# Table of Contents

What is AWS Mainframe Modernization? ......................................................... 1
Features of AWS Mainframe Modernization .................................................. 1
Patterns ............................................................................................................. 2
How to get started with AWS Mainframe Modernization .................................. 2
Related services .............................................................................................. 3
Accessing AWS Mainframe Modernization .................................................... 3
Are you a first-time AWS Mainframe Modernization user? ............................... 3
Pricing .............................................................................................................. 3

Setting up AWS Mainframe Modernization ................................................. 4
Sign up for an AWS account ........................................................................... 4
Create an administrative user ......................................................................... 4

Getting started ............................................................................................... 6

Tutorial: Managed Runtime for AWS Blu Age .............................................. 6
Prerequisites ..................................................................................................... 6
Step 1: Upload the demo application ............................................................... 7
Step 2: Create the application definition ....................................................... 7
Step 3: Create a runtime environment ............................................................ 8
Step 4: Create an application .......................................................................... 10
Step 5: Deploy an application ......................................................................... 12
Step 6: Start an application ............................................................................ 12
Step 7: Access the application ........................................................................ 13
Step 8: Test the application ............................................................................ 13
Clean up resources ......................................................................................... 14
Next steps ....................................................................................................... 14

Tutorial: Managed runtime for Micro Focus .................................................. 15
Prerequisites ..................................................................................................... 15
Step 1: Create a database ................................................................................ 15
Step 2: Allow AWS Mainframe Modernization to retrieve the database admin credentials ............................................. 16
Step 3: Create a runtime environment ............................................................ 17
Step 4: Create an application .......................................................................... 20
Step 5: Deploy an application ......................................................................... 23
Step 6: Import data sets .................................................................................. 24
Step 7: Start an application ............................................................................ 28
Step 8: Connect to the BankDemo sample application ................................... 28
Clean up resources ......................................................................................... 30
Next steps ....................................................................................................... 31

Modernization approach .................................................................................. 32
Assess phase .................................................................................................... 32
Mobilize phase ............................................................................................... 32
Migrate and modernize phase ......................................................................... 32
Operate and optimize phase .......................................................................... 33

AWS Blu Age Refactoring .............................................................................. 34

AWS Blu Age Runtime Concepts ................................................................... 34
High Level Architecture .................................................................................. 35
Modernized Application Structure ................................................................. 37
Data Simplifier ............................................................................................... 68

AWS Blu Age Runtime (on Amazon EC2) Setup ............................................ 72
AWS Blu Age Runtime (on Amazon EC2) Prerequisites .................................. 73
AWS Blu Age Runtime (on Amazon EC2) onboarding ..................................... 73
Provisioning AWS Blu Age Components ...................................................... 76
Deploy a modernized application .................................................................... 85
Upgrade the AWS Blu Age Runtime (on Amazon EC2) version ....................... 89
Setting up licensed dependencies in AWS Blu Age Runtime (on Amazon EC2) .................................................................................. 91

AWS Blu Age Runtime (on Amazon EC2) Configuration .............................. 93
Set up a build and CI/CD pipeline

Prerequisites .......................................................... 181
Overview of tutorial .................................................. 182
Set up your CI/CD environment ..................................... 182
Set up your CI/CD build and pipeline .............................. 189
Clean up resources ..................................................... 195

AWS Blu Age Release Notes ........................................ 196
AWS Blu Age Release Notes 3.8.0 .................................. 196
AWS Blu Age - Runtime Release 3.8.0 ............................ 196
AWS Blu Age - Modernization tools Release 3.8.0 ............... 198
AWS Blu Age Release Notes 3.7.0 .................................. 200
AWS Blu Age - Runtime Release 3.7.0 ............................ 200
AWS Blu Age - Modernization tools Release 3.7.0 ............... 202
AWS Blu Age Release Notes 3.6.0 .................................. 203
AWS Blu Age - Runtime Release 3.6.0 ............................ 204
AWS Blu Age - Modernization tools Release 3.6.0 ............... 206
AWS Blu Age Release Notes 3.5.0 .................................. 208
AWS Blu Age - Runtime Release 3.5.0 ............................ 208
AWS Blu Age - Modernization tools Release 3.5.0 ............... 210

Micro Focus Replatforming ......................................... 212
Micro Focus Runtime (on Amazon EC2) Setup ....................... 212
Prerequisites .......................................................... 212
Create the Amazon VPC Endpoint for Amazon S3 ................. 213
Request the Allowlist update for the Account ..................... 214
Creating the AWS Identity and Access Management role ........ 215
Grant License Manager the required permissions ................... 220
Subscribe to the Amazon Machine Images ......................... 220
Launch an AWS Mainframe Modernization Micro Focus Instance 223
Subnet or VPC with no Internet Access ............................ 226
Troubleshooting license issues ...................................... 230

Tutorial: Set up AppStream 2.0 for Enterprise Analyzer and Enterprise Developer .......... 232
Prerequisites .......................................................... 233
Step 1: Get the AppStream 2.0 images ............................ 233
Step 2: Create the stack using the AWS CloudFormation template .... 233
Step 3: Create a user in AppStream 2.0 ............................ 235
Step 4: Log in to AppStream 2.0 .................................... 236
Step 5: Verify buckets in Amazon S3 (optional) .................... 237
Next steps .................................................................. 238
Clean up resources ..................................................... 238

Set up Enterprise Analyzer ........................................... 238
Image contents .......................................................................................................................... 239
Prerequisites ............................................................................................................................. 239
Step 1: Setup ............................................................................................................................. 240
Step 2: Create the Amazon S3 based virtual folder on Windows .............................................. 240
Step 3: Create an ODBC source for the Amazon RDS instance ................................................ 241
Subsequent sessions .................................................................................................................. 242
Troubleshooting workspace connection ................................................................................... 242
Clean up resources ................................................................................................................... 247
Set up Enterprise Developer ................................................................................................. 248
Image contents .......................................................................................................................... 248
Prerequisites ............................................................................................................................. 249
Step 1: Setup by individual Enterprise Developer users ............................................................... 249
Step 2: Create the Amazon S3-based virtual folder on Windows (optional) .............................. 249
Step 3: Clone the repository ..................................................................................................... 250
Subsequent sessions .................................................................................................................. 251
Clean up resources ................................................................................................................... 251
Set up AppStream 2.0 Automation ......................................................................................... 251
Set up automation at session start ........................................................................................... 252
Set up automation at session end ............................................................................................. 252
View Data Sets as Tables .......................................................................................................... 252
Prerequisites ............................................................................................................................. 253
Step 1: Set up ODBC Connection to Micro Focus datastore (Amazon RDS database) ............. 253
Step 2: Create the MFDBFH.cfg file ........................................................................................ 255
Step 3: Create a structure (STR) file for your copybook layout .............................................. 256
Step 4: Create a database view using the structure (STR) file .................................................. 257
Step 5: View Micro Focus data sets as tables and columns ...................................................... 258
Use templates with Enterprise Developer .............................................................................. 258
Use Case 1 - Using the COBOL Project Template containing source components .................. 259
Use Case 2 - Using the COBOL Project Template without source components ...................... 262
Use Case 3 - Using the pre-defined COBOL project linking to the source folders ................... 264
Using the Region Definition JSON Template ........................................................................ 267
Tutorial: Set up the build for the BankDemo sample .............................................................. 270
Prerequisites ............................................................................................................................. 271
Step 1: Create Amazon S3 buckets .......................................................................................... 271
Step 2: Create the build spec file ............................................................................................ 271
Step 3: Upload the source files ............................................................................................... 271
Step 4: Create IAM policies .................................................................................................... 272
Step 5: Create an IAM role ........................................................................................................ 273
Step 6: Attach the IAM policies to the IAM role ..................................................................... 274
Step 7: Create the CodeBuild project ...................................................................................... 275
Step 8: Start the build ............................................................................................................. 275
Step 9: Download output artifacts ........................................................................................... 276
Clean up resources ................................................................................................................... 276
Tutorial: Setting up a CI/CD pipeline for use with Micro Focus Enterprise Developer ................ 276
Prerequisites ............................................................................................................................. 277
Create CI/CD pipeline basic infrastructure .............................................................................. 278
Create AWS CodeCommit repository and CI/CD pipeline ....................................................... 280
Enterprise Developer AppStream 2.0 Creation ........................................................................ 284
Enterprise Developer Setup and Test ....................................................................................... 284
Exercise 1: Enhance Loan Calculation in BANKDEMO Application ........................................ 287
Exercise 2: Extract loan calculation in BankDemo application ................................................. 290
Clean up resources ................................................................................................................... 292
Batch Utilities ............................................................................................................................ 293
Binary Location .......................................................................................................................... 293
M2SFTP Batch Utility .............................................................................................................. 293
M2WAIT Batch Utility .............................................................................................................. 298
TXT2PDF Batch Utility ............................................................................................................ 299
Data set .......................................................................................................................... 357
Environment ................................................................................................................ 357
Mainframe modernization ............................................................................................. 357
Migration journey .......................................................................................................... 358
Mount point .................................................................................................................. 358
Automated Refactoring ............................................................................................... 358
Replatforming ................................................................................................................ 358
Resource ....................................................................................................................... 358
Runtime engine ............................................................................................................. 358
Security ......................................................................................................................... 359
  Data protection ........................................................................................................... 359
    Data that AWS Mainframe Modernization collects .............................................. 360
    Data encryption at rest for AWS Mainframe Modernization service ............... 361
    How AWS Mainframe Modernization uses grants in AWS KMS ....................... 362
    Create a customer managed key .......................................................................... 363
    Specifying a customer managed key for AWS Mainframe Modernization ....... 364
    AWS Mainframe Modernization encryption context ......................................... 365
    Monitoring your encryption keys ......................................................................... 366
    Learn more ............................................................................................................. 376
    Encryption in transit ............................................................................................... 377
Identity and Access Management ............................................................................... 377
  Audience .................................................................................................................... 377
  Authenticating with identities .................................................................................. 378
  Managing access using policies .............................................................................. 380
  How AWS Mainframe Modernization works with IAM ...................................... 381
  Identity-based policy examples .............................................................................. 389
  Troubleshooting ....................................................................................................... 391
  Using service-linked roles ....................................................................................... 392
Compliance validation ................................................................................................ 395
Resilience ...................................................................................................................... 395
Infrastructure security ................................................................................................. 396
Tutorial: AWS Blu Age Application Security in AWS Mainframe Modernization ... 396
  Amazon Cognito user pools .................................................................................... 396
  Third-party identity providers .............................................................................. 396
  Example setup .......................................................................................................... 397
  Clean up resources ................................................................................................. 405
AWS PrivateLink ......................................................................................................... 406
  Considerations ........................................................................................................ 406
  Create an interface endpoint ............................................................................... 406
  Create an endpoint policy ..................................................................................... 406
Monitoring .................................................................................................................... 408
  Monitoring with CloudWatch ............................................................................... 408
    Runtime Environment Metrics .......................................................................... 408
    Application Metrics ............................................................................................ 409
    Dimensions .......................................................................................................... 412
CloudTrail logs ............................................................................................................. 412
  AWS Mainframe Modernization information in CloudTrail ................................ 413
  Understanding AWS Mainframe Modernization log file entries ...................... 413
Troubleshooting ........................................................................................................ 415
  Error: Time out while waiting for dataset name to be unlocked ...................... 415
    How this error occurs ......................................................................................... 415
    How do you know if this is your situation? ....................................................... 415
    What can you do? ............................................................................................... 415
    Force the lock to release .................................................................................... 416
    Configure the Blusam auto repairing mechanism ........................................... 416
    Blusam locks manager ....................................................................................... 417
  Cannot access an application's URL ...................................................................... 417
What is AWS Mainframe Modernization?

AWS Mainframe Modernization helps you modernize your mainframe applications to AWS managed runtime environments. It provides tools and resources to help you plan and implement migration and modernization. You can analyze your existing mainframe applications, develop or update them using COBOL or PL/I, and implement an automated pipeline for continuous integration and continuous delivery (CI/CD) of the applications. You can choose between automated refactoring and replatforming patterns, depending on your clients' needs. If you are a consultant helping a client migrate their mainframe workloads, you can use AWS Mainframe Modernization tools for all phases of the migration and modernization journey, from initial planning to post-migration cloud operations.

You can use AWS Mainframe Modernization to help you efficiently create and manage the runtime environment on AWS for your mainframe applications, as well as to manage and monitor your modernized applications.

Topics

- Features of AWS Mainframe Modernization (p. 1)
- Patterns (p. 2)
- How to get started with AWS Mainframe Modernization (p. 2)
- Related services (p. 3)
- Accessing AWS Mainframe Modernization (p. 3)
- Are you a first-time AWS Mainframe Modernization user? (p. 3)
- Pricing (p. 3)

Note

Have you engaged with AWS Mainframe Migration Competency Partners or AWS Professional Services for your mainframe modernization project? If not, we highly recommend that you engage experts for your project.

- AWS Mainframe Modernization Competency Partners
- AWS Professional Services

The features and use cases of AWS Mainframe Modernization support an evolutionary modernization approach, which provides short-term wins by improving agility and plenty of opportunities to optimize and innovate later on. For more information, see Modernization approach (p. 32).

Features of AWS Mainframe Modernization

AWS Mainframe Modernization features support the following use cases:

- Assess: AWS Mainframe Modernization's assessment capability can help you assess, scope, and plan a migration and modernization project.
- Refactor: powered by AWS Blu Age, you can use refactoring to convert legacy application programming languages, to create macroservices or microservices, and to modernize user interfaces (UIs) and application software stacks.
AWS Blu Insights is now available from the AWS Management Console through single sign-on. You do not have to manage separate AWS Blu Insights credentials any longer. You can access both the AWS AWS Blu Age Codebase and Transformation Center features directly from the AWS Management Console.

- Replatform: powered by the Micro Focus Enterprise solution, you can port the application where much of the application source code is recompiled without changes.
- Developer IDE: AWS Mainframe Modernization offers an on-demand integrated development environment (IDE) so developers can write code quicker with smart editing and debugging, instant code compilation, and unit testing.
- Managed runtime: The AWS Mainframe Modernization managed execution environment continually monitors your clusters to keep enterprise workloads running with self-healing compute and automated scaling.
- Continuous integration and delivery (CI/CD): AWS Mainframe Modernization’s CI/CD feature helps application development teams deliver code changes more frequently and reliably, which accelerates migration speed, increases quality, and helps reduce time-to-market for releasing new business functions.
- Integrations with other AWS services: AWS Mainframe Modernization supports AWS CloudFormation, AWS PrivateLink, and AWS Key Management Service for repeatable deployment and greater security and compliance.
- Expanded availability: AWS Mainframe Modernization is now available in US East (Ohio), US West (N. California), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Tokyo), Europe (London), and Europe (Paris).

For more information on AWS Mainframe Modernization features, see http://aws.amazon.com/mainframe-modernization/features/.

Patterns

The Automated Refactoring pattern, powered by AWS Blu Age, is focused on accelerating modernization by converting the complete legacy application stack and its data layer into a modern Java-based application while preserving functional equivalence. During this automated transformation, it creates a multi-tier application with an Angular-based front-end, an API-enabled Java backend and a data layer accessing modern data stores. The refactoring process provides equivalent functionality to the legacy stack to increase project automation resulting in speed, quality, and lower cost for achieving business benefits quicker. For more information, see AWS Mainframe Modernization Automated Refactor.

The Replatforming pattern, powered by Micro Focus Enterprise suite, is focused on preserving the application language, code, and artifacts in order to minimize the impact to the application assets and teams. It helps customers maintain the application knowledge and skills. While the application changes are limited, this pattern also facilitates a modernization of the infrastructure and the processes. The infrastructure is changed to a modern cloud-based managed service while the processes are changed to follow best practices for application development and IT operations. For more information, see AWS Mainframe Modernization Replatform.

How to get started with AWS Mainframe Modernization

Try it! We offer tutorials and sample applications to help you get a sense of what AWS Mainframe Modernization offers. Choose either the Tutorial: Managed Runtime for AWS Blu Age (p. 6) or the Tutorial: Managed runtime for Micro Focus (p. 15) for a complete, step-by-step tutorial.
If you are interested in automated refactoring, check out the AWS Blu Age tools at BluInsights. You can also set up AppStream 2.0 to access the AWS Blu Age Developer IDE, or the Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer tools.

The tutorials and sample applications only give you a sense of what AWS Mainframe Modernization provides. When you are ready to start a modernization project, see Modernization approach (p. 32) for details about the stages and tasks of a modernization project.

Related services

In addition to Blu Insights for automated refactoring, you can use the following AWS services with AWS Mainframe Modernization.

- Amazon RDS for hosting your migrated databases.
- Amazon S3 for storing application binaries and definition files.
- Amazon FSx or Amazon EFS for storing application data.
- Amazon AppStream for access to the Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer tools.
- AWS CloudFormation for the automated DevOps pipeline that you can use to set up CI/CD for your migrated applications.
- AWS Migration Hub
- AWS DMS for migrating your databases.

Accessing AWS Mainframe Modernization

Currently, you can access AWS Mainframe Modernization through the console at https://console.aws.amazon.com/m2/. For a list of regions where AWS Mainframe Modernization is available, see AWS Mainframe Modernization endpoints and quotas in the Amazon Web Services General Reference.

Are you a first-time AWS Mainframe Modernization user?

If you are a first-time user of AWS Mainframe Modernization, we recommend that you begin by reading the following sections:

- Getting Started (p. 6)
- Setting up (p. 4)

Pricing

AWS Mainframe Modernization charges for the usage of instances supporting the managed runtime environments. In addition, AWS Mainframe Modernization offers some tools without additional charges. You are responsible for fees incurred for other AWS services that you use in connection with AWS Mainframe Modernization. AWS will provide 30 days' notice before any pricing changes take effect for use of AWS Mainframe Modernization. For more information, see Mainframe Modernization with AWS.

With AWS Blu Insights, you pay for Transformation Center usage. For more information, see AWS Mainframe Modernization pricing.
Setting up AWS Mainframe Modernization

Before you can start using AWS Mainframe Modernization you or your administrator need to complete some steps.

Topics
- Sign up for an AWS account (p. 4)
- Create an administrative user (p. 4)

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account
2. Follow the online instructions.
   - Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.
   - When you sign up for an AWS account, an AWS account root user is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to an administrative user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing My Account.

Create an administrative user

After you sign up for an AWS account, create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user
1. Sign in to the AWS Management Console as the account owner by choosing Root user and entering your AWS account email address. On the next page, enter your password.
   - For help signing in by using root user, see Signing in as the root user in the AWS Sign-In User Guide.
2. Turn on multi-factor authentication (MFA) for your root user.
   - For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the IAM User Guide.
Create an administrative user

- For your daily administrative tasks, grant administrative access to an administrative user in AWS IAM Identity Center.

  For instructions, see Getting started in the AWS IAM Identity Center User Guide.

Sign in as the administrative user

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the AWS Sign-In User Guide.
Getting started with AWS Mainframe Modernization

To get started with AWS Mainframe Modernization you can follow tutorials that introduce you to the service and each runtime engine.

**Topics**
- Tutorial: Managed Runtime for AWS Blu Age (p. 6)
- Tutorial: Managed runtime for Micro Focus (p. 15)

If you want to continue learning, you can also follow tutorials on the build tools and CI/CD pipelines.
- Set up a build and CI/CD pipeline (p. 181)
- Tutorial: Setting up the Micro Focus build for the BankDemo sample application (p. 270)
- Tutorial: Setting up a CI/CD pipeline for use with Micro Focus Enterprise Developer (p. 276)

**Tutorial: Managed Runtime for AWS Blu Age**

This tutorial shows how to deploy a AWS Blu Age modernized application into an AWS Mainframe Modernization runtime environment.

**Topics**
- Prerequisites (p. 6)
- Step 1: Upload the demo application (p. 7)
- Step 2: Create the application definition (p. 7)
- Step 3: Create a runtime environment (p. 8)
- Step 4: Create an application (p. 10)
- Step 5: Deploy an application (p. 12)
- Step 6: Start an application (p. 12)
- Step 7: Access the application (p. 13)
- Step 8: Test the application (p. 13)
- Clean up resources (p. 14)
- Next steps (p. 14)

**Prerequisites**

To complete this tutorial, download the demo application archive PlanetsDemo-v1.zip.

The running demo application requires a modern browser for access. Whether you run this browser from your desktop or from an Amazon Elastic Compute Cloud instance, for example, within the VPC, determines your security settings.
Step 1: Upload the demo application

Upload the demo application to an Amazon S3 bucket. Make sure that this bucket is in the same AWS Region where you will deploy the application. The following example shows a bucket named planetsdemo, with a key prefix, or folder, named v1 and an archive named planetsdemo-v1.zip.

![Image of Amazon S3 bucket with planetsdemo-v1.zip]

**Note**
The folder in the bucket is required.

Step 2: Create the application definition

To deploy an application to the managed runtime, you need an AWS Mainframe Modernization application definition. This definition is a JSON file that describes the application location and settings. The following example is such an application definition for the demo application:

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

Change the s3-bucket entry to the name of the bucket where you stored the sample application zip file.

For more information on the application definition, see AWS Blu Age application definition sample (p. 335).
Step 3: Create a runtime environment

To create the AWS Mainframe Modernization runtime environment, perform the following steps:

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where you want to create the environment. This AWS Region must match the Region where you created the S3 bucket in Step 1: Upload the demo application (p. 7).
3. Under Modernize mainframe applications, choose Refactor with Blu Age, and then choose Get started.

4. Under How would you like to start with AWS Mainframe Modernization, choose Deploy and Create runtime environment.

5. In the left navigation, choose Environments, then choose Create environment. On the Specify basic information page, enter a name and description for your environment, and then make sure AWS Blu Age engine is selected. Optionally, you can add tags to the created resource. Then choose Next.
6. On the Specify configurations page, choose Standalone runtime environment.

7. Under Security and network, make the following changes:

   - Choose **Allow applications deployed to this environment to be publicly accessible**. This option assigns a public IP address to the application so that you can access it from your desktop.
   - Choose a VPC. You can use the **Default**.
   - Choose two subnets. Make sure that the subnets allow the assignment of public IP addresses.
   - Choose a security group. You can use the **Default**. Make sure that the security group that you choose allows access from the browser IP address to the port you specified in the listener property of the application definition. For more information, see **Step 2: Create the application definition (p. 7)**.
Step 4: Create an application

1. Navigate to **AWS Mainframe Modernization** in the AWS Management Console.

If you want to access the application from outside the VPC that you chose, make sure that the inbound rules for that VPC are configured properly. For more information, see [Cannot access an application’s URL](#).  

8. Choose **Next**.

9. In **Attach storage - Optional**, leave the default selections and choose **Next**.

10. In **Schedule maintenance**, choose **No preference**, and then choose **Next**.

11. In **Review and create**, review the information, and then choose **Create environment**.

---

If you want to access the application from outside the VPC that you chose, make sure that the inbound rules for that VPC are configured properly. For more information, see Cannot access an application’s URL (p. 417).

8. Choose Next.

9. In Attach storage - Optional, leave the default selections and choose Next.

10. In Schedule maintenance, choose No preference, and then choose Next.

11. In Review and create, review the information, and then choose Create environment.

Step 4: Create an application

1. Navigate to AWS Mainframe Modernization in the AWS Management Console.
2. In the navigation pane, choose Applications, and then choose Create application. On the Specify basic information page, enter a name and description for the application, and make sure that the AWS Blu Age engine is selected. Then choose Next.

3. On the Specify resources and configurations page, copy and paste the updated application definition JSON.

4. In Review and create, review your choices, and then choose Create application.
Step 5: Deploy an application

After you successfully create both the AWS Mainframe Modernization runtime environment and application, and both are in the **Available** state, you can deploy the application into the runtime environment. To do this, complete the following steps:

1. Navigate to **AWS Mainframe Modernization** in the AWS Management Console. In the navigation pane, choose **Environments**. The Environments list page is displayed.

2. Choose the previously created runtime environment. The environment details page is displayed.

3. Choose **Deploy application**.

4. Choose the previously created application, then choose **Deploy**.

5. Wait until the application finishes its deployment. You'll see a banner with the message **Application was deployed successfully**.

Step 6: Start an application

1. Navigate to **AWS Mainframe Modernization** in the AWS Management Console and choose **Applications**.
2. Choose your application, and then go to **Deployments**. The status of the application should be **Succeeded**.

3. Choose **Actions**, and then choose **Start application**.

### Step 7: Access the application

1. Wait until the application is in the **Running** state. You'll see a banner with the message **Application was started successfully**.
2. Copy the application DNS hostname. You can find this hostname in the **Application information** section of the application.
3. In a browser, navigate to `http://{hostname}:{portname}/PlanetsDemo-web-1.0.0/`, where:
   - hostname is the DNS hostname copied previously.
   - portname is the Tomcat port defined in the application definition you created in **Step 2: Create the application definition** (p. 7).

The JICS screen appears.

If you can't access the application, see **Cannot access an application's URL** (p. 417).

### Step 8: Test the application

In this step, you run a transaction in the migrated application.

1. On the JICS screen, enter PINQ in the input field, and choose **Run** (or press Enter) to start the application transaction.

   The demo app screen should appear.
2. Type a planet name in the corresponding field and press Enter.

You should see details about the planet.

**Clean up resources**

If you no longer need the resources that you created for this tutorial, delete them to avoid additional charges. To do so, complete the following steps:

- If the AWS Mainframe Modernization application is still running, stop it.
- Delete the application. For more information, see Delete an AWS Mainframe Modernization application (p. 307).
- Delete the runtime environment. For more information, see Delete an AWS Mainframe Modernization runtime environment (p. 355).

**Next steps**

To learn more, you can explore the following tutorials:

- Set up a build and CI/CD pipeline (p. 181)
Tutorial: Managed runtime for Micro Focus

This tutorial shows how to deploy and run the BankDemo sample application in an AWS Mainframe Modernization managed runtime environment with the Micro Focus runtime engine. The Bank Demo sample application is a simplified banking application similar to what bank employees might use to check customer account balances, transfer funds, or calculate the cost of a loan. In the tutorial, you create resources in other AWS services. These include Amazon Simple Storage Service, Amazon Relational Database Service, or Amazon Aurora, and AWS Secrets Manager.

Topics
- Prerequisites (p. 15)
- Step 1: Create a database (p. 15)
- Step 2: Allow AWS Mainframe Modernization to retrieve the database admin credentials (p. 16)
- Step 3: Create a runtime environment (p. 17)
- Step 4: Create an application (p. 20)
- Step 5: Deploy an application (p. 23)
- Step 6: Import data sets (p. 24)
- Step 7: Start an application (p. 28)
- Step 8: Connect to the BankDemo sample application (p. 28)
- Clean up resources (p. 30)
- Next steps (p. 31)

Prerequisites

Before you start the tutorial, make sure that you complete the following prerequisites:

- Download the BankDemo sample application, unzip it, and upload the files to a bucket in Amazon S3. Make sure that the bucket is in the same Region as AWS Mainframe Modernization. For more information, see the Creating, configuring, and working with Amazon S3 buckets.
- Download the catalog files that the BankDemo sample application requires to locate the application data, unzip them, and then upload them to an Amazon S3 bucket, such as s3://m2-tutorial.
- Make sure that you have the CREATEDB permission.

Step 1: Create a database

In this step, you create a PostgreSQL database in either Amazon Relational Database Service (Amazon RDS) or Amazon Aurora. For the tutorial, this database contains the data sets that the BankDemo sample application uses for customer tasks, such as checking balances or transferring funds.

To create a database in Amazon RDS

1. To create the database, follow the instructions in Creating a PostgreSQL DB instance and connecting to a database on a PostgreSQL DB instance in the Amazon RDS User Guide.

   For Credential Settings, don't select Manage master credentials in AWS Secrets Manager. Save the username and password that you use for the database because you need them later in this tutorial.

2. After you create the database, configure it as follows:
   - Create a custom parameter group. Follow the instructions in Creating a DB parameter group.
   - Change the max_prepared_transactions parameter value to 100. Follow the instructions in Modifying parameters in a DB parameter group.
Step 2: Allow AWS Mainframe Modernization to retrieve the database admin credentials

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. In the navigation pane, choose Secrets.
3. Choose Store a new secret.
4. Set the Secret type to match the type of database that you created for this tutorial.
5. Enter the credentials that you specified when you created the database.
6. Select the encryption key that you want to use for this secret.
7. In the Database section, select the database that you created for this tutorial.
8. Choose Next.
9. Enter a name under Secret name.
10. In the Resource permissions section, choose Edit permissions, and then paste the following policy.

```
{
    "Effect" : "Allow",
    "Principal" : {
        "Service" : "m2.amazonaws.com"
    },
    "Action" : "secretsmanager:GetSecretValue",
    "Resource" : "*"
}
```

This policy allows AWS Mainframe Modernization to retrieve the credentials in the secret. For more information, see Attach a permissions policy to a secret in the AWS Secrets Manager User Guide.
11. Choose Next.

For more information on parameter groups, see Working with parameter groups. For more information on DB cluster parameter groups, see Working with DB cluster parameter groups.

To create a database in Aurora

1. To create the database, follow the instructions in Creating a DB cluster and connecting to a database on an Aurora PostgreSQL DB cluster in the Amazon Aurora User Guide.

For Credential Settings, don't select Manage master credentials in AWS Secrets Manager. Save the username and password that you use for the database because you need them later in this tutorial.

2. After you create the database, configure it as follows:

- Create a custom DB cluster parameter group. Follow the instructions in Creating a DB cluster parameter group.
- Change the max_prepared_transactions parameter value to 100. Follow the instructions in Modifying parameters in a DB cluster parameter group.
- Associate your custom parameter group with the database. Follow the instructions in Associating a DB cluster parameter group with a DB cluster.

For more information on parameter groups, see Working with parameter groups. For more information on DB cluster parameter groups, see Working with DB cluster parameter groups.

Step 2: Allow AWS Mainframe Modernization to retrieve the database admin credentials

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. In the navigation pane, choose Secrets.
3. Choose Store a new secret.
4. Set the Secret type to match the type of database that you created for this tutorial.
5. Enter the credentials that you specified when you created the database.
6. Select the encryption key that you want to use for this secret.
7. In the Database section, select the database that you created for this tutorial.
8. Choose Next.
9. Enter a name under Secret name.
10. In the Resource permissions section, choose Edit permissions, and then paste the following policy.

```
{
    "Effect" : "Allow",
    "Principal" : {
        "Service" : "m2.amazonaws.com"
    },
    "Action" : "secretsmanager:GetSecretValue",
    "Resource" : "*"
}
```

This policy allows AWS Mainframe Modernization to retrieve the credentials in the secret. For more information, see Attach a permissions policy to a secret in the AWS Secrets Manager User Guide.
11. Choose Next.

For more information on parameter groups, see Working with parameter groups. For more information on DB cluster parameter groups, see Working with DB cluster parameter groups.
12. (Optional) Configure automatic rotation.
13. Choose Next.
14. Review the configuration, and then choose Store.
15. In the list of secrets, choose this secret, and then copy its Amazon Resource Name (ARN) and save it. You need this ARN to create the AWS Mainframe Modernization application.

Step 3: Create a runtime environment

1. Open the AWS Mainframe Modernization console.

2. Under Modernize mainframe applications, choose Replatform with Micro Focus, and then choose Get started.
3. On the Get started page, under How would you like to start with AWS Mainframe Modernization?, choose Deploy. Then, in the Deploy section, choose Create runtime environment.
4. Choose Continue.

5. Specify a name for the runtime environment, such as Tutorial. Choose Next.
6. Under **Availability**, choose **High availability cluster**. Under **Resources**, choose either `M2.c5.large` or `M2.m5.large` for the instance type, and the number of instances that you want. You can specify up to two instances. Under **Security and network**, make sure that you choose at least two subnets, and choose **Allow applications deployed to this environment to be publicly accessible**. Then choose **Next**.
7. On the Attach storage page, choose Next.
8. On the Review and create page, review all the configurations that you provided for the runtime environment, and then choose Create Environment.
Step 4: Create an application

1. In the navigation pane, choose Applications. Then choose Create application.

2. On the Create application page, under Specify basic information, enter BankDemo for the application name and make sure Micro Focus is selected. Then choose Next.
3. Under **Specify resources and configurations**, choose how you want to specify the application definition. You can use the inline editor, or specify an application definition JSON file in an S3 bucket. Paste or type the application definition into the file, or provide the Amazon S3 location. Then choose **Next**.

You can use the following sample application definition file. Make sure to replace $s3_bucket and $s3-key-prefix with the correct values for your S3 bucket and the folder where you uploaded the BankDemo sample files. Also replace secret-manager-arn with the Amazon Resource Name (ARN) of the Secrets Manager secret you granted access to in Step 2.

```json
{
}
```
"template-version": "2.0",
"source-locations": [
  {
    "source-id": "s3-source",
    "source-type": "s3",
    "properties": {
      "s3-bucket": "my-bankdemo-bucket",
      "s3-key-prefix": "v1"
    }
  }
],
"definition": {
  "listeners": [
    {
      "port": 6000,
      "type": "tn3270"
    }
  ],
  "dataset-location": {
    "db-locations": [
      {
        "name": "Database1",
        "secret-manager-arn": "arn:aws:secretsmanager:Region:123456789012:secret:rds!PostgreSQL_database-1-n0A0BC"
      }
    ],
  },
  "batch-settings": {
    "initiators": [
      {
        "classes": ["A","B"],
        "description": "initiator_AB...."
      },
      {
        "classes": ["C","D"],
        "description": "initiator_CD...."
      }
    ],
    "jcl-file-location": "${s3-source}/jcl"
  },
  "cics-settings": {
    "binary-file-location": "${s3-source}/transaction",
    "csd-file-location": "${s3-source}/RDEF",
    "system-initialization-table": "BNKCICV"
  },
  "xa-resources": [
    {
      "name": "XASQL",
      "secret-manager-arn": "arn:aws:secretsmanager:Region:123456789012:secret:rds!PostgreSQL_database-1-n0A0BC",
      "module": "${s3-source}/xa/ESPGSQLXA64.so"
    }
  ]
}

**Note**
This file is subject to change.

For more information on the application definition, see [Micro Focus application definition (p. 339)](#).

4. On the **Review and create** page, review the information that you provided, and choose **Create application**.
Step 5: Deploy an application

1. In the navigation pane, choose Applications, and then choose BankDemo. Choose Actions, and then choose Deploy application.
Step 6: Import data sets

1. In the navigation pane, choose Applications, and then choose BankDemo. Choose the Data sets tab. Then choose how you want to specify the data sets. You can use a data set configuration JSON file in an S3 bucket, or specify the data set configuration values separately.

2. For each of the following data sets, complete these steps:
   - Choose Add new item.
   - Specify the data set name and location.
     
     Replace $S3_DATASET_PREFIX with the name of your S3 bucket that contains the catalog data, for example, $S3_DATASET_PREFIX=s3://m2-tutorial/catalog.

     **Note**
     Make sure you specify the S3 bucket name, not the bucket ARN. Do not specify an absolute path to resources in the bucket.
   - Choose Submit.
### Step 6: Import data sets

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>External Amazon S3 location</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFI01V.MFIDEMO.BNKACC</td>
<td>sql:// ES PACDatabase/VSAM/ MFI01V.MFIDEMO.BNKACC.DAT? folder=/DATA</td>
<td>$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKACC.DAT</td>
</tr>
<tr>
<td>MFI01V.MFIDEMO.BNKATYPE</td>
<td>sql:// ES PACDatabase/VSAM/ MFI01V.MFIDEMO.BNKATYPE.DAT? folder=/DATA</td>
<td>$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKATYPE.DAT</td>
</tr>
<tr>
<td>MFI01V.MFIDEMO.BNKCUST</td>
<td>sql:// ES PACDatabase/VSAM/ MFI01V.MFIDEMO.BNKCUST.DAT? folder=/DATA</td>
<td>$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKCUST.DAT</td>
</tr>
<tr>
<td>MFI01V.MFIDEMO.BNKHELP</td>
<td>sql:// ES PACDatabase/VSAM/ MFI01V.MFIDEMO.BNKHELP.DAT? folder=/DATA</td>
<td>$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKHELP.DAT</td>
</tr>
<tr>
<td>MFI01V.MFIDEMO.BNKTXN</td>
<td>sql:// ES PACDatabase/VSAM/ MFI01V.MFIDEMO.BNKTXN.DAT? folder=/DATA</td>
<td>$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKTXN.DAT</td>
</tr>
</tbody>
</table>

Alternatively, you can specify the data set configuration in a JSON file, such as the following. This file shows all the required data sets. Replace $S3_DATASET_PREFIX with the name of your S3 bucket that contains the catalog data, for example, m2-tutorial/catalog.

```json
{
  "dataSets": [
    {
      "dataSet": {
        "storageType": "Database",
        "datasetName": "MFI01V.MFIDEMO.BNKACC",
        "relativePath": "DATA",
        "datasetOrg": {
          "vsam": {
            "format": "KS",
            "encoding": "A",
            "primaryKey": {
              "length": 9,
              "offset": 5
            }
          }
        },
        "recordLength": {
          "min": 200,
          "max": 200
        }
      },
      "externalLocation": {
        "s3Location": "$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKACC.DAT"
      }
    }
  ]
}
```
Step 6: Import data sets

```json
{
  "dataSet": {
    "storageType": "Database",
    "datasetName": "MFI01V.MFIDEMO.BNKATYPE",
    "relativePath": "DATA",
    "datasetOrg": {
      "vsam": {
        "format": "KS",
        "encoding": "A",
        "primaryKey": {
          "length": 1,
          "offset": 0
        }
      }
    },
    "recordLength": {
      "min": 100,
      "max": 100
    }
  },
  "externalLocation": {
    "s3Location": "$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKATYPE.DAT"
  }
},
{
  "dataSet": {
    "storageType": "Database",
    "datasetName": "MFI01V.MFIDEMO.BNKCUST",
    "relativePath": "DATA",
    "datasetOrg": {
      "vsam": {
        "format": "KS",
        "encoding": "A",
        "primaryKey": {
          "length": 5,
          "offset": 0
        }
      }
    },
    "recordLength": {
      "min": 250,
      "max": 250
    }
  },
  "externalLocation": {
    "s3Location": "$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKCUST.DAT"
  }
},
{
  "dataSet": {
    "storageType": "Database",
    "datasetName": "MFI01V.MFIDEMO.BNKHELP",
    "relativePath": "DATA",
    "datasetOrg": {
      "vsam": {
        "format": "KS",
        "encoding": "A",
        "primaryKey": {
          "length": 8,
          "offset": 0
        }
      }
    },
    "recordLength": {
      "min": 83,
      "max": 83
    }
  },
  "externalLocation": {
    "s3Location": "$S3_DATASET_PREFIX/data/MFI01V.MFIDEMO.BNKHELP.DAT"
  }
}
```
When AWS Mainframe Modernization finishes importing the data sets, the **Status** field changes to **Completed**. Wait until the number of failed records is zero.
Step 7: Start an application

- In the navigation pane, choose Applications, and then choose BankDemo. Choose Actions, and then choose Start application.

When the BankDemo application starts to run successfully, a banner appears that says Application name was started successfully. The Status field changes to Available.

Step 8: Connect to the BankDemo sample application

Before you connect, make sure that the VPC and security group that you specified for the application are the same as the ones that you applied to your network interface.

To configure the TN3270 connection, you also need the DNS hostname of the network interface. To locate the DNS hostname for configuration, complete the following steps:

1. Open the AWS Mainframe Modernization console and choose Applications.
2. In the list of applications, choose the application whose detail page you want to display.
3. Note the string of characters under DNS Hostname.

Note
To ensure that the Network Load Balancer doesn't silently drop your connection, configure the keepalive setting on your TN3270 terminal to at least 180 seconds.

Continue to connect to the BankDemo sample application, as follows:

1. Start a terminal emulator. This tutorial uses Micro Focus Rumba+.
2. Choose Connection, then Configure, then TN3270.
To add the DNS hostname address that you noted earlier, choose Insert, and specify 6000 for the Telnet Port.

3. Enter the BANK transaction name.

4. Type B0001 for the username and A for the password.
5. After you log in successfully, you can navigate through the BankDemo application.

Clean up resources

If you no longer need the resources that you created for this tutorial, delete them to avoid additional charges. To do so, complete the following steps:

- If necessary, stop the BankDemo application.
- Delete the application. For more information, see Delete an AWS Mainframe Modernization application (p. 307).
Next steps

As next steps, see the following resources:

- To learn how to set up a build for your modernized applications, see Tutorial: Setting up the Micro Focus build for the BankDemo sample application (p. 270)
- To learn how to set up a CI/CD pipeline for your modernized applications, see Tutorial: Setting up a CI/CD pipeline for use with Micro Focus Enterprise Developer (p. 276)
Modernization approach

Migration is complex and has many variables. AWS Mainframe Modernization offers an evolutionary approach that provides some short-term wins by improving agility with plenty of opportunities to optimize and innovate later on. In addition, AWS Mainframe Modernization helps simplify the journey and still respects the particulars of your client’s company and business. The two main approaches that AWS Mainframe Modernization supports are automated refactoring or replatforming. Which to choose depends on your client’s situation.

Automated refactoring uses AWS Blu Age tools to automatically convert code, data, and dependencies to modern language, datastore, and frameworks, while at the same time guaranteeing functional equivalence with the same business functions.

Replatforming uses Micro Focus tools to transform mainframe workloads into agile services on AWS.

You can think of the modernization journey in stages. The first stage includes three phases: assess, mobilize, and migrate and modernize. The next stage includes the operate and optimize phase, where you can identify more opportunities for innovation.

Topics

• Assess phase (p. 32)
• Mobilize phase (p. 32)
• Migrate and modernize phase (p. 32)
• Operate and optimize phase (p. 33)

Assess phase

At the highest level, the Assess phase looks at whether you are ready to migrate. You define a business case, and then educate your team with workshops and an immersion day (demos and labs) offered by AWS. Workshops and immersion days address different topics. These tasks are conducted outside of AWS Mainframe Modernization.

Mobilize phase

In the Mobilize phase, you start your project with a kickoff, and then run through a discovery process that extracts data from your mainframe applications and ingests it to a migration tool. You identify the applications you want to migrate and select a few applications to pilot. You refine your business case, write your migration plan, and decide how you want to handle security and compliance, account governance, and your operational model. You set up a cloud center of excellence with the right people from your team. You run the pilots and document what you learned. You refine your migration plan and business case. Many of these tasks are conducted outside of AWS Mainframe Modernization.

Migrate and modernize phase

The Migrate and Modernize phase applies to each application and consists of several tasks, including assigning people, running in-depth discovery, figuring out the right application architecture on AWS, setting up application runtime environments, replatforming or refactoring your code, integrating
with other systems, and, of course, testing. At the end of the phase, you deploy the replatformed or refactored applications to production and cut over to the new system on AWS. Most or all of these tasks are conducted in AWS Mainframe Modernization, in another AWS service, or in a tool to which AWS Mainframe Modernization provides access.

If you want to use automated refactoring, see [Blu Insights](#). AWS Blu Insights is now available from the AWS Management Console through single sign-on. You do not have to manage separate AWS Blu Insights credentials any longer. You can access both the AWS AWS Blu Age Codebase and Transformation Center features directly from the AWS Management Console.

For migrating data from the mainframe to AWS, we recommend the AWS SCT and the AWS Database Migration Service. For more information, see [What is the AWS Schema Conversion Tool?](#) in the AWS Schema Conversion Tool User Guide and [What is AWS Database Migration Service?](#) in the AWS Database Migration Service User Guide.

**Operate and optimize phase**

In the Operate and Optimize phase, you focus on monitoring your deployed applications, managing resources, and ensuring that security and compliance are up to date. You also assess opportunities to optimize the migrated workloads.
Refactoring applications automatically with AWS Blu Age

Automated refactoring with AWS Blu Age provides an end to end solution for migrating and modernizing your mainframe applications. The steps in the refactoring process are as follows:

- Analyze inventory
- Analyze dependencies
- Automatically transform code
- Capture and manage test scenarios

You can complete the previous steps in the Blu Insights tool, available through single sign-on from the AWS Mainframe Modernization console. For more information on Blu Insights, see the Blu Insights documentation.

When you are satisfied with the transformed source code, it's time to move to AWS, where you will complete the following steps:

- Build and deploy the refactored application.
- Deploy and monitor your application in AWS Mainframe Modernization.

AWS Blu Age Runtime (on Amazon EC2) is one of the offerings of the AWS Mainframe Modernization service along with AWS Blu Age managed. With AWS Blu Age managed, you can deploy your modernized application to an AWS-managed environment that simplifies your experience, so you don't need to manage the underlying infrastructure that runs your modernized application. In contrast, with AWS Blu Age Runtime (on Amazon EC2) you can deploy your modernized application in your own AWS account, so you can manage your own infrastructure. With AWS Blu Age Runtime (on Amazon EC2) you have the flexibility to operate all the technical components required to run your modernized application the way you want.

Topics

- AWS Blu Age Runtime Concepts (p. 34)
- AWS Blu Age Runtime (on Amazon EC2) Setup (p. 72)
- AWS Blu Age Runtime (on Amazon EC2) Configuration (p. 93)
- AWS Blu Age Runtime (on Amazon EC2) APIs (p. 121)
- Modify the source code with Blu Age Developer IDE (p. 165)
- Set up a build and CI/CD pipeline (p. 181)
- AWS Blu Age Release Notes (p. 196)

AWS Blu Age Runtime Concepts

Understanding the basic concepts of the AWS Blu Age Runtime can help you understand how your applications are modernized with automated refactoring.

Topics

- AWS Blu Age Runtime High Level Architecture (p. 35)
- AWS Blu Age Structure of a Modernized Application (p. 37)
- Data Simplifier (p. 68)
AWS Blu Age Runtime High Level Architecture

As a part of the AWS Blu Age solution for modernizing legacy programs to Java, the AWS Blu Age Runtime provides a unified, REST-based entry point for modernized applications, and a framework of execution for such applications, through libraries providing legacy constructs and a standardization of programs code organization.

Such modernized applications are the result of the AWS Blu Age Automated Refactor process for modernizing mainframe and midrange programs (referred to in the following document as "legacy") to a web based architecture.

The AWS Blu Age Runtime goals are reproduction of legacy programs behavior (isofunctionality), performances (with respect to programs execution time and resources consumption), and ease of maintenance of modernized programs by Java developers, though the use of familiar environments and idioms such as tomcat, Spring, getters/setters, fluent APIs...

Topics
- AWS Blu Age runtime components (p. 35)
- Execution environments (p. 36)
- Statelessness and session handling (p. 37)
- High Availability and statelessness (p. 37)

AWS Blu Age runtime components

The AWS Blu Age Runtime environment is composed of two kinds of components:

- A set of java libraries (jar files) often referenced as “the shared folder”, and providing legacy constructs and statements.
- A set of web applications (war files) containing Spring-based web applications providing a common set of frameworks and services to modernized programs.

The following sections detail the role of both of these components.

AWS Blu Age libraries

The AWS Blu Age libraries are a set of jar files stored in a shared/subfolder added to the standard tomcat classpath, so as to make them available to all modernized Java programs. Their goal is to provide features that are neither natively nor easily available in the Java programming environment, but typical of legacy development environments. Those features are exposed in a way that is as familiar as possible to Java developers (getters/setters, class-based, fluent APIs). An important example is the Data Simplifier library, which provides legacy memory layout and manipulation constructs (encountered in COBOL, PL1 or RPG languages) to Java programs. Those jars are a core dependency of the modernized Java code generated from legacy programs. For more information about the Data Simplifier, see Data Simplifier (p. 68).

Web application

Web Application Archives (WARs) are a standard way of deploying code and applications to the tomcat application server. The ones provided as part of the AWS Blu Age runtime aim at providing a set of execution frameworks reproducing legacy environments and transaction monitors (JCL batches, CICS, IMS...), and associated required services.

The most important one is gapwalk-application (often shortened as “Gapwalk”), which provides a unified set of REST-based entry points to trigger and control transactions, programs and batches execution. For more information, see AWS Blu Age Runtime (on Amazon EC2) APIs (p. 121).
This web application allocates Java execution threads and resources to run modernized programs in the context for which they were designed. Examples of such reproduced environments are detailed in the following section.

Other web applications add to the execution environment (more precisely, to the "Programs Registry" described below) programs emulating the ones available to, and callable from, the legacy programs. Two important categories of such are:

- Emulation of OS-provided programs: JCL-driven batches especially expect to be able to call a variety of file and database manipulating programs as part of their standard environment. Examples include SORT/DFSORT or IDCAMS. For this purpose, Java programs are provided that reproduce such behavior, and are callable using the same conventions as the legacy ones.
- "Drivers", which are specialized programs provided by the execution framework or middleware as entry points. An example is CBLTDLI, which COBOL programs executing in the IMS environment depend on to access IMS-related services (IMS DB, user dialog through MFS, etc.).

Programs registry

To participate in and take advantage of those constructs, frameworks and services, Java programs modernized from legacy ones adhere to a specific structure documented in AWS Blu Age Structure of a Modernized Application (p. 37). At startup, the AWS Blu Age Runtime will collect all such programs in a common "Programs Registry" so that they can be invoked (and call each other) afterwards. The Program Registry provides loose coupling and possibilities of decomposition (since programs calling each other do not have to be modernized simultaneously).

Execution environments

Frequently encountered legacy environments and choreographies are available:

- JCL-driven batches, once modernized to Java programs and Groovy scripts, can be started in a synchronous (blocking) or asynchronous (detached) way. In the latter case, their execution can be monitored through REST endpoints.
- A AWS Blu Age subsystem provides an execution environment similar to CICS through:
  - an entry point used to start a CICS transaction and run associated programs while respecting CICS "run levels" choreography,
  - an external storage for Resource Definitions,
  - an homogeneous set of Java fluent APIs reproducing EXEC CICS statements,
  - a set of pluggable classes reproducing CICS services, such as Temporary Storage Queues, Temporary Data Queues or files access (multiple implementations are usually available, such as Amazon Managed Service for Apache Flink, Amazon Simple Queue Service, or RabbitMQ for TD Queues),
  - for user-facing applications, the BMS screen description format is modernized to an Angular web application, and the corresponding "pseudo-conversational" dialog is supported.
- Similarly, another subsystem provides IMS message-based choreography, and supports modernization of UI screens in the MFS format.
- In addition, a third subsystem allows execution of programs in an iSeries-like environment, including modernization of DSPF (Display File)-specified screens.

All of those environments build upon common OS-level services such as:

- the emulation of legacy memory allocation and layout (Data Simplifier),
- Java thread-based reproduction of the COBOL "run units" execution and parameters passing mechanism (CALL statement),
- emulation of flat files, VSAM and GDG like file organizations, through the Bluesam set of libraries,
access to data stores, such as RDBMS (EXEC SQL statements).

Statelessness and session handling

An important feature of the AWS Blu Age Runtime is to enable High Availability (HA) and horizontal scalability scenarios when executing modernized programs.

The cornerstone for this is statelessness, an important example of which is HTTP session handling.

Session handling

Tomcat being web-based, an important mechanism for this is HTTP session handling (as provided by tomcat and Spring) and stateless design. As such statelessness design is based on the following:

- users connect though HTTPS,
- application servers are deployed behind a Load balancer,
- when a user first connects to the application it will be authenticated and the application server will create an identifier (typically within a cookie)
- this identifier will be used as a key to save and retrieve the user context to/from an external cache (data store).

Cookie management is done automatically by the AWS Blu Age framework and the underlying tomcat server, this is transparent to the user. The user internet browser will manage this automatically.

The Gapwalk web application may store the session state (the context) in various data stores:

- Amazon ElastiCache for Redis
- Redis cluster
- in memory map (only for development and standalone environments, not suitable for HA).

High Availability and statelessness

More generally, a design tenet of the AWS Blu Age framework is statelessness: most non-transient states required to reproduce legacy programs behavior are not stored inside the application servers, but shared through an external, common "single source of truth".

Examples of such states are CICS's Temporary Storage Queues or Resource Definitions, and typical external storages for those are Redis-compatible servers or relational databases.

This design, combined with load balancing and shared sessions, leads to most of user-facing dialog (OLTP, "Online Transactional Processing") to be distributable between multiple "nodes" (here, tomcat instances).

Indeed a user may execute a transaction on any server and not care if the next transaction call is performed on a different server. Then when a new server is spawned (because of auto scaling, or to replace a non healthy server), we can guarantee that any reachable and healthy server can run the transaction as expected with the proper results (expected returned value, expected data change in database, etc.).

AWS Blu Age Structure of a Modernized Application

This document provides details about the structure of modernized applications (using AWS Mainframe Modernization refactoring tools), so that developers can accomplish various tasks, such as:
Modernized Application Structure

- navigating into applications smoothly.
- developing custom programs that can be called from the modernized applications.
- safely refactoring modernized applications.

We assume that you already have basic knowledge about the following:

- legacy common coding concepts, such as records, data sets and their access modes to records -- indexed, sequential --, VSAM, run units, jcl scripts, CICS concepts, and so on.
- java coding using the Spring framework.
- Throughout the document, we use short class names for readability. For more information, see AWS Blu Age fully qualified name mappings (p. 67) to retrieve the corresponding fully qualified names for AWS Blu Age runtime elements and Third party fully qualified name mappings (p. 67) to retrieve the corresponding fully qualified names for third party elements.
- All artifacts and samples are taken from the modernization process outputs of the sample COBOL/CICS CardDemo application.

Topics

- Artifacts organization (p. 38)
- Running and calling programs (p. 55)
- Write your own program (p. 60)
- Fully qualified name mappings (p. 67)

Artifacts organization

AWS Blu Age modernized applications are packaged as java web applications (.war), that you can deploy on a JEE server. Typically, the server is a Tomcat instance that embeds the AWS Blu Age Velocity runtime, which is currently built upon the Springboot and Angular (for the UI part) frameworks.

The war aggregates several component artifacts (.jar). Each jar is the result of the compilation (using the maven tool) of a dedicated java project whose elements are the result of the modernization process.

The basic organization relies on the following structure:
• Entities project: contains business model and context elements. The project name generally ends with "-entities". Typically, for a given legacy COBOL program, this corresponds to the modernization of the I/O section (data sets) and the data division. You can have more than one entities project.

• Service project: contains legacy business logic modernization elements. Typically, the procedure division of a COBOL program. You can have more than one service project.

• Utility project: contains shared common tools and utilities, used by other projects.

• Web project: contains the modernization of UI-related elements when applicable. Not used for batch-only modernization projects. These UI elements could come from CICS BMS maps, IMS MFS components, and other mainframe UI sources. You can have more than one Web project.

Entities project contents

Note
The following descriptions only apply to COBOL and PL/I modernization outputs. RPG modernization outputs are based on a different layout.

Before any refactoring, the packages organization in the entities project is tied to the modernized programs. You can accomplish this in a couple of different ways. The preferred way is to use the Refactoring toolbox, which operates before you trigger the code generation mechanism. This is an advanced operation, which is explained in the BluAge trainings. For more information, see Refactoring workshop. This approach allows you to preserve the capability to re-generate the java code later, to benefit from further improvements in the future, for instance). The other way is to do regular java refactorings, directly on the generated source code, using any java refactoring approach you might like to apply -- at your own risk.
Program related classes

Each modernized program is related to two packages, a business.context and a business.model package.
• *base package.program.business.context*

The business.context sub-package contains two classes, a configuration class and a context class.

• One configuration class for the program, which contains specific configuration details for the given program, such as the character set to use to represent character-based data elements, the default byte value for padding data structure elements and so on. The class name ends with "Configuration". It is marked with the @org.springframework.context.annotation.Configuration annotation and contains a single method that must return a properly setup Configuration object.

```java
package awslmage.13.workshop.cbact04c.business.context;

import com.neteffective.blugae_gapwalk.datasetsimplifier.configuration.Configuration;

/**
 * Creates Datasimplifier configuration for the Cbact04cContext context.
 */
@lacy
@org.springframework.context.annotation.Configuration
public class cbact04cConfiguration {

    @see(name = "Cbact04cContextConfiguration")
    public Configuration configuration() {
        return new ConfigurationBuilder()
            .encoding(Charset.forName("CP1047"))
            .humanReadableEncoding(Charset.forName("ISO-8859-15"))
            .initDefaultByte(0)
            .build();
    }

}
```

• One context class, which serves as a bridge between the program service classes (see below) and the data structures (Record) and data sets (File) from the model sub-package (see below). The class name ends with "Context" and is a subclass of the RuntimeContext class.
*base package.*

The model sub-package contains all the data structures that the given program can use. For instance, any 01 level COBOL data structure corresponds to a class in the model sub-package (lower level data structures are properties of their owning 01 level structure). For more information about how we modernize 01 data structures, see Data Simplifier (p. 68).
All classes extend the RecordEntity class, which represents the access to a business record representation. Some of the records have a special purpose, as they’re bound to a File. The binding between a Record and a File is made in the corresponding *FileHandler methods found in the context class when creating the file object. For example, the following listing shows how the TransactFileFile File is bound to the transactFile Record (from the model sub-package).

Service project contents

Every service project comes with a dedicated Springboot application, which is used as the backbone of the architecture. This is materialized through the class named SpringBootLauncher, located in the base package of the service java sources:
This class is notably responsible for:

- making the glue between the program classes and managed resources (datasources / transaction managers / data sets mappings / etc ...).
- providing a `ConfigurableApplicationContext` to programs.
- discovering all classes marked as spring components (`@Component`).
- ensuring programs are properly registered in the `ProgramRegistry` -- see the initialize method in charge of this registration.

```java
/**
 * Initialization method called when the spring application is ready.
 * Register all programs and services to the `ProgramContainer`.
 */
@EventListener
public void initialise(ApplicationReadyEvent event) {
    Map<String, ProgramContainers> programContainers = event.getApplicationContext().getBeansOfType(ProgramContainer.class);
    programContainers.values().forEach(registry -> registry.registerProgram);
    Map<String, ServiceContainers> serviceContainers = event.getApplicationContext().getBeansOfType(ServiceContainer.class);
    serviceContainers.values().forEach(registry -> registry.registerService);
}
```

**Program related artifacts**

Without prior refactoring, the business logic modernization outputs are organized on a two or three packages per legacy program basis:
The most exhaustive case will have three packages:

- **base package.program.service**: contains an interface named `ProgramProcess`, which has business methods to handle the business logic, preserving the legacy execution control flow.

- **base package.program.service.impl**: contains a class named `ProgramProcessImpl`, which is the implementation of the Process interface described previously. This is where the legacy statements are "translated" to java statements, relying on the AWS Blu Age framework:
• **base package.program.statemachine**: this package might not always be present. It is required when the modernization of the legacy control flow has to use a state machine approach (namely using the Spring StateMachine framework) to properly cover the legacy execution flow.

In that case, the statemachine sub-package contains two classes:

• **ProgramProcedureDivisionStateMachineController**: a class that extends a class implementing the StateMachineController (define operations needed to control the execution of a state machine) and StateMachineRunner (define operations required to run a state machine) interfaces, used to drive the Spring state machine mechanics; for instance, the SimpleStateMachineController as in the sample case.
The state machine controller defines the possible different states and the transitions between them, which reproduce the legacy execution control flow for the given program.

When building the state machine, the controller refers to methods that are defined in the associated service class located in the state machine package and described below:

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

- ProgramProcedureDivisionStateMachineService: this service class represents some business logic that is required to be bound with the state machine that the state machine controller creates, as described previously.

The code in the methods of this class use the Events defined in the state machine controller:
The statemachine service also makes calls to the process service implementation described previously:

In addition to that, a package named `base_package.program` plays a significant role, as it gathers one class per program, which will serve as the program entry point (more details about this later on). Each class implements the `Program` interface, marker for a program entry point.
Other artifacts

- BMS MAPs companions

In addition to program related artifacts, the service project can contain other artifacts for various purposes. In the case of the modernization of a CICS online application, the modernization process produces a json file and puts in the map folder of the /src/main/resources folder:
The Blu Age runtime consumes those json files to bind the records used by the SEND MAP statement with the screen fields.
• Groovy Scripts

If the legacy application had JCL scripts, those have been modernized as groovy scripts, stored in the /src/main/resources/scripts folder (more on that specific location later on):

Those scripts are used to launch batch jobs (dedicated, non-interactive, cpu-intensive data processing workloads).

• SQL files

If the legacy application was using SQL queries, the corresponding modernized SQL queries have been gathered in dedicated properties files, with the naming pattern program.sql, where program is the name of the program using those queries.
The contents of those SQL files are a collection of (key=query) entries, where each query is associated to a unique key, that the modernized program uses to run the given query:

For instance, the COSGN00C program is executing the query with key "COSGN00C_1" (the first entry in the SQL file):

Utilities project contents

The utilities project, whose name ends with "-tools", contains a set of technical utilities, which might be used by all the other projects.
Web project(s) contents

The web project is only present when modernizing legacy UI elements. The modern UI elements used to build the modernized application front-end are based on Angular. The sample application used to show the modernization artifacts is a COBOL/CICS application, running on a mainframe. The CICS system uses MAPs to represent the UI screens. The corresponding modern elements will be, for every map, a html file accompanied by Typescript files:
The web project only takes care of the frontend aspect of the application. The service project, which relies on the utility and entities projects, provides the backend services. The link between the frontend and the backend is made through the web application named Gapwalk-Application, which is part of the standard AWS Blu Age runtime distribution.

**Running and calling programs**

On legacy systems, programs are compiled as stand-alone executables that can call themselves through a CALL mechanism, such as the COBOL CALL statement, passing arguments when needed. The
modernized applications offer the same capability but use a different approach, because the nature of the involved artifacts differs from the legacy ones.

On the modernized side, program entry points are specific classes that implement the `Program` interface, are Spring components (@Component) and are located in service projects, in a package named `base package.program`.

### Programs registration

Each time the Tomcat server that hosts modernized applications is started, the service Springboot application is also started, which triggers the programs registration. A dedicated registry named `ProgramRegistry` is populated with program entries, each program being registered using its identifiers, one entry per known program identifier, which means that if a program is known by several different identifiers, the registry contains as many entries as there are identifiers.

The registration for a given program relies on the collection of identifiers returned by the `getProgramIdentifiers()` method:

```java
package aws.blage.lz.workshop.program;

import aws.blage.lz.workshop.springboot.Application;

@Component
@SnackBar
public class CBACT implements Program {

    private static final Set<String> programIdentifiers = Collections.unmodifiableSet(Collections.emptySet());
    private static final String programIdentifier = "CBACT";
    
    @Override
    public String getProgramIdentifier() {
        return programIdentifier;
    }

    @Override
    public List<String> getProgramIdentifiers() {
        return Collections.emptyList();
    }

    public void configureApplicationContext(ApplicationContext applicationContext) {
        return applicationContext.getBeansOfType(Application.class);
    }

    public void updateExecutionContext(ExecutionContext executionContext) {
        executionContext.setDatabaseSupport(DataSource);        
        executionContext.setDatabaseSupport(ExecutionContext.DATABASE_SUPPORT);        
        executionContext.setTransactionManager(TransactionManager);        
        executionContext.setSqlSessionFactory(SqlSessionFactory);        
        return executionContext.getSqlSessionFactory();
    }

    public String getProgramIdentifiers() {
        return programIdentifiers;
    }
}
```

In this example, the program is registered once, under the name 'CBACT04C' (look at the contents of the programIdentifiers collection). The tomcat logs show every program registration. The program registration only depends on the declared program identifiers and not the program class name itself (though typically the program identifiers and program class names are aligned).

The same registration mechanism applies to utility programs brought by the various utility AWS Blu Age web applications, which are part of the AWS Blu Age runtime distribution. For instance, the Gapwalk-Utility-Pgm webapp provides the functional equivalents of the z/OS system utilities (IDCAMS, ICEGENER, SORT, and so on) and can be called by modernized programs or scripts. All available utility programs that are registered at Tomcat startup are logged in the Tomcat logs.
Scripts and daemons registration

A similar registration process, at Tomcat startup time, occurs for groovy scripts that are located in the /src/main/resources/scripts folder hierarchy. The scripts folder hierarchy is traversed, and all groovy scripts that are discovered (except the special functions.groovy reserved script) are registered in the ScriptRegistry, using their short name (the part of the script file name located before the first dot character) as the key for retrieval.

**Note**

- If several scripts have file names that will result in producing the same registration key, only the latest is registered, overwriting any previously encountered registration for that given key.
- Considering the above note, pay attention when using sub-folders as the registration mechanism flattens the hierarchy and could lead to unexpected overwrites. The hierarchy does not count in the registration process: typically /scripts/A/myscript.groovy and /scripts/B/myscript.groovy will lead to /scripts/B/myscript.groovy overwriting /scripts/A/myscript.groovy.

The groovy scripts in the /src/main/resources/daemons folder are handled a bit differently. They're still registered as regular scripts, but in addition, they are launched once, directly at Tomcat startup time, asynchronously.

After scripts are registered in the ScriptRegistry, a REST call can launch them, using the dedicated endpoints that the Gapwalk-Application exposes. For more information, see the corresponding documentation.

Programs calling programs

Each program can call another program as a subprogram, passing parameters to it. Programs use an implementation of the ExecutionController interface to do so (most of the time, this will be an ExecutionControllerImpl instance), along with a fluent API mechanism named the CallBuilder to build the program call arguments.

All programs methods take both a RuntimeContext and an ExecutionController as method arguments, so an ExecutionController is always available to call other programs.

See, for instance, the following diagram, which shows how the CBST03A program calls the CBST03B program as a sub-program, passing parameters to it:
The first argument of the `ExecutionController.callSubProgram` is an identifier of the program to call (that is, one of the identifiers used for the program registration -- see paragraphs above).

The second argument, which is the result of the build on the `CallBuilder`, is an array of `Record`, corresponding to the data passed from caller to callee.

The third and last argument is the caller `RuntimeContext` instance.

All three arguments are mandatory and cannot be null, but the second argument can be an empty array.

The callee will be able to deal with passed parameters only if it was originally designed to do so. For a legacy COBOL program, that means having a LINKAGE section and a USING clause for the procedure division to make use of the LINKAGE elements.

For instance, see the corresponding [CBSTM03B.CBL](https://github.com/aws-samples/aws-mainframe-modernization-carddemo/blob/main/app/cbl/CBSTM03B.CBL) COBOL source file:

```cobol
LINKAGE SECTION.
 01 LK-N038-AREA.
    05 LK-N038-DD     PIC X(8).
    05 LK-N038-OPER   PIC X(91).
   88 M038-OPEN      VALUE 'O'.
   88 M038-CLOSE     VALUE 'C'.
   88 M038-READ     VALUE 'R'.
   88 M038-READ-X   VALUE 'K'.
   88 M038-WRITE    VALUE 'N'.
   88 M038-REWRITE  VALUE 'Z'.
    05 LK-N038-RC    PIC X(92).
    05 LK-N038-KEY   PIC X(25).
  05 LK-N038-KEY-LN PIC 99(4).
    05 LK-N038-FLDT  PIC X(1000).
PROCEDURE DIVISION USING LK-N038-AREA.
```

So the CBSTM03B program takes a single `Record` as a parameter (an array of size 1). This is what the `CallBuilder` is building, using the `byReference()` and `getArguments()` methods chaining.

The `CallBuilder` fluent API class has several methods available to populate the array of arguments to pass to a callee:

- `asPointer(RecordAdaptable)`: add an argument of pointer kind, by reference. The pointer represents the address of a target data structure.
- `byReference(RecordAdaptable)`: add an argument by reference. The caller will see the modifications that the callee performs.
- `byReference(RecordAdaptable...)`: varargs variant of the previous method.
- `byValue(Object)`: add an argument, transformed to a `Record`, by value. The caller won’t see the modifications the callee performs.
- `byValue(RecordAdaptable)`: same as the previous method, but the argument is directly available as a `RecordAdaptable`.
- `byValueWithBounds(Object, int, int)`: add an argument, transformed to a `Record`, extracting the byte array portion defined by the given bounds, by value.

Finally, the `getArguments` method will collect all added arguments and return them as an array of `Record`. 
**Note**

It is the responsibility of the caller to make sure the arguments array has the required size, that the items are properly ordered and compatible, in terms of memory layout with the expected layouts for the linkage elements.

**Scripts calling programs**

Calling registered programs from groovy scripts require using a class instance implementing the `MainProgramRunner` interface. Usually, getting such an instance is achieved through Spring's `ApplicationContext` usage:

```groovy
REPROC.groovy

// Import
1 import com.neteffective.bluesp.gpwalk.rt.provider.ScriptRegistry
2 import com.neteffective.bluesp.gpwalk.rt.call.MainProgramRunner
3 import com.neteffective.bluesp.gpwalk.io.support.FileConfigurationUtils
4 import com.neteffective.bluesp.gpwalk.rt.job.support.DefaultJobContext
5 import com.neteffective.bluesp.gpwalk.rt.util.GroovyUtils
6 import com.neteffective.bluesp.gpwalk.rt.job.support.FileConfiguration
7 import com.neteffective.bluesp.gpwalk.rt.call.exception.GroovyExecutionException
8 import com.neteffective.bluesp.gpwalk.rt.call.ExecutionController
9
10 // Variables
11 def applicationContext = new ApplicationContex
12 def job = new Job
13 def task = new Task
14 def program = new Program
15
16 def programRunner = ApplicationContext.getBean("MainProgramRunner")
17
18 def result = ProgramRunner.runProgram(job, task, program, programRunner)

// After a MainProgramRunner interface is available, use the runProgram method to call a program and pass the identifier of the target program as a parameter:

```groovy

In the previous example, a job step calls IDCAMS (file handling utility program), providing a mapping between actual data set definitions and their logical identifiers.

When dealing with data sets, legacy programs mostly use logical names to identify data sets. When the program is called from a script, the script must map logical names with actual physical data sets. These data sets could be on the filesystem, in a Blusam storage or even defined by an inline stream, the concatenation of several data sets, or the generation of a GDG.

Use the `withFileConfiguration` method to build a logical to physical map of data sets and make it available to the called program.
Write your own program

Writing your own program for scripts or other modernized programs to call is a common task. Typically, on modernization projects, you write your own programs when an executable legacy program is written in a language that the modernization process doesn’t support, or the sources have been lost (yes, that can happen), or the program is an utility whose sources are not available.

In that case, you might have to write the missing program, in java, by yourself (assuming you have enough knowledge about what the program expected behaviour should be, the memory layout of the program's arguments if any, and so on.) Your java program must comply with the program mechanics described in this document, so that other programs and scripts can run it.

To make sure the program is usable, you must complete two mandatory steps:

• Write a class that implements the Program interface properly, so that it can be registered and called.
• Make sure your program is registered properly, so that it is visible from other programs/scripts.

Writing the program implementation

Use your IDE to create a new java class that implements the Program interface:
The following image shows the Eclipse IDE, which takes care of creating all mandatory methods to be implemented:
Spring integration

First, the class must be declared as a Spring component. Annotate the class with the `@Component` annotation:

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

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

import aws.bluage.l3.workshop.SpringBootLauncher;

@Component
public class MyUtilityProgram implements Program {

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

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

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

    public void run(ExecutionController ctrl) {
        // TODO Auto-generated method stub
    }
}
```

Next, implement the required methods properly. In the context of this sample, we added the `MyUtilityProgram` to the package that already contains all modernized programs. That placement permits the program to use the existing Springboot application to provide the required `ConfigurableApplicationContext` for the `getSpringApplication` method implementation:
You might choose a different location for your own program. For instance, you might locate the given program in another dedicated service project. Make sure the given service project has its own Springboot application, which makes it possible to retrieve the ApplicationContext (that should be a ConfigurableApplicationContext).

**Giving an identity to the program**

To be callable by other programs and scripts, the program must be given at least one identifier, which must not collide with any other existing registered program within the system. The identifier choice might be driven by the need to cover an existing legacy program replacement; in that case, you'll have to use the expected identifier, as met in CALL occurrences found throughout the legacy programs. Most of the program identifiers are 8 characters long in legacy systems.

Creating an unmodifiable set of identifiers in the program is one way of doing this. The following example shows choosing “MYUTILPG” as the single identifier:

```java
public class MyUtilityProgram implements Program {

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

**Associate the program to a context**

The program needs a companion RuntimeContext instance. For modernized programs, AWS Blu Age automatically generates the companion context, using the data structures that are part of the legacy program.

If you're writing your own program, you must write the companion context as well.

Referring to [Program related classes (p. 40)](#), you can see that a program requires at least two companion classes:

- a configuration class.
- a context class that uses the configuration.

If the utility program uses any extra data structure, it should be written as well and used by the context.

Those classes should be in a package that is part of a package hierarchy that will be scanned at application startup, to make sure the context component and configuration will be handled by the Spring framework.

Let's write a minimal configuration and context, in the base package `myutilityprogram.business.context` package, freshly created in the entities project:
Here is the configuration content. It is using a configuration build similar to other modernized programs nearby. You'll probably have to customize this for your specific needs.

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

import java.nio.charset.Charset;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Lazy;
import com.neteffective.blau.ge.map.data.simplifier.configuration.Configuration;
import com.neteffective.blau.ge.map.data.simplifier.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"))
        .humantReadableEncoding(Charset.forName("ISO-8859-15"))
        .initDefaultByte(8)
        .build();
}
}
```

Notes:

- General naming convention is `ProgramNameConfiguration`.
- It must use the `@org.springframework.context.annotation.Configuration` and `@Lazy` annotations.
- The bean name usually follows the `ProgramNameContextConfiguration` convention, but this is not mandatory. Make sure to avoid bean name collisions across the project.
- The single method to implement must return a `Configuration` object. Use the `ConfigurationBuilder` fluent API to help you build one.

And the associated context:
The context class should extend an existing Context interface implementation (either RuntimeContext or JicsRuntimeContext, which is an enhanced RuntimeContext with JICS specifics items).

- General naming convention is ProgramNameContext.
- You must declare it as a Prototype component, and use the @Lazy annotation.
- The constructor refers to the associated configuration, using the @Qualifier annotation to target the proper configuration class.
- If the utility program uses some extra data structures, they should be:
  - written and added to the base package.business.model package.
  - referenced in the context. Take a look at other existing context classes to see how to reference data structures classes and adapt the context methods (constructor / clean-up / reset) as needed.

Now that a dedicated context is available, let the new program use it:
Notes:

- The `getContext` method must be implemented strictly as shown, using a delegation to the `getOrCreate` method of the `ProgramContextStore` class and the `@Autowired` Spring `BeanFactory`. A single program identifier is used to store the program context in the `ProgramContextStore`; this identifier is referenced as being the 'program main identifier'.

- The companion configuration and context classes must be referenced using the `@Import` spring annotation.

**Implementing the business logic**

When the program skeleton is complete, implement the business logic for the new utility program.

Do this in the `run` method of the program. This method will be executed anytime the program is called, either by another program or by a script.

Happy coding!

**Handling the program registration**

Finally, make sure the new program is properly registered in the `ProgramRegistry`. If you added the new program to the package that already contains other programs, there's nothing more to be done. The new program is picked up and registered with all its neighbour programs at application startup.

If you chose another location for the program, you must make sure the program is properly registered at Tomcat startup. For some inspiration about how to do that, look at the `initialize` method of the generated SpringBootLauncher classes in the service project(s) (see Service project contents (p. 43)).

Check the Tomcat startup logs. Every program registration is logged. If your program is successfully registered, you'll find the matching log entry.

When you're sure that your program is properly registered, you can start iterating on the business logic coding.
Fully qualified name mappings

This section contains lists of the AWS Blu Age and third-party fully qualified name mappings for use in your modernized applications.

AWS Blu Age fully qualified name mappings

<table>
<thead>
<tr>
<th>Short name</th>
<th>Fully qualified name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallBuilder</td>
<td>com.netfective.bluage.gapwalk.runtime.statements.CallBuilder</td>
</tr>
<tr>
<td>Configuration</td>
<td>com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration</td>
</tr>
<tr>
<td>ConfigurationBuilder</td>
<td>com.netfective.bluage.gapwalk.datasimplifier.configuration.ConfigurationBuilder</td>
</tr>
<tr>
<td>ExecutionController</td>
<td>com.netfective.bluage.gapwalk.rt.call.ExecutionController</td>
</tr>
<tr>
<td>ExecutionControllerImpl</td>
<td>com.netfective.bluage.gapwalk.rt.call.internal.ExecutionControllerImpl</td>
</tr>
<tr>
<td>File</td>
<td>com.netfective.bluage.gapwalk.rt.io.File</td>
</tr>
<tr>
<td>MainProgramRunner</td>
<td>com.netfective.bluage.gapwalk.rt.call.MainProgramRunner</td>
</tr>
<tr>
<td>Program</td>
<td>com.netfective.bluage.gapwalk.rt.provider.Program</td>
</tr>
<tr>
<td>ProgramContextStore</td>
<td>com.netfective.bluage.gapwalk.rt.context.ProgramContextStore</td>
</tr>
<tr>
<td>ProgramRegistry</td>
<td>com.netfective.bluage.gapwalk.rt.provider.ProgramRegistry</td>
</tr>
<tr>
<td>Record</td>
<td>com.netfective.bluage.gapwalk.datasimplifier.data.Record</td>
</tr>
<tr>
<td>RecordEntity</td>
<td>com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity</td>
</tr>
<tr>
<td>RuntimeContext</td>
<td>com.netfective.bluage.gapwalk.rt.context.RuntimeContext</td>
</tr>
<tr>
<td>SimpleStateMachineController</td>
<td>com.netfective.bluage.gapwalk.rt.statemachine.SimpleStateMachineController</td>
</tr>
<tr>
<td>StateMachineController</td>
<td>com.netfective.bluage.gapwalk.rt.statemachine.StateMachineController</td>
</tr>
<tr>
<td>StateMachineRunner</td>
<td>com.netfective.bluage.gapwalk.rt.statemachine.StateMachineRunner</td>
</tr>
</tbody>
</table>

Third party fully qualified name mappings

<table>
<thead>
<tr>
<th>Short name</th>
<th>Fully qualified name</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Autowired</td>
<td>org.springframework.beans.factory.annotation.Autowired</td>
</tr>
<tr>
<td>@Bean</td>
<td>org.springframework.context.annotation.Bean</td>
</tr>
<tr>
<td>BeanFactory</td>
<td>org.springframework.beans.factory.BeanFactory</td>
</tr>
<tr>
<td>@Component</td>
<td>org.springframework.stereotype.Component</td>
</tr>
<tr>
<td>ConfigurableApplicationContext</td>
<td>org.springframework.context.ConfigurableApplicationContext</td>
</tr>
<tr>
<td>@Import</td>
<td>org.springframework.context.annotation.annotation.Import</td>
</tr>
<tr>
<td>@Lazy</td>
<td>org.springframework.context.annotation.annotationLazy</td>
</tr>
</tbody>
</table>
Data Simplifier

On mainframe and midrange systems (referred to in the following topic as "legacy" systems), frequently used programming languages such as COBOL, PL/I or RPG provide low-level access to memory. This access focuses on memory layout accessed through native types such as zoned, packed, or alphanumeric, possibly also aggregated through groups or arrays.

A mix of accesses to a given piece of memory, through both typed fields and as direct access to bytes (raw memory), coexists in a given program. For example, COBOL programs will pass arguments to callees as contiguous sets of bytes (LINKAGE), or read/write data from files in the same manner (records), while interpreting such memory ranges with typed fields organized in copybooks.

Such combinations of raw and structured access to memory, the reliance on precise, byte-level memory layout, and legacy types, such as zoned or packed, are features that are neither natively nor easily available in the Java programming environment.

As a part of the AWS Blu Age solution for modernizing legacy programs to Java, the Data Simplifier library provides such constructs to modernized Java programs, and exposes those in a way that is as familiar as possible to Java developers (getters/setters, byte arrays, class-based). It is a core dependency of the modernized Java code generated from such programs.

For simplicity, most of the following explanations are based on COBOL constructs, but you can use the same API for both PL1 and RPG data layout modernization, since most of the concepts are similar.

Topics
- Main classes (p. 68)
- Data binding and access (p. 71)
- FQN of discussed Java types (p. 71)

Main classes

For easier reading, this document uses the Java short names of the AWS Blu Age API interfaces and classes. For more information, see FQN of discussed Java types (p. 71).

Low level memory representation

At the lowest level, memory (a contiguous range of bytes accessible in a fast, random way) is represented by the Record interface. This interface is essentially an abstraction of a byte array of a fixed size. As such, it provides setters and getters able to access or modify the underlying bytes.

Structured data representation

To represent structured data, such as "01 data items", or "01 copybooks", as found in COBOL DATA DIVISION, subclasses of the RecordEntity class are used. Those are normally not written by hand, but generated by the AWS Blu Age modernization tools from the corresponding legacy constructs. It is still useful to know about their main structure and API, so you can understand how the code in a modernized program uses them. In the case of COBOL, that code is Java generated from their PROCEDURE DIVISION.

Generated code represents each "01 data item" with a RecordEntity subclass; each elementary field or aggregate composing it is represented as a private Java field, organized as a tree (each item has a parent, except for the root one).

For illustration purposes, here is an example COBOL data item, followed by the corresponding AWS Blu Age generated code that modernizes it:
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);
}

Elementary fields

Fields of class Elementary (or Filler, when unnamed) represent a "leaf" of the legacy data structure. They are associated with a contiguous span of underlying bytes ("range") and commonly have a type (possibly parameterized) expressing how to interpret and modify those bytes (by respectively "decoding" and "encoding" a value from/to a byte array).

All elementary types are subclasses of RangeType. Common types are:

<table>
<thead>
<tr>
<th>COBOL Type</th>
<th>Data Simplifier Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC X(n)</td>
<td>AlphanumericType</td>
</tr>
<tr>
<td>PIC 9(n)</td>
<td>ZonedType</td>
</tr>
<tr>
<td>PIC 9(n) COMP-3</td>
<td>PackedType</td>
</tr>
<tr>
<td>PIC 9(n) COMP-5</td>
<td>BinaryType</td>
</tr>
</tbody>
</table>

Aggregate fields

Aggregate fields organize the memory layout of their contents (other aggregates or elementary fields). They do not have an elementary type themselves.

Group fields represent contiguous fields in memory. Each of their contained fields are laid out in the same order in memory, the first field being at offset 0 with respect to the group field position in memory, the second field being at offset 0 + (size in bytes of first field), etc. They are used to represent sequences of COBOL fields under the same containing field.

Union fields represent multiples fields accessing the same memory. Each of their contained fields are laid out at offset 0 with respect to the union field position in memory. They are for example used to represent the COBOL "REDEFINES" construct (the first Union children being the redefined data item, the second children being its first redefinition, etc.).
Array fields (subclasses of Repetition) represent the repetition, in memory, of the layout of their child field (be it an aggregate itself or an elementary item). They lay out a given number of such child layouts in memory, each being at offset \( \text{index} \times (\text{size in bytes of child}) \). They are used to represent COBOL "OCCURS" constructs.

**Primitives**

In some modernization cases, "Primitives" may also used to present independent, "root" data items. Those are very similar in use to RecordEntity but do not heritate from it, nor are based on generated code. Instead they are directly provided by the AWS Blu Age runtime as subclasses of the Primitive interface. Examples of such provided classes are Alphanumeric or ZonedDecimal.

**Data binding and access**

Association between structured data and underlying data can be done in multiple ways.

An important interface for this purpose is RecordAdaptable, which is used to obtain a Record providing a "writable view" on the RecordAdaptable underlying data. As we will see below, multiple classes implement RecordAdaptable. Reciprocally, AWS Blu Age APIs and code manipulating low-level memory (such as programs arguments, file I/O records, CICS commarea, allocated memory...) will often expect a RecordAdaptable as an handle to that memory.

In the COBOL modernization case, most data items are associated with memory which will be fixed during the life time of the corresponding program execution. For this purpose, RecordEntity subclasses are instantiated once in a generated parent object (the program Context), and will take care of instanciating their underlying Record, based on the RecordEntity byte size.

In other COBOL cases, such as associating LINKAGE elements with program arguments, or modernizing the SET ADDRESS OF construct, a RecordEntity instance must be associated with a provided RecordAdaptable. For this purpose, two mechanisms exist:

- if the RecordEntity instance already exists, the `RecordEntity.bind(RecordAdaptable)` method (inherited from Bindable) can be used to make this instance "point" to this RecordAdaptable. Any getter or setter called on the RecordEntity will then be backed (bytes reading or writing) by the underlying RecordAdaptable bytes.
- if the RecordEntity is to be instantiated, a generated constructor accepting a RecordAdaptable is available.

Conversely, the Record currently bound to structured data can be accessed. For this, RecordEntity implements RecordAdaptable, so `getRecord()` can be called on any such instance.

Finally, many COBOL or CICS verbs require access to a single field, for reading or writing purpose. The `RangeReference` class is used to represent such access. Its instances can be obtained from RecordEntity generated `getXXXReference()` methods (XXX being the accessed field), and passed to runtime methods. RangeReference is typically used to access whole RecordEntity or Group, while its subclass `ElementaryRangeReference` represents accesses to Elementary fields.

Note that most observations above apply to Primitive subclasses, since they strive at implementing similar behavior as RecordEntity while being provided by the AWS Blu Age runtime (instead of generated code). For this purpose, all subclasses of Primitive implement RecordAdaptable, ElementaryRangeReference and Bindable interfaces so as to be usable in place of both RecordEntity subclasses and elementary fields.

**FQN of discussed Java types**

The following table shows the fully qualified names of the Java types discussed in this section.
AWS Blu Age Runtime (on Amazon EC2) Setup

This section explains the steps to set up AWS Blu Age Runtime (on Amazon EC2) on your AWS infrastructure.

**Topics**
- [AWS Blu Age Runtime (on Amazon EC2) Prerequisites (p. 73)]
- [AWS Blu Age Runtime (on Amazon EC2) onboarding (p. 73)]
- [Provisioning AWS Blu Age Components (p. 76)]
- [Deploy a modernized application (p. 85)]
- [Upgrade the AWS Blu Age Runtime (on Amazon EC2) version (p. 89)]
- [Setting up licensed dependencies in AWS Blu Age Runtime (on Amazon EC2) (p. 91)]
AWS Blu Age Runtime (on Amazon EC2) Prerequisites

Before you begin the AWS Blu Age Runtime (on Amazon EC2) onboarding process, make sure that you have:

- An AWS account.
- A modernized application.

**Note**
If you want to test the features of AWS Blu Age Runtime (on Amazon EC2), you can use the demo application Planets Demo, which you can download from PlanetsDemo-v1.zip.

AWS Blu Age Runtime (on Amazon EC2) onboarding

To onboard, contact AWS Support to request onboarding to access AWS Blu Age Runtime (on Amazon EC2) in a specific region that you specify. Include your AWS account ID. AWS Support will then contact you to share the link to an Amazon S3 bucket that contains the AWS Blu Age Runtime (on Amazon EC2) AMI and the Velocity framework.

AWS Blu Age Runtime (on Amazon EC2) buckets

We store the AWS Blu Age Runtime (on Amazon EC2) artifacts in different Amazon S3 buckets by Region. To access the bucket for your AWS Region, use the name listed in the following table.

<table>
<thead>
<tr>
<th>AWS Region</th>
<th>Bucket</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (Ohio)</td>
<td>aws-bluage-runtime-artifacts-055777665268-us-east-2</td>
</tr>
<tr>
<td>US East (N. Virginia)</td>
<td>aws-bluage-runtime-artifacts-139023371234-us-east-1</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>aws-bluage-runtime-artifacts-788454048782-us-west-1</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>aws-bluage-runtime-artifacts-836771190483-us-west-2</td>
</tr>
<tr>
<td>EU (Ireland)</td>
<td>aws-bluage-runtime-artifacts-925278190477-eu-west-1</td>
</tr>
</tbody>
</table>

**Note**
Currently, AWS Blu Age Runtime (on Amazon EC2) is available in these five regions only.

Using the command line to list the contents of the bucket

After you are onboarded, you can list the contents of the bucket by running the following command in a terminal.

```bash
aws s3 ls bucket-name
```

Replace `bucket-name` with the name of the bucket for your AWS Region from the previous table.
This command returns a list of folders that correspond to different versions of the AWS Blu Age Runtime (on Amazon EC2) AMI and Velocity framework, such as

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE 3.4.0/</td>
</tr>
<tr>
<td>PRE 3.5.0/</td>
</tr>
</tbody>
</table>

You can choose the version that suits your needs, although it's recommended to use the latest version available. To list the available AMIs for a specific version, run the following command:

```bash
aws s3 ls s3://bucket-name/version/AMI/
```

Replace `bucket-name` with the name of the bucket for your AWS Region and `version` with the version you want.

For example, to list the AMI for the US East (N. Virginia) Region, version 3.4.0, run the following command.

```bash
aws s3 ls s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/AMI/
```

The command will return a list of AMIs, such as:

```
2023-06-05 10:22:23 1384854136 ami-080f7481699b1b9eb.bin
2023-06-05 10:26:50 45 ami-080f7481699b1b9eb.bin.checksumSHA256
```

To List the frameworks, run the following command.

```bash
aws s3 ls s3://bucket-name/version/Framework/
```

Replace `bucket-name` with the name of the bucket for your AWS Region and `version` with the version you want.

The command will return a list of frameworks, such as:

```
2023-06-05 10:26:52 97960225 aws-bluage-runtime-3.5.0.tar.gz
2023-06-05 10:27:12 45 aws-bluage-runtime-3.5.0.tar.gz.checksumSHA256
2023-06-05 10:27:14 138497123 aws-bluage-webapps-3.5.0.tar.gz
2023-06-05 10:27:44 45 aws-bluage-webapps-3.5.0.tar.gz.checksumSHA256
```

**Restoring the AWS Blu Age Runtime (on Amazon EC2) AMI for Amazon EC2 usage**

To create a new Amazon EC2 image based on the AWS Blu Age Runtime (on Amazon EC2) AMI, run the following command.

```bash
aws ec2 create-restore-image-task \
  --object-key version/AMI/ami-id.bin \
  --bucket bucket \
  --name "ec2ImageName"
```
Where:

version

the version of the AMI you chose, such as 3.3.0.2.

ami-id

the ID of the AMI to create an image from. You can obtain this value by running the previous commands.

bucket

the bucket corresponding to the AWS Region where you have been onboarded.

ec2ImageName

a name for the new EC2 image.

For example:

```
aws ec2 create-restore-image-task \
--object-key 3.5.0/AMI/ami-00893e745e7244af4.bin \
--bucket aws-bluage-runtime-artifacts-139023371234-us-east-1 \
--name "MyM2CustomAMI-3.5.0"
```

### Download the framework

You can download the framework for example to upgrade the velocity runtime version on an existing Amazon EC2 instance.

```
aws s3 cp s3://bucket-name/version/Framework/ folder-of-your-choice --recursive
```

Where:

folder-of-your-choice

folder path where you’d like to download the framework.

For example: `aws s3 cp s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.4.0/Framework/ . --recursive`

Output of the command is as follows:

```
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.5.0/Framework/aws-bluage-runtime-3.5.0.tar.gz.checksumSHA256 to ./aws-bluage-runtime-3.5.0.tar.gz.checksumSHA256
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.5.0/Framework/aws-bluage-webapps-3.5.0.tar.gz.checksumSHA256 to ./aws-bluage-webapps-3.5.0.tar.gz.checksumSHA256
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/3.5.0/Framework/aws-bluage-webapps-3.5.0.tar.gz to ./aws-bluage-webapps-3.5.0.tar.gz
```
You can list the framework files as follows:

```bash
ls -l
```

Output of the `ls` command is as follows:

```bash
total 230928
-rw-rw-r-- 1 cloudshell-user cloudshell-user 97960225 Jun  5 10:26 aws-bluage-runtime-3.5.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user  45 Jun  5 10:27 aws-bluage-runtime-3.5.0.tar.gz.checksumSHA256
-rw-rw-r-- 1 cloudshell-user cloudshell-user 138497123 Jun  5 10:27 aws-bluage-webapps-3.5.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user  45 Jun  5 10:27 aws-bluage-webapps-3.5.0.tar.gz.checksumSHA256
```

**Provisioning AWS Blu Age Components**

The infrastructure CDK is an AWS CDK project that you can use to provision AWS resources required for AWS Blu Age Runtime (on Amazon EC2), à la carte; that is, you can provision only the resources you need.

**Topics**

- Getting started (p. 76)
- Prerequisites (p. 77)
- Retrieving the AWS CDK project from Amazon S3 (p. 77)
- Hands on the project (p. 77)
- Configuring the project (p. 77)
- Bootstrapping the AWS CDK (p. 78)
- Deploying (p. 78)
- Connecting to your Amazon EC2 instance (p. 79)
- Destroying deployed stacks (p. 80)
- Parameter reference (p. 80)

**Getting started**

The AWS Blu Age Runtime (on Amazon EC2) infrastructure CDK is a "code as infrastructure" project written in Typescript (a superset of Javascript) and managed by npm.

It works as follows. You provide your input configuration details such as:

- AWS account ID
- AWS Region where you would like to use AWS Blu Age Runtime (on Amazon EC2)
- Your project's name
- Network configuration (VPC, subnets)
- Application configuration (AMI, EC2, key pair, auto-scaling)
- Databases configuration (Business, Blusam, Jics, JHDB, AS400)
And then, depending on the input, AWS CDK will create the required AWS resources.

**Prerequisites**

- AWS CLI, configured with your credentials and AWS Region.
- Node.js version >= 18, with npm
- IDE such as Visual Studio Code or IntelliJ IDEA
- The retrieved and restored AMI from the binary AMI file located in the Amazon S3 bucket that we shared with you during your AWS Blu Age Runtime (on Amazon EC2) onboarding. For more information, see create-restore-image-task.

**Retrieving the AWS CDK project from Amazon S3**

To get started, retrieve the AWS CDK project from the Amazon S3 bucket that we shared with you during your AWS Blu Age Runtime (on Amazon EC2) onboarding. The project is called aws-ba-infrastructure-cdk. Place the project anywhere you like.

**Hands on the project**

Open the project with your favourite IDE and start a terminal window.

Run the following commands in the terminal window from the root of the project's directory, where packages.json is located.

```bash
npm install -g aws-cdk
npm install
```

These commands install the project's dependencies.

**Configuring the project**

After you open the AWS Blu Age infrastructure CDK project with your IDE, you must first configure parameters/inputs.ts.

This file contains some settings related to network, application, databases and so on that you can customize.

**Example:**

```javascript
account: "123456789123", // Your AWS account id
region : "us-east-1", // Region: 'us-east-1', 'eu-west-3', 'ap-southeast-1' ...
projectName : "my-project-name", // Your project's name, in lowercase
execution : { // Execution will allow you to deploy this cdk multiple times in one account, each with a different id.
    name: "ID-",
    id : "15FDX", // Give your next CDK deployment a specific ID (No more than 5 characters)
    useRandomId: false
    // 'true': ignores the provided id and generates a random ID for next execution.
    // 'false': uses the provided id.
},
```
Bootstrapping the AWS CDK

From a terminal, run the following command from the root of the AWS Blu Age CDK project, which will provision the required AWS resources needed for CDK itself.

```bash
cdk bootstrap
```

Deploying

If you specified a runtime execution name and id, or turned on the `useRandomId` flag for your CDK deployment (in `parameters/input.ts`), then we recommend that you run the deployment with the `--output` argument.

Make sure to save the output folder, because you will need it in the future if you wish to update, rollback or destroy that particular deployment.

```bash
cdk deploy --all --output "CustomApp"
```

Otherwise, if you want to deploy everything using the default output, then you can run:

```bash
cdk deploy --all
```

TIP: If you would like to deploy multiple stacks in parallel, you can add the `--concurrency` argument to the command line above. Example: `cdk deploy --all --concurrency 5 --require-approval=never`

For more information, see [AWS CDK Toolkit](https://docs.aws.amazon.com/cdk/latest/guide/). 

After you enter the `cdk deploy` command, the AWS CDK uses your input configuration to create the following stacks:

<table>
<thead>
<tr>
<th>Stack</th>
<th>Required or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWSBANetworkStack</td>
<td>Required</td>
<td>Configures VPC, subnets, endpoints, security groups and so on.</td>
</tr>
<tr>
<td>AWSBAApplicationStack</td>
<td>Required</td>
<td>Configures AMI, Amazon EC2, auto-scaling group, load balancer, ports, SSM parameter store for Amazon EC2, Amazon S3 bucket for your data, and so on.</td>
</tr>
<tr>
<td>AWSBAEFSStack</td>
<td>Optional</td>
<td>Configures Amazon EFS.</td>
</tr>
<tr>
<td>AWSBAElasticacheBlusamLocks</td>
<td>Optional</td>
<td>Configures Elasticache (redis) for Blusam locks.</td>
</tr>
<tr>
<td>AWSBAElasticacheJICS</td>
<td>Optional</td>
<td>Configures Elasticache (redis) for JICS.</td>
</tr>
</tbody>
</table>
### Connecting to your Amazon EC2 instance

Before you start, complete the following steps.

- Install the Session Manager plugin for AWS CLI. For more information, see [Install the Session Manager plugin for the AWS CLI](#).
- Retrieve the id of the instance you would like to connect to.

You can connect to your Amazon EC2 instance in two possible ways:

- Using the Amazon EC2 Console.
- Using Port forwarding and SSH.

You can access your instance with SSH Tunnelling, a feature of Systems Manager *Session Manager*, which is capable of Port Forwarding (Tunneling).

Port Forwarding allows you to securely create tunnels between your instances deployed in private subnets, without the need to start the SSH service on the server, to open the SSH port in the security group or the need to use a bastion host.

**Note**
For more information on SSH Tunnelling, see [New – Port Forwarding Using AWS System Manager Session Manager](#).

Next, retrieve the private key required to connect to your instance.

When you created the `_input_` configuration file, you either provided a custom key pair or allowed the CDK to generate one for you.

If the AWS CDK generated the key pair, you can use the AWS Management Console to access it by completing the following steps:

1. Open the *Systems Manager* console and choose *Parameter Store*. 

### Provisioning AWS Blu Age Components

<table>
<thead>
<tr>
<th>Stack</th>
<th>Required or Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWSBAElasticacheJICSTSSoap</td>
<td>Optional</td>
<td>Configures Elasticache (redis) for JICS TS Queues.</td>
</tr>
<tr>
<td>AWSBAElasticacheHTTPSsessions</td>
<td>Optional</td>
<td>Configures Elasticache (redis) for HTTPS session (online applications).</td>
</tr>
<tr>
<td>AWSBAElasticacheDatasetCatalog</td>
<td>Optional</td>
<td>Configures Elasticache (redis) for Dataset Catalog.</td>
</tr>
<tr>
<td>AWSBAauroraDB-Business</td>
<td>Optional</td>
<td>Configures Aurora Database (postgres) for Business.</td>
</tr>
<tr>
<td>AWSBAauroraDB-Blusam</td>
<td>Optional</td>
<td>Configures Aurora Database (postgres) for Blusam.</td>
</tr>
<tr>
<td>AWSBAauroraDB-JHDB</td>
<td>Optional</td>
<td>Configures Aurora Database (postgres) for JHDB.</td>
</tr>
<tr>
<td>AWSBAauroraDB-JICS</td>
<td>Optional</td>
<td>Configures Aurora Database (postgres) for JICS.</td>
</tr>
</tbody>
</table>
2. Find parameter */AWS-BA-VPC-ProjectName-Region/AWSBAApplication-ID/EC2-KeyPair.
3. Note the Value.
4. Return to Parameter Store.
5. Find parameter /ec2/keypair/Value.
6. Click and Reveal Value.
7. Save your key locally.

Next, open a terminal window and run the following command:

```bash
aws ssm start-session --target InstanceId \
    --document-name AWS-StartPortForwardingSession \
    --parameters '{"portNumber":"SshPortNumber","localPortNumber":"LocalPortNumber"}''
```

Replace InstanceId with your instance id, SshPortNumber with the SSH port (usually 22) and LocalPortNumber with any free port on your local machine.

For example:

```bash
aws ssm start-session --target i-012ab3456123456c0 \
    --document-name AWS-StartPortForwardingSession \
    --parameters '{"portNumber":"22","localPortNumber":"5689"}''
```

Finally, use an SSH client to link your private key and connect to your Amazon EC2 instance through localhost at port LocalPortNumber with user root or ec2-user.

**Destroying deployed stacks**

If you want to destroy one or more deployed stacks, run:

```
cdk destroy StackName1 StackName2 ...
```

**Note**
This command only works if you deployed the CDK without the --output argument, and therefore the default cdk.out folder is created for you, which will be used here.

If you have specified a custom output (i.e using --output argument) for your CDK deployment, then you need to run the following command instead:

```
cdk destroy --app "CustomApp" StackName1 StackName2 ...
```

**TIP**: You do not have to explicitly specify every stack name. You can directly use the --all argument instead.

**Parameter reference**

The following table contains all configurable input parameters, expressed as key-value pairs, both required and optional. All parameters in bold are required.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>account</strong></td>
<td>Your AWS account id.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>region</strong></td>
<td>The region where you would like to deploy the AWS Blu Age infrastructure CDK.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>projectName</strong></td>
<td>Your project’s name.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>execution</strong></td>
<td>Allows you to deploy the CDK multiple times in one AWS account. You might want to do this to set up separate environments for DEV, QA, and PROD, for example. Provide an execution name and value if you wish, or use a randomly generated execution value.</td>
<td>Custom ID</td>
</tr>
<tr>
<td><strong>vpcCidr</strong></td>
<td>VPC CIDR block. For more information, see <a href="#">VPC CIDR blocks</a>.</td>
<td>10.190.0.0/16</td>
</tr>
<tr>
<td><strong>subnetPublicName</strong></td>
<td>Name of the public subnet that is associated to the VPC.</td>
<td>public</td>
</tr>
<tr>
<td><strong>subnetApplicationName</strong></td>
<td>Name of the private application subnet that is associated to the VPC.</td>
<td>isolated-app</td>
</tr>
<tr>
<td><strong>subnetDatabaseName</strong></td>
<td>Name of the private database subnet that is associated to the VPC.</td>
<td>isolated-db</td>
</tr>
<tr>
<td><strong>ec2AppInstanceClass</strong></td>
<td>Amazon EC2 instance class type. For a complete list, see <a href="#">enum InstanceClass</a> in the AWS CDK documentation.</td>
<td>st3</td>
</tr>
<tr>
<td><strong>ec2AppInstanceSize</strong></td>
<td>Amazon EC2 instance class size. For a complete list, see <a href="#">enum InstanceSize</a> in the AWS CDK documentation.</td>
<td>large</td>
</tr>
<tr>
<td><strong>ec2AppInstanceMin</strong></td>
<td>Minimum number of active Amazon EC2 instances, managed by the Auto-Scaling group.</td>
<td>1</td>
</tr>
<tr>
<td><strong>ec2AppInstanceMax</strong></td>
<td>Maximum number of active Amazon EC2 instances, managed by the Auto-Scaling group.</td>
<td>2</td>
</tr>
<tr>
<td><strong>ec2AMI</strong></td>
<td>The AMI id to be used when a new Amazon EC2 instance is launched. This is the result of the AMI restore command described in <em>Prerequisites</em>.</td>
<td>N/A</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>useCustomKeypairForEc2Velocity</td>
<td>Specifies whether to use an already predefined key pair or generate a new one and store it in an SSM parameter store. Directly linked with key customKeypairNameForEc2Velocity.</td>
<td>false</td>
</tr>
<tr>
<td>customKeypairNameForEc2Velocity</td>
<td>The name of the Amazon EC2 key pair to use when launching a new Amazon EC2 instance.</td>
<td>N/A</td>
</tr>
<tr>
<td>s3UserDataBucket</td>
<td>The name of the Amazon S3 bucket which is created for you and where your data will be uploaded. You can put your data in the CDK project's folder parameters/userdata/.</td>
<td>aws-ba-user-data-bucket</td>
</tr>
<tr>
<td>internetFacingLoadBalancer</td>
<td>Whether or not to expose the load balancer to the internet. Setting this to true will make the load balancer accessible from outside the VPC.</td>
<td>false</td>
</tr>
<tr>
<td>Application</td>
<td>Whether or not to deploy the application layer stack (EC2, ALB, ASG, ...etc.)</td>
<td>false</td>
</tr>
<tr>
<td>EfsFileSystem</td>
<td>Whether or not to deploy an EFS File System layer stack.</td>
<td>false</td>
</tr>
<tr>
<td>RedisDatasetCatalog</td>
<td>Whether or not to use Redis as Catalog storage.</td>
<td>false</td>
</tr>
<tr>
<td>RedisDatasetCatalogNbShard</td>
<td>For Dataset Catalog: Number of shards to be used by the Redis replication group.</td>
<td>cache.t3.micro</td>
</tr>
<tr>
<td>RedisDatasetCatalogNbReplicaPerShard</td>
<td>For Dataset Catalog: Number of replica nodes in each shard.</td>
<td>0</td>
</tr>
<tr>
<td>RedisDatasetCatalogcacheNodeType</td>
<td>For Dataset Catalog: The compute and memory capacity of the nodes in the shard.</td>
<td>cache.t3.micro</td>
</tr>
<tr>
<td>RedisHttpSsessions</td>
<td>Whether or not to use Redis to store HTTP sessions data.</td>
<td>false</td>
</tr>
<tr>
<td>RedisHttpSsessionsNbShard</td>
<td>For HTTPS sessions: Number of shards to be used by the Redis replication group.</td>
<td>2</td>
</tr>
<tr>
<td>RedisHttpSsessionsNbReplicaPerShard</td>
<td>HTTPS sessions: Number of replica nodes in each shard.</td>
<td>2</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>RedisHttpSsessionsCacheNodeType</td>
<td>For HTTPS sessions: The compute and memory capacity of the nodes in the shard.</td>
<td>cache.t3.micro</td>
</tr>
<tr>
<td>AuroraPostgresqlUsername</td>
<td>Common Aurora: Database secret username.</td>
<td>clusteradmin</td>
</tr>
<tr>
<td>AuroraPostgresqlPassword</td>
<td>Common Aurora: Database secret password.</td>
<td>password</td>
</tr>
<tr>
<td>AuroraPostgresqlPrimary</td>
<td>Whether or not to create a primary relational database.</td>
<td>false</td>
</tr>
<tr>
<td>AuroraPostgresqlPrimaryInstanceClass</td>
<td>Business (Aurora): Database instance class.</td>
<td>t3</td>
</tr>
<tr>
<td>AuroraPostgresqlPrimaryInstanceSize</td>
<td>Business (Aurora): Database instance size.</td>
<td>medium</td>
</tr>
<tr>
<td>AuroraPostgresqlAs400</td>
<td>Whether or not to create a relation database from AS/400 modernization.</td>
<td>false</td>
</tr>
<tr>
<td>AuroraPostgresqlAs400InstanceClass</td>
<td>For AS400 (Aurora): Database instance class.</td>
<td>t3</td>
</tr>
<tr>
<td>AuroraPostgresqlAs400InstanceSize</td>
<td>For AS400 (Aurora): Database instance size.</td>
<td>medium</td>
</tr>
<tr>
<td>AuroraPostgresqlJhdb</td>
<td>Whether or not to create a relation database for JHDB.</td>
<td>false</td>
</tr>
<tr>
<td>AuroraPostgresqlJhdbInstanceClass</td>
<td>For JHDB (Aurora): Database instance class.</td>
<td>t3</td>
</tr>
<tr>
<td>AuroraPostgresqlJhdbInstanceSize</td>
<td>For JHDB (Aurora): Database instance size.</td>
<td>medium</td>
</tr>
<tr>
<td>AuroraPostgresqlBluesam</td>
<td>Whether or not to use Aurora Postgresql database to store Bluesam data.</td>
<td>false</td>
</tr>
<tr>
<td>AuroraPostgresqlBluesamInstanceClass</td>
<td>For Bluesam (Aurora): Database instance class.</td>
<td>t3</td>
</tr>
<tr>
<td>AuroraPostgresqlBluesamInstanceSize</td>
<td>For Bluesam (Aurora): Database instance size.</td>
<td>medium</td>
</tr>
<tr>
<td>RedisBlusamCache</td>
<td>Whether or not to use Redis to cache Bluesam data (required if Bluesam is selected).</td>
<td>false</td>
</tr>
<tr>
<td>RedisBlusamCacheNbShard</td>
<td>For Bluesam Cache: Number of shards to be used by the Redis replication group.</td>
<td>2</td>
</tr>
<tr>
<td>RedisBlusamCacheNbReplicaPerShard</td>
<td>For Bluesam Cache: Number of replica nodes in each shard.</td>
<td>1</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>RedisBlusamCacheCacheNodeType</td>
<td>For Blusam Cache: The compute and memory capacity of the nodes in the shard.</td>
<td>cache.t3.micro</td>
</tr>
<tr>
<td>RedisBlusamLocks</td>
<td>Whether or not to use Redis to store Blusam locks.</td>
<td>false</td>
</tr>
<tr>
<td>RedisBlusamLocksNbShard</td>
<td>For JICS TS Queues: Number of shards to be used by the Redis replication group.</td>
<td>2</td>
</tr>
<tr>
<td>RedisBlusamLocksNbReplicaPerShard</td>
<td>For JICS TS Queues: Number of replica nodes in each shard.</td>
<td>1</td>
</tr>
<tr>
<td>RedisBlusamLocksCacheNodeType</td>
<td>For JICS TS Queues: The compute and memory capacity of the nodes in the shard.</td>
<td>cache.t3.micro</td>
</tr>
<tr>
<td>RedisJics</td>
<td>Whether or not to use Redis to store JICS data. (Option 1)</td>
<td>false</td>
</tr>
<tr>
<td>RedisJicsNbShard</td>
<td>For JICS: Number of shards to be used by the Redis replication group.</td>
<td>2</td>
</tr>
<tr>
<td>RedisJicsNbReplicaPerShard</td>
<td>For JICS: Number of replica nodes in each shard.</td>
<td>1</td>
</tr>
<tr>
<td>RedisJicsCacheNodeType</td>
<td>For JICS: The compute and memory capacity of the nodes in the shard.</td>
<td>cache.t3.micro</td>
</tr>
<tr>
<td>AuroraPostgresqlJics</td>
<td>Whether or not to use Aurora Postgresql to store JICS data. (Option 2)</td>
<td>false</td>
</tr>
<tr>
<td>AuroraPostgresqlJicsInstanceClass</td>
<td>For JICS (Aurora): Database instance class.</td>
<td>t3</td>
</tr>
<tr>
<td>AuroraPostgresqlJicsInstanceSize</td>
<td>For JICS (Aurora): Database instance size.</td>
<td>medium</td>
</tr>
<tr>
<td>RedisJicsTSQueues</td>
<td>Whether or not to use Redis to store JICS TS Queues data.</td>
<td>false</td>
</tr>
<tr>
<td>RedisJicsTSQueuesNbShard</td>
<td>For JICS TS Queues: Number of shards to be used by the Redis replication group.</td>
<td>2</td>
</tr>
<tr>
<td>RedisJicsTSQueuesNbReplicaPerShard</td>
<td>JICS TS Queues: Number of replica nodes in each shard.</td>
<td>1</td>
</tr>
<tr>
<td>RedisJicsTSQueuesCacheNodeType</td>
<td>JICS TS Queues: The compute and memory capacity of the nodes in the shard.</td>
<td>cache.t3.micro</td>
</tr>
</tbody>
</table>
Important
If you have data (applications, files, configurations, etc.) that you would like to set up before using the AWS Blu Age runtime engine, you can place everything you need in the parameters/userdata/folder and modify startup.sh as needed. Your data will be uploaded to an Amazon S3 bucket which is created for you. Also, this script will be executed every time a new Amazon EC2 is launched.

Deploy a modernized application

This guide explains how to set up and deploy the PlanetsDemo application on an AWS Blu Age Runtime (on Amazon EC2) AMI. You can use these same steps for your modernized applications.

Topics
• Prerequisites (p. 85)
• Setting up (p. 85)
• Test the PlanetsDemo application (p. 87)

Prerequisites
Before you begin, make sure you complete the following prerequisites.

• Complete AWS Blu Age Runtime (on Amazon EC2) Setup (p. 72).
• Create an Amazon EC2 instance based on the latest AWS Blu Age Runtime (on Amazon EC2) AMI. For more information, see Get started with Amazon EC2 Linux instances.
• Make sure you can connect to the Amazon EC2 instance successfully, for example by using SSM.
• Download the PlanetsDemo application archive.

Setting up
To set up the PlanetsDemo sample application, complete the following steps.

1. Unzip the archive and upload the application to an Amazon S3 bucket of your choice.
2. Connect to your Amazon EC2 instance and change the user to su by running the following command.

   ```bash
   sudo su
   ```

   You need Superuser privilege to run commands in this tutorial.

3. Check the status of the Linux services tomcat.service and tomcat-webapps.service by running the following commands.

   ```bash
   systemctl status tomcat.service
   systemctl status tomcat-webapps.service
   ```

4. If the services are running, stop them by running the following commands.

   ```bash
   systemctl stop tomcat.service
   systemctl stop tomcat-webapps.service
   ```
5. Navigate to the /m2-anywhere/tomcat.gapwalk/velocity/webapps folder.

   ```
   cd /m2-anywhere/tomcat.gapwalk/velocity/webapps
   ```

6. Configure the AWS CLI by following the steps in Configuring the AWS CLI.

7. Copy the PlanetsDemo binaries available at PlanetsDemo-v1/webapps/ folder from the Amazon S3 bucket using the following command.

   ```
   aws s3 cp s3://path-to-demo-app-webapps/ . --recursive
   ```

   **Note**
   Replace `path-to-demo-app-webapps` with the correct Amazon S3 URI for the bucket where you previously unzipped the PlanetsDemo archive.

8. Copy the content of PlanetsDemo-v1/config/ folder to /m2-anywhere/tomcat.gapwalk/velocity/config/.

9. Create an Aurora PostgreSQL database for JICS and provide the connection information in the following snippet in the application-main.yml. For more information see, Creating and connecting to an Aurora PostgreSQL DB cluster.

   ```yaml
   datasource:
   jicsDs:
     driver-class-name:
     url:
     username:
     password:
     type:
   ```


11. Start the tomcat services by running the following commands.

   ```
   systemctl start tomcat.service
   systemctl start tomcat-webapps.service
   ```

12. Check the status of the services to make sure they are running by running the following commands.

   ```
   systemctl status tomcat.service
   systemctl status tomcat-webapps.service
   ```

   To check the status of the deployed PlanetsDemo application, run the following commands.

   ```
   curl http://localhost:8080/gapwalk-application/
   ```

   The following message should display.

   ```
   Jics application is running
   ```
curl http://localhost:8181/jac/api/services/rest/jicsservice/

The following message should display.

Jics application is running

curl http://localhost:8181/bac/api/services/rest/bluesamserver/serverIsUp

The response should be empty.

13. Note the name of the web binary (PlanetsDemo-web-1.0.0, if unchanged). To access the PlanetsDemo application, use a URL of the following format.

https://load-balancer-DNS-name:listener-port/web-binary-name

Note

Replace load-balancer-DNS-name, listener-port and web-binary-name with the correct values for your setup.

Test the PlanetsDemo application

After the PlanetsDemo application starts, the home page is displayed.

Enter PINQ in the text box and then press Enter. The data inquiry page is displayed.
Enter EARTH, for example, in PlanetsDemo name field and then press Enter. The page for the planet you entered is displayed.
Upgrade the AWS Blu Age Runtime (on Amazon EC2) version

This guide describes how to upgrade the AWS Blu Age Runtime (on Amazon EC2) version.

Topics
- Prerequisites (p. 89)
- Upgrade the velocity runtime (p. 90)

Prerequisites

Before you begin, make sure you meet the following prerequisites.

- Complete AWS Blu Age Runtime (on Amazon EC2) Setup (p. 72).
- Create an Amazon EC2 instance based on the latest AWS Blu Age AMI. For more information, see Get started with Amazon EC2 Linux instances.
- Make sure you can connect to the Amazon EC2 instance successfully, for example by using SSM.
- Download the version of the AWS Blu Age Runtime framework you'd like to upgrade to. For more information, see AWS Blu Age Runtime (on Amazon EC2) Setup (p. 72)
of two binary files: \texttt{gapwalk-x.x.x.x-m2-anywhere.tar.gz} and \texttt{webapps-x.x.x.x-m2-anywhere.tar.gz}.

### Upgrade the velocity runtime

Complete the following steps to upgrade the velocity runtime.

1. Connect to your Amazon EC2 instance and change the user to \texttt{su} by running the following command.
   
   ```bash
   sudo su
   ```
   
   You need Superuser privilege to run commands in this tutorial.

2. Create two folders, one for each binary file.
3. Name each folder with the same name as the binary file.
4. Copy each binary file to the corresponding folder.

   **Warning**
   
   Extracting each binary produces a folder with the same name so if you extract both binary files at the same location one after another you will overwrite the content.

5. To extract the binaries, use the following commands. Run the commands in each folder.

   ```bash
   tar xvf aws-bluage-runtime-x.x.x.x.tar.gz
   tar xvf aws-bluage-webapps-x.x.x.x.tar.gz
   ```

6. Stop the Tomcat services using the following commands.

   ```bash
   systemctl stop tomcat.service
   systemctl stop tomcat-webapps.service
   ```

7. Replace the content of \texttt{/m2-anywhere/tomcat.gapwalk/velocity/shared/} with the content of \texttt{aws-bluage-runtime-x.x.x.x/velocity/shared/}.
8. Replace \texttt{/m2-anywhere/tomcat.gapwalk/velocity/webapps/gapwalk-application.war} with \texttt{aws-bluage-runtime-x.x.x.x/velocity/webapps/gapwalk-application.war}.
9. Replace the war files in \texttt{/m2-anywhere/tomcat.webapps/velocity/webapps/}, namely \texttt{bac.war} and \texttt{jac.war}, with the same files from \texttt{aws-bluage-webapps-x.x.x.x/velocity/webapps/}.
10. Start the tomcat services by running the following commands.

    ```bash
    systemctl start tomcat.service
    systemctl start tomcat-webapps.service
    ```

11. Check status of the services to make sure they are running by running the following commands.
To check the status of the deployed application, run the following commands.

```bash
systemctl status tomcat.service
systemctl status tomcat-webapps.service
```

To check the status of the deployed application, run the following commands.

```bash
curl http://localhost:8080/gapwalk-application/
```

The following message should display.

```bash
Jics application is running
```

```bash
curl http://localhost:8181/jac/api/services/rest/jicsservice/
```

The following message should display.

```bash
Jics application is running
```

```bash
curl http://localhost:8181/bac/api/services/rest/bluesamserver/serverIsUp
```

The response should be empty.

The AWS Blu Age runtime is successfully upgraded.

### Setting up licensed dependencies in AWS Blu Age Runtime (on Amazon EC2)

This guide describes how to set up additional licensed dependencies that you can use with AWS Blu Age Runtime (on Amazon EC2).

**Topics**

- Prerequisites (p. 91)
- Overview (p. 92)
- Set up the dependencies for JAC and BAC webapps (p. 93)

**Prerequisites**

Before you begin, make sure you complete the following prerequisites.

- Complete AWS Blu Age Runtime (on Amazon EC2) Setup (p. 72).
- Create an Amazon EC2 instance based on the latest AWS Blu Age Runtime (on Amazon EC2) AMI. For more information, see Get started with Amazon EC2 Linux instances.
- Make sure you can connect to the Amazon EC2 instance successfully, for example by using SSM.
- Get the following dependencies from their source.

**Oracle database**

Supply an Oracle database driver. We tested the AWS Blu Age Runtime (on Amazon EC2) functionality with version ojdbc8-19.8.0.0.jar, but a more recent version might be compatible.
IBM MQ connection

Supply an IBM MQ client. We tested the AWS Blu Age Runtime (on Amazon EC2) functionality with version com.ibm.mq.allclient-9.1.5.0.jar, but a more recent version might be compatible.

With this dependency version, also supply the following transitive dependencies:

- javax.jms-api-2.0.1.jar
- json-20080701.jar

DDS Printer files

Supply the Jasper reports library. We tested the AWS Blu Age Runtime (on Amazon EC2) functionality with jasperreports-6.16.0.jar, but a more recent version might be compatible.

With this dependency version, also supply the following transitive dependencies:

- 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.js.jar
- javax.inject-1.jar
- jcommon-1.0.23.jar
- jfreechart-1.0.19.jar

Overview

To install the dependencies, complete the following steps.

1. Connect to your Amazon EC2 instance and change the user to su by running the following command.

   ```
   sudo su
   ```

   You need Superuser privilege to run commands in this tutorial.

2. Navigate to the /m2-anywhere/tomcat.gapwalk/velocity/extra/ folder.

   ```
   cd /m2-anywhere/tomcat.gapwalk/velocity/extra/
   ```

3. Copy any of the above dependencies as required at this folder.

4. Stop and start the tomcat.service by running the following commands.

   ```
   systemctl stop tomcat.service
   ```

   ```
   systemctl start tomcat.service
   ```

5. Check the status of the service to make sure it is running.

   ```
   systemctl status tomcat.service
   ```
Set up the dependencies for JAC and BAC webapps

1. If your JICS or Blusam database is hosted on Oracle then you need to provide the Oracle database driver in */m2-anywhere/tomcat.webapps/velocity/extra*
2. Create the folder if it is not present already.
3. Stop and start the service by running the following commands.

```bash
systemctl stop tomcat-webapps.service
systemctl start tomcat-webapps.service
```

AWS Blu Age Runtime (on Amazon EC2) Configuration

The Velocity framework and the client code are web applications using the Spring Boot framework. It leverages Spring capabilities to supply configuration, with several possible locations and precedence rules. There are also similar precedence rules for supplying many other files, such as groovy scripts, sql, etc.

The Velocity framework also contains additional optional web applications, that can be opted-in if needed.

Topics

- Application configuration basics (p. 93)
- Application precedence (p. 95)
- JNDI for databases (p. 95)
- Using AWS secrets (p. 95)
- Other files (groovy, sql, etc.) (p. 97)
- Additional web application (p. 97)
- Enabling properties (p. 98)
- Configure authentication for Gapwalk applications (p. 119)

Application configuration basics

The default way to handle application configuration is through the use of dedicated yaml files to be supplied in the application server's config folder. There are two main yaml configuration files:

- application-main.yml
- application-*profile*.yml (where profile value is setup during application generation).

The first file configures the framework, i.e. Gapwalk-application.war, while the second one is for additional options specifically for the client application. This works with the use of spring profiles: the Gapwalk application uses the main profile, while the client application uses the profile profile.

The following example shows a typical main YAML file.
The following example shows a typical client YAML file.

```yaml
### 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
    unita behind
```

For information about the content of the YAML files, see [Enabling properties (p. 98)](#).
Application precedence

For these configuration files, Spring precedence rules apply. Notably:

- The application-main yaml file appears in the Gapwalk main war file with default values, and the one in the config folder supersedes it.
- The same should be done for the client application configuration
- Additional parameters may be passed on the command line at server launch time. They would override the yaml ones.

For more information, see Official Spring Boot documentation.

JNDI for databases

The database configuration might be supplied with JNDI in the context.xml file in Tomcat. Any such configuration would override the yaml one. But pay attention that using this will not allow to wrap your credentials in a secret manager (see below).

The following example shows sample configurations for jics and blusam databases.

```
<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

Would be jdbc/jics for the jics database and jdbc/bluesam (pay attention to the 'e') for the blusam database.

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

The database url, username and password.

Using AWS secrets

Some of the resource configurations that contain credentials can be further secured by using AWS secrets. The idea is to store critical data in an AWS secret and have a reference to the secret in the yml configuration so the secret content is picked on the fly at tomcat startup.

Secrets for Aurora

Aurora database configuration (for jics, blusam, customer db, etc) will use the built-in database secret, which will populate all the relevant fields automatically from the corresponding database.

**Note**

The dbname key is optional, depending on your database configuration, it will get into the secret or not. You may add it there manually, or by supplying the name to the yaml file.

Other secrets

Other secrets are for resources that have a single password (notably password-protected redis caches). In this case the other type of secret must be used, with a single password key.
YAML references to secrets

The application-main.yml can reference the secret arn for various resources. The most important ones are:

- Jics database credentials with spring.aws.jics.db.secret
- Blusam database credentials with spring.aws.client.bluesam.db.secret
- Blusam cache password with spring.aws.client.bluesam.redis.secret
- Blusam locks cache password with spring.aws.client.bluesam.locks.redis.secret

The following example shows how to declare these secrets in a YAML file.

```yaml
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:
        db:
          secret: arn:aws:secretsmanager:XXXX
```

dbname: bluesam

In this example, the name of the database is not in the secret and is supplied here instead.

The client application-profile.yml can reference the secret ARN for the client database. This requires an additional property to list the datasources, showcased in the example below:

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

names: primary,host

An example with two client datasources named primary and host, each with their database and credentials.

dbname: mydb

In this example, the name of the "host" database is not in the secret and is supplied here instead, while for the "primary" database it is in the secret.
Note
If the client database is using XA, the regular yaml key
spring.jta.atomikos.datasource.XXXX.unique-resource-name must still be supplied
in addition to the above ones.

Other files (groovy, sql, etc.)

The other files used by the customer project use similar precedence rules as the ones for spring
configuration. Examples:

- Groovy scripts are .groovy files in the scripts folder or subfolders.
- SQL scripts are .sql files in the sql folder or subfolders.
- Daemon scripts are .groovy files in the daemons folder or subfolders.
- Queries Database mapping file are files named queries-database.mapping files in the sql folder
  subfolders.
- Jasper templates are .jrxml files in the templates folder or subfolders.
- Dataset catalogs are .json files in the catalog folder.
- Lnk files are .json files in the lnk folder.

All these locations can be overridden by way of a system property.

- For Groovy scripts: configuration.scripts
- For SQL scripts: configuration.sql
- For Daemon scripts: configuration.daemons
- For Queries Database mapping file: configuration.databaseMapping
- For Jasper templates: configuration.templates
- For Dataset catalogs: configuration.catalog
- For Lnk files: configuration.lnk

If the system property is not found, the files will be taken from the default location mentioned above.
The lookup will first be done with the tomcat root folder as a root, and lastly in the application war file.

Additional web application

The Velocity framework contains additional web applications in its webapps-extra folder. These
applications are not served by default by the tomcat server.

Opting-in to these web applications is modernization project dependent and is done by moving the
desired war file from the webapps-extra folder to the webapps folder. After that, the war will be
served by the tomcat server at next startup.

Some project-specific additional configuration can also be added in a yaml configuration file for each
additional war, as is done in the application-main.yml file and explained above. The additional wars are:

- gapwalk-utility-pgm.war: contains support for ZOS utility programs and uses application-
  utility-pgm.yml as its configuration.
- gapwalk-cl-command.war: contains support for AS/400 utility programs and uses application-
  cl-command.yml as its configuration.
Enabling properties

In Spring Boot applications application-main.yml is the configuration file in which we define different kinds of properties such as the listening port, database connectivity... and many more.

Topics

- **YML notation** (p. 98)
- **Quick start / Use cases** (p. 98)
- **Available properties for the main application** (p. 102)
- **Available properties for optional web applications** (p. 113)

YML notation

In the following documentation, a property such as parent.child1.child2=true is written as follows in YML format.

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

Quick start / Use cases

The following use cases show examples of the applicable keys and values.

- Default application-main.yml

```yaml
----
##### DEFAULT APPLICATION-MAIN.YML FILE #####
##### SHOWING USEFUL CONFIGURATION ELEMENTS #####
##### SHOULD BE OVERRIDEN 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 : postgresql #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.eibtmid: TERM
#jics.parameters.userid: 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 #####
####################################################

100
#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
scheduled: # list of schedulers
- name: queue1
  threadCount: 5
- name: queue2
  threadCount: 5
# Use variable length files with LISTCAT commands

```plaintext
[**/*.*]
enencoding=IBM930
reencoding=false

[globals]
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)
```

### Provide Null Byte Indicator Value in LOAD/UNLOAD utility

```plaintext
# 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
```

### Available properties for the main application

This table provides an exhaustive view of key/values parameters.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging.config</td>
<td>Path</td>
<td>classpath:logback-main.xml</td>
<td>Standard key for the reference to the logback configuration file. Other standard logging keys are available too.</td>
</tr>
</tbody>
</table>
## Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spring.jta.enabled</td>
<td>boolean</td>
<td>false</td>
<td>Standard key. If the datasource support mode is not static-xa, spring JTA transactions autoconfiguration must be disabled.</td>
</tr>
<tr>
<td>datasource.jicsDs + -driver-class-name + -url + -username + -password + -type</td>
<td>Standard spring datasource with subkeys</td>
<td>Contains the connection information for the Jics database. Alternately, use of AWS secrets is strongly encouraged, as explained in <a href="../configuration/configuration.adoc">Configuration</a>.</td>
<td></td>
</tr>
<tr>
<td>datasource.bluesamDs + -driver-class-name + -url + -username + -password + -type</td>
<td>Standard spring datasource with subkeys</td>
<td>Contains the connection information for the Blusam database. Alternately, use of AWS secrets is strongly encouraged, as explained in <a href="../configuration/configuration.adoc">Configuration</a>.</td>
<td></td>
</tr>
<tr>
<td>bluesam.disabled</td>
<td>boolean</td>
<td>false</td>
<td>Whether to disable blusam entirely.</td>
</tr>
<tr>
<td>bluesam.cache</td>
<td>string</td>
<td></td>
<td>If not set, the blusam cache will not be used. Possible values (cache implementations) are ehcache and redis.</td>
</tr>
<tr>
<td>forcedDate</td>
<td>string</td>
<td></td>
<td>Forces the date to the date provided if there is one.</td>
</tr>
<tr>
<td>frozenDate</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to freeze the date. Applies only if forcedDate is also set.</td>
</tr>
<tr>
<td>externalSort.threshold</td>
<td>datasize (ex. 12MB)</td>
<td></td>
<td>The sort threshold: when to switch to external (merge) sort.</td>
</tr>
<tr>
<td>jics.parameters.datatype</td>
<td>string</td>
<td>MMDDYY</td>
<td>The date form.</td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jics.initList`</td>
<td>string</td>
<td></td>
<td>The initialize jics list, separated by commas. If present, it defines comma separated names of lists to activate at tomcat startup among CICS lists. Example value: $UUU, DFH $IVPL, PEZ1. This will cascade to groups contained in those lists and their underlying resource definitions, which will then be visible to the runtime. Empty by default.</td>
</tr>
<tr>
<td>jics.parameters.applid</td>
<td>string</td>
<td>VELOCITY</td>
<td>The applid to identify application in JICS (at least 4 characters, no max length).</td>
</tr>
<tr>
<td>jics.parameters.sysid</td>
<td>string</td>
<td>CICS</td>
<td>The system identification (SYSID).</td>
</tr>
<tr>
<td>jics.parameters.eibtrmid</td>
<td>string</td>
<td>TERM</td>
<td>The terminal identifier (4 characters maximum, 1 minimum).</td>
</tr>
<tr>
<td>jics.parameters.userid</td>
<td>string</td>
<td></td>
<td>The user id (8 characters maximum, no minimum). When no value is provided (blank by default) the HTTP session id is used as the user id.</td>
</tr>
<tr>
<td>jics.parameters.username</td>
<td>string</td>
<td>MYUSERNAME</td>
<td>The username (10 characters maximum, 1 minimum).</td>
</tr>
<tr>
<td>jics.parameters.netname</td>
<td>string</td>
<td>MYNETNAME</td>
<td>The network name (8 characters maximum, 1 minimum).</td>
</tr>
<tr>
<td>jics.parameters.opid</td>
<td>string</td>
<td>XXX</td>
<td>The 3-character operator identification.</td>
</tr>
<tr>
<td>jics.parameters.jobname</td>
<td>string</td>
<td>MJOBNAME</td>
<td>The jobname.</td>
</tr>
<tr>
<td>jics.parameters.sysname</td>
<td>string</td>
<td>SYSNAME</td>
<td>The AS400 system name (sysname).</td>
</tr>
<tr>
<td>jics.parameters.cwa.length</td>
<td>number</td>
<td>0</td>
<td>The common work area (cwa) length.</td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jics.parameters.charset</td>
<td>string</td>
<td>CP037</td>
<td>JICS globally used character set.</td>
</tr>
<tr>
<td>jics.parameters.tsqimpl</td>
<td>string</td>
<td>bluesam</td>
<td>JICS Temporary Storage Queue (TSQ) implementation (allowed values are bluesam/memoty/redis)</td>
</tr>
<tr>
<td>jics.queues.ts.redis.hostname</td>
<td>string</td>
<td>127.0.0.1</td>
<td>The hostname of the jics cache redis server.</td>
</tr>
<tr>
<td>jics.queues.ts.redis.port</td>
<td>number</td>
<td>6379</td>
<td>The port of the jics cache redis server.</td>
</tr>
<tr>
<td>jics.queues.ts.redis.password</td>
<td>string</td>
<td>redis</td>
<td>The password for the jics cache redis server.</td>
</tr>
<tr>
<td>jics.queues.ts.redis.username</td>
<td>string</td>
<td></td>
<td>Th username for the jics cache redis server. Default is blank (no username).</td>
</tr>
<tr>
<td>jics.queues.ts.redis.mode</td>
<td>string</td>
<td>standalone</td>
<td>The jics cache mode. Possible values are standalone or cluster. Default is standalone.</td>
</tr>
<tr>
<td>lockTimeout</td>
<td>number</td>
<td>500</td>
<td>The lock timeout, in milliseconds.</td>
</tr>
<tr>
<td>sqlCodePointShift</td>
<td>number</td>
<td></td>
<td>Optional. The sql code point shift. Shifts the codepoint for control characters that we might encounter when migrating legacy rdbms data to a modern rdbms. For example, you could specify 384 to match unicode character \u0180.</td>
</tr>
<tr>
<td>sqlIntegerOverflowAllowed</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to allow the SQL integer overflow, meaning whether placing larger values in the host variable is allowed.</td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>database.cursor.overflow.allowed</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to allow the cursor overflow. Set to true to perform a next call on the cursor whatever its position. Set to false to check whether the cursor is at the last position before performing a next call on cursor. Only enable if cursor is SCROLLABLE (SENSITIVE or INSENSITIVE).</td>
</tr>
<tr>
<td>reportOutputPath</td>
<td>string</td>
<td>/reports</td>
<td>The report output path.</td>
</tr>
<tr>
<td>spring.session.store.type</td>
<td>string</td>
<td>none</td>
<td>The session cache for high-availability environments. Possible values are none or redis. Default is none.</td>
</tr>
<tr>
<td>stopExecutionWhenProgNotFound</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to stop running if a program isn’t found. If set to true, interrupts the run if a program is not found.</td>
</tr>
<tr>
<td>forceHR</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to use Human Readable SYSPRINT, either on console or file output.</td>
</tr>
<tr>
<td>rollbackOnRTE</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to rollback implicit run unit transaction on runtime exceptions.</td>
</tr>
<tr>
<td>sctThreadLimit</td>
<td>long</td>
<td>5</td>
<td>The thread limit for triggering scripts.</td>
</tr>
<tr>
<td>dataSimplifier.onInvalidNumericData</td>
<td>string</td>
<td>reject</td>
<td>How to react when decoding invalid numeric data. Allowed values are reject, toleratespaces, toleratespaceslowvalues, toleratemost. Default is reject.</td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>filesDirectory</td>
<td>string</td>
<td></td>
<td>The directory for batches input/output files.</td>
</tr>
<tr>
<td>ims.messages.extendedSize</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to set the extendedSize on ims messages.</td>
</tr>
<tr>
<td>defaultKeepExistingFiles</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to set the dataset default previous value.</td>
</tr>
<tr>
<td>jics.db.ddlScriptLocation</td>
<td>string</td>
<td></td>
<td>The Jics ddl script location. Allows you to initiate the jics database schema using a .sql script. Blank by default. For example,.jics/sql/jics.sql.</td>
</tr>
<tr>
<td>jics.db.schemaTestQueryLocation</td>
<td>string</td>
<td></td>
<td>Location of the sql file that should contain a unique query that returns the number of objects in the jics schema (if any).</td>
</tr>
<tr>
<td>jics.db.dataScriptLocation</td>
<td>string</td>
<td></td>
<td>Location of the initJics.sql script, prepared by Analyzer from parsing CSD exports from the mainframe.</td>
</tr>
<tr>
<td>jics.db.dataTestQueryLocation</td>
<td>string</td>
<td></td>
<td>Location of a sql script containing a single sql query that is expected to return a count of objects (for example: counting number of records in the jics program table). If the count equals 0, database will be loaded using the jics.db.dataScriptLocation script, otherwise database load will be skipped.</td>
</tr>
<tr>
<td>jics.data.dataJsonInitLocation</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jics.xa.agent.timeout</td>
<td>number</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>query.useConcatCondition</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether key condition is built by key concatenation or not.</td>
</tr>
<tr>
<td>system.qdecfmt</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disposition.checkexistence</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to release a check on file existence for Dataset with DISP SHR or OLD.</td>
</tr>
<tr>
<td>useControlMVariable</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to use control-M specification for variable replacement.</td>
</tr>
<tr>
<td>card.encoding</td>
<td>string</td>
<td>CP1145</td>
<td>Card encoding: to be used with useControlMVariable.</td>
</tr>
<tr>
<td>mapTransfo.prefixes</td>
<td>string</td>
<td>&amp;,@,%%</td>
<td>List of prefixes to be used when transforming controlM variables. Each one separated by comma.</td>
</tr>
<tr>
<td>checkinputfilesize</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to release a check if the file size is a multiple of record size.</td>
</tr>
<tr>
<td>stepFailWhenAbend</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to raise an abend if a step fails or completes execution.</td>
</tr>
<tr>
<td>bluesam.fileLoading.commitInterval</td>
<td>number</td>
<td>100000</td>
<td>The bluesam commit interval.</td>
</tr>
<tr>
<td>uppercaseUserInput</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether user input must be in uppercase.</td>
</tr>
<tr>
<td>jhdb.lterm</td>
<td>string</td>
<td></td>
<td>Allow you to face a common logical terminal ID in the case of an IMS emulation. If not set then sessionId is used.</td>
</tr>
<tr>
<td>jhdb.identificationCardData</td>
<td>string</td>
<td></td>
<td>Used to hardcode some &quot;operator identification card data&quot; to the MID field designated by the CARD parameter. Blank by default, no input restriction.</td>
</tr>
</tbody>
</table>
### Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>string</td>
<td>ASCII</td>
<td>The encoding used in projects (not in groovy files). Expects a valid encoding CP1047, IBM930, ASCII, UTF-8,...</td>
</tr>
<tr>
<td>cl.configuration.context.encoding</td>
<td>string</td>
<td>CP297</td>
<td>The encoding of CL files. Expects a valid encoding CP1047, IBM930, ASCII, UTF-8. Default value is CP297</td>
</tr>
<tr>
<td>cl.zonedMode</td>
<td>string</td>
<td>EBCDIC_STRICT</td>
<td>The mode for encoding or decoding control language (CL) commands. Allowed values are EBCDIC_STRICT / EBCDIC_MODIFIED / AS400.</td>
</tr>
<tr>
<td>ims.programs</td>
<td>string</td>
<td></td>
<td>List of IMS programs to use. Separate each parameter with a semicolon (;) and each transaction with a comma (,). For example: PCP008, PCT008; PCP054, PCT054; PCP066, PCT066; PCP068, PCT068;</td>
</tr>
<tr>
<td>jhdb.configuration.context.encoding</td>
<td>string</td>
<td>CP297</td>
<td>The JHDB (Java Hierarchical Database) encoding. Expects a valid encoding string CP1047, IBM930, ASCII, UTF-8...</td>
</tr>
<tr>
<td>jhdb.metadata.extrapath</td>
<td>string</td>
<td>file:/setup/</td>
<td>A configuration parameter that specifies an extra, runtime-specific root folder for psbs and dbds folders.</td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jhdb.checkpointPersistence</td>
<td>string</td>
<td>none</td>
<td>The checkpoint persistence mode. Allowed values are none /add /end. Use add to persist checkpoints when a new one is created and added to the registry. Use end to persist checkpoint at server shutdown. Any other values disable the persistence. Note that each time a new checkpoint is added to the registry, all the existing checkpoints will be serialized and the file will be erased. It is not an append to the existing data in the file. So depending on the number of checkpoints, it can have some effects on performances.</td>
</tr>
<tr>
<td>jhdb.checkpointPath</td>
<td>string</td>
<td>file:/setup/</td>
<td>If jhdb.checkpointPersistence is not none then this parameter allows you to setup the checkpoint persistence path (checkpoint.dat file storage location), all the checkpoints data contained in the registry are serialized and backed up in a file (checkpoint.dat) located in provided folder. Note that only checkpoint data (scriptId, stepId, database position and checkpoint area) are concerned by this backup.</td>
</tr>
<tr>
<td>jhdb.navigation.cachexmit</td>
<td>number</td>
<td>5000</td>
<td>The cache duration (in milliseconds) used in hierarchical navigation for an RDBMS.</td>
</tr>
<tr>
<td>jhdb.use-db-prefix</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to enable a database prefix in hierarchical navigation for an RDBMS.</td>
</tr>
</tbody>
</table>
## Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jhdb.query.limitJoinUsage</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to use the limit join usage parameter on RDBMS graphs.</td>
</tr>
<tr>
<td>taskExecutor.corePoolSize</td>
<td>number</td>
<td>5</td>
<td>When a transaction in a terminal is initiated via a groovy script, a new thread is created. Use this parameter to setup the core pool size.</td>
</tr>
<tr>
<td>taskExecutor.maxPoolSize</td>
<td>number</td>
<td>10</td>
<td>When a transaction in a terminal is initiated via a groovy script, a new thread is created. Use this parameter to setup the max pool size (max number of parallel threads).</td>
</tr>
<tr>
<td>taskExecutor.queueCapacity</td>
<td>number</td>
<td>50</td>
<td>When a transaction in a terminal is initiated via a groovy script, a new thread is created. Use this parameter to setup the queue size. (maximum number of pending transactions when taskExecutor.maxPoolSize is reached)</td>
</tr>
<tr>
<td>taskExecutor.allowCoreThreadTimeOut</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to allow core threads to time out in JCIS. This enables dynamic growing and shrinking even in combination with a non-zero queue (since the max pool size will only grow once the queue is full).</td>
</tr>
<tr>
<td>jics.runUnitLauncherPool.enable</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to activate the run unit launcher pool in JICS.</td>
</tr>
<tr>
<td>jics.runUnitLauncherPool.size</td>
<td>number</td>
<td>20</td>
<td>The run unit launcher pool size in JICS.</td>
</tr>
<tr>
<td>jics.runUnitLauncherPool.validationInterval</td>
<td>number</td>
<td>1000</td>
<td>The validation interval of the run unit launcher pool in JICS, expressed in milliseconds.</td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>spring.aws.application.credentials</td>
<td>string</td>
<td>null</td>
<td>Load the AWS credentials from the credential profiles file in JICS.</td>
</tr>
<tr>
<td>jics.queues.sqs.region</td>
<td>string</td>
<td>eu-west-1</td>
<td>The AWS Region for AWS Simple Queue Service, used in JICS.</td>
</tr>
<tr>
<td>mq.queues.sqs.region</td>
<td>string</td>
<td>eu-west-3</td>
<td>The AWS Region for the AWS SQS MQ service.</td>
</tr>
<tr>
<td>quartz.scheduler.standby-if-error</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to trigger job execution if the job scheduler is in standby mode. If true, when enabled job execution is not triggered.</td>
</tr>
<tr>
<td>databaseStatistics</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to allow SQL builders to collect and display statistics information.</td>
</tr>
<tr>
<td>dbDateFormat</td>
<td>string</td>
<td>yyyy-MM-dd</td>
<td>The db target date format.</td>
</tr>
<tr>
<td>dbTimeFormat</td>
<td>string</td>
<td>HH:mm:ss</td>
<td>The db target time format.</td>
</tr>
<tr>
<td>dbTimestampFormat</td>
<td>string</td>
<td>yyyy-MM-dd</td>
<td>The db target timestamp format.</td>
</tr>
<tr>
<td>dateTimeFormat</td>
<td>string</td>
<td>ISO</td>
<td>The dateTimeFormat describes how to spill database date time timestamp type into data simplifier entities. Allowed values are ISO / EUR / EUR / USA / LOCAL.</td>
</tr>
<tr>
<td>localDateFormat</td>
<td>string</td>
<td>yyyy-MM-dd</td>
<td>List of local date formats. Separate each format with .</td>
</tr>
<tr>
<td>localTimeFormat</td>
<td>string</td>
<td>yyyy-MM-dd</td>
<td>List of local time formats. Separate each format with .</td>
</tr>
<tr>
<td>localTimestampFormat</td>
<td>string</td>
<td>yyyy-MM-dd</td>
<td>List of local timestamp formats. Separate each format with .</td>
</tr>
<tr>
<td>pgmDateFormat</td>
<td>string</td>
<td>yyyy-MM-dd</td>
<td>The date time format.</td>
</tr>
</tbody>
</table>
### Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgmTimeFormat</td>
<td>string</td>
<td>HH.mm.ss</td>
<td>The time format used for pgm (programs) execution.</td>
</tr>
<tr>
<td>pgmTimestampFormat</td>
<td>string</td>
<td>yyyy-MM-dd-HH.mm.ss.SSSSSS</td>
<td>The timestamp format.</td>
</tr>
<tr>
<td>cacheMetadata</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to cache database metadata.</td>
</tr>
<tr>
<td>forceDisableSQLTrimStringType</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to disable trim of all sql string parameters.</td>
</tr>
<tr>
<td>fetchSize</td>
<td>number</td>
<td></td>
<td>The fetchSize value for cursors. Use when fetching data using chunks by load/unload utils.</td>
</tr>
<tr>
<td>check-groovy-file</td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to check groovy files content before registering.</td>
</tr>
<tr>
<td>qtemp.uuid.length</td>
<td>number</td>
<td>9</td>
<td>The QTEMP unique id length.</td>
</tr>
<tr>
<td>qtemp.dblog</td>
<td>boolean</td>
<td>false</td>
<td>Whether to enable QTEMP Database logging.</td>
</tr>
<tr>
<td>qtemp.cleanup.threshold.hours</td>
<td>number</td>
<td>0</td>
<td>To specify when qtemp.dblog is enabled. The db partition lifetime (in hours).</td>
</tr>
<tr>
<td>sort.function</td>
<td>string</td>
<td></td>
<td>The sort function name for the blu4iv database.</td>
</tr>
</tbody>
</table>

### Available properties for optional web applications

Depending on your modernized application, you might need to configure one or more optional web applications that represent support for dependencies such as z/OS, AS/400, or IMS/MFS. The following tables contain lists of the available key/value parameters for configuring each optional web application.

**gapwalk-utility-pgm.war**

This optional web application contains support for Z/OS utility programs.

This table provides an exhaustive view of key/values parameters for this application.
### Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging.config</td>
<td>Path</td>
<td>classpath:logback-utility.xml</td>
<td>Standard key for the reference to the logback configuration file. Other standard logging keys are available too.</td>
</tr>
<tr>
<td>spring.jta.enabled</td>
<td>boolean</td>
<td>false</td>
<td>Standard key. If the datasource support mode is not static-xa, spring JTA transactions autoconfiguration must be disabled.</td>
</tr>
<tr>
<td>spring.datasource.primary.jndi-name</td>
<td>string</td>
<td>jdbc/primary</td>
<td>The jndi name (Java Naming And Directory Interface) for the primary datasource, if using JNDI.</td>
</tr>
<tr>
<td>primary.datasource +-driver-class-name +-url +-username +-password</td>
<td>Standard spring datasource with subkeys</td>
<td>Contains the connection information for the application database, if not using JNDI. Must have the same configuration as in the modernized application yml file. Alternately, use of AWS secrets is strongly encouraged, as explained in xref:../configuration/configuration.adoc[Configuration].</td>
<td></td>
</tr>
<tr>
<td>encoding</td>
<td>string</td>
<td>ASCII</td>
<td>The encoding used in utility programs. Expects a valid encodingCP1047,IBM930,ASCII,UTF-8.</td>
</tr>
<tr>
<td>sysPunchEncoding</td>
<td>string</td>
<td>ASCII</td>
<td>The syspunch encoding character set. Expects a valid encodingCP1047,IBM930,ASCII,UTF-8.</td>
</tr>
<tr>
<td>zonedMode</td>
<td>string</td>
<td>EBCDIC STRICT</td>
<td>The mode for encoding or decoding zoned data types. Allowed values areEBCDIC STRICT /EBCDIC_MODIFIED /AS400.</td>
</tr>
<tr>
<td>unload.chunkSize</td>
<td>number</td>
<td>0</td>
<td>Chunk size used for unload utility.</td>
</tr>
</tbody>
</table>
### Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unload.sqlCodePointShift</td>
<td>number</td>
<td>0</td>
<td>The SQL code point shift for unload utility. Runs the shifting characters process. Required when your target database from DB2 is Postgresql.</td>
</tr>
<tr>
<td>unload.columnFiller</td>
<td>string</td>
<td>space</td>
<td>The unload utility column filler.</td>
</tr>
<tr>
<td>unload.varCharIsNull</td>
<td>boolean</td>
<td>false</td>
<td>Use this parameter in INFTILB program, if set to true then all not nullable fields with blank (space) values returns an empty string.</td>
</tr>
<tr>
<td>unload.useDatabaseConfiguration</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to use the date or time configuration from application-main.yml in unload utility.</td>
</tr>
<tr>
<td>unload.format.date</td>
<td>string</td>
<td>MM/dd/yyyy</td>
<td>If unload.useDatabaseConfiguration is enabled, the date format to use in the unload utility.</td>
</tr>
<tr>
<td>unload.format.time</td>
<td>string</td>
<td>HH.mm.ss</td>
<td>If unload.useDatabaseConfiguration is enabled, the time format to use in the unload utility.</td>
</tr>
<tr>
<td>unload.format.timestamp</td>
<td>string</td>
<td>yyyy-MM-dd-HH.mm.ss.SSSSSS</td>
<td>If unload.useDatabaseConfiguration is enabled, the timestamp format to use in the unload utility.</td>
</tr>
<tr>
<td>unload.nbi.whenNull</td>
<td>hexadecimal</td>
<td>6F</td>
<td>The Null Byte Indicator (nbi) value to add when value from database is null.</td>
</tr>
<tr>
<td>unload.nbi.whenNotNull</td>
<td>hexadecimal</td>
<td>00</td>
<td>The Null Byte Indicator (nbi) value to add when value from database is not null.</td>
</tr>
<tr>
<td>unload.nbi.writeNullIndicator</td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to write out the null indicator in the unload output file.</td>
</tr>
<tr>
<td>unload.fetchSize</td>
<td>number</td>
<td>0</td>
<td>Allows you to tune the fetch size when handling cursors in the unload utility.</td>
</tr>
</tbody>
</table>
## Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>treatLargeNumberAsInteger</code></td>
<td>boolean</td>
<td>false</td>
<td>Specifies whether to treat large numbers as Integer. They are treated as BigDecimal by default.</td>
</tr>
<tr>
<td><code>load.batchSize</code></td>
<td>number</td>
<td>0</td>
<td>The load utility batch size.</td>
</tr>
<tr>
<td><code>load.format.localDate</code></td>
<td>string</td>
<td>dd.MM.yyyy\dd/MM/yyyy-MM-dd</td>
<td>The load utility local date format to use.</td>
</tr>
<tr>
<td><code>load.format.localTime</code></td>
<td>string</td>
<td>HH:mm:ss\HH.mm.ss</td>
<td>The load utility local time format to use.</td>
</tr>
<tr>
<td><code>load.format.dbDate</code></td>
<td>string</td>
<td>yyyy-MM-dd</td>
<td>The load utility database format to use.</td>
</tr>
<tr>
<td><code>load.format.dbTime</code></td>
<td>string</td>
<td>HH:mm:ss</td>
<td>The load utility database time to use.</td>
</tr>
<tr>
<td><code>load.sqlCodePointShift</code></td>
<td>number</td>
<td>0s</td>
<td>The SQL code pointshift for load utility. Runs the shifting characters process. Required when your target database from DB2 is Postgresql.</td>
</tr>
<tr>
<td><code>forcedDate</code></td>
<td>string</td>
<td></td>
<td>Forces the date to the date provided if there is one.</td>
</tr>
<tr>
<td><code>frozenDate</code></td>
<td>boolean</td>
<td>true</td>
<td>Specifies whether to freeze the date. Applies only if forcedDate is also set.</td>
</tr>
<tr>
<td><code>jcl.type</code></td>
<td>string</td>
<td>mvss</td>
<td>.jcl file type. Allowed values are jcl / vse. The IDCAMS utility PRINT/REPRO commands return 4 if the file is empty for non-vse jcl.</td>
</tr>
<tr>
<td><code>hasGraphic</code></td>
<td>boolean</td>
<td>false</td>
<td>Whether the INFUTILB utility needs to handle GRAPHIC DB2 columns.</td>
</tr>
</tbody>
</table>

**gapwalk-cl-command.war**

This optional web application contains support for AS/400 utility programs.

This table provides an exhaustive view of key/values parameters for this application.
### Enabling properties

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging.config</td>
<td>Path</td>
<td>classpath:logback-utility.xml</td>
<td>Standard key for the reference to the logback configuration file. Other standard logging keys are available too.</td>
</tr>
<tr>
<td>spring.jta.enabled</td>
<td>boolean</td>
<td>false</td>
<td>Standard key. If the datasource support mode is not static-xa, spring JTA transactions autoconfiguration must be disabled.</td>
</tr>
<tr>
<td>spring.datasource.primary.jndi-name</td>
<td>string</td>
<td>jdbc/primary</td>
<td>The jndi name (Java Naming And Directory Interface) for the primary datasource, if using JNDI.</td>
</tr>
<tr>
<td>primary.datasource +-driver-class-name +url +username +password</td>
<td>Standard spring datasource with subkeys</td>
<td>Contains the connection information for the application database, if not using JNDI. Must have the same configuration as in the modernized application yml file. Alternately, use of AWS secrets is strongly encouraged, as explained in xref:../configuration/configuration.adoc[Configuration].</td>
<td></td>
</tr>
<tr>
<td>encoding</td>
<td>string</td>
<td>ASCII</td>
<td>The encoding used in utility programs. Expects a valid encodingCP1047,IBM930,ASCII,UTF-8...</td>
</tr>
<tr>
<td>zonedMode</td>
<td>string</td>
<td>EBCDIC STRICT</td>
<td>The mode for encoding or decoding zoned data types. Allowed values areEBCDIC STRICT /EBCDIC_MODIFIED /AS400.</td>
</tr>
<tr>
<td>commands-off</td>
<td>string</td>
<td></td>
<td>List of commands to turn off, separated by comma. Allowed values arePGM BASIC,RCVMSG,SNDRCVF,CHGVAR,QUICKIN,RTVJOBA,ADDLFM,ADDPFM,RCVF,OVRDBF,DLTOVR,CPYF,SNDDTAQ. Usefull when you want to disable or overwrite an existing program.PGM_BASIC</td>
</tr>
</tbody>
</table>
### gapwalk-hierarchical-support.war

This optional web application contains IMS/MFS transaction support.

This table provides an exhaustive view of key/values parameters for this application.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging.config</td>
<td>Path</td>
<td>classpath:logback-utility.xml</td>
<td>Standard key for the reference to the logback configuration file. Other standard logging keys are available too.</td>
</tr>
<tr>
<td>spring.jta.enabled</td>
<td>boolean</td>
<td>false</td>
<td>Standard key. If the datasource support mode is not static-xa, spring JTA transactions autoconfiguration must be disabled.</td>
</tr>
<tr>
<td>jhdb.configuration.context.encoding</td>
<td>string</td>
<td>none</td>
<td>The JHDB (Java Hierarchical Database) encoding. Expects a valid encoding string CP1047, IBM930, ASCII, UTF-8...</td>
</tr>
<tr>
<td>jhdb.checkpointPersistence</td>
<td>boolean</td>
<td>none</td>
<td>The checkpoint persistence mode. Allowed values are none /add /end. Use add to persist checkpoints when a new one is created and added to the registry. Use end too persist checkpoint at server shutdown. Any other values disable the persistence. Note that each time a new checkpoint is added to the registry, all the existing checkpoints will be serialized and the file will be erased. It is not an append to the existing data in the file. So depending on the number of checkpoints,</td>
</tr>
</tbody>
</table>
Configure authentication for Gapwalk applications

This section describes how to configure OAuth2 authentication for Gapwalk applications using an Identity Provider (IdP) such as Cognito, Azure AD, Keycloak, etc...

Topics
- Prerequisite (p. 119)
- Amazon Cognito Setup (p. 119)
- Integration of Amazon Cognito in Gapwalk application (p. 120)

Prerequisite

In this tutorial we will use Amazon Cognito as the IdP and planetDemo as the modernized project. You can use any other external identity provider. The ClientRegistration information must be obtained from your IdP and are required for gapwalk authentication. For more information, see your IdP’s official documentation.

The ClientRegistration information:

client-id
the id of the ClientRegistration, in our example it will be planetDemo.

client-secret
your client secret.

authorization endpoint
The authorization endpoint URI for the authorization server.

token endpoint
The Token Endpoint URI for the Authorization Server.

jwks endpoint
The URI used to get the JSON Web Key (JWK) containing the keys for validating JSON Web Signature issued by the authorization server.

redirect URI
URI to which the authorization server redirects the end-user if access is granted.

Amazon Cognito Setup

First we will create and configure a Amazon Cognito user pool and user that we will use with our deployed gapwalk application for testing purpose. If you are using an other IdP you can skip this step.

Create user pool

1. Go to Amazon Cognito in the AWS Management Console and authenticate using your AWS credentials.
2. Choose **User Pools**.
3. Choose **Create a user pool**.
4. In **Configure sign-in experience**, keep the **Cognito user pool** default provider type. You can choose one or multiple **Cognito user pool sign-in options**; for now, choose **User name**, then choose **Next**.
5. In **Configure security requirements**, keep the defaults and disable **Multi-factor authentication** by choosing No MFA then choose **Next**.
6. Disable **Enable self-registration** as a security measure and choose **Next**.
7. Choose **Send email with Cognito**. Choose **Next**.
8. In **Integrate your app**, choose a name for your user pool. In **Hosted authentication pages**, choose **Use the Cognito Hosted UI**.
9. For simplicity’s sake, in **Domain**, choose **Use a Cognito domain** and enter a domain prefix; for example, https://planetsdemo. The demo app must be added as a client.
   1. In **Initial app client**, choose **Confidential client**. Enter an app client name, such as ‘planetsdemo’ and choose **Generate a client secret**.
   2. In **Allowed callback URL** enter the url to redirect the user to after authentication. A temporary http://localhost:8080/planetsdemo URL can also be used, then edited later.
   3. Keep default values in the **Advanced app client settings** and **Attribute read and write permissions** sections.
10. Choose **Next**.

For more information, see **Create user pool**.

### User Creation

Since self-registration is disabled, create an Amazon Cognito user. Navigate to Amazon Cognito in the AWS Management Console. Choose the user pool you created, then in **Users** choose **Create user**.

In **User information**, choose **Send an email invitation**, enter a user name and an email address, and choose **Generate a password**. Choose **Create user**.

### Integration of Amazon Cognito in Gapwalk application

Now that your Amazon Cognito user pool and user are ready, go the main-application.yml file of your modernized application and add the following code:

```yaml
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
```

120
AWS Mainframe Modernization User Guide
AWS Blu Age Runtime (on Amazon EC2) APIs

| token-uri: ${gapwalk-application.security.domainName}/oauth2/token |
| user-name-attribute: cognito:username |
| resource-server: |
| 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

Replace the following placeholders as described:

1. Go to Amazon Cognito in the AWS Management Console and authenticate using your AWS credentials.
2. Choose User Pools and choose the user pool you created. You can find your pool-id in User pool ID.
3. Choose App integration where you can find your your-cognito-domain and go to App clients and analytics and choose your app.
4. In App client: yourApp you can find the client-name, client-id and client-secret (Show client secret).
5. region-id corresponds to the region id where you created your Amazon Cognito user and user pool, eu-west-3; for example.
6. For redirect-uri enter the URI to redirect the user to after authentication.

You can deploy your Gapwalk application now and use the user created previously to sign-in to your app.

AWS Blu Age Runtime (on Amazon EC2) APIs

The AWS Blu Age runtime uses several web-applications to expose REST endpoints, providing ways to interact with the modernized applications using REST clients (e.g. calling jobs using a scheduler).

The purpose of this document is to list available REST endpoints, giving details about:

- their role
- the way to use them properly

The endpoints listing is organized into categories, depending on the nature of the provided service and the web-application exposing the endpoints.

We assume that you already have a basic knowledge of using REST endpoints using dedicated tools such as POSTMAN, Thunder Client, CURL, web browsers, etc ...) or writing your own piece of code to make an API call.

Topics
- Building URLs (p. 122)
- Gapwalk-Application (p. 122)
- Blusam Application Console REST endpoints (p. 135)
- JICS Application Console (p. 147)
- Data Structures (p. 159)
Building URLs

Each web-application below is defining a root path, shared by all endpoints. Each endpoint then adds its own dedicated path. The resulting URL to use is the result of the concatenation of the paths. For instance, considering the first endpoint for the Gapwalk-Application, we have:

- /gapwalk-application for the root web-application path.
- /scripts for the dedicated endpoint path.

The resulting URL to use will be http://server:port/gapwalk-application/scripts

server

points at the server name (the one hosting the given web-application).

port

the port exposed by the server.

Gapwalk-Application

Endpoints for the Gapwalk web application use the root path /gapwalk-application.

Topics

- Batch jobs (modernized JCLs and alike) related endpoints (p. 122)
- Metrics Endpoints (p. 128)
- Other Endpoints (p. 129)
- Job Queues related endpoints (p. 131)

Batch jobs (modernized JCLs and alike) related endpoints

Batch jobs can be run either synchronously or asynchronously (see details below). Batch jobs are being executed using groovy scripts that are the results of the modernization of legacy scripts (JCL).

Topics

- List deployed scripts (p. 123)
- Launch a script synchronously (p. 123)
- Launch a script asynchronously (p. 124)
- Listing triggered scripts (p. 125)
- Retrieving job execution details (p. 125)
- Listing asynchronously launched scripts that can be killed (p. 126)
- Listing synchronously launched scripts that can be killed (p. 126)
- Killing a given job execution (p. 126)
- Listing existing checkpoints for restartability (p. 126)
- Restarting a job (synchronously) (p. 127)
- Restarting a job (asynchronously) (p. 127)
- Setting thread limit for asynchronous job executions (p. 127)
List deployed scripts

- Supported method: GET
- Path: /scripts
- Arguments: none
- This endpoint returns the list of deployed groovy scripts on the server, as a String. This endpoint is primarily intended to be used from a web browser, since the resulting String is a HTML page, with active links (a link per launchable script -- see sample below).

Sample response:

```
<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**
The links represent the url to use to launch each listed script **synchronously**.

- Supported method: GET
- Path: /triggerscripts
- Arguments: none
- This endpoint returns the list of deployed groovy scripts on the server, as a String. This endpoint is primarily intended to be used from a web browser, since the resulting String is a HTML page, with active links (a link per launchable script -- see sample below).

As opposed to the previous endpoint response, the links represent the url to use to launch each listed script **asynchronously**.

```
localhost:8080/gapwalk-applicati: X +

← → ① localhost:8080/gapwalk-application/triggerscripts
```

COMBTRAN
CREASTMT
INTCALC
POSTTRAN
REPROC
TRANBKP
TRANREPT

Launch a script synchronously

This endpoint has two variants with dedicated paths for GET and POST usage (see below).
• Supported method: GET
• Path: /script/{scriptId:.+}
• Supported method: POST
• Path: /post/script/{scriptId:.+}
• Arguments:
  • identifier of the script to launch
  • optionally: parameters to pass to the script, using request parameters (seen as a Map<String,String>). The given parameters will be automatically added to the bindings of the invoked groovy script.

The call will launch the script with the given identifier, using extra parameters if provided and wait for script execution completion before returning a message (String) that'll be either:
• "Done." (if job execution ran smoothly).
• A JSON error message with details about what went wrong during job execution. Further details can be retrieved from the server logs, to understand what went wrong with the job execution.

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

Looking at the server logs, we can figure out that this a deployment issue (the expected program has not been properly deployed, so it cannot be found, making job execution fail):

```
2023-09-10 11:39:12 [INFO] ScriptController - Started
2023-09-10 11:39:12 [INFO] ScriptController - Could not find program "CBACT04C" in the program registry.
2023-09-10 11:39:12 [INFO] ScriptController - Could not find program "CBACT04C" in the program registry.
```

**Note**
The synchronous calls should be reserved for short time running jobs. Long times running jobs should rather be launched asynchronously (see dedicated endpoint below).

**Launch a script asynchronously**

• Supported methods: GET / POST
• Path: /triggerscript/{scriptId:.+}
• arguments:
  • identifier of the script to launch
  • optionally: parameters to pass to the script, using request parameters (seen as a Map<String,String>). The given parameters will be automatically added to the https://docs.groovy-lang.org/latest/html/api/groovy/lang/Binding.html[bindings] of the invoked groovy script.

As opposed to the synchronous mode above, the endpoint is not waiting for the job execution to finish to send a response. The job execution is launched at once, if an available thread can be found to do so, and a response is sent immediately to caller, with the job execution id, a unique identifier representing the job execution, that can be used to query job execution status or force kill a job execution that is supposed to be malfunctioning. The format of the response is:

```
Triggered script <script identifier> [unique job execution id] @ <date and time>
```
Since the job asynchronous execution relies on a fixed limited number of threads, the job execution might not be launched if no available thread could be found. In that case, the returned message will rather look like:

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

See the `settriggerthreadlimit` endpoint below to learn how to increase the thread limit.

Sample response:

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

The unique job execution identifier permits to quickly retrieve related log entries in the server logs if required. It is also used by several other endpoints detailed below.

**Listing triggered scripts**

- Supported methods: GET
- Paths: `/triggeredscripts/{status:.+}`, `/triggeredscripts/{status:.+}/{namefilter}`
- Arguments:
  - Status (mandatory): the status of the triggered scripts to retrieve. Possibles values are:
    - `all`: show all job execution details, whether the jobs are still running or not.
    - `running`: only show jobs details for jobs that are currently running.
    - `done`: only show jobs details for jobs whose execution is over.
    - `killed`: only show jobs details for jobs whose execution has been forcefully killed using the dedicated endpoint (see below).
    - `triggered`: only show jobs details for jobs which have been triggered but not yet launched.
    - `failed`: only show jobs details for jobs whose execution has been marked as failed.
  - `_namefilter (optional)_`: retrieve only executions for the given script identifier.
- Returns a collection of job executions details as JSON. For more information, see [Job Execution Details message structure](#) (p. 159).

Sample response:

```
[
  {
    "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"
  }
]
```

**Retrieving job execution details**

- Supported method: GET
• Path: /getjobexecutioninfo/{jobexecutionid:.+}
• Arguments:
  • jobexecutionid (mandatory): the unique job execution identifier to retrieve the corresponding job
    execution details.
• Returns: a JSON string representing a single job execution details (see Job Execution Details message
  structure (p. 159)) or an empty response if no job execution details could be found for the given
  identifier.

Listing asynchronously launched scripts that can be killed

• Supported method: GET
• Path: /killablescripts
• Returns a collection of job execution identifiers of jobs which have been launched asynchronously that
  are still currently running and can be forcefully killed (see the /kill endpoint below).

Listing synchronously launched scripts that can be killed

• Supported method: GET
• Path: /killablesyncscripts
• Returns a collection of job execution identifiers of jobs which have been launched synchronously, are
  still currently running and can be forcefully killed (see the /kill endpoint below).

Killing a given job execution

• Supported method: GET
• Path: /kill/{identifier:.+}
• argument: job execution identifier (mandatory): the unique job execution identifier to point at the job
  execution to be forcefully killed.
• Returns: a textual message detailing the job execution kill attempt outcome; the message will contain
  the script identifier, the job execution unique identifier and the date and time at which the execution
  kill occurred. If no running job execution could be found for the given identifier, an error message will
  be returned instead.

Warning

• The runtime makes its best effort to kill the target job execution nicely. Thus, the response
  from the /kill endpoint might take a bit of time to reach the caller, as the AWS Blu Age
  runtime will try to minimize the business impact of killing the job.
• Forcefully killing a job execution should not be done lightly, as it may have direct business
  consequences, including possible data loss or corruption. It should be reserved for cases where
  a given job execution has gone sideways and data remediation means are clearly identified.
• Killing a job should lead to further investigations (post-mortem analysis) to figure out what
  went wrong and take proper remediations actions.
• In any case, attempt to kill a running job will be logged in the server logs with warning level
  messages.

Listing existing checkpoints for restartability

Job restartability relies on the ability for the scripts to register checkpoints in the
CheckpointRegistry to track down the job execution progress. If a job execution fails to end properly,
and restart checkpoints have been registered, one can simply restart the job execution from the last known registered checkpoint (without having to execute the steps above the checkpoint).

- Supported method: GET
- Path: /restarts
- Returns the list of existing restart points, that can be used to restart a job whose execution did not come to and end properly, as an html page. If no checkpoints were registered by any scripts, the page contents will be "No registered checkpoints."

**Restarting a job (synchronously)**

- Supported method: GET
- Path: /restart/{hashcode}
- Arguments: hashcode (integer - mandatory): restart a previously aborted job execution, using the provided hashcode as checkpoint value (see the /restarts endpoint above to learn how to retrieve a valid checkpoint value).
- Returns: see script return description above.

**Restarting a job (asynchronously)**

- Supported method: GET
- Path: /triggerrestart/{hashcode}
- Arguments: hashcode (integer - mandatory): restart a previously aborted job execution, using the provided hashcode as checkpoint value (see the /restarts endpoint above to learn how to retrieve a valid checkpoint value).
- Returns: see trigger script return description above.

**Setting thread limit for asynchronous job executions**

The job asynchronous execution relies on a dedicated pool of threads in the JVM. That pool has a fixed limit regarding the number of available threads. The used has the ability to adjust the limit according to the host capabilities (number of CPUs, available memory, etc...). By default, the thread limit is set to 5 threads.

- Supported method: GET
- Path: /settriggerthreadlimit/{threadlimit:.+}
- Argument (integer): the new thread limit to apply. Must be a strictly positive integer.
- Returns a message (String) giving the new thread limit and the previous one, or an error message if the provided thread limit value is not valid (not a strictly positive integer).

Sample response:

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

**Counting currently running triggered job executions**

- Supported method: GET
- Path: /countrunningtriggeredscripts
- Returns a message indicating the number of running jobs launched asynchronously and the thread limit (that is the maximum number of triggered jobs that can run simultaneously).
Sample response:

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

**Note**
This can be used to check, prior to launching a job, if the thread limit has not been reached (which would prevent the job from being launched).

**Purge job executions information**

The job executions information remain in the server memory as long as the server is up. It might be convenient to purge oldest informations from the memory, as they are not relevant anymore; this is the purpose of this endpoint.

- Supported method: GET
- Path: /purgejobinformation/{age:.+}
- Arguments: a strictly positive integer value representing the age in hours of informations to be purged.
- Returns a message with the following informations:
  - Name of the purge file where purged job execution informations are being stored for archiving purpose.
  - Number of purged job execution informations.
  - Number of remaining job execution informations in memo

**Metrics Endpoints**

**JVM**

This endpoint returns available metrics related to the JVM.

- Supported method: GET
- Path: /metrics/jvm
- Arguments: none
- Returns a message with the following information:
  - threadActiveCount: Number of active threads.
  - jvmMemoryUsed: Memory actively used by the Java Virtual Machine.
  - jvmMemoryMax: Maximum memory allowed for the Java Virtual Machine.
  - jvmMemoryFree: Available memory not currently in use by the Java Virtual Machine.

**Session**

This endpoint returns metrics related to currently opened HTTP sessions.

- Supported method: GET
- Path: /metrics/session
- Arguments: none
- Returns a message with the following information:
  - sessionCount: Number of active user sessions currently maintained by the server.

**Batch**

- Supported method: GET
• Path: /metrics/batch
  • Arguments:
    • startTimestamp (optional, number): Starting timestamp for data filtering.
    • endTimestamp (optional, number): Ending timestamp for data filtering.
    • page (optional, number): Page number for pagination.
    • pageSize (optional, number): Number of items per page in pagination.
  • Returns a message with the following information:
    • content: List of batch execution metrics.
    • pageNumber: Current page number in pagination.
    • pageSize: Number of items displayed per page.
    • totalPages: Total number of pages available.
    • numberOfElements: Count of items on the current page.
    • last: Boolean flag for the last page.
    • first: Boolean flag for the first page.

Transaction

• Supported method: GET
• Path: /metrics/transaction
• Arguments:
  • startTimestamp (optional, number): Starting timestamp for data filtering.
  • endTimestamp (optional, number): Ending timestamp for data filtering.
  • page (optional, number): Page number for pagination.
  • pageSize (optional, number): Number of items per page in pagination.
• Returns a message with the following information:
  • content: List of transaction execution metrics.
  • pageNumber: Current page number in pagination.
  • pageSize: Number of items displayed per page.
  • totalPages: Total number of pages available.
  • numberOfElements: Count of items on the current page.
  • last: Boolean flag for the last page.
  • first: Boolean flag for the first page.

Other Endpoints

Use these endpoints to list registered programs or services, discover health status, and manage JICS transactions.

Topics

• Listing registered programs (p. 130)
• Listing registered services (p. 130)
• Health status (p. 130)
• Listing available JICS transactions (p. 130)
• Launch a JICS transaction (p. 130)
• Launch a JICS transaction (alternative) (p. 1291)
Listing registered programs

- Supported method: GET
- Path: /programs
- Returns the list of registered programs, as a html page. Each program is designated by its main program identifier. Both modernized legacy programs and utility programs (IDCAMS, IEBGENER, etc...) are being returned in the list. Please note that the available utility programs will depend on the utility web-applications that have been deployed on your tomcat server. For instance, z/OS utility support programs might not be available for modernized iSeries assets, as they are not relevant.

Listing registered services

- Supported method: GET
- Path: /services
- Returns the list of registered runtime services, as a html page. The given services are brought by the AWS Blu Age runtime as utilities, that can be used for instance in groovy scripts. Blusam load services (to create Blusam datasets from legacy datasets) fall into that category.

Sample response:

<pre>
BluesamESDSFileLoader<br>BluesamKSDSFileLoader<br>BluesamRRDSFileLoader
</pre>

Health status

- Supported method: GET
- Path: /
- Returns a simple message, indicating that the gapwalk-application is up and running (Jics application is running.)

Listing available JICS transactions

- Supported method: GET
- Path: /transactions
- Returns a html page listing all available JICS transactions. This only makes sense for environments with JICS elements (modernization of legacy CICS elements).

Sample response:

<pre>
INQ1<br>MENU<br>MNT2<br>ORD1<br>PRNT
</pre>

Launch a JICS transaction

- Supported methods: GET,POST
- Path: /jicstransrunner/{jtrans:.+}
- arguments:
  - JICS transaction identifier (string, required) : identifier of the JICS transaction to be launched (8 characters long at max.)
  - required: additional input data to pass to the transaction, as a Map<String, Object>. The contents of this map will be used to feed the COMMAREA that will be consumed by the JICS transaction. The map can be empty if no data is required to run the transaction.
optional: Http headers entries, to customize the run environment for the given transaction. The following header keys are being supported:

- **jics-channel**: The name of the JICS CHANNEL to be used by the program that will be launched by this transaction launch.
- **jics-container**: The name of the JICS CONTAINER to be used for this JICS transaction launch.
- **jics-startcode**: the STARTCODE (String, up to 2 characters) to use at JICS transaction start. See [STARTCODE](#) for possible values (browse down the page).
- **jicxa-xid**: The XID (X/Open transaction identifier XID structure) of a "global transaction" (XA), initiated by the caller, to which the current JICS transaction launch will participate.


For more information about the details of the structure, see [Transaction launch outcome structure](#) (p. 161).

**Launch a JICS transaction (alternative)**

- supported methods: GET, POST
- path: `/jicstransaction/{jtrans:.+}`
- arguments:

  **JICS transaction identifier (string, required)**

  identifier of the JICS transaction to be launched (8 characters long at max.)

  **required: additional input data to pass to the transaction, as a Map<String, Object>**

  The contents of this map will be used to feed the [COMMAREA](#) that will be consumed by the JICS transaction. The map can be empty if no data is required to run the transaction.

  **optional: Http headers entries, to customize the run environment for the given transaction.**

  The following header keys are being supported:

  - **jics-channel**: The name of the JICS CHANNEL to be used by the program that will be launched by this transaction launch.
  - **jics-container**: The name of the JICS CONTAINER to be used for this JICS transaction launch.
  - **jics-startcode**: the STARTCODE (String, up to 2 characters) to use at JICS transaction start. For possible values, see [STARTCODE](#) (browse down the page).
  - **jicxa-xid**: The XID (X/Open transaction identifier XID structure) of a "global transaction" (XA), initiated by the caller, to which the current JICS transaction launch will participate.

  returns: a [com.netfective.bluage.gapwalk.rt.shared.web.RecordHolderBean](#) JSON serialization, representing the outcome of the JICS transaction launch. The details of the structure can be found in [Transaction launch record outcome structure](#) (p. 161).

**Job Queues related endpoints**

Job Queues are the AWS Blu Age support for the AS400 jobs submission mechanism. Job Queues are used in AS400 to run job on specific thread pools. A job queue is defined by a name and a maximum number of threads that corresponds to the maximum number of programs that can be run simultaneously on that queue. If more jobs are submitted on the queue than the maximum number of threads, jobs will wait for a thread to be available.

For an exhaustive list of status for a job on a queue, see [Possible status of a job on a queue](#) (p. 162).
Operations on job queues are handled through the following dedicated endpoints.

Topics
- List available queues (p. 132)
- Start or restart a job queue (p. 132)
- Submit a job for launch (p. 133)
- List all scheduled jobs (p. 133)
- List all jobs that are "on hold" (p. 133)
- List all active jobs (p. 133)
- List all jobs that are waiting to be launched (p. 133)
- Release all jobs that are "on hold" (p. 135)
- Release all jobs that are "on hold" for a given job name (p. 135)
- Release a given job for a job number (p. 135)

List available queues
- Supported method: GET
- Path: list-queues
- Returns the list of available queues along with their status, as a JSON list of key-values.

Sample response:
```json
{"Default":"STAND_BY","queue1":"STARTED","queue2":"STARTED"}
```

Possible status for a job queue are:

**STAND_BY**
the job queue is waiting to be started.

**STARTED**
the job queue is up and running.

**UNKNOWN**
the job queue status cannot be determined.

Start or restart a job queue
- Supported method: POST
- Path: /restart/{name}
- Argument: the name of the queue to be started/restarted, as a String - mandatory.
- The endpoint does not return anything but rather relies on http status to indicate the outcome of the start/restart operation:
  - HTTP 200
    the start/restart operation went well: the given job queue is now STARTED.
  - HTTP 404
    the job queue does not exist.
HTTP 503

an exception occurred during the start/restart attempt (server logs should be inspected to figure out what went wrong).

Submit a job for launch

- Supported method: POST
- Argument: mandatory as request body, a JSON serialization of a com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage object. For more information, see Submit job input (p. 162).
- Returns: a JSON containing the original SubmitJobMessage and a log indicating if the job has been submitted or not.

List all scheduled jobs

- Supported method: GET
- Path: list-jobs
- Returns: a list of all scheduled jobs, as a JSON string. For a sample response, see List of scheduled jobs response (p. 163).

List all jobs that are "on hold"

- Supported method: GET
- Path: list-jobs-hold
- Returns: a list of all scheduled jobs, as a JSON string. For a sample response, see List of "on hold" jobs response (p. 164).

List all active jobs

- Supported method: GET
- Path: list-jobs-active
- Returns: a list of all jobs with status ACTIVE, as a JSON string. The response is similar in structure with the one from List of scheduled jobs response (p. 163).

List all jobs that are waiting to be launched

- Supported method: GET
- Path: list-jobs-waiting
- Returns: a list of all jobs with status EXECUTION_WAIT (jobs that are waiting for a thread to be available for launch), as a JSON string. The response is similar in structure with the one from List of scheduled jobs response (p. 163).

This is the structure of the list-jobs job queue endpoint. Please note that the submit job message that was used to submit that job is part of the response. This can be used for tracking or testing / resubmitting purpose. When a job is completed, the start date and end date will also be populated.

```json
[
  {
    "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",
  "jobOnHold": true
}
Release all jobs that are "on hold"

- Supported method: POST
- Path: release-all
- Returns: a message indicating the outcome for the release attempt operation. Two possible cases here:
  - HTTP 200 and a message "All job released with success!" if all jobs were successfully released.
  - HTTP 503 and a message "Jobs not released. An unknown error occured. See log for more details" if something went wrong with the release attempt.

Release all jobs that are "on hold" for a given job name

For a given job name, multiple jobs can be submitted, with different job numbers (the unicity of a job run is garanteed by a couple <job name, job number>). The endpoint will attempt to release all job submissions with the given job name, which are "on hold".

- Supported method: POST
- Path: /release/{name}
- Arguments: the job name to look for, as a string. Mandatory.
- Returns: a message indicating the outcome for the release attempt operation. Two possible cases here:
  - HTTP 200 and a message "Jobs in group <name> (<number of released jobs>) released with success!" jobs were successfully released.
  - HTTP 503 and a message "Jobs in group <name> not released. An unknown error occured. See log for more details" if something went wrong with the release attempt.

Release a given job for a job number

The endpoint will attempt to release the unique job submission which is "on hold", for the given couple <job name, job number>.

- Supported method: POST
- Path: /release/{name}/{number}
- Arguments:
  - name
    - the job name to look for, as a string. Mandatory.
  - number
    - the job number to look for, as an integer. Mandatory.
- Returns
  - a message indicating the outcome for the release attempt operation. Two possible cases here:
    - HTTP 200 and a message "Job <name/number> released with success!" if the job was successfully released.
    - HTTP 503 and a message "Job <name/number> not released. An unknown error occured. See log for more details" if something went wrong with the release attempt.

Blusam Application Console REST endpoints

The Blusam Application Console is an API designed to simplify the management of modernized VSAM datasets. Endpoints for the Blusam web application use the root path /bac.
Data sets related endpoints

Use the following endpoints to create or manage a specific data set.

Create a data set

Create a data set endpoint allows to create a data set definition, and it requires authentication.

- Supported methods: POST
- Path: `/api/services/rest/bluesamservice/createDataSet`
- Arguments:
  - `name` (required, string): the name of the data set.
  - `type` (required, string): the data set type. Possible values are: ESDS, KSDS, RRDS.
  - `recordSize` (optional, string): Maximum size of each record of the data set.
  - `fixedLength` (optional, boolean): Indicates if the records length is fixed.
  - `compression` (optional, boolean): Indicates if the dataset is compressed.
  - `cacheEnable` (optional, boolean): Indicates if caching is enabled for the dataset.
  - `alternativeKeys` (optional, list of keys):
    - `offset` (required, number)
• length (required, number)
• name (required, number)

Returns a json file representing the newly created data set.

Sample request:

```json
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
}
```

Sample response:

```json
{
  "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,
    "alterativeKeys": []
  }
}
```
Upload a file

This endpoint allows to upload files to the server. The file is stored in a temporary folder that corresponds to each specific user. This endpoint should be used each time a file is needed to be uploaded. It requires authentication.

- Supported methods: POST
- Path: `/api/services/rest/bluesamservice/upload`
- Arguments:
  - `file` (required, multipart/form-data): The file to upload.
- Returns a boolean reflecting the status of the upload

Load a data set

Once the data set definition is created using the create data set endpoint described earlier, you can load records that are associated to the uploaded file, to a specific data set. It requires authentication.

- Supported methods: POST
- Path: `/api/services/rest/bluesamservice/loadDataSet`
- Arguments:
  - `name` (required, string): the name of the data set.
- Returns the status of the request and the loaded data set.

Load a data set from an Amazon S3 bucket

Loads a data set using a listcat file from an Amazon S3 bucket.

- Supported methods: GET
- Path: `/api/services/rest/bluesamservice/loadDataSetFromS3`
- Arguments:
  - `listcatFileS3Location` (required, string): the Amazon S3 location of the listcat file.
  - `datasetFileS3Location` (required, string): the Amazon S3 location of the data set file.
  - `region` (required, string): the Amazon S3 AWS Region where the files are stored.
• Returns the newly created data set

Sample request:

/BAC/api/services/rest/bluesamservice/loadDataSetFromS3?region=us-east-1&listcatFileS3Location=s3://bucket-name/listcat.json&datasetFileS3Location=s3://bucket-name/dataset.DAT

Export a data set to an Amazon S3 bucket

Exports a data set to the specified Amazon S3 bucket.

• Supported methods: GET
• Path: /api/services/rest/bluesamservice/exportDataSetToS3
• Arguments:
  s3Location
  (required, string): the Amazon S3 location to export the data set to.
  datasetName
  (required, string): the name of the data set to export.
  region
  (required, string): the AWS Region of the Amazon S3 bucket.
• Returns the exported data set

Sample request:

/BAC/api/services/rest/bluesamservice/exportDataSetToS3?region=eu-west-1&s3Location=s3://bucket-name/dump&datasetName=dataset

Clear a data set

Clears all records from a data set. It requires authentication.

• Supported methods: POST
• Path: /api/services/rest/bluesamservice/clearDataSet
• Arguments:
  name
  (required, string): the name of the data set to clear.
• Returns the status of the request.

Delete a data set

Deletes the data set definiton and records. It requires authentication.

• Supported methods: POST
• Path: /api/services/rest/bluesamservice/deleteDataSet
• Arguments:
name

(required, string): the name of the data set to delete.
• Returns the status of the request and the deleted data set.

Count data set records

This endpoint returns the number of records associated to a data set. It requires authentication.

• Supported methods: POST
• Path: /api/services/rest/bluesamservice/countRecords
• Arguments:
  name

  (required, string): the name of the data set.
• Returns: the number of records

Bulk data sets related endpoints

Use the following endpoints to create or manage multiple data sets at once.

Topics
• Export data sets (p. 140)
• Create multiple data sets (p. 140)
• List all data sets (p. 141)
• Direct List all data sets (p. 141)
• Delete all data sets (p. 141)
• Get Data set definitions from listcat file (p. 141)
• Get data set definitions from uploaded list cat file (p. 142)
• Load listcat from json file (p. 142)

Export data sets

• Supported methods: GET
• Path: /api/services/rest/bluesamservice/exportDataSet
• Arguments:
  name

  (required, string): the name of the data set to delete.
  datasetOutputFile

  (optional, string): the path where to store the exported dataset on the server
  rdw

  (optional, boolean): should the RDW fields be exported.
• returns the status of the request and the file containing the exported data set.

Create multiple data sets

• Supported methods: POST
Blusam Application Console REST endpoints

- Path: /api/services/rest/bluesamservice/createAllDataSets
  - Arguments:
    - List of data sets
      - name
        (required, string): the name of the data set.
      - type
        (required, string): the data set type. Possible values are: ESDS, KSDS, RRDS.
      - recordSize
        (optional, string): Maximum size of each record of the data set.
      - fixedLength
        (optional, boolean): Indicates if the records length is fixed.
      - compression
        (optional, boolean): Indicates if the dataset is compressed.
      - cacheEnable
        (optional, boolean): Indicates if caching is enabled for the dataset.
  - Returns: the status of the request and the newly created data set.

List all data sets

- Supported methods: GET
- Path: /api/services/rest/bluesamservice/listDataSet
- Arguments: None
- Returns: the status of the request and the list of the data sets.

Direct List all data sets

- Supported methods: GET
- Path: /api/services/rest/bluesamservice/directListDataSet
- Arguments: None
- Returns: the status of the request and the list of the data sets.

Delete all data sets

- Supported methods: POST
- Path: /api/services/rest/bluesamservice/removeAll
- Arguments: None
- Returns: a boolean representing the status of the request.

Get Data set definitions from listcat file

- Supported methods: POST
- Path: /api/services/rest/bluesamservice/getDataSetsDefinitionFromListcat
- Arguments:
paramFilePath

(required, string): The path to the listcat file.
- Returns: a list of data sets

Get data set definitions from uploaded list cat file

- Supported methods: POST
- Path: `/api/services/rest/bluesamservice/getDataSetsDefinitionFromUploadedListcat`
- Arguments: None
- Returns: a list of data sets

Load listcat from json file

- Supported methods: GET
- Path: `/api/services/rest/bluesamservice/loadListcatFromJsonFile`
- Arguments:
  - filePath
    - (required, string): The path to the listcat file.
- Returns: a list of data sets

Records

Use the following endpoints to create or manage records within a data set.

Topics

- Create a record (p. 142)
- Read a data set (p. 143)
- Delete a record (p. 143)
- Update a record (p. 143)
- Save a record (p. 143)

Create a record

This endpoint allows to create a new record. It requires authentication.

- Supported methods: POST
- Path: `/api/services/rest/crud/createRecord`
- Arguments:
  - dataset
    - (required, DataSet): the data set object
  - mask
    - (required, mask): the mask object.
- Returns the status of the request and the created record.
Read a data set

This endpoint allows to read a data set.

- Supported methods: GET
- Path: /api/services/rest/crud/readDataSet
- Arguments:
  dataset
    (required, DataSet): the data set object.
- Returns the status of the request and the data set with the records.

Delete a record

This endpoint allows to delete a record from a data set. It requires authentication.

- Supported methods: DELETE
- Path: /api/services/rest/crud/deleteRecord
- Arguments:
  dataset
    (required, DataSet): the data set object
  record
    (required, Record): the record to delete
- Returns the status of the deletion.

Update a record

This endpoint allows to update a record associated to a data set. It requires authentication.

- Supported methods: POST
- Path: /api/services/rest/crud/updateRecord
- Arguments:
  dataset
    (required, DataSet): the data set object
  record
    (required, Record): the record to update
- Returns the status of the request and the data set with the records.

Save a record

This endpoint allows to save a record to a data set and using a mask. It requires authentication.

- Supported methods: POST
- Path: /api/services/rest/crud/saveRecord
- Arguments:
  dataset
    (required, DataSet): the data set object
record

(required, Record): the record to save

• Returns the status of the request and the data set with the records.

## Masks

Use the following endpoints to load or apply masks to a data set.

### Topics

- Load masks (p. 144)
- Apply mask (p. 144)
- Apply mask filter (p. 144)

### Load masks

This endpoint allows to retrieve all the masks that are associated to a specific data set. It requires authentication.

• Supported methods: POST
• Path: /api/services/rest/crud/loadMasks
• Arguments:
  dataset

  (required, DataSet): the data set object

• Returns the status of the request and the list of the masks.

### Apply mask

This endpoint allows to apply a mask to a specific data set. It requires authentication.

• Supported methods: POST
• Path: /api/services/rest/crud/applyMask
• Arguments:
  dataset

  (required, DataSet): the data set object

  mask

  (required, Mask): the data set object

• Returns the status of the request and the data set with the applied mask.

### Apply mask filter

This endpoint allows to apply a mask and a filter to a specific data set. It requires authentication.

• Supported methods: POST
• Path: /api/services/rest/crud/applyMaskFilter
• Arguments:
dataset
(required, DataSet): the data set object
mask
(required, Mask): the data set object

- Returns the status of the request and the data set with the applied mask and filter.

**Other**

Use the following endpoints to manage cache for a data set or check data set characteristics

**Topics**
- Check warm up cache (p. 145)
- Check cache enabled (p. 145)
- Enable cache (p. 145)
- Check allocated Ram cache (p. 146)
- Check persistence (p. 146)
- Check supported data set types (p. 146)
- Check server health (p. 146)

**Check warm up cache**

Checks if the warmup cache is enabled for a specific data set. It requires authentication.

- Supported methods: POST
- Path: `/api/services/rest/bluesamservice/warmupCache`
- Arguments:
  name
  (required, string): the name of the data set.
- Returns: true if the warm up cache is enabled and false otherwise.

**Check cache enabled**

Checks if the cache is enabled for a specific data set. It requires authentication.

- Supported methods: GET
- Path: `/api/services/rest/bluesamservice/isEnableCache`
- Arguments: None
- Returns true if the caching is enabled.

**Enable cache**

- Supported methods: GET
- Path: `/api/services/rest/bluesamservice/enableDisableCache/{enable}`
- Arguments:
  enable
  (required, boolean): if set to true, it will enable the caching.
• Returns None

**Check allocated Ram cache**

This endpoint allows to retrieve the allocated RAM cache memory. It requires authentication.

• Supported methods: GET
• Path: /api/services/rest/bluesamservice/allocatedRamCache
• Arguments: None
• Returns: the size of the memory as a string

**Check persistence**

• Supported methods: GET
• Path: /api/services/rest/bluesamservice/persistence
• Arguments: None
• Returns: the persistence used as a string

**Check supported data set types**

• Path: /api/services/rest/bluesamservice/getDataSetTypes
• Arguments: None
• Returns: the list of supported data set types as a list of strings.

**Check server health**

• Supported methods: GET
• Path: /api/services/rest/bluesamservice/serverIsUp
• Arguments: None
• Returns: None

**Users**

Use the following endpoints to manage user interactions.

**Topics**

• [Login](p. 146)
• [Check user account](p. 147)
• [Sign on](p. 147)
• [List all users](p. 147)
• [Logout](p. 147)

**Login**

• Supported methods: POST
• Path: /api/services/security/service/login
• Arguments:
username
(required, string)
password
(required, string)
• Returns the username and the roles of the logged in user

Sample response

```json
{"login":"some-user","roles":[{"id":0,"roleName":"ROLE_ADMIN"}]}
```

**Check user account**

- Supported methods: POST
- Path: /api/services/security/servicelogin/hasAccount
- Arguments: None
- Returns: true if the user is already logged in

**Sign on**

- Supported methods: POST
- Path: /api/services/security/servicelogin/recorduser
- Arguments: None
- Returns: true if the user is already logged in

**List all users**

- Supported methods: GET
- Path: /api/services/security/servicelogin/listusers
- Arguments: None
- Returns: the list of all users

**Logout**

- Supported methods: POST
- Path: /api/services/security/servicelogout/logout
- Arguments: None
- Returns: true if the user has been logged out successfully.

**JICS Application Console**

The JICS component is the AWS Blu Age support for modernization of the legacy CICS resources. The JICS Application Console web-application is dedicated to administrate JICS resources. The following endpoints allow to perform the administration tasks without having to interact with the JAC user interface. Whenever an endpoint requires authentication, the request will have to include authentication details (username/password typically, as required by Basic Authentication). Endpoints for the JICS Appliation Console web application use the root path /jac/.
JICS Resources management

All following endpoints are related to JICS resources management, allowing JICS administrators to deal with resources on daily basis.

List JICS LISTS and GROUPS

The LIST and GROUPS are the main owning container resources within the JICS component. All JICS resources must belong to a GROUP. Groups can belong to LISTS, but this is not mandatory. LISTS might
even not exist on a given JICS environment, but most of the time, LISTS are there to give an extra layer of organization for resources. For more information about the CICS resources organization, see CICS resources.

- Supported method: GET
- Requires authentication
- Path: /api/services/rest/jicsservice/listJicsListsAndGroups
- Returns: a list of serialized JicsContainer objects, both LISTS and GROUPS, as JSON.

Sample response:

```
[
  {
    "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
  }
]
```

**List JICS GROUPS**

- Supported method: GET
- Requires authentication
- Path: /api/services/rest/jicsservice/listJicsGroups
- Returns: a list of serialized JicsContainer objects (GROUPS) as JSON. The GROUPS are being returned without their owning LIST information.

Sample response:

```
[
  {
    "jacType": "JACGroup",
    "name": "MURACHS",
    "isActive": true,
    "children": []
  },
  {
    "jacType": "JACGroup",
    "name": "OTHER",
    "isActive": true,
    "children": []
  }
]
```
List JICS GROUPS for a given LIST

- Supported method: POST
- Requires authentication
- Path: `/api/services/rest/jicsservice/listGroupsForList`
- Arguments: a JSON payload, representing the JICS LIST whose GROUPS we're looking for. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACList` object.

Sample request:

```
{
   "jacType": "JACList",
   "name": "MURACHS",
   "isActive": true
}
```

- Returns: a list of serialized JicsContainer objects (GROUPS) as JSON, that are attached to the given LIST. The GROUPS are being returned without their owning LIST information.

Sample response:

```
[
   {
      "jacType": "JACGroup",
      "name": "MURACHS",
      "isActive": true,
      "children": []
   }
]
```

LIST JICS resources for a given GROUP

- Supported method: POST
- Requires authentication
- Path: `/api/services/rest/jicsservice/listResourcesForGroup`
- Arguments: a JSON payload, representing the JICS GROUP whose resources we're looking for. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACGroup` object. You do not need to specify all fields for the GROUP, but the name is mandatory.

Sample request:

```
{
   "jacType": "JACGroup",
   "name": "MURACHS",
   "isActive": true
}
```

- Returns: a list of serialized JicsResource objects, owned by the given GROUP. The objects are being returned in no particular order and are of different types (PROGRAM, TRANSACTION, FILE, etc ...).
LIST JICS resources for a given GROUP (alternative using a name)

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/listResourcesForGroupName
- Arguments: the name of the GROUP owning the resources we're looking for.
- Returns: a list of serialized JicsResource objects, owned by the given GROUP. The objects are being returned in no particular order and are of different types (PROGRAM, TRANSACTION, FILE, etc ...)

Editing the owned GROUPS of several LISTS

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/editGroupsList
- Arguments: a JSON representation of a collection of LISTS with children GROUPS;
  
  Sample request:

  ```json
  [
    {
      "jacType": "JACList",
      "name": "MURACHS",
      "isActive": true,
      "children": [
        {
          "jacType": "JACGroup",
          "name": "MURACHS",
          "isActive": true,
          "children": []
        },
        {
          "jacType": "JACGroup",
          "name": "TEST",
          "isActive": true,
          "children": []
        }
      ]
    }
  ]
  ```

  Prior to this editing, only the group named "MURACHS" was belonging to the LIST named "MURACHS". With this editing, we "add" the group named "TEST" to the LIST named "MURACHS".

  - Returns a boolean value. If the value is 'true', the LISTS modifications have been properly persisted to the underlying JICS storage.

Delete a LIST

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/deleteList
- Arguments: a JSON payload, representing the JICS LIST to delete. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACList object.
- Returns a boolean value. If the value is 'true', the LIST deletion have been properly operated on the underlying JICS storage.
Delete a GROUP

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/deleteGroup
- Arguments: a JSON payload, representing the JICS GROUP to delete. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACGroup object.
- Returns a boolean value. If the value is 'true', the GROUP deletion have been properly operated on the underlying JICS storage.

Delete a TRANSACTION

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/deleteTransaction
- Arguments: a JSON payload, representing the JICS Transaction to delete. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACTransaction object.
- Returns a boolean value. If the value is 'true', the TRANSACTION deletion have been properly operated on the underlying JICS storage.

Delete a PROGRAM

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/deleteProgram
- Arguments: a JSON payload, representing the JICS Program to delete. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACProgram object.
- Returns a boolean value. If the value is 'true', the PROGRAM deletion have been properly operated on the underlying JICS storage.

Delete a FILE

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/deleteFile
- Arguments: a JSON payload, representing the JICS File to delete. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACFile object.
- Returns a boolean value. If the value is 'true', the FILE deletion have been properly operated on the underlying JICS storage.

Delete a TDQUEUE

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/deleteTDQueue
- Arguments: a JSON payload, representing the JICS TDQUEUE to delete. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACTDQueue` object.
• Returns a boolean value. If the value is 'true', the TDQUEUE deletion have been properly operated on the underlying JICS storage.

Delete a TSMODEL

• Supported method: POST
• Requires authentication
• Path: /api/services/rest/jicsservice/deleteTSModel
• Arguments: a JSON payload, representing the JICS TSMODEL to delete. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACTSModel` object.
• Returns a boolean value. If the value is 'true', the TSMODEL deletion have been properly operated on the underlying JICS storage.

Create a LIST

• Supported method: POST
• Requires authentication
• Path: /api/services/rest/jicsservice/createList
• Arguments: a JSON payload, representing the JICS LIST to create. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACList` object.
• Returns a boolean value. If the value is 'true', the LIST has been properly created in the underlying JICS storage.

Note
The LIST will always be created empty. Attaching GROUPS to the LIST will require another operation.

Create a GROUP

• Supported method: POST
• Requires authentication
• Path: /api/services/rest/jicsservice/createGroup
• Arguments: a JSON payload, representing the JICS GROUP to create. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACGroup` object.
• Returns a boolean value. If the value is 'true', the GROUP has been properly created in the underlying JICS storage.

Note
The GROUP will always be created empty. Attaching RESOURCES to the GROUP will require additional operations (creating resources will automatically attach them to a given GROUP.

Common RESOURCES creation considerations

All the following endpoints are related to JICS RESOURCES creation and share some common constraints: in the request payload to be sent to the endpoint, the groupName field has to be valued.

GROUP ownership constraint:

No resource can be created without being attached to an existing group, and the endpoint uses the groupName to retrieve the group to which this resource will be attached. The groupName must point to
the name of an existing GROUP. An error message with HTTP STATUS 400 will be sent if the `groupName` is not pointing at an existing group in the JICS underlying storage.

Unicity constraint within a GROUP:

A given resource with a given name must be unique within a given group. The check for unicity will be performed by each resource creation endpoint. If the given payload does not respect the unicity constraint, the endpoint will send a response with HTTP STATUS 400 (BAD REQUEST) -- see the sample response below.

Sample payload: we try to create the transaction 'ARIT' in the 'TEST' group, but a transaction with that name already exists in this GROUP.

```json
{
    "jacType":"JACTransaction",
    "name":"ARIT",
    "groupName":"TEST",
    "isActive":true
}
```

We received the following error response:

```json
{
    "timestamp": 1686759054510,
    "status": 400,
    "error": "Bad Request",
    "path": "/jac/api/services/rest/jicsservice/createTransaction"
}
```

Inspecting servers logs will confirm the origin of the issue:

```
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(JicsServiceImpl.java:1280)
```

Create a TRANSACTION

- Supported method: POST
- Requires authentication
- Path: `/api/services/rest/jicsservice/createTransaction`
- Arguments: a JSON payload, representing the JICS TRANSACTION to create. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACTransaction` object.
- Returns a boolean value. If the value is 'true', the TRANSACTION has been properly created in the underlying JICS storage.

Create a PROGRAM

- Supported method: POST
- Requires authentication
- Path: `/api/services/rest/jicsservice/createProgram`
Arguments: a JSON payload, representing the JICS PROGRAM to create. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACProgram` object. 
Returns a boolean value. If the value is 'true', the PROGRAM has been properly created in the underlying JICS storage.

Create a FILE

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/createFile
- Arguments: a JSON payload, representing the JICS FILE to create. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACFile` object.
- Returns a boolean value. If the value is 'true', the FILE has been properly created in the underlying JICS storage.

Create a TDQUEUE

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/createTDQueue
- Arguments: a JSON payload, representing the JICS TDQUEUE to create. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACTDQueue` object.
- Returns a boolean value. If the value is 'true', the TDQUEUE has been properly created in the underlying JICS storage.

Create a TSMODEL

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/createTSModel
- Arguments: a JSON payload, representing the JICS TSMODEL to create. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACTSModel` object.
- Returns a boolean value. If the value is 'true', the TSMODEL has been properly created in the underlying JICS storage.

Update a LIST

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/updateList
- Arguments: a JSON payload, representing the JICS LIST to update. This is the JSON serialization of a `com.netfective.bluage.jac.entities.JACList` object. There's no need to supply the children of the LIST, the LIST update mechanism won't take this into account.
- Returns a boolean value. If the value is 'true', the LIST has been properly updated in the underlying JICS storage.

Updating the LIST 'isActive' flag will propagate to all owned elements of the LIST, that is, all GROUPS owned by the LIST and all RESOURCES owned by those GROUPS. This is a convenient way of disactivating a lot of resources with a single operation, over several GROUPS.
Update a GROUP

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/updateGroup
- Arguments: a JSON payload, representing the JICS GROUP to update. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACGroup object. There's no need to supply the children of the GROUP, the GROUP update mechanism won't take this into account.
- Returns a boolean value. If the value is 'true', the GROUP has been properly updated in the underlying JICS storage.

**Note**
Updating the GROUP 'isActive' flag will propagate to all owned elements of the GROUP, that is, all RESOURCES owned by the GROUP. This is a convenient way of disactivating a lot of resources with a single operation within a given GROUP.

Common RESOURCES update considerations

All following endpoints are about updating JICS RESOURCES. Using the groupName field, you can change the owning GROUP of any JICS RESOURCE, provided the field value points to an existing GROUP in the underlying JICS storage (otherwise, you will get a BAD REQUEST response (HTTP STATUS 400) from the endpoint).

Update a TRANSACTION

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/updateTransaction
- Arguments: a JSON payload, representing the JICS TRANSACTION to update. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACTransaction object.
- Returns a boolean value. If the value is 'true', the TRANSACTION has been properly updated in the underlying JICS storage.

Update a PROGRAM

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/updateProgram
- Arguments: a JSON payload, representing the JICS PROGRAM to update. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACProgram object.
- Returns a boolean value. If the value is 'true', the PROGRAM has been properly updated in the underlying JICS storage.

Update a FILE

- Supported method: POST
- Requires authentication
- Path: /api/services/rest/jicsservice/updateFile
- Arguments: a JSON payload, representing the JICS FILE to update. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACFile object.
• Returns a boolean value. If the value is 'true', the FILE has been properly updated in the underlying JICS storage.

Update a TDQUEUE
• Supported method: POST
• Requires authentication
• Path: /api/services/rest/jicsservice/updateTDQueue
• Arguments: a JSON payload, representing the JICS TDQUEUE to update. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACTDQueue object.
• Returns a boolean value. If the value is 'true', the TDQueue has been properly updated in the underlying JICS storage.

Update a TSMODEL
• Supported method: POST
• Requires authentication
• Path: /api/services/rest/jicsservice/updateTSModel
• Arguments: a JSON payload, representing the JICS TSMODEL to update. This is the JSON serialization of a com.netfective.bluage.jac.entities.JACTSModel object.
• Returns a boolean value. If the value is 'true', the TSMODEL has been properly updated in the underlying JICS storage.

JAC Users management endpoints

Topics
• Deleting an user (p. 157)
• Logging a user (p. 158)
• Testing if at least an user exists in the system (p. 158)
• Recording a new user (p. 158)
• Listing users (p. 159)
• Listing users (p. 159)
• Logout the current user (p. 159)
• Jics server health status (p. 159)

Deleting an user
• Supported method: POST
• Requires authentication and ADMIN rights
• Path: /api/services/security/servicelogin/deleteuser
• Arguments: the JSON serialization of a com.netfective.bluage.jac.entities.SignOn object, representing the user to be removed from the storage.
• Returns a boolean value. If the value is 'true', the user has been properly removed from the JICS system.

Note
This action cannot be undone, deleted user won't be able to connect to JAC application again. Use with caution.
Logging a user

- Supported method: POST
- Requires authentication and ADMIN rights
- Path: /api/services/security/servicelogin/login
- Returns the JSON serialization of a com.netfective.bluage.jac.entities.SignOn object, representing the user whose credentials are provided in the current request. The password is hidden from the view in the returned object. The roles given to the used are being listed.

Sample response:

```
{
    "login": "sadmin",
    "password": null,
    "roles": [
        {
            "id": 0,
            "roleName": "ROLE_SUPER_ADMIN"
        }
    ]
}
```

Testing if at least an user exists in the system

- Supported method: GET
- Path: /api/services/security/servicelogin/hasAccount
- Returns a boolean value, whose value is true if at least an user -- other than the default super admin user -- has been created in the JICS system.

Recording a new user

- Supported method: POST
- Requires authentication and ADMIN rights
- Path: /api/services/security/servicelogin/recorduser
- Arguments: the JSON serialization of a com.netfective.bluage.jac.entities.SignOn object, representing the user to be added to the storage. The Roles for the user should be defined, otherwise the user might not be able to use the JAC facility and endpoints.

Sample request:

```
{
    "login": "simpleuser",
    "password": "simpleuser",
    "roles": [
        {
            "id": 2,
            "roleName": "ROLE_USER"
        }
    ]
}
```

Only the following roles can be used when recording a new user:

- ROLE_ADMIN : can manage JICS resources and users.
• ROLE_USER: can manage JICS resources but not users.

Listing users
• Supported method: GET
• Requires authentication and ADMIN rights
• Path: /api/services/security/servicelogin/listusers
• Returns a list of com.netfective.bluage.jac.entities.SignOn, serialized as JSON.

Logout the current user
• Supported method: GET
• Path: /api/services/security/servicelogout/logout
• Returns the following JSON message: "{"success":true}" if logout of current user was successful (the related HTTP session will be invalidated).

Jics server health status
• Supported method: GET
• Path: /api/services/rest/jicsserver/serverIsUp
• Indicates that the JICS server is up and running, if you get a response having HTTP STATUS 200.

Data Structures
This section describes details of the various data structures.

Topics
• Job Execution Details message structure (p. 159)
• Transaction launch outcome structure (p. 161)
• Transaction launch record outcome structure (p. 161)
• Possible status of a job on a queue (p. 162)
• Submit job input (p. 162)
• List of scheduled jobs response (p. 163)
• List of "on hold" jobs response (p. 164)

Job Execution Details message structure
Each job execution details will have the following fields:
scriptId
the identifier of the called script.
caller
I.P. address of the caller.
identifier
unique job execution identifier.
startTime
date and time at which the job execution started.
endTime
date and time at which the job execution ended.
status
a status for the job execution. One possible value amongst:
• DONE: job execution ended normally.
• TRIGGERED: job execution triggered but not launched yet.
• RUNNING: job execution is running.
• KILLED: job execution has been killed.
• FAILED: job execution has failed.
executionResult
a message to sum up the outcome of the job execution. This message can either be a simple message if the job execution is not finished yet or a JSON structure with the following fields:
• exitCode: numeric exit code; negative values indicate failure situations.
• program: latest program launched by the job.
• status: one possible value amongst:
  • Error: when exitCode = -1; this corresponds to an (technical) error occurring during job execution.
  • Failed: when exitcode = -2; This corresponds to a failure occurring during a service program execution (like an ABEND situation).
  • Succeeded: when exitCode >= 0;
• stepName: name of the latest step executed in the job.
executionMode
either SYNCHRONOUS or ASYNCHRONOUS, depending on the way the job has been launched.

Sample output:

```json
{
  "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"
}
```
Transaction launch outcome structure

The structure might contain the following fields:

outCome
  a string representing the transaction execution outcome. Possible values are:
  • Success: transaction execution went to the end properly.
  • Failure: transaction execution failed to end properly, some problem(s) were encountered.

commarea
  a string representing the COMMAREA final value, as a byte64 encoded byte array. Might be an empty string.

containerRecord
  (optional) a string representing the CONTAINER's record content as a byte64 encoded byte array.

serverDescription
  May contain information about the server which served the request (for debugging purpose). Might be an empty string.

abendCode
  (optional) if the program referenced by the launched transaction abended, the abend code value will be returned as a string in this field.

Sample responses:

Success

```
{
  "outCome": "Success",
  "commarea": "",
  "serverDescription": ""
}
```

Failure

```
{
  "outCome": "Failure",
  "commarea": "",
  "serverDescription": "",
  "abendCode": "AEIA"
}
```

Transaction launch record outcome structure

The structure might contain the following fields:

recordContent
  a string representing the COMMAREA's record content as a byte64 encoded byte array.

containerRecord
  a string representing the CONTAINER's record content as a byte64 encoded byte array.

serverDescription
  May contain information about the server which served the request (for debugging purpose). Might be an empty string.
Sample responses:
Success

```
{
   "recordContent": ",",
   "serverDescription": ""
}
```

**Possible status of a job on a queue**

On a queue, jobs can have the following status:

**ACTIVE**

The job is currently being run on the queue.

**EXECUTION_WAIT**

The job is waiting for a thread to be available.

**SCHEDULED**

Jobs is scheduled for execution at a specific date and time.

**HOLD**

Job is waiting to be released before being run.

**COMPLETED**

Job has been executed successfully.

**FAILED**

Job execution has failed.

**UNKNOWN**

Status is unknown.

**Submit job input**

The submit job input is the JSON serialization of a com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage object. The sample input below exhibits all the fields for such a bean.

Sample input:

```
{
   "messageQueueName"":null,
   "scheduleDate":null,
   "scheduleTime":null,
   "programName":"PTA0044",
   "programParams":
   {
      "wmind":"
   },
   "localDataAreaValue":",
   "userName":"USER1",
   "jobName":"PTA0044",
   "jobNumber":9,
   "jobPriority":5,
   "executionDate":"20181231",
   "jobQueue":"queue1",
   "jobOnHold":false
}
```
jobNumber

if the jobnumber is 0, the job number will be automatically generated using the next number in the job number sequence. That value should be set to 0 (except for testing purpose).

jobPriority

Default job priority in AS400 is 5. Valid range is 0-9, 0 being the highest priority.

jobOnHold

If a job is submitted on hold, it won't be executed right away but only when somebody “releases” it. A job can be released using the REST API (/release or /release-all).

scheduleDate and scheduleTime

If these values are not null, the job will be executed at the specified date and time.

Date
can be provided with format MMddyy or ddMMyyyy (size of the input will determine what format is used)

Time
can be provided with format HHmm or HHmmss (size of the input will determine what format is used)

programParams

this will be passed to the program as a map.

List of scheduled jobs response

This is the structure of the list-jobs job queue endpoint. Please note that the submit job message that was used to submit that job is part of the response. This can be used for tracking or testing / resubmitting purpose. When a job is completed, the start date and end date will also be populated.

```
[  
  {  
    "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",  
```
List of "on hold" jobs response

The structure is similar to the previous one, except that all returned jobs will be "on hold".
Modify the source code with Blu Age Developer IDE

If you are using the AWS-managed AWS Blu Age runtime engine, you can use Blu Age Developer to modify the generated source code. You might want to do this if you need to update the modernized code for some reason, or if a portion of the legacy source code couldn't be modernized. You access Blu Age Developer through Amazon AppStream 2.0. This section describes how to set up Blu Age Developer on AppStream 2.0. It also explains how to use Blu Age Developer to update source code, using the sample application PlanetsDemo.

Topics
- Tutorial: Set up AppStream 2.0 for AWS Blu Age Developer IDE (p. 165)
- Tutorial: Use AWS Blu Age Developer on AppStream 2.0 (p. 168)

Tutorial: Set up AppStream 2.0 for AWS Blu Age Developer IDE

AWS Mainframe Modernization provides several tools through Amazon AppStream 2.0. AppStream 2.0 is a fully managed, secure application streaming service that lets you stream desktop applications to users without rewriting applications. AppStream 2.0 provides users with instant access to the applications that they need with a responsive, fluid user experience on the device of their choice. Using AppStream 2.0 to host runtime engine-specific tools gives customer application teams the ability to use the tools directly from their web browsers, interacting with application files stored in either Amazon S3 buckets or CodeCommit repositories.

For information about browser support in AppStream 2.0 see System Requirements and Feature Support (Web Browser) in the Amazon AppStream 2.0 Administration Guide. If you have issues when you are using AppStream 2.0 see Troubleshooting AppStream 2.0 User Issues in the Amazon AppStream 2.0 Administration Guide.

This document describes how to set up AWS Blu Age Developer IDE on an AppStream 2.0 fleet.

Topics
- Prerequisite (p. 166)
- Step 1: Create an Amazon S3 bucket (p. 166)
- Step 2: Attach a policy to the S3 bucket (p. 166)
- Step 3: Upload files to the Amazon S3 bucket (p. 166)
- Step 4: Download AWS CloudFormation templates (p. 166)
- Step 5: Create the fleet with AWS CloudFormation (p. 166)
- Step 6: Access an instance (p. 167)
- Clean up resources (p. 168)
Prerequisite

Download the archive file that contains the artifacts that you need to set up AWS Blu Age Developer IDE under AppStream 2.0.

Note

This is a large file. If you have problems with the operation timing out, we recommend using an Amazon EC2 instance to improve the upload and download performance.

Step 1: Create an Amazon S3 bucket

Create an Amazon S3 bucket in the same AWS Region as the AppStream 2.0 fleet that you will create. This bucket will contain the artifacts that you need to complete this tutorial.

Step 2: Attach a policy to the S3 bucket

Attach the following policy to the bucket that you create for this tutorial. Make sure to replace MYBUCKET with the actual name of the bucket that you create.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowAppStream2.0ToRetrieveObjects",
         "Effect": "Allow",
         "Principal": {
            "Service": "appstream.amazonaws.com"
         },
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::MYBUCKET/*"
      }
   ]
}
```

Step 3: Upload files to the Amazon S3 bucket

Unzip the files you downloaded in the Prerequisite and upload the appstream folder to your bucket. Uploading this folder creates the correct structure in your bucket. For more information, see Uploading objects in the Amazon S3 User Guide.

Step 4: Download AWS CloudFormation templates

Download the following AWS CloudFormation templates. You need these templates to create and populate the AppStream 2.0 fleet.

- `cfn-m2-appstream-elastic-fleet-linux.yaml`
- `cfn-m2-appstream-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`

Step 5: Create the fleet with AWS CloudFormation

In this step, you use the `cfn-m2-appstream-elastic-fleet-linux.yaml` AWS CloudFormation template to create an AppStream 2.0 fleet and stack to host the AWS Blu Age Developer IDE. After you
create the fleet and stack, you will run the other AWS CloudFormation templates you downloaded in the previous step to install the Developer IDE and other required tools.

1. Navigate to AWS CloudFormation in the AWS Management console, and choose **Stacks**.
2. In **Stacks**, choose **Create stack** and With new Resources (standard):

3. In **Create stack**, choose **Template is ready** and **Upload a template file**:

4. Choose **Choose file**, and navigate to file `cfn-m2-appstream-elastic-fleet-linux.yaml`. Choose **Next**.

5. In **Specify stack details**, provide the following information:
   - A name for the stack.
   - Your default security group and two subnets of that security group.

6. Choose **Next**, and then choose **Next** again.

7. Choose **I acknowledge that AWS CloudFormation might create IAM resources with custom names.**, and then choose **Submit**.

8. After you create the fleet, create CloudFormation stacks with the other downloaded templates to finish setting up the applications. Make sure to update **BucketName** each time to point to the correct S3 bucket. You can edit the **BucketName** in the CloudFormation console. Alternatively, you can edit the template files directly and update the **S3Bucket** property.

   **Note**
   The downloaded templates expect to find assets in an S3 bucket with a folder structure called `appstream/bluage/developer-ide/`. The bucket must be in the same AWS Region as the fleet that you created.

**Step 6: Access an instance**

After you create and start the fleet, you can create a temporary link to access the fleet through the native client.

1. Navigate to AppStream 2.0 in the AWS Management Console and choose the previously created stack:
2. On the stack details page, choose **Action**, then choose **Create Streaming URL**:

![Create Streaming URL](image)

3. In **Create Streaming URL**, enter an arbitrary User ID and a URL expiration time, and then choose **Get URL**. You get an URL that you can use to stream to a browser or into the native client. We recommend that you stream into the native client.

### Clean up resources

For the procedure to clean up the created stack and fleets, see [Create an AppStream 2.0 Fleet and Stack](#).

When you've deleted the AppStream 2.0 objects, you or the account administrator can also clean up the S3 buckets for Application Settings and Home Folders.

**Note**
The home folder for a given user is unique across all fleets, so you might need to retain it if other AppStream 2.0 stacks are active in the same account.

You can't use the AppStream 2.0 console to delete users. Instead, you must use the service API with the AWS CLI. For more information, see [User Pool Administration](#) in the *Amazon AppStream 2.0 Administration Guide*.

### Tutorial: Use AWS Blu Age Developer on AppStream 2.0

This tutorial shows you how to access AWS Blu Age Developer on AppStream 2.0 and use it with a sample application so you can try out the features. When you finish this tutorial, you can use the same steps with your own applications.

**Topics**

- [Step 1: Create a database](#)
- [Step 2: Access the environment](#)
- [Step 3: Set up the runtime](#)
- [Step 4: Start the Eclipse IDE](#)
- [Step 5: Set up a Maven project](#)
- [Step 6: Configure a Tomcat server](#)
- [Step 7: Deploy to Tomcat](#)
Step 1: Create a database

In this step, you use Amazon RDS to create a managed PostgreSQL database that the demo application uses to store configuration information.

1. Open the Amazon RDS console.
2. Choose Databases > Create database.
3. Choose Standard create > PostgreSQL, leave the default version, and then choose Free tier.
4. Choose a DB instance identifier.
5. For Credential Settings, choose Manage master credentials in AWS Secrets Manager. For more information, see Password management with Amazon RDS and AWS Secrets Manager in the Amazon RDS User Guide.
6. Ensure that the VPC is the same as the one that you use for the AppStream 2.0 instance. You can ask your admin for this value.
7. For VPC security group, choose Create New.
8. Leave Public access set to No.
9. Leave all other default values. Review these values.
10. Choose Create database.

To make the database server accessible from your instance, select the database server in Amazon RDS. Under Connectivity & security, choose the VPC security group for the database server. This security group was previously created for you and should have a description similar to the one in Created by RDS management console. Choose Action > Edit inbound rules, choose Add rule, and create a rule of type PostgreSQL. For rule source, use the security group default. You can start to type the source name in the Source field and then accept the suggested ID. Finally, choose Save rules.

Step 2: Access the environment

In this step, you access the AWS Blu Age development environment on AppStream 2.0.

1. Contact your administrator for the proper way to access your AppStream 2.0 instance. For general information about possible clients and configurations, see AppStream 2.0 Access Methods and Clients in the Amazon AppStream 2.0 Administration Guide. Consider using the native client for the best experience.
2. In AppStream 2.0 choose Desktop.

Step 3: Set up the runtime

In this step, you set up the AWS Blu Age runtime. You must set up the runtime at first launch and again if you are notified of a runtime upgrade. This step populates your .m2 folder.

1. Choose Applications, from the menu bar, and then choose Terminal.
2. Enter the following command:

```bash
~/_install-velocity-runtime.sh
```
Step 4: Start the Eclipse IDE

In this step, you start the Eclipse IDE and choose a location where you want to create a workspace.

1. In AppStream 2.0 choose the Launch Application icon on the toolbar, and then choose Eclipse JEE.

2. When the launcher opens, enter the location where you want to create your workspace, and choose Launch.
Optionaly, you can launch Eclipse from the command line, as follows:

```
~/eclipse &
```

**Step 5: Set up a Maven project**

In this step, you import a Maven project for the Planets demo application.

1. Upload `PlanetsDemo-pom.zip` to your Home folder. You can use the native client “My Files” feature to do this.
2. Use the `unzip` command line tool to extract the files.
3. Navigate inside the unzipped folder and open the root `pom.xml` of your project in a text editor.
4. Edit the `gapwalk.version` property so that it matches the installed AWS Blu Age runtime.

   If you are unsure of the installed version, issue the following command in a terminal:

   ```
   cat ~/runtime-version.txt
   ```

   This command prints the currently available runtime version, for example, `3.1.0-b3257-dev`.

   **Note**
   Don't include the `-dev` suffix in `gapwalk.version`. For example, a valid value would be `<gapwalk.version>3.1.0-b3257</gapwalk.version>`.

5. In Eclipse, choose File, then Import. In the Import, dialog window, expand Maven and choose Existing Maven Projects. Choose Next.
6. In **Import Maven Projects**, provide the location of the extracted files and choose Finish.

You can safely ignore the following popup. Maven downloads a local copy of node.js to build the Angular (`*-web`) part of the project:
Wait until the end of the build. You can follow the build in the Progress View.

7. In Eclipse, select the project and choose Run as. Then choose Maven install. After the Maven installation succeeds, it creates the war file under PlanetsDemoPom/PlanetsDemo-web/target/PlanetsDemo-web-1.0.0.war.

**Step 6: Configure a Tomcat server**

In this step, you configure a Tomcat server where you deploy and start your compiled application.

1. In Eclipse, choose Window > Show View > Servers to show the Servers view:

   ![Missing node.js dialog box](image)

   *Cannot find node.js. This will result in editors missing key features. Please make sure node.js is installed and that your PATH environment variable contains the location to the ‘node’ executable.*

   ![No servers are available. Click this link to create a new server...](image)

   Choose No servers are available. Click this link to create a new server... The New Server wizard appears. In the Select the server type field of the wizard, enter tomcatv9, and choose Tomcat v9.0 Server. Then choose Next.
3. Choose **Browse**, and choose the **tomcat** folder at the root of the Home folder. Leave the JRE at its default value and choose **Finish**.

A **Servers** project is created in the workspace, and a Tomcat v9.0 server is now available in the **Servers** view. This is where the compiled application will be deployed and started:

---

**Step 7: Deploy to Tomcat**

In this step, you deploy the Planets demo application to the Tomcat server so you can run the application.
1. Select the PlanetsDemo-web file and choose **Run As > Maven install**. Select PlanetsDemo-web again and choose **Refresh** to ensure that the npm-compiled frontend is properly compiled to a .war and noticed by Eclipse.

2. Upload the [PlanetsDemo-runtime.zip](#) to the instance, and unzip the file at an accessible location. This ensures that the demo application can access the configuration folders and files that it requires.

3. Copy the contents of PlanetsDemo-runtime/tomcat-config into the Servers/Tomcat v9.0... subfolder that you created for your Tomcat server:

4. Open the **tomcat v9.0** server entry in the Servers view. The server properties editor appears:

5. In the **Overview** tab, increase the **Timeouts** values to 450 seconds for Start, and 150 seconds for Stop, as shown here:
6. Choose **Open launch configuration**. A wizard appears. In the wizard, navigate to the **Arguments** folder and, for **Working directory**, choose **Other**. Choose **File System**, and navigate to the PlanetsDemo-runtime folder unzipped earlier. This folder should contain a direct subfolder called **config**.

7. Choose the **Modules** tab of the server properties editor and make the following changes:
   - Choose **Add Web Module** and add PlanetsDemo-service.
   - Choose **Add External Web Module**. The **Add Web Module** dialog window appears. Make the following changes:
     - In **Document base**, choose **Browse** and navigate to ~/webapps/gapwalk-application...war
     - In **Path**, enter /gapwalk-application.
   - Choose **OK**.
   - Choose **Add External Web Module** again and make the following changes:
     - For **Document base**, enter the path to the frontend .war (in PlanetsDemo-web/target)
- For **Path**, enter `/demo`
- Choose OK
- Save the editor modifications (Ctrl + S).

The editor content should now be similar to the following.

### Step 8: Create the JICS database

In this step, you connect to the database that you created in *Step 1: Create a database (p. 169).*

1. From the AppStream 2.0 instance, issue the following command in a terminal to launch pgAdmin:

   ```
   ../pgadmin-start.sh
   ```

2. Choose an email address and password as login identifiers. Take note of the provided URL (typically http://127.0.0.1:5050). Launch Google Chrome in the instance, copy and paste the URL into the browser, and log in with your identifiers.
3. After you log in, choose **Add New Server** and enter the connection information to the previously created database as follows.
4. When you connect to the database server, use **Object > Create > Database** and create a new database named **jics**.

5. Edit the database connection information that the demo app used. This information is defined in `PlanetsDemo-runtime/config/application-main.yml`. Search for the `jicsDs` entry. To retrieve the values for **username** and **password**, in the Amazon RDS console, navigate to the database. On the **Configuration** tab, under **Master Credentials ARN**, choose **Manage in Secrets Manager**. Then, in the Secrets Manager console, in the secret, choose **Retrieve secret value**.

**Step 9: Start and test the application**

In this step, you start the Tomcat server and the demo application so that you can test it.

1. To start the Tomcat server and the previously deployed applications, select the server entry in the **Servers** view and choose **Start**. A console appears that displays startup logs.

2. Check the server status in the **Servers** view, or wait for the **Server startup in [xxx] milliseconds** message in the console. After the server starts, check that gapwalk-application is properly deployed. To do this, access the **http://localhost:8080/gapwalk-application** URL in a Google Chrome browser. You should see the following.
3. Access the deployed application frontend from Google Chrome at http://localhost:8080/demo. The following **Transaction Launcher** page should appear.

4. To start the application transaction, enter PINQ in the input field, and choose **Run** (or press Enter). The demo app screen should appear.

5. Type a planet name in the corresponding field and press Enter.
Step 10: Debug the application

In this step, you test using the standard Eclipse debugging features. These features are available when you work on a modernized application.

1. To open the main service class, press Ctrl + Shift + T. Then enter PlanetsinqProcessImpl.

2. Navigate to the searchPlanet method, and put a breakpoint there.
3. Select the server name and select Restart in Debug.
4. Repeat the previous steps. That is, access the application, input a planet name, and press Enter.

   Eclipse will stop the application in the searchPlanet method. Now you can examine it.
Clean up resources

If you no longer need the resources that you created for this tutorial, delete them so that you don't incur additional charges. Complete the following steps:

- If the Planets application is still running, stop it.
- Delete the database that you created in Step 1: Create a database (p. 169). For more information, see Deleting a DB instance.

Set up a build and CI/CD pipeline

AWS Mainframe Modernization provides you with the ability to set up builds and continuous integration/continuous delivery (CI/CD) pipelines for your migrated applications. These builds and pipelines use AWS CodeBuild, AWS CodeCommit, and AWS CodePipeline to provide these capabilities. CodeBuild is a fully managed build service that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. CodeCommit is a version control service that enables you to privately store and manage Git repositories in the AWS Cloud. CodePipeline is a continuous delivery service that enables you to model, visualize, and automate the steps required to release your software.

This guide explains how to set up a build and deployment pipeline for the Planets demo application on an Amazon Elastic Compute Cloud instance. It uses AWS CodeArtifact, AWS CodeBuild, AWS CodeCommit, and AWS CodePipeline to store the project dependencies and source code and set up a build project and a pipeline. When you finish this tutorial, you'll have a working pipeline that can trigger an automatic build and deploy whenever you make changes to the source code.

Note
This tutorial requires you to create resources in other AWS services. These resources might result in charges to your AWS account.

Topics
- Prerequisites (p. 181)
- Overview of tutorial (p. 182)
- Set up your CI/CD environment (p. 182)
- Set up your CI/CD build and pipeline (p. 189)
- Clean up resources (p. 195)

Prerequisites

Before you begin, make sure you can access both the AWS Management Console, and an Amazon Elastic Compute Cloud instance that you'll use to set up the build and the pipeline. If you don't have an Amazon EC2 instance available, create one and connect to it. Then download, install, and configure the following software. For best results, we recommend that you install and configure each application as if you are doing it for the first time.

- Download PlanetsDemo-pom.zip and unzip it to a folder of your choice. This file contains the PlanetsDemo application binaries.
- Download the gapwalk distribution and unzip it to a folder of your choice. This file contains the framework binaries.
- Download build-pipeline.zip (US) or build-pipeline.zip (Europe) and unzip it to a folder of your choice. This archive contains configuration files and scripts.
- Download and install Maven. Maven is a build automation tool.
• If necessary, download and install a Java Development Kit (JDK) (version 8 or 11). The JDK is a collection of software tools and libraries for Java applications.
• Download and install Git. Git is a distributed version control system.

Overview of tutorial

This tutorial contains two sections:

• Set up CI/CD environment tasks. Most of these are one-time only tasks. Many of them require you to create resources in other AWS services. This tutorial includes all the steps to create these resources.
• Create build and pipeline tasks. These are tasks that you typically complete at least once for every application or code project.

Note
If you already have a CodeArtifact repo with all the dependencies, you can skip the Maven and the CodeArtifact steps.

Set up your CI/CD environment

The steps in this section help you set up a CI/CD pipeline environment for the PlanetsDemo sample application. We recommend that you complete each of these steps as though you were doing it for the first time. Even if you have existing resources, such as a CodeArtifact repository, create another one for this tutorial. You can delete it when you finish the tutorial. These steps assume that you are connected to an available Amazon EC2 instance and that you can access the AWS Management Console at any time.

Topics
• Step 1: Configure Git (p. 182)
• Step 2: Install or upgrade and configure the AWS CLI (p. 183)
• Step 3: Create a CodeArtifact domain and repository (p. 183)
• Step 4: Generate HTTPS Git credentials for CodeCommit (p. 184)
• Step 5: Create an CodeCommit repository (p. 185)
• Step 6: Connect to the repository and clone it (p. 186)
• Step 7: Update pom.xml (p. 187)
• Step 8: Push the code to your CodeCommit repo (p. 187)
• Step 9: Set CodeArtifact environment variables (p. 188)
• Step 10: Get a CodeArtifact token (p. 188)

Note
This tutorial can be completed on both Windows and Linux based Amazon EC2 instances. If you are using Windows we recommend you use the Bash command prompt that is installed by default with Git. The Bash prompt is available under the Git group on the Windows Start menu. If you kept the default options when installing Git, the Bash prompt is also available from the context menu of File Explorer.

Step 1: Configure Git

In this step, you configure Git with your username and email address to identify yourself to Git. This is a one-time task.

1. Run the following command:
Step 2: Install or upgrade and configure the AWS CLI

In this step, you install the AWS Command Line Interface (AWS CLI) on your Amazon EC2 instance so that you can call CodeArtifact commands. This is a one-time task. If you have an older version of the AWS CLI installed, you must upgrade it so that the CodeArtifact commands are available. CodeArtifact commands are available in the following AWS CLI versions:

- **AWS CLI 1**: 1.18.77 and newer
- **AWS CLI 2**: 2.0.21 and newer

To check the version, use the `aws --version` command.

**To install and configure the AWS CLI**

1. Install or upgrade the AWS CLI. Follow the instructions in the [Installing the AWS Command Line Interface](#) guide.
2. Configure the AWS CLI with the following command: `aws configure`

   When you are prompted, specify your AWS access key and AWS secret access key. When you are prompted for the default Region name, specify the Region where you will create the build and pipeline, such as `us-east-1`. When you are prompted for the default output format, specify `json`.

   **Important**
   When you configure the AWS CLI, you are prompted to specify an AWS Region. Choose one of the supported Regions listed in the [AWS Mainframe Modernization endpoints and quotas](#) in the Amazon Web Services General Reference. All the resources that you create must be in the same AWS account and AWS Region.

   For more information, see the [Configuring the AWS CLI](#) in the AWS Command Line Interface User Guide and the [Managing access keys for IAM users](#) in the IAM User Guide.
3. To verify the installation or upgrade, call the following command: `aws m2 help`.

   If the installation is successful, it displays a list of available AWS Mainframe Modernization commands.

Step 3: Create a CodeArtifact domain and repository

In this step, you create a CodeArtifact repository to contain dependencies that are required by the PlanetsDemo sample application. In CodeArtifact, repositories are part of domains, so you must also create a domain for your repository. When you create the repository, you must specify the public Maven upstream connection. This connection gives you access to the most recent Maven dependencies. This is a one-time step. You complete these steps in the CodeArtifact console. For more information, see the [Getting started with CodeArtifact](#) in the CodeArtifact User Guide.

1. Open the CodeArtifact console.
2. In the Region selector, choose the AWS Region where you want to create the domain and repository.
3. Choose **Create repository** and provide the following information.

   
   b. (Optional) In **Repository description**, enter a description for your repository so that you can identify it as different from any other CodeArtifact repositories.
   
   c. In **Public upstream repositories**, choose `maven-central-store` to create a repository connected to maven that is upstream from your `my-bluage-dependency-repo`.

   ![Create repository](image)

   d. Choose **Next**.

4. In **Domain**, choose **This AWS account**.

5. In **Domain name** enter `my-bluage-domain` and choose **Next**.

6. In **Review and create**, review what CodeArtifact is creating for you. When you’re ready, choose **Create repository**.

### Step 4: Generate HTTPS Git credentials for CodeCommit

In this step, you download specific credentials to configure access to CodeCommit. You’ll use these credentials when you connect to the CodeCommit repository from your Amazon EC2 instance.

1. Sign in to the AWS Management Console and open the IAM console at [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/).

2. In the IAM console navigation pane, choose **Users**, and then choose the user you want to configure for CodeCommit access.

3. On the user details page, choose the **Security Credentials** tab, and in **HTTPS Git credentials for AWS CodeCommit**, choose **Generate credentials**.

   ![HTTPS Git credentials for AWS CodeCommit](image)

4. Copy the credentials that IAM generated for you. You can show, copy, and paste this information into a secure file on your local computer, or you can choose **Download credentials** to download this information as a .CSV file. You need this information to connect to CodeCommit.
Step 5: Create an CodeCommit repository

In this step, you create an empty repository in CodeCommit. This repository will hold the source code for the PlanetsDemo sample application, that you will upload in a later step.


2. In the Region selector, choose the AWS Region where you want to create the repository. Make sure it is the same Region as the one where you created the CodeArtifact domain and repository.

3. On the Repositories page, choose Create repository.

4. On the Create repository page, in Repository name, enter my-planetsdemo-repo or another name, if that one is unavailable.

   **Note**

   Repository names are case sensitive. The name must be unique in the AWS Region for your AWS account.

5. (Optional) In Description, enter a description for the repository. This can help you and other users identify the purpose of the repository.
Step 6: Connect to the repository and clone it

In this step, you copy the HTTPS URL for the repository and use it to clone to your Amazon EC2 instance. Although the repository is currently empty, cloning it to your instance now makes it possible for you to upload all the files for the PlanetsDemo sample in bulk, rather than uploading them one by one through the CodeCommit console.

1. On the details page for the repository, choose Clone URL and then choose Clone HTTPS.

2. Connect to your Amazon EC2 instance and navigate to a directory where you want the repository to be cloned.

3. Enter `git clone`. Then paste in the HTTPS URL that you copied. This command will create a directory named after the repository (e.g. my-planetsdemo-repo), and create the directories that Git uses to manage the clone.

4. When prompted, enter the HTTPS Git credentials that you generated previously.

You can disregard the warning message about cloning an empty repository. You’ll add code to the repository in a later step.

6. Choose Create.
**Step 7: Update pom.xml**

In this step, you update the `gapwalk.version` property in the `pom.xml` file of the application to match the gapwalk distribution that you downloaded earlier. This step is required for the application to run.

1. On your Amazon EC2 instance, locate and open `pom.xml` with the text editor of your choice. The file is in the extracted `PlanetsDemo-pom` folder.
2. Find the `<properties>` section of the file. It looks something like the following:

   ```xml
   <properties>
     <spring.boot.version>2.5.12</spring.boot.version>
     <gapwalk.version>3.1.0-SNAPSHOT</gapwalk.version>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   ```

3. Change `gapwalk.version` to match the current version `3.1.0-b3257-SNAPSHOT`.

   **Note**
   
   The value that you use for the `gapwalk.version` property must point to an existing gapwalk distribution version.

**Step 8: Push the code to your CodeCommit repo**

In this step, you push the sample source code for PlanetsDemo to the CodeCommit repository that you created previously.

1. On your Amazon EC2 instance, copy the `PlanetsDemoRepo` code to your cloned CodeCommit repository.
2. In a Bash command prompt, change to the directory (cd) where you copied the code. Add new files, commit the change, and push the code to the CodeCommit repository using the following commands:

   ```bash
   git add -A
   git commit -m "Initial source load"
   git push
   ```
Step 9: Set CodeArtifact environment variables

In this step, you export variables for your CodeArtifact domain and repository. You need these variables in a later step when you get a CodeArtifact token that you need for authorization.

- On your Amazon EC2 instance, run the following commands at Bash prompt:

```
export CODEARTIFACT_DOMAIN=my-bluage-domain
export CODEARTIFACT_REPO=my-bluage-dependency-repo
```

Step 10: Get a CodeArtifact token

In this step, you get a CodeArtifact token. CodeArtifact uses this token to authenticate and authorize requests from Maven.

1. On your Amazon EC2 instance, run the following command in the same Bash prompt:

```
export CODEARTIFACT_TOKEN=`aws codeartifact get-authorization-token --domain $CODEARTIFACT_DOMAIN --query authorizationToken --output text`
```

2. Run the following command to verify the CodeArtifact token:

```
echo $CODEARTIFACT_TOKEN
```

**Note**

When the token expires, you must refresh it with the previous command. Tokens are valid for a default of 12 hours.
Set up your CI/CD build and pipeline

The steps in this section help you set up the PlanetsDemo sample application build and a pipeline for the build. These are steps that you will complete each time that you want to set up a CI/CD pipeline for one of your migrated applications. We recommend that you complete each of these steps as though you were doing it for the first time. Even if you ran the scripts previously, use different names or targets so that you can run them again and not overwrite anything. These steps assume that you are connected to an available Amazon EC2 instance and that you can access the AWS Management Console at any time.

Topics

- **Step 1: Generate a Maven configuration file** (p. 189)
- **Step 2: Copy the Maven configuration file to the gapwalk directory** (p. 190)
- **Step 3: Deploy AWS Blu Age dependencies into the CodeArtifact repository** (p. 190)
- **Step 4: Edit the build parameters file** (p. 190)
- **Step 5: Create and deploy the CodeBuild project** (p. 191)
- **Step 6: Start a build** (p. 191)
- **Step 7: View build information** (p. 192)
- **Step 9: Specify the default branch of the source code repository** (p. 193)
- **Step 10: Edit the pipeline parameters file** (p. 193)
- **Step 11: Create and deploy the CodePipeline pipeline** (p. 194)
- **Step 12: View the pipeline** (p. 194)

**Step 1: Generate a Maven configuration file**

In this step, you run the provided `generate-maven-config.sh` script to generate a `settings.xml` file for Maven. The script is provided in the `build-pipeline.zip` archive, that you downloaded and extracted previously.

1. Locate the `generate-maven-config.sh` script.
2. (Optional) If you use named profiles with the AWS CLI, edit the script and append the `--profile your-profile-name` option to ensure that the script can access the correct environment and generate a valid configuration file.
3. Run the script with the command `./generate-maven-config.sh`
4. Open the `settings.xml` file that the script generates to see how the variables you exported are used.

```
<settings>
  <servers>
    <server>
      <id>codeartifact</id>
      <username>aws</username>
      <password>${env.CODEARTIFACT_TOKEN}</password>
    </server>
  </servers>
  <mirrors>
    <mirror>
      <id>codeartifact</id>
      <name>codeartifact</name>
    </mirror>
  </mirrors>
</settings>
```

189
Step 2: Copy the Maven configuration file to the gapwalk directory

In this step, you make sure that the Maven configuration file that you created previously occupies the same directory as the gapwalk JAR files.

1. Locate the `settings.xml` file that the `generate-maven-config.sh` script created in the previous step.
2. Copy the file to the same directory as the gapwalk JAR files.

```
cp settings.xml directory-with-extracted-assets/
```

Step 3: Deploy AWS Blu Age dependencies into the CodeArtifact repository

In this step, you deploy the AWS Blu Age dependencies into CodeArtifact so that they are available for the build that you will set up later.

1. Locate the `deploy-to-codeartifact.sh` script.
2. Run the following command: `.deploy-to-codeartifact.sh`

Step 4: Edit the build parameters file

In this tutorial, you create a CodeBuild project with AWS CloudFormation. This service creates AWS resources for you based on information that you provide. We’ve provided an AWS CloudFormation template that you can run from the command line on your Amazon EC2 instance. That template creates a CodeBuild project. In this step, you’ll edit a JSON file that contains some parameters required by the AWS CloudFormation template.

1. On your Amazon EC2 instance, locate the `build-project-params.json` file. It looks like the following:

```json
{
```
Set up your CI/CD build and pipeline

2. Edit the file as follows:

   - For **CodeRepositoryName**, enter the name of the CodeCommit repository that you created for this tutorial, for example, `PlanetsDemoRepo`.
   - For **ArtifactRepositoryName**, enter the name of the CodeArtifact repository that you created for this tutorial, for example, `my-bluage-dependency-repo`.
   - For **ArtifactRepositoryDomain**, enter the name of the CodeArtifact domain where you located the CodeArtifact repository that you created, for example, `my-bluage-domain`.

   Your updated file should look something like this:

   ```json
   {     "Parameters": {       "CodeRepositoryName": "PlanetsDemoRepo",       "ArtifactRepositoryName": "my-bluage-dependency-repo",       "ArtifactRepositoryDomain": "my-bluage-domain"     }   }
   ```

Step 5: Create and deploy the CodeBuild project

In this step, you use AWS CloudFormation to create and deploy a CodeBuild project. That project uses the parameters that you supplied in the previous step to identify the location of the source code and the project dependencies. You call AWS CloudFormation from your Amazon EC2 instance with the AWS CLI.

   - Run the following command:

     ```bash
     aws cloudformation deploy --stack-name your-stack-name --template-file build-project.yaml --capabilities CAPABILITY_IAM --parameter-overrides file://build-project-params.json
     ```

     It takes a few minutes for AWS CloudFormation to finish creating the resources. You can watch the progress in the AWS CloudFormation console.

Step 6: Start a build

In this step, you use CodeBuild to run a build with the build project that AWS CloudFormation created for you in the previous step.

2. In the navigation pane, choose **Build projects**.
3. In the list of build projects, choose the project that starts with M2Build, and then choose **Start**
    **build**. The build starts immediately.

Wait for the build to finish.

**Step 7: View build information**

In this step, you verify the build status and output artifacts by viewing the build information.

1. In the CodeBuild console, if you’re not already viewing the details of the build that you started in the
   previous step, choose **Build history**. Then choose the **Build run** link for the build project that starts
   with M2Build.

2. On the **Build Status** page, in **Phase details**, make sure that each of the build phases shows
   **Succeeded** in the **Status** column:

   ![Build Phases Succeeded](image)

3. With the CodeBuild console still open and the build details page still displayed from the previous
   step, choose **Build details** and scroll down to the **Artifacts** section.
The link to the Amazon S3 folder that contains the build output artifact appears after **Artifacts upload location.** This link opens the folder in Amazon S3 where you find the build output artifact file.

<table>
<thead>
<tr>
<th>Artifacts</th>
<th>Artifacts upload location</th>
<th>Disable artifact encryption</th>
<th>Override artifact name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>am:aws:ss-m2-build:98365870</td>
<td>False</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>us-east-1/build</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>M2Build-zdbMen7zd056</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Step 9: Specify the default branch of the source code repository

In this step, you edit the file that will create the CodePipeline pipeline for you with AWS CloudFormation. You must customize the default branch of the code repository specified in the file.

1. On your Amazon EC2 instance, edit the build-pipeline.yaml file.
2. Locate the CodeRepositoryBranch parameter.

   ```
   Parameters:
   CodeRepositoryBranch:
     Type: String
     Default: main
   ```

3. Change `main` to `master`.

### Step 10: Edit the pipeline parameters file

Similar to how you created the CodeBuild project in a previous step, you'll create the CodePipeline pipeline with AWS CloudFormation. We've provided an AWS CloudFormation template that you'll run from the command line on your Amazon EC2 instance. That template creates a CodePipeline pipeline. In this step, you'll edit a JSON file that contains some parameters required by the AWS CloudFormation template.

1. On your Amazon EC2 instance, locate the build-pipeline-params.json file. It looks like the following listing:

   ```
   {
     "Parameters": {
       "CodeRepositoryName": "",
       "ArtifactRepositoryName": "",
       "ArtifactRepositoryDomain": ""  
     }
   }
   ```

2. Edit the file as follows:
   - For `CodeRepositoryName`, enter the name of the CodeCommit repository that you created for this tutorial, for example, PlanetsDemoRepo.
   - For `ArtifactRepositoryName`, enter the name of the CodeArtifact repository that you created for this tutorial, for example, my-blugue-dependency-repo.
Set up your CI/CD build and pipeline

- For ArtifactRepositoryDomain, enter the name of the CodeArtifact domain where you located the CodeArtifact repository that you created, for example, my-bluage-domain.

Your updated file should look something like this:

```json
{
    "Parameters": {
        "CodeRepositoryName": "PlanetsDemoRepo",
        "ArtifactRepositoryName": "my-bluage-dependency-repo",
        "ArtifactRepositoryDomain": "my-bluage-domain"
    }
}
```

Step 11: Create and deploy the CodePipeline pipeline

In this step, you use AWS CloudFormation to create and deploy a CodePipeline pipeline. That pipeline will use the parameters that you supplied in the previous step to identify the location of the source code and the project dependencies. You call AWS CloudFormation from your Amazon EC2 instance with the AWS CLI.

- Run the following command:

```
aws cloudformation deploy --stack-name your-stack-name --template-file build-pipeline.yaml --capabilities CAPABILITY_IAM --parameter-overrides file://build-pipeline-params.json
```

It takes a few minutes for AWS CloudFormation to finish creating the pipeline. You can watch the progress in the AWS CloudFormation console.

Step 12: View the pipeline

In this step, you view the pipeline that AWS CloudFormation created for you to make sure that it is working. This pipeline downloads the source code from the CodeCommit repository. It then builds the project, including installing all dependencies to your CodeArtifact repository, and deploys the project to an Amazon S3 bucket.

2. Under Name choose the pipeline that was created for you.
3. On the details page, review the Source, Build, and DeployStaging stages.
This pipeline starts automatically when you make changes to the source code and then push the changes to your CodeCommit repository.

**Clean up resources**

If you no longer need the resources you created for this tutorial, delete them to avoid additional charges. To do so, complete the following steps:

- Delete the CodePipeline pipeline. For more information, see [Delete a pipeline in CodePipeline](#) in the [AWS CodePipeline User Guide](#).
- Delete the CodeBuild project. For more information, see [Delete a build project in CodeBuild](#) in the [AWS CodeBuild User Guide](#).
- Delete the CodeCommit repository. For more information, see [Delete an CodeCommit repository](#) in the [AWS CodeCommit User Guide](#).
- Delete the CodeArtifact repository. For more information, see [Delete a repository](#) in the [CodeArtifact User Guide](#).
AWS Blu Age Release Notes

This section contains the release notes of AWS Blu Age Runtime and Modernization Tools from version 3.5.0 onward, most recent first, organized by version number.

Note
For release notes predating this document, please contact AWS Blu Age delivery services.

Topics
- AWS Blu Age Release Notes 3.8.0 (p. 196)
- AWS Blu Age - Runtime Release 3.8.0 (p. 196)
- AWS Blu Age - Modernization tools Release 3.8.0 (p. 198)
- AWS Blu Age Release Notes 3.7.0 (p. 200)
- AWS Blu Age - Runtime Release 3.7.0 (p. 200)
- AWS Blu Age - Modernization tools Release 3.7.0 (p. 202)
- AWS Blu Age Release Notes 3.6.0 (p. 203)
- AWS Blu Age - Runtime Release 3.6.0 (p. 204)
- AWS Blu Age - Modernization tools Release 3.6.0 (p. 206)
- AWS Blu Age Release Notes 3.5.0 (p. 208)
- AWS Blu Age - Runtime Release 3.5.0 (p. 208)
- AWS Blu Age - Modernization tools Release 3.5.0 (p. 210)

AWS Blu Age Release Notes 3.8.0

This release of AWS Blu Age Runtime and Modernization Tools is focused on multiple transversal enhancements across the product to improve its quality and security, along with improvements in performance for caching and the unification of commands supports in a single distribution. Some key features and changes in this release are:

- Version upgrade from Spring 2.5 to Spring 2.7, increasing the maintenance support, performance, and security of the platform.
- Unification of more than 82 CL commands support as part of the over-the-counter distribution in order to facilitate the usage and deployment of modernized applications previously using CL scripting.
- New APIs available to operate and interact better with BluSAM datasets, such as integrated import to the managed service and the capability to list dataset metadata information.
- Performance improvements and extension of the usage of Redis, including availability in cluster mode, high-availability data retrieval, standardization of the usage of secrets.

For more information about the changes included in this release, see the following sections.

AWS Blu Age - Runtime Release 3.8.0

zOS

New features
• Handling key definition as a string for DynamicFileBuilder
• DFSORT: Added support for multi-items in OUTFIL TRAILER1 + DFSORT grammar initialization
• CommonDDUtils tool: handling record size in in-stream data
• Indexed file: handling the GENKEY option

Improvements
• Externalized BluSAM loading services in a separate jar
• Added support to setup location for storing temporary files
• Improved shared cache mechanisms for multi-nodes cases
• Shared cache usage: IDCAMS verify optimization
• Improve ROWID injection for embedded select
• JCL: Each in-stream job procedure is now generated in a distinct groovy file
• Ensure card-demo-v2 coverage on IDCAMS JCL cards
• BluSAM: Avoid duplicate warmUp when using multiple instances
• Reduced memory footprint on cache hydration
• Jedis pool config support
• Added line separator to stream if used in file concatenation
• Support for EBCDIC cards + block comments (/ * ... / ) in IDCAMS utility
• Database support query: support for double byte strings in the conversion of level49 towards SQL
• DFSORT grammar: implements 17 control statements + integration of 2 of them (OMIT/INCLUDE)
• Enhance GRAPHIC columns fetch INFUTILB
• Support for reading file with variable Size table
• Support for ZonedType with nibble signed where the first bit of last byte is 'E'
• DFSORT/ICETOOL adds support for NOMATCH=(..) argument if a record does not match any of the
CHANGE find constants
• Redis Cluster compatibility
• Handling Job Status (Failed) based on groovy exit code
• Improved CICS SYNCPOINT ROLLBACK support.
• Pre-fetch window to optimize redis cache usage
• JCL/GROOVY: Inherits isRDW property from previous step's dataset when DISP=(,PASS)
• Handling partial copy of data with variable size array

AS400

New features
• Support for I/O cards for display files
• Support for additional message information for DSPF keywords ERRMSGISD and CHKMSGISD
• Support for multiple error messages on frontend screen
• Added or improved support of 82 CL commands within the gapwalk-cl-command application

Improvements
• Improved support for DELETE and READ under commitment control
• ConvertDate inside of builtin %dec
• Enforced XSS security headers
• Improved robustness and consistency of STM generation (better handling of: continuation line in free form rpg, commas for decimal part, free form blocks in definition/declaration)
• Improved DataHolderMapper generation
• Added robustness and change scope in DataAreaFactory
• Improved the focus shifting on tab key
• Improved performance on Jasper report generation
• Improved decimal display with padding 0s
• Improved support for ROW/COL field in INFDS
• Improve support for modified fields from the screen
• Added getters for generated report name and path
• Improved on Dataqueue length
• Improved autoconfiguration of Job Queues to match new standards in Spring Boot 2.7
• Improved workstation updates for multiple concurrent sessions

Transversal capabilities

New features
• Support for No Invalid Data Tolerance for Packed
• Added pagination/filtering to list dataset endpoints

Improvements
• Enhanced ORACLE query transformation strategy in column comparison against empty string
• Handling BLOB DB2 with DSNTEP and INFUTILB utility programs. BLOB DB2 are now modernized to BYTEA type postgres.
• Improvement of deletion of last item of cursor
• Enhanced support for delete RRDS file
• Improved AWS BlueSAM secret performance
• Improved handling of database connections in SQL framework
• Standardized AWS multi-datasource seceret manager keys
• Performance regression fixes
• Improved check function for PackedType
• Improved handling of LOW-VALUE for PackedType
• Upgraded spring security packaging for cognito connection
• Not applying codeshiftpoint encoding and decoding on DB2 targeted databases

Third party
• Spring Boot upgrade from 2.5 to 2.7

AWS Blu Age - Modernization tools Release 3.8.0

zOS

New features
• JCL: Handling stream with carriage return "\r"
Improvements

- Improved logging to prevent division by zero when modernizing a DIVIDE with ON SIZE ERROR clause
- JCL: Enhanced support for calling a procedure in a procedure
- Support for OF keyword in FORMATTIME CICS command when there are ambiguous fields
- JCL: support for Å¥ character in variables
- JCL: computing RC based on previous steps
- Comparing bytes instead of strings when PL1 SUBSTR is used
- Improvement of initialization of multidimensional arrays from single source
- Improved parsing of COBOL when it involves a single SQL query in an IF block

AS400

New features

- Support for nested IF statement in CL
- Improved support for ENDDO statement in RPG freeform

Improvements

- Improved support for conditioning Control Level
- Improved prototype return with LIKE
- Improved support for handling functions %months, %year, %days
- Support for help feature for the whole screen
- Handling figurative BLANKS passed as a parameter
- Improvement on expression EVAL with "" operator
- Handling START command without KEY PHASE
- Improvement on handling the Keyword LIKEREC.
- Improvement on unnamed subfields
- Improvement on procedure returning an unsigned type
- Improved support for RESET operation (Free RPG), %CHAR and %DEC builtins.
- Improvement in the builtin function %LOOKUPXX.
- Improved support for LIKEDS keyword on procedure without prototype
- Handling Dim keyword array type (VAR, AUTO)
- Improved support for XFOOT
- COBOL: improved support for RENAMES fields
- CL: support while(true) condition
- Improved the handling of standalone arrays with LIKE keyword
- Improvement of built-in function %INT
- Improved RPG Full Free parsing
- Improved support for array in the linkage
- CL2GROOVY: Support Select Statement
- Improvement in DSPF keyword "ERRMSGID"
- Improved the handling of initializing bytes with leading zeroes
- Improvement on authorizedValues for numerical fields
- Handling extender H for Free form EVAL statement
- CL to Groovy: Support substring of LDA
- Improved support for RESET on a record
- Improved the handling of EDTCDE and EDTWRD with references
- Improved input-field mapping with DDS fields
- Improved support for MOVEA character into IN array
- Improvement in prototype with LIKEDS keyword
- Improved support for the DSPF keyword DSPATR
- Improved parsing of D-card with +/-
- Added robustness in program calls
- Added robustness in the field-resolving process

Transversal capabilities

Improvements

- FrontEnd: Simulate paste event for IME input

Third party

- Spring Boot upgrade from 2.5 to 2.7

AWS Blu Age Release Notes 3.7.0

This release of AWS Blu Age Runtime and Modernization Tools mainly includes enhancements to better support commands and utilities, capabilities to integrate with AWS Secrets Manager and new monitoring features. Some of the key changes in this release are:

- Multiple runtime components can now use AWS Secrets Manager to increase the security setup of modernized applications, mostly related to utilities data sources, Redis for TS Queues, BluSam cache and locks.
- Monitoring endpoint that allows to retrieve transaction, batch, and JVM metrics for resource usage optimization and operational management, such as status, duration, volume, etc.
- New features to support IBM MQ calls in RPG, and increased JCL SORT and IDCAMS transformation coverage.

For more information about the changes included in this release, see the following sections.

AWS Blu Age - Runtime Release 3.7.0

Topics

- zOS (p. 196)
- AS400 (p. 197)
- Transversal capabilities (p. 198)

zOS

New features

- Improve parsing queries involved in program utility application by using SQL like grammar. (V7-9401)
• Handle indexed Variable Size Array when offset (V7-9904)
• Support INSERT SQL TIME column into DB2 with 24:00:00 hour format (V7-10023)
• Support INSERT SQL query from arrays with FOR ROWS and ATOMIC options (V7-10105)
• JCL SORT - enhance TranscodeTool to support OUTREC with IFTHEN (V7-10124)
• JCL SORT - add support for DATE keyword in OUTREC command (V7-10125)
• JCL - add support of In-Stream procedures (V7-10223)

**Improvements**

• A dataset marked with the "PASS" disposition should be available across all job steps (V7-9504)
• Support JCL attribute SCHENV (V7-9570)
• Support SEND with CTLCHAR option (V7-9714)
• COBOL - Handle different line separator charsets in ACCEPT statements (V7-9875)
• Avoid multiple rollback (V7-9958)
• Allow use of MOD disposition to append at the end of GDG files (V7-10031)
• Optimization: putAll refactoring (V7-10063)
• PutAll refactoring: adding pagination (V7-10063)
• Make Jedis client read timeout configurable (V7-10063)
• UseSel support for standalone mode (V7-10114)
• Support EIBDS after opening file successfully (V7-10147)
• Support EIBDS after a file control request (V7-10147)
• Improve CICS SYNCPOINT support (V7-10187)
• BluesamRedisSerializer: issue with metadataPersistence (V7-10202)
• Support Redis AWS Secrets Manager for TS queues (V7-10204)
• Support JCLBCICS on customizing DD name size (V7-10224)
• Adds support for absolute path in IDCAMS DELETE statement (V7-10308)

**AS400**

**New features**

• Implementation of the help feature for AS400 screens (V7-9673)

**Improvements**

• Number of records in INFDS (V7-9377)

**Transversal capabilities**

**New features**

• Support for Runtime on EC2 to send logs to Amazon CloudWatch (D87990246)
• Added new endpoint to retrieve metrics about batches, transactions, and JVM (D88393832)

**Improvements**

• Support datasources AWS Secrets Manager for utility pgm (V7-9570)
• Added Db2 support for DSNUTILB DISCARD (V7-9798)
• Support for writing into logger instead of default system output stream in default SYSPRINT and SYSPUNCH files (V7-10098)
• Support BluSam redis cache and locks connection properties in AWS Secrets Manager (V7-10238)
• Support for SSL connection on Db2 XA AWS secret (V7-10258)
• Updated metadata for IDCAMS REPRO and VERIFY (V7-10281)
• Improved IDCAMS Abend Return Code Management (V7-10307)

AWS Blu Age - Modernization tools Release 3.7.0

Topics
• zOS (p. 198)
• AS400 (p. 199)
• Transversal capabilities (p. 200)

zOS

New features
• PLI - Improved assignment for array cross section and two-dimensional arrays (V7-9830)

AS400

New features
• Handling of control level indicators (V7-9227)
• Support for EXTNAME parameter *INPUT (V7-9897)
• Enhanced Goto Rewriting: Support for tags located in SELECT OTHER statements (V7-9973)
• Support REFSHIT DSPF keyword (V7-10049)

Improvements
• Improvement on handling file description keyword EXTIND(*INUx) (V7-7404)
• Improved SQLDDS file transformation (V7-7687)
• File objects no longer generated for AS400 files (V7-9062)
• Improved handling of file description keyword EXTDESC (V7-9268)
• Improved handling of %CHAR builtin (V7-9311)
• Improved support for pagedown on last record without SFLEND (V7-9322)
• Improved support for prefixed data structures (V7-9436)
• Support for dimension defined with %SIZE (V7-9472)
• Support for handling PF field name declared within double quotes (V7-9557)
• Improved file operation - case insensitive (V7-9785)
• Support for field initialized to *USER (V7-9806)
• Support for COMP type in AS400 (V7-9840)
• Improved COBOL400 parsing on (Not)InvalidKey (V7-9922)
• Improved handling of SCAN operation (V7-9971)
• Improved support of GOTO opcode (V7-9973)
• Improved handling of EXCEPT operation (V7-9977)
- Improved prefix support (V7-10000)
- Support for MQ calls in RPG (V7-10007)
- Improved `%LOOKUP builtin (keyed array data structure) (V7-10022)
- Support for Close *All operation (V7-10036)
- Support for UPDATE AS ROW CHANGE SQLDDS statement (V7-10051)
- Improvement to handle literal value type Long (V7-10073)
- Improved RPG grammar (the use of the keyword INZ as name of subroutine) (V7-10074)
- Improved RPG grammar to support numeric values with empty fractional part (V7-10077)
- Improved support for fields shared between CL and external file (V7-10081)
- Improved support for DDS conditional indicators (V7-10084)
- Support for DDS binary type with COBOL programs (V7-10100)
- Improved name collision with linkage (V7-10109)
- Support for mixing main and export procedures (V7-10112)
- Improved support for DataStructure in a sub-procedure (V7-10113)
- Improved support of CLEAR (V7-10126)
- Improved support of DO loop (V7-10134)
- Support SQLTYPE in Full-Free RPG (V7-10151)
- Improved parsing of conditions on DDS keyword (V7-10155)
- Improved DSL generation (V7-10163)
- Improvement for processIndicators when the condition is a binary expression. (V7-10164)
- Improved GOTOs with Else condition (V7-10168)
- Support for type Time and Timestamp in DSPF (V7-10173)
- Improved parsing of continuation line for DDS (V7-10183)
- COBOL support for RENAMES FLD OF RECORD (V7-10195)
- Improved conditional indicator parsing on DSPF fields (V7-10221)
- Support parsing of DDS keyword NOALTSEQ (V7-10288)
- Support Help menu and hidden fields (V7-10314)
- Improved DSPF help keyword sanity check (V7-10328)
- No longer propagating all keywords on Ref field (V7-10347)

Transversal capabilities

New features

- Data Migrator - Handling CLOB data (V7-9665)

Improvements

- Propagating JCL property SCHENV from JOB to PROC GROOVY definition through JobContext (V7-10225)
- FrontEnd - Adjusting window size in case of no border (V7-10358)

AWS Blu Age Release Notes 3.6.0

This release of AWS Blu Age Runtime and Modernization Tools provides new features for both zOS and AS400 legacy migrations, mainly oriented to expanding CICS support mechanisms, complementing JCL.
capabilities, optimizing performance in concurrent and high-volume features, and adding multi-data-source capabilities. Some of the key changes in this release are:

- Enhancement of JCL dynamic file handling, expansion of current statements and management of concatenated datasets, execution of multiple statements in a single block, and data transfer from batches to programs.
- Enhanced support of multiple CICS commands, including inquiry for several CICS resource types.
- The capability to have different databases when using Blu Age Runtime Utilities, best suited for scenarios when business data is distributed across multiple sources.

For more information about the changes included in this release, see the following sections.

AWS Blu Age - Runtime Release 3.6.0

Topics

- zOS (p. 196)
- AS400 (p. 197)
- Transversal capabilities (p. 198)

zOS

New features

- JCL - DynamicFileBuilder - Enhanced file-handle management (V7-9408)
- Enhanced format conversion on some built-in SQL DB2 functions when calling the INFUTILB UNLOAD utility (V7-9554)
- Enhanced PLI multi-dimensional array assignments (V7-9592)
- Handling of sysout redirect to file (V7-9992)

Improvements

- Add triggering of stored procedures for DB2 RDBMS (V7-9155)
- SORT handles conversion to PDF format (V7-9286)
- JCL/GROOVY - Enhance REPRO statement to support DUMMY datasets (V7-9424)
- Improve CICS UNLOCK support (V7-9606)
- Handle default value size for Union (V7-9648)
- JCL/GROOVY handle different termination/disposition in concatenated datasets (V7-9653)
- Make pageSize configurable for blusam datasets (V7-9680)
- DSNUTIL - allow loading of 24:00:00 as valid TIME in DB2LUW (V7-9697)
- Support HIGH-VALUES (0xff) comparison in NumberUtils.ne() / NumberUtils.eq() (V7-9731)
- JCL/GROOVY - support DO ... THEN keywords in IDCAMS IF-THEN-ELSE clauses to execute multiple statements in a single block (V7-9750)
- Invalid JHDB called program outside JHDBBatchRunner (V7-9782)
- Support whitespace characters in SORT OUTFIL control card (V7-9808)
- Improve CICS READ PREV support (V7-9845)
- Improve concurrent access for dataset indexes (V7-9864)
- Improve CICS REWRITE support (V7-9873)
- COBOL - support for multiline SYSIN in ACCEPT statements to pass data from batch (JCL) to a program (COBOL) (V7-9875)
• Groovy - Better handling of ConcatenatedFileConfiguration at files creation step (V7-9876)
• IDCAMS UTILITY - Handling of DEFINE PATH statement (V7-9878)
• SORT BUILD - Adjust TRAN option and handle implicit blanks (V7-9925)
• Improve CICS DELETE with GENERIC option support (V7-9939)
• Improve CICS STARTBR and ENDBR support (V7-9952)
• Improve close performance on concurrent access (V7-9953)
• Improve file status handling on start (V7-9991)
• Groovy - Allow call of getDisposition()/getNormalTermination()/getAbnormalTermination() on ConcatenatedFileConfiguration (V7-10012)

AS400

New features
• Support external indicators on COMMIT keywords (V7-6035)
• Reset ReadC loop after SFLCTL write (V7-8061)
• Support LR indicator in CALL (V7-9250)
• Add new type of dynamic-field (split) to handle input field on multiple lines (V7-9370)
• Support primary/secondary file (V7-9390)
• Local Data Area are now passed to the called job when submitting a job (V7-9775)
• Support of QTEMP for data area and support of datarea value creation. (V7-9916)
• Commitment Control - support for enable/disable commitment control (V7-9956)
• Support external indicators on COMMIT keywords

Improvements
• Improve 0 value display and EDTWRD (V7-8933)
• Support of DSPF keyword "CHKMSGID" (V7-9125)
• SQL commit transaction upon batch termination (V7-9232)
• Improve support of keywords EXPORT and IMPORT for field and datastructure (V7-9265)
• Support lower case in DateHelper (V7-9461)
• Support conversion *CYMD to *ISO (numeric) (V7-9488)
• Improve the handle of built-in %len for a varying field (left-hand and right-hand side of an expression) (V7-9733)
• Improve support for built-in functions "%LOOKUPXX" XX ("LE","LT","GE","GT") (V7-10064)

Transversal capabilities

New features
• CICS - Improve Inquire transaction for option status (V7-9712)
• JCL - Improve Load for sysprint with system out file (V7-9797)
• CICS - Improve INQUIRE TSQUEUE (V7-9823)
• CICS - Improve Inquire terminal for option userid (V7-9906)

Improvements
• Improve the handle of the comparison with blank (V7-8047)
• Improve logging for Jics and BluSam (V7-8847)
• Support BMS extended attributes SOSI and programmed symbol F8 for dynamic fields (V7-8857)
• Handle buffer overflow in program parameter (V7-9138)
• Improve threads write concurrency for Blusam locks registry (V7-9505)
• Support multiple datasources configuration for Utility-pgm (V7-9570)
• Blusam record level locking only mode (V7-9626)
• Ensure metadata persistence resists to server restart (V7-9748)
• Improve DAO clean-up on exception (Browser Close) (V7-9790)
• Support DummyFile for INFUTILB SYSPUNCH (V7-9799)
• Enhance support for negative values on NumericEditedType (V7-9935)

AWS Blu Age - Modernization tools Release 3.6.0

Topics
• zOS (p. 198)
• AS400 (p. 199)
• Transversal capabilities (p. 200)

zOS

New features
• JCL - Enhance logging for end of procedure (V7-8509)
• PL1 - Enhance bags generation for data type PakedLong (V7-8917)
• JCL - Enhance logging for end of procedure when the file contains the "end" marker // (V7-9509)
• PL1 - Enhance support for GET EDIT with Fixed-point and SYSIN stream (V7-9593)
• DB2 - Enhance support for VARGRAPHIC DB2 type (V7-9809)
• CICS - Improve command QUERY SECURITY for option LOGMESSAGE (V7-9969)
• PL1 - Improve bags generation for CHARG/chargraphic built-in (V7-9989)

Improvements
• PL1 - Enhance support for INCLUDEX keyword (V7-9588)
• PL/I - Handle CHARGRAPHIC keyword as a valid parameter of any method call (V7-9589)
• Improving PL1 host variable resolution when named with specific characters @ # $ §. (V7-9654)
• COBOL - Support of C01...C12 & S01...S05 keywords as parameter of WRITE ADVANCING statement at parsing step (V7-9669)

AS400

New features
• Support SQL-DDS transformation in Analyzer (V7-7687)
• Automate SQL-DDS file detection (V7-7687)
• Implementation of SQL-DDS preprocessing (V7-7687)
• Support ALIGN keyword (V7-9254)
• Support ExtName to DSPF and multi-dim array (V7-9663)
• Support InvalidKey statements on COBOL WRITE (V7-9793)

Improvements

• Improvement on TESTB opcode (V7-8865)
• Improve support of DECFMT on focus (V7-8933)
• Handling resulting indicator on MOVE (V7-9224)
• Improve support of keyword TEMPLATE for field and datastructure (V7-9278)
• Improvement of LIKEDS (DS defined using LIKEDS is automatically qualified) (V7-9302)
• COBOL - Improve generation of indicators structure (V7-9423)
• Const parameter in prototype is not read-only (V7-9437)
• Improve EDTCDE keyword with edit code "Y" (V7-9443)
• Support generation of *ROUTINE field in PDS and INFDS (V7-9487)
• Improve rewriting field XXX to standalone (default value is lost while rewriting) (V7-9522)
• Improve Support of DSPF keywords (V7-9658)
• Handling ZEROES default value on binary (V7-9666)
• Support implicit pointer (V7-9719)
• Improve the handling of built-in call %size with one parameter (V7-9730)
• Improve the handling of datastructure references in built-in calls (%ELEM) (V7-9736)
• Improve the handling of signed length for field with LIKE reference in definition specification (V7-9738)
• Improvement on REWRITE (V7-9791)
• Improvement of the generation of indexes from DDS files (V7-9803)
• Improve mappers robustness with invalid numeric value (V7-9813)
• Improve SQLModel and allIndexes files generation (V7-9818)
• Improve qualified DS support (V7-9863)
• Improve support of LOOKUP (with a standalone field LIKE a DS in parameter) (V7-9961)
• Improve LIKE on indicator (V7-9985)
• Handling resulting indicator on MVR (V7-9995)
• Support character N with tilde (V7-10021)
• Improve modern DDL files generation from SQLDDS legacy files (V7-10067)

Transversal capabilities

New features

• Customize resource location with a yml property (D88816105)
• COBOL - Support of EXIT PERFORM statement to exit from an inline PERFORM without using a GO TO / PERFORM ... THROUGH (V7-9582)
• Specifying default legacy encoding to consider into global metadata. (V7-9883)

Improvements

• Improve mask generation (V7-9602)
• Improve context warm-up (V7-9621)
• Make Charset CUSTOM930 thread safe. (V7-9674)
• Improvement on MOVEA (V7-9773)
AWS Blu Age Release Notes 3.5.0

This release of AWS Blu Age Runtime and Modernization Tools provides new features for both zOS and AS400 legacy migrations, mainly oriented to datasets and messaging optimization, as well as extended Java capabilities as a resulting asset of the transformation process. Some of the key changes in this release are:

- Capability of migrating CL programs to Java in addition to the pre-existent groovy scripts feature, to facilitate its integration with other modernized programs, and to simplify customer learning curve by unifying the resulting programming language.
- Time reduction and optimization of the performance of dataset loads in Redis with the new data bulk feature.
- Ability to operate and pass datasets within job steps to modernize traditional datasets behaviors.
- Extension of SQL migration to support VB input files and Java 11 simplified migration.
- Multiple new mechanisms for faster integration with IBM MQ including additional headers, extended GET/PUT support and automatic retrieve of queue metadata.
- REST Endpoint for data sets metadata and import data sets from S3 buckets.

For more information about the changes included in this release, see the following sections.

AWS Blu Age - Runtime Release 3.5.0

Topics

- zOS (p. 196)
- AS400 (p. 197)
- Transversal capabilities (p. 198)

zOS

New features

- JCL SORT - Handle new keyword overlay (V7-9409)
- ZOS COBOL - enhance support of floating char (V7-9404)
- Port of RedisJicsTSQueue to RedisTemplate & ListOperations (V7-9212)
- ZOS JCL - enhance temporary directory's path with files directory if defined through UserDefinedParameters (V7-9012)
- Handle FUNCTION ORD-MAX with ALL (all array items) (V7-9366)
- Prefixed and human-readable keys are now used when storing TS Queues in Redis (V7-9212)
- Add get data set endpoint for blusam API
- JCL - ADD support for batch job with name involving special character like # (V7-9136)
- TSModel fetching is now robustly performed on demand (V7-9212)

Improvements

- Non-versioned INCLUDE support in LNK files (V7-6022)
- MQ - Enhance encoding support (V7-9652)
- Improving support for double bytes or mixed charsets for varying character type (V7-9596)
- JCL - Support of filesDirectory configuration in IDCAMS delete NONVSAM statements (V7-9609)
- Support bulk mode for ESDS and RRDS datasets loading from files (V7-8639)
• Handle the opening of empty ESDS in input mode. (V7-9287)
• Enhance DEFINE CLUSTER statement with ORD/UNORD abbreviation support (V7-9451)
• BluSam redis lock performance improvements (V7-8639)
• Enhance DEFINE CLUSTER statement to support RECORDSIZE provided in DATA() argument scope (V7-9337)
• Adds support of BUFFERSPACE/UNIQUE attributes on DEFINE CLUSTER statements (V7-9419)
• Improve BluSam read operation for variable length record dataset. (V7-9391)
• CICS ADDRESS properly represents missing CWA as null (V7-9491)
• Remove Unnecessary write at end locks (V7-8639)
• Handle redis cache template injection in cache (V7-9510)
• Decode correctly BPXWDYN parameter (V7-9417)
• Improvement on LISTCAT export consumption (V7-9201)
• Non-printable chars support in BluSam TS Queues name (V7-9212)
• Handle receive Map building for field with mapset null (V7-9486)
• Improve BluesamRelativeFile delete and rewrite operation for dynamic access mode. (V7-8989)

AS400

New features
• Add a feature to generate CL files as Java programs through standard DS/STM pivot (V7-9427)
• Support Input File with ADD mode (V7-9378)
• Improved sort order and retrieval management to support cl command OPNQRYF (Open Query File) and added support of SHARE parameter in OverrideItem. (V7-9364)

Improvements
• Support SFLNXTCHG on UpdateSubfile (V7-8061)
• Modify scope of CL context when run CL command (V7-9624)
• Handle return code for program BPXWDYN (V7-9417)
• Clear local monitors. (V7-9624)
• Support of DSPF keyword RTNCSRLOC (V7-9389)
• setOnGreaterOrEqual() not setting Equal to 1 (V7-9342)
• Update fields cache on UpdateSubfileRecord (V7-9376)
• Improve Support SFLNXTCHG (V7-8061)

Transversal capabilities

New features
• Ignore G prefix on literal graphic string. (V7-9420)
• ZOS COBOL - Enhance support of Field.initialize() for some special structures (V7-9485)
• Allow initialization of context asynchronously to improve performance of program startup (V7-9446)
• SQL Release explicitly the opened prepare statement and ResultSet. (V7-9422)
• Enhance JMS MQ - support MQRFH2 for MQ PUT / V7-7085 - support of default queue manager (V7-9400)
• SQL Management : Enable lambda conversions on parameters for SET commands (V7-9492)
• ZOS MQ JMS - Add support to MQCOMIT and MQBACK (V7-9399)
• ZOS IBMMQ - Enhance support to MQINQ (V7-9544)
• Handle CONCAT operation with byte instead of string when using double byte encoding. (V7-8932)
• ZOS IBMMQ - Enhance support of PUT command with options SET_ALL_CONTEXT (V7-9544)

Improvements
• Handle gdg file names with $ character (V7-9066)
• SQL Diagnostic return 1 as NUMBER clause when previous SQL statement is successful. (V7-9410)
• Outlining for field with non null length (V7-7536)
• Support built-in PL1 GRAPHIC function (V7-9245)
• MQ - Add support of version for MQGMO fields setting (V7-9500)
• JMS MQ GET - Message returned dataLength improvement (V7-9502)
• Set sqlerrd(3) with number of fetched items in ROWSET context. (V7-9371)

AWS Blu Age - Modernization tools Release 3.5.0

Topics
• zOS (p. 198)
• AS400 (p. 199)
• Transversal capabilities (p. 200)

zOS

New features
• ZOS PLI - Support asterisk index in assignment with binary expression (V7-9178)
• JCL to BatchScript - A "//" marks the end of job execution (V7-9304)
• ZOS PLI - enhance support of floating char and sign in numeric edited type (V7-8982)
• COBOL - Support of built-in SUM function (V7-9367)
• JCL- optionally, comment dead code after null statement /// (V7-9202)
• JCL- Support of operator '|' in condition statement (V7-9499)
• PL/I - Comment of precompilation directives at preprocessing step to prevent parsing exceptions (V7-9507)

Improvements
• Handle Stream definition with delimiter (V7-9615)
• Improving LISTCAT exports handling. (V7-9201)
• PL/I - Enhancement to support implicit 'null' arguments (V7-9204)

AS400

New features
• Support of DDS keyword CONCAT (V7-9439)
• Refactor the generated java code for DSPF keywords. (V7-7700)
• Support Varying keyword on fields within a data structure definition (V7-9029)
Improvements

- Improve parsing of logical relationship AND/OR (V7-9352)
- COBOL: Improve mapping between vo and dsEntity (V7-9449)
- Display empty value if numerical input is focused (V7-9374)
- Local variable in SQL Declare Cursor (V7-9456)
- Scope problem with empty DS (V7-9466)
- Truncate lines after col 80 before parsing (V7-9632)
- Improve the handle of field references and built-in calls in keywords (DIM, LIKE,...) in definition specification (V7-9358)
- Support SQL comments (--) (V7-9632)
- FullFree parsing, type Date/Time/Timestamp (V7-9542)
- Include SQLCA from FullFree parsing (V7-9333)
- Improve Support of Control Level. (V7-9610)
- Handle DS comparison with *BLANKS (V7-9668)
- Improve support of multiple indicators in DDS (V7-9318)
- Improve support of multiple DSPF program (V7-9657)
- Improve the handle of field with LIKE (case of liked data structure and case of liked data structure in an array) (V7-9213)
- Free RPG, Handle continuation on literal (V7-9686)
- Improve Support of end of program records (V7-9452)
- Support of the LINKAGE phrase in the CALL statement. (V7-9685)
- CASXX operation code (CASBB without CASXX group) (V7-9357)
- Improve FullFreeRPG parsing (V7-9457)
- Built-in %LEN does not support DS as argument (V7-9267)
- Improvements of MOVEA when factor 2 is "ALL'X...' (V7-9228)
- Support assign with RENAME field (V7-9385)

Transversal capabilities

New features

- SQL Migrator tool: Add OID option for variable record length at ebcidic loading step. (V7-9380)
- SQL Migrator tool: Support for Java 11 on OID option (V7-9599)

Improvements

- Improve support for nested arrays (V7-9595)
- Replace `~` character by `!` in case of `~` is supported by original encoding. (V7-9465)
- JCL: Support of PASS normal termination to share datasets between job steps (V7-9504)
- Apply ON NULL to column definition on ORACLE when deals with VARCHAR and nullable db column type. (V7-9681)
- Improve Spring injection compliance (V7-9635)
Replatforming applications with Micro Focus

This section describes each step in the replatforming process. It describes all tasks and includes information on configuring and operating AWS Mainframe Modernization runtime on Amazon EC2.

Topics

- Micro Focus Runtime (on Amazon EC2) Setup (p. 212)
- Tutorial: Set up AppStream 2.0 for use with Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer (p. 232)
- Tutorial: Set up Enterprise Analyzer on AppStream 2.0 (p. 238)
- Tutorial: Set up Micro Focus Enterprise Developer on AppStream 2.0 (p. 248)
- Set up Automation for Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer Streaming Sessions (p. 251)
- View Data Sets as Tables and Columns in Enterprise Developer (p. 252)
- Tutorial: Use templates with Micro Focus Enterprise Developer (p. 258)
- Tutorial: Setting up the Micro Focus build for the BankDemo sample application (p. 270)
- Tutorial: Setting up a CI/CD pipeline for use with Micro Focus Enterprise Developer (p. 276)
- Batch Utilities in AWS Mainframe Modernization (p. 293)

Micro Focus Runtime (on Amazon EC2) Setup

AWS Mainframe Modernization provides several Amazon Machine Images (AMIs) that include Micro Focus licensed products. These AMIs allow you to quickly provision Amazon Elastic Compute Cloud (Amazon EC2) instances to support Micro Focus environments that you control and manage. This topic provides the steps required to access and launch these AMIs. Using these AMIs is entirely optional and they are not required to complete the tutorials in this user guide.

Topics

- Prerequisites (p. 212)
- Create the Amazon VPC Endpoint for Amazon S3 (p. 213)
- Request the Allowlist update for the Account (p. 214)
- Creating the AWS Identity and Access Management role (p. 215)
- Grant License Manager the required permissions (p. 220)
- Subscribe to the Amazon Machine Images (p. 220)
- Launch an AWS Mainframe Modernization Micro Focus Instance (p. 223)
- Subnet or VPC with no Internet Access (p. 226)
- Troubleshooting license issues (p. 230)

Prerequisites

Make sure you meet the following prerequisites.
• Administrator access to the account where the Amazon EC2 instances will be created.
• Identify the AWS Region where the Amazon EC2 instances will be created and verify the AWS Mainframe Modernization service is available. See AWS Services by Region. Make sure to choose a Region where the service is available.
• Identify the Amazon Virtual Private Cloud (Amazon VPC) where the Amazon EC2 instances will be created.

Create the Amazon VPC Endpoint for Amazon S3

In this section, you create a Amazon VPC endpoint for Amazon S3 to use.

1. Navigate to Amazon VPC in the AWS Management Console.
2. In the navigation pane, choose Endpoints.
3. Choose Create endpoint.

4. Enter a meaningful name tag, for example: “Micro-Focus-License-S3”.
5. Choose AWS Services as the Service Category.

6. Under Services search for the Amazon S3 Gateway service: com.amazonaws.[region].s3.

   For us-west-1 this would be: com.amazonaws.us-west-1.s3.
7. Choose the Gateway service.
8. For VPC choose the VPC you will be using.

9. Choose all of the route tables for the VPC.


11. Choose Create Endpoint.

**Request the Allowlist update for the Account**

Work with your AWS representative to have your account allowlisted for the AWS Mainframe Modernization AMIs. Please provide the following information:

- The AWS account ID.
- The AWS Region where the Amazon VPC endpoint was created.
- The Amazon VPC Amazon S3 endpoint ID created in Create the Amazon VPC Endpoint for Amazon S3 (p. 213). This is the `vpce-xxxxxxxxxxxxxxxxx` id for the `com.amazonaws.[region].s3 Gateway` endpoint.
- The number of licenses required across all Micro Focus Enterprise Suite AMI Amazon EC2 instances. One license is required per CPU core (per 2 vCPUs for most Amazon EC2 instances).

For more information, see Optimize CPU options.

The requested number can be adjusted in the future by AWS.

**Note**
The AWS representative must open the support ticket for the Allowlist request. It cannot be directly requested and the request may take several days to complete.

## Creating the AWS Identity and Access Management role

Create an AWS Identity and Access Management policy and role to be used by the AWS Mainframe Modernization Amazon EC2 instances. Creating the role through the IAM console will create an associated instance profile of the same name. Assigning this instance profile to the Amazon EC2 instances allows Micro Focus Licenses to be assigned. For more information on instance profiles, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances.

### Create an IAM Policy

An IAM policy is created first and then attached to the role.

1. Navigate to AWS Identity and Access Management in the AWS Management Console.
2. Choose Policies and then Create Policy.

3. Choose the JSON tab.

4. Replace `us-west-1` in the following JSON with the AWS Region where the Amazon S3 endpoint was defined, then copy and paste the JSON into the policy editor.
Creating the AWS Identity and Access Management role

```json
{
   "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**
The Actions under the Sid OtherRequiredActions do not support resource-level permissions and must specify * in the resource element.

5. Choose **Next: Tags**.
6. Optionally enter any tags, then choose Next: Review.

7. Enter a name for the policy, for example “Micro-Focus-Licensing-policy”. Optionally enter a description, for example “A role that includes this policy must be attached to each AWS Mainframe Modernization Amazon EC2 instance.”

8. Choose Create Policy.

Create the IAM Role

1. Navigate to IAM in the AWS Management Console.

2. Choose Roles and then Create Role.

3. Leave Trusted entity type as AWS service and choose the EC2 common use case.
4. Choose **Next**.

5. Enter “Micro” into the filter and press enter to apply the filter.

6. Choose the policy that was just created, for example the “Micro-Focus-Licensing-policy”.

7. Choose **Next**.

8. Enter the Role name, for example “Micro-Focus-Licensing-role”.

9. Replace the description with one of your own, for example “Allows Amazon EC2 instances with this role to obtain Micro Focus Licenses”.
10. Under **Step 1: Select trusted entities** review the JSON and confirm it has the following values:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Principal": {
        "Service": [
          "ec2.amazonaws.com"
        ]
      }
    }
  ]
}
```

**Note**
The order of the Effect, Action, and Principal are not significant.

11. Confirm that **Step 2: Add permissions** shows your Licensing policy.

12. Choose **Create role**.
After the allowlist request is complete, continue with the following steps.

**Grant License Manager the required permissions**

1. Navigate to AWS License Manager in the AWS Management Console.

2. Choose **Start using AWS License Manager**.

3. If you see the following pop-up, view the details, then choose the check-box and press **Grant Permissions**.

**Subscribe to the Amazon Machine Images**

After you are subscribed to an AWS Marketplace product, you can launch an instance from the product's AMI.

1. Navigate to AWS Marketplace Subscriptions in the AWS Management Console.

2. Choose **Manage subscriptions**.
3. Copy and paste one of the following links into the browser address bar.

   **Note**
   Only choose a link for one of the products you have been authorized to use.

   - Enterprise Server: [https://aws.amazon.com/marketplace/pp/prodview-g5emev63l7blc](https://aws.amazon.com/marketplace/pp/prodview-g5emev63l7blc)
   - Enterprise Developer with Visual Studio 2022: [https://aws.amazon.com/marketplace/pp/prodview-m413lqiszo6cm](https://aws.amazon.com/marketplace/pp/prodview-m413lqiszo6cm)

4. Choose **Continue to Subscribe**.

5. If the Terms and Conditions are acceptable, choose **Accept Terms**.
6. The subscription might take a few minutes to process.

7. After the Thank you message shows, copy and paste the next link from step 3 to continue adding subscriptions.

8. Stop when **Manage subscriptions** shows all your subscribed AMIs.

   **Note**
   The panel preferences (gear icon) are set to show the View as a Table.
Launch an AWS Mainframe Modernization Micro Focus Instance

1. Navigate to AWS Marketplace Subscriptions in the AWS Management Console.
2. Locate the AMI to be launched and choose Launch New Instance.

![AWS Marketplace screenshot showing Enterprise Server on Windows and Enterprise Analyzer](image1)

3. In the launch new instance dialog, ensure the allowlisted region is selected.
4. Press Continue to launch through EC2.

**Note**
The following example shows a launch of an Enterprise Developer AMI, but the process is the same for all the AWS Mainframe Modernization AMIs.

![AWS Marketplace screenshot showing Enterprise Developer](image2)

5. Enter a name for the server.
6. Choose an instance type.

The Instance type selected should be determined by the project performance and cost requirements. The following are suggested starting points:

- For Enterprise Analyzer, an r6i.xlarge
- For Enterprise Developer, an r6i.large
- For a standalone instance of Enterprise Server, an r6i.xlarge
- For Micro Focus Performance Availability Cluster (PAC) with scale-out, an r6i.large

**Note**
The Application and OS Images section has been collapsed for the screen shot.

7. Choose or create (and save) a key-pair (not shown).

For more information on key pairs for Linux instances, see [Amazon EC2 key pairs and Linux instances](#).

For more information on key pairs for Windows instances, see [Amazon EC2 key pairs and Windows instances](#).

8. Edit the Network settings and choose the allowlisted VPC and appropriate Subnet.

9. **Choose or create a Security Group.** If this is an Enterprise Server EC2 instance it is typical to allow TCP traffic to ports 86 and 10086 to administer the Micro Focus configuration.

10. Optionally configure the storage for the Amazon EC2 instance.

11. Important - Expand Advanced details and under IAM instance profile choose the Licensing role created earlier, for example “Micro-Focus-Licensing-role”.

**Note**
If this step is missed, after the instance is created you can modify the IAM role from the Security option of the Action menu for the EC2 instance.
12. Review the Summary and push **Launch Instance**.

13. The instance launch will fail if an invalid virtual server type is chosen.

   If this happens, choose **Edit instance config** and change the instance type.
14. Once the “Success” message is shown choose **Connect to instance** to get connection details.

15. Alternatively, navigate to **EC2** in the AWS Management Console.

16. Choose **Instances** to see the status of the new instance.

---

**Subnet or VPC with no Internet Access**

Make these additional changes if the subnet or VPC does not have outbound Internet access.

The license manager requires access to the following AWS services:

- com.amazonaws.region.s3
- com.amazonaws.region.ec2
- com.amazonaws.region.license-manager
- com.amazonaws.region.sts

The earlier steps defined the com.amazonaws.region.s3 service as a gateway endpoint. This endpoint needs a route table entry for any subnets without Internet access.

The additional three services will be defined as interface endpoints.

**Add the Route table entry for the Amazon S3 endpoint**

1. Navigate to **VPC** in the AWS Management Console and choose **Subnets**.
2. Choose the subnet where the Amazon EC2 instances will be created and choose the Route Table tab.
3. Note a few trailing digits of the Route table id. For example, the 6b39 in the image below.

![Route table image]

4. Choose **Endpoints** from the navigation pane.
5. Choose the endpoint created earlier and then **Manage Route tables**, either from the Route Tables tab for the endpoint, or from the Actions drop down.
6. Choose the Route table using the digits identified earlier and press Modify route tables.

![Manage route tables image]

**Define the required security group**

The Amazon EC2, AWS STS, and License Manager services communicate over HTTPS via port 443. This communication is bi-directional and requires inbound and outbound rules to allow the instance to communicate with the services.

1. Navigate to Amazon VPC in the AWS Management Console.
2. Locate **Security Groups** in the navigation bar and choose **Create security group**.
3. Enter a Security group name and description, for example “Inbound-Outbound HTTPS”.
4. Press the X in the VPC selection area to **remove the default VPC**, and choose the VPC that contains the S3 endpoint.
5. Add an Inbound Rule that allows **TCP traffic on Port 443** from anywhere.

**Note**
The inbound (and outbound rules) can be restricted further by limiting the Source. For more information, see [Control traffic to your AWS resources using security groups](https://docs.aws.amazon.com/vpc/latest/userguide/security-groups.html) in the *Amazon VPC User Guide*. 

---

227
6. Press **Create security group.**

**Create the service endpoints**

Repeat this process three times – once for each service.

1. Navigate to Amazon VPC in the AWS Management Console and choose **Endpoints.**
2. Press **Create endpoint.**
3. Enter a name, for example “Micro-Focus-License-EC2”, “Micro-Focus-License-STS”, or “Micro-Focus-License-Manager”.
4. Choose the **AWS Services** Service Category.

5. Under Services search for the matching Interface service which is one of:
   - “com.amazonaws.region.ec2”
   - “com.amazonaws.region.sts”
   - “com.amazonaws.region.license-manager”

   For example:
   - “com.amazonaws.us-west-1.ec2”
   - “com.amazonaws.us-west-1.sts”
   - “com.amazonaws.us-west-1.license-manager”

6. Choose the matching Interface service.

   `com.amazonaws.region.ec2`
com.amazonaws.region.sts:

com.amazonaws.region.license-manager:

7. For VPC choose the VPC for the instance.

8. Choose the Availability Zone and the Subnets for the VPC.

10. Under Policy choose **Full Access**.

11. Choose **Create Endpoint**.

12. Repeat this process for the remaining interfaces.

**Troubleshooting license issues**

If you have trouble accessing or using the AMIs, the following information might help you.

**Verify the Amazon EC2 Instance has the IAM Licensing role**

This can be checked on the Security tab of the Amazon EC2 Instance Details. This can be changed using the Security Option of the Actions drop down menu.
Use the Reachability Analyzer

Find the Reachability Analyzer on the AWS Network Manager Console page.

Create and analyze a path between the Amazon EC2 instance created from the AMI and the Amazon S3 VPC Endpoint.

If the Amazon EC2 Instance does not have internet access repeat the path analysis to all 4 endpoints.

For more information on the Reachability Analyzer, see Getting started with Reachability Analyzer in the Reachability Analyzer guide.

Run the license-daemon

On Windows Enterprise Developer use the following command from a Command Prompt:

```
"C:\Program Files (x86)\Micro Focus\Enterprise Developer\AdoptOpenJDK\bin\java" -jar "C:\Program Files (x86)\Micro Focus\Licensing\aws-license-daemon.jar"
```

and examine the output. Ignore the SLF4J messages and look for the first exception.

On Enterprise Analyzer use the following command from a Command Prompt:

```
"C:\Program Files (x86)\Micro Focus\AdoptOpenJDK\bin\java" -jar "C:\Program Files (x86)\Micro Focus\Licensing\aws-license-daemon.jar"
```

and examine the output. Ignore the SLF4J messages and look for the first exception.

On Linux run:

```java -jar /var/microfocuslicensing/bin/aws-license-daemon.jar```

Ignore the SLF4J messages and look for the first exception.
For example, if the Amazon S3 resource is not available, the exception is as follows:

```
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)
```

The exception message indicates which resource is not available. Compare the configuration values to those shown in this topic.

**Tutorial: Set up AppStream 2.0 for use with Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer**

AWS Mainframe Modernization provides several tools through Amazon AppStream 2.0. AppStream 2.0 is a fully managed, secure application streaming service that lets you stream desktop applications to users without rewriting applications. AppStream 2.0 provides users with instant access to the applications they need with a responsive, fluid user experience on the device of their choice. Using AppStream 2.0 to host runtime engine-specific tools gives customer application teams the ability to use the tools directly from their web browsers, interacting with application files stored in either Amazon S3 buckets or CodeCommit repositories.


This document is intended for members of the customer operations team. It describes how to set up Amazon AppStream 2.0 fleets and stacks to host the Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer tools used with AWS Mainframe Modernization. Micro Focus Enterprise Analyzer is usually used during the Assess phase and Micro Focus Enterprise Developer is usually used during the Migrate and Modernize phase of the AWS Mainframe Modernization approach. If you plan to use both Enterprise Analyzer and Enterprise Developer you must create separate fleets and stacks for each tool. Each tool requires its own fleet and stack because their licensing terms are different.

**Important**
The steps in this tutorial are based on the downloadable AWS CloudFormation template [cfn-m2-appstream-fleet-ea-ed.yml](https://aws.amazon.com/appstream20/administration-guide/).

**Topics**
- Prerequisites (p. 233)
- Step 1: Get the AppStream 2.0 images (p. 233)
- Step 2: Create the stack using the AWS CloudFormation template (p. 233)
- Step 3: Create a user in AppStream 2.0 (p. 235)
- Step 4: Log in to AppStream 2.0 (p. 236)
- Step 5: Verify buckets in Amazon S3 (optional) (p. 237)
- Next steps (p. 238)
- Clean up resources (p. 238)
Prerequisites

- Download the template: [cfn-m2-appstream-fleet-ea-ed.yml](#).
- Get the ID of your default VPC and security group. For more information on the default VPC, see [Default VPCs](#) in the Amazon VPC User Guide. For more information on the default security group, see [Default and custom security groups](#) in the Amazon EC2 User Guide for Linux Instances.
- Make sure you have the following permissions:
  - create stacks, fleets, and users in AppStream 2.0.
  - create stacks in AWS CloudFormation using a template.
  - create buckets and upload files to buckets in Amazon S3.
  - download credentials (`access_key_id` and `secret_access_key`) from IAM.

Step 1: Get the AppStream 2.0 images

In this step, you share the AppStream 2.0 images for Enterprise Analyzer and Enterprise Developer with your AWS account.

1. Open the AWS Mainframe Modernization console at [https://console.aws.amazon.com/m2/](https://console.aws.amazon.com/m2/).
2. In the left navigation, choose **Tools**.
3. In **Analysis, development, and build assets**, choose Share assets with my AWS account.

Step 2: Create the stack using the AWS CloudFormation template

In this step, you use the downloaded AWS CloudFormation template to create an AppStream 2.0 stack and fleet for running Micro Focus Enterprise Analyzer. You can repeat this step later to create another AppStream 2.0 stack and fleet for running Micro Focus Enterprise Developer, since each tool requires its own fleet and stack in AppStream 2.0. For more information on AWS CloudFormation stacks, see Working with stacks in the AWS CloudFormation User Guide.

**Note**
AWS Mainframe Modernization adds an additional fee to the standard AppStream 2.0 pricing for the use of Enterprise Analyzer and Enterprise Developer. For more information, see [AWS Mainframe Modernization Pricing](#).

1. Download the [cfn-m2-appstream-fleet-ea-ed.yml](#) template, if necessary.
2. Open the AWS CloudFormation console and choose **Create Stack** and **with new resources (standard)**.
3. In **Prerequisite - Prepare template**, choose **Template is ready**.
4. In **Specify Template**, choose **Upload a template file**.
5. In **Upload a template file**, choose **Choose file** and upload the [cfn-m2-appstream-fleet-ea-ed.yml](#) template.
6. Choose **Next**.
7. On **Specify stack details**, enter the following information:
   
   - In **Stack name**, enter a name of your choice. For example, **m2-ea**.
   - In **AppStreamApplication**, choose **ea**.
   - In **AppStreamFleetSecurityGroup**, choose your default VPC’s default security group.
   - In **AppStreamFleetVpcSubnet**, choose a subnet within your default VPC.
   - In **AppStreamImageName**, choose the image starting with **m2-enterprise-analyzer**. This image contains the currently supported version of the Micro Focus Enterprise Analyzer tool.
   - Accept the defaults for the other fields, then choose **Next**.
8. Accept all defaults, then choose Next again.
9. On Review, make sure all the parameters are what you intend.
10. Scroll to the bottom, choose I acknowledge that AWS CloudFormation might create IAM resources with custom names, and choose Create Stack.

It takes between 20 and 30 minutes for the stack and fleet to be created. You can choose Refresh to see the AWS CloudFormation events as they occur.

**Step 3: Create a user in AppStream 2.0**

While you are waiting for AWS CloudFormation to finish creating the stack, you can create one or more users in AppStream 2.0. These users are those who will be using Enterprise Analyzer in AppStream 2.0. You will need to specify an email address for each user, and ensure that each user has sufficient permissions to create buckets in Amazon S3, upload files to a bucket, and link to a bucket to map its contents.

1. Open the AppStream 2.0 console.
2. In the left navigation, choose User pool.
3. Choose Create user.
4. Provide an email address where the user can receive an email invitation to use AppStream 2.0, a first name and last name, and choose Create user.
5. Repeat if necessary to create more users. The email address for each user must be unique.
Step 4: Log in to AppStream 2.0

For more information on creating AppStream 2.0 users, see [AppStream 2.0 User Pools](#) in the Amazon AppStream 2.0 Administration Guide.

When AWS CloudFormation finishes creating the stack, you can assign the user you created to the stack, as follows:

1. Open the AppStream 2.0 console.
2. Choose the user name.
3. Choose Action, then Assign stack.
4. In Assign stack, choose the stack that begins with m2-appstream-stack-ea.
5. Choose Assign stack.

Assigning a user to a stack causes AppStream 2.0 to send an email to the user at the address you provided. This email contains a link to the AppStream 2.0 login page.

**Step 4: Log in to AppStream 2.0**

In this step, you log in to AppStream 2.0 using the link in the email sent by AppStream 2.0 to the user you created in [Step 3: Create a user in AppStream 2.0](#) (p. 235).

1. Log in to AppStream 2.0 using the link provided in the email sent by AppStream 2.0.
2. Change your password, if prompted. The AppStream 2.0 screen that you see is similar to the following:
3. Choose Desktop.
4. On the task bar, choose Search and enter D: to navigate to the Home Folder.

   **Note**
   If you skip this step, you might get a Device not ready error when you try to access the Home Folder.

At any point, if you have trouble signing into AppStream 2.0, you can restart your AppStream 2.0 fleet and try to sign in again, using the following steps.

1. Open the AppStream 2.0 console.
2. In the left navigation, choose Fleets.
3. Choose the fleet you are trying to use.
4. Choose Action, then choose Stop.
5. Wait for the fleet to stop.
6. Choose Action, then choose Start.

This process can take around 10 minutes.

**Step 5: Verify buckets in Amazon S3 (optional)**

One of the tasks completed by the AWS CloudFormation template you used to create the stack was to create two buckets in Amazon S3, which are necessary to save and restore user data and application settings across work sessions. These buckets are as follows:

- Name starts with appstream2-.. This bucket maps data to your Home Folder in AppStream 2.0 (D: \PhotonUser\My Files\Home Folder).

   **Note**
   The Home Folder is unique for a given email address and is shared across all fleets and stacks in a given AWS account. The name of the Home Folder is a SHA256 hash of the user’s email address, and is stored on a path based on that hash.

- Name starts with appstream-app-settings-. This bucket contains user session information for AppStream 2.0, and includes settings such as browser favorites, IDE and application connection profiles, and UI customizations. For more information, see [How Application Settings Persistence Works](#) in the *Amazon AppStream 2.0 Administration Guide*.

To verify that the buckets were created, follow these steps:

1. Open the Amazon S3 console.
2. In the left navigation, choose Buckets.
3. In **Find buckets by name**, enter `appstream` to filter the list.

If you see the buckets, no further action is necessary. Just be aware that the buckets exist. If you do not see the buckets, then either the AWS CloudFormation template is not finished running, or an error occurred. Go to the AWS CloudFormation console and review the stack creation messages.

**Next steps**

Now that the AppStream 2.0 infrastructure is set up, you can set up and start using Enterprise Analyzer. For more information, see [Tutorial: Set up Enterprise Analyzer on AppStream 2.0](p. 238). You can also set up Enterprise Developer. For more information, see [Tutorial: Set up Micro Focus Enterprise Developer on AppStream 2.0](p. 248).

**Clean up resources**

The procedure to clean up the created stack and fleets is described in [Create an AppStream 2.0 Fleet and Stack](p. 238).

When the AppStream 2.0 objects have been deleted, the account administrator can also, if appropriate, clean up the Amazon S3 buckets for Application Settings and Home Folders.

**Note**

The home folder for a given user is unique across all fleets, so you might need to retain it if other AppStream 2.0 stacks are active in the same account.

Finally, AppStream 2.0 does not currently allow you to delete users using the console. Instead, you must use the service API with the CLI. For more information, see [User Pool Administration](p. 248) in the Amazon AppStream 2.0 Administration Guide.

**Tutorial: Set up Enterprise Analyzer on AppStream 2.0**

This tutorial describes how to set up Micro Focus Enterprise Analyzer to analyze one or more mainframe applications. The Enterprise Analyzer tool provides several reports based on its analysis of the application source code and system definitions.

This setup is designed to foster team collaboration. Installation uses an Amazon S3 bucket to share the source code with virtual disks. Doing this makes use of Rclone on the Windows machine. With a common Amazon RDS instance running PostgreSQL, any member of the team can access to all requested reports.

Team members can also mount the virtual Amazon S3 backed disk on their personal machines and update the source bucket from their workstations. They can potentially use scripts or any other form of automation on their machines if they are connected to other on-premises internal systems.

The setup is based on the AppStream 2.0 Windows images that AWS Mainframe Modernization shares with the customer. Setup is also based on the creation of AppStream 2.0 fleets and stacks as described in [Tutorial: Set up AppStream 2.0 for use with Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer](p. 232).

**Important**

The steps in this tutorial assume that you set up AppStream 2.0 with the downloadable AWS CloudFormation template `cfn-m2-appstream-fleet-ea-ed.yml`. For more information, see
Tutorial: Set up AppStream 2.0 for use with Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer (p. 232)

To perform the steps in this tutorial, you must have set up your Enterprise Analyzer fleet and stack and they must be running.

For a complete description of Enterprise Analyzer features and deliverables, see the Enterprise Analyzer Documentation on the Micro Focus website.

Image contents

In addition to Enterprise Analyzer application itself, the image contains the following tools and libraries.

Third-party tools

- Python
- Rclone
- pgAdmin
- git-scm
- PostgreSQL ODBC driver

Libraries in C:\Users\Public

- BankDemo source code and project definition for Enterprise Developer: m2-bankdemo-template.zip.
- MFA install package for the mainframe: mfa.zip. For more information, see Mainframe Access Overview in the Micro Focus Enterprise Developer documentation.
- Command and config files for Rclone (instructions for their use in the tutorials): m2-rclone.cmd and m2-rclone.conf.

Topics

- Prerequisites (p. 239)
- Step 1: Setup (p. 240)
- Step 2: Create the Amazon S3 based virtual folder on Windows (p. 240)
- Step 3: Create an ODBC source for the Amazon RDS instance (p. 241)
- Subsequent sessions (p. 242)
- Troubleshooting workspace connection (p. 242)
- Clean up resources (p. 247)

Prerequisites

- Upload the source code and system definitions for the customer application that you want to analyze to an S3 bucket. The system definitions include CICS CSD, DB2 object definitions, and so on. You can create a folder structure within the bucket that makes sense for how you want to organize the application artifacts. For example, when you unzip the BankDemo sample, it has the following structure:

```
demo
  |--- jcl
```
Create and start an Amazon RDS instance running PostgreSQL. This instance will store the data and results produced by Enterprise Analyzer. You can share this instance with all members of the application team. In addition, create an empty schema called m2_ea (or any other suitable name) in the database. Define credentials for authorized users that allow them to create, insert, update, and delete items in this schema. You can obtain the database name, its server endpoint URL, and TCP port from the Amazon RDS console or from the account administrator.

Make sure you have set up programmatic access to your AWS account. For more information, see Programmatic access in the Amazon Web Services General Reference.

Step 1: Setup

1. Start a session with AppStream 2.0 with the URL that you received in the welcome email message from AppStream 2.0.
2. Use your email as your user ID, and define your permanent password.
3. Select the Enterprise Analyzer stack.
4. On the AppStream 2.0 menu page, choose Desktop to reach the Windows desktop that the fleet is streaming.

Step 2: Create the Amazon S3 based virtual folder on Windows

Note
If you already used Rclone during the AWS Mainframe Modernization preview, you must update m2-rclone.cmd to the newer version located in C:\Users\Public.

1. Copy the m2-rclone.conf and m2-rclone.cmd files provided in C:\Users\Public to your home folder C:\Users\PhotonUser\My Files\Home Folder using File Explorer.
2. Update the m2-rclone.conf config parameters with your AWS access key and corresponding secret, as well as your AWS Region.

```conf
[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. In m2-rclone.cmd, make the following changes:
   - Change your-s3-bucket to your Amazon S3 bucket name. For example, m2-s3-mybucket.
   - Change your-s3-folder-key to your Amazon S3 bucket key. For example, myProject.
   - Change your-local-folder-path to the path of the directory where you want the application files synced from the Amazon S3 path that contains them. For example, D:\PhotonUser\My Files\Home Folder\m2-new. This synced directory must be a subdirectory of the Home Folder in order for AppStream 2.0 to properly back up and restore it on session start and end.
Step 3: Create an ODBC source for the Amazon RDS instance

1. To start the EA_Admin tool, navigate to the application selector menu in the top left corner of the browser window and choose MF EA_Admin.

2. From the Administer menu, choose ODBC Data Sources, and choose Add from the User DSN tab.

3. In the Create New Data Source dialog box, choose the PostgreSQL Unicode driver, and then choose Finish.

4. In the PostgreSQL Unicode ODBC Driver (psqlODBC) Setup dialog box, define and take note of the data source name that you want. Complete the following parameters with the values from the RDS instance that you previously created:

   **Description**
   
   Optional description to help you identify this database connection quickly.

   **Database**
   
   The Amazon RDS database you created previously.

   **Server**
   
   The Amazon RDS endpoint.

   **Port**
   
   The Amazon RDS port.

   **User Name**
   
   As defined in the Amazon RDS instance.

   **Password**
   
   As defined in the Amazon RDS instance.

5. Choose Test to validate that the connection to Amazon RDS is successful, and then choose Save to save your new User DSN.
6. Wait until you see the message that confirms creation of the proper workspace, and then choose **Ok** to finish with ODBC Data Sources and close the EA_Admin tool.

7. Navigate again to the application selector menu, and choose Enterprise Analyzer to start the tool. Choose **Create New**.

8. In the Workspace configuration window, enter your workspace name and define its location. The workspace can be the Amazon S3 based disk if you work under this config, or your home folder if you prefer.

9. Choose **Choose Other Database** to connect to your Amazon RDS instance.

10. Choose the **Postgre** icon from the options, and then choose **ok**.

11. For the Windows settings under **Options – Define Connection Parameters**, enter the name of the data source that you created. Also enter the database name, the schema name, the user name, and password. Choose **Ok**.

12. Wait for Enterprise Analyzer to create all the tables, indexes, and so on that it needs to store results. This process might take a couple of minutes. Enterprise Analyzer confirms when the database and workspace are ready for use.

13. Navigate again to the application selector menu and choose Enterprise Analyzer to start the tool.

14. The Enterprise Analyzer startup window appears in the new, selected workspace location. Choose **Ok**.

15. Navigate to your repository in the left pane, select the repository name, and choose **Add files / folders to your workspace**. Select the folder where your application code is stored to add it to the workspace. You can use the previous BankDemo example code if you want. When Enterprise Analyzer prompts you to verify those files, choose **Verify** to start the initial Enterprise Analyzer verification report. It might take some minutes to complete, depending on the size of your application.

16. Expand your workspace to see the files and folders that you've added to the workspace. The object types and cyclomatic complexity reports are also visible in the top quadrant of the **Chart Viewer** pane.

You can now use Enterprise Analyzer for all needed tasks.

**Subsequent sessions**

1. Start a session with AppStream 2.0 with the URL that you received in the welcome email message from AppStream 2.0.

2. Log in with your email and permanent password.

3. Select the Enterprise Analyzer stack.

4. Launch **Rclone** to connect to the Amazon S3 backed disk if you use this option to share the workspace files.

5. Launch Enterprise Analyzer to do your work.

**Troubleshooting workspace connection**

When you try to reconnect to your Enterprise Analyzer workspace, you might see an error like this:

> 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?

To resolve this issue, choose **OK** to clear the message, and then complete the following steps.
1. In AppStream 2.0, choose the Launch Application icon on the toolbar, and then choose **EA_Admin** to start the Micro Focus Enterprise Analyzer Administration tool.

2. From the **Administer** menu, choose **Refresh Workspace Path**...
3. Under **Select workspace**, choose the workspace that you want, and then choose **OK**.
4. Choose OK to confirm the error message.
5. Under **Workspace directory network path**, enter the correct path to your workspace, for example, 
\[D:\ PhotonUser\My Files\Home Folder\EA\MyWorkspace3.\]

6. Close the Micro Focus Enterprise Analyzer Administration tool.

7. In AppStream 2.0, choose the Launch Application icon on the toolbar, and then choose **EA** to start 
Micro Focus Enterprise Analyzer.
8. Repeat steps 3 - 5.

Micro Focus Enterprise Analyzer should now open with the existing workspace.

**Clean up resources**

If you no longer need the resources that you created for this tutorial, delete them so that you don't incur further charges. Complete the following steps:

- Use the **EA_Admin** tool to delete the workspace.
Tutorial: Set up Micro Focus Enterprise Developer on AppStream 2.0

This tutorial describes how to set up Micro Focus Enterprise Developer for one or more mainframe applications in order to maintain, compile, and test them using the Enterprise Developer features. The setup is based on the AppStream 2.0 Windows images that AWS Mainframe Modernization shares with the customer and on the creation of AppStream 2.0 fleets and stacks as described in Tutorial: Set up AppStream 2.0 for use with Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer (p. 232).

**Important**
The steps in this tutorial assume that you set up AppStream 2.0 using the downloadable AWS CloudFormation template `cfn-m2-appstream-fleet-ea-ed.yaml`. For more information, see Tutorial: Set up AppStream 2.0 for use with Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer (p. 232).

You must perform the steps of this setup when the Enterprise Developer fleet and stack are up and running.

For a complete description of Enterprise Developer v7 features and deliverables, check out its up-to-date online documentation (v7.0) on the Micro Focus site.

**Image contents**

In addition to Enterprise Developer itself, the image contains Rumba (a TN3270 emulator). It also contains the following tools and libraries.

Third-party tools

- Python
- Rclone
- pgAdmin
- git-scm
- PostgreSQL ODBC driver

Libraries in `C:\Users\Public`

- BankDemo source code and project definition for Enterprise Developer: `m2-bankdemo-template.zip`.
- MFA install package for the mainframe: `mfa.zip`. For more information, see Mainframe Access Overview in the Micro Focus Enterprise Developer documentation.
- Command and config files for Rclone (instructions for their use in the tutorials): `m2-rclone.cmd` and `m2-rclone.conf`.

If you need to access source code that is not yet loaded into CodeCommit repositories, but that is available in an Amazon S3 bucket, for example to perform the initial load of the source code into git, follow the procedure to create a virtual Windows disk as described in Tutorial: Set up Enterprise Analyzer on AppStream 2.0 (p. 238).
Prerequisites

- One or more CodeCommit repositories loaded with the source code of the application to be maintained. The repository setup should match the requirements of the CI/CD pipeline above to create synergies by combination of both tools.
- Each user must have credentials to the CodeCommit repository or repositories defined by the account administrator according to the information in Authentication and access control for AWS CodeCommit. The structure of those credentials is reviewed in Authentication and access control for AWS CodeCommit and the complete reference for IAM authorizations for CodeCommit is in the CodeCommit permissions reference: the administrator may define distinct IAM policies for distinct roles having credentials specific to the role for each repository and limiting its authorizations of the user to the specific set of tasks that he has to accomplish on a given repository. So, for each maintainer of the CodeCommit repository, the account administrator will generate a primary user and grant this user permissions to access the required repository or repositories via selecting the proper IAM policy or policies for CodeCommit access.

Step 1: Setup by individual Enterprise Developer users

1. Obtain your IAM credentials:
   2. Follow the procedure described in step 3 of Setup for HTTPS users using Git credentials in the AWS CodeCommit User Guide.
   3. Copy the CodeCommit-specific sign-in credentials that IAM generated for you, either by showing, copying, and then pasting this information into a secure file on your local computer, or by choosing Download credentials to download this information as a .CSV file. You need this information to connect to CodeCommit.
2. Start a session with AppStream 2.0 based on the url received in the welcome email. Use your email as user name and create your password.
3. Select your Enterprise Developer stack.
4. On the menu page, choose Desktop to reach the Windows desktop streamed by the fleet.

Step 2: Create the Amazon S3-based virtual folder on Windows (optional)

If there is a need for Rclone (see above), create the Amazon S3-based virtual folder on Windows: (optional if all application artefacts exclusively come from CodeCommit access).

Subsequent sessions

Clean up resources
Step 3: Clone the repository

1. Navigate to the application selector menu in the top left corner of the browser window and select Enterprise Developer.

2. Complete the workspace creation required by Enterprise Developer in your Home folder by choosing C:\Users\PhotonUser\My Files\Home Folder (aka D: \PhotonUser\My Files\Home Folder) as location for the workspace.
3. In Enterprise Developer, clone your CodeCommit repository by going to the Project Explorer, right click and choose **Import, Import …, Git, Projects** from **Git Clone URI**. Then, enter your CodeCommit-specific sign-in credentials and complete the Eclipse dialog to import the code.

The CodeCommit git repository in now cloned in your local workspace.

Your Enterprise Developer workspace is now ready to start the maintenance work on your application. In particular, you can use the local instance of Microfocus Enterprise Server (ES) integrated with Enterprise Developer to interactively debug and run your application to validate your changes locally.

**Note**
The local Enterprise Developer environment, including the local Enterprise Server instance, runs under Windows while AWS Mainframe Modernization runs under Linux. We recommend that you run complementary tests in the Linux environment provided by AWS Mainframe Modernization after you commit the new application to CodeCommit and rebuild it for this target and before you roll out the new application to production.

**Subsequent sessions**

As you select a folder that is under AppStream 2.0 management like the home folder for the cloning of your CodeCommit repository, it will be saved and restored transparently across sessions. Complete the following steps the next time you need to work with the application:

1. Start a session with AppStream 2.0 based on the url received in the welcome email.
2. Login with your email and permanent password.
3. Select the Enterprise Developer stack.
4. Launch **Rc1one** to connect (see above) to the Amazon S3-backed disk when this option is used to share the workspace files.
5. Launch Enterprise Developer to do your work.

**Clean up resources**

If you no longer need the resources you created for this tutorial, delete them so that you won't continue to be charged for them. Complete the following steps:

- Delete the CodeCommit repository you created for this tutorial. For more information, see **Delete an CodeCommit repository** in the **AWS CodeCommit User Guide**.
- Delete the database you created for this tutorial. For more information, see **Deleting a DB instance**.

**Set up Automation for Micro Focus Enterprise Analyzer and Micro Focus Enterprise Developer Streaming Sessions**

You can automatically run a script at session start and end to allow automation that is specific to your customer context. For more information on this AppStream 2.0 feature, see **Use Session Scripts to Manage Your AppStream 2.0 Users’ Streaming Experience** in the **Amazon AppStream 2.0 Administration Guide**.

This feature requires that you have at least the following versions of the Enterprise Analyzer and Enterprise Developer images:
Set up automation at session start

If you want to run an automation script when users connect to AppStream 2.0, create your script and name it `m2-user-setup.cmd`. Store the script in the AppStream 2.0 Home folder for the user. The AppStream 2.0 images that AWS Mainframe Modernization provides look for a script with that name in that location, and run it if it exists.

**Note**
The script duration cannot exceed the limit set by AppStream 2.0, which is currently 60 seconds. For more information, see `Run Scripts Before Streaming Sessions Begin` in the Amazon AppStream 2.0 Administration Guide.

Set up automation at session end

If you want to run an automation script when users disconnect from AppStream 2.0, create your script and name it `m2-user-teardown.cmd`. Store the script in the AppStream 2.0 Home folder for the user. The AppStream 2.0 images that AWS Mainframe Modernization provides look for a script with that name in that location, and run it if it exists.

**Note**
The script duration cannot exceed the limit set by AppStream 2.0, which is currently 60 seconds. For more information, see `Run Scripts After Streaming Sessions End` in the Amazon AppStream 2.0 Administration Guide.

View Data Sets as Tables and Columns in Enterprise Developer

You can access mainframe datasets that are deployed in AWS Mainframe Modernization using the Micro Focus runtime. You can view the migrated data sets as tables and columns from an Micro Focus Enterprise Developer instance. Viewing data sets this way allows you to:

- Perform SQL `SELECT` operations on the migrated data files.
- Expose data outside the migrated mainframe application without changing the application.
- Easily filter data and save as CSV or other file formats.

**Note**
Steps 1 and 2 are one time activities. Repeat steps 3 and 4 for each data set to create the database views.

Topics
- Prerequisites (p. 253)
Prerequisites

- You must have access to Micro Focus Enterprise Developer Desktop via AppStream 2.0.
- You must have an application deployed and running under AWS Mainframe Modernization using the Micro Focus runtime engine.
- You are storing your application data in Aurora PostgreSQL-Compatible Edition.

Step 1: Set up ODBC Connection to Micro Focus datastore (Amazon RDS database)

In this step, you set up an ODBC connection to the database that contains the data you want to view as tables and columns. This is a one-time only step.

1. Log in to Micro Focus Enterprise Developer Desktop using AppStream 2.0 streaming URL.
2. Open ODBC Data Source Administrator, choose User DSN and then choose Add.
3. In Create New Data Source, choose PostgreSQL ANSI and then choose Finish.
4. Create a data source for PG.POSTGRES by providing the necessary database information, as follows:

<table>
<thead>
<tr>
<th>Data Source</th>
<th>PG.POSTGRES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>postgres</td>
</tr>
<tr>
<td>Server</td>
<td>rds_endpoint.rds.amazonaws.com</td>
</tr>
<tr>
<td>Port</td>
<td>5432</td>
</tr>
<tr>
<td>User Name</td>
<td>user_name</td>
</tr>
<tr>
<td>Password</td>
<td>user_password</td>
</tr>
</tbody>
</table>

Step 2: Create the MFDBFH.cfg file (p. 255)

Step 3: Create a structure (STR) file for your copybook layout (p. 256)

Step 4: Create a database view using the structure (STR) file (p. 257)

Step 5: View Micro Focus data sets as tables and columns (p. 258)
5. Choose **Test** to make sure the connection works. You should see the message **Connection successful** if the test succeeds.

If the test doesn't succeed, review the following information.

- [Troubleshooting for Amazon RDS](#)
- [How do I resolve problems when connecting to my Amazon RDS DB instance?](#)

6. Save the data source.

7. Create a data source for **PG.VSAM**, test the connection, and save the data source. Provide the following database information:

<table>
<thead>
<tr>
<th>Data Source</th>
<th>PG.VSAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>MicroFocus$SEE$Files$VSAM</td>
</tr>
<tr>
<td>Server</td>
<td>rds_endpoint.rds.amazonaws.com</td>
</tr>
<tr>
<td>Port</td>
<td>5432</td>
</tr>
<tr>
<td>User Name</td>
<td>user_name</td>
</tr>
<tr>
<td>Password</td>
<td>user_password</td>
</tr>
</tbody>
</table>
Step 2: Create the MFDBFH.cfg file

In this step, you create a configuration file that describes the Micro Focus data store. This is a one-time only configuration step.

1. In your Home Folder, for example, in D:\PhotonUser\My Files\Home Folder\MFED\cfg \MFDBFH.cfg, create the MFDBFH.cfg file with the following content:

```xml
<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. Verify the MFDBFH configuration by running the following commands to query the Micro Focus datastore:

```bash
***
*** 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
```
Step 3: Create a structure (STR) file for your copybook layout

In this step, you create a structure file for your copybook layout so that you can use it later to create database views from the data sets.

1. Compile the program that is associated with your copybook. If no program is using the copybook, create and compile a simple program like the following with a COPY statement for your copybook.

```assembler
IDENTIFICATION DIVISION.
  PROGRAM-ID. TESTPGM1.

  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.

  DATA DIVISION.
  WORKING-STORAGE SECTION.

  COPY CVTRA05Y.

  PROCEDURE DIVISION.
  GOBACK.
```

2. After successful compilation, right click on the program and choose Create Record Layout File. This will open the Micro Focus Data File Tools using the .idy file generated during the compilation.

3. Right click on the Record structure and choose Create Default Layout (single structure) or Create Conditional Layout (multi structure) depending on the layout.

   For more information, see Creating Structure Files and Layouts in the Micro Focus documentation.
Step 4: Create a database view using the structure (STR) file

In this step, you use the previously created structure file to create a database view for a data set.

- Use the `dbfhview` command to create a database view for a data set that is already in the Micro Focus datastore as shown in the following example.

```
## 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://ESPACEDatabase/VSAM/AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT?folder=/DATA
```
Step 5: View Micro Focus data sets as tables and columns

In this step, connect to the database using pgAdmin so you can run queries to view the datasets like tables and columns.

- Connect to the database MicroFocus$SEE$Files$VSAM using pgAdmin and query the database view you created in step 4.

```
SELECT * FROM public."V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT";
```

Tutorial: Use templates with Micro Focus Enterprise Developer

This tutorial describes how to use templates and predefined projects with Micro Focus Enterprise Developer. It covers three use cases. All of the use cases use the sample code provided in the BankDemo sample. To download the sample, choose bankdemo.zip.
Important
If you use the version of Enterprise Developer for Windows, the binaries generated by the compiler can run only on the Enterprise Server provided with Enterprise Developer. You cannot run them under the AWS Mainframe Modernization runtime, which is based on Linux.

Topics
- Use Case 1 - Using the COBOL Project Template containing source components (p. 259)
- Use Case 2 - Using the COBOL Project Template without source components (p. 262)
- Use Case 3 - Using the pre-defined COBOL project linking to the source folders (p. 264)
- Using the Region Definition JSON Template (p. 267)

Use Case 1 - Using the COBOL Project Template containing source components

This use case requires you to copy the source components into the Template directory structure as part of the demo pre setup steps. In the bankdemo.zip this has been changed from the original AWSTemplates.zip delivery to avoid having two copies of the source.

1. Start Enterprise Developer and specify the chosen workspace.

![Eclipse Launcher](image)

Select a directory as workspace
Eclipse uses the workspace directory to store its preferences and development artifacts.

Workspace: /awsWorkspace

2. Within the Application Explorer view, from the Enterprise Development Project tree view item, choose New Project from Template from the context menu.

![Application Explorer](image)
3. Enter the template parameters as shown.

   **Note**
   The Template Path will refer to where the ZIP was extracted.

4. Choosing OK will create a local development Eclipse Project based on the provided template, with a complete source and execution environment structure.
The System structure contains a complete resource definition file with the required entries for BANKDEMO, the required catalog with entries added and the corresponding ASCII data files.

Because the source template structure contains all the source items, these files are copied to the local project and therefore are automatically built in Enterprise Developer.
Use Case 2 - Using the COBOL Project Template without source components

Steps 1 to 3 are identical to Use Case 1 - Using the COBOL Project Template containing source components (p. 259).

The System structure in this use case also contains a complete resource definition file with the required entries for BankDemo, the required catalog with entries added, and the corresponding ASCII data files.

However, the template source structure does not contain any components. You must import these into the project from whatever source repository you are using.

1. Choose the project name. From the related context menu, choose Import.

2. From the resulting dialog, under the General section, choose File System and then choose Next.
3. Populate the From directory field by browsing the file system to point to the repository folder. Choose all the folders you wish to import, such as sources. The Into folder field will be pre-populated. Choose Finish.

After the source template structure contains all the source items, they are built automatically in Enterprise Developer.
Use Case 3 - Using the pre-defined COBOL project linking to the source folders

1. Start Enterprise Developer and specify the chosen workspace.

![Eclipse Launcher](image)

   Select a directory as workspace
   Eclipse uses the workspace directory to store its preferences and development artifacts.

   Workspace: `C:AWSWorkspace`

   Use this as the default and do not ask again
   Recent Workspaces

   ![Launch](image)

   ![Cancel](image)

2. From the File menu, choose Import.
3. From the resulting dialog, under General, choose Projects from Folder or Archive and choose Next.
4. Populate **Import source**, Choose **Directory** and browse through the file system to select the pre-defined project folder. The project contained within has links to the source folders in the same repository.

Choose **Finish**.

Because the project is populated by the links to the source folder, the code is automatically built.
Using the Region Definition JSON Template

1. Switch to the Server Explorer view. From the related context menu, choose **Open Administration Page**, which starts the default browser.

2. From the resulting Enterprise Server Common Web Administration (ESCWA) screen, choose **Import**.

3. Choose the **JSON** import type and choose **Next**.

4. Upload the supplied BANKDEMO.JSON file.
Once selected, choose Next.

On the Select Regions panel, ensure that the Clear Ports from Endpoints option is not selected, and then continue to choose Next through the panels until the Perform Import panel is shown. Then choose Import.

Finally click Finish. The BANKDEMO region will then be added to the server list.
5. Navigate to the **General Properties** for the BANKDEMO region.

6. Scroll to the **Configuration** section.

7. The ESP environment variable needs to be set to the System folder relevant to the Eclipse Project created in the previous steps. This should be `workspacefolder/projectname/System`.

---

**ADDITIIONAL**

```plaintext
[ES-Environment]
ESP={Enter Project System Folder Here}
MF_CHARSET=A
EXTFH=$ESP/EXTFH.cfg
```

8. Click **Apply**.

---

The region is now fully configured to run in conjunction with the Eclipse COBOL project.

9. Finally, back in Enterprise Developer, associate the imported region with the project.
The Enterprise Developer environment is now ready to use, with a complete working version of BankDemo. You can edit, compile, and debug code against the region.

**Important**
If you use the version of Enterprise Developer for Windows, the binaries generated by the compiler can run only on the Enterprise Server provided with Enterprise Developer. You cannot run them under the AWS Mainframe Modernization runtime, which is based on Linux.

**Tutorial: Setting up the Micro Focus build for the BankDemo sample application**

AWS Mainframe Modernization provides you with the ability to set up builds and continuous integration/continuous delivery (CI/CD) pipelines for your migrated applications. These builds and pipelines use AWS CodeBuild, AWS CodeCommit, and AWS CodePipeline to provide these capabilities. CodeBuild is a fully managed build service that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. CodeCommit is a version control service that enables you to privately store and manage Git repositories in the AWS Cloud. CodePipeline is a continuous delivery service that enables you to model, visualize, and automate the steps required to release your software.

This tutorial demonstrates how to use AWS CodeBuild to compile the BankDemo sample application source code from Amazon S3 and then export the compiled code back to Amazon S3.

AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you can use prepackaged build environments, or you can create custom build environments that use your own build tools. This demo scenario uses the second option. It consists of a CodeBuild build environment that uses a pre-packaged Docker image.

**Important**
Before you start your mainframe modernization project, we recommend that you learn about the AWS Migration Acceleration Program (MAP) for Mainframe or contact AWS mainframe specialists to learn about the steps required to modernize a mainframe application.

**Topics**
- Prerequisites (p. 271)
- Step 1: Create Amazon S3 buckets (p. 271)
Prerequisites

Before you start this tutorial, complete the following prerequisites.

- Download the BankDemo sample application and unzip it to a folder. The source folder contains COBOL programs and Copybooks, and CICS BMS definitions. It also contains a JCL folder for reference, although you do not need to build JCL. The folder also contains the meta files required for the build.
- In the AWS Mainframe Modernization console, choose Tools. In Analysis, development, and build assets, choose Share assets with my AWS account.

Step 1: Create Amazon S3 buckets

In this step, you create two Amazon S3 buckets. The first is an input bucket to hold the source code, and the other is an output bucket to hold the build output. For more information, see Creating, configuring, and working with Amazon S3 buckets in the Amazon S3 User Guide.

1. To create the input bucket, log in to the Amazon S3 console and choose Create bucket.
2. In General configuration, provide a name for the bucket and specify the AWS Region where you want to create the bucket. An example name is codebuild-regionId-accountId-input-bucket, where regionId is the AWS Region of the bucket, and accountId is your AWS account ID.

   **Note**
   If you are creating the bucket in a different AWS Region from US East (N. Virginia), specify the LocationConstraint parameter. For more information, see Create Bucket in the Amazon Simple Storage Service API Reference.

3. Retain all other settings and choose Create bucket.
4. Repeat steps 1-3 to create the output bucket. An example name is codebuild-regionId-accountId-output-bucket, where regionId is the AWS Region of the bucket and accountId is your AWS account ID.

   Whatever names you choose for these buckets, be sure to use them throughout this tutorial.

Step 2: Create the build spec file

In this step, you create a build spec file. This file provides build commands and related settings, in YAML format, for CodeBuild to run the build. For more information, see Build specification reference for CodeBuild in the AWS CodeBuild User Guide.

1. Create a file named buildspec.yml in the directory that you unzipped as a prerequisite.
2. Add the following content to the file and save. No changes are required for this file.
Step 3: Upload the source files

In this step, you upload the source code for the BankDemo sample application to your Amazon S3 input bucket.

1. Log in to the Amazon S3 console and choose Buckets in the left navigation pane. Then choose the input bucket you created previously.
2. Under Objects, choose Upload.
3. In the Files and folders section, choose Add Files.
5. Choose Upload.
Step 4: Create IAM policies

In this step, you create two IAM policies. One policy grants permissions for AWS Mainframe Modernization to access and use the Docker image that contains the Micro Focus build tools. This policy is not customized for customers. The other policy grants permissions for AWS Mainframe Modernization to interact with the input and output buckets, and with the Amazon CloudWatch logs that CodeBuild generates.

To learn about creating an IAM policy, see Editing IAM policies in the IAM User Guide.

To create a policy for accessing Docker images

1. In the IAM console, copy the following policy document and paste it into the policy editor.

```json
{
  "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. Provide a name for the policy, for example, m2CodeBuildPolicy.

To create a policy that allows AWS Mainframe Modernization to interact with buckets and logs

1. In the IAM console, copy the following policy document and paste it into the policy editor. Make sure to update `regionId` to the AWS Region, and `accountId` to your AWS account.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```
Step 5: Create an IAM role

In this step, you create a new IAM role that allows CodeBuild to interact with AWS resources for you, after you associate the IAM policies that you previously created with this new IAM role.

For information about creating a service role, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

1. Log in to the IAM console and choose Roles in the left navigation pane.
2. Choose Create role.
3. Under Trusted entity type, choose AWS service.
4. Under Use cases for other AWS services, choose CodeBuild, and then choose CodeBuild again.
5. Choose Next.
6. On the Add permissions page, choose Next. You assign a policy to the role later.
7. Under Role details, provide a name for the role, for example, BankdemoCodeBuildServiceRole.
8. Under Select trusted entities, verify that the policy document looks like the following:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "codebuild.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```
9. Choose Create role.

**Step 6: Attach the IAM policies to the IAM role**

In this step, you attach the two IAM policies you previously created to the BankdemoCodeBuildServiceRole IAM role.

1. Log in to the IAM console and choose Roles in the left navigation pane.
2. In Roles, choose the role you created previously, for example, BankdemoCodeBuildServiceRole.
3. In Permissions policies, choose Add permissions, and then Attach policies.
4. In Other permissions policies, choose the policies that you created previously, for example, m2CodeBuildPolicy and BankdemoCodeBuildRolePolicy.
5. Choose Attach policies.

**Step 7: Create the CodeBuild project**

In this step, you create the CodeBuild project.

1. Log in to the CodeBuild console and choose Create build project.
2. In the Project configuration section, provide a name for the project, for example, codebuild-bankdemo-project.
3. In the Source section, for Source provider, choose Amazon S3, and then choose the input bucket you created previously, for example, codebuild-regionId-accountId-input-bucket.
4. In the S3 object key or S3 folder field, enter the name of the zip file that you uploaded to the S3 bucket. In this case, the file name is bankdemo.zip.
5. In the Environment section, choose Custom image.
6. In the Environment type field, choose Linux.
7. Under Image registry, choose Other registry.
8. In the External registry URL field, enter 673918848628.dkr.ecr.us-west-2.amazonaws.com/m2-enterprise-build-tools:latest.
9. Under Service role, choose Existing service role, and in the Role ARN field, choose the service role you created previously; for example, BankdemoCodeBuildServiceRole.
10. In the Buildspec section, choose Use a buildspec file.
11. In the Artifacts section, under Type, choose Amazon S3, and then choose your output bucket, for example, codebuild-regionId-accountId-output-bucket.
12. In the Name field, enter the name of a folder in the bucket that you want to contain the build output artifacts, for example, bankdemo-output.zip.
14. Choose Create build project.

**Step 8: Start the build**

In this step, you start the build.
Step 9: Download output artifacts

In this step, you download the output artifacts from Amazon S3. The Micro Focus build tool can create several different executable types. In this tutorial, it generates shared objects.

1. Log in to the Amazon S3 console.
2. In the Buckets section, choose the name of your output bucket, for example, codebuild-regionId-accountId-output-bucket.
3. Choose Download.
4. Unzip the downloaded file. Navigate to the target folder to see the build artifacts. These include the .so Linux shared objects.

Clean up resources

If you no longer need the resources that you created for this tutorial, delete them to avoid additional charges. To do so, complete the following steps:

- Delete the S3 buckets that you created for this tutorial. For more information, see Deleting a bucket in the Amazon Simple Storage Service User Guide.
- Delete the IAM policies that you created for this tutorial. For more information, see Deleting IAM policies in the IAM User Guide.
- Delete the IAM role that you created for this tutorial. For more information, see Deleting roles or instance profiles in the IAM User Guide.
- Delete the CodeBuild project that you created for this tutorial. For more information, see Delete a build project in CodeBuild in the AWS CodeBuild User Guide.

Tutorial: Setting up a CI/CD pipeline for use with Micro Focus Enterprise Developer

This tutorial shows you how to import, edit, compile, and run the BankDemo sample application in Micro Focus Enterprise Developer, and then to commit your changes to trigger a CI/CD pipeline.

Contents
Prerequisites

Download the following files.

- basic-infra.yaml
  - Download from Europe (Frankfurt) Region.
  - Download from US East (N. Virginia) Region.
- pipeline.yaml
  - Download from Europe (Frankfurt) Region.
  - Download from US East (N. Virginia) Region.
- m2-code-sync-function.zip
  - Download from Europe (Frankfurt) Region.
  - Download from US East (N. Virginia) Region.
- config_git.yml
  - Download from Europe (Frankfurt) Region.
  - Download from US East (N. Virginia) Region.
- BANKDEMO-source.zip
  - Download from Europe (Frankfurt) Region.
  - Download from US East (N. Virginia) Region.
- BANKDEMO-exercise.zip

Prerequisites (p. 277)

Create CI/CD pipeline basic infrastructure (p. 278)

Create AWS CodeCommit repository and CI/CD pipeline (p. 280)
  - Sample YAML Trigger File config_git.yml (p. 281)

Enterprise Developer AppStream 2.0 Creation (p. 284)

Enterprise Developer Setup and Test (p. 284)
  - Clone the BankDemo CodeCommit repository in Enterprise Developer (p. 285)
  - Create BankDemo mainframe COBOL project and build application (p. 285)
  - Create local BankDemo CICS and batch environment for testing (p. 285)
  - Start the BANKDEMO server from Enterprise Developer (p. 286)
  - Start the Rumba 3270 terminal (p. 286)
  - Run a BankDemo transaction (p. 287)
  - Stop the BANKDEMO server from Enterprise Developer (p. 287)

Exercise 1: Enhance Loan Calculation in BANKDEMO Application (p. 287)
  - Add loan analysis rule to Enterprise Developer Code Analysis (p. 288)
    - Step 1: Perform code analysis for loan calculation (p. 288)
    - Step 2: Modify CICS BMS map and COBOL program and test (p. 289)
    - Step 3: Add total amount calculation in COBOL program (p. 289)
    - Step 4: Commit changes and run CI/CD pipeline (p. 290)

Exercise 2: Extract loan calculation in BankDemo application (p. 290)
  - Step 1: Refactor loan calculation routine into a COBOL section (p. 290)
  - Step 2: Extract loan calculation routine to a standalone COBOL program (p. 291)
  - Step 3: Commit changes and run the CI/CD pipeline (p. 292)

Clean up resources (p. 292)
Create CI/CD pipeline basic infrastructure

Use the AWS CloudFormation template basic-infra.yaml to create the CI/CD pipeline basic infrastructure stack through the AWS CloudFormation console. This stack creates Amazon S3 buckets where you upload your application code and data, and a supporting AWS Lambda function to create other necessary resources such as an AWS CodeCommit repository and an AWS CodePipeline pipeline.

**Note**

To launch this stack you need permissions to administer IAM, Amazon S3, Lambda, and AWS CloudFormation and permissions to use AWS KMS.

2. Create a new stack by using one of the following options:

   - Choose **Create Stack**. This is the only option if you have a currently running stack.
• On the **Stacks** page, choose **Create Stack**. This option is visible only if you have no running stacks.

3. **On the Specify template page:**

• In **Prepare template**, choose **Template is ready**.

• In **Specify template**, choose **Amazon S3 URL** as the template source and enter one of the following URLs depending on your AWS Region.

  - 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

• To accept your settings, choose **Next**.

The **Create stack** page opens.

### 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?

- No (Create one)

**Which VPC ID should be used?**

If you selected "Yes" for the existing VPC, this parameter is required. Otherwise, this value will be ignored.

**Which private subnet ID should be used?**

If you selected "Yes" for the existing VPC, 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 the existing VPC, this parameter is required. Otherwise, this value will be ignored.

Enter the CIDR block that should be used for the new VPC

- 11.11.0.0/16

**CIDR bits for creating subnets. Choose 5 for /27, 6 for /26, 7 for /25, 8 for /24 range**

- 5

**Deployment Configuration**

Name of the S3 bucket which contains the source files for this stack deployment

- m2

Name of the source package file for the infrastructure Lambda function

- cicd/m2-code-sync-function.zip

**Full URL of the pipeline CloudFormation template file**

- 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 cannot prefix for the S3 buckets that will be created by this stack.

- m2
Create AWS CodeCommit repository and CI/CD pipeline

In this step, you create a CodeCommit repository and provision a CI/CD pipeline stack by calling a Lambda function that calls AWS CloudFormation to create the pipeline stack.

1. Download the BankDemo sample application to your local machine.
2. Upload bankdemo.zip from your local machine to the Amazon S3 bucket created in Create CI/CD pipeline basic infrastructure (p. 278).
3. Download config_git.yml.
4. **Modify the `config_git.yml` if needed, as follows:**

   - Add your own target repository name, target branch and commit message.

   ```yaml
   repository-config:
   target-repository: bankdemo-repo
   target-branch: main
   commit-message: Initial commit for bankdemo-repo main branch
   
   • Add the email address you want to receive notifications.

   ```yaml
   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. **Upload the `config_git.yml` file containing the definition of the CodeCommit repository folder structure to the Amazon S3 bucket created in Create CI/CD pipeline basic infrastructure (p. 278).** This will invoke the Lambda function that will automatically provision the repository and pipeline.

   This will create a CodeCommit repository with the name provided in the `target-repository` defined in the `config_git.yml` file; for example, bankdemo-repo.

   The Lambda function will also create the CI/CD pipeline stack through AWS CloudFormation. The AWS CloudFormation stack will have the same prefix as the `target-repository` name provided followed by a random string (for example `bankdemo-repo-01234567`). You can find the CodeCommit repository URL and the URL to access the created pipeline in the AWS Management Console.

   6. **If the CodeCommit repository creation is complete, the CI/CD pipeline will be triggered immediately to perform a full CI/CD.**

   7. **Once the file has been pushed it will automatically trigger the pipeline which will build, deploy in staging, run some tests and wait for manual approval before getting it deployed in the production environment.**

**Sample YAML Trigger File 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'
    - 'ESPSQLXA.dll'
    - 'ESPSQLXA64.so'
    - 'ESPSQLXA64_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
      - 'ESPSQLXA.dll': XA switch module Windows
      - 'ESPSQLXA64.so': XA switch module Linux
      - 'ESPSQLXA64_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'
Create AWS CodeCommit repository and CI/CD pipeline

- 'KBNKSRT1.txt'

# 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:
Enterprise Developer AppStream 2.0 Creation

To set up Micro Focus Enterprise Developer on AppStream 2.0, see Tutorial: Set up Micro Focus Enterprise Developer on AppStream 2.0 (p. 248).

To connect the CodeCommit repository to Enterprise Developer, use the name specified in target-repository in Sample YAML Trigger File config_git.yml (p. 281).

Enterprise Developer Setup and Test

Topics

- Clone the BankDemo CodeCommit repository in Enterprise Developer (p. 284)
- Create BankDemo mainframe COBOL project and build application (p. 285)
- Create local BankDemo CICS and batch environment for testing (p. 285)
- Start the BANKDEMO server from Enterprise Developer (p. 286)
- Start the Rumba 3270 terminal (p. 286)
- Run a BankDemo transaction (p. 287)
- Stop the BANKDEMO server from Enterprise Developer (p. 287)

Connect to the Enterprise Developer AppStream 2.0 instance you created in Enterprise Developer AppStream 2.0 Creation (p. 284).

1. Start Enterprise Developer from Windows Start. Choose Micro Focus Enterprise Developer, then choose Enterprise Developer for Eclipse. If you are starting for the first time, it might take some time.

2. In the Eclipse Launcher, in Workspace: enter C:\Users\<username>\workspace then choose Launch.

   Note
   Make sure you choose the same location after reconnecting to the AppStream 2.0 instance. Workspace selection is not persistent.

3. In Welcome, choose Open COBOL Perspective. This will only be shown the first time for a new workspace.

Clone the BankDemo CodeCommit repository in Enterprise Developer

1. Choose Window / Perspective / Open Perspective / Other ... / Git.
2. Choose Clone a Git repository.
3. In **Clone Git Repository**, enter the following information:
   - **In Location URI**, enter the HTTPS URL of the CodeCommit repository.

   **Note**
   Copy the Clone URL HTTPS for the CodeCommit repository in the AWS Management Console and paste it here. The URI will be split into the **Host** and **Repository** paths.
   - **The user CodeCommit repository credentials in Authentication User and Password** and choose **Store in Secure Store**.

4. In **Branch Selection**, choose **Main branch**, then choose **Next**.
5. In **Local Destination**, in **Directory**, enter **C:\Users\<username>\workspace** and choose **Finish**.

The clone process is completed when **BANKDEMO [main]** is shown in the **Git Repositories** view.

### Create BankDemo mainframe COBOL project and build application

1. Change to **COBOL Perspective**.
2. In **Project**, disable **Build Automatically**.
3. In **File**, choose **New**, then **Mainframe COBOL Project**.
4. In **New Mainframe COBOL Project**, enter the following information:
   - **In Project name**, enter **BankDemo**.
   - Choose **Micro Focus template [64 bit]**.
   - Choose **Finish**.
5. In **COBOL Explorer**, expand the new BankDemo project.

   **Note**
   **[BANKDEMO main]** in square brackets indicates that the project is connected with the local BankDemo CodeCommit repository.

6. If the tree view does not show entries for COBOL Programs, Copybooks, BMS Source, and JCL Files, choose **Refresh** from the BankDemo project context menu.
7. From the BankDemo context menu, choose **Properties / Micro Focus / Project Settings / COBOL**:
   - Choose **Character Set - ASCII**.
   - Choose **Apply**, then **Close**.
8. If the build of the BMS and COBOL source does not immediately start, check in the **Project** menu, that the option **Build Automatically** is enabled.

   The Build output will be displayed in the **Console** view and should complete after a few minutes with messages **BUILD SUCCESSFUL** and **Build finished with no errors**.

   The BankDemo application should now be compiled and ready for local execution.

### Create local BankDemo CICS and batch environment for testing

1. In **COBOL Explorer**, expand **BANKDEMO / config**.
2. In the editor, open **BANKDEMO_ED.json**.
3. Find string **ED_Home=** and change path to point to the Enterprise Developer project, as follows: **D:\\<username>\\workspace\\BANKDEMO**. Note the use of double slashes (`\`) in the path definition.
4. Save and close the file.
5. Choose **Server Explorer**.

6. From the **Default** context menu, choose **Open Administration Page**. The Micro Focus Enterprise Server **Administration** page is opened in the default browser.

7. For AppStream 2.0 sessions only, make the following changes so you can preserve your local Enterprise Server region for local testing:
   - In **Directory Server / Default**, choose **PROPERTIES / Configuration**.
   - Replace **Repository Location** with `D:\<username>\My Files\Home Folder\MFDS`.

   **Note**
   You must complete steps 5 - 8 after every new connection to an AppStream 2.0 instance.

8. In **Directory Server / Default**, choose **Import**, then complete the following steps:
   - In **Step 1: Import Type**, choose **JSON** and choose **Next**.
   - In **Step 2: Upload**, click to upload file in blue square.
   - In **Choose File to Upload**, enter:
     - **File name:** `D:\<username>\workspace\BANKDEMO\config\BANKDEMO_ED.json`
     - Choose **Open**.
     - Choose **Next**.
   - In **Step 3: Regions** clear **Clear Ports from Endpoints**.
     - Choose **Next**.
   - In **Step 4: Import**, choose **Import**.
   - Choose **Finish**.

The list will now show a new server name BANKDEMO.

**Start the BANKDEMO server from Enterprise Developer**

1. Choose **Enterprise Developer**.

2. In **Server Explorer**, choose **Default**, then choose **Refresh** from the context menu.

   The server list should now also show BANKDEMO.

3. Choose **BANKDEMO**.

4. From the context menu, choose **Associate with project**, then choose **BANKDEMO**.

5. From the context menu, choose **Start**.

   The Console view should display the log for the server startup.

   If the message **BANKDEMO CASSI5030I PLTPI Phase 2 List(PI) Processing Completed** is displayed, the Server is ready for testing the CICS BANKDEMO application.

**Start the Rumba 3270 terminal**


2. In **Welcome**, choose **CREATE NEW SESSION / Mainframe Display**.

3. In **Mainframe Display**, choose **Connection / Configure**.

4. In **Session Configuration**, choose **Connection / TN3270**.

5. In **Host Name / Address**, choose **Insert** and enter IP address `127.0.0.1`.
6. In Telnet Port, enter port 6000.
7. Choose Apply.
8. Choose Connect.

The CICS welcome screen displays screen with row 1 message: This is the Micro Focus MFE CICS region BANKDEMO.

Run a BankDemo transaction

1. In an empty screen, enter BANK.
2. In screen BANK10, in the input field for User id....:, enter guest and press Enter.
3. In screen BANK20, in the input field before Calculate the cost of a loan, enter / (forward slash) and press Enter.
4. In screen BANK70:
   • In The amount you would like to borrow...:, enter 10000.
   • In At an interest rate of................:, enter 5.0.
   • In For how many months...................:, enter 10.
   • Press Enter.

The following result should be displayed:

```
Resulting monthly payment.............:   $1023.06
```

This completes the BANKDEMO application setup in Enterprise Developer.

Stop the BANKDEMO server from Enterprise Developer

1. In Server Explorer, choose Default, then choose Refresh from the context menu.
2. Choose BANKDEMO.
3. From the context menu, choose Stop.

The Console view should display the log for the server stopping.

If the message Server: BANKDEMO stopped successfully is displayed, the server has successfully shut down.

Exercise 1: Enhance Loan Calculation in BANKDEMO Application

Topics
- Add loan analysis rule to Enterprise Developer Code Analysis (p. 288)
- Step 1: Perform code analysis for loan calculation (p. 288)
- Step 2: Modify CICS BMS map and COBOL program and test (p. 289)
- Step 3: Add total amount calculation in COBOL program (p. 289)
**Exercise 1: Enhance Loan Calculation in BANKDEMO Application**

**Step 4: Commit changes and run CI/CD pipeline (p. 290)**

In this scenario, you walk through the process of making a sample change to the code, deploying it, and testing it.

The Loan department wants a new field on the Loan Calculation screen BANK70 to show the Total Loan Amount. This requires a change of the BMS screen MBANK70.CBL, adding a new field and the corresponding screen handling program SBANK70P.CBL with related copybooks. In addition, the loan calculation routine in BBANK70P.CBL needs to be extended with the additional formula.

To complete this exercise, make sure you complete the following prerequisites.

- Download BANKDEMO-exercise.zip to D:\PhotonUser\My Files\Home Folder.
- Extract the zip file to D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise.
- Create folder D:\PhotonUser\My Files\Home Folder\AnalysisRules.
- Copy the rules file Loan+Calculation+Update.General-1.xml from the BANKDEMO-exercise folder to D:\PhotonUser\My Files\Home Folder\AnalysisRules.

**Note**

Code changes in *.CBL and *.CPY are marked with EXER01 in column 1 - 6 for this exercise.

**Add loan analysis rule to Enterprise Developer Code Analysis**

Analysis rules defined in Micro Focus Enterprise Analyzer can be exported from Enterprise Analyzer and imported into Enterprise Developer to run same analysis rules across the sources in the Enterprise Developer project.

1. Open Window/Preferences/Micro Focus/COBOL/Code Analysis/Rules.
2. Choose Edit... and enter the folder name D:\PhotonUser\My Files\Home Folder\AnalysisRules containing the rules file Loan+Calculation+Update.General-1.xml.
3. Choose Finish.
4. Choose Apply, then choose Close.
5. From the BANKDEMO project context menu, choose Code Analysis.

You should see an entry for Loan Calculation Update.

**Step 1: Perform code analysis for loan calculation**

With the new analysis rule we want to identify the COBOL programs and lines of code in there that are matching the search patterns *PAYMENT*, *LOAN* and *RATE* in expressions, statements and variables. This will help to navigate through the code and identify required code changes.

1. From the BANKDEMO project context menu, choose Code Analysis/Loan Calculation Update.

   This will run the search rule and list the results in a new tab called Code Analysis. The analysis run is completed when the green progress bar at the bottom right disappears.

   The Code Analysis tab should display an expanded list of BBANK20P.CBL, BBANK70P.CBL, SBANK70P.CBL, each listing the statements, expressions and variables matching the search patterns.

   Looking at the result for BBANK20P.CBL there are only literals moved that have a match with search pattern. So this program can be ignored.

2. In the tab menu bar choose - Icon to collapse all.
3. Expand SBANK70P.CBL and select any lines in any order with a double-click to see how this will open the source and highlight the line selected in source code. You will also recognize that all identified source lines are marked.

**Step 2: Modify CICS BMS map and COBOL program and test**

First we will change the BMS map MBANK70.BMS and the screen handling program SBANK70P.CBL and copybook CBANKDAT.CPY to display the new field. To avoid unnecessary coding in this exercise, modified source modules are available in the D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise\Exercise01 folder. Normally a developer would use the Code Analysis results to navigate and modify the sources. If you have the time and want to do the manual changes do so with the information provided in *Manual change in MBANK70.BMS and SBANK70P.CBL (Optional)*.

For quick changes, copy the following files:

1. ..\BANKDEMO-exercise\Exercis01\screens\MBANK70.BMS to D:\PhotonUser\workspace\bankdemo\source\screens.
2. ..\BANKDEMO-exercise\Exercis01\cobol\SBANK70P.CBL to D:\PhotonUser\workspace\bankdemo\source\cobol.
3. ..\BANKDEMO-exercise\Exercis01\copybook\CBANKDAT.CPY to D:\PhotonUser\workspace\bankdemo\source\copybook.
4. To ensure that all programs impacted by the changes are compiled, choose Project/Clean.../Clean all project.

For manual changes to MBANK70.BMS and SBANK70P.CBL, complete the following steps:

- For manual change in BMS MBANK70.BMS source add after the PAYMENT field:
  - TXT09 with same attributes as TXT08 and INITIAL value “Total Loan Amount”
  - TOTAL with same attributes as PAYMENT

**Test changes**

To test the changes, repeat the steps in the following sections:

1. Start the BANKDEMO server from Enterprise Developer (p. 286)
2. Start the Rumba 3270 terminal (p. 286)
3. Run a BankDemo transaction (p. 287)
   In addition you should now also see the text Total Loan Amount..............:
4. Stop the BANKDEMO server from Enterprise Developer (p. 287)

**Step 3: Add total amount calculation in COBOL program**

In the second step we will change BBANK70P.CBL and add the calculation for the total loan amount. The prepared source with required changes is available in D:\PhotonUser\My Files\Home Folder\BANKDEMO-exercise\Exercise01 folder. If you have the time and want to do the manual changes do so with the information provided in *Manual change in BBANK70P.CBL (Optional)*.

For quick change, copy the following file:

- ..\BANKDEMO-exercise\Exercis01\source\cobol\BBANK70P.CBL to D:\PhotonUser\workspace\bankdemo\source\cobol.
To make a manual change to BBANK70P.CBL, complete the following steps:

- Use the Code Analysis result to identify the required changes.

**Test changes**

To test the changes, repeat the steps in the following sections:

1. [Start the BANKDEMO server from Enterprise Developer (p. 286)](#)
2. [Start the Rumba 3270 terminal (p. 286)](#)
3. [Run a BankDemo transaction (p. 287)](#)
   
   In addition you should now also see the text Total Loan Amount: $10230.60.
4. [Stop the BANKDEMO server from Enterprise Developer (p. 287)](#)

**Step 4: Commit changes and run CI/CD pipeline**

Commit the changes to the central CodeCommit repository and trigger the CI/CD pipeline to build, test, and deploy the changes.

1. From BANKDEMO project, in the context menu, choose Team/Commit.
2. In the Git Staging tab, enter the following commit message: Added Total Amount Calculation.
3. Choose Commit and Push....
4. Open the CodePipeline console and check status of the pipeline execution.

   **Note**
   
   In case you face any problem with the Enterprise Developer or Teams function Commit or Push, use the Git Bash command line interface.

**Exercise 2: Extract loan calculation in BankDemo application**

**Topics**

- Step 1: Refactor loan calculation routine into a COBOL section (p. 290)
- Step 2: Extract loan calculation routine to a standalone COBOL program (p. 291)
- Step 3: Commit changes and run the CI/CD pipeline (p. 292)

In this next exercise, you work through another sample change request. In this scenario, the Loan department want to reuse the loan calculation routine as a standalone WebService. The routine should remain in COBOL and should also still be callable from the existing CICS COBOL program BBANK70P.CBL.

**Step 1: Refactor loan calculation routine into a COBOL section**

In the first step we extract the loan calculation routine into a COBOL Section. This step is required to extract the code into a stand-alone COBOL program in the next step.

1. Open BBANK70P.CBL in the COBOL Editor.
2. In the editor, choose from the context menu **Code Analysis/Loan Calculation Update**. This will only scan the current source for patterns defined in the analysis rule.

3. In the result in the **Code Analysis** tab, find the first arithmetic statement `DIVIDE WS-LOAN-INTEREST BY 12`.

4. Double click on the statement to navigate to source line in Editor. This is the first statement of the loan calculation routine.

5. Mark the following code block for loan calculation routine to be extracted to a section.

   ```cobol
   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. From the context menu in the editor, choose **Refactor/Extract to Section**.

7. Enter **New section name: LOAN-CALCULATION**.

8. Choose OK.

   The marked code block has now been extracted to the new LOAN-CALCULATION section and the code block has been replaced with the PERFROM LOAN-CALCULATION statement.

**Test changes**

To test the changes repeat the steps described in the following sections.

1. **Start the BANKDEMO server from Enterprise Developer (p. 286)**

2. **Start the Rumba 3270 terminal (p. 286)**

3. **Run a BankDemo transaction (p. 287)**

   In addition you should now also see the text `Total Loan Amount.....................: $10230.60`.

4. **Stop the BANKDEMO server from Enterprise Developer (p. 287)**

**Note**

If you want to avoid the above steps to extract the code block to a section you can copy the modified source for Step 1 from `..\BANKDEMO-exercise\Exercises02\Step1\cobol \BBANK70P.CBL` to `D:\PhotonUser\workspace\bankdemo\source\cobol`.

**Step 2: Extract loan calculation routine to a standalone COBOL program**

In Step 2 the code block in the LOAN-CALCULATION section will be extracted to a standalone program and the original code will be replaced with code to call the new subprogram.

1. Open `BBANK70P.CBL` in editor and find the new `PERFORM LOAN-CALCULATION` statement created in Step 1.

2. Place the cursor within the section name. It will be marked grey.

3. From the context menu, select **Refactor->Extract Section/Paragraph to Program**.
4. In Extract Section/Paragraph to Program, enter New file name: LOANCALC.CBL.
5. Choose OK.

   The new LOANCALC.CBL program will open in the editor.
6. Scroll down and review the code being extracted and generated for the call interface.
7. Select editor with BBANK70P.CBL and go to LOAN-CALCULATION SECTION. Review the code being generated to call the new sub-program LOANCALC.CBL.

   **Note**
   The CALL statement is using DFHEIBLK and DFHCOMMAREA to call LOANCALC with CICS control blocks. Because we want to call the new LOANCALC.CBL sub-program as non-CICS program, we have to remove DFHEIBLK and DFHCOMMAREA from the call either by commenting out or deleting.

**Test changes**

To test the changes repeat the steps described in the following sections.

1. **Start the BANKDEMO server from Enterprise Developer (p. 286)**
2. **Start the Rumba 3270 terminal (p. 286)**
3. **Run a BankDemo transaction (p. 287)**

   In addition you should now also see the text Total Loan Amount.....................: $10230.60.
4. **Stop the BANKDEMO server from Enterprise Developer (p. 287)**

   **Note**
   If you want to avoid the above steps to extract the code block to a section you can copy the modified source for Step 1 from ..\BANKDEMO-exercise\Exercise02\Step2\cobol\BBANK70P.CBL and LOANCALC.CBL to D:\PhotonUser\workspace\bankdemo\source\cobol.

**Step 3: Commit changes and run the CI/CD pipeline**

Commit the changes to the central CodeCommit repository and trigger the CI/CD Pipeline to build, test and deploy the changes.

1. From BANKDEMO project, in the context menu, choose Team/Commit.
2. In the Git Staging tab

   • Add in Unstaged Stages LOANCALC.CBL and LOANCALC.CBL.mfdiritset.
   • Enter a commit message: Added Total Amount Calculation.
3. Choose Commit and Push....
4. Open the CodePipeline console and check status of the pipeline execution.

   **Note**
   In case you face any problem with the Enterprise Developer or Teams function Commit or Push, use the Git Bash command line interface.

**Clean up resources**

If you no longer need the resources you created for this tutorial, delete them so that you won’t continue to be charged for them. Complete the following steps:
Batch Utilities in AWS Mainframe Modernization

Mainframe applications often use batch utility programs to perform specific functions such as sorting data, transferring files using FTP, loading data into databases like DB2, unloading data from databases, and so on.

When you migrate your applications to AWS Mainframe Modernization, you need functionally equivalent replacement utilities that can perform the same tasks as the ones you used on the mainframe. Some of these utilities might already be available as part of the AWS Mainframe Modernization runtime engines, but we are providing the following replacement utilities:

- M2SFTP - enables secure file transfer using SFTP protocol.
- M2WAIT - waits for a specified amount of time before continuing with the next step in a batch job.
- TXT2PDF - converts text files to PDF format.

We developed these batch utilities based on customer feedback and designed them to provide the same functionality as the mainframe utilities. The goal is to make your transition from mainframe to AWS Mainframe Modernization as smooth as possible.

**Topics**

- Binary Location (p. 293)
- M2SFTP Batch Utility (p. 293)
- M2WAIT Batch Utility (p. 298)
- TXT2PDF Batch Utility (p. 299)

**Binary Location**

These utilities are preinstalled on the Micro Focus Enterprise Developer (ED) and Micro Focus Enterprise Server (ES) products. You can find them in the following location for all variants of ED and ES:

- Linux: `/opt/aws/m2/microfocus/utilities/64bit`
- Windows (32 bit): `C:\AWS\M2\MicroFocus\Utilities\32bit`
- Windows (64 bit): `C:\AWS\M2\MicroFocus\Utilities\64bit`

**M2SFTP Batch Utility**

M2SFTP is a JCL utility program designed to perform secure file transfers between systems using the Secure File Transfer Protocol (SFTP). The program uses the Putty SFTP client, `psftp`, to perform the
actual file transfers. The program works similarly to a mainframe FTP utility program and uses user and password authentication.

**Note**
Public key authentication is not supported.

To convert your mainframe FTP JCLs to use SFTP, change PGM=FTP to PGM=M2SFTP.

**Topics**
- Supported Platforms (p. 294)
- Installing Dependencies (p. 294)
- Configure M2SFTP for AWS Mainframe Modernization Managed (p. 294)
- Configure M2SFTP for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0) (p. 295)
- Sample JCLs (p. 295)
- Putty SFTP (PSFTP) client command reference (p. 297)
- Next steps (p. 298)

**Supported Platforms**

You can use M2SFTP on any of the following platforms:

- AWS Mainframe Modernization Micro Focus Managed
- Micro Focus Runtime (on Amazon EC2)
- All variants of Micro Focus Enterprise Developer (ED) and Micro Focus Enterprise Server (ES) products.

**Installing Dependencies**

**To install the Putty SFTP client on Windows**

- Download the **PuTTY SFTP** client and install it.

**To install the Putty SFTP client on Linux:**

- Run the following command to install the Putty SFTP client:

  ```
  sudo yum -y install putty
  ```

**Configure M2SFTP for AWS Mainframe Modernization Managed**

If your migrated applications are running on AWS Mainframe Modernization Managed, you will need to configure M2SFTP as follows.

- Set the appropriate Micro Focus Enterprise Server environment variables for MFFTP. Here are few examples:
  - MFFTP_TEMP_DIR
  - MFFTP_SENDEOL
  - MFFTP_TIME
• **MFFTP_ABEND**

You can set as few or as many of these variables as you want. You can set them in your JCL using the `ENVAR DD` statement. For more information on these variables, see [MFFTP Control Variables](#) in the Micro Focus documentation.

To test your configuration, see [Sample JCLs](#).

**Configure M2SFTP for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)**

If your migrated applications are running on AWS Mainframe Modernization runtime on Amazon EC2, configure M2SFTP as follows.

1. Change the [Micro Focus JES Program Path](#) to include the binary location for batch utilities. If you need to specify multiple paths, use colons (:) to separate paths on Linux and semicolons (;) on Windows.
   - Linux: `/opt/aws/m2/microfocus/utilities/64bit`
   - Windows (32bit): `C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows (64bit): `C:\AWS\M2\MicroFocus\Utilities\64bit`

2. Set the appropriate Micro Focus Enterprise Server environment variables for MFFTP. Here are few examples:
   - `MFFTP_TEMP_DIR`
   - `MFFTP_SENDEOL`
   - `MFFTP_TIME`
   - `MFFTP_ABEND`

You can set as few or as many of these variables as you want. You can set them in your JCL using the `ENVAR DD` statement. For more information on these variables, see [MFFTP Control Variables](#) in the Micro Focus documentation.

To test your configuration, see [Sample JCLs](#).

**Sample JCLs**

To test the installation, you can use either of the following sample JCL files.

**M2SFTP1.jcl**

This JCL shows how to call M2SFTP to send a file to a remote SFTP server. Notice the environment variables that are set in the `ENVAR DD` statement.

```bash
//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*
// Sample SFTP JCL step to send a file to SFTP server*
```
M2SFTP2.jcl

This JCL shows how to call M2SFTP to receive a file from a remote SFTP server. Notice the environment variables set in the ENVVAR DD statement.

```
//M2SFTP22 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=*  
//STDDOUT DD SYSOUT=* 
//INPUT DD *
open 127.0.0.1
sftpuser
sftppass
locsite recfm=fb lrecl=150
get 'AWS.M2.CARDDEMO.CARDDATA.PS' AWS.M2.CARDDEMO.CARDDATA.PS1.txt +
'AWS.M2.CARDDEMO.CARDDATA.PS2' (replace
quit
//*
//ENVVAR DD *
```
MFFTP_VERBOSE_OUTPUT=ON
MFFTP_KEEP=N
/*
//*
//

**Note**
We strongly recommend storing FTP credentials in a NETRC file and restricting access to only authorized users.

**Putty SFTP (PSFTP) client command reference**

The PSFTP client does not support all FTP commands. The following list shows all the commands that PSFTP does support.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Run a local command</td>
</tr>
<tr>
<td>bye</td>
<td>Finish your SFTP session</td>
</tr>
<tr>
<td>cd</td>
<td>Change your remote working directory</td>
</tr>
<tr>
<td>chmod</td>
<td>Change file permissions and modes</td>
</tr>
<tr>
<td>close</td>
<td>Finish your SFTP session but do not quit PSFTP</td>
</tr>
<tr>
<td>del</td>
<td>Delete files on the remote server</td>
</tr>
<tr>
<td>dir</td>
<td>List remote files</td>
</tr>
<tr>
<td>exit</td>
<td>Finish your SFTP session</td>
</tr>
<tr>
<td>get</td>
<td>Download a file from the server to your local machine</td>
</tr>
<tr>
<td>help</td>
<td>Give help</td>
</tr>
<tr>
<td>lcd</td>
<td>Change local working directory</td>
</tr>
<tr>
<td>lpwd</td>
<td>Print local working directory</td>
</tr>
<tr>
<td>ls</td>
<td>List remote files</td>
</tr>
<tr>
<td>mget</td>
<td>Download multiple files at once</td>
</tr>
<tr>
<td>mkdir</td>
<td>Create directories on the remote server</td>
</tr>
<tr>
<td>mput</td>
<td>Upload multiple files at once</td>
</tr>
<tr>
<td>mv</td>
<td>Move or rename file(s) on the remote server</td>
</tr>
<tr>
<td>open</td>
<td>Connect to a host</td>
</tr>
<tr>
<td>put</td>
<td>Upload a file from your local machine to the server</td>
</tr>
<tr>
<td>pwd</td>
<td>Print your remote working directory</td>
</tr>
<tr>
<td>quit</td>
<td>Finish your SFTP session</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>reget</td>
<td>Continue downloading files</td>
</tr>
<tr>
<td>ren</td>
<td>Move or rename file(s) on the remote server</td>
</tr>
<tr>
<td>reput</td>
<td>Continue uploading files</td>
</tr>
<tr>
<td>rm</td>
<td>Delete files on the remote server</td>
</tr>
<tr>
<td>rmdir</td>
<td>Remove directories on the remote server</td>
</tr>
</tbody>
</table>

Next steps

To upload and download files into Amazon Simple Storage Service using SFTP, you could use M2SFTP in conjunction with the AWS Transfer Family, as described in the following blog posts.

- [Using AWS SFTP logical directories to build a simple data distribution service](#)
- [Enable password authentication for AWS Transfer for SFTP using AWS Secrets Manager](#)

M2WAIT Batch Utility

M2WAIT is a mainframe utility program that enables you to introduce a wait period in your JCL scripts by specifying a time duration in seconds, minutes, or hours. You can call M2WAIT directly from JCL by passing the time you want to wait as an input parameter. Internally, the M2WAIT program calls the Micro Focus supplied module C$SLEEP to wait for a specified time.

**Note**

You can use Micro Focus aliases to replace what you have in your JCL scripts. For more information, see [JES Alias](#) in the Micro Focus documentation.

Topics

- [Supported Platforms (p. 298)](#)
- [Configure M2WAIT for AWS Mainframe Modernization Managed (p. 298)](#)
- [Configure M2WAIT for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0) (p. 299)](#)
- [Sample JCL (p. 299)](#)

Supported Platforms

You can use M2WAIT on any of the following platforms:

- AWS Mainframe Modernization Micro Focus Managed
- Micro Focus Runtime (on Amazon EC2)
- All variants of Micro Focus Enterprise Developer (ED) and Micro Focus Enterprise Server (ES) products.

Configure M2WAIT for AWS Mainframe Modernization Managed

If your migrated applications are running on AWS Mainframe Modernization Managed, you will need to configure M2WAIT as follows.

- Use the program M2WAIT in your JCL by passing input parameter as shown in [Sample JCL (p. 299)](#).
Configure M2WAIT for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)

If your migrated applications are running on AWS Mainframe Modernization runtime on Amazon EC2, configure M2WAIT as follows.

1. Change the **Micro Focus JES Program Path** to include the binary location for batch utilities. If you need to specify multiple paths, use colons (:) to separate paths on Linux and semicolons (;) on Windows.
   - Linux: `/opt/aws/m2/microfocus/utilities/64bit`
   - Windows (32bit): `C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows (64bit): `C:\AWS\M2\MicroFocus\Utilities\64bit`

2. Use the program M2WAIT in your JCL by passing the input parameter as shown in Sample JCL (p. 299).

Sample JCL

To test the installation, you can use the M2WAIT1.jcl program.

This sample JCL shows how to call M2WAIT and pass it several different durations.

```
//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 Utility

TXT2PDF is a mainframe utility program commonly used to convert TEXT files to a PDF file. This utility uses the same source code for TXT2PDF (z/OS freeware). We modified it to run under the AWS Mainframe Modernization Micro Focus runtime environment.
Supported Platforms

You can use TXT2PDF on any of the following platforms:

- AWS Mainframe Modernization Micro Focus Managed
- Micro Focus Runtime (on Amazon EC2)
- All variants of Micro Focus Enterprise Developer (ED) and Micro Focus Enterprise Server (ES) products.

Configure TXT2PDF for AWS Mainframe Modernization Managed

If your migrated applications are running on AWS Mainframe Modernization Managed, configure TXT2PDF as follows.

- Create a REXX EXEC library called AWS.M2.REXX.EXEC. Download these REXX modules and copy them into the library.
  - TXT2PDF.rex - TXT2PDF z/OS freeware (modified)
  - TXT2PDFFD.rex - TXT2PDF z/OS freeware (unmodified)
  - TXT2PDFX.rex - TXT2PDF z/OS freeware (modified)
  - M2GETOS.rex - To check the OS type (Windows or Linux)

To test your configuration, see Sample JCL (p. 301).

Configure TXT2PDF for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)

If your migrated applications are running on AWS Mainframe Modernization runtime on Amazon EC2, configure TXT2PDF as follows.

1. Set the Micro Focus environment variable MREXX_CHARSET to the appropriate value, such as "A" for ASCII data.
   
   **Important**
   
   Entering the wrong value could cause data conversion issues (from EBCDIC to ASCII), making the resulting PDF unreadable or inoperable. We recommend setting MREXX_CHARSET to match MF_CHARSET.

2. Change the Micro Focus JES Program Path to include the binary location for batch utilities. If you need to specify multiple paths, use colons (:) to separate paths on Linux and semicolons (;) on Windows.
   
   - Linux: /opt/aws/m2/microfocus/utilities/64bit
3. Create a REXX EXEC library called `AWS.M2.REXX.EXEC`. Download these [REXX modules](#) and copy them into the library.

- `TXT2PDF.rex` - TXT2PDF z/OS freeware (modified)
- `TXT2PDFD.rex` - TXT2PDF z/OS freeware (unmodified)
- `TXT2PDFX.rex` - TXT2PDF z/OS freeware (modified)
- `M2GETOS.rex` - To check the OS type (Windows or Linux)

To test your configuration, see [Sample JCL (p. 301)](#).

**Sample JCL**

To test the installation, you can use either of the following sample JCL files.

**TXT2PDF1.jcl**

This sample JCL file uses a DD name for the TXT2PDF conversion.

```plaintext
//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 *
//THIS IS THE FIRST LINE ON THE PAGE 1
@THIS 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
//THIS IS THE FIRST LINE ON THE PAGE 2
@THIS 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,
```
TXT2PDF2.jcl

This sample JCL uses a DSN name for the TXT2PDF conversion.

//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=SHR
//*
//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

302
Modifications

To make the TXT2PDF program run on the AWS Mainframe Modernization Micro Focus runtime environment, we made the following changes:

- Changes to the source code to ensure compatibility with the Micro Focus REXX runtime
- Changes to ensure that the program can run on both Windows and Linux operating systems
- Modifications to support both EBCDIC and ASCII runtime

References

TXT2PDF references and source code:

- Text to PDF converter
- z/OS Freeware TCP/IP and Mail Tools
- TXT2PDF User Reference Guide
Applications in AWS Mainframe Modernization

If you're new to AWS Mainframe Modernization see the following topics to get started:

- What is AWS Mainframe Modernization? (p. 1)
- Setting up AWS Mainframe Modernization (p. 4)
- Tutorial: Managed Runtime for AWS Blu Age (p. 6)
- Tutorial: Managed runtime for Micro Focus (p. 15)

An application in AWS Mainframe Modernization contains a migrated mainframe workload. The application is analogous to a workload on the mainframe and is associated with a runtime environment. You can add batch files and data sets to applications and monitor applications as they run. You create AWS Mainframe Modernization applications for each workload that you migrate. When you create an AWS Mainframe Modernization application, you specify the engine that the application runs on when you create it. Choose AWS Blu Age if you are using the automated refactoring pattern, and choose Micro Focus if you are using the replatforming pattern.

Topics

- Create an AWS Mainframe Modernization application (p. 304)
- Deploy an AWS Mainframe Modernization application (p. 305)
- Update an AWS Mainframe Modernization application (p. 306)
- Delete an AWS Mainframe Modernization application from an environment (p. 307)
- Delete an AWS Mainframe Modernization application (p. 307)
- Submit batch jobs for AWS Mainframe Modernization applications (p. 308)
- Import data sets for AWS Mainframe Modernization applications (p. 308)
- Manage transactions for AWS Mainframe Modernization applications (p. 309)
- Create AWS resources for a migrated application (p. 310)
- Configure the managed application (p. 312)
- AWS Mainframe Modernization application definition reference (p. 334)
- AWS Mainframe Modernization data set definition reference (p. 343)

Create an AWS Mainframe Modernization application

Use the AWS Mainframe Modernization console to create an AWS Mainframe Modernization application.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4).

Create an application

To create an application

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where you want to create the application.
3. On the Applications page, choose Create application.
4. On the Specify basic information page, in the Name and description section, enter a name for the application.
5. (Optional) In the Application description field, enter a description for the application. This description can help you and other users identify the purpose of the application.
6. In the Engine type section, choose Blu Age for automated refactoring, or Micro Focus for replatforming.
7. In the KMS key section, choose Customize encryption settings if you want to use a customer managed AWS KMS key. For more information, see Data encryption at rest for AWS Mainframe Modernization service (p. 361).

   **Note**
   By default, AWS Mainframe Modernization encrypts your data with a AWS KMS key that AWS Mainframe Modernization owns and manages for you. However, you can choose to use a customer managed AWS KMS key.

8. (Optional) Choose an AWS KMS key by name or Amazon Resource Name (ARN), or choose Create an AWS KMS key to go to the AWS KMS console and create a new AWS KMS key.
9. (Optional) In the Tags section, choose Add new tag to add one or more application tags to your application. An application tag is a custom attribute label that helps you organize and manage your AWS resources.
10. Choose Next.
11. In the Resources and configurations section, use the inline editor to enter the application definition. Alternatively, choose Use an application definition JSON file in an Amazon S3 bucket and provide the location of the application definition that you want to use. For more information, see AWS Blu Age application definition sample (p. 335) or Micro Focus application definition (p. 339).
12. Choose Next.
13. On the Review and create page, review the information that you entered, and then choose Create application.

### Deploy an AWS Mainframe Modernization application

Use the AWS Mainframe Modernization console to deploy an AWS Mainframe Modernization application.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4).

### Deploy an application

To run an AWS Mainframe Modernization application, you must first deploy it to a runtime environment. An application can have more than one version. Each version of an application has its own application definition. To deploy an application, you must specify the version that you want to deploy.

You can deploy only one version of a given application at a time. If you deploy a version of an application, then decide to deploy a different version instead, you must first stop the application if it is running.

**To deploy an application**

1. Open the AWS Mainframe Modernization console at [https://console.aws.amazon.com/m2/](https://console.aws.amazon.com/m2/).
Update an application

An AWS Mainframe Modernization application can have multiple versions, each with its own application definition. To update an application, provide a new application definition. This creates a new version of the application.

To update an application

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where the application that you want to update was created.
3. On the Applications page, choose the application that you want to update.
4. On the application details page, in the Current definition section, choose Edit to update the current application definition.
5. On the Update application page, use the inline editor to update the current application definition.

Alternatively, choose Use an application definition JSON file in an Amazon S3 bucket and provide the location of the application definition that you want to use. For more information, see AWS Blu Age application definition sample (p. 335) or Micro Focus application definition (p. 339).

6. When you're finished updating the application definition, choose Update.
Delete an AWS Mainframe Modernization application from an environment

You can delete an AWS Mainframe Modernization application from an environment using the AWS Mainframe Modernization console.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4).

Delete an application from an environment

If you need to delete an AWS Mainframe Modernization application, and it is running, make sure that you stop it first. You can see the application status on the Applications page.

To delete an application from an environment

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where the application that you want to delete from the environment was created.
3. On the Applications page, choose the application that you want to delete from the environment, and then choose Actions.
4. (Optional) If the status of the application is Running, choose Stop application.
5. Choose Delete from environment.
   The delete process starts immediately.

Delete an AWS Mainframe Modernization application

Use the AWS Mainframe Modernization console to delete an AWS Mainframe Modernization application.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4).

Delete an application

If you need to delete an AWS Mainframe Modernization application, and it is running, make sure that you stop it first. You can see the application status on the Applications page.

To delete an application

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where the application that you want to delete was created.
3. On the **Applications** page, choose the application that you want to delete, and then choose **Actions**.
4. (Optional) If the status of the application is **Running**, choose **Stop application**.
5. Choose **Delete application**.
6. In the **Delete application** window, enter `delete` to confirm that you want to delete the application, and then choose **Delete**.

## Submit batch jobs for AWS Mainframe Modernization applications

In AWS Mainframe Modernization you can submit batch jobs for your applications. You can submit or cancel batch jobs and review details about batch job executions. Each time that you submit a batch job, AWS Mainframe Modernization creates a separate batch job execution. You can monitor this job execution. You can search for batch jobs by name and supply JCL or script files to batch jobs.

**Important**

If you cancel a batch job, this doesn't delete the job. It cancels a particular execution of the batch job. The batch job records remain available for you to view in the details for the batch job execution.

If your batch job requires access to one or more data sets, use the AWS Mainframe Modernization console or the AWS Command Line Interface (AWS CLI) to import the data sets. For more information, see [Import data sets for AWS Mainframe Modernization applications](#).

These instructions assume that you have completed the steps in [Setting up AWS Mainframe Modernization](#) and in [Create an AWS Mainframe Modernization application](#).

### Submit a batch job

**To submit a batch job**

1. Open the AWS Mainframe Modernization console at [https://console.aws.amazon.com/m2/](https://console.aws.amazon.com/m2/).
2. In the AWS Region selector, choose the Region where the application that you want to submit a batch job for was created.
3. On the **Applications** page, choose the application that you want to submit a batch job for.
   
   **Note**
   
   Before you can submit a batch job to an application, you must deploy the application successfully.
4. On the application details page, choose **Batch jobs**.
5. Choose **Submit job**.
6. In the **Select a script** section, choose a script. You can search for the script that you want by name.
7. Choose **Submit job**.

### Import data sets for AWS Mainframe Modernization applications

With AWS Mainframe Modernization you can import data sets to use with your applications. You can specify the data sets in a JSON file stored in an Amazon S3 bucket, or you can specify data set
configuration values separately. After you import the data sets, you can review the details of the import task to confirm that the data sets that you wanted were imported. All cataloged data sets for an application are listed together in the console.

Use the AWS Mainframe Modernization console to import data sets for a AWS Mainframe Modernization application.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4) and in Create an AWS Mainframe Modernization application (p. 304).

**Import a data set**

**To import a data set**

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where the application that you want to import data sets for was created.
3. On the Applications page, choose the application that you want to import data sets for.
4. On the application details page, choose Data sets.
5. Choose Import.
6. Do one of the following:
   - Choose Use data set configuration JSON file in an Amazon S3 bucket and provide the location of the data set configuration.
   - Choose Specify the data set configuration values separately with guided configuration. Refer the section called “Data set definition reference” (p. 343) for specific definition details.
   
     Enter the name, data set organization (VSAM, GDG, PO, PS), location, and external Amazon S3 location, and parameter settings for each data set configuration value. In guided configuration you can also choose Generate JSON to review JSON configuration from your input.
7. Choose Submit.

**Manage transactions for AWS Mainframe Modernization applications**

With AWS Mainframe Modernization you can run an application, by request, at the same time as many other users who submit requests to run the same application using the same files and programs. A single transaction consists of one or more application programs that carry out the needed processing.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4) and in Create an AWS Mainframe Modernization application (p. 304).

**Manage transactions for applications**

You can display and edit transactions for applications.

**To manage transactions for applications**

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the AWS Region where the application that you want to run was created.
3. On the Applications page, choose the application that you want to manage transactions for.
4. On the **Transactions** tab, under **Transaction resources**, choose how you want your resources displayed from the dropdown list. You can display resources according to transaction resources, groups, lists, or SITs.

- **Transaction resources** allow you to choose the resource type according to file definitions, transaction definitions, program definitions, or transient data queue definitions.
  
  **Note**
  
  The AWS Mainframe Modernization Managed transaction supports more resource types than these, but you can't directly edit them here. Other resources must be edited externally and you need to recreate the application with updated resource entries.

- **Groups** are collection of transaction resources. You can also choose groups that you want to associate with your transaction resource.

- **Lists** are ordered collection of groups. You can see all your transaction resources and groups in a list view. The **startup list** determines which resources are loaded when the server is initialized.
  
  - With Blu Age refactor engine, you specify the lists to be included at the startup and there is no limit to number of lists.
  
  - With Micro Focus replatform engine, you can specify up to four lists in one SIT.

- **SIT (System Initialization Table)** displays all available transaction configurations. You can find SITs according to properties (name, description, and startup lists). You can also choose lists to associate with your chosen SIT.
  
  **Note**
  
  SITs are only applicable for the Micro Focus replatform engine.

5. Choose a transaction resource to display all the resource information. You can also view all attributes associated with your transaction resource and view or edit any additional attributes.

6. If you want to edit any information related to your current transaction resource, choose **Edit**.

   a. On the **Edit** page, you can add or change resource description.

      1. For transaction resources displayed in lists, you can also add, remove, or reorder lists as desired.

      2. For specific resource types (file definitions, transaction definitions, program definitions, and transient data queue definitions), you can edit the individual resource properties. These properties vary by engine and resource type.

   b. After making the desired changes, choose **Save changes**.

      You see a message when the resource is updated successfully.

---

**Create AWS resources for a migrated application**

In order to run your migrated application in AWS, you must create some AWS resources with other AWS services. The resources you must create include the following:

- An S3 bucket to hold application code, configuration, data files, and other required artifacts.
- An Amazon RDS or Amazon Aurora database to hold the data that the application requires.
- An AWS KMS key, which is required by AWS Secrets Manager to create and store secrets.
- A Secrets Manager secret to hold the database credentials.

**Note**

Each migrated application requires its own set of these resources. This is a minimum set. Your application might also require additional resources, such as Amazon Cognito secrets or MQ queues.
**Required permissions**

Make sure that you have the following permissions:

- `s3:CreateBucket`, `s3:PutObject`
- `rds:CreateDBInstance`
- `kms:CreateKey`
- `secretsmanager:CreateSecret`

**Amazon S3 bucket**

Both refactored and replatformed applications require an S3 bucket that you configure as follows:

```
bucket-name/root-folder-name/application-name
```

- **bucket-name**
  
  Any name within the constraints of Amazon S3 naming. We recommend that you include the AWS Region name as part of your bucket name. Make sure that you create the bucket in the same Region where you plan to deploy the migrated application.

- **root-folder-name**
  
  Name required to satisfy constraints in the application definition, which you create as part of the AWS Mainframe Modernization application. You can use the `root-folder-name` to distinguish between different versions of an application, for example, V1 and V2.

- **application-name**
  
  The name of your migrated application, for example, PlanetsDemo or BankDemo.

**Database**

Both refactored and replatformed applications might require a database. You must create, configure, and manage the database according to specific requirements for each runtime engine. AWS Mainframe Modernization supports encryption in transit on this database. If you enable SSL on your database, make sure that you specify `sslMode` in the database secret along with the connection details of the database. For more information, see [AWS Secrets Manager secret](p. 312).

If you use the AWS Blu Age refactoring pattern, and you need a BluSam database, the AWS Blu Age runtime engine expects an Amazon Aurora PostgreSQL database, which you must create, configure, and manage. The BluSam database is optional. Create this database only if your application requires it. To create the database, follow the steps in [Creating an Amazon Aurora DB cluster](in the Amazon Aurora User Guide).

If you are using the Micro Focus replatforming pattern, you can create either an Amazon RDS or an Amazon Aurora PostgreSQL database. To create the database, follow the steps in [Creating an Amazon RDS DB instance](in the Amazon RDS User Guide) or in [Creating an Amazon Aurora DB cluster](in the Amazon Aurora User Guide).

For both runtime engines, you must store the database credentials in AWS Secrets Manager using an AWS KMS key to encrypt them.
AWS Key Management Service key

You must store the credentials for the application database securely in AWS Secrets Manager. To create a secret in Secrets Manager, you must create an AWS KMS key. To create an KMS key, follow the steps in Creating keys in the AWS Key Management Service Developer Guide.

After you create the key, you must update the key policy to grant AWS Mainframe Modernization decrypt permissions. Add the following policy statements:

```json
{
    "Effect": "Allow",
    "Principal": {
        "Service": "m2.amazonaws.com"
    },
    "Action": "kms:Decrypt",
    "Resource": "*"
}
```

AWS Secrets Manager secret

You must store the credentials for the application database securely in AWS Secrets Manager. To create a secret follow the steps in Create a database secret in the AWS Secrets Manager User Guide.

AWS Mainframe Modernization supports encryption in transit on this database. If you enable SSL on your database, make sure that you specify sslMode in the database secret along with the connection details of the database. You can specify one of the following values for sslMode: verify-full, verify-ca, or disable.

During the key creation process, choose Resource permissions - optional, and then choose Edit permissions. In the policy editor, add a resource-based policy, such as the following, to retrieve the content of the encrypted fields.

```json
{
    "Effect": "Allow",
    "Principal": {
        "Service": "m2.amazonaws.com"
    },
    "Action": "secretsmanager:GetSecretValue",
    "Resource": "*"
}
```

Configure the managed application

You can configure your application to include access to legacy utilities. You can customize additional properties as well. In order to understand what you can configure and where, it helps to understand the overall structure of an AWS Blu Age modernized application.

Topics
- Structure of AWS Blu Age managed applications (p. 313)
- Configuring access to utilities for managed applications (p. 314)
- Add configuration properties for the AWS Blu Age engine (p. 320)
Structure of AWS Blu Age managed applications

If you use the AWS Blu Age refactoring pattern, the AWS Blu Age runtime engine expects the following structure inside the application-name folder in your S3 bucket:

- **config**
  - Contains the YAML files for your project. These are the YAML files specific to your application, typically named something like `application-planetsdemo.yaml` and not the `application-main.yaml` file that AWS Mainframe Modernization supplies and sets up automatically for you.

- **webapps**
  - Contains the WAR files for your application. Those files are an output of the modernization process.

An application can also have the following optional folders:

- **jics/sql**
  - Contains the `initJics.sql` script that initializes the JICS database for your application.

- **scripts**
  - Contains application scripts, which you can also supply directly inside the WAR files.

- **sql**
  - Contains application SQL files, which you can also supply directly inside the WAR files.

- **lnk**
  - Contains application LNK files, which you can also supply directly inside the WAR files.

Managing an application's Java memory consumption

To manage the Java memory consumption for the application, add a properties file named `tomcat.properties` to the application-name folder. This file can have two properties: `xms`, which
specifies the minimum Java memory consumption, and xmx, which specifies the maximum Java memory consumption. The following is an example of the contents of a valid `tomcat.properties` file.

```
xms=512M
xmx=1G
```

The values that you specify for these two properties can be in any of the following units:

- Bytes: don't specify a unit.
- Kilobytes: append a K to the value.
- Megabytes: append an M to the value.
- Gigabytes: append a G to the value.

## Configuring access to utilities for managed applications

When you refactor a mainframe application with AWS Blu Age, you might need to provide support for various legacy platform utility programs, such as IDCAMS, INFUTILB, SORT, and so on, if your application depends on them. AWS Blu Age refactoring provides this access with a dedicated web application that is deployed alongside modernized applications. This web application requires a configuration file, `application-utility-pgm.yml`, that you must provide. If you don't provide this configuration file, the web application can't deploy alongside your application and won't be available.

### Topics

- [Configuration properties (p. 315)](#)

This topic describes all the possible properties that you can specify in the `application-utility-pgm.yml` configuration file, along with their defaults. The topic describes both required and optional properties. The following example is a complete configuration file. It lists properties in the order that we recommend. You can use this example as a starting point for your own configuration file.

```
# 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
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.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

Configuration properties

You can specify the following properties in your configuration file.

**spring.jta.enabled**

Optional. Controls whether JTA support is enabled. For utilities, we recommend that you set this value to false.

```
spring.jta.enabled : false
```

**logging.config**

Required. Specifies the path to the dedicated logger configuration file. We recommend that you use the name logback-utility.xml and provide this file as part of the modernized application. The common way to organize these files is to put all logger configuration files in the same place, usually in the subfolder /config/logback where /config is the folder that contains yaml configuration files. For more information, see [Chapter 3: Logback configuration](#) in the Logback documentation.

```
logging.config : classpath:logback-utility.xml
```

**encoding**

Required. Specifies the character set that the utility program uses. For most cases, when you migrate from z/OS platforms, this character set is an EBCDIC variant, and should match the character set that is configured for the modernized applications. Default if not set is ASCII.

```
encoding : cp1047
```
sysPunchEncoding

Optional. Specifies the character set that INFUTILB and DSNUTILB use to generate and read SYSPUNCH files. If you use the SYSPUNCH files from the legacy platform as they are, this value should be an EBCDIC variant. Default if not set is ASCII.

```text
sysPunchEncoding : cp1047
```

Primary data source configuration

Some database-related utilities, such as LOAD and UNLOAD, require access to a target database through a data source. Like other data source definitions within AWS Mainframe Modernization, this access requires that you use AWS Secrets Manager. The properties that point to the proper secrets in Secrets Manager are as follows:

**spring.aws.client.datasources.primary.secret**

Optional. Specifies the secret in Secrets Manager that contains the data source properties.

```text
spring.aws.client.datasources.primary.secret: datasource-secret-ARN
```

**spring.aws.client.datasources.primary.dbname**

Optional. Specifies the target database name if the database name isn't provided directly in the database secret, with the dbname property.

```text
spring.aws.client.datasources.primary.dbname: target-database-name
```

**treatLargeNumberAsInteger**

Optional. Related to Oracle database engine specifics and DSNTEP2/DSNTEP4 utilities usage. If you set this flag to true, large numbers coming from the Oracle database (NUMBER (38,0)) are treated as integers. Default: false

```text
treatLargeNumberAsInteger : false
```

**zonedMode**

Optional. Sets the zoned mode to encode or decode zoned data types. This setting influences the way sign digits are represented. The following values are valid:

- **EBCDIC_STRICT**: Default. Use strict definition for signs handling. Depending on whether the character set is EBCDIC or ASCII, the sign digit representation uses the following characters:
  - EBCDIC characters that correspond to bytes (Cn+Dn) to represent positive and negative digit ranges (+0 to +9, -0 to -9). The characters are displayed as {A to I }, J to R
  - ASCII characters that correspond to bytes (3n+7n) to represent positive and negative digit ranges (+0 to +9, -0 to -9). The characters are displayed as 0 to 9, p to y
- **EBCDIC_MODIFIED**: Use a modified definition for signs handling. For both EBDIC and ASCII, the same list of characters represent the sign digits, that is, +0 to +9 mapped to { + A to I and -0 to -9 mapped to } + J to R.
- **AS400**: Use for modernized legacy assets that come from iSeries (AS400) platforms.

```text
zonedMode:EBCDIC_STRICT
```
jcl.type

Optional. Indicates the legacy type of modernized JCL scripts. The IDCAMS utility uses this setting to tailor the return code if the invoking JCL is of type vse. Valid values are as follows:

- mvs (Default)
- vse

jcl.type : mvs

Database Unload utilities related properties

Use these properties to configure utilities that unload database tables to data sets. All of the following properties are optional.

This example shows all the possible unload properties.

```plaintext
# 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
```

sqlCodePointShift

Optional. Specifies an integer value that represents the SQL code point shift used on data. The default is 0. This means that no code point shifting is made. Align this setting with the SQL code point shift parameter used for modernized applications. When code point shifting is in use, the most common value for this parameter is 384.

```plaintext
unload.sqlCodePointShift: 0
```

nbi

Optional. Specifies a null indicator byte. This is a hexadecimal value (as a string) added to the right of the data value. The two possible values are as follows:

- **whenNull**: Add the hexadecimal value when the data value is null. Default is 6`. Sometimes the high value FF is used instead.

```plaintext
unload.nbi.whenNull: "6F"
```

- **whenNotNull**: Add the hexadecimal value when the data value is not null, but the column is nullable. Default is 00 (low value).
unload.nbi.whenNotNull: "00"

useDatabaseConfiguration

Optional. Specifies date and time formatting properties. This is used to deal with date/time objects in UNLOAD queries. Default is false.

- If set to true, uses the pgmDateFormat, pgmTimeFormat, and pgmTimestampFormat properties from the main configuration file (application-main.yml).
- If set to false, uses the following date and time formatting properties:
  - unload.format.date: Specifies a date formatting pattern. Default is MM/dd/yyyy.
  - unload.format.time: Specifies a time formatting pattern. Default is HH.mm.ss.
  - unload.format.timestamp: Specifies a timestamp formatting pattern. Default is yyyy-MM-dd-HH.mm.ss.SSSSSS.

chunkSize

Optional. Specifies the size of data chunks used to create SYSREC data sets. These data sets are the target of the data set unload operation, with parallel operations. Default is 0 (no chunks).

unload.chunkSize: 0

fetchSize

Optional. Specifies the data fetch size. The value is the number of records to fetch at one time when a data chunks strategy is used. Default: 0.

unload.fetchSize: 0

varCharIsNull

Optional. Specifies how to handle a non nullable varchar column with blank content. Default is false.

If you set this value to true, the column content is treated as an empty string for unload purposes, instead of a single space string. Set this flag to true for the Oracle database engine case only.

unload.varCharIsNull: false

columnFiller

Optional. Specifies the value to use for padding unloaded columns in varchar columns. Possible values are space or low values. Default is space.

unload.columnFiller: space

Database Load related properties

Use these properties to configure utilities that load data set records into a target database, for example, DSNUTILB. All of the following properties are optional.

This example shows all of the possible load properties.
# Batch size for DSNUTILB Load Task

```yaml
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
```

**sqlCodePointShift**

Optional. Specifies an integer value that represents the SQL code point shift that is used on data. Defaults to 0, which means that applications make no code point shifting. Align this setting with the SQL code point shift parameter used for modernized applications. When you use code point shifts, the most common value for this parameter is 384.

```yaml
load.sqlCodePointShift : 384
```

**batchSize**

Optional. Specifies an integer value that represents the number of records to treat before you send an actual batch statement to the database. Defaults to 0.

```yaml
load.batchSize: 500
```

**format**

Optional. Specifies the date and time formatting patterns to use for date/time conversions during the database load operations.

- `load.format.localDate`: Local date formatting pattern. This defaults to `dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd`.
- `load.format.dbDate`: Database date formatting pattern. This defaults to `yyyy-MM-dd`.
- `load.format.localTime`: Local time formatting pattern. This defaults to `HH:mm:ss|HH.mm.ss`.
- `load.format.dbTime`: Database time formatting pattern. This defaults to `HH:mm:ss`.

**table-mappings**

Optional. Specifies a collection of customer-provided mappings between legacy and modern table names. The DSNUTILB utility program consumes these mappings.

Specify the values in the following format: `MODERN_TABLE_NAME : LEGACY_TABLE_NAME`

Here is an example:

```yaml
table-mappings:
  TABLE_1_NAME : LEGACY_TABLE_1_NAME
  TABLE_2_NAME : LEGACY_TABLE_2_NAME
  ...
  TABLE_*N*_NAME : LEGACY_TABLE_*N*_NAME
```
Configure additional properties

Add configuration properties for the AWS Blu Age engine

You can add a file in the config folder for your refactored application that will give you access to new features in the AWS Blu Age runtime engine. You must name this file `user-properties.yml`. This file doesn't replace the application definition but extends it. This topic describes the properties you can include in the `user-properties.yml` file.

**Note**
You can't change some parameters because they are controlled either by AWS Mainframe Modernization or by the application definition. All parameters defined in the application definition for your application have priority over the parameters you specify in `user-properties.yml`.

For more information about the structure of refactored applications, see Structure of AWS Blu Age managed applications (p. 313).

The following diagram shows where to locate the `user-properties.yml` file within the structure of the AWS Blu Age sample application, PlanetsDemo.

```
PlanetsDemo-v1/
## config/
#  ## application-PlanetsDemo.yml
#  ## user-properties.yml
## jics/
## webapps/
```

Configuration properties reference

This is the list of available properties. All parameters are optional.

**Topics**

- Gapwalk application properties (p. 320)
- Gapwalk batchscript properties (p. 323)
- Gapwalk Blugen properties (p. 324)
- Gapwalk CL command properties (p. 324)
- Gapwalk CL runner properties (p. 324)
- Gapwalk JHDB properties (p. 325)
- Gapwalk JICS properties (p. 326)
-Gapwalk runtime properties (p. 328)
- Gapwalk utility program properties (p. 330)
- Other properties (p. 333)

Gapwalk application properties

`bluesam.fileLoading.commitInterval`

Optional. The bluesam commit interval.
**card.encoding**

Optional. Card encoding: to be used with useControlMVVariable.

Type: string
Default: CP1145

**checkinputfilesize**

Optional. Specifies whether to release a check if the file size is a multiple of record size.

Type: boolean
Default: false

**database.cursor.overflow.allowed**

Optional. Specifies whether to allow the cursor overflow. Set to true to perform a next call on the cursor whatever its position. Set to false to check whether the cursor is at the last position before performing a next call on cursor. Only enable if cursor is SCROLLABLE (SENSITIVE or INSENSITIVE)

Type: boolean
Default: true

**dataSimplifier.onInvalidNumericData**

Optional. How to react when decoding invalid numeric data. Allowed values are reject, toleratespaces, toleratespaceslowvalues, toleratemost.

Type: string
Default: reject

**defaultKeepExistingFiles**

Optional. Specifies whether to set the dataset default previous value.

Type: boolean
Default: false

**disposition.checkexistence**

Optional. Specifies whether to release a check on file existence for Dataset with DISP SHR or OLD.

Type: boolean
Default: false

**externalSort.threshold**

Optional. The sort threshold: when to switch to external (merge) sort.

Type: string
Default: null

`externalSort.threshold: 12MB`
forceHR

Optional. Specifies whether to use Human Readable SYSPRINT, either on console or file output.

Type: boolean

Default: false

forcedDate

Optional. Forces a specific date and time in the database. Use only during development and testing.

Default: null

forcedDate: 2022-08-26T12:59:58.123456+01:57

frozenDate

Optional. Freezes the date and time in the database. Use only during development and testing.

Default: false

frozenDate: false

ims.messages.extendedSize

Optional. Specifies whether to set the extendedSize on ims messages.

Type: boolean

Default: false

lockTimeout

Optional. The timeout in milliseconds of a transaction when unable to acquire a lock within a specified timeframe.

Type: number

Default: 500

mapTransfo.prefixes

Optional. List of prefixes to be used when transforming controlM variables. Each one separated by comma.

Type: string

Default: &,@,%%

query.useConcatCondition

Optional. Specifies whether key condition is built by key concatenation or not.

Type: boolean

Default: false

rollbackOnRTE

Optional. Specifies whether to rollback implicit run unit transaction on runtime exceptions.

Type: boolean

Default: false
sctThreadLimit

  Optional. The thread limit for triggering scripts.

  Type: number

  Default: 5

sqlCodePointShift

  Optional. The sql code point shift. Shifts the codepoint for control characters that we might
  encounter when migrating legacy rdbms data to a modern rdbms. For example, you could specify
  384 to match unicode character \u0180.

  Type: number

  Default: 0

sqlIntegerOverflowAllowed

  Optional. Specifies whether to allow the SQL integer overflow, meaning whether placing larger
  values in the host variable is allowed.

  Type: boolean

  Default: false

stepFailWhenAbend

  Optional. Specifies whether to raise an abend if a step fails or completes execution.

  Type: boolean

  Default: true

stopExecutionWhenProgNotFound

  Optional. Specifies whether to stop running if a program isn't found. If set to true, interrupts the
  run if a program is not found.

  Type: boolean

  Default: true

uppercaseUserInput

  Optional. Specifies whether user input must be in uppercase.

  Type: boolean

  Default: true

useControlMVariable

  Optional. Specifies whether to use control-M specification for variable replacement.

  Type: boolean

  Default: false

Gapwalk batchscript properties

encoding

  Optional. The encoding used in batchscript projects (not with groovy). Expects a valid encoding
  CP1047, IBM930, ASCII, UTF-8...
Configure additional properties

Type: string
Default: ASCII

**Gapwalk Blugen properties**

**managers.trancode**

Optional. The dialog manager trancode mapping. Allows you to map a JICS transaction code to a dialog manager. Expected format is trancode1:dialogManager1;trancode2:dialogManager2;

Type: string
Default: null

managers.trancode: OR12:MYDIALOG1

**Gapwalk CL command properties**

**commands-off**

Optional. List of commands to turn off, separated by comma. Allowed values are PGM_BASIC, RCVMSG, SNDRCVF, CHGVAR, QCLRDTAQ, RTVJOBA, ADDLFM, ADDPFM, RCVF, OVRDBF, DLTOVR, CPYF, SNDDTAQ. Useful when you want to disable or overwrite an existing program. PGM_BASIC is a specific velocity program designed for debug purpose.

Type: string
Default: null

**spring.datasource.primary.jndi-name**

Optional. The primary Java Naming And Directory Interface (jndi) datasource.

Type: string
Default: jdbc/primary

**zonedMode**

Optional. The mode for encoding or decoding zoned data types. Allowed values are EBCDIC_STRICT / EBCDIC_MODIFIED / AS400.

Type: string
Default: EBCDIC_STRICT

**Gapwalk CL runner properties**

**cl.configuration.context.encoding**

Optional. The encoding of CL files. Expects a valid encoding CP1047, IBM930, ASCII, UTF-8...

Type: string
Default: CP297
**cl.zonedMode**

Optional. The mode for encoding or decoding control language (CL) commands. Allowed values are `EBCDIC_STRICT` / `EBCDIC_MODIFIED` / `AS400`.

Type: string
Default: EBCDIC_STRICT

**Gapwalk JHDB properties**

**ims.programs**

Optional. List of IMS programs to use. Separate each parameter with a semicolon (;) and each transaction with a comma (,). For example: `ims.programs: PCP008,PCT008;PCP054,PCT054;PCP066,PCT066;PCP068,PCT068;`

Type: string
Default: null

**jhdb.checkpointPath**

Optional. If `jhdb.checkpointPersistence` is not `none` then this parameter allows you to set up the checkpoint persistence path (checkpoint.dat file storage location), all the checkpoints data contained in the registry are serialized and backed up in a file (checkpoint.dat) located in provided folder. Note that only checkpoint data (scriptId, stepId, database position and checkpoint area) are concerned by this backup.

Type: string
Default: file:./setup/

**jhdb.checkpointPersistence**

Optional. The checkpoint persistence mode. Allowed values are `none` / `add` / `end`. Use `add` to persist checkpoints when a new one is created and added to the registry. Use `end` to persist checkpoint at server shutdown. Any other values disable the persistence. Note that each time a new checkpoint is added to the registry, all the existing checkpoints will be serialized and the file will be erased. It is not an append to the existing data in the file. So depending on the number of checkpoints, it can have some effect on performance.

Type: string
Default: none

**jhdb.configuration.context.encoding**

Optional. The JHDB (Java Hierarchical Database) encoding. Expects a valid encoding string `CP1047`, `IBM930`, `ASCII`, `UTF-8`...

Type: string
Default: CP297

**jhdb.identificationCardData**

Optional. Used to hardcode some "operator identification card data" to the MID field designated by the CARD parameter.

Type: string
Configure additional properties

**Default:** ""

**jhdb.lterm**

Optional. Allow you to force a common logical terminal ID in the case of an IMS emulation. If not set then sessionId is used.

Type: string

Default: null

**jhdb.metadata.extrapath**

A configuration parameter that specifies an extra, runtime-specific root folder for psbs and dbds folders.

Type: string

Default: file:./setup/

**jhdb.navigation.cachenexts**

Optional. The cache duration (in milliseconds) used in hierarchical navigation for an RDBMS.

Type: number

Default: 5000

**jhdb.query.limitJoinUsage**

Optional. Specifies whether to use the limit join usage parameter on RDBMS graphs.

Type: boolean

Default: true

**jhdb.use-db-prefix**

Optional. Specifies whether to enable a database prefix in hierarchical navigation for an RDBMS.

Type: boolean

Default: true

**Gapwalk JICS properties**

**jics.data.dataJsonInitLocation**

Optional. Location of the json file prepared by the Analyzer from parsing CSD, and used to initialize the jics database,

Type: string

Default: ""

**jics.db.dataScriptLocation**

Optional. Location of the initJics.sql script, prepared by Analyzer from parsing CSD exports from the mainframe.

Type: string

Default: ""
jics.db.dataTestQueryLocation

Optional. Location of a sql script containing a single sql query that is expected to return a count of objects (for example: counting number of records in the jics program table). If the count equals 0, database will be loaded using the jics.db.dataScriptLocation script, otherwise database load will be skipped.

Type: string
Default: ""

jics.db.ddlScriptLocation

Optional. The Jics ddl script location. Allows you to initiate the jics database schema using a .sql script.

Type: string
Default: ""

jics.db.ddlScriptLocation: ./jics/sql/jics.sql

jics.db.schemaTestQueryLocation

Optional. Location of the sql file that should contain a unique query that returns the number of objects in the jics schema (if any).

Type: string
Default: ""

jics.runUnitLauncherPool.enable

Optional. Specifies whether to activate the run unit launcher pool in JICS.

Type: boolean
Default: false

jics.runUnitLauncherPool.size

Optional. The run unit launcher pool size in JICS.

Type: number
Default: 20

jics.runUnitLauncherPool.validationInterval

Optional: The validation interval of the run unit launcher pool in JICS, expressed in milliseconds.

Type: number
Default: 1000

jics.queues.sqs.region

Optional. The AWS Region for Amazon SQS, used in JICS. It is advised to be set the same region of the deployed application for performance, but it is not mandatory.

Type: string
Default: eu-west-1
Configure additional properties

**jics.xa.agent.timeout**

Optional. Defines the maximum duration for the xa agent responsible for managing distributed transactions, to complete its operations.

Type: number
Default: null

**mqqueues.sqs.region**

Optional. The AWS Region for the Amazon SQS MQ service.

Type: string
Default: eu-west-3

**taskExecutor.allowCoreThreadTimeOut**

Optional. Specifies whether to allow core threads to time out in JCIS. This enables dynamic growing and shrinking even in combination with a non-zero queue (since the max pool size will only grow once the queue is full).

Type: boolean
Default: false

**taskExecutor.corePoolSize**

Optional. When a transaction in a terminal is initiated via a groovy script, a new thread is created. Use this parameter to setup the core pool size.

Type: number
Default: 5

**taskExecutor.maxPoolSize**

Optional. When a transaction in a terminal is initiated via a groovy script, a new thread is created. Use this parameter to setup the max pool size (max number of parallel threads).

Type: number
Default: 10

**taskExecutor.queueCapacity**

Optional. When a transaction in a terminal is initiated via a groovy script, a new thread is created. Use this parameter to setup the queue size. (= maximum number of pending transactions when taskExecutor.maxPoolSize is reached)

Type: number
Default: 50

**Gapwalk runtime properties**

**cacheMetadata**

Optional. Specifies whether to cache database metadata.

Type: boolean
check-groovy-file

Optional. Specifies whether to check groovy files content before registering.

Type: boolean

Default: true

databaseStatistics

Optional. Specifies whether to allow SQL builders to collect and display statistics information.

Type: boolean

Default: false

dateTimeFormat

Optional. The dateTimeFormat describes how to spill database date time timestamp type into data simplifier entities. Allowed values are ISO / EUR / USA / LOCAL

Type: string

Default: ISO

dbDateFormat

Optional. The database target date format.

Type: string

Default: yyyy-MM-dd

dbTimeFormat

Optional. The database target time format.

Type: string

Default: HH:mm:ss

dbTimestampFormat

Optional. The database target timestamp format.

Type: string

Default: yyyy-MM-dd HH:mm:ss.SSSSSS

fetchSize

Optional. The fetchSize value for cursors. Use when fetching data using chunks by load/unload utils.

Type: number

Default: 10

forceDisableSQLTrimStringType

Optional. Specifies whether to disable trim of all sql string parameters.

Type: boolean

Default: false
localDateFormat
Optional. List of local date formats. Separate each format with |.
Type: string

localTimeFormat
Optional. List of local time formats. Separate each format with |.
Type: string

localTimestampFormat
Optional. List of local timestamp formats. Separate each format with |.
Type: string
Default:

pgmDateFormat
Optional. The date time format used in the programs.
Type: string
Default: yyyy-MM-dd

pgmTimeFormat
Optional. The time format used for pgm (programs) execution.
Type: string
Default: HH.mm.ss

pgmTimestampFormat
Optional. The timestamp format.
Type: string
Default: yyyy-MM-dd-HH.mm.ss.SSSSSS

Gapwalk utility program properties

jcl.type
Optional. jcl file type. Allowed values are jcl / vse. The IDCAMS utility PRINT/REPRO commands return 4 if the file is empty for non-vse jcl.
Type: string
Default: mvs

listcat.variablelengthpreprocessor.enabled
Optional. Specifies whether to enable the variable length preprocessor for the LISTCAT command.
Type: boolean
Default: false
listcat.variablelengthpreprocessor.type

Optional. The type of objects contained in the listcat file, if you enable listcat.variablelengthpreprocessor.enabled. Allowed values are rdw / bdw.

Type: string
Default: rdw

load.batchSize

Optional. The load utility batch size.

Type: number
Default: 0

load.format.dbDate

Optional. The load utility database format to use.

Type: string
Default: yyyy-MM-dd

load.format.dbTime

Optional. The load utility database time to use.

Type: string
Default: HH:mm:ss

load.format.localDate

Optional. The load utility local date format to use.

Type: string
Default: dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd

load.format.localTime

Optional. The load utility local time format to use.

Type: string
Default: HH:mm:ss|HH.mm:ss

load.sqlCodePointShift

Optional. The SQL codepointshift for load utility. Runs the shifting characters process. Required when your target database from DB2 is Postgresql.

Type: number
Default: 0

sysPunchEncoding

Optional. The syspunch encoding character set. Supported values are Cp1047 / ASCII.

Type: string
Default: ASCII
**treatLargeNumberAsInteger**

Optional. Specifies whether to treat large numbers as Integer. They are treated as BigDecimal by default.

Type: boolean
Default: false

**unload.chunkSize**

Optional. Chunk size used for unload utility.

Type: number
Default: 0

**unload.columnFiller**

Optional. The unload utility column filler.

Type: string
Default: space

**unload.fetchSize**

Optional. Allows you to tune the fetch size when handling cursors in the unload utility.

Type: number
Default: 0

**unload.format.date**

Optional. If unload.useDatabaseConfiguration is enabled, the date format to use in the unload utility. For more information, see `unload.format.date`.

Type: string
Default: MM/dd/yyyy

**unload.format.time**

Optional. If unload.useDatabaseConfiguration is enabled, the time format to use in the unload utility.

Type: string
Default: HH.mm.ss

**unload.format.timestamp**

Optional. If unload.useDatabaseConfiguration is enabled, the timestamp format to use in the unload utility.

Type: string
Default: yyyy-MM-dd-HH.mm.ss.SSSSSS

**unload.nbi.whenNotNull**

Optional. The Null Byte Indicator (nbi) value to add when value from database is not null.

Type: hexadecimal
Configure additional properties

Default: 00

unload.nbi.whenNull

Optional. The Null Byte Indicator (nbi) value to add when value from database is null.

Type: hexadecimal

Default: 6F

unload.nbi.writeNullIndicator

Optional. Specifies whether to write out the null indicator in the unload output file.

Type: boolean

Default: false

unload.sqlCodePointShift

Optional. The SQL code pointshift for unload utility. Runs the shifting characters process. Required when your target database from DB2 is Postgresql.

Type: number

Default: 0

unload.useDatabaseConfiguration

Optional. Specifies whether to use the date or time configuration from application-main.yml in unload utility.

Type: boolean

Default: false

unload.varCharIsNull

Optional. Use this parameter in INFTILB program, if set to true then all not nullable fields with blank (space) values returns an empty string.

Type: boolean

Default: false

Other properties

qtemp.cleanup.threshold.hours

Optional. To specify when qtemp.dblog is enabled. The db partition lifetime (in hours).

Type: number

Default: 0

qtemp.dblog

Optional. Whether to enable QTEMP Database logging.

Type: boolean

Default: false
qtemp.uuid.length

Optional. The QTEMP unique id length.

Type: number
Default: 9

quartz.scheduler.stand-by-if-error

Optional. Specifies whether to trigger job execution if the job scheduler is in standby mode. If true, when enabled job execution is not triggered.

Type: boolean
Default: false

warmUpCache

Optional. Specifies whether to load all datacom table data into a warm up cache at server start.

Type: boolean
Default: false

AWS Mainframe Modernization application definition reference

In AWS Mainframe Modernization, you configure migrated mainframe applications in an application definition JSON file, which is specific to the runtime engine you choose. An application definition contains both general information and engine-specific information. This topic describes both the AWS Blu Age and Micro Focus application definitions and identifies all required and optional elements.

Topics
- General header section (p. 334)
- Definition section overview (p. 335)
- AWS Blu Age application definition sample (p. 335)
- AWS Blu Age definition details (p. 336)
- Micro Focus application definition (p. 339)
- Micro Focus definition details (p. 340)

General header section

Each application definition starts with general information about the template version and source locations. The current version of the application definition is 2.0. Although version 1 still works, it is on a deprecation path. We recommend that you use version 2 when you create or update applications.

Use the following structure to specify the template version and source locations.

```
"template-version": "2.0",
"source-locations": [
```
Definition section overview

Specifies the resource definitions of the services, settings, data, and other typical resources that the application needs to run. When you update an application definition, AWS Mainframe Modernization detects changes by comparing the source-locations and definition lists from both the previous and the current versions of the application definition JSON file.

The definition section is engine-specific and subject to change. The following sections show sample engine-specific application definitions for both engines.

AWS Blu Age application definition sample

```json
{
  "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"
      }
    ]
  }
}
```
AWS Blu Age definition details

Listener(s) - required

Specify the port you will use to access the application through the AWS Mainframe Modernization-created Elastic Load Balancing. Use the following structure:

```json
"listeners": [{
  "port": 8194,
  "type": "http"
}]
```

**port**

Required. You can use any available port except for the well-known ports of 0 to 1023. We recommend using the range from 8192 to 8199. Make sure there's no other listeners or applications operating on this port.

**type**

Required. Currently, only http is supported.

AWS Blu Age application - required

Specify the location where the engine picks up the application image file using the following structure.

```json
"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

The specific location in Amazon S3 where the application image file is stored.

files-directory

Optional. The location of the input/output files for batches. Must be a subfolder of the Amazon EFS or Amazon FSx mount point setup at environment level.

enable-jics

Optional. Specifies whether to enable JICS. Defaults to true. Setting this to false prevents the JICS database from being spawned.

shared-app-location

Optional. Further location in Amazon S3 where shared application elements are stored. It can contain the same kind of application structure as app-location.

BluSAM - required

Specify the BluSAM database and Redis cache using the following structure.

```json
"blusam": {
  "db": {
    "nb-threads": 8,
    "batch-size": 10000,
    "name": "blusam",
  },
  "redis": {
    "hostname": "blusam.c3geul.ng.0001.usw2.cache.amazonaws.com",
    "port": 6379,
    "useSsl": true,
  }
}
```

db

Specifies the properties of the database used with the application. The database must be an Aurora PostgreSQL database. You can specify the following properties:

- nb-threads - Optional. Specifies how many dedicated threads are used for the write-behind mechanism that the blusam engine relies on. The default is 8.
- batch-size - Optional. Specifies the threshold that the write-behind mechanism uses to start batch storage operations. The threshold represents the number of modified records that will start a batch storage operation to ensure that modified records are persisted. The trigger itself is based on a combination of batch-size and an elapsed time of one second, whichever is reached first. The default is 10000.
- name - Optional. Specifies the name of the database.
- secret-manager-arn - Specifies the Amazon Resource Name (ARN) of the secret that contains the database credentials. For more information, see Step 2: Allow AWS Mainframe Modernization to retrieve the database admin credentials (p. 16).
redis

Specifies the properties of the Redis cache that the application uses to store temporary data that it needs in a central location to improve performance. We recommend that you both encrypt and password-protect the Redis cache.

- **hostname** - Specifies the location of the Redis cache.
- **port** - Specifies the port, typically 6379, where the Redis cache sends and receives communication.
- **useSsl** - Specifies whether the Redis cache is encrypted. If the cache is not encrypted, set useSsl to false.
- **secret-manager-arn** - Specifies the Amazon Resource Name (ARN) of the secret that contains the Redis cache password. If the Redis cache is not password-protected, do not specify secret-manager-arn. For more information, see Step 2: Allow AWS Mainframe Modernization to retrieve the database admin credentials (p. 16).

AWS Blu Age message queues - optional

Specify the JMS-MQ connection details for AWS Blu Age application.

```json
"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**

Required. Specifies the product type. Currently, this can only be "JMS-MQ" for AWS Blu Age applications.

**queue-manager**

Required. Specifies the name of the queue manager.

**channel**

Required. Specifies the name of the server-connection channel.

**hostname**

Required. Specifies the hostname of the message queue server.
port

Required. Specifies the listener port number the server is listening on.

user-id

Optional. Specifies the user account ID permitted to perform message queue operations on the specified channel.

secret-manager-arn

Optional. Specifies the Amazon Resource Name (ARN) of Secrets Manager that provides the password of the specified user.

Amazon Cognito authentication and authorization handler - optional

AWS Mainframe Modernization uses Amazon Cognito for authentication and authorization for migrated applications. Specify the Amazon Cognito authentication handler using the following structure.

```
"cognito-auth-handler": {
  "user-pool-id": "cognito-idp.Region.amazonaws.com/Region_rvYFnQIxl",
  "client-id": "58k05jb8grukjjsudm5Hhn1v87",
  "identity-pool-id": "Region:64464b12-0bfb-4dea-4b22c6c245f6"
}
```

user-pool-id

Specifies the Amazon Cognito user pool that AWS Mainframe Modernization uses to authenticate users of the migrated application. The AWS Region for the user pool should match the AWS Region for the AWS Mainframe Modernization application.

client-id

Specifies the migrated application that the authenticated user can access.

identity-pool-id

Specifies the Amazon Cognito identity pool where the authenticated user exchanges a user pool token for credentials that allow the user to access AWS Mainframe Modernization. The AWS Region for the identity pool should match the AWS Region for the AWS Mainframe Modernization application.

Micro Focus application definition

The following sample definition section is for the Micro Focus runtime engine, and contains both required and optional elements.

```
{
  "template-version": "2.0",
  "source-locations": [
    {
      "source-id": "s3-source",
      "source-type": "s3",
      "properties": {
        "s3-bucket": "mainframe-deployment-bucket-aaa",
```
Micro Focus definition details

The content in the definition section of the Micro Focus application definition file varies, depending on the resources that your migrated mainframe application requires at runtime.

**Listener(s) - required**

Specify a listener using the following structure:

```
"listeners": [{
  "port": 5101,
  "type": "tn3270"
},
]
```

**port**

For tn3270, the default is 5101. For other types of service listeners, the port varies. You can use any available port except for the well-known ports of 0 to 1023. Each listener should have a distinctive
port. Listeners should not share ports. For more information, see Listener Control in the Micro Focus Enterprise Server documentation.

type

Specifies the type of service listener. For more information, see Listeners in the Micro Focus Enterprise Server documentation.

Data set locations - required

Specify the data set location using the following structure.

```
"dataset-location": {
  "db-locations": [{
    "name": "Database1",
    "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456"
  }]
}
```

db-locations

- Specifies the location of the data sets that the migrated application creates. Currently, AWS Mainframe Modernization supports only data sets from a single VSAM database.
- name - Specifies the name of the database instance that contains the data sets that the migrated application creates.
- secret-manager-arn - Specifies the Amazon Resource Name (ARN) of the secret that contains the database credentials.

Amazon Cognito authentication and authorization handler - optional

AWS Mainframe Modernization uses Amazon Cognito for authentication and authorization for migrated applications. Specify the Amazon Cognito authentication handler using the following structure.

```
"cognito-auth-handler": {
  "user-pool-id": "cognito-idp.Region.amazonaws.com/Region_rvYFnQIXxl",
  "client-id": "S8k05jb8grukjjsudm5hnn1v87",
  "identity-pool-id": "Region:64464b12-0bfb-4dea-ab35-5c22c6c245f6"
}
```

user-pool-id

- Specifies the Amazon Cognito user pool that AWS Mainframe Modernization uses to authenticate users of the migrated application. The AWS Region for the user pool should match the AWS Region for the AWS Mainframe Modernization application.

client-id

- Specifies the migrated application that the authenticated user can access.

identity-pool-id

- Specifies the Amazon Cognito identity pool where the authenticated user exchanges a user pool token for credentials that allow the user to access AWS Mainframe Modernization. The AWS
Region for the identity pool should match the AWS Region for the AWS Mainframe Modernization application.

**Batch settings - required**

Specify the details required by the batch jobs that run as part of the application using the following structure.

```json
"batch-settings": {
  "initiators": [
    {
      "classes": ["A", "B"],
      "description": "initiator...."
    },
    {
      "classes": ["C", "D"],
      "description": "initiator...."
    }
  ],
  "jcl-file-location": "${s3-source}/batch/jcls"
}
```

**initiators**

Specifies a batch initiator that starts when the migrated application starts successfully and continues running until the application stops. You can define one or multiple classes per initiator. You can also define multiple initiators. For example:

```json
"batch-settings": {
  "initiators": [
    {
      "classes": ["A", "B"],
      "description": "initiator...."
    },
    {
      "classes": ["C", "D"],
      "description": "initiator...."
    }
  ],
  "jcl-file-location": "${s3-source}/batch/jcls"
}
```

For more information, see [To define a batch initiator or printer SEP](#) in the Micro Focus Enterprise Server documentation.

- **classes** - Specifies the job classes that the initiator can run. You can use up to 36 characters. You can use the following characters: A-Z or 0-9.
- **description** - Describes what the initiator is for.
- **jcl-file-location** - Specifies the location of the JCL files that are required by the batch jobs the migrated application runs.

**CICS settings - required**

Specify the details required for the CICS transactions that run as part of the application using the following structure.

```json
"cics-settings": {
```
"binary-file-location": "${s3-source}/cics/binaries",
"csd-file-location": "${s3-source}/cics/def",
"system-initialization-table": "BNKCICV"
}

### binary-file-location

Specifies the location of the CICS transaction program files.

### csd-file-location

Specifies the location of the CICS resource definition (CSD) file for this application. For more information, see [CICS Resource Definitions](#) in the *Micro Focus Enterprise Server* documentation.

### system-initialization-table

Specifies the system initialization table (SIT) that the migrated application uses. The name of the SIT table can be up to 8 characters. You can use A-Z, 0-9, $, @, and #. For more information, see [CICS Resource Definitions](#) in the *Micro Focus Enterprise Server* documentation.

### XA resources - required

Specify the details required for the XA resources that the application requires using the following structure.

```
"xa-resources" : [{
  "name": "XASQL",
  "module": "${s3-source}/xa/ESPGSQLXA64.so"
}]
```

### name

Required. Specifies the name of the XA resource.

### secret-manager-arn

Specifies the Amazon Resource Name (ARN) for the secret that contains the credentials for connecting to the database.

### module

Specifies the location of the RM switch module executable file. For more information, see [Planning and Designing XARs](#) in the *Micro Focus Enterprise Server* documentation.

---

**AWS Mainframe Modernization data set definition reference**

If your application requires more than a few data sets for processing, entering them one by one in the AWS Mainframe Modernization console is inefficient. Instead, we recommend that you create a JSON file to specify each data set. Different data set types are specified differently in the JSON, although many parameters are common. This document describes the details of the JSON required to import different types of data sets.
Note
Before you import any data sets, you must transfer the data sets from the mainframe to AWS. Then you must make sure that the data sets are converted from the mainframe format to a format that AWS can use. If necessary, transform the data as needed and store the transformed data sets in Amazon S3. Specify the name of the bucket and folder in the data set definition JSON file.

If you are using the Micro Focus runtime engine, you can use the DFCONV utility to convert the data sets. We include this utility in our Micro Focus Enterprise Developer and Enterprise Server images. For more information, see DFCONV Batch File Conversion in the Micro Focus Enterprise Developer documentation.

Topics
• Common properties (p. 344)
• Sample data set request format for VSAM (p. 345)
• Sample data set request format for GDG Base (p. 347)
• Sample data set request format for PS or GDG Generations (p. 347)
• Sample data set request format for PO (p. 348)

Common properties
Several parameters are common to all data sets. These parameters cover the following areas:

• Information about the data set (datasetName, datasetOrg, recordLength, encoding)
• Information about the location you are importing from; that is, the source location of the data set. This is not the location on the mainframe. It is the path to the Amazon S3 location where you uploaded the data set (externalLocation).
• Information about the location you are importing to; that is, the target location of the data set. This location is either a database or a file system, depending on your runtime engine. (storageType and relativePath).
• Information about the data set type (specific data set type, format, encoding, and so on).

Each data set definition has the same JSON structure. The following example JSON shows all these common parameters.

```json
{
  "dataSet": {
    "storageType": "Database",
    "datasetName": "MFI01V.MFIDEMO.BNKACC",
    "relativePath": "DATA",
    "datasetOrg": {
      "type": {
        type-specific properties ...
      },
    },
  },
}
```

The following properties are common to all data sets.

**storageType**

Required. Applies to the target location. Specifies whether the data set is stored in a database or a file system. Possible values are Database or FileSystem.
AWS Blu Age runtime engine: file systems are not supported. You must use a database.

Micro Focus runtime engine: databases and file systems are both supported. You can use either Amazon Relational Database Service or Amazon Aurora for databases, and Amazon Elastic File System or Amazon FSx for Lustre for file systems.

datasetName

Required. Specifies the fully qualified name of the data set as it appears on the mainframe.

relativePath

Required. Applies to the target location. Specifies the relative location of the data set in the database or file system.

datasetOrg

Required. Specifies the type of data set. Possible values are vsam, gdg, ps, po, or unknown.

- AWS Blu Age runtime engine: only VSAM type data sets are supported.
- Micro Focus runtime engine: VSAM, GDG, PS, PO, or Unknown type data sets are supported.

Note

If your application requires files that are not COBOL data files but are PDF or other binary files, you can specify them as follows:

```
"datasetOrg": {
    "type": PS,
    "format": U
}
```

Sample data set request format for VSAM

- AWS Blu Age runtime engine: supported.
- Micro Focus runtime engine: supported.

If you are importing VSAM data sets, specify vsam as the datasetOrg. Your JSON should resemble the following example:

```
{
    "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
    }
}
```
The following properties are supported for VSAM data sets.

**encoding**

Required. Specifies the character set encoding of the data set. Possible values are ASCII (A), EBCDIC (E), and Unknown (?).

**format**

Required. Specifies the VSAM data set type and the record format.

- AWS Blu Age runtime engine: possible values are ESDS (ES), KSDS (KS), and RRDS (RR). Record format can be fixed or variable.
- Micro Focus runtime engine: possible values are ESDS (ES), KSDS (KS), and RRDS (RR). The VSAM definition includes the record format, so you don't need to specify it separately.

**primaryKey**

Applies to VSAM KSDS data sets only. Specifies the primary key. Consists of the primary key name, key offset, and key length. The name is optional; offset and length are required.

**recordLength**

Required. Specifies the length of a record. For fixed-length record formats, these values must match.

- AWS Blu Age runtime engine: for VSAM ESDS, KSDS, and RRDS, min is optional and max is required.
- Micro Focus runtime engine: min and max are required.

**externalLocation**

Required. Specifies the source location: that is, the Amazon S3 bucket where you uploaded the data set.

**Blu Age engine-specific properties**

The AWS Blu Age runtime engine supports compression for VSAM data sets. The following example shows how you can specify this property in JSON.

```json
{
    "datasetOrg": {
        "vsam": {
            "compressed": boolean,
            "common properties"
        }
    }
}
```

Specify the compression property as follows:
Sample data set request format for GDG Base

- AWS Blu Age runtime engine: not supported.
- Micro Focus runtime engine: supported.

If you are importing GDG base data sets, specify gdg as the datasetOrg. Your JSON should resemble the following example:

```json
{
    "storageType": "Database",
    "datasetName": "AWS.M2.GDG",
    "relativePath": "DATA",
    "datasetOrg": {
        "gdg": {
            "limit": "3",
            "rollDisposition": "Scratch and No Empty"
        }
    }
}
```

The following properties are supported for GDG base data sets.

**limit**

Required. Specifies the number of active generations, or biases. For a GDG base cluster, the maximum is 255.

**rollDisposition**

Optional. Specifies how to handle generation data sets when the maximum is reached or exceeded. Possible values are No Scratch and No Empty, Scratch and No Empty, Scratch and Empty, or No Scratch and Empty. The default is Scratch and No Empty.

Sample data set request format for PS or GDG Generations

- AWS Blu Age runtime engine: not supported.
- Micro Focus runtime engine: supported.

If you are importing PS or GDG generations data sets, specify ps as the datasetOrg. Your JSON should resemble the following example:

```json
{
    "storageType": "Database",
    "datasetName": "AWS.M2.PS.FB",
    "relativePath": "DATA",
    "datasetOrg": {
        "ps": {
            "limit": "3",
            "rollDisposition": "Scratch and No Empty"
        }
    }
}
```
The following properties are supported for PS or GDG generations data sets.

**format**

Required. Specifies the format of the data set records. Possible values are F, FA, FB, FBA, FBM, FBS, FM, FS, LSEQ, U, V, VA, VB, VBA, VBM, VBS, VM, and VS.

**encoding**

Required. Specifies the character set encoding of the data set. Possible values are ASCII (A), EBCDIC (E), and Unknown (\?).

**recordLength**

Required. Specifies the length of a record. You must specify both the minimum (min) and maximum (max) length of the record. For fixed-length record formats, these values must match.

**externalLocation**

Required. Specifies the source location: that is, the Amazon S3 bucket where you uploaded the data set.

### Sample data set request format for PO

If you are importing PO data sets, specify po as the datasetOrg. Your JSON should resemble the following example:

```
{
    "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/"
}
```
The following properties are supported for PO data sets.

**format**

Required. Specifies the format of the data set records. Possible values are F, FA, FB, FBA, FBM, FBS, FM, FS, LSEQ, U, V, VA, VB, VBA, VBM, VBS, VM, and VS.

**encoding**

Required. Specifies the character set encoding of the data set. Possible values are ASCII (A), EBCDIC (E), and Unknown (?)

**memberFileExtensions**

Required. Specifies an array containing one or more filename extensions, allowing you to specify which files to be included as PDS member.

**recordLength**

Optional. Specifies the length of a record. Both the minimum (min) and maximum (max) length of the record are optional. For fixed-length record formats, these values must match.

**externalLocation**

Required. Specifies the source location: that is, the Amazon S3 bucket where you uploaded the data set.

**Note**

The current implementation for the Micro Focus runtime engine adds PDS entries as dynamic data sets.
Managed Runtime Environments in AWS Mainframe Modernization

If you're new to AWS Mainframe Modernization see the following topics to get started:

- What is AWS Mainframe Modernization? (p. 1)
- Setting up AWS Mainframe Modernization (p. 4)
- Getting started with AWS Mainframe Modernization (p. 6)
- Tutorial: Managed Runtime for AWS Blu Age (p. 6)
- Tutorial: Managed runtime for Micro Focus (p. 15)

A runtime environment in AWS Mainframe Modernization is a named combination of AWS compute resources, a runtime engine, and the configuration details that you specify. The runtime environment hosts one or more applications. Applications in AWS Mainframe Modernization contain migrated mainframe workloads. You can choose the runtime engine for the environments that you create. Choose AWS Blu Age if you are using the automated refactoring pattern, and Micro Focus if you are using the replatforming pattern. You can also choose the amount of compute resources that are right for your application and optionally attach storage to runtime environments. AWS Mainframe Modernization enables Amazon CloudWatch metrics and logging for you so that you can monitor your runtime environment.

Topics

- Create an AWS Mainframe Modernization runtime environment (p. 350)
- Update an AWS Mainframe Modernization runtime environment (p. 352)
- Stop an AWS Mainframe Modernization runtime environment (p. 353)
- Restart an AWS Mainframe Modernization runtime environment (p. 354)
- Delete an AWS Mainframe Modernization runtime environment (p. 355)

Create an AWS Mainframe Modernization runtime environment

Use the AWS Mainframe Modernization console to create an AWS Mainframe Modernization environment.

These instructions assume that you've completed the steps in Setting up AWS Mainframe Modernization (p. 4).

Create a runtime environment

To create a runtime environment

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where you want to create the environment.
3. On the Environments page, choose Create environment.
4. On the Specify basic information page, provide the following information:
   a. In the Name and description section, enter a name for the environment.
b. (Optional) In the **Environment description** field, enter a description for the environment. This description can help you and other users identify the purpose of the runtime environment.

c. In the **Engine options** section, choose **Blu Age** for automated refactoring, or **Micro Focus** for replatforming.

d. Choose a version for the engine that you selected.

e. (Optional) In the **Tags** section, choose **Add new tag** to add one or more environment tags to your environment. An environment tag is a custom attribute label that helps you organize and manage your AWS resources.

f. Choose **Next**.

5. On the **Specify configurations** page, provide the following information:

a. In the **Availability** section, choose **Standalone runtime environment** or **High availability cluster**.

The availability pattern determines how available your application will be when it runs. **Standalone** is fine for development purposes. **High availability** is for applications that must be available at all times.

b. In **Resources**, choose an instance type and desired capacity.

These resources are the AWS Mainframe Modernization managed Amazon EC2 instances that will host your runtime environment. Standalone runtime environments offer two choices for instance type and permit only one instance. High availability runtime environments offer two choices for instance type and permit up to two instances.

For more information, see [Amazon EC2 Instance Types](https://docs.aws.amazon.com/AWSEC2/latest/quickstart/quickstarts-ec2-instance-types.html), and contact an AWS mainframe specialist for guidance.

6. In the **Security and network** section, do the following:

a. If you want the applications to be publicly accessible, choose **Allow applications deployed to this environment to be publicly accessible**.

b. Choose a Virtual Private Cloud (VPC).

c. If you’re using the high availability pattern, choose two or more subnets. If you’re using the standalone pattern with the AWS Blu Age engine, choose two or more subnets. If you’re using the standalone pattern with the Micro Focus engine, you can specify one subnet.

d. Choose a security group for the VPC that you selected.

   **Note**
   AWS Mainframe Modernization creates a Network Load Balancer for you to distribute connections to your runtime environment. Make sure your security group inbound rules allow access from an IP address to the port you specified in the listener property of the application definition. For more information, see [Register targets](https://docs.aws.amazon.com/mainframe-modernization/latest/userguide/register-targets.html) in the [User Guide for Network Load Balancers](https://docs.aws.amazon.com/mainframe-modernization/latest/userguide/network-load-balancers.html).

e. In the **KMS key** field, choose **Customize encryption settings** if you want to use a customer managed AWS KMS key. For more information, see [Data encryption at rest for AWS Mainframe Modernization service](https://docs.aws.amazon.com/mainframe-modernization/latest/userguide/data-encryption-at-rest.html) (p. 361).

   **Note**
   By default, AWS Mainframe Modernization encrypts your data with a AWS KMS key that AWS Mainframe Modernization owns and manages for you. However, you can choose to use a customer managed AWS KMS key.

f. (Optional) Choose an AWS KMS key by name or Amazon Resource Name (ARN). Alternately, choose **Create an AWS KMS key** to go to the AWS KMS console and create a new AWS KMS key.

   Choose **Next**.

7. (Optional) On the **Attach storage** page, choose one or more Amazon EFS or Amazon FSx file systems, and then choose **Next**.
8. In the **Maintenance window** section, choose when you want to apply pending changes to the environment.

- If you choose **No preference**, AWS Mainframe Modernization chooses an optimized maintenance window for you.
- If you want to specify a particular maintenance window, choose **Select new maintenance window**. Then choose a day of the week, a start time, and a duration for the maintenance window.

For more information about the maintenance window, see [AWS Mainframe Modernization maintenance window](p. 353).

Choose **Next**.

9. On the **Review and create** page, review the information that you entered, and then choose **Create environment**.

---

**Update an AWS Mainframe Modernization runtime environment**

Use the AWS Mainframe Modernization console to update an AWS Mainframe Modernization runtime environment. You can update the minor version of the runtime engine or the instance type that hosts the runtime environment. You can choose whether you want to apply updates immediately or during the preferred maintenance window.

These instructions assume that you have completed the steps in [Setting up AWS Mainframe Modernization](p. 4).

**Update a runtime environment**

**To update a runtime environment**

1. Open the AWS Mainframe Modernization console at [https://console.aws.amazon.com/m2/](https://console.aws.amazon.com/m2/).
2. In the AWS Region selector, choose the Region where the environment that you want to update was created.
3. On the **Environments** page, choose the environment that you want to update.
4. On the details page for the environment, choose **Actions**, and then choose **Edit environment**.
5. Make any of the following changes:

   - In the **Engine options** section, choose the engine version that you want.
   - In the **Resources** section, choose the instance type that you want.
   - In the **Maintenance window** section, choose the day, time, and duration that you want.

   **Note**
   The only changes that you can choose to apply during the maintenance window are changes to the engine version. You must apply all other changes immediately.

6. Choose **Next**.
7. In **When to apply these changes**, choose **Immediately** or **During the next maintenance window**. Then choose **Update environment**.

If you choose **Immediately**, you see a message when the environment has finished updating.
AWS Mainframe Modernization maintenance window

Every runtime environment has a weekly one-hour maintenance window. Any system changes are applied during this time. The maintenance window is your chance to control when modifications and software patching occur. If a maintenance event is scheduled for a given week, it begins during that one-hour maintenance window. Most maintenance events also complete during the one-hour maintenance window, although larger maintenance events might take more than an hour to complete.

The one-hour maintenance window is selected at random from an 8 hour block of time per Region. If you don’t specify a maintenance window when you create a runtime environment, AWS Mainframe Modernization assigns a 1 hour maintenance window on a randomly selected day of the week.

AWS Mainframe Modernization consumes some of the resources in your environment instance while maintenance is being applied. You might observe a minimal effect on performance or some disruptions in applications during maintenance.

The following table shows the default time blocks when maintenance windows are assigned for each Region.

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
<th>Time Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>03:00–11:00 UTC</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>06:00–14:00 UTC</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
<td>06:00–14:00 UTC</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
<td>14:00–22:00 UTC</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
<td>12:00–20:00 UTC</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
<td>13:00–21:00 UTC</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
<td>03:00–11:00 UTC</td>
</tr>
<tr>
<td>Europe (Frankfurt)</td>
<td>eu-central-1</td>
<td>21:00–05:00 UTC</td>
</tr>
<tr>
<td>Europe (Ireland)</td>
<td>eu-west-1</td>
<td>22:00–06:00 UTC</td>
</tr>
<tr>
<td>Europe (London)</td>
<td>eu-west-2</td>
<td>22:00–06:00 UTC</td>
</tr>
<tr>
<td>Europe (Paris)</td>
<td>eu-west-3</td>
<td>23:59–07:29 UTC</td>
</tr>
<tr>
<td>South America (São Paulo)</td>
<td>sa-east-1</td>
<td>00:00–08:00 UTC</td>
</tr>
</tbody>
</table>

Stop an AWS Mainframe Modernization runtime environment

Use the AWS Mainframe Modernization console to stop an AWS Mainframe Modernization runtime environment. When you stop an environment the current application deployments are retained and you won't be charged for the environment until the environment is restarted.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4).
Stop a runtime environment

If you need to stop an AWS Mainframe Modernization runtime environment, you follow similar steps as the update environment section.

Use the AWS Mainframe Modernization console to stop an AWS Mainframe Modernization runtime environment. When you stop an environment, the current application deployments are retained and you won't be charged for the environment until the environment is restarted.

To stop a runtime environment

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where the environment that you want to stop was created.
3. On the Environments page, choose the environment that you want to stop.
4. On the details page for the environment, choose Actions, and then choose Edit environment.
5. On the Edit environment page, find Resources section, and update the desired capacity to zero.
   
   **Note**
   You must stop all applications before stopping environment.

6. Choose Next.
7. In When to apply these changes, choose Immediately. Then choose Update environment.

You see a message when the environment capacity is updated.

Restart an AWS Mainframe Modernization runtime environment

Use the AWS Mainframe Modernization console to restart an AWS Mainframe Modernization runtime environment. When you restart a runtime environment, the billing for the environment will be resumed.

To restart a runtime environment

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where the environment that you want to restart was created.
3. On the Environments page, choose the environment that you want to restart.
4. On the details page for the environment, choose Actions, and then choose Edit environment.
Delete a runtime environment

5. On the Edit environment page, find Resources section, and update the desired capacity from zero to the required capacity.
6. Choose Next.
7. In When to apply these changes, choose Immediately. Then choose Update environment.

You see a message when the environment capacity is updated and the environment is restarted.

Delete an AWS Mainframe Modernization runtime environment

Use the AWS Mainframe Modernization console to delete an AWS Mainframe Modernization runtime environment.

These instructions assume that you have completed the steps in Setting up AWS Mainframe Modernization (p. 4).

Delete a runtime environment

If you need to delete an AWS Mainframe Modernization runtime environment, make sure that you delete any deployed applications from the environment first. You can’t delete a runtime environment where applications are deployed.

To delete an environment
1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region where the environment that you want to delete was created.
3. On the Environments page, choose the environment that you want to delete, and then choose Actions and Delete environment.
4. In the Delete environment window, enter delete to confirm that you want to delete the runtime environment, and then choose Delete.
Application

A running mainframe workload in AWS Mainframe Modernization. A set of batch jobs, interactive transactions (CICS or IMS), or other components comprise an application. You define the scope. You must define and specify any components or resources that the workload needs, such as CICS transactions or batch jobs.

Application definition

The definition or specification of the components and resources needed by an application (mainframe workload) running in AWS Mainframe Modernization. Separating the definition from the application itself is important because it is possible to reuse the same definition for multiple stages (Pre-production, Production), represented by different runtime environments.

Batch job

A scheduled program that is configured to run without requiring user interaction. In AWS Mainframe Modernization, you will need to store both batch job JCL files and batch job binaries in an Amazon S3 bucket, and provide the location of both in the application definition file. When you run a batch job, AWS Mainframe Modernization reports the following status values:

Submitting

The batch job is in the process of being submitted.
**Configuration**

The characteristics of an environment or application. Environment configurations consist of engine type, engine version, availability patterns, optional file system configurations, and more.

Application configurations can be static or dynamic. Static configurations change only when you update an application by deploying a new version. Dynamic configurations, which are usually an operational activity such as turning tracing on or off, change as soon as you update them.

**Data set**

A file containing data for use by applications.

**Environment**

A named combination of AWS compute resources, a runtime engine, and configuration details created to host one or more applications.

**Mainframe modernization**

The process of migrating applications from a legacy mainframe environment to AWS.
Migration journey

The end-to-end process of migrating and modernizing legacy applications, typically made of the following phases: Assess, Mobilize, Migrate and modernize, and Operate and optimize.

Mount point

A directory in a file system that provides access to the files stored within that system.

Automated Refactoring

The process of modernizing legacy application artifacts for running in a modern cloud environment. It can include code and data conversion. For more information, see AWS Mainframe Modernization Automated Refactor.

Replatforming

The process of moving an application and application artifacts from one computing platform to a different computing platform. For more information, see AWS Mainframe Modernization Replatform

Resource

A physical or virtual component within a computer system.

Runtime engine

Software that facilitates the running of an application.
Security in AWS Mainframe Modernization

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Mainframe Modernization, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Mainframe Modernization. It shows you how to configure AWS Mainframe Modernization to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Mainframe Modernization resources.

AWS Mainframe Modernization provides its own IAM-protected resources (application, environment, deployment etc), which are the AWS Mainframe Modernization administrative resources, on which any action must be allowed by IAM policies.

AWS Mainframe Modernization for replatforming is also secured by IAM. IAM grants or denies permission to a principal for a specific action on a defined resource, derived from the original mainframe environment, through standard IAM policies as well. The AWS Mainframe Modernization replatforming runtime calls the IAM authorization service when an application attempts such action on a protected resource. IAM will return allow or deny based on standard IAM policy evaluation mechanisms.

Contents

- Data protection in AWS Mainframe Modernization (p. 359)
- Identity and Access Management for AWS Mainframe Modernization (p. 377)
- Compliance validation for AWS Mainframe Modernization (p. 395)
- Resilience in AWS Mainframe Modernization (p. 395)
- Infrastructure security in AWS Mainframe Modernization (p. 396)
- Tutorial: AWS Blu Age Application Security in AWS Mainframe Modernization (p. 396)
- Access AWS Mainframe Modernization using an interface endpoint (AWS PrivateLink) (p. 406)

Data protection in AWS Mainframe Modernization

The AWS shared responsibility model applies to data protection in AWS Mainframe Modernization. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on
this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a Name field. This includes when you work with AWS Mainframe Modernization or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data that AWS Mainframe Modernization collects

AWS Mainframe Modernization collects several types of data from you:

- Application configuration. This is a JSON file that you create to configure your application. It contains your choices for the different options that AWS Mainframe Modernization offers. The file also contains information for dependent AWS resources such as Amazon Simple Storage Service paths where application artifacts are stored or the Amazon Resource Name (ARN) for AWS Secrets Manager where your database credentials are stored.
- Application executable (binary). This is a binary that you compile and that you intend to deploy on AWS Mainframe Modernization.
- Application JCL or scripts. This source code manages batch jobs or other processing on behalf of your application.
- User application data. When you import data sets, AWS Mainframe Modernization stores them in the relational database so your application can access them.
- Application source code. Through Amazon AppStream 2.0, AWS Mainframe Modernization provides a development environment for you to write and compile code.

AWS Mainframe Modernization stores this data natively in AWS. The data we collect from you is stored in an AWS Mainframe Modernization-managed Amazon S3 bucket. When you deploy an application, AWS Mainframe Modernization downloads the data onto an Amazon Elastic Block Store-backed Amazon Elastic Compute Cloud instance. When cleanup is triggered, the data is removed from the Amazon EBS volume and from Amazon S3. The Amazon EBS volumes are single-tenanted, meaning that one instance is used for one customer. Instances are never shared. When you delete a runtime environment, the Amazon EBS volume is also deleted. When you delete an application, the artifacts and configuration are deleted from Amazon S3.
Application logs are stored in Amazon CloudWatch. Customer application log messages are exported to CloudWatch as well. The CloudWatch logs might contain customer-sensitive data, such as business data or security information in debug messages). For more information, see Monitoring AWS Mainframe Modernization with Amazon CloudWatch (p. 408).

In addition, if you choose to attach one or more Amazon Elastic File System or Amazon FSx file systems to your runtime environment, the data within those systems will be stored in AWS. You will need to clean up that data if you decide to stop using the file systems.

You can use all available Amazon S3 encryption options to secure your data when you place it in the Amazon S3 bucket that AWS Mainframe Modernization uses for application deployment and dataset imports. In addition, you can use the Amazon EFS and Amazon FSx encryption options if you attach one or more of these file systems to your runtime environment.

**Data encryption at rest for AWS Mainframe Modernization service**

AWS Mainframe Modernization integrates with AWS Key Management Service to provide transparent server side encryption (SSE) on all dependent resources that store data permanently; namely Amazon Simple Storage Service, Amazon DynamoDB, and Amazon Elastic Block Store. AWS Mainframe Modernization creates and manages symmetric encryption AWS KMS keys for you in AWS KMS.

Encryption of data at rest by default helps reduce the operational overhead and complexity involved in protecting sensitive data. At the same time, it enables you to migrate applications that require strict encryption compliance and regulatory requirements.

Although you can't disable this layer of encryption or select an alternate encryption type, you can add a second layer of encryption by choosing a customer managed key when you create runtime environments, applications, and deployments.

You can use your own customer managed key for AWS Mainframe Modernization applications and runtime environments to encrypt Amazon S3 and Amazon EBS resources.

For your AWS Mainframe Modernization applications, you can use this key to encrypt your application definition as well as other application resources, like JCL files, which are saved in the Amazon S3 bucket that is created in the service's account. For more information, see Create an application (p. 304).

For your AWS Mainframe Modernization runtime environments, AWS Mainframe Modernization uses your customer managed key to encrypt the Amazon EBS volume that it creates and attaches to your AWS Mainframe Modernization Amazon EC2 instance, which is also in the service's account. For more information, see Create a runtime environment (p. 350).

**Note**

DynamoDB resources are always encrypted using an AWS managed key in the AWS Mainframe Modernization service account. You cannot encrypt DynamoDB resources using a customer managed key.

AWS Mainframe Modernization uses your customer managed key for the following tasks:

- Redeploying an application.
- Replacing a AWS Mainframe Modernization Amazon EC2 instance.

AWS Mainframe Modernization doesn't use your customer managed key to encrypt Amazon Relational Database Service or Amazon Aurora databases, Amazon Simple Queue Service queues, and Amazon ElastiCache caches that are created to support a AWS Mainframe Modernization application, because none of them contain customer data.
For more information, see Customer managed keys in the AWS Key Management Service Developer Guide.

The following table summarizes how AWS Mainframe Modernization encrypts your sensitive data.

<table>
<thead>
<tr>
<th>Data type</th>
<th>AWS managed key encryption</th>
<th>Customer managed key encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Contains the definition for a particular application.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnvironmentSummary</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Contains information about the runtime environment.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ApplicationSummary</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Contains information about the AWS Mainframe Modernization application.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeploymentSummary</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Contains information about a deployment of an AWS Mainframe Modernization application.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note**
AWS Mainframe Modernization automatically enables encryption at rest using AWS managed keys to protect your sensitive data at no charge. However, AWS KMS charges apply for using a customer managed key. For more information about pricing, see AWS Key Management Service Pricing.

For more information on AWS KMS, see AWS Key Management Service

**How AWS Mainframe Modernization uses grants in AWS KMS**

AWS Mainframe Modernization requires a grant to use your customer managed key.

When you create an application or runtime environment, or deploy an application in AWS Mainframe Modernization encrypted with a customer managed key, AWS Mainframe Modernization creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give AWS Mainframe Modernization access to a KMS key in a customer account.

AWS Mainframe Modernization requires the grant to use your customer managed key for the following internal operations:

- Send DescribeKey requests to AWS KMS to verify that the symmetric customer managed key ID entered when creating an application, runtime environment, or application deployment is valid.
- Send GenerateDataKey requests to AWS KMS to encrypt the Amazon EBS volume attached to Amazon EC2 instances that host AWS Mainframe Modernization runtime environments.
- Send Decrypt requests to AWS KMS to decrypt encrypted content on Amazon EBS.
AWS Mainframe Modernization uses AWS KMS grants to decrypt your secrets stored in Secrets Manager and in workflows such as creating a runtime environment, creating or redeploying an application, and creating a deployment. The grants that AWS Mainframe Modernization creates support the following operations:

- Create or update a runtime environment grant:
  - Decrypt
  - Encrypt
  - ReEncryptFrom
  - ReEncryptTo
  - GenerateDataKey
  - DescribeKey
  - CreateGrant
- Create or redeploy an application grant:
  - GenerateDataKey
- Create a deployment grant:
  - Decrypt

You can revoke access to the grant, or remove the service's access to the customer managed key at any time. If you do, AWS Mainframe Modernization won't be able to access any of the data encrypted by the customer managed key, which affects operations that depend on the data. For example, if AWS Mainframe Modernization tried to access an application definition encrypted by a customer managed key without the grant to that key, the application creation operation would fail.

AWS Mainframe Modernization collects user application configurations (JSON files) and artifacts (binaries and executables). It also creates metadata that tracks various entities used for the operation of AWS Mainframe Modernization, and creates logs and metrics. The logs and metrics that are customer-visible include:

- CloudWatch logs that reflect application and runtime engine (either AWS Blu Age or Micro Focus).
- CloudWatch metrics for operation dashboards.

In addition, AWS Mainframe Modernization collects usage data and metrics for metering, activity reporting, and so on about the services. This data is not customer-visible.

AWS Mainframe Modernization stores this data in different places depending on the type of data. Customer data that you upload is stored in an Amazon S3 bucket. Service data is stored in both Amazon S3 and DynamoDB. When you deploy an application, both your data and service data are downloaded onto Amazon EBS volumes. If you choose to attach Amazon EFS or Amazon FSx storage to your runtime environment, data stored in those file systems is also downloaded to the Amazon EBS volume.

Encryption at rest is configured by default. You cannot disable it or change it. Currently, you cannot change its configuration either.

**Create a customer managed key**

You can create a symmetric customer managed key by using the AWS Management Console or the AWS KMS APIs.

**To create a symmetric customer managed key**

Follow the steps for [Creating symmetric customer managed key](https://docs.aws.amazon.com/kms/latest/developerguide/cmk-create-sync.html) in the AWS Key Management Service Developer Guide.
**Key policy**

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it. When you create your customer managed key, you can specify a key policy. For more information, see [Managing access to customer managed keys](https://docs.aws.amazon.com/kms/latest/APIReference/API_CreateGrant.html) in the *AWS Key Management Service Developer Guide*.

To use your customer managed key with your AWS Mainframe Modernization resources, the following API operations must be permitted in the key policy:

- **kms:CreateGrant** – Adds a grant to a customer managed key. Grants control access to a specified KMS key, which allows access to grant operations AWS Mainframe Modernization requires. For more information about [Using Grants](https://docs.aws.amazon.com/kms/latest/APIReference/API_CreateGrant.html), see the *AWS Key Management Service Developer Guide*.

  This allows AWS Mainframe Modernization to do the following:
  - Call *GenerateDataKey* to generate an encrypted data key and store it, because the data key isn't immediately used to encrypt.
  - Call *Decrypt* to use the stored encrypted data key to access encrypted data.
  - Set up a retiring principal to allow the service to *RetireGrant*.
- **kms:DescribeKey** – Provides the customer managed key details to allow AWS Mainframe Modernization to validate the key.

AWS Mainframe Modernization requires kms:CreateGrant and kms:DescribeKey permissions in the customer's key policy. AWS Mainframe Modernization uses this policy to create a grant for itself.

```json
{
"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**

The role shown for `Principal` in the preceding example is the one that you use for AWS Mainframe Modernization operations such as `CreateApplication` and `CreateEnvironment`.

For more information about [specifying permissions in a policy](https://docs.aws.amazon.com/kms/latest/APIReference/API_CreateGrant.html), see the *AWS Key Management Service Developer Guide*.

For more information about [troubleshooting key access](https://docs.aws.amazon.com/kms/latest/APIReference/API_CreateGrant.html), see the *AWS Key Management Service Developer Guide*.

## Specifying a customer managed key for AWS Mainframe Modernization

You can specify a customer managed key as a second layer encryption for the following resources:
• Application
• Deployment
• Environment

When you create a resource, you can specify the key by entering a **KMS ID**, which AWS Mainframe Modernization uses to encrypt the sensitive data stored by the resource.

• **KMS ID**— A [key identifier](#) for a customer managed key. Enter a key ID, key ARN, alias name, or alias ARN.

You can specify a customer managed key using the AWS Management Console or the AWS CLI.

To specify your customer managed key when creating a runtime environment in the AWS Management Console, see [Create an AWS Mainframe Modernization runtime environment](#). To specify your customer managed key when creating an application in the AWS Management Console, see [Create an AWS Mainframe Modernization application](#).

To specify your customer managed key when you create a runtime environment with the AWS CLI, specify the `kms-key-id` parameter, as follows:

```bash
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
```

To add your customer managed key when you create an application with the AWS CLI, specify the `kms-key-id` parameter, as follows:

```bash
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 encryption context**

An **encryption context** is an optional set of key-value pairs that contain additional contextual information about the data.

AWS KMS uses the encryption context as **additional authenticated data** to support **authenticated encryption**. When you include an encryption context in a request to encrypt data, AWS KMS binds the encryption context to the encrypted data. To decrypt data, you include the same encryption context in the request.

**AWS Mainframe Modernization encryption context**

AWS Mainframe Modernization uses the same encryption context in all AWS KMS cryptographic operations, where the key is `aws:m2:app` and the value is the unique identifier of the application.

**Example**

```
"encryptionContextSubset": {
```
Monitoring your encryption keys

Using encryption context for monitoring

When you use a symmetric customer managed key to encrypt your applications or runtime environments, you can also use the encryption context in audit records and logs to identify how the customer managed key is being used.

Using encryption context to control access to your customer managed key

You can use the encryption context in key policies and IAM policies as conditions to control access to your symmetric customer managed key. You can also use encryption context constraints in a grant.

AWS Mainframe Modernization uses an encryption context constraint in grants to control access to the customer managed key in your account or region. The grant constraint requires that the operations that the grant allows use the specified encryption context. The following example shows a request to CreateGrant during the CreateApplication workflow that uses an encryption context constraint.

```json
//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"
        }
    }
}
```

Monitoring your encryption keys for AWS Mainframe Modernization

When you use an AWS KMS customer managed key with your AWS Mainframe Modernization resources, you can use AWS CloudTrail or Amazon CloudWatch Logs to track requests that AWS Mainframe Modernization sends to AWS KMS.

Examples for runtime environments

The following examples are AWS CloudTrail events for DescribeKey, CreateGrant, GenerateDataKey, and Decrypt to monitor KMS operations called by AWS Mainframe Modernization to access data encrypted by your customer managed key:

DescribeKey

AWS Mainframe Modernization uses the DescribeKey operation to verify if the AWS KMS customer managed key associated with your runtime environment exists in the account and region.

The following example event records the DescribeKey operation:

```json
{
    "eventVersion": "1.08",
    "userIdentity": {
```
When you use an AWS KMS customer managed key to encrypt your runtime environment, AWS Mainframe Modernization sends several CreateGrant requests on your behalf to perform necessary KMS operations. Some of the grants that AWS Mainframe Modernization creates are retired immediately after use. Others are retired when you delete the runtime environment.

The following example event records the CreateGrant operation for the Lambda execution role associated with the Create Environment workflow.

```json
{...
}```
Monitoring your encryption keys

"eventVersion": "1.08",
"userIdentity": {
  "type": "AssumedRole",
  "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
  "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
  "accountId": "111122223333",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
  "sessionContext": {
    "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": "0ab0ac0d0b000f000a000c0b000c000f0000000c0b0a000aaafSAMPLE",
  "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
},
"requestID": "ff000af-00eb-00ce-0000-000f00000000c0b0a000aaafSAMPLE",
"eventID": "ff000af-00eb-00ce-0000-000f00000000c0b0a000aaafSAMPLE",
"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"
The following example event records the `CreateGrant` operation for the Auto Scaling group service-linked role. The Lambda execution role associated with the Create Environment workflow calls this `CreateGrant` operation. It grants permission for the execution role to create a subgrant against the Auto Scaling group's service-linked role.

```json

{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65MZFUPM4J0:EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfXDJ5zxz86tr",
        "arn": "arn:aws:sts::111122223333:assumed-role/EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4ABVQEEKN/EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfXDJ5zxz86tr",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4ABVQEEKN",
                "accountId": "111122223333",
                "userName": "EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4ABVQEEKN"
            },
            "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": "0ab0ac0d0b0f0000a0000f0000000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    }
}
```
When you enable an AWS KMS customer managed key for your runtime environment resource, Auto Scaling creates a unique key for encrypting the Amazon EBS volume associated with the runtime environment. It sends a GenerateDataKey request to AWS KMS that specifies the AWS KMS customer managed key for the resource.

The following example event records the GenerateDataKey operation:

```json
{
    "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",
    "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"
    }
}
```
Decrypt

When you access an encrypted runtime environment, Amazon EBS calls the Decrypt operation to use the stored encrypted data key to access the encrypted data.

The following example event records the Decrypt operation:

```json
{
   "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",
   "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",
   "eventCategory": "Management"
}
```
Examples for applications

The following examples are AWS CloudTrail events for CreateGrant and GenerateDataKey to monitor KMS operations called by AWS Mainframe Modernization to access data encrypted by your customer managed key:

CreateGrant

When you use an AWS KMS customer managed key to encrypt your application resources, the Lambda execution role sends a CreateGrant request on your behalf to access the KMS key in your AWS account. The grant allows the Lambda execution role to upload customer application resources to Amazon S3 using your customer managed key. This grant is retired immediately after the application is created.

The following example event records the CreateGrant operation:

```json
{
   "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": "2022-12-06T21:51:45Z",
   "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",
```
When you enable an AWS KMS customer managed key for your application resource, the Lambda execution role creates a key that it uses to encrypt and upload customer data to Amazon Simple Storage Service. The Lambda execution role sends a GenerateDataKey request to AWS KMS that specifies the AWS KMS customer managed key for the resource.

The following example event records the GenerateDataKey operation:

```json

"operations": [
  "GenerateDataKey"
],
"granteePrincipal": "m2.us-west-2.amazonaws.com",
"responseElements": {
  "grantId": "0ab0ac0bde00f0a00cc0a0ef0c00ce000c000f000000c0bc0a0000aaafSAMPLE",
  "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"
}
```
Examples for deployments

The following examples are AWS CloudTrail events for CreateGrant and Decrypt to monitor KMS operations called by AWS Mainframe Modernization to access data encrypted by your customer managed key:

CreateGrant

When you use an AWS KMS customer managed key to encrypt your deployment resources, AWS Mainframe Modernization sends two CreateGrant requests on your behalf. The first grant is against the current Lambda execution role to call ListBatchJobScriptFiles, and is retired immediately after deployment finishes. The second grant is against the Amazon EC2 scoped down instance role so that Amazon EC2 can download customer application resources from Amazon S3. This grant is retired when the application is deleted from the runtime environment.

The following example event records the CreateGrant operation:

```json
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIL05Q00NN7EXAMPLE3",
    "sessionContext": {
      "sessionIssuer": {
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "accountId": "111122223333"
      }
    }
  },
  "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"
}
```
"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": "0ab0ac0d0b000f00ea00cc0a000f000be00c000f00000000c0bc0a000aaaafSAMPLE",
    "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
},
"requestID": "ff000af-00eb-00ce-0e00-0a000f00fbaa0SAMPLE",
"eventID": "ff000af-00eb-00ce-0e00-0a000f00fbaa0SAMPLE",
"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

When you access a deployment, Amazon EC2 calls the Decrypt operation to use the stored encrypted data key to decrypt and download encrypted customer data from Amazon S3.

The following example event records the Decrypt operation:
Learn more

The following resources provide more information about data encryption at rest.
• For more information about AWS Key Management Service basic concepts, see the AWS Key Management Service Developer Guide.
• For more information about Security best practices for AWS Key Management Service, see the AWS Key Management Service Developer Guide.

Encryption in transit

For interactive applications that are part of transactional workloads, the data exchanges between the terminal emulator and the AWS Mainframe Modernization service endpoint for TN3270 protocol are not encrypted in transit. If the application requires encryption in transit, you might want to implement some additional tunneling mechanisms.

AWS Mainframe Modernization uses HTTPS to encrypt the service APIs. All other communication within AWS Mainframe Modernization is protected by the service VPC or security group, as well as HTTPS. AWS Mainframe Modernization transfers application artifacts, configurations, and application data. Application artifacts are copied from an Amazon S3 bucket that you own, as are application data. You can provide application configurations using a link to Amazon S3 or by uploading a file locally.

Basic encryption in transit is configured by default, but does not apply to the TN3270 protocol. AWS Mainframe Modernization uses HTTPS for API endpoints, which are also configured by default.

Identity and Access Management for AWS Mainframe Modernization

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use AWS Mainframe Modernization resources. IAM is an AWS service that you can use with no additional charge.

Topics
• Audience (p. 377)
• Authenticating with identities (p. 378)
• Managing access using policies (p. 380)
• How AWS Mainframe Modernization works with IAM (p. 381)
• Identity-based policy examples for AWS Mainframe Modernization (p. 389)
• Troubleshooting AWS Mainframe Modernization identity and access (p. 391)
• Using service-linked roles for Mainframe Modernization (p. 392)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS Mainframe Modernization.

Service user – If you use the AWS Mainframe Modernization service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Mainframe Modernization features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS Mainframe Modernization, see Troubleshooting AWS Mainframe Modernization identity and access (p. 391).
Service administrator – If you're in charge of AWS Mainframe Modernization resources at your company, you probably have full access to AWS Mainframe Modernization. It's your job to determine which AWS Mainframe Modernization features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS Mainframe Modernization, see How AWS Mainframe Modernization works with IAM (p. 381).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Mainframe Modernization. To view example AWS Mainframe Modernization identity-based policies that you can use in IAM, see Identity-based policy examples for AWS Mainframe Modernization (p. 389).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the AWS Sign-In User Guide.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the IAM User Guide.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the AWS IAM Identity Center User Guide and Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the IAM User Guide.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A federated identity is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.
For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the AWS IAM Identity Center User Guide.

IAM users and groups

An **IAM user** is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the IAM User Guide.

An **IAM group** is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the IAM User Guide.

IAM roles

An **IAM role** is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the IAM User Guide. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the AWS IAM Identity Center User Guide.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the IAM User Guide.

- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional
Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the IAM User Guide.

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the iam:GetRole action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the IAM User Guide.

Identity-based policies can be further categorized as inline policies or managed policies. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the IAM User Guide.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-
based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

**Access control lists (ACLs)**

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the Amazon Simple Storage Service Developer Guide.

**Other policy types**

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the IAM User Guide.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the AWS Organizations User Guide.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the IAM User Guide.

**Multiple policy types**

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the IAM User Guide.

**How AWS Mainframe Modernization works with IAM**

Before you use IAM to manage access to AWS Mainframe Modernization, learn what IAM features are available to use with AWS Mainframe Modernization.
IAM features you can use with AWS Mainframe Modernization

<table>
<thead>
<tr>
<th>IAM feature</th>
<th>AWS Mainframe Modernization support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity-based policies (p. 382)</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource-based policies (p. 382)</td>
<td>No</td>
</tr>
<tr>
<td>Policy actions (p. 383)</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy resources (p. 383)</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy condition keys (p. 387)</td>
<td>Yes</td>
</tr>
<tr>
<td>ACLs (p. 387)</td>
<td>No</td>
</tr>
<tr>
<td>ABAC (tags in policies) (p. 388)</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporary credentials (p. 388)</td>
<td>Yes</td>
</tr>
<tr>
<td>Principal permissions (p. 388)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service roles (p. 389)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service-linked roles (p. 389)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

To get a high-level view of how AWS Mainframe Modernization and other AWS services work with most IAM features, see AWS services that work with IAM in the IAM User Guide.

Identity-based policies for AWS Mainframe Modernization

Supports identity-based policies Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the IAM User Guide.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can’t specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the IAM User Guide.

Identity-based policy examples for AWS Mainframe Modernization

To view examples of AWS Mainframe Modernization identity-based policies, see Identity-based policy examples for AWS Mainframe Modernization (p. 389).

Resource-based policies within AWS Mainframe Modernization

Supports resource-based policies No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the
resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the IAM User Guide.

Policy actions for AWS Mainframe Modernization

<table>
<thead>
<tr>
<th>Supports policy actions</th>
<th>Yes</th>
</tr>
</thead>
</table>

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as permission-only actions that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called dependent actions.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS Mainframe Modernization actions, see Actions Defined by AWS Mainframe Modernization in the Service Authorization Reference.

Policy actions in AWS Mainframe Modernization use the following prefix before the action:

m2

To specify multiple actions in a single statement, separate them with commas.

"Action": [
  "m2:StartApplication",
  "m2:StopApplication"
]

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word List, include the following action:

"Action": "m2:List*"

To view examples of AWS Mainframe Modernization identity-based policies, see Identity-based policy examples for AWS Mainframe Modernization (p. 389).

Policy resources for AWS Mainframe Modernization

<table>
<thead>
<tr>
<th>Supports policy resources</th>
<th>Yes</th>
</tr>
</thead>
</table>
Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name (ARN)](https://aws.amazon.com/documentation/). You can do this for actions that support a specific resource type, known as **resource-level permissions**.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*
```

You can restrict access to specific AWS Mainframe Modernization resources by using their ARNs to identify the resource that the IAM policy applies to. For more information about the format of ARNs, see [Amazon Resource Names (ARNs)](https://aws.amazon.com/documentation/arn) in the [AWS General Reference](https://aws.amazon.com/documentation/).

For example, an AWS Mainframe Modernization environment has the following ARN.

```
"Resource": "arn:aws:m2:regionId:accountId:env/service-generated-unique-identifier"
```

An AWS Mainframe Modernization application has the following ARN.

```
"Resource": "arn:aws:m2:regionId:accountId:app/service-generated-unique-identifier"
```

Not all AWS Mainframe Modernization actions support resource-level permissions. For actions that don't support resource-level permissions, you must use the wildcard (*).

The following AWS Mainframe Modernization actions do not support resource-level permissions.

```
ListApplications
ListApplicationVersions
ListBatchJobDefinitions
ListBatchJobExecutions
ListDataSetImportHistory
ListDataSets
ListDeployments
ListEngineVersions
ListEnvironments
ListTagsForResource
```

To see a list of AWS Mainframe Modernization resource types and their ARNs, see [Resources Defined by AWS Mainframe Modernization](https://aws.amazon.com/documentation/) in the [Service Authorization Reference](https://aws.amazon.com/documentation/). To learn with which actions you can specify the ARN of each resource, see [Actions Defined by AWS Mainframe Modernization](https://aws.amazon.com/documentation/).

To view examples of AWS Mainframe Modernization identity-based policies, see [Identity-based policy examples for AWS Mainframe Modernization](https://docs.aws.amazon.com/).
• Each AWS Mainframe Modernization API operation
• The corresponding actions for which you can grant permissions to perform the action
• The AWS resource for which you can grant the permissions

You specify the actions in the policy's Action field and the resource value in the policy's Resource field.

You can use AWS global condition keys in your AWS Mainframe Modernization policies to express conditions. For a complete list of AWS keys, see Available Global Condition Keys in the IAM User Guide.

**Note**
To specify an action, use the `m2:` prefix followed by the API operation name (for example, `m2:CreateApplication`).

### AWS Mainframe Modernization API and required permissions for actions

<table>
<thead>
<tr>
<th>AWS Mainframe Modernization API Operations</th>
<th>Required Permissions (API Actions)</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>CancelBatchJobExecution</td>
<td>s3:GetObject</td>
<td>Application</td>
</tr>
<tr>
<td>CreateApplication</td>
<td>s3:ListBucket</td>
<td>Application</td>
</tr>
<tr>
<td>CreateDataSetImportTask</td>
<td>m2:CreateDataSetImportTask</td>
<td>Application</td>
</tr>
<tr>
<td>CreateDeployment</td>
<td>elasticloadbalancing:AddTags</td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>elasticloadbalancing:CreateListener</td>
<td></td>
</tr>
<tr>
<td></td>
<td>elasticloadbalancing:CreateTargetGroup</td>
<td></td>
</tr>
<tr>
<td></td>
<td>elasticloadbalancing:RegisterTargets</td>
<td></td>
</tr>
<tr>
<td>CreateEnvironment</td>
<td>ec2:CreateNetworkInterface</td>
<td>Environment</td>
</tr>
<tr>
<td></td>
<td>ec2:CreateNetworkInterfacePermission</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ec2:DescribeNetworkInterfaces</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ec2:DescribeSecurityGroups</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ec2:DescribeSubnets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ec2:DescribeVpcAttribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ec2:DescribeVpcs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ec2:ModifyNetworkInterfaceAttribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>elasticfilesystem:DescribeMountTargets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>elasticloadbalancing:AddTags</td>
<td></td>
</tr>
<tr>
<td></td>
<td>elasticloadbalancing:CreateLoadBalancer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fsx:DescribeFileSystems</td>
<td></td>
</tr>
</tbody>
</table>
### AWS Mainframe Modernization API Operations

<table>
<thead>
<tr>
<th>Required Permissions (API Actions)</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>iam:CreateServiceLinkedRole</td>
<td></td>
</tr>
<tr>
<td>DeleteApplication</td>
<td>elasticloadbalancing:DeleteListener, Application elasticloadbalancing:DeleteTargetGroup logs:DeleteLogDelivery</td>
</tr>
<tr>
<td>DeleteApplicationFromEnvironment</td>
<td>elasticloadbalancing:DeleteListener, Application elasticloadbalancing:DeleteTargetGroup</td>
</tr>
<tr>
<td>DeleteEnvironment</td>
<td>elasticloadbalancing:DeleteLoadBalancer</td>
</tr>
<tr>
<td>GetApplication</td>
<td>Application</td>
</tr>
<tr>
<td>GetApplicationVersion</td>
<td>Application</td>
</tr>
<tr>
<td>GetBatchJobExecution</td>
<td>Application</td>
</tr>
<tr>
<td>GetDataSetDetails</td>
<td>Application</td>
</tr>
<tr>
<td>GetDataSetImportTask</td>
<td>Application</td>
</tr>
<tr>
<td>GetDeployment</td>
<td>Application</td>
</tr>
<tr>
<td>GetEnvironment</td>
<td>Environment</td>
</tr>
<tr>
<td>ListApplications</td>
<td>*</td>
</tr>
<tr>
<td>ListApplicationVersions</td>
<td>*</td>
</tr>
<tr>
<td>ListBatchJobDefinitions</td>
<td>*</td>
</tr>
<tr>
<td>ListBatchJobExecutions</td>
<td>*</td>
</tr>
<tr>
<td>ListDataSetImportHistory</td>
<td>*</td>
</tr>
<tr>
<td>ListDataSets</td>
<td>*</td>
</tr>
<tr>
<td>ListDeployments</td>
<td>*</td>
</tr>
<tr>
<td>ListEngineVersions</td>
<td>*</td>
</tr>
<tr>
<td>ListEnvironments</td>
<td>*</td>
</tr>
<tr>
<td>ListTagsForResource</td>
<td>*</td>
</tr>
<tr>
<td>StartApplication</td>
<td>Application</td>
</tr>
<tr>
<td>StartBatchJob</td>
<td>Application</td>
</tr>
<tr>
<td>StopApplication</td>
<td>Application</td>
</tr>
<tr>
<td>TagResource</td>
<td>*</td>
</tr>
<tr>
<td>UntagResource</td>
<td>*</td>
</tr>
</tbody>
</table>
How AWS Mainframe Modernization works with IAM

<table>
<thead>
<tr>
<th>AWS Mainframe Modernization API Operations</th>
<th>Required Permissions (API Actions)</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateApplication</td>
<td>s3:GetObject</td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>s3:ListBucket</td>
<td></td>
</tr>
<tr>
<td>UpdateEnvironment</td>
<td></td>
<td>Environment</td>
</tr>
</tbody>
</table>

### Policy condition keys for AWS Mainframe Modernization

<table>
<thead>
<tr>
<th>Supports service-specific policy condition keys</th>
<th>Yes</th>
</tr>
</thead>
</table>

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use **condition operators**, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical **AND** operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical **OR** operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the IAM User Guide.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the IAM User Guide.

The following condition keys are specific to AWS Mainframe Modernization:

- m2:EngineType
- m2:InstanceType

To see a list of AWS Mainframe Modernization condition keys, see Condition Keys for AWS Mainframe Modernization in the Service Authorization Reference. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS Mainframe Modernization.

To view examples of AWS Mainframe Modernization identity-based policies, see Identity-based policy examples for AWS Mainframe Modernization (p. 389).

### Access control lists (ACLs) in AWS Mainframe Modernization

<table>
<thead>
<tr>
<th>Supports ACLs</th>
<th>No</th>
</tr>
</thead>
</table>

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.
Attribute-based access control (ABAC) with AWS Mainframe Modernization

| Supports ABAC (tags in policies) | Yes |

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called tags. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is Yes for the service. If a service supports all three condition keys for only some resource types, then the value is Partial.

For more information about ABAC, see What is ABAC? in the IAM User Guide. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the IAM User Guide.

Using Temporary credentials with AWS Mainframe Modernization

| Supports temporary credentials | Yes |

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the IAM User Guide.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the IAM User Guide.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for AWS Mainframe Modernization

| Supports principal permissions | Yes |

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both
actions. To see whether an action requires additional dependent actions in a policy, see AWS Mainframe Modernization in the *Service Authorization Reference*.

**Important**
These tokens give AWS Mainframe Modernization access to customer data without your explicit agreement; for example, AWS Mainframe Modernization deploys application artifacts with associated business data from an Amazon S3 bucket without obtaining explicit permission from the customer. You might need to update any compliance documentation accordingly.

**Service roles for AWS Mainframe Modernization**

<table>
<thead>
<tr>
<th>Supports service roles</th>
<th>Yes</th>
</tr>
</thead>
</table>

A service role is an **IAM role** that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](https://docs.aws.amazon.com/iam/latest/userguide/creating-service-role.html) in the *IAM User Guide*.

AWS Mainframe Modernization supports service roles for activity hooks (transaction / jobs abends or completion, etc).

**Warning**
Changing the permissions for a service role might break AWS Mainframe Modernization functionality. Edit service roles only when AWS Mainframe Modernization provides guidance to do so.

**Choosing an IAM role in AWS Mainframe Modernization**

If you have previously created an IAM role that your applications running on Amazon EC2 can assume, you can choose this role when you create a launch template or launch configuration. AWS Mainframe Modernization provides you with a list of roles to choose from. When creating these roles, it's important to associate least privilege IAM policies that restrict access to the specific API calls that the application requires. For more information, see [IAM role for applications that run on Amazon EC2 instances](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-service-role-application.html) in the *Amazon EC2 Auto Scaling User Guide*.

**Service-linked roles for AWS Mainframe Modernization**

<table>
<thead>
<tr>
<th>Supports service-linked roles</th>
<th>Yes</th>
</tr>
</thead>
</table>

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing AWS Mainframe Modernization service-linked roles, see [Using service-linked roles for Mainframe Modernization](https://docs.aws.amazon.com/mainframe-modernization/latest/userguide/service-linked-roles.html) on page 392.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](https://docs.aws.amazon.com/IAM/latest/UserGuide/latest/servicelinked-roles.html). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the Yes link to view the service-linked role documentation for that service.

**Identity-based policy examples for AWS Mainframe Modernization**

By default, users and roles don't have permission to create or modify AWS Mainframe Modernization resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they
need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the IAM User Guide.

For details about actions and resource types defined by AWS Mainframe Modernization, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for AWS Mainframe Modernization in the Service Authorization Reference.

Topics
- Policy best practices (p. 390)
- Using the AWS Mainframe Modernization console (p. 390)
- Allow users to view their own permissions (p. 391)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS Mainframe Modernization resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- Get started with AWS managed policies and move toward least-privilege permissions – To get started granting permissions to your users and workloads, use the AWS managed policies that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the IAM User Guide.

- Apply least-privilege permissions – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as least-privilege permissions. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the IAM User Guide.

- Use conditions in IAM policies to further restrict access – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the IAM User Guide.

- Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the IAM User Guide.

- Require multi-factor authentication (MFA) – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the IAM User Guide.

For more information about best practices in IAM, see Security best practices in IAM in the IAM User Guide.

Using the AWS Mainframe Modernization console

To access the AWS Mainframe Modernization console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS Mainframe Modernization
resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS Mainframe Modernization console, also attach the AWS Mainframe Modernization ConsoleAccess or ReadOnly AWS managed policy to the entities. For more information, see Adding permissions to a user in the IAM User Guide.

**Allow users to view their own permissions**

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```json
{
  "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": "*"
    }
  ]
}
```

**Troubleshooting AWS Mainframe Modernization identity and access**

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS Mainframe Modernization and IAM.
I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS Mainframe Modernization.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS Mainframe Modernization. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS Mainframe Modernization resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS Mainframe Modernization supports these features, see [How AWS Mainframe Modernization works with IAM](p. 381).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](p. 381) in the IAM User Guide.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](p. 381) in the IAM User Guide.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users (identity federation)](p. 381) in the IAM User Guide.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](p. 381) in the IAM User Guide.

Using service-linked roles for Mainframe Modernization

AWS Mainframe Modernization uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Mainframe Modernization.
Service-linked roles are predefined by Mainframe Modernization and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Mainframe Modernization easier because you don't have to manually add the necessary permissions. Mainframe Modernization defines the permissions of its service-linked roles, and unless defined otherwise, only Mainframe Modernization can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Mainframe Modernization resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have Yes in the Service-linked roles column. Choose a Yes with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Mainframe Modernization

Mainframe Modernization uses the service-linked role named AWSServiceRoleForAWSM2 – configure the network to connect to your VPC and access resources such as file systems.

The AWSServiceRoleForAWSM2 service-linked role trusts the following services to assume the role:

- m2.amazonaws.com

The role permissions policy named AWSM2ServicePolicy allows Mainframe Modernization to complete the following actions on the specified resources:

- Create, delete, describe, and attach permissions to Amazon EC2 network interfaces for the Mainframe Modernization environment to establish connectivity to the customer VPC.
- Register or de-register entries from Elastic Load Balancing, which is how customers connect to the Mainframe Modernization environment.
- Describe the Amazon EFS or Amazon FSx file system, if used.
- Emit metrics to the customer's CloudWatch from the runtime environment.

```json
{
"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": "*"
}
]
```
You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the IAM User Guide.

Creating a service-linked role for Mainframe Modernization

You don't need to manually create a service-linked role. When you create a runtime environment in the AWS Management Console, the AWS CLI, or the AWS API, Mainframe Modernization creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a runtime environment, Mainframe Modernization creates the service-linked role for you again.

Editing a service-linked role for Mainframe Modernization

Mainframe Modernization does not allow you to edit the AWSServiceRoleForAWSM2 service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role because various entities might reference the role. For more information, see Editing a service-linked role in the IAM User Guide.

Deleting a service-linked role for Mainframe Modernization

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.
Note
If the Mainframe Modernization service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Mainframe Modernization resources used by the AWSServiceRoleForAWSM2

- Delete the runtime environments in Mainframe Modernization. Make sure to delete applications from an environment before deleting the environment itself.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAWSM2 service-linked role. For more information, see Deleting a service-linked role in the IAM User Guide.

Supported regions for Mainframe Modernization service-linked roles

Mainframe Modernization supports using service-linked roles in all of the regions where the service is available. For more information, see AWS regions and endpoints.

Compliance validation for AWS Mainframe Modernization

Third-party auditors assess the security and compliance of AWS Mainframe Modernization as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS Mainframe Modernization is determined by the sensitivity of your data, your company’s compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the AWS Config Developer Guide – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in AWS Mainframe Modernization

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency,
high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

### Infrastructure security in AWS Mainframe Modernization

As a managed service, AWS Mainframe Modernization is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

You use AWS published API calls to access Mainframe Modernization through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

### Tutorial: AWS Blu Age Application Security in AWS Mainframe Modernization

This tutorial documents AWS Mainframe Modernization application authentication through Amazon Cognito.

#### Amazon Cognito user pools

AWS Mainframe Modernization-deployed AWS Blu Age applications can be configured to use Amazon Cognito user pools as an identity provider. Amazon Cognito will then store users and their attributes and handle their login process. Password policies, Multi Factor Authentication and customizable email/phone number messages are available. AWS Blu Age applications security is based on SpringSecurity; its OAuth2 features are used to interface with Amazon Cognito. For more information on Amazon Cognito user pools, see [Amazon Cognito user pools](#) in the Amazon Cognito Developer Guide.

#### Third-party identity providers

Amazon Cognito can delegate authentication to an existing, third party identity provider through the Federated Identity Providers feature. This article [Role-based access control using Amazon Cognito and an external identity provider](#) provides an example of such setup.
Example setup

Topics

• Demo Resources (p. 397)
• Domain registration (p. 397)
• TLS certificate emission (p. 397)
• Amazon Cognito Setup (p. 398)
• Secret Manager (p. 401)
• User Creation (p. 402)
• Demo App Upload (p. 403)
• Application Resources Creation (p. 403)
• Application Definition Adjustments (p. 403)
• AWS Mainframe Modernization Environment Creation (p. 404)
• AWS Mainframe Modernization Application Creation (p. 404)
• AWS Mainframe Modernization Application Deployment (p. 405)
• AWS Mainframe Modernization Application Startup (p. 405)
• Accessing the Application (p. 405)

The following sections will document the deployment of an AWS Mainframe Modernization AWS Blu Age application delegating its authentication to Amazon Cognito.

Demo Resources

This demo application archive will be needed in the following steps: PlanetsDemo-v1.zip.

Domain registration

Amazon Cognito requires the integrating ("client") application to be exposed through an HTTPS-secure URL. For this purpose, we will need a domain name, and a TLS certificate for this domain.

The combination of Route 53 and AWS Certificate Manager provides us with an integrated solution for this task.

First navigate to the Route 53 console. There, choose Registered domains, then choose Register Domain.

Use Choose a domain name to pick an available one. On the following page, enter your contact details then choose Add to Cart. Do a final review of your choices, then choose Complete Order.

You may need to answer a validation email, then wait for the domain registration to take place.

TLS certificate emission

In this step, you request a certificate for the domain you registered previously.

1. When the previously created domain name is available, navigate to ACM in the AWS Management Console.
2. Choose Request a certificate, and in Certificate type, choose Request a public certificate.
3. Choose **Next**.

4. On **Request public certificate**, enter the domain name you previously registered and leave **DNS validation** selected.

You should then get a **Successfully requested certificate** message.

5. Choose **View Certificate**, and notice that its current status is **Pending validation**. Since it relates to a Route 53 domain, you can use the console to create the necessary DNS-validation records. For more information, see [Validating domain ownership](#) in the **AWS Certificate Manager User Guide**. For this, choose **Create records in Route 53 in Domains**, then choose **Create records**.

You should get a **Successfully created DNS records** message. Wait until the certificate gets in the **Issue** state.

**Amazon Cognito Setup**

For testing purposes we will create a user pool managing local user profiles.
1. Navigate to Amazon Cognito in the AWS Management Console. Choose User pools, then Create user pool.

2. In Configure sign-in experience, keep the Cognito user pool default provider type. You can choose one or multiple Cognito user pool sign-in options; for now, choose User name, then choose Next.

3. In Configure security requirements, either keep the defaults or adjust passwords and login requirements as appropriate. Choose Next.

4. In Configure sign-up experience, since this is a demo app, disable Enable self-registration as a security measure and choose Next.

5. In Configure message delivery, you can use Send email with Amazon SES if you already have an Amazon SES identity existing in your account. Otherwise choose Send email with Cognito for testing purposes. You can add a valid REPLY-TO email address if you like. Choose Next.
6. In **Integrate your app**, choose a name for your user pool. In **Hosted authentication pages**, choose **Use the Cognito Hosted UI**:

7. For simplicity’s sake, in **Domain**, choose **Use a Cognito domain** and enter a domain prefix; for example, `https://planetsdemo`.

8. The demo app must be added as a client. In **Initial app client**, choose **Confidential client**. Enter an app client name, such as `planetsdemo` and choose **Generate a client secret**.
9. In **Allowed callback URL** use an URL with the following canonical format:

```
https://{m2-address}/{app-name}:{app-port}/login/oauth2/code/cognito
```

Where:

- **m2-address** is the DNS name of the AWS Mainframe Modernization entry point (use the domain specified in the certificate registered earlier).
- **app-name** is the name of the application deployed on AWS Mainframe Modernization (*-web or *-service typically).
- **app-port** is the (tomcat) port where this application is deployed (as specified below in the application definition).

You can also use a temporary http://localhost URL, then edit it later.

Keep default values in the **Advanced app client settings** and **Attribute read and write permissions** sections, then choose **Next**.

10. In **Review and create**, verify your choices then choose **Create user pool**.

**Secret Manager**

A secret manager is needed to host the client secret generated by Amazon Cognito during the previous step. For this purpose use AWS Secrets Manager.

1. Navigate to Secrets Manager in the AWS Management Console.
2. Choose **Store a new secret**.
3. In **Secret type**, choose **Other type of secret**, and enter **client-secret** for the key. For the value, copy and paste the client secret generated by Amazon Cognito. You can find this value at Cognito > User pool > App client > Show client secret. Choose **Next**.
4. In **Secret name and description**, give the secret a name, then choose **Next**.
5. In **Secret rotation**, leave **Automatic rotation** disabled. Choose **Next**.
6. In **Review**, look over the information you provided, then choose **Store**.
7. After the secret is created, you must enable access to it from AWS Mainframe Modernization. To do this, choose the secret, then choose **Edit permissions**:

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

8. In the displayed JSON editor, paste the following policy and choose **Save**:

**User Creation**

Since self-registration is disabled, create a Amazon Cognito user.

1. Navigate to Amazon Cognito in the AWS Management Console.
2. Choose the user pool you created, then in **Users** choose **Create user**.
3. In User information, choose **Send an email invitation**, enter a user name and an email address, and choose **Generate a password**. Choose **Create user**.

![User information form](image)

**Demo App Upload**

Upload the demo application to an Amazon S3 bucket; here, a bucket named `planetsdemo` will be used, with a key prefix (folder) of name `v1` and an archive name of `planetsdemo-v1.zip`.

**Application Resources Creation**

This demo application does not need extra AWS resources.

**Application Definition Adjustments**

The following is an AWS Mainframe Modernization application definition for the demo application:

```json
{
  "resources": [
    {
      "resource-type": "listener",
      "resource-id": "tomcat",
      "properties": {
        "port": 8196,
        "type": "http"
      }
    },
    {
      "resource-type": "ba-application",
      "resource-id": "planetsdemo",
      "properties": {
        "app-location": "${s3-source}/PlanetsDemo-v1.zip"
      }
    }
  ]
}
```
Replace the following placeholders as described; needed information can be found on the App integration tab of your previously created Amazon Cognito user pool:

- `{REGION_ID}`: the region where the application is to be deployed.
- `{POOL_ID}`: the Amazon Cognito Pool ID.
- `{DOMAIN_NAME}`: the full URL of the Amazon Cognito domain created during the “integrate your app” step.
- `{CLIENT_ID}`: the Client ID value created for your application by Amazon Cognito.
- `{SECRET_MANAGER_ARN}`: the ARN of the previously created secret in Secrets Manager.

Adjust the `ba-application` and `s3-source` entries as needed to point to the Amazon S3 bucket where the application was uploaded.

**AWS Mainframe Modernization Environment Creation**

Create the AWS Mainframe Modernization runtime environment by following these steps:

1. Navigate to AWS Mainframe Modernization in the AWS Management Console.
2. Choose Environments, the choose Create environment.
3. In Specify basic information, enter a name and description, then choose the AWS Blu Age engine.
4. In Specify configurations:
   - Choose Standalone runtime environment.
   - Choose Allow applications deployed to this environment to be publicly accessible.
   - Choose two subnets.
5. In Attach storage - Optional, leave the default values.
6. In Review and create, review the information, then choose Create environment.

**AWS Mainframe Modernization Application Creation**

1. Navigate to AWS Mainframe Modernization in the AWS Management Console.
2. Choose Applications, then choose Create application.
3. In Specify basic information, enter a name and description, then choose the AWS Blu Age engine.
4. In Specify resources and configurations, copy and paste the previously updated Application Definition JSON.
5. In Review and create, review the information, then choose Create application.

**AWS Mainframe Modernization Application Deployment**

After both the AWS Mainframe Modernization environment and application are created successfully and are both in the Available state:

1. Navigate to AWS Mainframe Modernization in the AWS Management Console.
2. Choose Environments, then choose the previously created environment.
3. Choose Deploy application.
4. Choose the previously created application.
5. Choose Deploy.

**AWS Mainframe Modernization Application Startup**

1. Navigate to AWS Mainframe Modernization in the AWS Management Console.
2. Choose Applications.
3. Choose your application, then scroll to Deployments: it should be in the Ready state.
4. Choose Actions, then choose Start application.

**Accessing the Application**

1. Wait until the application is in the Running state.
2. Copy the application DNS hostname.
3. In a browser, navigate to https://{hostname}:{portname}/PlanetsDemo-web-1.0.0/.

**Clean up resources**

If you no longer need the resources you created for this tutorial, delete them so that you won't continue to be charged for them. Complete the following steps:

- If the AWS Mainframe Modernization application is still running, stop it.
- Delete the AWS Mainframe Modernization application. For more information, see Delete an AWS Mainframe Modernization application (p. 307).
- Delete the AWS Mainframe Modernization runtime environment. For more information, see Delete an AWS Mainframe Modernization runtime environment (p. 355).
- Delete the Amazon Cognito resources you created. For more information, see Tutorial: Cleaning up your AWS resources in the Amazon Cognito Developer Guide.
- Delete the Secrets Manager secret you created. For more information, see Delete a secret in the AWS Secrets Manager User Guide.
Access AWS Mainframe Modernization using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and AWS Mainframe Modernization. You can access Mainframe Modernization as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access Mainframe Modernization.

You establish this private connection by creating an interface endpoint, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Mainframe Modernization.

For more information, see Access AWS services through AWS PrivateLink in the AWS PrivateLink Guide.

Considerations for Mainframe Modernization

Before you set up an interface endpoint for Mainframe Modernization, review Considerations in the AWS PrivateLink Guide.

Mainframe Modernization supports making calls to all of its API actions through the interface endpoint.

Create an interface endpoint for Mainframe Modernization

You can create an interface endpoint for Mainframe Modernization using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the AWS PrivateLink Guide.

Create an interface endpoint for Mainframe Modernization using the following service name:

```
com.amazonaws.region.m2
```

If you enable private DNS for the interface endpoint, you can make API requests to Mainframe Modernization using its default Regional DNS name. For example, m2.us-east-1.amazonaws.com.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Mainframe Modernization through the interface endpoint. To control the access allowed to Mainframe Modernization from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the AWS PrivateLink Guide.
Example: VPC endpoint policy for Mainframe Modernization actions

The following is an example of a custom endpoint policy. When you attach this policy to your interface endpoint, it grants access to the listed Mainframe Modernization actions for all principals on all resources.

```json
//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": "*"
  }
]
```
Monitoring AWS Mainframe Modernization

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Mainframe Modernization and your other AWS solutions. AWS provides the following monitoring tools to watch AWS Mainframe Modernization, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](https://docs.aws.amazon.com/AmazonCloudWatch/latest/userguide/)

- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/)

- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/)

Monitoring AWS Mainframe Modernization with Amazon CloudWatch

You can monitor AWS Mainframe Modernization using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](https://docs.aws.amazon.com/AmazonCloudWatch/latest/userguide/)

The following tables list the metrics and dimensions for AWS Mainframe Modernization. The namespace for these metrics is AWS/M2.

### Runtime Environment Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUUtilization</td>
<td>The CPU utilization of instances in the environment.</td>
</tr>
<tr>
<td></td>
<td>Dimension: environmentId</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
</tbody>
</table>
## Application Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>InboundNetworkThroughput</strong></td>
<td>Inbound network throughput of instances in the environment.</td>
</tr>
<tr>
<td></td>
<td>Dimension: environmentId</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes per second</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>MemoryUtilization</strong></td>
<td>The memory utilization of instances in the environment.</td>
</tr>
<tr>
<td></td>
<td>Dimension: environmentId</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>OutboundNetworkThroughput</strong></td>
<td>Outbound network throughput of the instances in the environment.</td>
</tr>
<tr>
<td></td>
<td>Dimension: environmentId</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes per second</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
</tbody>
</table>

**Application Metrics**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BatchJobCompletedCount</strong></td>
<td>The number of completed jobs during the time interval.</td>
</tr>
<tr>
<td></td>
<td>This metric is available for Micro Focus and for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum</td>
</tr>
<tr>
<td><strong>BatchJobFailedCount</strong></td>
<td>The number of failed jobs during the time interval.</td>
</tr>
<tr>
<td></td>
<td>This metric is available for Micro Focus and for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JvmMemoryFree</td>
<td>The amount of available memory that is not currently in use by the Java Virtual Machine.</td>
</tr>
<tr>
<td></td>
<td>This metric is only available for the AWS Blu Age runtime engine. It is available for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>JvmMemoryMax</td>
<td>The maximum amount of memory allowed for the Java Virtual Machine.</td>
</tr>
<tr>
<td></td>
<td>This metric is only available for the AWS Blu Age runtime engine. It is available for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>JvmMemoryUsed</td>
<td>The amount of memory actively used by the Java Virtual Machine.</td>
</tr>
<tr>
<td></td>
<td>This metric is only available for the AWS Blu Age runtime engine. It is available for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>ProcessesActiveCount</td>
<td>The active number of concurrent service execution processes that are processing requests.</td>
</tr>
<tr>
<td></td>
<td>This metric is only available for the Micro Focus runtime engine.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum</td>
</tr>
</tbody>
</table>
### Application Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionCount</td>
<td>The number of HTTP sessions for the application.</td>
</tr>
<tr>
<td></td>
<td>This metric is only available for the AWS Blu Age runtime engine. It is available for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>SharedMemoryFree</td>
<td>The memory that is available for the enterprise server to store all the information it needs to run transactions and jobs.</td>
</tr>
<tr>
<td></td>
<td>This metric is only available for the Micro Focus runtime engine.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>ThreadActiveCount</td>
<td>The number of engine threads that are processing requests.</td>
</tr>
<tr>
<td></td>
<td>This metric is only available for the AWS Blu Age runtime engine. It is available for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>TransactionCompletedCount</td>
<td>The number of committed transactions during the time interval.</td>
</tr>
<tr>
<td></td>
<td>This metric is available for Micro Focus and for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum</td>
</tr>
</tbody>
</table>


Dimensions

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationId</td>
<td>This dimension filters the metric to the identified application by ID.</td>
</tr>
<tr>
<td>environmentId</td>
<td>This dimension filters the metric to the identified environment by ID.</td>
</tr>
</tbody>
</table>

Logging AWS Mainframe Modernization API calls using AWS CloudTrail

AWS Mainframe Modernization is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Mainframe Modernization. CloudTrail captures all API calls for AWS Mainframe Modernization as events. The calls captured include calls from the AWS Mainframe Modernization console and code calls to the AWS Mainframe Modernization API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Mainframe Modernization. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to AWS Mainframe Modernization, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](https://docs.aws.amazon.com/cLOUDTRAIL/latest/userguide/).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransactionFailedCount</td>
<td>The number of failed transactions during the time interval. This metric is available for Micro Focus and for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum</td>
</tr>
<tr>
<td>TransactionResponseTime</td>
<td>The amount of time from the moment that a user sends a request until the time that the application indicates that the request has been completed.</td>
</tr>
<tr>
<td></td>
<td>This metric is available for Micro Focus and for AWS Blu Age 3.7.0 and later releases.</td>
</tr>
<tr>
<td></td>
<td>Dimension: applicationId</td>
</tr>
<tr>
<td></td>
<td>Units: Milliseconds</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum</td>
</tr>
</tbody>
</table>
AWS Mainframe Modernization information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Mainframe Modernization, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS Mainframe Modernization, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions
- Receiving CloudTrail log files from multiple accounts

All AWS Mainframe Modernization actions are logged by CloudTrail and are documented in the AWS Mainframe Modernization API Reference. For example, calls to the CreateApplication, CreateEnvironment and CreateDeployment actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding AWS Mainframe Modernization log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files don't have an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the CreateApplication action.

```json
{
  "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/lshfmwhw7fffrossf2lncwqcua",
  "applicationId": "lshfmwhw7fffrossf2lncwqcua"
},
"requestID": "36982d38-fcde-4bfe-a89a-7bd78d43c926",
"eventID": "d7f0fc36-46ae-4157-9a79-c79f385fda98",
"readOnly": false,
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "444455556666",
"eventCategory": "Management"}
Troubleshooting

Use the information in this section to help you troubleshoot common errors in AWS Mainframe Modernization applications and runtime environments using both the AWS Blu Age and Micro Focus engines.

Topics

- Error: Time out while waiting for dataset name to be unlocked (p. 415)
- Cannot access an application's URL (p. 417)
- AWS Blu Insights doesn't open from the console (p. 418)

Error: Time out while waiting for dataset name to be unlocked

- Engine: AWS Blu Age
- Component: Blusam

If you see this error in the Amazon CloudWatch logs for a AWS Mainframe Modernization application using the AWS Blu Age engine and running in an environment with the High Availability pattern, it indicates that another application is holding a lock on a shared dataset. Typically, this situation occurs if the other application crashes or otherwise fails and does not release the lock.

How this error occurs

Application example-app-1 tries to lock a record example-record-1 for a write operation. This operation creates both a lock on dataset example-dataset-1, which owns example-record-1, and a lock on example-record-1 itself. Now another application, example-app-2, tries to lock the same record example-record-1. The dataset and the record are already locked, so example-app-2 waits for the lock to release. If example-app-1 crashes, the held lock on dataset example-dataset-1 still exists, which causes example-app-2 to cancel its write attempt and raise a timeout exception. This deadlock situation prevents all applications from reaching example-dataset-1.

How do you know if this is your situation?

Look for a failed application and check whether it uses the same dataset mentioned in the error message. Check whether the application is running in a runtime environment with the High Availability pattern. The application that raised the timeout exception cannot proceed and will display the Failed status.

What can you do?

To resolve the situation immediately, you can force the lock to release. To prevent a similar situation from occurring in the future, you can configure two parameters that control the Blusam auto repairing mechanism.
**Force the lock to release**

The Blusam lock manager uses Amazon ElastiCache for Redis to provide shared locks between applications. To release locks in ElastiCache, use the Redis CLI utility. You cannot delete an individual record lock. You must remove all locks from the owning dataset. Complete the following steps:

1. Connect to your ElastiCache using the following command:

   ```
   redis-cli -h hostname -p port
   ```

   You can find the details of your ElastiCache in the ElastiCache console at https://console.aws.amazon.com/elasticache/.

2. Enter your password.

3. Enter the command you want to run, as follows:

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYS *</td>
<td>Get all existing keys.</td>
</tr>
<tr>
<td>KEYS *YOUR_DATASET_NAME</td>
<td>Get a dataset lock key.</td>
</tr>
<tr>
<td>DEL THE_RETURNED_KEY</td>
<td>Delete a dataset lock.</td>
</tr>
<tr>
<td>FLUSHDB</td>
<td>Clean the entire Redis.</td>
</tr>
</tbody>
</table>

   **Warning**
   All data in the Redis cache will be lost. If the Redis is used for other purposes, such as handling http sessions, you might not want to use FLUSHDB.

**Configure the Blusam auto repairing mechanism**

The Blusam locks manager includes an auto repairing mechanism to prevent deadlocks on datasets or records. You can adjust the following parameters in the application definition (application-main.yml) to configure the auto repairing mechanism:

- **locksDeadTime**: refers to the maximum time an application can hold a lock. When this time passes, the lock is declared expired and released immediately. The `locksDeadTime` value is in milliseconds, and the default value is 1000.

- **locksCheck**: defines the Blusam locks manager strategy for checking locks. All Blusam locks in ElastiCache are timestamped and have an expiration time. The `locksCheck` parameter value determines whether expired locks are removed.
  - **off**: no check is executed at any time. Deadlocks might occur. (Not recommended)
  - **reboot**: checks are executed when an AWS Mainframe Modernization application instance running in an AWS Mainframe Modernization runtime environment is started or rebooted. All expired locks are released immediately. (Default)
  - **timeout**: checks are executed when an AWS Mainframe Modernization application instance running in an AWS Mainframe Modernization runtime environment is started or rebooted, or when a timeout expires during an attempt to lock a dataset. Expired locks are released immediately.

For more information on the application definition for a AWS Blu Age application, see [AWS Blu Age application definition sample (p. 335)](#).
Blusam locks manager

In the context of an AWS Mainframe Modernization runtime environment using the High Availability pattern, a AWS Blu Age application might be deployed multiple times. For those applications that handle Blusam data sets, concurrent access problems might occur. The Blusam locks manager ensures data integrity and manages read and write access to records and data sets by providing shared locks between applications using ElastiCache. This mechanism allows more than one application to read the record concurrently, and ensures that only one application at a time writes the record.

Write locks

To update or delete a specific record, the application must first lock the dataset that owns the record, then lock the record itself. When the record is locked, the dataset lock is released, and other records from the same dataset are available for use. When the update or delete operation is complete, the held record lock is released. Only one application at a time can update the record, which blocks other applications from either reading or writing until the lock is released, if the defined application policy allows waiting for release.

Read locks

As long as no write lock is held on the record or the dataset, multiple applications can read the same records at the same time. To lock a record for a write operation, all read locks must be released.

Note

The Blusam locks manager handles the access from multiple threads in a given application using the same locking mechanism.

Cannot access an application's URL

- Engine: AWS Blu Age and Micro Focus
- Component: applications

If you cannot access the URL for a running AWS Mainframe Modernization application that you created and deployed to an AWS Mainframe Modernization runtime environment, you might need to configure the inbound rules on the security group that you associated with the runtime environment.

How this error occurs

When you create a runtime environment, the security group you provide, including the default security group, must have inbound rules configured to allow traffic to the deployed applications from outside the VPC, if you want to allow this type of access.

How do you know if this is your situation?

The application started successfully and is running normally, but you are unable to connect to it using its URL.

What can you do?

Check whether the Amazon VPC security group associated with the runtime environment allows traffic to the environment on the appropriate application ports. To check the security group rules, complete the following steps:
AWS Blu Insights doesn't open from the console

- Engine: AWS Blu Age
- Component: Blu Insights

When you try to access Blu Insights from the AWS Mainframe Modernization console, it does not open and the new tab is closed immediately.

How this error occurs

The role you are using to access Blu Insights does not have sufficient permissions.

What can you do?

Attach an IAM policy to the role to allow it to access Blu Insights. Make sure the policy includes at least the following permissions.

```json
{
   "Version": "2012-10-17",
   "Statement": [
   {
      "Effect": "Allow",
      "Action": [
      "m2:GetSignedBluinsightsUrl"
      ],
      "Resource": "arn:aws:m2:region:account:*"
   }
   ]
}
```
Make sure to replace region and account with the correct AWS Region and AWS account.
## Document history for the AWS Mainframe Modernization User Guide

The following table describes the documentation releases for AWS Mainframe Modernization.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage transactions for applications</td>
<td>New feature released to display and edit transactions for applications for AWS Mainframe Modernization.</td>
<td>October 16, 2023</td>
</tr>
<tr>
<td>Release notes for AWS Blu Age Runtime and Modernization Tools version 3.6.0</td>
<td>This release of AWS Blu Age Runtime and Modernization Tools provides new features for both zOS and AS400 legacy migrations, mainly oriented to expanding CICS support mechanisms, complementing JCL capabilities, optimizing performance in concurrent and high-volume features, and adding multi-data-source capabilities.</td>
<td>August 4, 2023</td>
</tr>
<tr>
<td>You can now deploy a new version of an application when the application is stopped.</td>
<td>Previously, to deploy a new version of an application, you had to delete the deployed version. Now you can just stop the deployed version and deploy a new version.</td>
<td>July 26, 2023</td>
</tr>
<tr>
<td>AWS Blu Age runtime packaged for easier Amazon EC2 deployment (p. 420)</td>
<td>AWS Mainframe Modernization with AWS Blu Age runtime is now available with more flexibility for configuring the complete stack and deployment on Amazon EC2 instances in your AWS account.</td>
<td>July 6, 2023</td>
</tr>
<tr>
<td>Single sign-on to AWSAWS Blu Age Blu Insights. (p. 420)</td>
<td>AWS AWS Blu Age Blu Insights available from the AWS Management Console through single sign-on.</td>
<td>March 31, 2023</td>
</tr>
<tr>
<td>GA release (p. 420)</td>
<td>GA release of the AWS Mainframe Modernization User Guide.</td>
<td>June 8, 2022</td>
</tr>
<tr>
<td>Initial release (p. 420)</td>
<td>Initial release (public preview) of the AWS Mainframe Modernization User Guide.</td>
<td>November 30, 2021</td>
</tr>
</tbody>
</table>