュリティグループのインバウンドルールによって、IP アドレスからアプリケーション定義の `listener` プロパティで指定したポートへのアクセスが許可されていることを確認してください。詳細については、Network Load Balancer ユーザーガイドの「ターゲットを登録する」を参照してください。

   e. カスタマーマネージド AWS KMS key キーを使用する場合は、[KMS キー] フィールドで、[暗号化の設定をカスタマイズ] を選択します。詳細については、「AWS Mainframe Modernization サービスの保管時のデータ暗号化」を参照してください。

> **Note**
>
> デフォルトでは、AWS Mainframe Modernization は、AWS Mainframe Modernization が所有および管理する AWS KMS key を使用してデータを暗号化します。ただし、カスタマーマネージド AWS KMS key の使用を選択できます。

f. (オプション) AWS KMS key を名前または Amazon リソースネーム (ARN) で選択します。または、[AWS KMS key を作成] をクリックして AWS KMS コンソールに移動し、新しい AWS KMS key を作成します。

g. [次へ] をクリックします。

7. (オプション) [ストレージをアタッチ] ページで、1 つ以上の Amazon EFS または Amazon FSx ファイルシステムを選択し、[次へ] をクリックします。

8. [メンテナンスウィンドウ] セクションで、保留中の変更をいつ環境に適用するかを選択します。

   - [指定なし] を選択すると、AWS Mainframe Modernization が最適なメンテナンスウィンドウを選択します。
   - 特定のメンテナンスウィンドウを指定する場合は、[新しいメンテナンスウィンドウを選択] を選択します。続いて、メンテナンスウィンドウの曜日、開始時刻、期間を選択します。

   メンテナンスウィンドウの詳細については、「AWS Mainframe Modernization のメンテナンスウィンドウ」を参照してください。

   [次へ] をクリックします。

9. [確認と作成] ページで入力した情報を確認してから、[環境を作成] を選択します。

AWS Mainframe Modernization ランタイム環境を更新する

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization ランタイム環境を更新します。ランタイムエンジンのマイナーバージョンまたはランタイム環境をホストするインスタンスタイプを更新することができます。更新をすぐに適用するか、希望するメンテナンスウィンドウ中に適用するかを選択できます。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

ランタイム環境を更新する

ランタイム環境を更新するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、更新する環境が作成されたリージョンを選択します。
3. [環境] ページで、更新する環境を選択します。
4. 環境の詳細ページで、[アクション] を選択し、[環境を編集] を選択します。
5. 次の変更のいずれか、あるいはすべてを実行します。
   - [エンジンオプション] セクションで、必要なエンジンバージョンを選択します。
   - [リソース] セクションで、必要なインスタンスタイプを選択します。
   - [メンテナンスウィンドウ] セクションで、希望する日付、時間、期間を選択します。

   > **Note**
   > メンテナンスウィンドウ中に適用できる変更は、エンジンバージョンの変更のみです。その他の変更はすべて直ちに適用する必要があります。
6. [次へ] をクリックします。
7. [これらの変更を適用するタイミング] では、[今すぐ] または [次のメンテナンスウィンドウ中] を選択します。次に、[環境を更新] をクリックします。

[今すぐ] を選択した場合、環境の更新が完了するとメッセージが表示されます。

AWS Mainframe Modernization のメンテナンスウィンドウ

すべてのランタイム環境には、毎週 1 時間のメンテナンスウィンドウがあります。システムの変更はすべてこの時間に適用されます。メンテナンスウィンドウは、変更やソフトウェアのパッチなどが実行されるタイミングをコントロールする機会です。メンテナンスイベントが特定の週に予定されている場合、この 1 時間のメンテナンスウィンドウ中に開始されます。ほとんどのメンテナンスイベントは 1 時間のメンテナンスウィンドウ中に完了しますが、大規模なメンテナンスイベントは 1 時間以上かかる場合があります。

1 時間のメンテナンスウィンドウは、リージョンごとに定められた 8 時間の時間ブロックからランダムに選択されます。ランタイム環境の作成時に、必要なメンテナンスウィンドウを指定しない場合、AWS Mainframe Modernization では、ランダムに選択された曜日に対して 1 時間のメンテナンスウィンドウを割り当てます。

メンテナンスの適用中は、AWS Mainframe Modernization で環境インスタンスのリソースの一部が使用されます。メンテナンス中は、パフォーマンスにごくわずかな影響が出る場合や、アプリケーションの一部の中断が発生する場合があります。

次の表に、各リージョンに割り当てられているメンテナンスウィンドウのデフォルトの時間ブロックを示します。

リージョン名	リージョン	時間ブロック
米国東部 (バージニア北部)	us-east-1	03:00 ~ 11:00 UTC
米国西部 (オレゴン)	us-west-2	06:00 ~ 14:00 UTC
アジアパシフィック (ムンバイ)	ap-south-1	06:00 ~ 14:00 UTC
アジアパシフィック (シンガポール)	ap-southeast-1	14:00 ~ 22:00 UTC
アジアパシフィック (シドニー)	ap-southeast-2	12:00 ~ 20:00 UTC
アジアパシフィック (東京)	ap-northeast-1	13:00 ~ 21:00 UTC
カナダ (中部)	ca-central-1	03:00 ~ 11:00 UTC
欧州 (フランクフルト)	eu-central-1	13:00 ~ 21:00 UTC
欧州 (アイルランド)	eu-west-1	22:00 ~ 06:00 UTC
欧州 (ロンドン)	eu-west-2	22:00 ~ 06:00 UTC

リージョン名	リージョン	時間ブロック
欧州 (パリ)	eu-west-3	23:00 ~ 07:00 UTC
南米 (サンパウロ)	sa-east-1	13:00 ~ 21:00 UTC

AWS Mainframe Modernization ランタイム環境を停止する

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization ランタイム環境を停止します。環境を停止しても、現在のアプリケーションのデプロイは保持され、環境が再起動されるまでその環境に料金は発生しません。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

ランタイム環境を停止する

AWS Mainframe Modernization ランタイム環境を停止する必要がある場合は、環境の更新セクションと同様の手順に従います。

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization ランタイム環境を停止します。環境を停止しても、現在のアプリケーションのデプロイは保持され、環境が再起動されるまでその環境に料金は発生しません。

ランタイム環境を停止する

AWS Mainframe Modernization ランタイム環境を停止するには、環境の更新セクションと同様の手順に従います。

> **Note**
>
> 環境を停止する前に、すべてのアプリケーションを停止する必要があります。

ランタイム環境を停止するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、停止する環境が作成されたリージョンを選択します。

3. [環境] ページで、停止する環境を選択します。
4. 環境の詳細ページで、[アクション] を選択し、[環境を編集] を選択します。
5. [環境を編集] ページで [リソース] セクションを探し、必要な容量をゼロに更新します。

> **Note**
>
> 環境を停止する場合、直ちに停止することしか選択できません。

6. [次へ] をクリックします。
7. [これらの変更を適用するタイミング] では、[今すぐ] を選択します。次に、[環境を更新] をクリックします。

環境容量が更新されると、メッセージが表示されます。

AWS Mainframe Modernization ランタイム環境を再起動する

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization ランタイム環境を再起動します。ランタイム環境を再起動すると、その環境に対する請求が再開されます。

ランタイム環境を再起動する

AWS Mainframe Modernization のランタイム環境を再起動するには、環境の停止セクションと同様の手順に従います。

ランタイム環境を再起動するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、再起動する環境が作成されたリージョンを選択します。
3. [環境] ページで、再起動する環境を選択します。
4. 環境の詳細ページで、[アクション] を選択し、[環境を編集] を選択します。

> **Note**
>
> スタンドアロン環境に必要な容量は 1 にしか更新できません。ランタイム環境を再起動する場合、すぐに再起動することしか選択できません。

5. [環境を編集] ページで [リソース] セクションを探し、必要な容量をゼロから必要な容量に更新します。
6. [次へ] をクリックします。
7. [これらの変更を適用するタイミング] では、[今すぐ] を選択します。次に、[環境を更新] をクリックします。

環境の容量が更新され、環境が再起動されると、メッセージが表示されます。

AWS Mainframe Modernization ランタイム環境を削除する

AWS Mainframe Modernization コンソールを使用して、AWS Mainframe Modernization ランタイム環境を削除します。

これらの手順では、AWS Mainframe Modernization のセットアップ のステップを完了していることを前提としています。

ランタイム環境を削除する

AWS Mainframe Modernization ランタイム環境を削除する必要がある場合は、デプロイされたアプリケーションをその環境からまず削除してください。アプリケーションがデプロイされているランタイム環境は削除できません。

環境を削除するには

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクタで、削除する環境が作成されたリージョンを選択します。
3. [環境] ページで、削除する環境を選択し、[アクション] と [環境を削除] を選択します。
4. [環境を削除] ウィンドウで、`delete` を入力して削除するランタイム環境を確認し、[削除] を選択します。

AWS Mainframe Modernization でのアプリケーションテスト

AWS アプリケーションテストは AWS Mainframe Modernization のプレビューリリースであり、変更される可能性があります。この機能は、本番環境ではなくテストデータとアプリケーションでのみ使用することをお勧めします。

AWS Mainframe Modernization アプリケーションテストは、移行プロジェクトのための自動機能等価テストを提供します。

トピック

AWS Mainframe Modernization Application Testing とは

AWS Application Testing は AWS Mainframe Modernization のプレビューリリースに含まれており、変更されることがあります。この機能は、本番環境ではなくテストデータとアプリケーションでのみ使用することをお勧めします。

テストは移行プロジェクトに大きな影響を与えます。移行、モダナイゼーション、増強プロジェクトの時間と労力の最大 70% を消費する場合があります。AWSAWS Mainframe Modernization の 1 機能である Application Testing は、移行されたアプリケーションの自動機能同等性テストを実施します。機能同等性テストは、AWS クラウド のアプリケーションがメインフレーム上のアプリケーションと同等であることを検証するのに役立ちます。AWSApplication Testing では、メインフレームと AWS の間でデータセット、データベースレコード、オンライン 3270 画面の変更を自動的に比較し

ます。また、Application Testing ではテストの反復が可能なため、ターゲットアーキテクチャの更新、問題の解決、アプリケーションの完全移行に向けた進行状況に合わせて、テストシナリオを何度も実行できます。移行後も、アプリケーションテストをリグレッションテストに使用して、ランタイムエンジンやその他のコンポーネントの更新によってリグレッションが発生していないかどうかを確認できます。アプリケーションテストはコスト効率に優れています。ターゲットテスト環境は、Infrastructure as Code (IaC) の概念を活用し、ユーザー提供の CloudFormation テンプレートを使用して作成されます。Application Testing は、クラウドの伸縮性を利用して移行プロジェクトを加速します。独立したテストシナリオを並列環境で必要な数だけ実行できるため、テスト時間を短縮できます。

トピック

Application Testing を初めてお使いになる方向けの情報

Application Testing を初めて使用する方には、以下のセクションを先に読むことをお勧めします。

- アプリケーションテストの概念
- チュートリアル: CardDemo のセットアップ

Application Testing の利点

Application Testing には、移行プロセスに役立ついくつかの利点があります。

- テストの加速、俊敏性、柔軟性
- 「メインフレームで一度記録し、AWS で複数回リプレイ」というテストコンセプト
- IaC: ユーザー提供の CloudFormation テンプレートによるターゲット環境の作成
- 高度なテスト再現性

- スケーラビリティと伸縮性を念頭に置いてクラウド向けに構築
- 高度な自動化による大規模テスト
- コスト効率の良さ

AWS CloudFormation との統合

Application Testing では、AWS CloudFormation でインフラストラクチャをコードとして使用します。この設計上の選択により、テストエクスペリエンスが簡素化され、向上します。 AWS CloudFormation により自律性と独立性が得られ、ニーズに合わせた優れたインフラストラクチャを定義できます。多くのパラメータ (インスタンスサイズ、RDS インスタンス、最適なセキュリティグループ) を個別に選択したり定義したりできます。テスト条件下でアプリケーションが正しく動作するのに必要な Amazon SQS キューなどのリソースを追加できます。

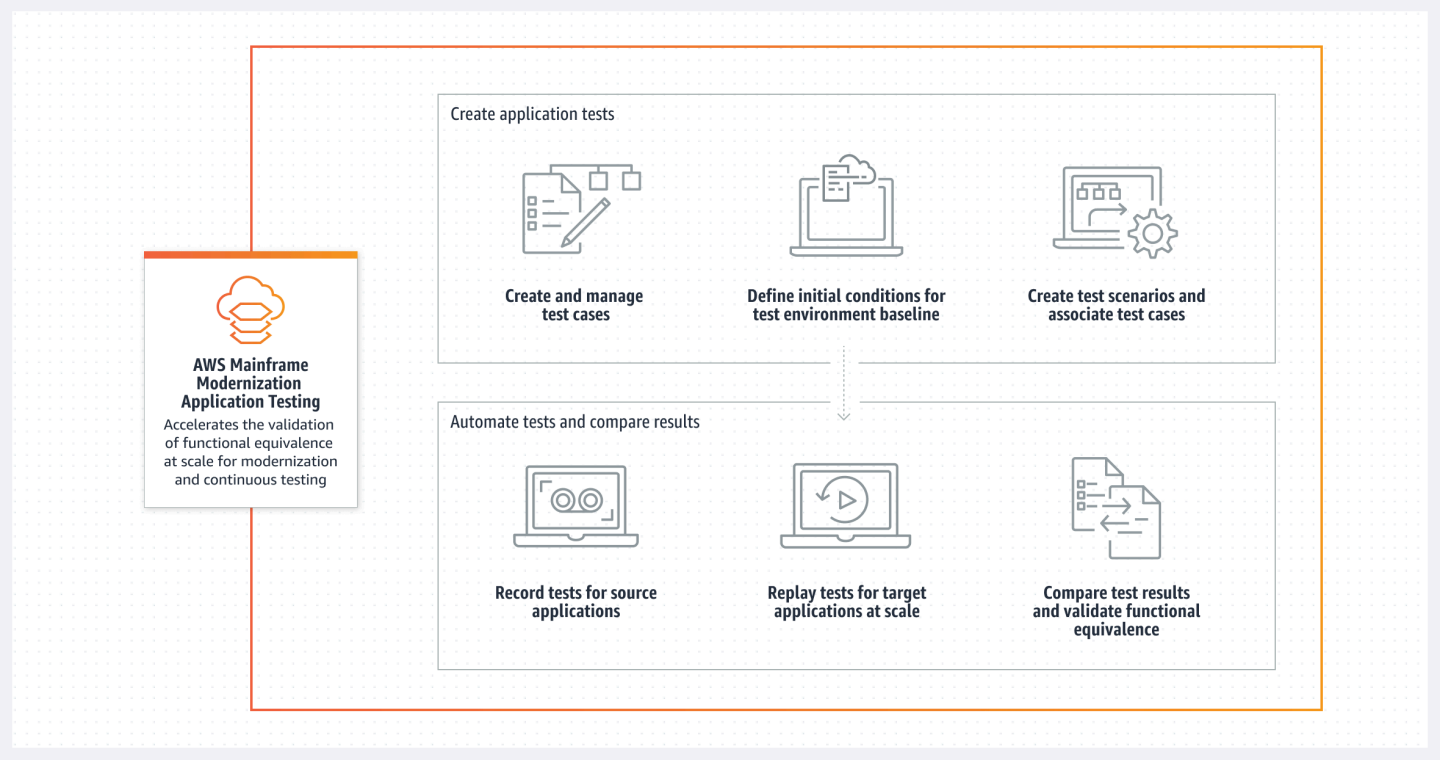
ダウンロード用に用意されている AWS CloudFormation テンプレートには、いくつかの一般的な機能が含まれています。

- Application Testing では、AWS Mainframe Modernization のランタイム環境とアプリケーションを含む完全に分離したスタックを、独自のネットワークとセキュリティ定義で作成します。同じ AWS アカウント アカウント内の他のアクターがテストアクティビティに干渉できないため、この分離したスタックには回復力があります。また、システムオペレーターがデフォルトの VPC またはセキュリティグループを変更したことが原因でテストアクティビティに失敗する、という状況も回避できます。
- セキュリティグループでは、テストに使用されるリソースへの外部アクセスを制御することもできます。例えば、データベースには機密データが含まれている場合があります。
- 完全に分離することで、VPC を共有する他のアクターがトラフィックを覗き見できないようにします。
- パフォーマンスを向上させます。例えば、テンプレートで作成される AWS Mainframe Modernization アプリケーションとその Amazon RDS データベースとの間の通信は別のネットワーク (プライベート VPC) で行われるため、他のアクターによるトラフィックの遅延を回避できます。

作成した AWS CloudFormation テンプレートにもこれらの機能を実装することをお勧めします。

Application Testing のしくみ

次の図は、Application Testing における機能同等性テストのしくみの概要を示しています。



- - AWS Mainframe Modernization File Transfer またはメインフレームデータ転送用の任意のツールを使用して、入力データをソースから AWS に転送できます。
- ソースとターゲットの両方で同じビジネスロジックを実行します。
- Application Testing は、ソースとターゲットの両方からの出力データ (データセット、リレーショナルデータベースの変更、3270 画面、ユーザーインタラクション) を自動的に比較します。テストシナリオをメインフレームで実行したら、出力データをキャプチャして AWS に転送し、テストシナリオをターゲットでリプレイします。Application Testing は、AWS でのテスト実行からの出力データとソースからの出力データを自動的に比較します。どのレコードが同一か、同等か、異なるか、または欠落しているかが一目でわかります。また、同等性ルールを定義して、同一ではないがビジネス上の意味が同じ記録を同等とみなすようにすることができます。

Application Testing のワークフローは次のステップで構成されます。

1. テストケースを作成します。テストケースはテストアクションの最小単位です。テストケースを作成するときは、ソースとターゲットの機能同等性を最もよく表す比較対象のデータタイプも特定します。
2. テストシナリオを作成します。テストシナリオは、関連するテストケースを指定の順序で実行できるようにグループ化します。

3. 初期条件を作成します。初期条件では、メインフレームにテストを記録し、AWS CloudFormation を使用して自動的に AWS に同じ状態を生み出す方法を明確化します。
4. ソースで記録し、ターゲットでリプレイします。入力データセットと出力データセットをメインフレームでキャプチャし、AWS にアップロードします。次に、AWS でテストシナリオをリプレイします。
5. ソースデータセットとターゲットデータセットを比較します。Application Testing では、ソースとターゲットの両方の出力データセットが自動的に比較されるため、正しいものと正しくないものが一目でわかります。

テストシナリオの最終的なアクションとプロセス全体の目標はどちらも、ソースとターゲットのテストランの不一致を特定することです。Application Testing では、テスト実行中にすべてのインタラクションチャネルでキャプチャされたデータについて、ソースバージョンとターゲットバージョンを比較します。また、関連データの最終状態 (テストケースで定義されているとおり) も比較します。

関連サービス

Application Testing は AWS Mainframe Modernization の機能の 1 つです。また、AWS CloudFormation でインフラストラクチャをコードとして使用することで、テストの再現性、自動化、コスト効率を確実に実現します。詳細については、以下を参照してください。

- AWS Mainframe Modernization
- AWS CloudFormation

Application Testing へのアクセス

AWS Mainframe Modernization コンソールから Application Testing にアクセスするには、左側のナビゲーションで [Application Testing] を選択します。

Application Testing の料金

Application Testing の料金は、「AWS Mainframe Modernization の料金」を参照してください。

AWS Mainframe Modernization アプリケーションテストの概念

AWS アプリケーションテストは AWS Mainframe Modernization のプレビューリリースであり、変更される可能性があります。この機能は、本番環境ではなくテストデータとアプリケーションでのみ使用することをお勧めします。

AWS アプリケーションテストでは、他のテストサービスやソフトウェアパッケージが使用する可能性のある用語を、少し異なる意味で使用します。以下のセクションでは、 AWS Mainframe Modernization アプリケーションのテストでこの用語を使用する方法について説明します。

トピック

- テストケース
- テストシナリオ
- テストプロジェクト
- 初期条件
- 記録 (キャプチャ)
- リプレイ
- 比較
- データベースの比較
- データセットの比較
- 比較ステータス
- 同等性ルール
- 最終状態のデータセットの比較
- 状態と進行状況のデータベース比較
- 機能同等性 (FE)
- 3270 のオンライン画面比較
- 記録
- リプレイデータ
- 参照データ
- 記録、リプレイ、比較
- 差異

- 同等性
- ソースアプリケーション
- ターゲットアプリケーション

テストケース

テストケースは、テストワークフローにおいて最もアトミックなアクションの個別単位です。通常、テストケースはデータを変更する、独立したビジネスロジックの単位を表すために使用されます。比較はテストケースごとに行われます。テストケースはテストシナリオに追加されます。テストケースには、テストケースが変更するデータアーティファクト (データセット、データベース) に関するメタデータと、テストケースの実行時にトリガーされるビジネス機能 (バッチジョブ、3270 のインタラクティブダイアログなど) に関するメタデータが含まれています。例えば、データセットの名前やコードページなどです。

入力データ → テストケース → 出力データ

テストケースはオンラインタイプまたはバッチタイプのいずれかとなります。

- オンラインテストケースは、ユーザーがインタラクティブなスクリーンダイアログ (3270) を実行して、新しいビジネスデータ (データベースやデータセットのレコード) の読み取り、変更、作成を行うテストケースです。
- バッチテストケースは、新しいビジネスデータ (データセットやデータベースレコード) の読み取り、処理、変更、生成のためにバッチを送信する必要があるテストケースです。

テストシナリオ

テストシナリオは、1 つずつ順番に実行される一連のテストケースです。リプレイはテストシナリオレベルで行われます。テストシナリオがリプレイされると、テストシナリオのすべてのテストケースがターゲットのテスト環境で実行されます。リファレンステストアーティファクトとリプレイテストアーティファクトを比較したときに違いがあれば、テストケースレベルで表示されます。

例えば、テストシナリオ A は以下のようになります。

テストケース 1、テストケース 2、テストケース 3 など。

テストプロジェクト

テストプロジェクトは、目的のテストマイルストーンに到達するためのテストシナリオ群です。例えば、特定のアプリケーションの移行を 1 つのテストプロジェクトと見なすことができます。テストシナリオをテストプロジェクトにグループ化することで、テスト管理者は合格したテストと不合格だったテストを含むテストプロジェクトのステータスを追跡できます。

初期条件

初期条件には、作成する必要のある一連のリソース (コンピューティング、データストアなど) と、テストシナリオを実行する前に作成したリソースで復元する必要のあるアプリケーションデータが含まれます。これにより、ターゲットのテスト環境の基準が作成されます。これにより、 AWS CloudFormation テンプレートを提供できます。テンプレートを使用してターゲットのテスト環境を作成し、テストシナリオでデータベースレコードが変更された場合は、任意でソースデータベースから DDL を抽出します。すべてのテストシナリオは初期条件に関連付けられます。初期条件は複数のテストシナリオに関連付けることができます。結果の一貫性を保ちながら再現性を確保し、データがすでに変更されていることによる誤検出を回避するには、各テストシナリオを実行する前に初期条件を復元する必要があります。

データベースレコードを変更するテストケースを含むテストシナリオでは、初期条件はソースデータベースのスキーマとテーブルの DDL エクスポートも参照します。

記録 (キャプチャ)

記録はテストシナリオレベルで行われます。記録時には、比較対象となるソースメインフレームのアーティファクト、データセット、リレーショナルデータベース CDC ジャーナルを含む Amazon S3 ロケーションを指定する必要があります。これらはソースメインフレームからの参照データと見なされます。リプレイ中、生成されたリプレイデータは記録された参照データと比較され、アプリケーションの同等性が確認されます。

リプレイ

リプレイはテストシナリオレベルで行われます。リプレイ中、 AWS Mainframe Modernization アプリケーションテストは、 CloudFormation関連する初期条件で参照されているスクリプトを使用してターゲットテスト環境を作成し、アプリケーションを実行します。リプレイ時に変更されたデータセットとデータベースレコードがキャプチャされ、メインフレームからの参照データと比較されます。通常、メインフレームに 1 回記録し、機能同等性が得られるまで複数回リプレイします。

比較

リプレイが正常に終了すると、自動的に比較が行われます。比較の際には、記録フェーズでアップロードしてキャプチャした参照データが、リプレイフェーズで生成されたリプレイデータと比較されます。比較は、データセット、データベースレコード、オンライン画面の個別のテストケースレベルで個別に行われます。

データベースの比較

アプリケーションテストでは、ソースアプリケーションとターゲットアプリケーションの間でデータベースレコードの変更を比較する際に、状態と進行状況の照合機能が採用されます。状態と進行状況の照合では、プロセスの最後にテーブル行を比較するのとは異なり、INSERT、UPDATE、DELETE ステートメントを実行するごとに差異を比較します。状態と進行状況の照合は他の方法よりも効率がよく、変更されたデータのみを比較し、トランザクションフロー内の自己修正エラーを検出することで、迅速で正確な比較が可能になります。アプリケーションテストでは、CDC (変更データキャプチャ) テクノロジーを使用して、リレーショナルデータベースの個々の変更を検出し、ソースとターゲットの間で比較できます。

リレーショナルデータベースは、SQL INSERT、UPDATE、DELETE などの DML (データ修正言語) ステートメントを使用してテストされたアプリケーションコードによってソースとターゲットで変更が生成されますが、アプリケーションがストアドプロシージャを使用している場合や、データベーストリガーが一部のテーブルに設定されている場合、参照整合性を保証するために CASCADE DELETE が使用されて自動的に追加の削除がトリガーされる場合にも間接的に生成されます。

データセットの比較

アプリケーションテストは、ソース (記録) システムとターゲット (リプレイ) システムで生成された参照データセットとリプレイデータセットを自動的に比較します。

データセットを比較するには

1. ソースとターゲットの両方で同じ入力データ (データセット、データベース) で開始します。
2. ソースシステム (メインフレーム) でテストケースを実行します。
3. 生成されたデータセットをキャプチャし、Amazon S3 バケットにアップロードします。CDC ジャーナル、画面、データセット AWS を使用して、ソースから に入力データセットを転送できます。
4. テストケースを記録したときにメインフレームデータセットがアップロードされた Amazon S3 バケットの場所を指定します。

リプレイが完了すると、アプリケーションテストは出力参照データセットとターゲットデータセットを自動的に比較して、レコードが同一か、同等か、異なるか、欠落しているかを示します。例えば、ワークロードの実行時と関連する (実行時 + 1 日、当月末など) 日付フィールドは、自動的に同等とみなされます。また、オプションで同等性ルールを定義して、同一ではないがビジネス上の意味が同じレコードに同等のフラグを付けられるようにすることもできます。

比較ステータス

アプリケーションテストでは、「IDENTICAL」、「EQUIVALENT」、「DIFFERENT」という比較ステータスを使用します。

IDENTICAL

ソースデータとターゲットデータはまったく同じです。

EQUIVALENT

ソースとターゲットのデータには、ワークロードが実行された時点と比較した場合に、日付やタイムスタンプなど、機能同等性に影響を与えず同等であると考えられる見せかけの違いが含まれています。同等性ルールを定義して、これらの差異を特定できます。リプレイしたすべてのテストシナリオを参照テストシナリオと比較して、IDENTICAL または EQUIVALENT のステータスが表示されたら、テストシナリオは機能的に同等であるということになります。

DIFFERENT

ソースデータとターゲットデータには、データセット内のレコード数が異なる、同じレコードの値が異なるなどの差異があります。

同等性ルール

同等の結果と見なすことができる、見せかけの差異を識別するための一連のルール。オフラインの機能同等性テスト (OFET) では、ソースシステムとターゲットシステムの間で結果の一部に違いが生じることは避けられません。例えば、更新タイムスタンプは設計によって異なります。同等性ルールは、これらの違いを調整し、比較時に誤検出が発生しないようにする方法を明確にします。例えば、特定のデータ列の日付がランタイム + 2 日である場合、同等性ルールはそれを記述し、参照記録の同じ列と厳密に一致する値ではなく、ターゲットシステムでのランタイムに 2 日を足した時間を受け入れます。

最終状態のデータセットの比較

作成または変更されたデータセットの最終状態。初期状態からデータセットに加えられたすべての変更または更新を含みます。データセットの場合、アプリケーションテストはテストケース実行の最後にそれらのデータセットのレコードを調べ、結果を比較します。

状態と進行状況のデータベース比較

データベースレコードに加えられた変更を、個々の一連の DML (削除、更新、挿入) ステートメントとして比較します。アプリケーションテストでは、ソースデータベースからターゲットデータベースへの個々の変更 (テーブルの行の挿入、更新、削除) を比較し、個々の変更の違いを見極めます。例えば、個々の INSERT ステートメントを使用して、ソースデータベースとターゲットデータベースの値が異なる行をテーブルに挿入できます。

機能同等性 (FE)

2 つのシステムは、入力データが同じ場合、観測可能なすべての操作で同じ結果が得られる場合、機能的に同等であるとみなされます。例えば、同じ入力データから (画面、データセットの変更、データベースの変更を通じて) 同一の出力データが生成される場合、2 つのアプリケーションは機能的に同等であるとみなされます。

3270 のオンライン画面比較

メインフレーム 3270 画面の出力と、ターゲットシステムが の AWS Blu Age ランタイムで実行されている場合のモダナイズされたアプリケーションウェブ画面の出力を比較します AWS クラウド。また、ターゲットシステムが AWS クラウドの Micro Focus ランタイムで実行されている場合、メインフレーム 3270 画面の出力と再ホストされているアプリケーションの 3270 画面の出力を比較します。

記録

データの周知の状態を復元し、ソースシステムで参照テストシナリオの参照データを (1 つまたは複数のテストケースを順番に) キャプチャまたは記録する処理。

リプレイデータ

リプレイデータは、ターゲットテスト環境でテストシナリオをリプレイして生成したデータを記述するために使用されます。例えば、 AWS Mainframe Modernization サービスアプリケーションでテス

トシナリオが実行されている場合、再生データが生成されます。その後、リプレイデータはソースからキャプチャされた参照データと比較されます。ターゲット環境でワークロードをリプレイするたびに、新世代のリプレイデータが生成されます。

参照データ

参照データは、ソースメインフレームでキャプチャされたデータを記述するために使用されます。リプレイ (ターゲット) で生成されたデータが比較される参照元です。多くの場合、参照データを作成するメインフレームのレコードごとに、リプレイが複数回行われます。これは、通常、ユーザーがメインフレームでアプリケーションの正しい状態をキャプチャし、モダナイズされたターゲットアプリケーションでテストケースをリプレイして同等性を検証するためです。バグが見つかった場合は、修正され、テストケースが再びリプレイされます。多くの場合、リプレイ、バグの修正、発生を検証するためのリプレイを複数サイクル繰り返します。これは、テストの「1 回キャプチャ、複数回リプレイ」パラダイムと呼ばれます。

記録、リプレイ、比較

アプリケーションテストは次の 3 つのステップで行われます。

- 記録: テストシナリオの各テストケースについて、メインフレーム上に作成された参照データをキャプチャします。これには、オンラインの 3270 画面、データセット、データベースレコードなどがあります。
  - オンラインの 3270 画面では、Blu Insights ターミナルエミュレータを使用してソースワークロードをキャプチャする必要があります。Blu Insights の詳細については、Blu Insights のドキュメントを参照してください。
  - データセットの場合、FTP や Mainframe Modernization のデータセット転送サービス部分などの一般的なツールを使用して、 AWS メインフレーム上の各テストケースによって生成されたデータセットをキャプチャする必要があります。
  - データベースの変更については、「Precisely を使用した AWS Mainframe Modernization データレプリケーション」のドキュメントを参照し、変更を含む CDC ジャーナルをキャプチャして生成してください。
- リプレイ: テストシナリオをターゲット環境でリプレイします。テストシナリオで指定されたすべてのテストケースを実行します。データセット、リレーショナルデータベースの変更、3270 画面など、個々のテストケースで作成された指定のデータタイプは、自動的にキャプチャされます。これらのデータはリプレイデータと呼ばれ、記録フェーズでキャプチャされた参照データと比較されます。

> **Note**
>
> リレーショナルデータベースの変更には、初期条件 CloudFormation テンプレートの DMS 固有の設定オプションが必要です。

- 比較: ソーステストの参照データとターゲットのリプレイデータが比較され、結果が同一のデータ、異なるデータ、同等のデータ、または欠落しているデータとして表示されます。

差異

データを比較することで、参照データセットとリプレイデータセットの間に差異が検出されたことを示します。例えば、オンラインの 3270 画面のフィールドで、ソースメインフレームとのモダナイズされたターゲットアプリケーションとの間で、ビジネスロジックの観点から見て異なる値が表示される場合、そのフィールドには差異があるとみなされます。もう 1 つの例として、ソースアプリケーションとターゲットアプリケーションでデータセットのレコードが同一ではない場合があります。

同等性

同等レコードとは、参照データセットとリプレイデータセットでは差異があるが、ビジネスロジックの観点からすると異なるものとして扱うべきではないレコードのことです。例えば、データセットが作成されたときのタイムスタンプ (ワークロード実行時間) を含むレコードなどです。カスタマイズ可能な同等性ルールを使用すると、参照データとリプレイデータの値が異なっていても、このような誤検出の差異を同等として扱うようにアプリケーションテストに指示できます。

ソースアプリケーション

比較対象となるソースメインフレームアプリケーション。

ターゲットアプリケーション

テストを実施する新規または変更後のアプリケーション。ソースアプリケーションとターゲットアプリケーション間の機能同等性を達成するために、ソースアプリケーションと比較されます。ターゲットアプリケーションは通常、 AWS クラウドで実行されています。

チュートリアル: CardDemo サンプルアプリケーションのセットアップ

AWS Application Testing は AWS Mainframe Modernization のプレビューリリースに含まれており、変更されることがあります。この機能は、本番環境ではなくテストデータとアプリケーションでのみ使用することをお勧めします。

このチュートリアルでは、Micro Focus on AWS Mainframe Modernization マネージドサービス、および AWS Mainframe Modernization Application Testingなどの機能を使用してリプラットフォームするための CardDemo サンプルアプリケーションをセットアップするのに役立つ AWS CloudFormation スタックを作成します。このチュートリアルでは、スタックの作成に使用できるサンプル AWS CloudFormation テンプレートについて説明します。必要なアプリケーションアーティファクトの ZIP ファイルも提供しています。サンプルテンプレートは、データベース、ランタイム環境、アプリケーション、および完全に分離されたネットワーク環境をプロビジョニングします。

このテンプレートは複数の AWS リソースを作成します。このテンプレートからスタックを作成した場合、それらに対する料金が発生します。

前提条件

- `IC3-card-demo-zip` と `datasets_Mainframe_ebcdic.zip` をダウンロードして解凍します。これらのファイルには AWS Application Testing で使用する CardDemo サンプルとサンプルデータセットが含まれています。
- CardDemo ファイルやその他のアーティファクトを保存する Amazon S3 バケットを作成します。例えば、`my-carddemo-bucket` です。

ステップ 1: CardDemo をセットアップする準備を整える

CardDemo サンプルファイルをアップロードし、CardDemo アプリケーションを作成する AWS CloudFormation テンプレートを編集します。

1. 解凍しておいた `datasets_Mainframe_ebcdic` と `IC3-card-demo` のフォルダをバケットにアップロードします。
2. バケットから `aws-m2-math-mf-carddemo.yaml` AWS CloudFormation テンプレートをダウンロードします。`IC3-card-demo`フォルダーにあります。

3. `aws-m2-math-mf-carddemo.yaml` AWS CloudFormation テンプレートを次のように編集します。

   - `BucketName` パラメータを、定義済みのバケットの名前 (`my-carddemo-bucket` など) に変更します。
   - `ImportJsonPath` をバケット内の `mf-carddemo-datasets-import.json` ファイルの場所に変更します。たとえば、`s3://my-carddemo-bucket/IC3-card-demo/mf-carddemo-datasets-import.json` この値を更新して、出力 `M2ImportJson` の値が正しくなるようにします。
   - (オプション) `EngineVersion` と `InstanceType` のパラメータを標準に合わせて調整します。

   > **Note**
   >
   > `M2EnvironmentId` と `M2ApplicationId` の出力は変更しないでください。Application Testing はこれらの値を使用して、相互作用するリソースを特定します。

ステップ 2: 必要なリソースをすべて作成する

カスタマイズした AWS CloudFormation テンプレートを実行して、このチュートリアルを正常に完了するのに必要なすべてのリソースを作成します。このテンプレートは、テストで使用できるように CardDemo アプリケーションをセットアップします。

1. AWS CloudFormation コンソールで [スタックの作成] を選択し、[新しいリソース (標準) の使用] を選択します。
2. [前提条件 - テンプレートの準備] で、[テンプレートの準備完了] を選択します。
3. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択し、[ファイルの選択] を選択します。
4. `aws-m2-math-mf-carddemo.yaml` をダウンロードした場所に移動してそのファイルを選択し、[次へ] を選択します。
5. [スタックの詳細を指定] で、一覧の中から見つけやすいようにスタックの名前を入力し、[次へ] を選択します。
6. [スタックオプションの設定] で デフォルト値をそのまま使用し、[次へ] を選択します。

7. [確認] で AWS CloudFormation が作成中の内容を確認し、[送信] を選択します。

AWS CloudFormation がスタックを作成するには約 10～15 分かかります。

> **Note**
>
> テンプレートは、作成するリソースの名前に固有のサフィックスを追加するように設定されています。つまり、このスタックテンプレートの複数のインスタンスを並行して作成できます。これは Application Testing の重要な機能であり、これにより複数のテストシナリオを同時に実行することができます。

ステップ 3: アプリケーションのデプロイと起動

AWS CloudFormation で作成した CardDemo アプリケーションをデプロイし、実行中であることを確認します。

1. AWS Mainframe Modernization コンソールを開き、左側のナビゲーションで [アプリケーション] を選択します。
2. `aws-m2-math-mf-carddemo-abc1d2e3` などの名前の CardDemo アプリケーションを選択します。
3. [アクション] を選択してから、[アプリケーションのデプロイ] を選択します。
4. [環境]で、アプリケーションに対応するランタイム環境を選択します。名前の末尾に同じ固有の識別子が付加されます。例えば、`aws-m2-math-mf-carddemo-abc1d2e3` です。
5. デプロイを選択します。アプリケーションが正常にデプロイされ、[準備] 完了状態になるまで待ちます。
6. アプリケーションを選択し、[アクション]、[アプリケーションを開始] の順に選択します。アプリケーションが [実行中] 状態になるまで待ちます。
7. アプリケーションの詳細ページで、実行中のアプリケーションに接続するために必要な [ポート] と [DNS ホスト名] をコピーします。

ステップ 4: 初期データをインポートする

CardDemo サンプルアプリケーションを使用するには、初期データセットをインポートする必要があります。以下の手順を実行します。

1. `mf-carddemo-datasets-import.json` ファイルをダウンロードします。
2. 任意のテキストエディタでファイルを編集します。
3. `s3Location` パラメータを見つけて、作成した Amazon S3 バケットを指定するように値を更新します。
4. `s3Location` のすべての箇所に同じ変更を行い、ファイルを保存します。
5. Amazon S3 コンソールにログインし、前に作成したバケットに移動します。
6. カスタマイズした `mf-carddemo-datasets-import.json` ファイルをアップロードします。
7. AWS Mainframe Modernization コンソールを開き、左側のナビゲーションで [アプリケーション] を選択します。
8. CardDemo アプリケーションを選択します。
9. [ファイル] を選択して、[インポート] を選択します。
10. カスタマイズ済みの JSON ファイルをアップロードした Amazon S3 内の場所に移動し、[送信] を選択します。

このジョブは 23 個のデータセットをインポートします。インポートジョブの結果をモニタリングするには、コンソールを確認します。すべてのデータセットが正常にインポートされたら、アプリケーションに接続します。

> **Note**
>
> Application Testing でこのテンプレートを使用すると、出力 `M2ImportJson` がインポートプロセスを自動的に処理します。

ステップ 5: CardDemo アプリケーションに接続する

選択した 3270 エミュレータを使用して CardDemo サンプルアプリケーションに接続します。

- アプリケーションの実行中に、3270 エミュレータを使用してアプリケーションに接続し、必要に応じて DNS ホスト名とポート名を指定します。

  例えば、オープンソースの c3270 エミュレータを使用している場合、コマンドは次のようになります。

```
c3270 -port port-number DNS-hostname
```

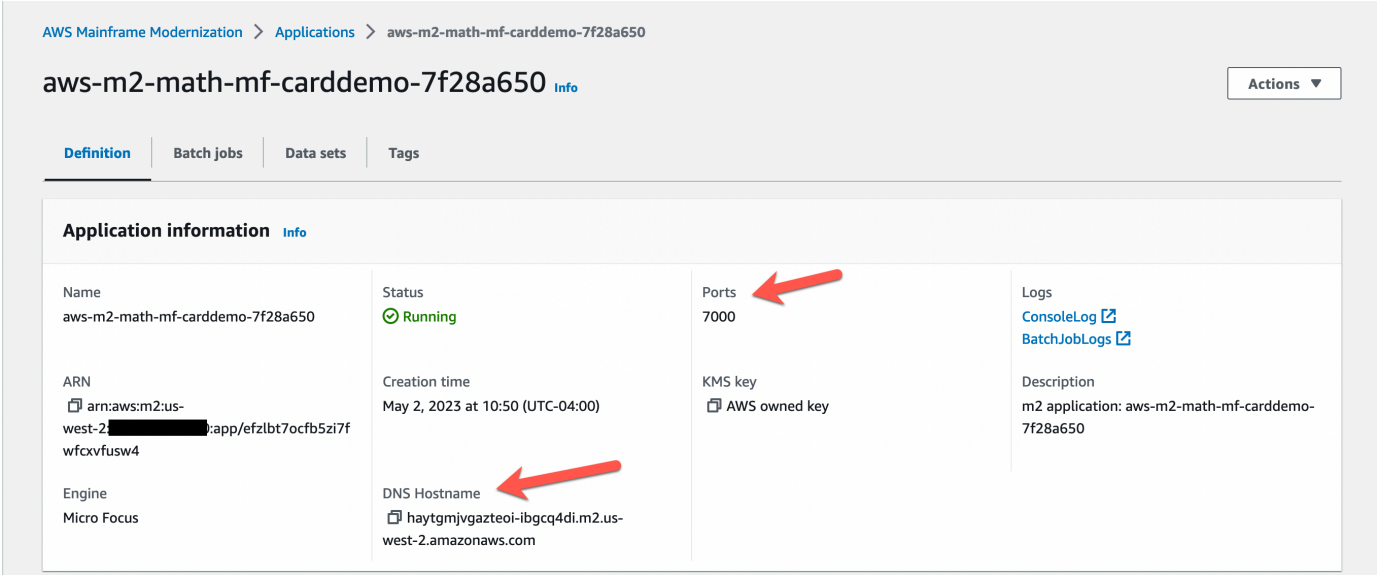
port

アプリケーション詳細ページで指定されているポート。例えば 6000 です。

ホスト名

アプリケーション詳細ページで指定されている DNS ホスト名。

次の図はポートと DSN ホスト名が見つかる場所を示しています。



チュートリアル: Amazon EC2 にデプロイされた CardDemo for AWS Blu Age を使用した AWS Mainframe Modernization アプリケーションのリプレイと比較

AWS Application Testing は AWS Mainframe Modernization のプレビューリリースに含まれており、変更されることがあります。この機能は、本番環境ではなくテストデータとアプリケーションでのみ使用することをお勧めします。

このチュートリアルでは、テストワークロードを再生して、Amazon EC2 にデプロイされた AWS Blu Age で実行されている CardDemo アプリケーションと比較するために必要なステップを完了します。

ステップ 1: AWS Blu Age Amazon EC2 Amazon マシンイメージ (AMI) を取得する

Amazon AWSAWS Amazon EC2 AMI で Blu Age にアクセスするために必要なオンボーディング手順については、「Blu Age ランタイム (Amazon EC2) セットアップチュートリアル」の指示に従ってください。 AWS Amazon EC2

ステップ 2: AWS Blu Age AMI を使用して Amazon EC2 インスタンスを起動する

1. AWS 認証情報をセットアップします。
2. Amazon S3 バケットから 3.5.0 Amazon EC2 AMI バイナリファイルの場所 (CLI のみ/AWS Blu Age バージョン) を特定します。

```
aws s3 ls s3://aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1/
aws s3 ls s3://aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1/3.5.0/AMI/
```

> **Note**
>
> Application Testing 機能は、本番環境の 4 つのリージョン (us-east-1、sa-east-1、eu-central-1、ap-southeast-2) でのみ使用できます。

3. 次のコマンドを使用して、アカウントの AMI を復元します。

```
aws ec2 create-restore-image-task --object-key 3.5.0/AMI/ami-0182ffe3b9d63925b.bin
 --bucket aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1 --region eu-west-1 --name
 "AWS BLUAGE RUNTIME AMI"
```

> **Note**
>
> AMI bin ファイル名と AMI を作成するリージョンを置き換えます。

4. Amazon EC2 インスタンスを作成した後、Amazon EC2 イメージカタログの Amazon S3 バケットから AMI に復元された正しい AMI ID を確認できます。

> **Note**
>
> このチュートリアルでは、AMI ID は ami-0d0fafcc636fd1e6d です。別の設定ファイルではこの ID を提供されている ID に変更する必要があります。

1. aws ec2 create-restore-image-taskが失敗した場合は、次のコマンドを使用して Python と CLI のバージョンを確認します。

```
aws --version
```

> **Note**
>
> Python のバージョンは 3 以上で、CLI のバージョンは 2 以上である必要があります。

2. これらのバージョンが古い場合は、CLI を更新する必要があります。CLI を更新するには:
   a. 「AWS CLI の最新バージョンをインストールまたは更新する」のステップに従います。
   b. CLI v1 は、次のコマンドを使用して削除します。

```
sudo yum remove awscli
```

   c. 次のコマンドを使用して CLI v2 をインストールします。

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
 "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

   d. 最後に、次のコマンドを使用して Python と CLI のバージョンを確認します。

```
aws --version
```

3. その後、aws ec2 create-restore-image-taskをやり直すことができます。

ステップ 3: CardDemo 依存ファイルを S3 にアップロードする

フォルダの [データベース]、[ファイルシステム]、[ユーザーデータ] の内容をコピーします。CardDemo アプリケーションをダウンロードして解凍します。これら 3 つのフォルダは、このドキュメントの [your-s3-bucket] というバケットの 1 つにコピーする必要があります。

ステップ 4: データベースをロードしてアプリケーションを初期化する CardDemo

アプリケーションに必要なデータベーススナップショットを生成するためのコンピューティングリソースとして使用する一時的な Amazon EC2 CardDemoインスタンスを作成します。この EC2 インスタンスは、 CardDemo アプリケーション自体を実行するのではなく、後で使用するデータベーススナップショットを生成します。

まず、load-and-create-ba「-snapshots.yml」という名前の CloudFormation テンプレートを編集します。これは、データベーススナップショットの生成に使用される Amazon EC2 インスタンスを作成するために使用される CloudFormation テンプレートです。

1. EC2 インスタンスに使用する EC2 キーペアを生成して提供します。詳細については、「キーペアの作成」を参照してください。

   例：

   ```
   Ec2KeyPair:
       Description: 'ec2 key pair'
       Default: 'm2-tests-us-west-2'
       Type: String
   ```

2. 前のステップで [データベース] フォルダを配置したフォルダの Amazon S3 パスを指定します。

   ```
   S3DBScriptsPath:
       Description: 'S3 DB scripts folder path'
       Type: String
       Default: 's3://your-s3-bucket/databases'
   ```

3. 前のステップで [ファイルシステム] フォルダを配置したフォルダの Amazon S3 パスを指定します。

   ```
   S3ApplicationFilesPath:
       Description: 'S3 application files folder path'
   ```

```
    Type: String
    Default: 's3://your-s3-bucket/file-system'
```

4. 前のステップで [データベース] フォルダを配置したフォルダの Amazon S3 パスを指定します。

```
S3UserDataPath:
    Description: 'S3 userdata folder path'
    Type: String
    Default: 's3://your-s3-bucket/userdata'
```

5. また、次のステップで使用する結果ファイルを保存する Amazon S3 パスを指定します。

```
S3SaveProducedFilesPath:
    Description: 'S3 path folder to save produced files'
    Type: String
    Default: 's3://your-s3-bucket/post-produced-files'
```

6. 以下のテンプレートを使用して、AMI ID をこのチュートリアルの前半で取得した正しい AMI ID に変更します。

```
  BaaAmiId:
    Description: 'ami id (AL2) for ba anywhere'
    Default: 'ami-0bd41245734fd20d9'
    Type: String
```

   - 必要に応じて、ロードデータベース の実行によって作成される 3 つのスナップショットの名前を変更できます withCloudFormation。これらは作成時に CloudFormation スタックに表示され、このチュートリアルの後半で使用します。データベーススナップショットに使用されている名前を忘れずに書き留めてください。

```
SnapshotPrimary:
    Description: 'Snapshot Name DB BA Primary'
    Type: String
    Default: 'snapshot-primary'

  SnapshotBluesam:
    Description: 'Snapshot Name DB BA Bluesam'
    Type: String
    Default: 'snapshot-bluesam'

  SnapshotJics:
```

```
    Description: 'Snapshot Name DB BA Jics'
    Type: String
    Default: 'snapshot-jics'
```

> **Note**
>
> このドキュメントでは、スナップショットの名前に一貫性があることを前提としています。

7. スタックの作成ボタンとウィザードを使用して、 CloudFormation CLI またはAWSコンソールで を実行します。プロセスの終了時に、選択した名前の後に固有の ID が付いているスナップショットが RDS コンソールに 3 つ表示されます。次のステップではこれらの名前が必要になります。

> **Note**
>
> RDS は、AWS CloudFormation テンプレートで定義されているスナップショット名に接尾辞を追加します。次のステップに進む前に、必ず RDS から完全なスナップショット名を取得してください。

CLI コマンドの例

```
aws cloudformation create-stack --stack-name load-and-create-ba-snapshots --
template-url https://your-apptest-bucket.s3.us-west-2.amazonaws.com/load-and-
create-ba-snapshots.yml --capabilities CAPABILITY_NAMED_IAM
```

Amazon S3 S3SaveProducedFilesPath パスで、データセットが正しく作成されたことを確認することもできます。

ステップ 5: AWS Blu Age ランタイムを起動する CloudFormation

CloudFormation を使用して、 Blu Age アプリケーションで Amazon EC2 CardDemo AWSインスタンスを実行します。CFN の起動時に YAML ファイル`m2-with-ba-using-snapshots-https-authentication.yml`を編集するか、コンソールの値を変更して、 CloudFormation という名前のの変数を置き換える必要があります。

1. を変更AllowedVpcEndpointPrincipalsして、以下のコマンドを使用してAWS、 Blu Age ランタイムにアクセスするための VPC エンドポイントに到達するアカウントを指定します。

```
AllowedVpcEndpointPrincipals:
    Description: 'comma-separated list of IAM users, IAM roles, or AWS accounts'
    Default: 'apptest.amazonaws.com'
    Type: String
```

2. 変数 SnapshotPrimaryDb、 SnapshotBlusamDb、 の値をスナップショット SnapshotJicsDb の名前に変更します。また、前のステップで作成したスナップショット名を RDS から取得します。

```
SnapshotPrimary:
    Description: 'Snapshot DB cluster for DB Primary'
    Type: String
    Default: 'snapshot-primary87d067b0'

  SnapshotBluesam:
    Description: 'Snapshot DB cluster for DB Bluesam'
    Type: String
    Default: 'snapshot-bluesam87d067b0'

  SnapshotJics:
    Description: 'Snapshot DB cluster for DB Jics'
    Type: String
    Default: 'snapshot-jics87d067b0'
```

> **Note**
>
> RDS はスナップショット名に独自の接尾辞を追加します。

3. 次のコマンドを使用して、EC2 インスタンスの Amazon EC2 キー ペアを指定します。

```
Ec2KeyPair:
    Description: 'ec2 key pair'
    Default: 'm2-tests-us-west-2'
    Type: String
```

4. 以下を使用してBaaAmiId、変数 の AMI 登録プロセス中に取得した AMI ID を指定します。

```
BaaAmiId:
```

```
    Description: 'ami id (AL2) for ba anywhere'
    Default: 'ami-0d0fafcc636fd1e6d'
    Type: String
```

5. 次のコマンドを使用して、前のステップで作成したファイルの保存に使用した Amazon S3 フォルダパスを指定します。

```
S3ApplicationFilesPath:
    Description: 'bucket name'
    Type: String
    Default: 's3://your-s3-bucket/post-produced-files'
```

6. 最後に、s3-userdata-folder-path のフォルダパスを指定します。

```
S3UserDataPath:
    Description: 'S3 userdata folder path'
    Type: String
    Default: 's3://your-s3-bucket/userdata'
```

- (オプション) tomcat の HTTPS モードと HTTP Basic 認証を有効にできます。ただし、デフォルト設定でも機能します。

> **Note**
>
> デフォルトでは、HTTPS モードは無効になっており、パラメータ でモード HTTP に設定されていますBacHttpsMode。

例:

```
BacHttpsMode:
    Description: 'http or https for Blue Age Runtime connection mode '
    Default: 'http'
    Type: String
    AllowedValues: [http, https]
```

- (オプション) HTTPS モードを有効にするには、値を HTTPS に変更し、変数 ACM の値を変更して ACM 証明書 ARN CertArnを指定する必要があります。

```
ACMCertArn:
    Type: String
```

```
    Description: 'ACM certificate ARN'
    Default: 'your arn certificate'
```

- (オプション) 基本認証はデフォルトで無効になっており、 パラメータは false にWithBacBasicAuthentication設定されています。有効にするには値を true に設定します。

```
WithBacBasicAuthentication:
    Description: 'false or true for Blue Age Runtime Basic Authentication '
    Default: false
    Type: String
    AllowedValues: [true, false]
```

7. 設定が完了したら、編集した CloudFormation テンプレートを使用してスタックを作成できます。

ステップ 6: AWS Blu Age Amazon EC2 インスタンスをテストする

CloudFormation テンプレートを手動で実行して CardDemo アプリケーションの AWS Blu Age Amazon EC2 インスタンスを作成し、エラーなく起動することを確認します。これは、アプリケーションテスト機能で CloudFormation テンプレートを使用する前に、 CloudFormation テンプレートとすべての前提条件が有効であることを確認するために行われます。その後、アプリケーションテストを使用して、再生中にターゲット AWS Blu Age Amazon EC2 インスタンスを自動的に作成し、初期条件 で比較できます。

1. スタック CloudFormation 作成コマンドを実行して AWS Blu Age Amazon EC2 インスタンスを作成し、前のステップで編集した m2-with-ba-using-snapshots-https-authentication.yml CloudFormation テンプレートを指定します。

```
aws cloudformation create-stack --stack-name load-and-create-ba-snapshots —-
template-url https://apptest-ba-demo.s3.us-west-2.amazonaws.com/m2-with-ba-using-
snapshots-https-authentication.yml --capabilities CAPABILITY_NAMED_IAM --region us-
west-2
```

> **Note**
>
> AWS Blu Age AMI が復元された正しいリージョンを必ず指定してください。

2. 実行中の Amazon EC2 インスタンスをコンソールで見つけて、すべてが正しく動作していることを確認します。Session Manager を使用して接続します。

3. Amazon EC2 インスタンスに接続した後、次のコマンドを使用します。

```
sudo su
cd /m2-anywhere/tomcat.gapwalk/velocity/logs
cat catalina.log
```

4. ログに例外やエラーがないことを確認します。
5. 次に、次のコマンドを使用してアプリケーションが応答していることを確認します。

```
curl http://localhost:8080/gapwalk-application/
```

「Jics アプリケーションは実行中です」というメッセージが表示されます。

ステップ 7: 前のステップが正しく完了したことを検証する

次のステップでは、 AWS Mainframe Modernization アプリケーションテストを使用して、CardDemo アプリケーションによって作成されたデータセットを再生および比較します。これらのステップは、このチュートリアルのこれまでのステップをすべて正常に完了していることが前提となります。次に進む前に、以下を確認します。

1. AWS CloudFormation テンプレートを使用して Amazon EC2 インスタンスで AWS Blu Age を正常に作成しました。
2. Amazon EC2 の AWS Blu Age 上の Tomcat サービスは、例外なく稼働しています。

CardDemo アプリケーションで EC2 インスタンスを実行したら、アプリケーションテストコンソールで次の手順を実行して、バッチデータセットのリプレイと比較を実行します。

ステップ 8: 初期条件を作成する

このステップでは、Blu Age CardDemo アプリケーションを Amazon EC2 AWS にデプロイするために使用した CloudFormation テンプレートを指定して、初期条件を作成します。

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. 左側のナビゲーションペインで、[Application Testing] を選択します。
3. [Application Testing] で [初期条件の作成] を選択します。

4. リソースを指す Amazon S3 パスとスナップショット ID を変更したローカルコピーを使用します。

ステップ 9: テストケースを作成する

このステップでは、Card Demo アプリケーションで作成されたデータセットを比較するためのテストケースを作成します。

1. 新しいテストケースを作成します。これに名前と説明を付けます。
2. JCL 名として `CREASTMT.JCL` を指定します。
3. テストケース定義に以下のデータセットを追加します。

名前	CCSID	RecordFormat	RecordLength
AWS.M2.CARDDEMO.STATEMNT.PS	"037"	FB	80
AWS.M2.CARDDEMO.STATEMNT.HTML	"037"	FB	100

> **Note**
>
> JCL 名とデータセットの詳細は一致している必要があります。

ステップ 10: テストシナリオを作成する

1. 新しいテストシナリオを作成して、名前と説明を入力します。
2. 前のステップで作成したテストケースをテストシナリオに追加します。
3. テストシナリオを作成したら、テストシナリオ概要ページのステップ 1 で作成した初期条件を選択します。

ステップ 11: テストシナリオを記録する

このステップでは、ソース上でテストケースを実行します。これを実行するには:

1. CardDemo アプリケーションのメインフレーム実行から生成されたデータセットをダウンロードして実行します。
2. 解凍したフォルダを Amazon S3 バケットにアップロードします。この Amazon S3 バケットは、他の Application Testing リソースと同じリージョンに存在する必要があります。

> **Note**
>
> 前のテストケースで渡されたデータセット名と一致する名前のファイルが 2 つあるはずです。

3. [テストシナリオの概要] ページで、[記録] ボタンを選択します。
4. [テストシナリオの記録] ページで、ソースメインフレームから取得したデータセットをアップロードした Amazon S3 の場所を指定します。
5. [送信] をクリックして記録処理を開始します。

> **Note**
>
> 記録が完了するのを待ってから、リプレイと比較を行います。

ステップ 12: リプレイと比較

Amazon EC2 環境でターゲット AWS AWS Blu Age でテストシナリオとテストケースを実行します。Application Testing では、リプレイで生成されたデータセットをキャプチャし、メインフレームに記録された参照データセットと比較します。

1. [リプレイと比較] を選択します。 CloudFormation スタックの作成、比較の実行、スタックの削除には約 3 分かかります。

すべてが完了すると、このデモ用に意図的に作成されたいくつかの差異を含む比較結果が得られます。

AWS Mainframe Modernization アプリケーションテストでサポートされているデータセットのコードページ

AWS アプリケーションテストは AWS Mainframe Modernization のプレビューリリースであり、変更される可能性があります。この機能は、本番環境ではなくテストデータとアプリケーションでのみ使用することをお勧めします。

次の表を使用して、データのコード化された文字セット識別子 (CCSID) が AWS アプリケーションテストでサポートされているかどうかを確認します。データにサポートされていない CCSID が使用されている場合は、サポートされている CCSID に変換するか、当社までお問い合わせいただくことをお勧めします。

CCSID	文字セット	説明
37	IBM037、IBM-037、Cp037	ホスト: 米国、カナダ (ESA)、オランダ、ポルトガル、ブラジル、オーストラリア、ニュージーランド
273	IBM273、IBM-273、Cp273	ホスト: オーストリア、ドイツ
277	IBM277、IBM-277、Cp277	ホスト: デンマーク、ノルウェー
278	IBM278、IBM-278、Cp278	ホスト: フィンランド、スウェーデン
280	IBM280、IBM-280、Cp280	ホスト: イタリア
284	IBM284、IBM-284、Cp284	ホスト: スペイン、ラテンアメリカ (スペイン語)
285	IBM285、IBM-285、Cp285	ホスト: 英国
297	IBM297、IBM-297、Cp297	ホスト: フランス
300	IBM-300	日本 DB EBCDIC

CCSID	文字セット	説明
301	IBM-301	PC データ: 日本 DB
437	IBM437、IBM-437、US-ASCII、ASCII、Cp437、US-ASCII	PC データ: PC ベース米国、その他多くの国
500	IBM500、IBM-500、Cp500	ホスト: ベルギー、カナダ (AS/400)、スイス、International Latin-1
720	IBM-720	MSDOS アラビア語
737	IBM-737、X-IBM737	MSDOS ギリシャ語
775	IBM775、IBM-775	MSDOS バルト語
808	IBM-808	PC データ: キリル文字、ロシア (ユーロを含む)
813	ISO-8859-7、ISO8859_7	ISO 8859-7: ギリシャ語
819	ISO-8859-1、ISO8859_1	ISO 8859-1: Latin-1 諸国
833	IBM-833	韓国語 EBCDIC
834	IBM-834、x-IBM834	韓国語 DB EBCDIC
835	IBM-835	中国語 (繁体字) DB EBCD
836	IBM-836	中国語 (簡体字) EBCDIC
837	IBM-837	中国語 (簡体字) EBCDIC
850	IBM850、IBM-850、Cp850	PC データ: Latin-1 諸国
855	IBM855、IBM-855、Cp855	PC データ: キリル文字
856	IBM-856、X-IBM856、Cp856	PC データ: ヘブライ語

CCSID	文字セット	説明
858	IBM00858、IBM-858、Cp858	PC データ: Latin-1 諸国 (ユーロを含む)
859	IBM-859	PC データ: Latin-9
860	IBM860、IBM-860	PC データ: ポルトガル語
861	IBM861、IBM-861	PC データ: アイスランド
862	IBM862、IBM-862、Cp862	PC データ: ヘブライ語 (マイグレーション)
863	IBM863、IBM-863	PC データ: カナダ
865	IBM865、IBM-865、Cp865	PC データ: デンマーク/ノルウェー
866	IBM866、IBM-866、Cp866	PC データ: キリル文字、ロシア
867	IBM-867	PC データ: ヘブライ語 (ユーロを含む)
870	IBM870、IBM-870、Cp870	ホスト: Latin-2 多言語
871	IBM871、IBM-871、Cp871	ホスト: アイスランド
874	X-IBM874	PC データ: タイ
875	IBM-875、X-IBM875、Cp875	ホスト: ギリシャ
897	IBM-897	PC データ: 日本 SB
912	ISO-8859-2、ISO8859_2	ISO 8859-2: Latin-2 多言語
915	ISO-8859-5、ISO8859_5	ISO 8859-5: キリル文字
916	ISO-8859-8、ISO8859_8	ISO 8859-8: ヘブライ文字

CCSID	文字セット	説明
918	IBM918、IBM-918、Cp918	ホスト: ウルドゥー語
920	ISO-8859-9、ISO8859_9	ISO 8859-9: Latin-5 (ECMA-128、トルコ TS-5881)
921	IBM-921、X-IBM921、Cp921	PC データ: ラトビア、リトアニア
922	IBM-922、X-IBM922、Cp922	PC データ: エストニア
923	ISO-8859-15、Cp923、ISO8859_15_FDIS	ISO 8859-15: Latin-9
924	IBM-924	ISO 8859-15: Latin-9
927	IBM-927	PC データ: 中国語 (繁体字)
930	IBM-930、X-IBM930、Cp930	カタカナホスト: 拡張 SBCS。漢字ホスト: DBCS (4370 字のユーザー定義文字を含む)
932	IBM-932	PC データ: 日本 Mix
933	IBM-933、X-IBM933、Cp933	ホスト: 拡張 SBCS。ホスト: DBCS (1880 字のユーザー定義文字と 11172 字の全ハングル文字を含む)
935	IBM-935、X-IBM935、Cp935	ホスト: 拡張 SBCS。ホスト: DBCS (1880 字のユーザー定義文字を含む)。
937	IBM-937、X-IBM937、Cp937	ホスト: 拡張 SBCS。ホスト: DBCS (6204 字のユーザー定義文字を含む)

CCSID	文字セット	説明
939	IBM-939、X-IBM939、Cp939	ラテンホスト: 拡張 SBCS。漢字ホスト: DBCS (4370 字のユーザー定義文字を含む)
942	IBM-942、IBM-942C、X-IBM942、x-IBM942C、Cp942、Cp942C	PC データ:拡張 SBCS。PC データ: DBCS (1880 字のユーザー定義文字を含む)
943	IBM-943、IBM-943C、Shift_JIS、windows-31j、windows-932、X-IBM943、X-IBM943C、Cp943、Cp943C、MS932	PC データ: SBCS。PC データ: オープン環境用の DBCS (1880 字の IBM• ユーザー定義文字を含む)
947	IBM-947	中国語 (繁体字) BIG-5
948	IBM-948、X-IBM948、Cp948	PC データ:拡張 SBCS。PC データ: DBCS (6204 字のユーザー定義文字を含む)
949	IBM-949、IBM-949C、X-IBM949、X-IBM949C、Cp949、Cp949C	IBM KS コード - PC データ: SBCS。IBM KS コード - PC データ: DBCS (1880 字のユーザー定義文字を含む)
950	Big5、IBM-950、X-IBM950、Cp950	PC データ: SBCS (IBM BIG5)。PC データ: DBCS (13493 字の CNS 文字、IBM が選択した 566 字、および 6204 字のユーザー定義文字を含む)
951	IBM-951	PC データ:IBM KS
954	EUC-JP、IBM-954、IBM-954C	G0: JIS X201 ローマン。G1: JIS X208-1990。G1: JIS X201 カタカナ。G1: JIS X212

CCSID	文字セット	説明
964	EUC-TW、IBM-964、X-IBM964、Cp964	G0: ASCII。G1: CNS 11643 プレーン 1。G1: CNS 11643 プレーン 2。
970	EUC-KR、X-IBM970、Cp970	G0: ASCII。G1: KSC X5601-1989 (1880 字のユーザー定義文字を含む)
971	IBM-971	韓国語の EUC
1006	IBM-1006、X-IBM1006、Cp1006	ISO-8: ウルドゥー語
1025	IBM-1025、X-IBM1025、Cp1025	ホスト: キリル文字 (多国語)
1026	IBM1026、IBM-1026、Cp1026	ホスト: Latin-5 (トルコ)
1027	IBM-1027	日本 Latin EBCD
1041	IBM-1041	PC データ: 日本
1043	IBM-1043	PC データ: 中国語 (繁体字)
1046	IBM-1046、IBM-1046S、X-IBM1046	アラビア語 - PC
1047	IBM1047、IBM-1047	ホスト: Latin-1
1051	hp-roman8	HP エミュレーション
1088	IBM-1088	PC データ:韓国 KS
1089	ISO-8859-6、ISO8859_6	ISO 8859-6: アラビア語
1097	IBM-1097、X-IBM1097、Cp1097	ホスト: ペルシア語

CCSID	文字セット	説明
1098	IBM-1098、X-IBM1098、Cp1098	PC データ: ペルシア語
1112	IBM-1112、X-IBM1112、Cp1112	ホスト: ラトビア、リトアニア
1114	IBM-1114	PC データ: 中国語 (繁体字) SB
1115	IBM-1115	PC データ: 中国語 (簡体字) GB
1122	IBM-1122、X-IBM1122、Cp1122	ホスト: エストニア
1123	IBM-1123、X-IBM1123、Cp1123	ホスト: キリル文字ウクライナ
1124	IBM-1124、X-IBM1124、Cp1124	8 ビット: キリル文字、ベラルーシ
1140	IBM01140、IBM-1140、Cp1140	ホスト: 米国、カナダ (ESA)、オランダ、ポルトガル、ブラジル、オーストラリア、ニュージーランド (ユーロを含む)
1141	IBM01141、IBM-1141、Cp1141	ホスト: オーストリア、ドイツ (ユーロを含む)
1142	IBM01142、IBM-1142、Cp1142	ホスト: デンマーク、ノルウェー (ユーロを含む)
1143	IBM01143、IBM-1143、Cp1143	ホスト: フィンランド、スウェーデン (ユーロを含む)

CCSID	文字セット	説明
1144	IBM01144、IBM-1144、Cp1144	ホスト: イタリア (ユーロを含む)
1145	IBM01145、IBM-1145、Cp1145	ホスト: スペイン、ラテンアメリカ (スペイン語) (ユーロを含む)
1146	IBM01146、IBM-1146、Cp1146	ホスト: 英国 (ユーロを含む)
1147	IBM01147、IBM-1147、Cp1147	ホスト: フランス (ユーロを含む)
1148	IBM01148、IBM-1148、Cp1148	ホスト: ベルギー、カナダ (AS/400)、スイス、International Latin-1 (ユーロを含む)
1149	IBM01149、IBM-1149、Cp1149	ホスト: アイスランド (ユーロを含む)
1200	UTF-16BE	文字セット 65535 を含む Unicode。バイトオーダーマーク (BOM) がない場合は、UTF-16 BE (ビッグエンディアン) と解釈します。
1202	UTF-16LE	UTF-16 LE (IBM PUA を使用)
1204	UTF-16	UTF-16 (IBM PUA を使用)
1208	UTF-8、UTF-8J、UTF8	文字セット 65535 を含む Unicode。UTF-8。
1232	UTF-32BE	UTF-32 BE (IBM PUA を使用)
1234	UTF-32LE	UTF-32 LE (IBM PUA を使用)

CCSID	文字セット	説明
1236	UTF-32	UTF-32 (IBM PUA を使用)
1351	IBM-1351	日本オープン
1362	IBM-1362	韓国語 MS-WIN
1363	IBM-1363、IBM-1363C、Windows 949、MS949	PC データ: MS Windows 韓国語 SBCS。PC データ:MS Windows 韓国語 DBCS (11172 字の全ハングルを含む)
1364	IBM-1364	ホスト: 拡張 SBCS。ホスト: DBCS (1880 字のユーザー定義文字と 11172 字の全ハングル文字を含む)
1370	IBM-1370	PC データ: 拡張 SBCS (ユーロを含む)。PC データ: DBCS (6204 字のユーザー定義文字を含む) (ユーロを含む)
1371	IBM-1371	ホスト: 拡張 SBCS (ユーロを含む)。ホスト: DBCS (6204 字のユーザー定義文字を含む) (ユーロを含む)
1375	Big5-HKSCS	HKSCS 用拡張 Mix Big-5
1380	IBM-1380	PC データ: 中国語 (簡体字) GB
1381	IBM-1381、x-IBM1381、Cp1381	PC データ: 拡張 SBCS (IBM GB)。PC データ: DBCS (IBM GB) (IBM が選択した 31 字、1880 字のユーザー定義文字を含む)

CCSID	文字セット	説明
1382	IBM-1382	中国語 (簡体字) EUC
1383	EUC-CN、GB2312、IBM-1383、X-IBM1383、Cp1383	G0: ASCII。G1: GB 2312-80 セット
1385	IBM-1385	PC データ: 中国語 (簡体字) GBK
1386	GBK、IBM-1386、windows-936、MS936	PC データ: 中国語 (簡体字) GBK および中国語 (繁体字) IBM BIG-5。PC データ: 中国語 (簡体字) GBK
1388	IBM-1388	ホスト: 拡張 SBCS。ホスト: DBCS (1880 字のユーザー定義文字を含む)
1390	IBM-1390	カタカナホスト: 拡張 SBCS (ユーロを含む)。漢字ホスト: DBCS (6205 字のユーザー定義文字を含む)
1399	IBM-1399	ホスト: 拡張 SBCS (ユーロを含む)。ホスト: DBCS (4370 字のユーザー定義文字を含む) (ユーロを含む)
5050	JIS0201、JIS0208、JIS0212、JIS0201、JIS0208、JIS0212	G0: JIS X201 ローマン。G1: JIS X208-1990。G1: JIS X201 カタカナ。G1: JIS X212
5054	ISO-2022-JP	日本語 TCP
5346	Windows-1250、Cp1250	MS Windows: Latin-2、バージョン 2 (ユーロを含む)

CCSID	文字セット	説明
5347	Windows-1251、Cp1251	MS Windows: キリル文字、バージョン 2 (ユーロを含む)
5348	Windows-1252、Cp1252	MS Windows: Latin-1、バージョン 2 (ユーロを含む)
5349	Windows-1253、Cp1253	MS Windows: ギリシャ、バージョン 2 (ユーロを含む)
5350	Windows-1254、Cp1254	MS Windows: トルコ、バージョン 2 (ユーロを含む)
5351	Windows-1255、Cp1255	MS Windows: ヘブライ語、バージョン 2 (ユーロを含む)
5352	windows-1256、windows-1256S、Cp1256	MS Windows: アラビア語、バージョン 2 (ユーロを含む)
5353	Windows-1257、Cp1257	MS Windows: バルト海沿岸語、バージョン 2 (ユーロを含む)
5354	Windows-1258、Cp1258	MS Windows: ベトナム語、バージョン 2 (ユーロを含む)
5488	GB18030	GB18030、1 バイトデータ GB18030、2 バイトデータ GB18030、4 バイトデータ
9030	IBM-838、Cp838	ホスト: タイ語拡張 SBCS
9066	IBM-874、Cp874	PC データ: タイ語拡張 SBCS
9400	CESU-8	CESU-8 (IBM PUA を使用)
25546	ISO-2022-KR	韓国語 TCP
33722	IBM-33722、IBM-33722C	IBMeucJP

AWS Mainframe Modernization でのファイル転送

AWS Mainframe Modernization File Transfer を使用すると、メインフレームのデータセットを Amazon S3 に転送および変換して、メインフレームのモダナイゼーション、移行、拡張のユースケースに対応できます。

トピック

AWS Mainframe Modernization File Transfer とは

AWS Mainframe Modernization File Transfer を使用すると、フルマネージドサービスでデータセットとファイルを転送および変換して、 AWS Mainframe Modernization サービスと Amazon S3 へのモダナイゼーション、移行、および拡張のユースケースを加速および簡素化できます。

トピック

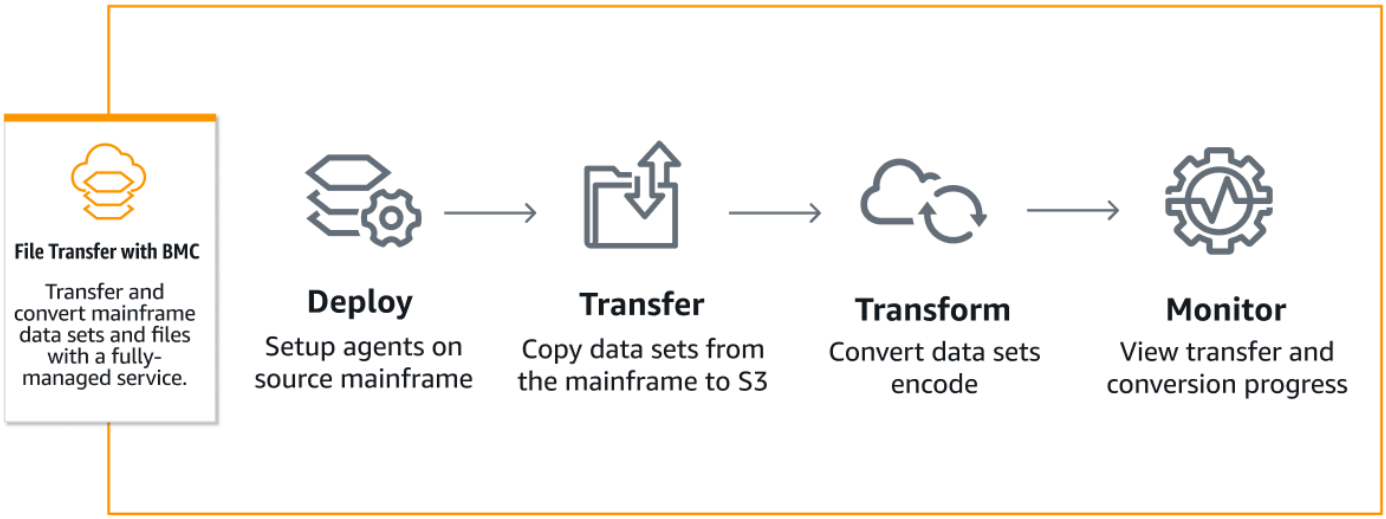
AWS Mainframe Modernization File Transfer の利点

AWS Mainframe Modernization File Transfer は、メインフレームから Amazon S3 へのデータセットの転送に役立ちます。次のような利点があります。

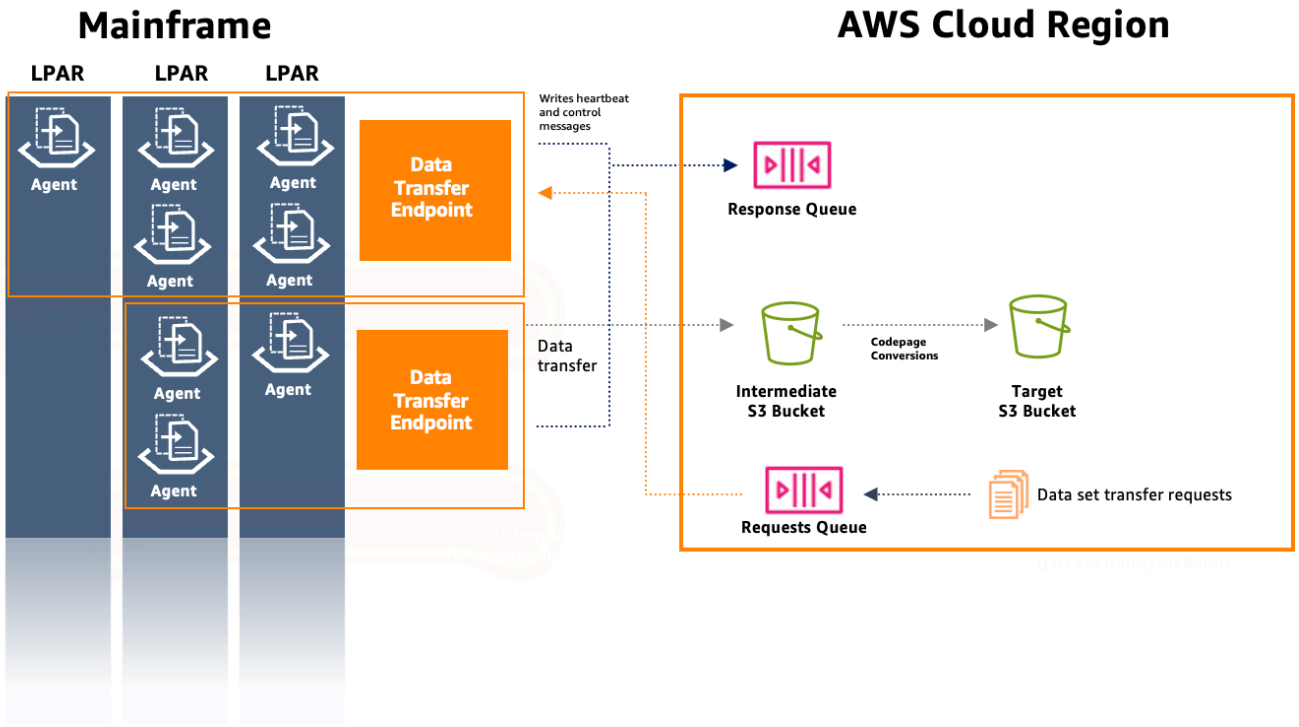
- ソースメインフレームのデータセットとアーティファクトの検出
- 自動転送とデータセット変換
- AWS へのデータセットの転送を高速化するスケーラビリティ、効率性、スピード

AWS Mainframe Modernization File Transfer の仕組み

次の図は、AWS Mainframe Modernization File Transfer の仕組みを概念レベルで示しています。



次の図は、AWS Mainframe Modernization File Transfer 機能のアーキテクチャ概要です。



ファイル転送エージェントのインストール

このガイドに従って step-by-step 、ソースメインフレームにエージェントをインストールするための前提条件を完了し、エージェントを設定します。

トピック

ステップ 1: ISPF にログインする

ISPF (Interactive System Productivity Facility) セッションにログインします。これには通常 3270 ターミナルエミュレータを使用します。

ステップ 2: z/FS にデータセットを割り当てる

ISPF のデータセットユーティリティを使用して、z/FS に新しいデータセットを割り当てます。通常、これはステップ 1 のデータセットリストユーティリティで行います。

1. オプション 3.4 ([Utilities] --> [Dataset]) に進みます。
2. キー「C」を押して、新しいデータセットを作成します。
3. データセットの名前 (例:「yourhlq.M2AGENT.ZFS」) を入力します。
4. プライマリサイズを 1000 シリンダー、セカンダリサイズを 200 にして、データセットタイプを「Large format」として指定します。
5. データセット構成 (DSORG) を PS に、レコードフォーマット (RECFM) を「U」(未定義) に設定します。
6. 作成プロセスを完了します。

ステップ 3: データセットを z/FS としてフォーマットする

データセットを作成したら、z/FS ファイルシステムとしてフォーマットします。

これを行う方法の 1 つは、以下のジョブ制御言語 (JCL) を使用することです。

```
//FORMAT  EXEC PGM=IOEAGFMT,PARM='AGGRNAME(yourhlq.M2AGENT.ZFS),FORMAT,AGGRSIZE(1200)'
//SYSPRINT DD  SYSOUT=A
```

このジョブを送信し、正常に完了したかどうかを確認してください。

ステップ 4: ファイルシステムを z/OS に定義する

DEF FILESYSTEM コマンドまたはバッチジョブを使用して z/FS を z/OS に定義します。以下のコマンドを使用します。

```
DEF FILESYSTEM('yourhlq.M2AGENT.ZFS') TYPE(ZFS) MODE(R/W) MOUNTPOINT('/usr/lpp/aws/m2-agent')
```

> **Note**
>
> このステップではシステムレベルの権限が必要です。

ステップ 5: ファイルシステムをマウントする

ファイルシステムをマウントするには、MOUNT コマンドを使用します。ファイルシステムは ISPF のコマンドラインまたはバッチでマウントできます。

例:

```
MOUNT FILESYSTEM('yourhlq.M2AGENT.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/usr/lpp/aws/
m2-agent')
```

ステップ 6: マウントを確認する

D OMVS,A コマンドを使用するか UNIX System Service (USS) 内でチェックして、ファイルシステムが正しくマウントされていることを確認します。

ステップ 7: OMV を入力する

以下のコマンドを使用して、OMV を入力します。

```
TSO OMVS
```

ステップ 8: エージェントのインストールディレクトリの環境変数を設定する

エージェントのインストールディレクトリ環境を設定するには、以下のコマンドを使用します。

```
export AGENT_DIR=/usr/lpp/aws/m2-agent
```

ステップ 9: 作業ディレクトリの環境変数を設定する

以下のコマンドを使用して、作業ディレクトリの環境変数を設定します。

```
export WORK_DIR=$AGENT_DIR/tmp
```

ステップ 10: 作業ディレクトリを作成する

以下のコマンドを使用して、作業ディレクトリ環境を設定します。

```
mkdir -p $WORK_DIR
```

ステップ 11: AWS Mainframe Modernization tar パッケージを z/OS の作業ディレクトリにコピーする

FTP やその他の転送方法を使用する場合は、tar ファイルがバイナリモードで転送されることを確認してください。

ステップ 12: ルートユーザーを引き受ける

以下のコマンドを使用して、ルートユーザーを引き受けます。

```
su
```

以下のステップに従って、エージェントのインストールを完了します。

> **Note**
>
> 以下の手順に進む前に、ルートユーザーを引き受ける必要があります。

1. 以下のコマンドを使用して、m2-agent バージョンの環境変数を現在インストールされているバージョンに設定します。

   ```
   export M2_AGENT_VERSION=1.0.0
   ```

2. 以下のコマンドを使用して、エージェント tar パッケージを抽出します。

   ```
   tar -xpf m2-agent-package-$M2_AGENT_VERSION.tar -C $AGENT_DIR
   ```

3. 以下のコマンドを使用して、現在のエージェントインストールディレクトリへの `current-version` シンボリックリンクを作成します。

   ```
   ln -s $AGENT_DIR/m2-agent-v$M2_AGENT_VERSION $AGENT_DIR/current-version
   ```

4. ファイル転送エージェントデータセットを作成するために、CPY#PDS を更新して送信します。

   > **Note**
   >
   > JCL は SYS2.AWS.M2 HLQ を使用します。

ファイル転送エージェントを作成するには、パラメータ行 000006-000012 を設定します。また、3 つのシンボリック変数 HLQ、VOLSER、AGNTPATH を JCL で後で使用できるように更新します。

```
oedit $AGENT_DIR/current-version/installation/CPY#PDS
submit $AGENT_DIR/current-version/installation/CPY#PDS
```

> **Note**
>
> この JCL は、メインフレームでのエージェントインストールの特定の要素の設定向けに調整されています。必要なデータセットを割り当ててから、UNIX ファイルシステムから特定のファイルをそれらのデータセットにコピーします。

アクセス許可と STC を設定する

1. 指示に従って SYS2.AWS.M2.SAMPLIB(SEC#RACF) (RACF アクセス許可の設定用) または SYS2.AWS.M2.SAMPLIB(SEC#TSS) (TSS アクセス許可の設定用) のいずれかを更新して送信します。これらのメンバーは前の CPY#PDS ステップで作成されたものです。
2. デフォルトのファイル転送エージェントディレクトリパス (/usr/lpp/aws/m2-agent) が変更された場合は、SYS2.AWS.M2.SAMPLIB(M2AGENT) STC JCL の PWD エクスポートを更新します。
3. SYS2.AWS.M2.SAMPLIB(M2AGENT) JCL を更新して SYS1.PROCLIB にコピーします。
4. 以下のコマンドを使用して、APF リストに SYS2.AWS.M2.LOADLIB を追加します。

```
SETPROG APF ADD DSNAME(SYS2.AWS.M2.LOADLIB) SMS
```

5. エージェントの logs と diag フォルダのグループおよび所有者をエージェントのユーザー/グループ (M2USER/M2GROUP) に設定します。以下のコマンドを使用します。

```
chown -R M2USER:M2GROUP $AGENT_DIR/current-version/logs
chown -R M2USER:M2GROUP $AGENT_DIR/current-version/diag
```

長期アクセス認証情報を使用して IAM ユーザーを作成する

レスポンスキューとリクエストキューを使用し、データセットを Amazon S3 バケットに保存するには、IAM ユーザーが必要とするメインフレームエージェントを作成する必要があります。

このユーザーを作成する場合は以下を行います。

1. [アクセス許可] オプションで直接 [ポリシーをアタッチ] を選択します。
2. ユーザーを作成したら、[セキュリティ認証情報] タブを開き、アクセスキーを作成します。IAM アクセスキーの作成に関する詳細については、「IAM ユーザーのアクセスキーの管理」を参照してください。
3. [アクセスキー] セクションで、ユースケースの入力を求められたら [その他] を選択します。

> **Note**
>
> [完了] をクリックする前に、アクセスキー作成ウィザードの最後のページに表示されるアクセスキーとシークレットアクセスキーを保存します。これらのキーはメインフレームエージェントの設定に使用されます。

> **Note**
>
> IAM ロールとの信頼関係を設定するために使用した IAM ユーザー ARN を保存します。

エージェントが引き受ける IAM ロールを作成する

[信頼されたエンティティタイプ] に [カスタム信頼ポリシー] を使用して新しい IAM ロールを作成します。ポリシーは以下のテンプレートを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DataTransferEndpointAgentSqsReceive",
            "Effect": "Allow",
            "Action": [
                "sqs:DeleteMessage",
                "sqs:ReceiveMessage"
```

```
            ],
            "Resource": "<data-transfer-endpoint-request-queue-arn>"
        },
        {
            "Sid": "DataTransferEndpointS3",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "<data-transfer-endpoint-intermediate-bucket-arn>/*"
        },
        {
            "Sid": "DataTransferEndpointAgentSqsSend",
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "<data-transfer-endpoint-response-queue-arn>"
        },
        {
            "Sid": "DataTransferEndpointAgentKmsDecrypt",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "<kms-key-id>"
        }
    ]
}
```

コードの説明は以下のとおりです。

- `request-queue-arn` および `response-queue-arn` は、データ転送エンドポイントの初期化中に Amazon SQS キューによって作成されたリクエストの ARN です。
- `transfer-bucket-arn` は、先ほど作成された転送バケットの ARN です。

> **Note**
>
> これらの値はすべて AWS コンソールを使用して検索できます。

> **Note**
>
> ロール名を保存します。この名前は後でメインフレームエージェントを設定するときに使用します。

エージェントの設定

ファイル転送エージェントを設定するには

1. $AGENT_DIR/current-version/config に移動します。
2. 以下のコマンドを使用して、エージェントの設定ファイル appication.properties を編集し、環境設定を追加します。

```
oedit $AGENT_DIR/current-version/config/application.properties
```

例:

```
agent.environments[0].account-id=<AWS_ACCOUNT_ID>
agent.environments[0].agent-role-name=<AWS_IAM_ROLE_NAME>
agent.environments[0].access-key-id=<AWS_IAM_ROLE_ACCESS_KEY>
agent.environments[0].secret-access-id=<AWS_IAM_ROLE_SECRET_KEY>
agent.environments[0].bucket-name=<AWS_S3_BUCKET_NAME>
agent.environments[0].environment-name=<AWS_REGION>
agent.environments[0].region=<AWS_REGION>
```

コードの説明は以下のとおりです。

- AWS_ACCOUNT_ID はお客様のアカウントの ID です。
- AWS_IAM_ROLE_NAME は、the section called “エージェントが引き受ける IAM ロールを作成する” で作成された IAM ロールの名前です。
- AWS_IAM_ROLE_ACCESS_KEY は、the section called “長期アクセス認証情報を使用して IAM ユーザーを作成する” で作成された IAM ユーザーのアクセスキーです。
- AWS_IAM_ROLE_SECRET_KEY は、the section called “長期アクセス認証情報を使用して IAM ユーザーを作成する” で作成された IAM ユーザーのアクセスシークレットキーです。
- AWS_S3_BUCKET_NAME は、データ転送エンドポイントで作成された転送バケットの名前です。
- AWS_REGION は、ファイル転送エージェントを設定するリージョンです。

> **Note**
>
> 括弧 — [0] — 内のインデックスが 1 つずつ増えていれば、このようなセクションが複数あっても構いません。

変更を有効にするには、エージェントを再起動する必要があります。

要件

1. パラメータを追加または削除したら、エージェントを停止して起動する必要があります。CLI で以下のコマンドを使用してファイル転送エージェントを起動します。

   ```
   /S M2AGENT
   ```

   M2 エージェントを停止するには、CLI で以下のコマンドを実行します。

   ```
   /P M2AGENT
   ```

2. 複数の環境を定義 AWS することで、File Transfer エージェントを の複数のリージョンとアカウントに転送できます。

   ```
   #Region 1
   agent.environments[0].account-id=AWS_ACCOUNT_ID
   agent.environments[0].agent-role-name=AWS_IAM_ROLE_NAME
   agent.environments[0].access-key-id=AWS_IAM_ROLE_ACCESS_KEY
   agent.environments[0].secret-access-id=AWS_IAM_ROLE_SECRET_KEY
   agent.environments[0].bucket-name=AWS_S3_BUCKET_NAME
   agent.environments[0].environment-name=AWS_REGION
   agent.environments[0].region=AWS_REGION

   #Region 2
   agent.environments[1].account-id=AWS_ACCOUNT_ID
   agent.environments[1].agent-role-name=AWS_IAM_ROLE_NAME
   agent.environments[1].access-key-id=AWS_IAM_ROLE_ACCESS_KEY
   agent.environments[1].secret-access-id=AWS_IAM_ROLE_SECRET_KEY
   agent.environments[1].bucket-name=AWS_S3_BUCKET_NAME
   agent.environments[1].environment-name=AWS_REGION
   agent.environments[1].region=AWS_REGION
   ```

データ転送エンドポイント

データ転送エンドポイントは、ソースメインフレーム上のエージェントの高可用性、スケーラビリティ、および効率的な管理を可能にします。個々のエージェントはメインフレームの LPAR にインストールされ、1 つのデータ転送エンドポイントにグループ化することができます。データセットの転送がリクエストされると、データ転送エンドポイントの 1 つのエージェントが転送を処理します。データ転送を開始するには、データ転送エンドポイントの少なくとも 1 つのエージェントがオンラインになっている必要があります。

この手順は、「AWS Mainframe Modernization のセットアップ」および「ソースメインフレーム上でファイル転送エージェントを設定する」の手順を完了していることを前提としています。

データ転送エンドポイントを作成する

ファイル転送用のデータ転送エンドポイントを作成するには、 AWS Mainframe Modernization コンソールで以下の手順を実行する必要があります。

データ転送エンドポイントを作成するには

1. https://console.aws.amazon.com/m2/ で AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクターで、メインフレームから Amazon S3 バケットにファイルを転送するリージョンを選択します。
3. [データ転送エンドポイント] ページの [ファイル転送] で、[データ転送エンドポイントの作成] を選択します。
4. [データ転送エンドポイントの前提条件] ページで、すべての指示を読み、これらの手順を完了していることを確認します。確認したら、[次へ] を選択します。
5. [データ転送エンドポイントの設定] ページで、データ転送エンドポイントの基本情報を追加します。

    1. 基本情報セクションに、データ転送エンドポイントの名前、説明、KMS キーを入力します。KMS キーの詳細については、「キーの作成」を参照してください。

    > **Note**
    >
    > データ転送エンドポイント名は、ソースメインフレームでファイル転送エージェントを設定したときに定義した名前と一致する必要があります。

> **Note**
>
> AWS Mainframe Modernization サービスが暗号化/復号化にこれらのキーを読み取って使用できるように、KMS に次のリソースベースのポリシーを追加する必要があります。

```
{
  "Sid" : "Enable AWS M2 Permissions",
  "Effect" : "Allow",
  "Principal" : {
      "Service" : "m2.amazonaws.com"
            },
   "Action" : [
       "kms:Encrypt",
       "kms:Decrypt"
              ],
   "Resource" : "*"
}
```

2. 中間データ用の S3 ロケーションを指定します。これは、メインフレームから転送されたデータセットが保存される S3 の中間ロケーションです。

> **Note**
>
> 転送タスク用の新しい Amazon S3 バケットを作成することをお勧めします。詳細については、「バケットの作成」を参照してください。[S3 を参照] オプションをクリックして、既存の Amazon S3 バケットを参照することもできます。

3. 必須フィールドに入力したら、[次へ] をクリックします。

6. [データ転送エンドポイントの確認と作成] ページで、前提条件を完了したかどうかを確認し、基本情報を確認します。確認したら、[作成して接続] をクリックします。

7. エージェント接続が確立されると、[接続の確認] ページに、すべてのエージェントが ID とハートビートとともに表示されます。接続が確立されると、「データ転送エンドポイントは正常に作成されました」というメッセージが表示されます。

> **Note**
>
> 「エージェント接続を確立できませんでした」というメッセージが表示された場合は、ステップ 4 に戻って、必要な前提条件をすべて完了したことを確認してください。

8. [終了] を選択します。

[データ転送エンドポイントの概要] ページにリダイレクトされ、すべてのデータ転送エンドポイントのリストが表示されます。また、使用可能なデータ転送エンドポイントや障害が発生したデータ転送エンドポイントも確認できます。

また、データ転送エンドポイントを名前で検索したり、使用可能な各エージェントの追加情報にアクセスしたりすることもできます。

転送タスク

転送タスクは、データセットのソースエンコードとターゲットエンコードを定義するために使用されます。ソースエンコーディングはソースデータセットの形式であり、ターゲットエンコーディングはこれらのデータセットがターゲット Amazon S3 バケットに格納される形式です。これらのターゲットバケットは転送タスクによって定義されます。

この手順は、「AWS Mainframe Modernization のセットアップ」のステップと「the section called “データ転送エンドポイント”」の設定が完了していることを前提としています。

トピック

- 転送タスクを作成する
- 転送タスクを表示する

転送タスクを作成する

ファイル転送の転送タスクを作成するには、 AWS Mainframe Modernization コンソールで以下の手順を実行する必要があります。

転送タスクを作成するには

> **Note**
>
> 新しい転送タスクを作成するには、データ転送エンドポイントが 1 つ以上あることが必要です。

1. https://console.aws.amazon.com/m2/ で AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクターで、メインフレームから Amazon S3 バケットにファイルを転送するリージョンを選択します。
3. 「転送タスク」ページの「ファイル転送」で、データ転送エンドポイントを選択して転送タスクを作成します。
4. データ転送エンドポイントに転送タスクがない場合は、[転送タスクの作成] をクリックして新しいタスクを作成できます。
5. 転送タスクの作成ページで、転送タスクのプロパティを設定します。

   - このページでは、転送タスク名、説明、シークレットキー、データセットの検索条件など、転送タスクの基本情報を入力します。

     > **Note**
     >
     > - データ転送エンドポイントで定義された KMS キーを使用してシークレットを暗号化します。シークレットには、`userId` および `password` キーを使用してメインフレーム上のデータセットにアクセスするために必要なメインフレーム認証情報も含まれている必要があります。詳細については、「AWS Secrets Manager シークレット」を参照してください。
     > - AWS Mainframe Modernization サービスがシークレットキーにアクセスしてデータ転送タスクを実行できるように、次のリソースベースのポリシーを使用してシークレットキーを設定する必要があります。
     >
     > ```
     > {
     >   "Version" : "2012-10-17",
     >   "Statement" : [ {
     >     "Effect" : "Allow",
     >     "Principal" : {
     >         "Service" : "m2.amazonaws.com"
     > ```

```
    },
    "Action" : [ "secretsmanager:GetSecretValue",
                 "secretsmanager:DescribeSecret" ],
    "Resource" : "*"
  } ]
}
```

> **Note**
>
> 現在サポートされているデータセットの最大サイズは 90 GiB です。

- ファイルのターゲット Amazon S3 バケットの場所を入力または参照します。
- デフォルトのデータ転送エンドポイントが選択されます。使用可能なエンドポイントからエンドポイントを選択することもできます。

6. [次へ] を選択します。
7. データセットの選択ページで、選択した各データセットのソースエンコーディングとターゲットエンコーディングを手動で更新する必要があります。ソースエンコーディングはソースデータセット形式であり、ターゲットエンコーディングはターゲットデータセット形式であり、データセットの変換に使用されます。
8. ソースとターゲットのエンコーディングを確認したら、[次へ] を選択します。
9. [確認と作成] ページで、転送タスクの情報を確認または編集できます。
10. [転送タスクの作成] をクリックします。

「転送タスクが正常に作成されました」というメッセージが表示されます。

転送タスクを表示する

ファイル転送の転送タスクを表示するには、 AWS Mainframe Modernization コンソールで以下の手順を実行する必要があります。

転送タスクを表示するには

1. https://console.aws.amazon.com/m2/ で AWS Mainframe Modernization コンソールを開きます。
2. AWS リージョン セレクターで、メインフレームから Amazon S3 バケットにファイルを転送するリージョンを選択します。

3. 「転送タスク」ページの「ファイル転送」で、データ転送エンドポイントを選択して転送タスクを表示します。
4. 転送タスクが既に存在するエンドポイントの場合、これらのタスクは [転送タスク] セクションに表示されます。このリストで転送タスクを選択して、詳細を表示できます。

チュートリアル: AWS Mainframe Modernization ファイル転送の開始方法

AWS Mainframe Modernization File Transfer を使用すると、メインフレームのモダナイゼーション、移行、拡張のユースケースに合わせてメインフレームデータセットを転送および変換できます。

このチュートリアルの手順に従って、AWS Mainframe Modernization ファイル転送のしくみを理解しましょう。

概要

ファイル転送は次のもので構成されます。

1. ソースメインフレームにインストールするエージェント。
2. AWS Mainframe Modernization 管理サービスコンソールからデータセットの検出、転送、変換機能に直接アクセスできます。

ユーザーは、データセットをメインフレームから、Amazon S3 バケットに転送できます。

トピック

- ステップ 1: エージェントバイナリ tar パッケージを からメインフレームの論理パーティションに転送 AWS する
- ステップ 2: ソースメインフレームでファイル転送エージェントを設定する
- ステップ 3: データ転送エンドポイントを作成する
- ステップ 4: 転送タスクを作成する
- ステップ 5: 転送タスクの進行状況を表示する

ステップ 1: エージェントバイナリ tar パッケージを からメインフレームの論理パーティションに転送 AWS する

M2-agent tar リンクから tar ファイルをダウンロードします。

ステップ 2: ソースメインフレームでファイル転送エージェントを設定する

このステップでは、ソースメインフレームで AWS Mainframe Modernization ファイル転送エージェントを設定して起動します。エージェントは、ファイル転送サービス機能とソースメインフレーム間の通信を円滑にするために必要です。メインフレームごとに少なくとも 1 つのエージェントが必要です。可用性を高め、スケーラビリティを向上させるために複数のエージェントを起動できます。

the section called “ファイル転送エージェントのインストール” ガイドの指示に従って、メインフレームへのファイル転送エージェントのインストールを完了します。

ステップ 3: データ転送エンドポイントを作成する

the section called “データ転送エンドポイント” ページの手順に従って、新しいデータ転送エンドポイントを作成します。

ステップ 4: 転送タスクを作成する

the section called “転送タスク” ページの手順に従って、転送タスクを作成して管理します。

ステップ 5: 転送タスクの進行状況を表示する

Mainframe Modernization AWS コンソールで転送タスクの進行状況を表示できます。詳細については、「the section called “転送タスクを表示する”」セクションを参照してください。

AWS Mainframe Modernization におけるセキュリティ

AWS では、クラウドのセキュリティが最優先事項です。AWS の顧客は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャから利点を得られます。

セキュリティは、AWS とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS のサービスを実行するインフラストラクチャを保護する責任を負います。また、AWS は、使用するサービスを安全に提供します。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。AWS Mainframe Modernization に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」「」を参照してください。
- クラウド内のセキュリティ – お客様の責任範囲は、ご使用の AWS のサービスに応じて異なります。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、AWS Mainframe Modernization を使用する際の責任共有モデルの適用について理解するのに役立ちます。ここでは、セキュリティとコンプライアンスの目的を満たすように AWS Mainframe Modernization を設定する方法について説明します。また、AWS Mainframe Modernization リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

AWS Mainframe Modernization は、IAM で保護された独自のリソース (アプリケーション、環境、デプロイなど) を提供します。これらは AWS Mainframe Modernization の管理リソースであり、IAM ポリシーによってすべてのアクションが許可されている必要があります。

リプラットフォームのための AWS Mainframe Modernization も IAM によって保護されています。IAM は、元のメインフレーム環境から派生した定義済みリソースに対する特定のアクションに対するアクセス許可を、標準の IAM ポリシーを通じてプリンシパルにも付与または拒否します。AWS Mainframe Modernization リプラットフォームランタイムは、保護されたリソースに対してアプリケーションがそのようなアクションを試みると IAM 承認サービスを呼び出します。IAM は標準の IAM ポリシー評価メカニズムに基づいて許可または拒否を返します。

目次

AWS Mainframe Modernization におけるデータ保護

AWS 責任共有モデル、 AWS Mainframe Modernization のデータ保護に適用されます。このモデルで説明したように、 AWS は、すべての を実行するグローバルインフラストラクチャを保護する責任を担います AWS クラウド。このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任はユーザーにあります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、「AWS セキュリティブログ」に投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護の目的で、 認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。こうすると、それぞれのジョブを遂行するために必要なアクセス許可のみを各ユーザーに付与できます。また、以下の方法でデータを保護することをお勧めします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2、できれば TLS 1.3 が必要です。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションを、 内のすべてのデフォルトのセキュリティコントロールとともに使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの機密情報やセンシティブ情報は、タグや名前フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これは、コンソール、API、 AWS CLIまたは SDK で AWS Mainframe Modernization または他の AWS のサービス を使用する場合も同様です。 AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

AWS Mainframe Modernization が収集するデータ

AWS Mainframe Modernization は、ユーザーからいくつかのタイプのデータを収集します。

- `Application configuration`: これは、アプリケーションを設定するために作成する JSON ファイルです。 AWS Mainframe Modernization が提供するさまざまなオプションの選択肢が含まれています。ファイルには、アプリケーションアーティファクトが保存されている Amazon Simple Storage Service パスや、データベース認証情報が保存され AWS Secrets Manager ている Amazon リソースネーム (ARN) などの依存 AWS リソースに関する情報も含まれています。
- `Application executable (binary)`: これはコンパイルし、 AWS Mainframe Modernization にデプロイする予定のバイナリです。
- `Application JCL or scripts`: このソースコードは、アプリケーションに代わってバッチジョブやその他の処理を管理します。
- `User application data`: データセットをインポートすると、 AWS Mainframe Modernization はそれらをリレーショナルデータベースに保存して、アプリケーションがデータセットにアクセスできるようにします。
- `Application source code`: Amazon AppStream 2.0 を通じて、 AWS Mainframe Modernization はコードを記述してコンパイルするための開発環境を提供します。

AWS Mainframe Modernization は、このデータをネイティブに に保存します AWS。収集したデータは、 AWS Mainframe Modernization が管理する Amazon S3 バケットに保存されます。アプリケーションをデプロイすると、 AWS Mainframe Modernization は Amazon Elastic Block Store-Backed Amazon Elastic Compute Cloud インスタンスにデータをダウンロードします。クリーンアップがトリガーされると、データは Amazon EBS ボリュームと Amazon S3 から削除されます。Amazon EBS ボリュームはシングルテナントです。つまり、1 人の顧客に 1 つのインスタンスが使用されます。インスタンスは決して共有されません。ランタイム環境を削除すると、Amazon EBS ボリュームも削除されます。アプリケーションを削除すると、Amazon S3 からアーティファクトと設定が削除されます。

アプリケーションログは Amazon に保存されます CloudWatch。お客様のアプリケーションログメッセージ CloudWatch も にエクスポートされます。 CloudWatch ログには、ビジネスデータやデバッグメッセージのセキュリティ情報など、顧客を区別するデータが含まれている場合があります) 。詳細については、「Amazon による AWS Mainframe Modernization のモニタリング CloudWatch」を参照してください。

さらに、1 つ以上の Amazon Elastic File System または Amazon FSx ファイルシステムをランタイム環境に接続することを選択した場合、それらのシステム内のデータは AWSに保存されます。ファイルシステムの使用を中止する場合は、そのデータをクリーンアップする必要があります。

AWS Mainframe Modernization がアプリケーションのデプロイとデータセットのインポートに使用する Amazon S3 バケットにデータを配置するときに、使用可能なすべての Amazon S3 暗号化オプションを使用してデータを保護できます。また、Amazon EFS と Amazon FSx の暗号化オプションを 1 つ以上ランタイム環境に接続すれば、これらのファイルシステムを使用できます。

AWS Mainframe Modernization サービスの保管時のデータ暗号化

AWS Mainframe Modernization は と統合 AWS Key Management Service され、データを永続的に保存するすべての依存リソース、つまり Amazon Simple Storage Service、Amazon DynamoDB、Amazon Elastic Block Store に透過的なサーバー側の暗号化 (SSE) を提供します。AWS Mainframe Modernization は、 で対称暗号化 AWS KMS キーを作成および管理します AWS KMS。

保管中のデータをデフォルトで暗号化して、機密データの保護に伴う運用のオーバーヘッドと複雑な作業を軽減できます。同時に、暗号化のコンプライアンスと規制の厳格な要件を必要とするアプリケーションを移行することもできます。

ランタイム環境とアプリケーションを作成するときに、この暗号化レイヤーを無効にしたり、別の暗号化タイプを選択したりすることはできません。

AWS Mainframe Modernization アプリケーションおよびランタイム環境には、独自のカスタマーマネージドキーを使用して Amazon S3 および Amazon EBS リソースを暗号化できます。

AWS Mainframe Modernization アプリケーションでは、このキーを使用して、アプリケーション定義と、サービスのアカウントで作成された Amazon S3 バケットに保存されている JCL ファイルなどの他のアプリケーションリソースを暗号化できます。詳細については、「アプリケーションの作成」を参照してください。

AWS Mainframe Modernization ランタイム環境では、 AWS Mainframe Modernization はカスタマーマネージドキーを使用して、作成して AWS Mainframe Modernization Amazon EC2 インスタンスに

アタッチする Amazon EBS ボリュームを暗号化します。このボリュームもサービスのアカウントにあります。詳細については、「ランタイム環境を作成する」を参照してください。

> **Note**
>
> DynamoDB リソースは常に AWS Mainframe Modernization サービスアカウントの を使用して暗号化 AWS マネージドキー されます。カスタマーマネージドキーを使用して DynamoDB リソースを暗号化することはできません。

AWS Mainframe Modernization は、次のタスクでカスタマーマネージドキーを使用します。

- アプリケーションの再デプロイ。
- AWS Mainframe Modernization Amazon EC2 インスタンスを置き換える。

AWS Mainframe Modernization は、カスタマーマネージドキーを使用して、 Mainframe Modernization アプリケーションをサポートするために作成された Amazon Relational Database Service または Amazon Aurora データベース、Amazon Simple Queue Service キュー、および Amazon AWS ElastiCache キャッシュを暗号化しません。どちらもカスタマーデータが含まれていないためです。

詳細については、「AWS Key Management Service デベロッパーガイド」の「カスタマーマネージドキー」を参照してください。

次の表は、 AWS Mainframe Modernization が機密データをどのように暗号化するかをまとめたものです。

データタイプ	AWS マネージドキー 暗号化	カスタマーマネージドキーの暗号化
`Definition` 特定のアプリケーションの定義が含まれています。	有効	有効
`EnvironmentSummary` ランタイム環境に関する情報が含まれています。	有効	有効

データタイプ	AWS マネージドキー 暗号化	カスタマーマネージドキーの暗号化
`ApplicationSummary` AWS Mainframe Modernization アプリケーションに関する情報が含まれています。	有効	有効
`DeploymentSummary` AWS Mainframe Modernization アプリケーションのデプロイに関する情報が含まれています。	有効	有効

> **Note**
>
> AWS Mainframe Modernization は、 を使用して保管時の暗号化を自動的に有効に AWS マネージドキー し、機密データを無料で保護します。ただし、カスタマーマネージドキーの使用には AWS KMS 料金が適用されます。料金の詳細については、「AWS Key Management Service 料金表」を参照してください。

の詳細については AWS KMS、「」を参照してください AWS Key Management Service。

AWS Mainframe Modernization が で許可を使用する方法 AWS KMS

AWS Mainframe Modernization では、カスタマーマネージドキーを使用するには許可が必要です。

アプリケーションまたはランタイム環境を作成するか、カスタマーマネージドキーで暗号化された AWS Mainframe Modernization にアプリケーションをデプロイすると、 AWS Mainframe Modernization は にCreateGrantリクエストを送信することで、ユーザーに代わってグラントを作成します AWS KMS。の許可 AWS KMS は、 AWS Mainframe Modernization に顧客アカウントの KMS キーへのアクセスを許可するために使用されます。

AWS Mainframe Modernization では、以下の内部オペレーションでカスタマーマネージドキーを使用するための権限が必要です。

- DescribeKey リクエストを に送信 AWS KMS して、アプリケーション、ランタイム環境、または アプリケーションのデプロイの作成時に入力した対称カスタマーマネージドキー ID が有効であることを確認します。
- AWS Mainframe Modernization ランタイム環境をホストする Amazon EC2 インスタンスにアタッチされた Amazon EBS ボリュームを暗号化 AWS KMS するGenerateDataKeyリクエストを に送信します。
- Decrypt リクエストを に送信 AWS KMS して、Amazon EBS で暗号化されたコンテンツを復号化します。

AWS Mainframe Modernization は、ランタイム環境の作成時、アプリケーションの作成または再デプロイ時、およびデプロイの作成時に、 AWS KMS 権限を使用して Secrets Manager に保存されているシークレットを復号します。 AWS Mainframe Modernization が作成する権限は、以下のオペレーションをサポートします。

- ランタイム環境許可の作成、または更新
  - Decrypt
  - 暗号化
  - ReEncryptFrom
  - ReEncryptTo
  - GenerateDataKey
  - DescribeKey
  - CreateGrant
- アプリケーション許可を作成または再デプロイします。
  - GenerateDataKey
- デプロイ許可の作成。
  - Decrypt

任意のタイミングで、許可に対するアクセス権を取り消したり、カスタマーマネージドキーに対するサービスからのアクセス権を削除したりできます。これを行うと、 AWS Mainframe Modernization はカスタマーマネージドキーによって暗号化されたデータにアクセスできなくなり、データに依存するオペレーションに影響します。例えば、 AWS Mainframe Modernization がカスタマーマネージドキーによって暗号化されたアプリケーション定義に、そのキーへの付与なしでアクセスしようとすると、アプリケーション作成オペレーションは失敗します。

AWS Mainframe Modernization は、ユーザーアプリケーション設定 (JSON ファイル) とアーティファクト (バイナリと実行可能ファイル) を収集します。また、 AWS Mainframe Modernization のオペレーションに使用されるさまざまなエンティティを追跡するメタデータを作成し、ログとメトリクスを作成します。顧客に表示されるログとメトリクスには以下が含まれます。

- CloudWatch アプリケーションとランタイムエンジン (AWS Blu Age または Micro Focus) を反映する ログ。
- CloudWatch オペレーションダッシュボードの メトリクス。

さらに、 AWS Mainframe Modernization は、サービスに関する計測、アクティビティレポートなどの使用状況データとメトリクスを収集します。このデータは顧客には表示されません。

AWS Mainframe Modernization は、データのタイプに応じて、このデータをさまざまな場所に保存します。アップロードした顧客データは Amazon S3 バケットに保存されます。サービスデータは Amazon S3 と DynamoDB の両方に保存されます。アプリケーションをデプロイすると、データとサービスデータの両方が Amazon EBS ボリュームにダウンロードされます。Amazon EFS または Amazon FSx ストレージをランタイム環境に接続することを選択した場合、それらのファイルシステムに保存されているデータも Amazon EBS ボリュームにダウンロードされます。

保管時の暗号化はデフォルトで構成されます。無効にしたり、変更したりすることはできません。現在、その設定を変更することもできません。

カスタマーマネージド キーを作成する

対称カスタマーマネージドキーは、 AWS Management Console または AWS KMS APIsを使用して作成できます。

対称カスタマーマネージドキーを作成するには

AWS Key Management Service デベロッパーガイドにある対称カスタマーマネージドキーの作成ステップを実行します。

キーポリシー

キーポリシーは、カスタマーマネージドキーへのアクセスを制御します。すべてのカスタマーマネージドキーには、キーポリシーが 1 つだけ必要です。このポリシーには、そのキーを使用できるユーザーとその使用方法を決定するステートメントが含まれています。カスタマーマネージドキーを作成する際に、キーポリシーを指定することができます。詳細については、AWS Key Management

Service デベロッパーガイド の「カスタマーマネージドキーへのアクセスの管理」を参照してください。

AWS Mainframe Modernization リソースでカスタマーマネージドキーを使用するには、キーポリシーで次の API オペレーションを許可する必要があります。

- kms:CreateGrant - カスタマーマネージドキーに許可を追加します。指定された KMS キーへのアクセスを制御します。これにより、 Mainframe Modernization に必要な許可オペレーション AWS へのアクセスが可能になります。詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMS でのグラント」を参照してください。

  これにより、 AWS Mainframe Modernization は次のことを実行できます。

  - GenerateDataKey を呼び出して、暗号化されたデータキーを生成して保存します。データキーは暗号化にすぐには使用されないからです。
  - Decrypt を呼び出して、保存されている暗号化データキーを使用して暗号化されたデータにアクセスします。
  - RetireGrant へのサービスを許可するために、削除プリンシパルを設定します。
- kms:DescribeKey — AWS Mainframe Modernization がキーを検証できるように、カスタマーマネージドキーの詳細を提供します。

AWS Mainframe Modernization では、お客様のキーポリシーに kms:CreateGrantおよび アクセスkms:DescribeKey許可が必要です。 AWS Mainframe Modernization は、このポリシーを使用して、それ自体の権限を作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "Enable IAM User Permissions",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::AccountId:role/ExampleRole"
        },
        "Action": [
            "kms:CreateGrant",
            "kms:DescribeKey"
        ],
        "Resource": "*"
    }]
}
```

> **Note**
>
> 前の例`Principal`で に示されているロールは、 `CreateApplication`や などの AWS Mainframe Modernization オペレーションに使用するロールです`CreateEnvironment`。

ポリシーでの許可の指定に関する詳細については、「AWS Key Management Service デベロッパーガイド」を参照してください。

キーアクセスのトラブルシューティングに関する詳細については、「AWS Key Management Service デベロッパーガイド」を参照してください。

AWS Mainframe Modernization のためのカスタマーマネージドキーの指定

次のリソースにカスタマーマネージドキーを指定できます。

- アプリケーション
- 環境

リソースを作成するときは、KMS ID を入力してキーを指定できます。これは、 AWS Mainframe Modernization がリソースに保存されている機密データを暗号化するために使用します。

- KMS ID - カスタマーマネージドキーのキー識別子。キー ID、キー ARN、エイリアス名、またはエイリアス ARN を入力します。

カスタマーマネージドキーは、 AWS Management Console または を使用して指定できます AWS CLI。

でランタイム環境を作成するときにカスタマーマネージドキーを指定するには AWS Management Console、「」を参照してくださいAWS Mainframe Modernization ランタイム環境を作成する。でアプリケーションを作成するときにカスタマーマネージドキーを指定するには AWS Management Console、「」を参照してくださいAWS Mainframe Modernization アプリケーションの作成。

を使用してランタイム環境を作成するときにカスタマーマネージドキーを追加するには AWS CLI、次のように `kms-key-id`パラメータを指定します。

```
aws m2 create-environment –engine-type microfocus –instance-type M2.m5.large
--publicly-accessible –engine-version 7.0.3 –name test
```

```
--high-availability-config desiredCapacity=2
--kms-key-id myEnvironmentKey
```

を使用してアプリケーションを作成するときにカスタマーマネージドキーを追加するには AWS CLI、次のように kms-key-idパラメータを指定します。

```
aws m2 create-application –name test-application –description my description
--engine-type microfocus
--definition content="$(jq -c . raw-template.json | jq -R)"
--kms-key-id myApplicationKey
```

AWS Mainframe Modernization 暗号化コンテキスト

暗号化コンテキストは、データに関する追加のコンテキスト情報が含まれたキーと値のペアのオプションのセットです。

AWS KMS は、追加の認証データとして暗号化コンテキストを使用して、認証された暗号化 をサポートします。データを暗号化するリクエストに暗号化コンテキストを含めると、 は暗号化コンテキストを暗号化されたデータに AWS KMS バインドします。データを復号化するには、そのリクエストに (暗号化時と) 同じ暗号化コンテキストを含めます。

AWS Mainframe Modernization 暗号化コンテキスト

AWS Mainframe Modernization は、アプリケーションに関連するすべての AWS KMS 暗号化オペレーション (アプリケーションの作成とデプロイの作成) で同じ暗号化コンテキストを使用します。ここで、キーは aws:m2:appで、値はアプリケーションの一意の識別子です。

Example

```
"encryptionContextSubset": {
        "aws:m2:app": "a1bc2defabc3defabc4defabcd"
}
```

モニタリングに暗号化コンテキストを使用する

対称カスタマーマネージドキーを使用してアプリケーションやランタイム環境を暗号化する場合は、カスタマーマネージドキーがどのように使用されているかを特定するために、暗号化コンテキストを監査記録やログで使用することもできます。

暗号化コンテキストを使用してカスタマーマネージドキーへのアクセスを制御する

対称カスタマーマネージドキー (CMK) へのアクセスを制御するための conditions として、キーポリシーと IAM ポリシー内の暗号化コンテキストを使用することができます。付与する際に、暗号化コンテキストの制約を使用することもできます。

AWS Mainframe Modernization は、権限で暗号化コンテキストの制約を使用して、アカウントまたはリージョンのカスタマーマネージドキーへのアクセスを制御します。権限の制約では、権限によって許可されるオペレーションで指定された暗号化コンテキストを使用する必要があります。次の例は、 AWS Mainframe Modernization がアプリケーションの作成時にアプリケーションアーティファクトを暗号化するために活用する許可です。

```
//This grant is retired immediately after create application finish
{
   "grantee-principal": m2.us-west-2.amazonaws.com,
   "retiring-principal": m2.us-west-2.amazonaws.com,
   "operations": [
       "GenerateDataKey"
   ]
   "condition": {
        "encryptionContextSubset": {
            “aws:m2:app”: “a1bc2defabc3defabc4defabcd”
   }
}
```

AWS Mainframe Modernization のための暗号化キーのモニタリング

AWS Mainframe Modernization リソースで AWS KMS カスタマーマネージドキーを使用すると、AWS CloudTrailまたは Amazon CloudWatch Logs を使用して、 AWS Mainframe Modernization がに送信するリクエストを追跡できます AWS KMS。

ランタイム環境の例

次の例はDescribeKey、カスタマーマネージドキーで暗号化されたデータにアクセスするために AWS Mainframe Modernization によって呼び出される KMS オペレーションをモニタリングDecryptするための CreateGrant、GenerateDataKey、、および の AWS CloudTrail イベントです。

DescribeKey

AWS Mainframe Modernization は DescribeKeyオペレーションを使用して、ランタイム環境に関連付けられた AWS KMS カスタマーマネージドキーがアカウントとリージョンに存在するかどうかを確認します。

以下のイベント例では DescribeKey オペレーションを記録しています。


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T19:40:26Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-12-06T20:23:43Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.182",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "keyId": "00dd0db0-0000-0000-ac00-b0c000SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
        "clientProvidedHostHeader": "kms.us-west-2.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

CreateGrant

AWS KMS カスタマーマネージドキーを使用してランタイム環境を暗号化すると、 AWS Mainframe Modernization はユーザーに代わって必要な KMS オペレーションを実行するための複数の`CreateGrant`リクエストを送信します。 AWS Mainframe Modernization が作成する許可の一部は、使用後すぐに廃止されます。その他は、ランタイム環境を削除すると廃止されます。

次のイベント例は、環境作成ワークフローに関連付けられた Lambda 実行ロールの `CreateGrant` オペレーションを記録します。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:11:45Z",
```

```
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "operations": [
            "Encrypt",
            "Decrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "GenerateDataKey",
            "GenerateDataKey",
            "DescribeKey",
            "CreateGrant"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

次のイベント例は、Auto Scaling グループのサービスにリンクされたロールの CreateGrant オペレーションを記録します。環境作成ワークフローに関連付けられた Lambda 実行ロールがこの CreateGrant オペレーションを呼び出します。Auto Scaling グループのサービスにリンクされたロールに対してサブグラントを作成するアクセス許可を実行ロールに付与します。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65MZFUPM4JO:EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfxDj5zz86tr",
        "arn": "arn:aws:sts::111122223333:assumed-role/EnvironmentWorkflow-
alpha-CreateEnvironmentLambdaS-1AU4A8VNQEEKN/EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfxDj5zz86tr",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4A8VNQEEKN",
                "accountId": "111122223333",
                "userName": "EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4A8VNQEEKN"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:22:28Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-12-06T20:23:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
```

```
    "sourceIPAddress": "54.148.236.160",
    "userAgent": "aws-sdk-java/2.18.21 Linux/4.14.255-276-224.499.amzn2.x86_64
 OpenJDK_64-Bit_Server_VM/11.0.14.1+10-LTS Java/11.0.14.1 vendor/Amazon.com_Inc. md/
internal exec-env/AWS_Lambda_java11 io/sync http/Apache cfg/retry-mode/legacy",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "operations": [
            "Encrypt",
            "Decrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "GenerateDataKey",
            "GenerateDataKey",
            "DescribeKey",
            "CreateGrant"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
```

```
        "clientProvidedHostHeader": "kms.us-west-2.amazonaws.com"
    }
}
}
```

GenerateDataKey

ランタイム環境リソースの AWS KMS カスタマーマネージドキーを有効にすると、Auto Scaling はランタイム環境に関連付けられた Amazon EBS ボリュームを暗号化するための一意のキーを作成します。リソースの AWS KMS カスタマーマネージドキー AWS KMS を指定するGenerateDataKeyリクエストを に送信します。

以下のイベント例では GenerateDataKey オペレーションを記録しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65EEXVIEH7D:AutoScaling",
        "arn": "arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForAutoScaling/
AutoScaling",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAutoScaling"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:23:16Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "autoscaling.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:18Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
```

```
    "awsRegion": "us-west-2",
    "sourceIPAddress": "autoscaling.amazonaws.com",
    "userAgent": "autoscaling.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:ebs:id": "vol-080f7a32d290807f3"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "numberOfBytes": 64
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

暗号化されたランタイム環境にアクセスすると、Amazon EBS は `Decrypt` オペレーションを呼び出し、保存されている暗号化されたデータキーを使用して暗号化されたデータにアクセスします。

以下のイベント例では `Decrypt` オペレーションを記録しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ebs.amazonaws.com"
    },
```

```
    "eventTime": "2022-12-06T20:23:22Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ebs.amazonaws.com",
    "userAgent": "ebs.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "aws:ebs:id": "vol-080f7a32d290807f3"
        }
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventCategory": "Management"
}
```

アプリケーションのサンプル

次の例は、カスタマーマネージドキーで暗号化されたデータにアクセスするために AWS Mainframe Modernization によって呼び出される KMS オペレーションをモニタリング`GenerateDataKey`するための `CreateGrant`および の AWS CloudTrail イベントです。

CreateGrant

AWS KMS カスタマーマネージドキーを使用してアプリケーションリソースを暗号化すると、Lambda 実行ロールは`CreateGrant`ユーザーに代わって AWS リクエストを送信し、アカウントの KMS キーにアクセスします。この許可により、Lambda 実行ロールはカスタマーマネー

ジドキーを使用してカスタマーアプリケーションリソースを Amazon S3 にアップロードできます。この許可は、アプリケーションが作成された直後に廃止されます。

以下のイベント例では CreateGrant オペレーションを記録しています。


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T21:51:45Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T22:47:04Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "constraints": {
            "encryptionContextSubset": {
                "aws:m2:app": "a1bc2defabc3defabc4defabcd"
            }
        },
        "retiringPrincipal": "m2.us-west-2.amazonaws.com",
```

```
        "operations": [
            "GenerateDataKey"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateDataKey

アプリケーションリソースの AWS KMS カスタマーマネージドキーを有効にすると、Lambda 実行ロールは、顧客データの暗号化と Amazon Simple Storage Service へのアップロードに使用するキーを作成します。Lambda 実行ロールは、リソースの AWS KMS カスタマーマネージドキー AWS KMS を指定するGenerateDataKeyリクエストを に送信します。

以下のイベント例では、GenerateDataKey オペレーションを記録しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65CLCEKKC7Z:ApplicationWorkflow-alpha-
CreateApplicationVersion-CstWZUn5R4u6",
```

```
        "arn": "arn:aws:sts::111122223333:assumed-role/ApplicationWorkflow-
alpha-CreateApplicationVersion-1IZRBZYDG20B/ApplicationWorkflow-alpha-
CreateApplicationVersion-CstWZUn5R4u6",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/ApplicationWorkflow-alpha-
CreateApplicationVersion-1IZRBZYDG20B",
                "accountId": "111122223333",
                "userName": "ApplicationWorkflow-alpha-
CreateApplicationVersion-1IZRBZYDG20B"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T23:28:32Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:29:08Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:m2:app": "a1bc2defabc3defabc4defabcd",
            "aws:s3:arn": "arn:aws:s3:::supernova-processedtemplate-111122223333-us-
west-2/111122223333/a1bc2defabc3defabc4defabcd/1/cics-transaction/ZBNKE35.so"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
```

```
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

デプロイの例

次の例は、カスタマーマネージドキーで暗号化されたデータにアクセスするために AWS Mainframe Modernization によって呼び出される KMS オペレーションをモニタリング`Decrypt`するための `CreateGrant`および の AWS CloudTrail イベントです。

CreateGrant

AWS KMS カスタマーマネージドキーを使用してデプロイリソースを暗号化すると、 AWS Mainframe Modernization はユーザーに代わって 2 つの`CreateGrant`リクエストを送信します。最初の許可は、現在の Lambda 実行ロールに対して を呼び出し ListBatchJobScriptFiles、デプロイが終了した直後に廃止されます。2 つ目の許可は、Amazon EC2 がお客様のアプリケーションリソースを Amazon S3 からダウンロードできるようにするための Amazon EC2 のスコープダウンインスタンスロールに対するものです。この許可は、アプリケーションがランタイム環境から削除されると廃止されます。

以下のイベント例では、`CreateGrant` オペレーションを記録しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
```

```
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T21:51:45Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:40:07Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:m2:app": "a1bc2defabc3defabc4defabcd"
            }
        },
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
```

```
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

デプロイにアクセスすると、Amazon EC2 は Decrypt オペレーションを呼び出し、保存されている暗号化されたデータキーを使用して、暗号化された顧客データを復号化して Amazon S3 からダウンロードします。

以下のイベント例では、Decrypt オペレーションを記録しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65BSPZ37E6G:m2-hm-bqe367dxtfcpdbzmnhfzranisu",
        "arn": "arn:aws:sts::111122223333:assumed-role/
SupernovaEnvironmentInstanceScopeDownRole/m2-hm-bqe367dxtfcpdbzmnhfzranisu",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/
SupernovaEnvironmentInstanceScopeDownRole",
                "accountId": "111122223333",
                "userName": "SupernovaEnvironmentInstanceScopeDownRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T23:19:29Z",
                "mfaAuthenticated": "false"
```

```
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:40:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:m2:app": "a1bc2defabc3defabc4defabcdm",
            "aws:s3:arn": "arn:aws:s3:::supernova-processedtemplate-111122223333-us-
west-2/111122223333/a1bc2defabc3defabc4defabcdm/1/cics-transaction/BBANK40P.so"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


詳細はこちら

次のリソースは、保管時のデータ暗号化についての詳細を説明しています。

- 詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS Key Management Service の概念」を参照してください。

- 詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS Key Management Serviceのセキュリティのベストプラクティス」を参照してください。

転送中の暗号化

トランザクションワークロードの一部であるインタラクティブなアプリケーションの場合、ターミナルエミュレータと TN3270 プロトコルの AWS Mainframe Modernization サービスエンドポイント間のデータ交換は転送中に暗号化されません。アプリケーションが転送中に暗号化を必要とする場合は、追加のトンネリングメカニズムを実装してください。

AWS Mainframe Modernization は HTTPS を使用してサービス APIs。 AWS Mainframe Modernization 内の他のすべての通信は、HTTPS. AWS Mainframe Modernization だけでなく、サービス VPC またはセキュリティグループによって保護されます。Mainframe Modernization は、アプリケーションのアーティファクト、設定、およびアプリケーションデータを転送します。アプリケーションアーティファクトは、アプリケーションデータと同様に、所有している Amazon S3 バケットからコピーされます。Amazon S3 へのリンクを使用するか、ファイルをローカルにアップロードして、アプリケーション設定を提供できます。

転送中の基本的な暗号化はデフォルトで設定されますが、TN3270 プロトコルには適用されません。AWS Mainframe Modernization は API エンドポイントに HTTPS を使用します。この HTTPS もデフォルトで設定されています。

AWS Mainframe Modernization の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS Mainframe Modernization リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- AWS Mainframe Modernization と IAM の連携方法

- AWS Mainframe Modernization のアイデンティティベースのポリシーの例
- AWS Mainframe Modernization のアイデンティティとアクセスのトラブルシューティング
- Mainframe Modernization のサービスリンクロールの使用

対象者

AWS Identity and Access Management (IAM) の使用方法は、 AWS Mainframe Modernization で行う作業によって異なります。

サービスユーザー – Mainframe Modernization AWS サービスを使用してジョブを実行する場合は、管理者から必要な認証情報とアクセス許可が与えられます。さらに多くの AWS Mainframe Modernization 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。 AWS Mainframe Modernization の機能にアクセスできない場合は、「AWS Mainframe Modernization のアイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の AWS Mainframe Modernization リソースを担当している場合は、通常、AWS Mainframe Modernization へのフルアクセスがあります。サービスのユーザーがどの AWS Mainframe Modernization 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。 AWS Mainframe Modernization で IAM を利用する方法の詳細については、「」を参照してくださいAWS Mainframe Modernization と IAM の連携方法。

IAM 管理者 – IAM 管理者は、 AWS Mainframe Modernization へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できる AWS Mainframe Modernization アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS Mainframe Modernization のアイデンティティベースのポリシーの例。

アイデンティティを使用した認証

認証は、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって認証 (にサインイン AWS) される必要があります。

ID ソース (AWS IAM Identity Center) から提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーションア

イデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けます。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 にサインインする方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、 認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストを自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、「IAM ユーザーガイド」のAWS 「 API リクエストの署名」を参照してください。

使用する認証方法を問わず、セキュリティ情報の提供を追加でリクエストされる場合もあります。例えば、 では、多要素認証 (MFA) を使用してアカウントのセキュリティを高めることを AWS 推奨しています。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証 (MFA)」および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての AWS のサービス およびリソースへの完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報を保護し、それらを使用してルートユーザーのみが実行できるタスクを実行してください。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーション ID

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な AWS のサービス 認証情報を使用して にアクセスする ID プロバイダーとのフェデレーションの使用を要求します。

フェデレーティッド ID とは、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、、 AWS Directory Service、Identity Center ディレクトリのユーザー、または ID ソースから提供された認証情報 AWS のサービス を使用して にアクセスするすべてのユーザーです。フェデレー

ティッド ID が にアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、すべての とアプリケーションで使用する独自の ID ソースのユーザー AWS アカウント とグループのセットに接続して同期することもできます。IAM アイデンティティセンターの詳細については、[AWS IAM Identity Center ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールを切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

一時的な認証情報を持った IAM ロールは、以下の状況で役立ちます。

- フェデレーションユーザーアクセス – フェデレーションアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーションアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「権限セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — 一部の では、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスにリンクされたロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、を呼び出すプリンシパルのアクセス許可 AWS のサービス を使用し AWS のサービス、ダウンストリームサービスにリクエストを行うリクエストと組み合わせて使用します。FAS リクエストは、他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストをサービスが受信した場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細に

ついては、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを作成しているアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「IAM ロールの作成が適している場合 (ユーザーではなく)」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは のオブジェクト AWS であり、アイデンティティまたはリソースに関連付けられると、これらのアクセス許可を定義します。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらに インラインポリシー または マネージドポリシー に分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。管理ポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーで IAM の AWS マネージドポリシーを使用することはできません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、追加の一般的でないポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- 権限の境界 - 権限の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティに権限の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとその権限の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、権限の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。権限の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティの権限の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。組織と SCP の詳細については、「AWS Organizations ユーザーガイド」の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限される範囲は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連する場合に がリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシーの評価ロジック」を参照してください。

AWS Mainframe Modernization と IAM の連携方法

IAM を使用して AWS Mainframe Modernization へのアクセスを管理する前に、 Mainframe Modernization で使用できる IAM AWS 機能について学びます。

AWS Mainframe Modernization で使用できる IAM の機能

IAM 機能	AWS Mainframe Modernization のサポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	いいえ
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー	はい
ACL	なし
ABAC (ポリシー内のタグ)	はい
一時的な認証情報	はい
転送アクセスセッション (FAS)	はい
サービスロール	あり
サービスリンクロール	はい

AWS Mainframe Modernization およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

AWS Mainframe Modernization のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

AWS Mainframe Modernization のアイデンティティベースのポリシーの例

AWS Mainframe Modernization アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS Mainframe Modernization のアイデンティティベースのポリシーの例。

AWS Mainframe Modernization 内のリソースベースのポリシー

リソースベースのポリシーのサポート	なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、リソースへのアクセス許可をプリンシパルエンティティ (ユーザーまたはロール) に付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにア

タッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

AWS Mainframe Modernization のポリシーアクション

ポリシーアクションに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーの`Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWS API オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

AWS Mainframe Modernization アクションのリストを確認するには、「サービス認証リファレンス」のAWS 「 Mainframe Modernization で定義されるアクション」を参照してください。

AWS Mainframe Modernization のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
m2
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "m2:StartApplication",
      "m2:StopApplication"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、List という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "m2:List*"
```

AWS Mainframe Modernization アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS Mainframe Modernization のアイデンティティベースのポリシーの例。

AWS Mainframe Modernization のポリシーリソース

ポリシーリソースに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシーの要素は、オブジェクトあるいはアクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティスとしては、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

ARNs を使用して IAM ポリシーが適用されるリソースを識別することで、特定の AWS Mainframe Modernization リソースへのアクセスを制限できます。ARN の形式の詳細については、「AWS 全般のリファレンス」の「Amazon リソースネーム (ARN)」を参照してください。

例えば、 AWS Mainframe Modernization 環境には次の ARN があります。

```
"Resource": "arn:aws:m2:regionId:accountId:env/service-generated-unique-identifier"
```

AWS Mainframe Modernization アプリケーションには、次の ARN があります。

```
"Resource": "arn:aws:m2:regionId:accountId:app/service-generated-unique-identifier"
```

すべての AWS Mainframe Modernization アクションがリソースレベルのアクセス許可をサポートしているわけではありません。リソースレベルの許可をサポートしていないアクションの場合、ワイルドカード (*) を使用する必要があります。

次の AWS Mainframe Modernization アクションは、リソースレベルのアクセス許可をサポートしていません。

```
ListApplications
            ListApplicationVersions
            ListBatchJobDefinitions
            ListBatchJobExecutions
            ListDataSetImportHistory
            ListDataSets
            ListDeployments
            ListEngineVersions
            ListEnvironments
            ListTagsForResource
```

AWS Mainframe Modernization リソースのタイプとその ARNs」のAWS 「 Mainframe Modernization で定義されるリソース」を参照してください。 どのアクションで各リソースの ARN を指定できるかについては、AWS 「 Mainframe Modernization で定義されるアクション」を参照してください。

AWS Mainframe Modernization アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS Mainframe Modernization のアイデンティティベースのポリシーの例。

AWS Mainframe Modernization API のアクセス許可: アクション、リソース、条件のリファレンス

IAM ID にアタッチできるアクセス許可ポリシー (ID ベースのポリシー) を作成する場合は、次の表をリファレンスとして使用できます。表には、次のものが含まれています。

- 各 AWS Mainframe Modernization API オペレーション
- アクションを実行するためのアクセス許可を付与できる、対応するアクション

- アクセス許可を付与できる AWS リソース

ポリシーの `Action` フィールドでアクションを指定し、ポリシーの `Resource` フィールドでリソースの値を指定します。

AWS Mainframe Modernization ポリシーで AWS グローバル条件キーを使用して、条件を表現できます。 AWS キーの完全なリストについては、IAM ユーザーガイドの「使用可能なグローバル条件キー」を参照してください。

> **Note**
>
> アクションを指定するには、API オペレーション名 (`m2:CreateApplication` など) の前に `m2:` プレフィックスを使用します。

AWS Mainframe Modernization API とアクションに必要なアクセス許可

AWS Mainframe Modernization API オペレーション	必要な許可 (API アクション)	リソース
CancelBatchJobExecution		アプリケーション
CreateApplication	`s3:GetObject` `s3:ListBucket`	アプリケーション
CreateDataSetImportTask	`m2:CreateDataSetImportTask` `s3:GetObject`	アプリケーション

CreateDeployment	`elasticloadbalancing:AddTags`
	`elasticloadbalancing:CreateListener`
	`elasticloadbalancing:CreateTargetGroup`
		アプリケーション

AWS Mainframe Modernization API オペレーション	必要な許可 (API アクション)	リソース
	elasticloadbalancing:RegisterTargets	
CreateEnvironment	ec2:CreateNetworkInterface ec2:CreateNetworkInterfacePermission ec2:DescribeNetworkInterfaces ec2:DescribeSecurityGroups ec2:DescribeSubnets ec2:DescribeVpcAttribute ec2:DescribeVpcs ec2:ModifyNetworkInterfaceAttribute elasticfilesystem:DescribeMountTargets elasticloadbalancing:AddTags elasticloadbalancing:CreateLoadBalancer fsx:DescribeFileSystems iam:CreateServiceLinkedRole	環境

AWS Mainframe Modernization API オペレーション	必要な許可 (API アクション)	リソース
DeleteApplication	elasticloadbalancing:DeleteListener elasticloadbalancing:DeleteTargetGroup logs:DeleteLogDelivery	アプリケーション
DeleteApplicationFromEnvironment	elasticloadbalancing:DeleteListener elasticloadbalancing:DeleteTargetGroup	アプリケーション 環境
DeleteEnvironment	elasticloadbalancing:DeleteLoadBalancer	環境
GetApplication		アプリケーション
GetApplicationVersion		アプリケーション
GetBatchJobExecution		アプリケーション
GetDataSetDetails		アプリケーション
GetDataSetImportTask		アプリケーション
GetDeployment		アプリケーション
GetEnvironment		環境
ListApplications		*
ListApplicationVersions		*

AWS Mainframe Modernization API オペレーション	必要な許可 (API アクション)	リソース
ListBatchJobDefinitions		*
ListBatchJobExecutions		*
ListDataSetImportHistory		*
ListDataSets		*
ListDeployments		*
ListEngineVersions		*
ListEnvironments		*
ListTagsForResource		*
StartApplication		アプリケーション
StartBatchJob		アプリケーション
StopApplication		アプリケーション
TagResource		*
UntagResource		*
UpdateApplication	`s3:GetObject` `s3:ListBucket`	アプリケーション
UpdateEnvironment		環境

AWS Mainframe Modernization のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。equal や less than などの条件演算子を使用して条件式を作成することによって、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS は AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理OR演算を使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、 グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

次の条件キーは Mainframe Modernization AWS に固有のものです。

```
m2:EngineType
            m2:InstanceType
```

AWS Mainframe Modernization の条件キーのリストを確認するには、「サービス認証リファレンス」のAWS 「 Mainframe Modernization の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、AWS 「 Mainframe Modernization で定義されるアクション」を参照してください。

AWS Mainframe Modernization アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS Mainframe Modernization のアイデンティティベースのポリシーの例。

AWS Mainframe Modernization のアクセスコントロールリスト (ACL)

ACL のサポート	なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

AWS Mainframe Modernization による属性ベースのアクセスコントロール (ABAC)

ABAC のサポート (ポリシー内のタグ)	はい

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。次に、プリンシパルのタグがアクセスを試行するリソースのタグと一致したときにオペレーションを許可するよう、ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを制御するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素 でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値は Yes です。サービスが一部のリソースタイプに対してのみ 3 つの条件キーすべてをサポートする場合、値は Partial です。

ABAC の詳細については、「IAM ユーザーガイド」の [ABAC とは?」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセス制御 (ABAC) を使用する」を参照してください。

AWS Mainframe Modernization での一時的な認証情報の使用

一時的な認証情報のサポート	はい

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドの「IAM AWS のサービス と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合は、一時的な認証情報を使用しています。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスは自動的に一時的な認証情報を作成します。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して、長期的なアクセスキーを使用する代わりに、動的に一時的な認証情報を生成する AWS. AWS recommends にアクセスできます。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

AWS Mainframe Modernization の転送アクセスセッション

フォワードアクセスセッション (FAS) をサポート	はい

IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可 AWS のサービス を使用し AWS のサービス、ダウンストリームサービスにリクエストを行うリクエストと組み合わせて使用します。FAS リクエストは、他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストをサービスが受信した場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

> **Important**
>
> これらのトークンは、明示的な合意なしに AWS Mainframe Modernization が顧客データにアクセスできるようにします。例えば、 AWS Mainframe Modernization は、顧客から明示的な許可を取得せずに、Amazon S3 バケットから関連するビジネスデータを含むアプリケーションアーティファクトをデプロイします。それに応じて、コンプライアンスドキュメントを更新する必要がある場合があります。

AWS Mainframe Modernization のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

AWS Mainframe Modernization は、アクティビティフック (トランザクション/ジョブの中止または完了など) のサービスロールをサポートします。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、 AWS Mainframe Modernization の機能が破損する可能性があります。 AWS Mainframe Modernization が指示する場合以外は、サービスロールを編集しないでください。

AWS Mainframe Modernization での IAM ロールの選択

Amazon EC2 で実行されているアプリケーションが引き受けることができる IAM ロールを以前に作成している場合は、起動テンプレートまたは起動設定を作成するときにこのロールを選択できます。AWS Mainframe Modernization は、選択するロールのリストを提供します。これらのロールを作成するとき、アプリケーションが必要とする特定の API コールへのアクセスを制限する最小権限の特権 IAM ポリシーを関連付けることが重要です。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに対する IAM ロール」を参照してください。

AWS Mainframe Modernization のサービスにリンクされたロール

サービスリンクロールのサポート	はい

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービ

スにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの権限を表示できますが、編集することはできません。

AWS Mainframe Modernization のサービスにリンクされたロールの作成または管理の詳細については、「」を参照してくださいMainframe Modernization のサービスリンクロールの使用。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携するAWS サービス」を参照してください。表の中から、「サービスにリンクされたロール」列に「Yes」と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

AWS Mainframe Modernization のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーとロールには AWS Mainframe Modernization リソースを作成または変更するアクセス許可はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者がロールに IAM ポリシーを追加すると、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドの「IAM ポリシーの作成」を参照してください。

各リソースタイプの ARN の形式など、 AWS Mainframe Modernization で定義されるアクションとリソースタイプの詳細については、「サービス認証リファレンス」の「Actions, Resources, and Condition Keys for AWS Mainframe Modernization」を参照してください。 ARNs

トピック

- ポリシーのベストプラクティス
- AWS Mainframe Modernization コンソールの使用
- 自分の許可の表示をユーザーに許可する

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰かが AWS Mainframe Modernization リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースのポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください。

- AWS 管理ポリシーの使用を開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的ユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、 などの特定の を通じてサービスアクションを使用する場合 AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

AWS Mainframe Modernization コンソールの使用

AWS Mainframe Modernization コンソールにアクセスするには、一連の最小限のアクセス許可が必要です。これらのアクセス許可により、 の AWS Mainframe Modernization リソースの詳細をリストおよび表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベー

スのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが AWS Mainframe Modernization コンソールを使用できるようにするには、エンティティに AWS Mainframe Modernization `ConsoleAccess`または `ReadOnly` AWS 管理ポリシーもアタッチします。詳細については、『IAM ユーザーガイド』の「ユーザーへの権限の追加」を参照してください。

自分の許可の表示をユーザーに許可する

この例では、ユーザーアイデンティティに添付されたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーを作成する方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
```

```
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS Mainframe Modernization のアイデンティティとアクセスのトラブルシューティング

次の情報は、 AWS Mainframe Modernization と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- iam を実行する権限がありません。PassRole
- 自分の 以外のユーザーに AWS Mainframe Modernization リソース AWS アカウント へのアクセスを許可したい

iam を実行する権限がありません。PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWS Mainframe Modernization にロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して AWS Mainframe Modernization でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新して Mary に `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに AWS Mainframe Modernization リソース AWS アカウントへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- AWS Mainframe Modernization がこれらの機能をサポートしているかどうかを確認するには、「」を参照してくださいAWS Mainframe Modernization と IAM の連携方法。
- 所有 AWS アカウント する のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント する別の の IAM ユーザーへのアクセスを許可する」を参照してください。
- リソースへのアクセスをサードパーティーの に提供する方法については AWS アカウント、IAM ユーザーガイドの「第三者 AWS アカウント が所有する へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権の提供」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Mainframe Modernization のサービスリンクロールの使用

AWS Mainframe Modernization には AWS Identity and Access Management (IAM) サービスリンクロールを使用します。サービスリンクロールは、Mainframe Modernization に直接リンクされた特殊な IAM ロールです。サービスリンクロールは、Mainframe Modernization によって事前定義されてお

り、これにはユーザーの代わりにサービスから他の AWS サービスを呼び出す際に必要な許可がすべて含まれています。

サービスリンクロールを使用すると、必要な許可を手動で追加する必要がないため、Mainframe Modernization のセットアップが簡単になります。Mainframe Modernization がサービスリンクロールの許可を定義し、他の定義がない限り、Mainframe Modernization のみこのロールを引き受けることができます。定義される許可には、信頼ポリシーと許可ポリシーが含まれており、この許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールは、まずその関連リソースを削除しなければ削除できません。これにより、リソースにアクセスするための許可が誤って削除されることを防ぐため、Mainframe Modernization リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS のサービス」を参照し、Service-linked roles (サービスにリンクされたロール) の列内で Yes (はい) と表記されたサービスを確認してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

サービスリンクロールにおけるアクセス許可

Mainframe Modernization では、AWSServiceRoleForAWSM2 というサービスリンクロールを使用します。この使用のために、VPC に接続してファイルシステムなどのリソースにアクセスできるようにネットワークを設定します。

サービスリンクロール AWSServiceRoleForAWSM2 は、以下のサービスを信頼してロールを引き受けます。

- `m2.amazonaws.com`

サービスリンクロール AWSM2ServicePolicy のロール許可ポリシーでは、Mainframe Modernization は、指定されたリソースにおいて以下のアクションを完了できます。

- Mainframe Modernization 環境用の Amazon EC2 ネットワークインターフェイスへのアクセス権限を作成、削除、記述、アタッチして、顧客の VPC 接続を確立します。
- Elastic Load Balancing からエントリを登録または登録解除します。これにより、顧客は Mainframe Modernization 環境に接続できます。
- Amazon EFS または Amazon FSx ファイルシステムを使用している場合は、その内容について記述してください。
- ランタイム環境から顧客の CloudWatch にメトリクスを送信します。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeSubnets",
    "ec2:CreateNetworkInterface",
    "ec2:DeleteNetworkInterface",
    "ec2:DescribeNetworkInterfaces",
    "ec2:CreateNetworkInterfacePermission",
    "ec2:ModifyNetworkInterfaceAttribute"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "elasticfilesystem:DescribeMountTargets"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "elasticloadbalancing:RegisterTargets",
    "elasticloadbalancing:DeregisterTargets"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "fsx:DescribeFileSystems"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "cloudwatch:PutMetricData"
   ],
   "Resource": "*",
```

```
    "Condition": {
     "StringEquals": {
      "cloudwatch:namespace": [
       "AWS/M2"
      ]
     }
    }
   }
  ]
}
```

サービスにリンクされたロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスリンクロールのアクセス許可」を参照してください。

Mainframe Modernization のサービスリンクロールの作成

サービスリンクロールを手動で作成する必要はありません。AWS Management Console、AWS CLI、AWS API でランタイム環境を作成した場合、Mainframe Modernization によってサービスリンクロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。ランタイム環境を作成すると、Mainframe Modernization によってサービスリンクロールが再度作成されます。

Mainframe Modernization のサービスリンクロールの編集

Mainframe Modernization では、サービスリンクロール [AWSServiceRoleForAWSM2] を編集できません。サービスリンクロールを作成した後は、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの編集」を参照してください。

Mainframe Modernization のサービスリンクロールの削除

サービスリンクロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンクロールのリソースをクリーンアップする必要があります。

> **Note**
>
> リソースを削除する際に Mainframe Modernization サービスがロールを使用している場合は、削除が失敗する可能性があります。失敗した場合は、数分待ってから操作を再試行してください。

AWSServiceRoleForAWSM2 によって使用されている Mainframe Modernization リソースを削除するには、

- Mainframe Modernization のランタイム環境を削除します。環境自体を削除する前に、必ず環境からアプリケーションを削除してください。

サービスにリンクされたロールを IAM で手動削除するには

IAM コンソール、AWS CLI、AWS API を使用して、サービスリンクロール AWSServiceRoleForAWSM2 を削除します。詳細については、IAM ユーザーガイド の サービスにリンクされたロールの削除 を参照してください。

サービスリンクロールをサポートするリージョン

Mainframe Modernization によって、このサービスを利用できるすべてのリージョンにおいてサービスリンクロールの使用がサポートされます。詳細については、「AWS リージョンとエンドポイント」を参照してください。

AWS Mainframe Modernization のためのコンプライアンス検証

AWS Mainfrome Modernization のセキュリティとコンプライアンスは、複数の AWS コンプライアンスプログラムの一環として、サードパーティー監査機関によって評価されます。これらのプログラムには、SOC、PCI、FedRAMP、HIPAA などが含まれます。

特定のコンプライアンスプログラムの対象となる AWS サービスのリストについては、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」を参照してください。一般的な情報については、AWS コンプライアンスプログラムを参照してください。

AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、「AWS Artifact 内のレポートのダウンロード」「」を参照してください。

AWS Mainframe Modernization のサービスを使用する際のお客様のコンプライアンス責任は、データの機密性、企業のコンプライアンス目的、適用法規によって決まります。AWS ではコンプライアンスに役立つ以下のリソースを用意しています。

- 「セキュリティとコンプライアンスのクイックスタートガイド」「」 – これらのデプロイガイドには、アーキテクチャ上の考慮事項の説明と、AWS でセキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイするためのステップが記載されています。
- HIPAA セキュリティおよびコンプライアンスホワイトペーパーのアーキテクチャの設計 - このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWSコンプライアンスのリソース– このワークブックおよびガイドのコレクションは、お客様の業界と拠点に適用されるものである場合があります。
- AWS Configデベロッパーガイドのルールでのリソースの評価 – AWS Configは、リソース設定が、社内のプラクティス、業界のガイドラインそして規制にどの程度適合しているのかを評価します。
- AWS Security Hub– AWSのこのサービスは、 AWS内でのユーザーのセキュリティ状態に関する包括的な見解を提供し、業界のセキュリティ標準、およびベストプラクティスに対するコンプライアンスを確認するために役立ちます。

AWS Mainframe Modernization の耐障害性

AWS のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心として構築されます。リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS グローバルインフラストラクチャを参照してください。

AWS Mainframe Modernization でのインフラストラクチャセキュリティ

マネージドサービスである AWS Mainframe Modernization は AWS グローバルネットワークセキュリティで保護されています。AWSセキュリティサービスと AWS がインフラストラクチャを保護す

る方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

ネットワーク経由で Mainframe Modernization にアクセスするには、AWS が発行した API コールを使用します。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS) TLS 1.2 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

インターフェイスエンドポイントを使用して AWS Mainframe Modernization にアクセス (AWS PrivateLink)

AWS PrivateLink を使用して、VPC と AWS Mainframe Modernization の間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、AWS Direct Connect 接続のいずれかを使用せずに、VPC 内にあるかのように Mainframe Modernization にアクセスできます。VPC のインスタンスは、パブリック IP アドレスがなくても Mainframe Modernization にアクセスできます。

このプライベート接続を確立するには、AWS PrivateLink を利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、Mainframe Modernization 宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については、『AWS PrivateLink ガイド)』の「AWS のサービス でアクセスする)」を参照してください。

Mainframe Modernization に関する考慮事項

Mainframe Modernization のインターフェイスエンドポイントを設定する前に、AWS PrivateLink ガイドの「考慮事項」を確認してください。

Mainframe Modernization は、インターフェイスエンドポイントを介してすべての API アクションの呼び出しをサポートしています。

Mainframe Modernization 用のインターフェイスエンドポイントの作成

Amazon VPC コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、Mainframe Modernization のインターフェイスエンドポイントを作成できます。詳細については、AWS PrivateLink ガイドの「Mainframe Modernization エンドポイントの作成」を参照してください。

以下のサービス名を使用して、Mainframe Modernization のインターフェイスエンドポイントを作成します。

```
com.amazonaws.region.m2
```

インターフェイスエンドポイントのプライベート DNS を有効にすると、リージョンのデフォルト DNS 名を使用して、Mainframe Modernization に API リクエストを実行できます。例えば、`m2.us-east-1.amazonaws.com` です。

インターフェイスエンドポイントのエンドポイントポリシーを作成する

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーでは、インターフェイスエンドポイント経由での Mainframe Modernization へのフルアクセスが許可されています。VPC から Mainframe Modernization への許可されたアクセスをコントロールするには、カスタムエンドポイントポリシーをインターフェイスエンドポイントにアタッチします。

エンドポイントポリシーは、以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、ユーザー、IAM ロール)。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、AWS PrivateLink ガイドのControl access to services using endpoint policies (エンドポイントポリシーを使用してサービスへのアクセスをコントロールする)を参照してください。

例: Mainframe Modernization アクションの VPC エンドポイントポリシー

以下は、カスタムエンドポイントポリシーの例です。インターフェイスエンドポイントにアタッチされると、このポリシーは、すべてのリソースですべてのプリンシパルに、リストされている Mainframe Modernization アクションへのアクセス権を付与します。

```
//Example of an endpoint policy where access is granted to the
//listed AWS Mainframe Modernization actions for all principals on all resources
{"Statement": [
      {"Principal": "*",
         "Effect": "Allow",
         "Action": [
            "m2:ListApplications",
            "m2:ListEnvironments",
            "m2:ListDeployments"
         ],
         "Resource":"*"
      }
   ]
}

//Example of an endpoint policy where access is denied to all the
//AWS Mainframe Modernization CREATE actions for all principals on all resources
{"Statement": [
      {"Principal": "*",
         "Effect": "Deny",
         "Action": [
            "m2:Create*"
         ],
         "Resource":"*"
      }
   ]
}
```

AWS Mainframe Modernization のモニタリング

モニタリングは、 AWS Mainframe Modernization およびその他の AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。AWS には、 AWS Mainframe Modernization を監視し、問題が発生した場合に報告し、必要に応じて自動アクションを実行するための以下のモニタリングツールが用意されています。

- Amazon CloudWatch は、 AWS リソースと AWS で実行しているアプリケーションをリアルタイムでモニタリングします。メトリクスを収集および追跡し、カスタマイズされたダッシュボードを作成し、指定されたメトリックが指定したしきい値に達したときに通知またはアクションを実行するアラームを設定できます。例えば、 で Amazon EC2 インスタンスの CPU 使用率やその他のメトリクス CloudWatch を追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- Amazon CloudWatch Logs を使用すると、Amazon EC2 インスタンスやその他のソースからのログファイルをモニタリング、保存 CloudTrail、およびアクセスできます。 CloudWatch Logs はログファイル内の情報をモニタリングし、特定のしきい値に達したときに通知できます。高い耐久性を備えたストレージにログデータをアーカイブすることもできます。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- AWS CloudTrail は、 AWS アカウントによって、またはアカウントに代わって行われた API コールおよび関連イベントを取得し、指定した Amazon S3 バケットにログファイルを配信します。を呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、および呼び出し日時を特定できます。詳細については、『AWS CloudTrail ユーザーガイド』を参照してください。

Amazon による AWS Mainframe Modernization のモニタリング CloudWatch

を使用して AWS Mainframe Modernization をモニタリングすることで CloudWatch、raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに加工することができます。これらの統計は 15 か月間保持されるため、履歴情報にアクセスし、ウェブアプリケーションまたはサービスの動作をより的確に把握できます。また、特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

次の表に、 AWS Mainframe Modernization のメトリクスとディメンションを示します。これらのメトリクスの名前空間は `AWS/M2` です。

ランタイム環境メトリクス

メトリクス	説明
CPUUtilization	環境内のインスタンスの CPU 使用率。 ディメンション: environmentId 単位: パーセント 有効な統計: Average、Minimum、Maximum
InboundNetworkThroughput	環境内のインスタンスのインバウンドネットワークスループット。 ディメンション: environmentId 単位: バイト/秒 有効な統計: Average、Minimum、Maximum
MemoryUtilization	環境内のインスタンスのメモリ使用率。 ディメンション: environmentId 単位: パーセント 有効な統計: Average、Minimum、Maximum
OutboundNetworkThroughput	環境内のインスタンスのアウトバウンドネットワークスループット。 ディメンション: environmentId 単位: バイト/秒 有効な統計: Average、Minimum、Maximum

アプリケーションメトリクス

メトリクス	説明
BatchJobCompletedCount	時間間隔の間に完了したジョブの数。 このメトリクスは、Micro Focus および AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Sum
BatchJobFailedCount	時間間隔の間に失敗したジョブの数。 このメトリクスは、Micro Focus および AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Sum
JvmMemoryFree	Java 仮想マシンが現在使用していない使用可能なメモリ量。 このメトリクスは Blu Age AWS ランタイムエンジンでのみ使用できます。 AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: バイト 有効な統計: Average、Minimum、Maximum

メトリクス	説明
JvmMemoryMax	Java 仮想マシンに許可されるメモリの最大量。 このメトリクスは Blu Age AWS ランタイムエンジンでのみ使用できます。 AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: バイト 有効な統計: Average、Minimum、Maximum
JvmMemoryUsed	Java 仮想マシンがアクティブに使用しているメモリ量。 このメトリクスは Blu Age AWS ランタイムエンジンでのみ使用できます。 AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: バイト 有効な統計: Average、Minimum、Maximum
ProcessesActiveCount	リクエストを処理しているアクティブな同時サービス実行プロセスの数。 このメトリクスは Micro Focus ランタイムエンジンにのみ使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Sum

メトリクス	説明
SessionCount	アプリケーションの HTTP セッションの数。 このメトリクスは Blu Age AWS ランタイムエンジンでのみ使用できます。 AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Average、Minimum、Maximum
SharedMemoryFree	トランザクションとジョブの実行に必要なすべての情報をエンタープライズサーバーが保存するのに使用できるメモリ。 このメトリクスは Micro Focus ランタイムエンジンにのみ使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Average、Minimum、Maximum
ThreadActiveCount	リクエストを処理しているエンジンスレッドの数。 このメトリクスは Blu Age AWS ランタイムエンジンでのみ使用できます。 AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Average、Minimum、Maximum

メトリクス	説明
TransactionCompletedCount	時間間隔の間にコミットされたトランザクション数。 このメトリクスは、Micro Focus および AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Sum
TransactionFailedCount	時間間隔の間に失敗したトランザクションの数。 このメトリクスは、Micro Focus および AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: カウント 有効な統計: Sum
TransactionResponseTime	ユーザーがリクエストを送信した時間から、リクエストが完了したことをアプリケーションが通知するまでの時間。 このメトリクスは、Micro Focus および AWS Blu Age 3.7.0 以降のリリースで使用できます。 ディメンション: applicationId 単位: ミリ秒 有効な統計: Average、Minimum、Maximum

ディメンション

ディメンション	説明
applicationId	このディメンションは、ID で特定したアプリケーションにメトリクスをフィルタリングします。
environmentId	このディメンションは、ID で特定した環境にメトリクスをフィルタリングします。

AWS CloudTrail を使用した AWS Mainframe Modernization API 呼び出しのロギング

AWS Mainframe Modernization は、AWS Mainframe Modernization のユーザー、ロール、または AWS サービスによって実行されたアクションを記録するサービスである AWS CloudTrail と統合されています。CloudTrail は、AWS Mainframe Modernization のすべての API 呼び出しをイベントとしてキャプチャします。キャプチャされた呼び出しには、AWS Mainframe Modernization コンソールからの呼び出しと、AWS Mainframe Modernization API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、AWS Mainframe Modernization のイベントを含む、CloudTrail イベントの Amazon S3 バケットへの継続的な配信を有効にすることができます 追跡を設定しない場合でも、CloudTrail コンソールの [Event history] (イベント履歴) で最新のイベントを表示できます。CloudTrail が収集した情報を使用して、AWS Mainframe Modernization に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時、および追加の詳細情報を確認できます。

CloudTrail の詳細については、AWS CloudTrail ユーザーガイドを参照してください。

CloudTrail のAWS Mainframe Modernization 情報

AWS アカウントを作成すると、そのアカウントに対して CloudTrail が有効になります。AWS Mainframe Modernization でアクティビティが発生すると、そのアクティビティは [イベント履歴] の他の AWS のサービスイベントとともに CloudTrail イベントに記録されます。最近のイベントは、AWSアカウントで表示、検索、ダウンロードできます。詳細については、「Viewing Events with CloudTrail Event History」(CloudTrail イベント履歴でのイベントの表示) を参照してください。

AWS Mainframe Modernization のイベントを含む、AWS アカウントのイベントの継続的な記録については、証跡を作成します。追跡により、CloudTrailはログファイルをSimple Storage Service (Amazon S3) バケットに配信できます。デフォルトでは、コンソールで作成した証跡がすべての AWS リージョンに適用されます。証跡は、AWS パーティションのすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS サービスを設定できます。詳細については、次を参照してください:

- 「証跡作成の概要」
- CloudTrail がサポートされているサービスと統合
- CloudTrail の Amazon SNS 通知の設定
- CloudTrail ログファイルの複数のリージョンからの受け取り
- 複数のアカウントから CloudTrail ログファイルを受け取る

すべての AWS Mainframe Moderanization アクションは CloudTrail により記録され、「AWS Mainframe Modernization API リファレンス」で文書化されます。例えば、`CreateApplication`、`CreateEnvironment`、`CreateDeployment` の各アクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するために役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか
- リクエストが、ロールとフェデレーションユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか。
- リクエストが、別の AWS サービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity エレメント」を参照してください。

AWS Mainframe Modernization のログファイルエントリについて

「証跡」は、指定した Simple Storage Service (Amazon S3) バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、パブ

リック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下の例は、CreateApplication アクションを示す CloudTrail ログエントリです。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAII6WZTHGYAEXAMPLE",
        "arn": "arn:aws:sts::444455556666:assumed-role/Admin/Mary_Major",
        "accountId": "444455556666",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAII6WZTHGYAEXAMPLE",
                "arn": "arn:aws:iam::444455556666:role/Admin",
                "accountId": "444455556666",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-06-01T20:38:22Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-06-01T20:40:39Z",
    "eventSource": "m2.amazonaws.com",
    "eventName": "CreateApplication",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.196.65",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101
 Firefox/91.0",
    "requestParameters": {
        "clientToken": "1abc23de-f45g-6789-h01i-jkl2m3456789",
        "name": "MyApp",
        "description": "",
        "engineType": "microfocus",
        "definition": {
            "content": "{}"
        },
        "tags": {}
```

```
    },
    "responseElements": {
        "applicationVersion": 1,
        "Access-Control-Expose-Headers": "x-amzn-RequestId,x-amzn-ErrorType,x-amzn-
ErrorMessage,Date",
        "applicationArn": "arn:aws:m2:us-east-1:444455556666:app/
lsfhmwhw7fffrosff2lncwqcua",
        "applicationId": "lsfhmwhw7fffrosff2lncwqcua"
    },
    "requestID": "36982d38-fcde-4bfe-a89a-7bd78d43c926",
    "eventID": "d7f0fc36-46ae-4157-9a79-c79f385fda98",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "444455556666",
    "eventCategory": "Management"
}
```

トラブルシューティング

このセクションの情報は、AWS Blu Age エンジンと Micro Focus エンジンの両方を使用する AWS Mainframe Modernization アプリケーションおよびランタイム環境でよく発生するエラーのトラブルシューティングに役立ちます。

トピック

エラー: データセット名のロックが解除されるのを待っている間にタイムアウトしました

- エンジン: AWS Blu Age
- コンポーネント: Blusam

Blu Age エンジンを使用しており、高可用性パターンの環境で実行されている AWS Mainframe Modernization アプリケーションの Amazon AWS CloudWatch ログにこのエラーが表示された場合は、別のアプリケーションが共有データセットのロックを保持していることを示しています。通常、この状況は、他のアプリケーションがクラッシュまたは障害を起こしてロックが解除されない場合に発生します。

このエラーが発生する仕組み

アプリケーション `example-app-1` が書き込みオペレーションのためにレコード `example-record-1` をロックしようとします。このオペレーションは、`example-record-1` を所有するデータセット `example-dataset-1` のロックと、`example-record-1` 自体のロックの両方を作成します。次に、別のアプリケーション `example-app-2` が同じレコード `example-record-1` をロックしようとします。データセットとレコードは既にロックされているため、`example-app-2` はロックが解除されるまで待ちます。`example-app-1` がクラッシュすると、データセット `example-dataset-1` に保持されていたロックが残っているため、`example-app-2` は書き込みをキャンセルし、タイムアウト例外を発生させます。このデッドロック状態では、すべてのアプリケーションが `example-dataset-1` に到達できなくなります。

これが自分の状況かどうかは、どうすればわかりますか。

障害が発生したアプリケーションを探して、そのアプリケーションがエラーメッセージに記載されているのと同じデータセットを使用しているかどうかを確認してください。アプリケーションが高可用性パターンのランタイム環境で実行されているかどうかを確認します。タイムアウト例外を発生させたアプリケーションは処理を続行できず、`Failed` ステータスが表示されます。

できること。

この状況をすぐに解決するには、ロックを強制的に解除します。今後、同様の状況が発生しないようにするには、Blusam 自動修復メカニズムを制御する 2 つのパラメータを設定できます。

ロックを強制的に解除する

Blusam ロックマネージャーは、Amazon ElastiCache for Redis を使用してアプリケーション間で共有ロックを提供します。でロックを解除するには ElastiCache、Redis CLI ユーティリティを使用します。個々のレコードロックは削除できません。所有しているデータセットからすべてのロックを解除する必要があります。以下のステップを実行します。

1. 次のコマンド ElastiCache を使用して に接続します。

   ```
   redis-cli -h hostname -p port
   ```

   の詳細は、https://console.aws.amazon.com/elasticache/ ElastiCache で ElastiCache コンソールで確認できます。
2. パスワードを入力します。
3. 実行するコマンドを次のように入力します。

Command	目的
KEYS *	既存のキーをすべて取得します。
KEYS *YOUR_DATASET_NAME	データセットのロックキーを取得します。
DEL THE_RETURNED_KEY	データセットロックを削除します。
FLUSHDB	Redis 全体をクリーンアップします。

Command	目的
	⚠ Warning Redis キャッシュ内のデータはすべて失われます。Redis が http セッションの処理など、他の目的で使用されている場合は、FLUSHDB を使用しないほうがよいかもしれません。

Blusam 自動修復メカニズムを設定する

Blusam ロックマネージャー には、データセットやレコードのデッドロックを防ぐための自動修復メカニズムが含まれています。アプリケーション定義 (`application-main.yml`) で次のパラメータを調整して、自動修復メカニズムを設定できます。

- `locksDeadTime`: アプリケーションがロックを保持できる最大時間を示します。この時間が経過すると、ロックは期限切れと宣言され、直ちに解除されます。`locksDeadTime` 値はミリ秒単位で、デフォルト値は 1000 です。
- `locksCheck`: ロックをチェックするための Blusam ロックマネージャー戦略を定義します。のすべての Blusam ロック ElastiCache にはタイムスタンプが付けられ、有効期限があります。`locksCheck` パラメータ値によって、期限切れのロックが削除されるかどうかが決まります。
  - `off`: チェックは常に実行されません。デッドロックが発生する可能性があります。(非推奨)
  - `reboot`: チェックは、AWS Mainframe Modernization ランタイム環境で実行されている AWS Mainframe Modernization インスタンスが起動または再起動されるときに実行されます。期限切れのロックはすべて直ちに解除されます。(デフォルト)
  - `timeout`: チェックは、AWS Mainframe Modernization ランタイム環境で実行されている AWS Mainframe Modernization アプリケーションインスタンスが起動または再起動されるとき、またはデータセットをロックしようとしたときにタイムアウトになったときに、チェックが実行されます。期限切れのロックは直ちに解除されます。

AWS Blu Age アプリケーションのアプリケーション定義の詳細については、「AWS Blu Age アプリケーション定義のサンプル」を参照してください。

Blusam ロックマネージャー

高可用性パターンを使用する AWS Mainframe Modernization ランタイム環境のコンテキストでは、AWS Blu Age アプリケーションが複数回デプロイされる可能性があります。Blusam データセットを処理するアプリケーションでは、同時アクセスの問題が発生する可能性があります。Blusam ロックマネージャーは、 を使用してアプリケーション間で共有ロックを提供することで、データの整合性を確保し、レコードとデータセットへの読み取り/書き込みアクセスを管理します ElastiCache。このメカニズムにより、複数のアプリケーションが同時にレコードを読み取ることができ、レコードを書き込むアプリケーションは一度に 1 つだけになります。

書き込みロック

特定のレコードを更新または削除するには、アプリケーションはまずレコードを所有するデータセットをロックし、次にレコード自体をロックする必要があります。レコードがロックされると、データセットのロックが解除され、同じデータセットの他のレコードが使用できるようになります。更新または削除操作が完了すると、保持されていたレコードロックは解除されます。レコードを更新できるのは一度に 1 つのアプリケーションのみで、定義されたアプリケーションポリシーで解除を待つことが許可されている場合、ロックが解除されるまで他のアプリケーションは読み取りまたは書き込みができなくなります。

読み取りロック

レコードまたはデータセットに書き込みロックがかかっていない限り、複数のアプリケーションが同じレコードを同時に読み取ることができます。書き込み操作のためにレコードをロックするには、すべての読み取りロックを解除する必要があります。

> **Note**
>
> Blusam ロックマネージャーは、同じロックメカニズムを使用して、特定のアプリケーション内の複数のスレッドからのアクセスを処理します。

アプリケーションの URL にはアクセスできません。

- エンジン: AWS Blu Age および Micro Focus
- コンポーネント: アプリケーション

AWS Mainframe Modernization ランタイム環境に作成してデプロイした、実行中の AWS Mainframe Modernization アプリケーションの URL にアクセスできない場合は、ランタイム環境に関連付けたセキュリティグループでインバウンドルールを設定する必要がある場合があります。

このエラーが発生する仕組み

ランタイム環境を作成する場合、このタイプのアクセスを許可するには、デフォルトのセキュリティグループを含め、提供するセキュリティグループに、VPC の外部からデプロイされたアプリケーションへのトラフィックを許可するインバウンドルールを設定する必要があります。

これが自分の状況かどうかは、どうすればわかりますか。

アプリケーションは正常に起動し、正常に動作しているが、URL を使用して接続できない。

できること。

ランタイム環境に関連付けられている Amazon VPC セキュリティグループが、適切なアプリケーションポート上の環境へのトラフィックを許可しているかどうかを確認します。これらのオプションのセキュリティグループルールを追加するには、以下のステップを実行します。

1. https://console.aws.amazon.com/m2/ にある AWS Mainframe Modernization コンソールを開きます。
2. 左側のナビゲーションで [環境] を選択します。
3. 接続するアプリケーションをホストするランタイム環境を選択します。
4. [設定] を選択します。
5. [ネットワークとセキュリティ] で、[セキュリティグループ] を選択します。リンクは、Amazon VPC コンソールにセキュリティグループの詳細を開きます。
6. 必要に応じて [インバウンドのルールの編集] を選択し、以下のルールがまだ存在しない場合は追加します。

   タイプ

   カスタム TCP

   ポート

   8196 またはアプリケーション定義で指定されたリスナープロパティと一致するポート。詳細については、「ステップ 2: アプリケーション定義を作成する」を参照してください。

ソース

アプリケーションの呼び出し元の IP アドレス。ドロップダウンから [myIP] を選択できます。それでもタイムアウトの問題が解決しない場合は、[Anywhere IPV4] または [Anywhere IPV6] を選択してみてください。セキュリティグループにインバウンドルールを追加したら、必ずアプリケーションを停止して再起動してください。

詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループルールの操作」を参照してください。

AWS Blu Insights がコンソールから開かない

- エンジン: AWS Blu Age
- コンポーネント: Blu Insights

AWS Mainframe Modernization コンソールから Blu Insights にアクセスしようとしても、コンソールが開かず、新しいタブがすぐに閉じます。

このエラーが発生する仕組み

Blu Insights へのアクセスに使用しているロールに十分なアクセス許可がありません。

できること。

IAM ポリシーをロールにアタッチして、そのロールが Blu Insights にアクセスできるようにします。ポリシーには少なくとも以下のアクセス許可が含まれていることを確認してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "m2:GetSignedBluinsightsUrl"
            ],
            "Resource": "arn:aws:m2:region:account:*"
            }
    ]
```

```
}
```

region と account を正しい AWS リージョン および AWS アカウント に置き換えてください。

AWS Mainframe Modernization ユーザーガイドのドキュメント履歴

以下の表は、AWS Mainframe Modernization のリリースドキュメントの内容をまとめたものです。

変更	説明	日付
Micro Focus のマネージドランタイムのチュートリアルを更新	このチュートリアルでは、Micro Focus ランタイムエンジンを使用して、 AWS Mainframe Modernization マネージドランタイム環境で CardDemo サンプルアプリケーションをデプロイして実行する方法を示します。	2024 年 2 月 5 日
AWS Blu Age ランタイムおよびモダナイゼーションツールバージョン 3.9.0 のリリースノート。	AWS Blu Age ランタイムおよびモダナイゼーションツールのこのリリースでは、高可用性アーキテクチャでパフォーマンスを向上させるための製品全体の複数のトランスバーサル機能強化と、次のレベルにジョブの実行を引き上げる新機能に焦点を当てています。	2023 年 12 月 18 日
メインフレームと AWS 間のファイル転送	ソースメインフレームから AWS にファイルを転送する新機能がリリースされました。	2023 年 11 月 27 日
アプリケーションのトランザクションの管理	AWS Mainframe Modernization 用アプリケーションのトランザクションを表示および編集する新機能がリリースされました。	2023 年 10 月 16 日

AWS Blu Age ランタイムとモダナイゼーションツールバージョン 3.6.0 のリリースノート。	AWS Blu Age ランタイムとモダナイゼーションツールのこのリリースでは、zOS と AS400 のレガシー移行の両方に関する新機能が提供されます。主に CICS サポートメカニズムの拡張、JCL 機能の補完、同時および大量の機能のパフォーマンスの最適化、multi-data-source 機能の追加を目的としています。	2023 年 8 月 4 日
アプリケーションが停止したときに、新しいバージョンのアプリケーションをデプロイできるようになりました。	以前は、アプリケーションの新しいバージョンをデプロイするには、デプロイ済みのバージョンを削除する必要がありました。デプロイ済みのバージョンを停止するだけで、新しいバージョンをデプロイできるようになりました。	2023 年 7 月 26 日
Amazon EC2 のデプロイが容易になるようにパッケージ化された AWS Blu Age ランタイム	スタック全体を構成してお客様の Amazon EC2 インスタンスにデプロイするため、ご使用の AWS アカウント で AWS Mainframe Modernization を AWS Blu Age ランタイムとともにより柔軟に利用できるようになりました。	2023 年 7 月 6 日
AWSAWS Blu Age Blu Insights へのシングルサインオン。	AWSAWS Blu Age Blu Insights はシングルサインオンで AWS Management Console からご利用いただけます。	2023 年 3 月 31 日

GA リリース	AWS Mainframe Modernization ユーザーガイドの GA リリース。	2022 年 6 月 8 日
初回リリース	AWS Mainframe Modernization ユーザーガイドの初回リリース (プレビュー公開)。	2021 年 11 月 30 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。