aws

User Guide

AWS Mainframe Modernization



Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Mainframe Modernization: User Guide

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS Mainframe Modernization 1
Features of AWS Mainframe Modernization 2
Patterns 3
How to get started with AWS Mainframe Modernization 3
Related services 4
Accessing AWS Mainframe Modernization 5
Are you a first-time AWS Mainframe Modernization user? 5
Pricing for AWS Mainframe Modernization 5
Set up for AWS Mainframe Modernization 6
Sign up for an AWS account 6
Create a user with administrative access 6
Concepts 9
Application 9
Application definition 9
Batch job 10
Configuration 11
Data set 11
Environment 11
Mainframe modernization 11
Migration journey 11
Mount point 11
Automated Refactoring 11
Replatforming 12
Resource 12
Runtime engine 12
Modernization approach 13
Assess phase 13
Mobilize phase 13
Migrate and modernize phase 14
Operate and optimize phase 14
Get started 15
Tutorial: Set up managed runtime for AWS Blu Age 15
Prerequisites 16
Step 1: Upload the demo application 16

Step 2: Create the application definition ... 16
Step 3: Create a runtime environment ... 17
Step 4: Create an application ... 21
Step 5: Deploy an application ... 23
Step 6: Start an application ... 25
Step 7: Access the application ... 25
Step 8: Test the application ... 26
Clean up resources ... 28
Tutorial: Set up managed runtime for Rocket Software ... 28
Prerequisites ... 29
Step 1: Create and load an Amazon S3 bucket ... 29
Step 2: Create and configure a database ... 31
Step 3: Create and configure an AWS KMS key ... 33
Step 4: Create and configure an AWS Secrets Manager database secret ... 34
Step 5: Add the sslMode to the secret ... 35
Step 6: Create a runtime environment ... 35
Step 7: Create an application ... 40
Step 8: Deploy an application ... 46
Step 9: Import data sets ... 48
Step 10: Start an application ... 54
Step 11: Connect to the CardDemo CICS application ... 55
Clean up resources ... 62
Next steps ... 63
Components lifecycle ... 64
Components lifecycle overview ... 64
Version upgrade ... 65
AWS Mainframe Modernization Refactor with AWS Blu Age release overview ... 66
Managed applications ... 68
Create AWS resources for a migrated application ... 69
Required permissions ... 69
Amazon S3 bucket ... 69
Database ... 70
AWS Key Management Service key ... 71
AWS Secrets Manager secret ... 71
Create an application ... 72
Create an application ... 72

Deploy an application ... 73
Deploy an application ... 73
Update an application ... 74
Update an application ... 74
Delete an application ... 75
Delete an application ... 75
Submit batch jobs for applications ... 76
Submit a batch job ... 76
Restart a batch job ... 77
Cancel batch jobs for applications ... 78
Cancel a batch job ... 79
Import data sets for applications ... 79
Import a data set ... 80
Export data sets for applications ... 80
Export a data set ... 81
Manage transactions for applications ... 81
Manage transactions for applications ... 82
Configure the Rocket Software managed application ... 83
Supported third party integrations for Rocket Software ... 83
Configure the AWS Blu Age managed application ... 85
Structure of AWS Blu Age managed applications ... 86
Configure access to utilities for managed applications ... 87
Configure additional properties for managed application ... 98
Application definition reference ... 119
General header section ... 120
Definition section overview ... 121
AWS Blu Age application definition sample ... 121
AWS Blu Age definition details ... 122
Rocket Software application definition ... 128
Rocket Software definition details ... 130
Data set definition reference ... 140
Common properties ... 141
Sample data set request format for VSAM ... 143
Sample data set request format for GDG base ... 145
Sample data set request format for PS or GDG generations ... 146
Sample data set request format for PO ... 147

Managed runtime environments 149
Create a runtime environment 149
Create a runtime environment 150
Update a runtime environment 153
Update a runtime environment 153
Maintenance window 154
Stop a runtime environment 155
Stop a runtime environment 155
Restart a runtime environment 156
Restart a runtime environment 156
Delete a runtime environment 157
Delete a runtime environment 157
Application Testing 158
What is Application Testing 158
Are you a first-time Application Testing user? 159
Benefits of Application Testing 159
Integration with AWS CloudFormation 160
How Application Testing works 160
Related services 4
Accessing Application Testing 162
Pricing for Application Testing 162
Application Testing concepts 162
Test case 163
Test suite 164
Test environment configuration 164
Upload 164
Replay 164
Compare 165
Database comparisons 165
Dataset comparisons 165
Comparison status 166
Equivalence rules 166
Final-state data set comparison 167
State-progress database comparisons 167
Functional equivalence (FE) 167
Online 3270 screen comparisons 167

Replay data ... 167
Reference data ... 168
Upload, Replay, and Compare ... 168
Differences ... 169
Equivalencies ... 169
Source application ... 169
Target application ... 169
Application Testing prerequisites ... 169
Console workflows in Application Testing ... 170
Create test cases in Application Testing ... 170
Create test suites in Application Testing ... 173
Create test environment configurations in Application Testing ... 175
Tutorial: Set up CardDemo application in Application Testing ... 177
Prerequisites ... 177
Step 1: Prepare to set up CardDemo ... 177
Step 2: Create all necessary resources ... 178
Step 3: Deploy and start the application ... 179
Step 4: Import initial data ... 179
Step 5: Connect to the CardDemo application ... 180
Tutorial: Replay and compare on AWS Blu Age using CardDemo ... 181
Step 1: Obtain AWS Blu Age Amazon EC2 Amazon Machine Image (AMI) ... 181
Step 2: Start an Amazon EC2 instance using the AWS Blu Age AMI ... 181
Step 3: Upload CardDemo dependent files to S3 ... 183
Step 4: Load databases and initialize the CardDemo application ... 183
Step 5: Launch AWS Blu Age runtime CloudFormation ... 186
Step 6: Testing the AWS Blu Age Amazon EC2 instance ... 188
Step 7: Validate previous steps were completed correctly ... 189
Step 8: Create the test case ... 190
Step 9: Create a test suite ... 190
Step 10: Create a test environment configuration ... 191
Step 11: Upload your input data in test suite ... 191
Step 12: Replay and compare ... 192
Supported data sets code pages in Application Testing ... 192
Data protection in Application Testing ... 203
Data collected by the AWS Mainframe Modernization Application Testing ... 204
Data encryption at rest for the AWS Mainframe Modernization Application Testing ... 205

Create a customer managed key ... 206
Specifying a customer managed key for AWS Mainframe Modernization Application Testing ... 207
AWS Mainframe Modernization Application Testing encryption context ... 207
Monitoring your encryption keys ... 208
Encryption in transit ... 208
How Application Testing works with IAM ... 209
Identity-based policies ... 209
Resource-based policies ... 210
Policy actions ... 211
Policy resources ... 211
Policy condition keys ... 213
ACLs ... 214
ABAC ... 214
Temporary credentials ... 215
Forward access sessions ... 215
Service roles ... 216
Service-linked roles ... 216
AWS Blu Age Refactoring ... 217
AWS Blu Age releases ... 218
AWS Blu Age versioning ... 218
AWS Blu Age Runtime options ... 219
AWS Blu Age release notes ... 223
AWS Blu Age security vulnerabilities ... 323
Upgrading AWS Blu Age ... 324
AWS Blu Age lifecycle ... 326
AWS Blu Age Runtime concepts ... 327
High level architecture ... 328
Modernized application structure ... 332
Understand data simplifiers ... 368
AWS Blu Age Blusam ... 376
Available programs in utility web application ... 397
Blusam Administration Console ... 399
AWS Blu Age Runtime configuration ... 438
Application configuration basics ... 439
Application precedence ... 440

JNDI for databases ... 440
AWS Blu Age Runtime secrets ... 441
Other files (groovy, sql, etc.) ... 453
Additional web application ... 454
Enable properties ... 454
Available Redis cache properties ... 527
Configure security for Gapwalk applications ... 543
AWS Blu Age Runtime APIs ... 559
Endpoints for building URLs ... 559
Endpoints for Gapwalk application ... 560
Blusam application console REST endpoints ... 579
Manage JICS application console ... 602
Data structures ... 623
Set up AWS Blu Age Runtime (non-managed) ... 632
AWS Blu Age Runtime prerequisites ... 632
Onboarding AWS Blu Age Runtime ... 633
Infrastructure setup requirements ... 638
AWS Blu Age Runtime artifacts ... 645
Deploy AWS Blu Age Runtime on Amazon EC2 ... 648
Deploy AWS Blu Age Runtime on Amazon ECS and Amazon EKS ... 658
Test the PlanetsDemo application ... 666
Modify the source code with Blu Age Developer IDE ... 670
Tutorial: Set up AppStream 2.0 for AWS Blu Age Developer IDE ... 670
Tutorial: Use AWS Blu Age Developer on AppStream 2.0 ... 675
AWS Blu Age FAQ ... 692
General ... 692
AWS Blu Age Runtime ... 694
Data ... 702
Transformation ... 704
Deployment ... 705
Rocket Software Replatforming ... 708
Set up Rocket Software (on Amazon EC2) ... 708
Rocket Software (on Amazon EC2) prerequisites ... 709
Create the Amazon VPC endpoint for Amazon S3 ... 709
Request the allowlist update for the account ... 712
Create the AWS Identity and Access Management role ... 713

Grant License Manager the required permissions ... 720
Subscribe to the Amazon Machine Images ... 721
Launch a Rocket Software instance ... 725
Subnet or VPC with no internet access ... 731
Set up AppStream 2.0 Automation ... 738
Set up automation at session start ... 739
Set up automation at session end ... 739
View data sets as tables in Enterprise Developer ... 739
Prerequisites ... 740
Step 1: Set up ODBC Connection to Rocket Software datastore (Amazon RDS database) ... 740
Step 2: Create the MFDBFH.cfg file ... 742
Step 3: Create a structure (STR) file for your copybook layout ... 743
Step 4: Create a database view using the structure (STR) file ... 746
Step 5: View Rocket Software (formerly Micro Focus) data sets as tables and columns ... 746
Edit data sets using Data File Tools in Enterprise Developer ... 747
Prerequisites ... 747
Launch Rocket Software(formerly Micro Focus) Data File Tools ... 748
Edit VSAM data sets stored in the MFDBFH database ... 749
Edit non-VSAM data sets stored in the MFDBFH database ... 752
Edit VSAM and non-VSAM data sets stored in the File System (EFS/FSx) ... 754
Tutorials for Rocket Software ... 754
Tutorial: Set up the build for the BankDemo sample application ... 755
Tutorial: Set up CI/CD pipeline with Rocket Enterprise Developer ... 765
Tutorial: Set up AppStream 2.0 for Enterprise Analyzer and Enterprise Developer ... 790
Tutorial: Use templates with Rocket Enterprise Developer ... 799
Tutorial: Set up Enterprise Analyzer ... 810
Tutorial: Set up Enterprise Developer ... 821
Batch utilities ... 826
Binary Location ... 827
M2SFTP batch utility ... 827
M2WAIT batch utility ... 834
TXT2PDF batch utility ... 836
M2DFUTIL batch utility ... 842
M2RUNCMD batch utility ... 849
File Transfer ... 853
What is File Transfer ... 853

Benefits of AWS Mainframe Modernization File Transfer ... 853
How AWS Mainframe Modernization File Transfer works ... 854
Install a File Transfer agent ... 855
Step 1: Create a zFS data set for the M2-agent ... 856
Step 2: Format the data set as zFS ... 856
Step 3: Mount the filesystem ... 856
Step 4: Verify the mount ... 857
Step 5: Enter OMVS ... 857
Step 6: Set the agent installation directory environment variable ... 857
Step 7: Set the work directory environment variable ... 857
Step 8: Create the work directory ... 857
Step 9: Copy the agent tar file and copy the work directory ... 858
Step 10: Finish the agent installation ... 858
Configure a File Transfer agent ... 859
Step 1: Configure permissions and Started Task Control (STC) ... 859
Step 2: Create Amazon S3 buckets ... 860
Step 3: Create an AWS KMS customer managed key for encryption ... 861
Step 4: Create an AWS Secrets Manager secret for the mainframe credentials ... 862
Step 5: Create an IAM policy ... 863
Step 6: Create an IAM user with long-term access credentials ... 865
Step 7: Create an IAM role for the agent to assume ... 866
Step 8: Agent configuration ... 867
Create data transfer endpoints ... 869
Create data transfer endpoints ... 869
Create transfer tasks ... 871
Create transfer tasks ... 872
View transfer tasks ... 875
Tutorial: Getting started with File Transfer ... 875
Overview ... 875
Step 1: Transfer the agent binaries tar package from AWS to the mainframe logical partition ... 876
Step 2: Configure the File Transfer agent on the source mainframe ... 876
Step 3: Create a data transfer endpoint ... 876
Step 4: Create a transfer task ... 876
Step 5: View transfer task progress ... 876
Supported source and target code pages ... 877

Mainframe data set types ... 877
Supported code pages ... 877
AWS Transform for mainframe ... 879
Key benefits ... 879
Transformation of mainframe application console walkthrough ... 880
Data replication with Precisely ... 881
Prerequisites ... 881
Subscribe to the Amazon Machine Image ... 881
Launch AWS Mainframe Modernization data replication with Precisely ... 882
Create an IAM policy ... 883
Create an IAM role ... 884
Attach the IAM role to the Amazon EC2 instance ... 884
Assembler Conversion with mLogica ... 885
What is Assembler Conversion with mLogica ... 885
Code conversion compilers ... 886
Code conversion architecture ... 886
Automation approach ... 887
Security ... 887
Additional resources ... 887
Understand Code conversion billing ... 887
Code conversion billing and scope ... 887
Code conversion concepts ... 890
Macro Handling ... 890
Code pages (EBCDIC vs ASCII) ... 890
CodeBuild ... 890
Understand components and process ... 891
AWS Mainframe Modernization container ... 891
S3 project bucket ... 892
Log file locations ... 892
Process overview ... 892
Tutorial: Convert code from Assembler to COBOL ... 893
Prerequisites ... 894
Step 1: Share the build assets with AWS account ... 894
Step 2: Create Amazon S3 buckets ... 894
Step 3: Create IAM policy ... 895
Step 4: Create an IAM role ... 897

Step 5: Attach the IAM policy to the IAM role ... 898
Step 6: Create the CodeBuild project ... 898
Step 7: Define the project and upload the source code ... 904
Step 8: Run the analysis and understand the reports ... 905
Step 9: Run the Code conversion ... 907
Step 10: Verify the Code conversion ... 911
Step 11: Download converted code ... 912
Clean up resources ... 912
Charon integration ... 913
Introduction to Charon-SSP ... 913
Supported guest operating systems ... 915
Charon-SSP cloud instance prerequisites ... 915
Instance prerequisites ... 917
Creating and configuring an AWS cloud instance for Charon (New GUI) ... 918
General prerequisites ... 918
Using the AWS Management Console to launch a new instance ... 919
Replatforming with NTT DATA ... 925
Prerequisites ... 925
Subscribe to the Amazon Machine Image ... 925
Launch AWS Mainframe Modernization replatform with NTT DATA instance ... 926
Getting started with NTT Data ... 926
Tutorial: Deploy CardDemo application on NTT DATA ... 928
Deployment flow diagram ... 928
Prerequisites ... 929
Step 1: Prepare the environment ... 930
Step 2: Create a TPE region ... 930
Step 3: Create the BPE node and subsystem ... 931
Step 4: Compile and deploy CardDemo application ... 940
Step 5: Import BPE and TPE catalog ... 942
Step 6: Start and connect TPE with BPE ... 942
Step 7: Run the CardDemo application ... 943
Troubleshooting ... 949
Security ... 951
Data protection ... 952
Data that AWS Mainframe Modernization collects ... 953
Data encryption at rest for AWS Mainframe Modernization service ... 954

How AWS Mainframe Modernization uses grants in AWS KMS ... 956
Create a customer managed key ... 958
Specifying a customer managed key for AWS Mainframe Modernization ... 960
AWS Mainframe Modernization encryption context ... 961
Monitoring your encryption keys ... 962
Learn more ... 977
Encryption in transit ... 977
Identity and Access Management ... 978
Audience ... 978
Authenticating with identities ... 979
Managing access using policies ... 982
How AWS Mainframe Modernization works with IAM ... 985
Identity-based policy examples ... 999
Troubleshooting ... 1002
Using service-linked roles ... 1003
Compliance validation ... 1006
Resilience ... 1007
Infrastructure security ... 1008
AWS PrivateLink ... 1008
Considerations ... 1009
Create an interface endpoint ... 1009
Create an endpoint policy ... 1009
Monitoring ... 1011
Monitoring with CloudWatch ... 1011
Runtime Environment Metrics ... 1012
Application Metrics ... 1013
Dimensions ... 1017
Logging API calls with CloudTrail ... 1017
AWS Mainframe Modernization information in CloudTrail ... 1018
Understanding AWS Mainframe Modernization log file entries ... 1019
Troubleshooting in M2 ... 1021
Troubleshooting error: Time out while waiting for data set name to be unlocked ... 1021
Common cause ... 1022
Resolution ... 1022
Force the lock to release ... 1022
Configure the Blusam auto repairing mechanism ... 1023

Blusam locks manager 1024
Troubleshooting error: Cannot access an application URL 1024
Common cause 1025
Resolution 1025
Troubleshooting: AWS Blu Insights does not open from the console 1026
Common cause 1026
Resolution 1026
Troubleshooting error: Environment unhealthy 1027
Common cause 1027
Resolution 1027
Troubleshooting license issues for Rocket Software 1028
Verify the Amazon EC2 instance has the IAM licensing role 1028
Use the reachability analyzer 1029
Run the license-daemon 1029
License issues with Enterprise Server or Enterprise Build Tools on Linux after OS patching 1030
Document history 1032

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
- Patterns
- How to get started with AWS Mainframe Modernization
- Related services
- Accessing AWS Mainframe Modernization
- Are you a first-time AWS Mainframe Modernization user?
- Pricing for AWS Mainframe Modernization

> **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.

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 runtime 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, see AWS 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 Rocket Software formerly 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.

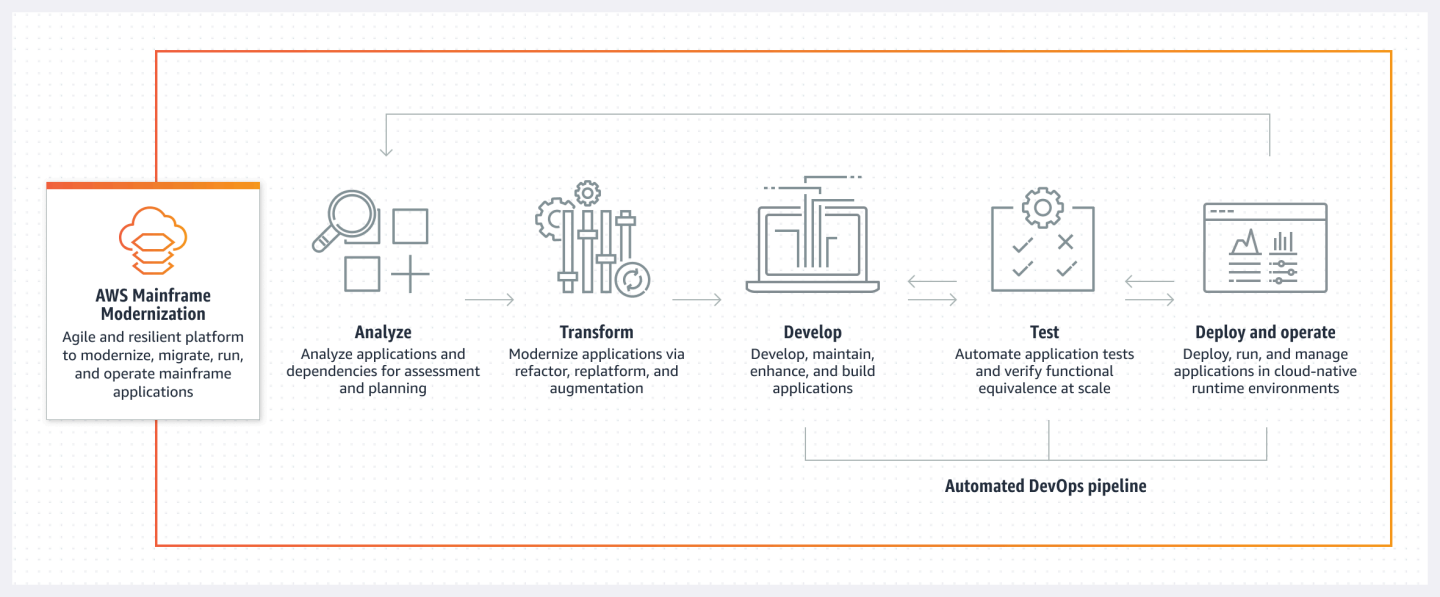
How to get started with AWS Mainframe Modernization

Try it! We offer tutorials and sample applications to help you get a sense of what AWS Mainframe Modernization offers. Choose either the Tutorial: Set up managed runtime for AWS Blu Age or the Tutorial: Set up managed runtime for Rocket Software (formerly Micro Focus) 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 Rocket Enterprise Analyzer (formerly Micro Focus Enterprise Analyzer) and Rocket Enterprise Developer (formerly 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 for details about the stages and tasks of a modernization project.

The following diagram shows the workflow of the AWS Mainframe Modernization service to analyze, transform, develop, test, and deploy and operate mainframe applications.



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 Rocket Enterprise Analyzer and Rocket 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:

- Get started with AWS Mainframe Modernization
- Set up for AWS Mainframe Modernization

Pricing for AWS Mainframe Modernization

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.

Set up for AWS Mainframe Modernization

Before you can start using AWS Mainframe Modernization you or your administrator need to sign up for an AWS account, create user with administrative settings, and secure your IAM users.

Topics

- Sign up for an AWS account
- Create a user with administrative access

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

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message 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 a 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 a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and 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 a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- 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*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

AWS Mainframe Modernization Concepts

AWS Mainframe Modernization provides tools and resources to help you migrate, modernize, and run mainframe workloads on AWS. You can use this page to understand about various concepts in AWS Mainframe Modernization including applications, modernization, environments, replatforming, refactoring, and runtime engines.

Topics

- Application
- Application definition
- Batch job
- Configuration
- Data set
- Environment
- Mainframe modernization
- Migration journey
- Mount point
- Automated Refactoring
- Replatforming
- Resource
- Runtime engine

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.

Holding

The batch job is on hold.

Dispatching

The batch job is in the process of being dispatched.

Running

The batch job is currently running.

Cancelling

The batch job is in the process of being cancelled.

Cancelled

The batch job is cancelled.

Succeeded

The batch job finished running successfully.

Failed

The batch job failed.

Succeeded With Warning

The batch job finished running successfully with a minor error reported. The job condition code returned as part of the GetBatchJobExecution response indicates the cause of the error.

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.

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 Rocket Software (formerly 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
- Mobilize phase
- Migrate and modernize phase
- Operate and optimize phase

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.

Get started with AWS Mainframe Modernization

You can get started with AWS Mainframe Modernization by following tutorials that introduce you to the service and each runtime engine.

Topics

- Tutorial: Set up managed runtime for AWS Blu Age
- Tutorial: Set up managed runtime for Rocket Software (formerly Micro Focus)

To continue learning, see the following tutorials.

- Tutorial: Setting up the Rocket Software (formerly Micro Focus) build for the BankDemo sample application
- Tutorial: Setting up a CI/CD pipeline for use with Rocket Enterprise Developer (formerly Micro Focus Enterprise Developer)

Tutorial: Set up managed runtime for AWS Blu Age

You can deploy a AWS Blu Age modernized application into an AWS Mainframe Modernization runtime environment with a demo application specified in this tutorial.

Topics

- Prerequisites
- Step 1: Upload the demo application
- Step 2: Create the application definition
- Step 3: Create a runtime environment
- Step 4: Create an application
- Step 5: Deploy an application
- Step 6: Start an application
- Step 7: Access the application
- Step 8: Test the application
- Clean up resources

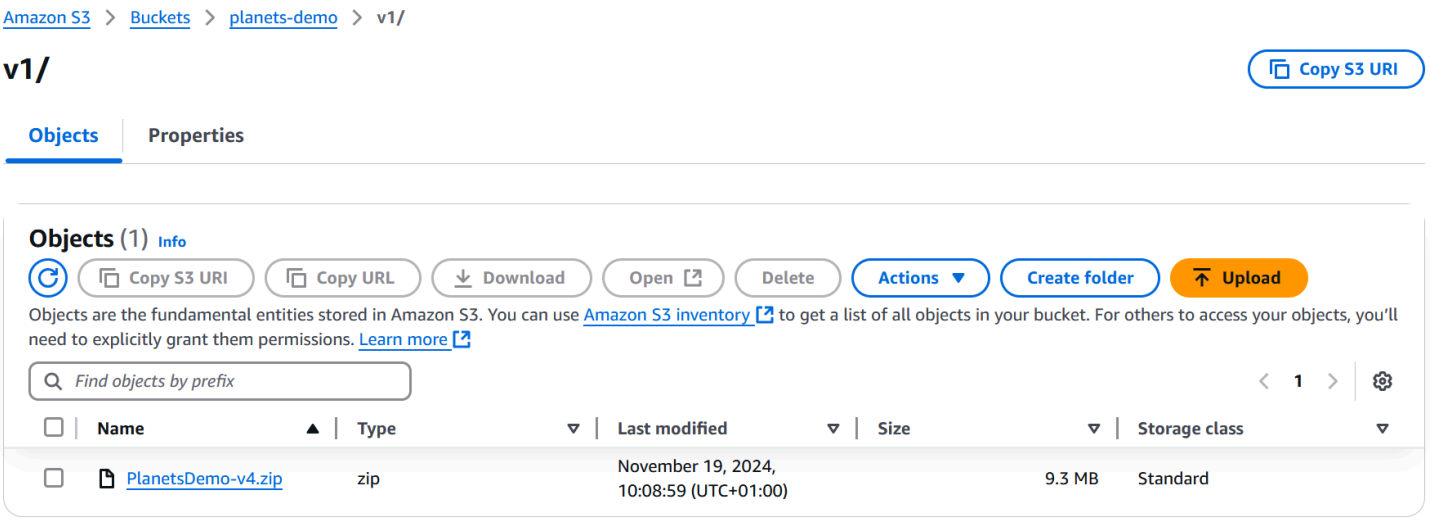
Prerequisites

To complete this tutorial, download the demo application archive PlanetsDemo-v4.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 **planets-demo**, with a key prefix, or folder, named **v1** and an archive named `planetsdemo-v4.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:

```
{
```

```
    "template-version": "2.0",
    "source-locations": [{
        "source-id": "s3-source",
        "source-type": "s3",
        "properties": {
            "s3-bucket": "planets-demo",
            "s3-key-prefix": "v1"
        }
    }],
    "definition": {
        "listeners": [{
            "port": 8196,
            "type": "http"
        }],
        "ba-application": {
            "app-location": "${s3-source}/PlanetsDemo-v4.zip"
        }
    }
}
```

Change the `s3-bucket` entry to the name of the sample application zip file (e.g., `planets-demo`), and the `app-location` entry to the S3 path where you stored the sample application zip file (e.g., `${s3-source}/PlanetsDemo-v4.zip`).

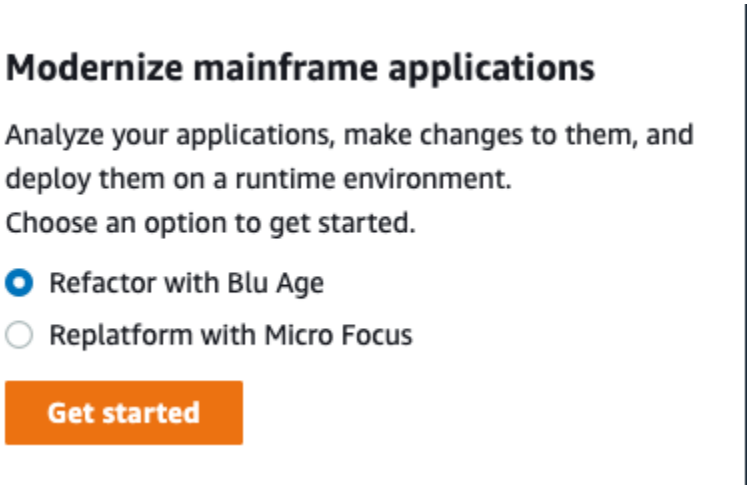
> **Note**
>
> Make sure to create the application definition file on your local as a text file.

For more information on the application definition, see AWS Blu Age application definition sample.

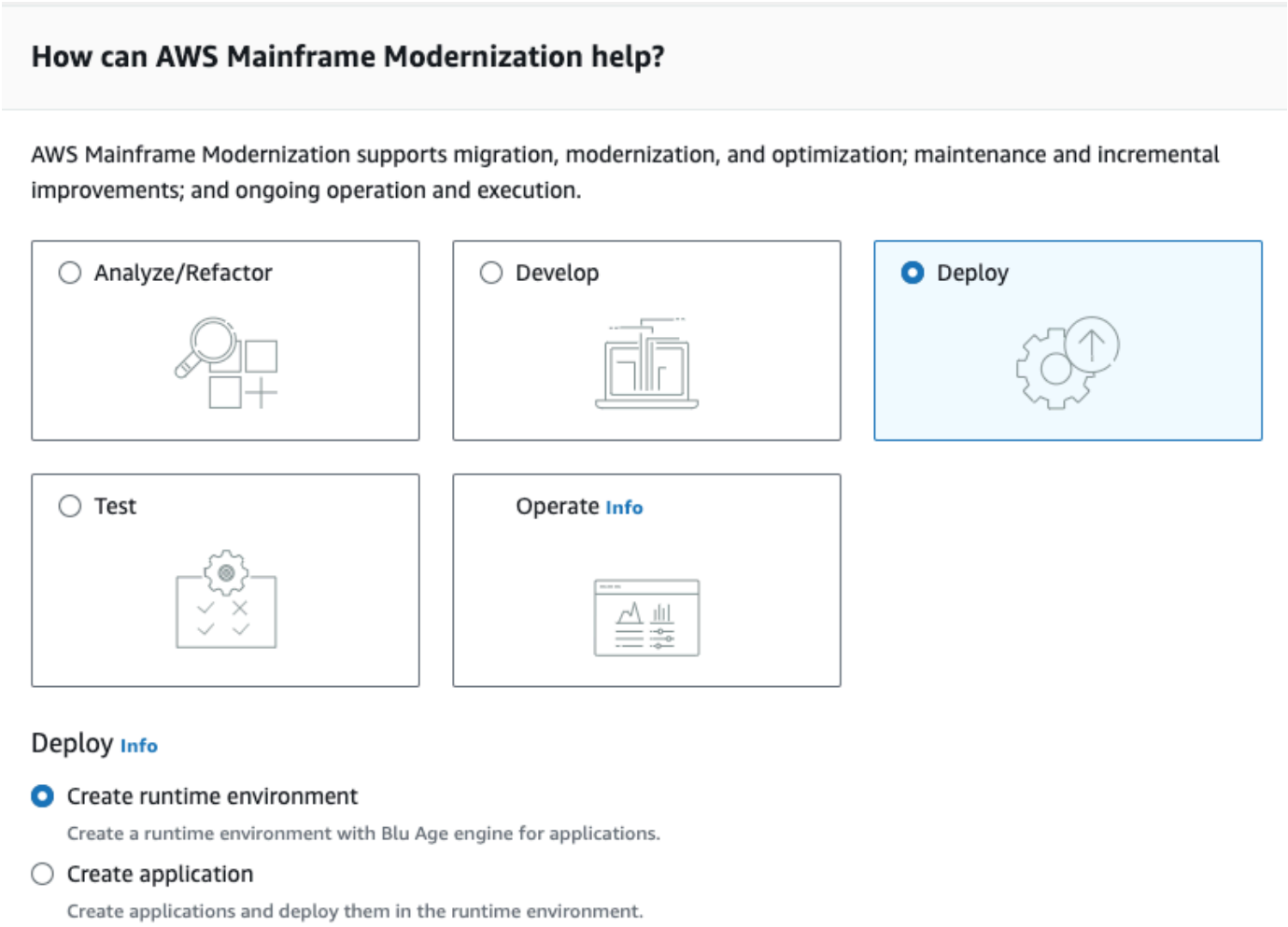
Step 3: Create a runtime environment

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

1. Open the AWS Mainframe Modernization console.
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.
3. Under **Modernize mainframe applications**, choose **Refactor with Blu Age**, and then choose **Get started**.

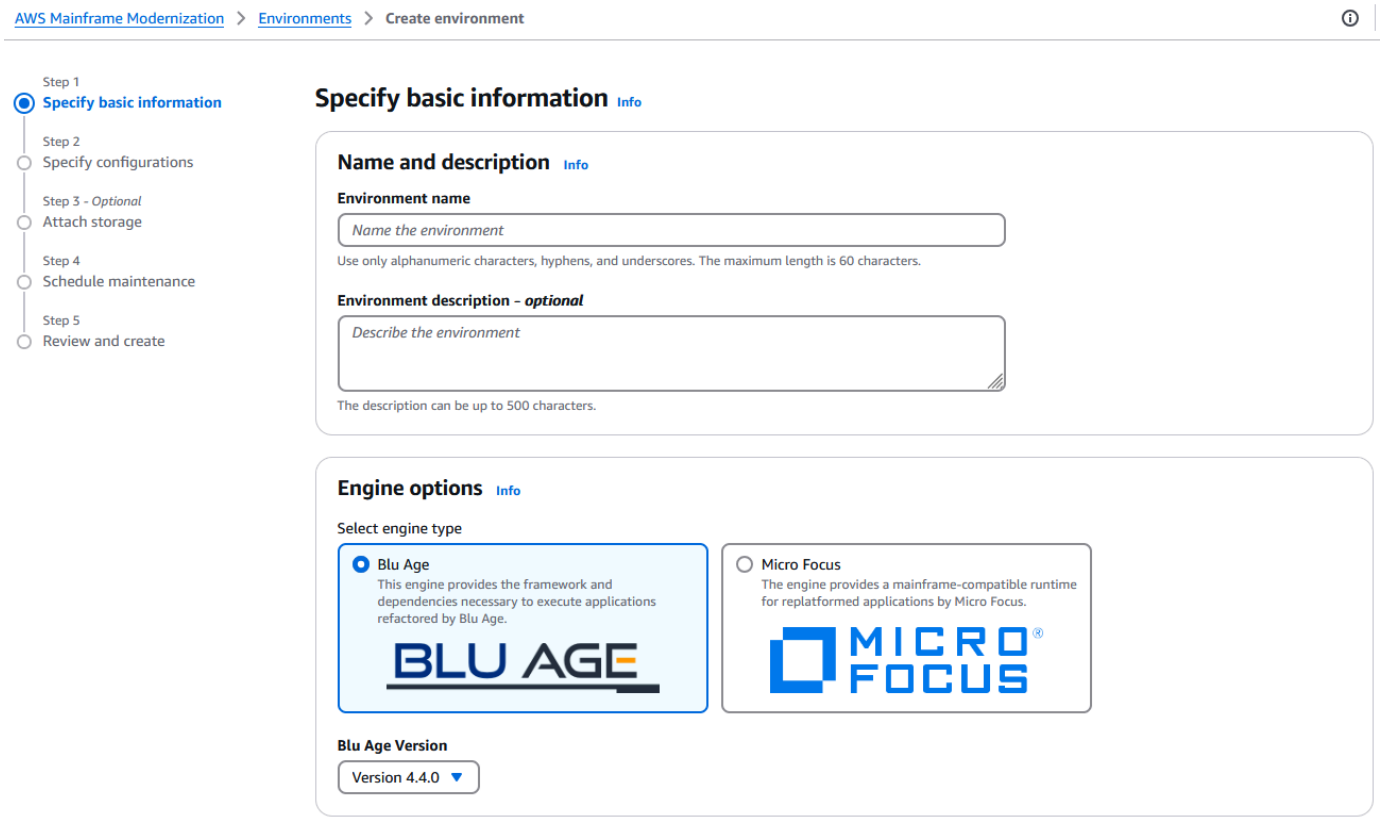


4. Under **How can AWS Mainframe Modernization help**, choose **Deploy** and **Create runtime environment**.

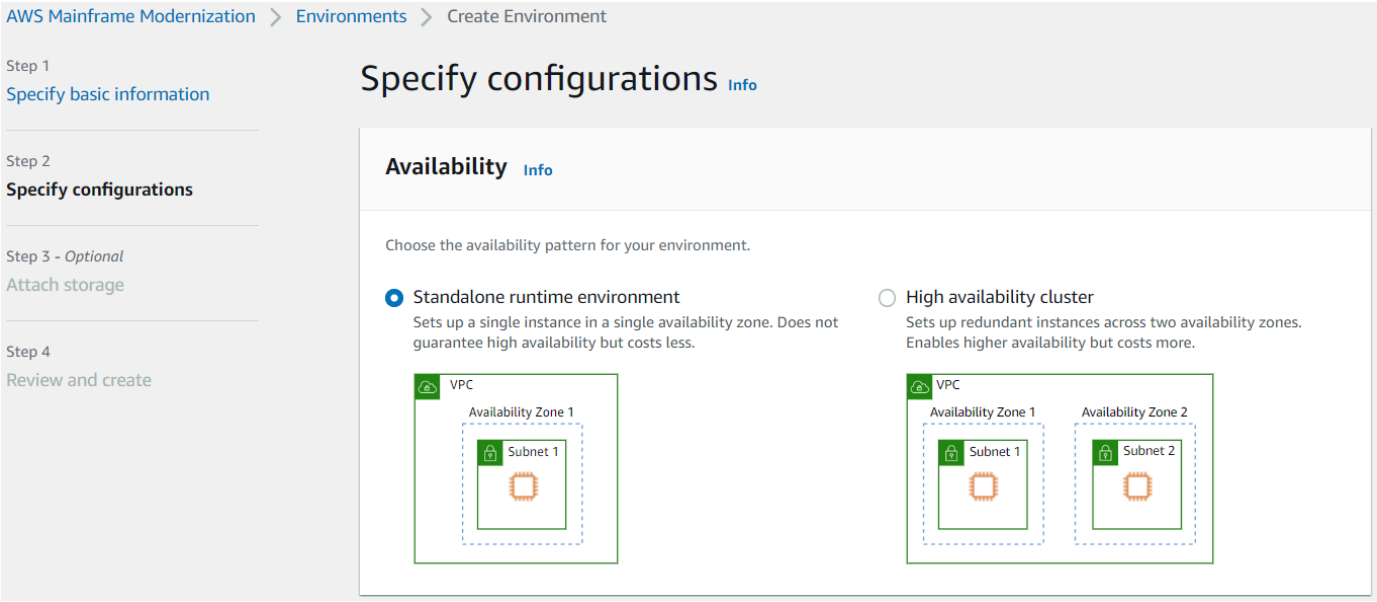


5. In the left navigation, choose **Environments**, then choose **Create environment**. On the **Specify basic information** page, enter a name and description for your environment, and then make

sure **AWS Blu Age** engine is selected. Optionally, you can add tags to the created resource. Then choose **Next**.

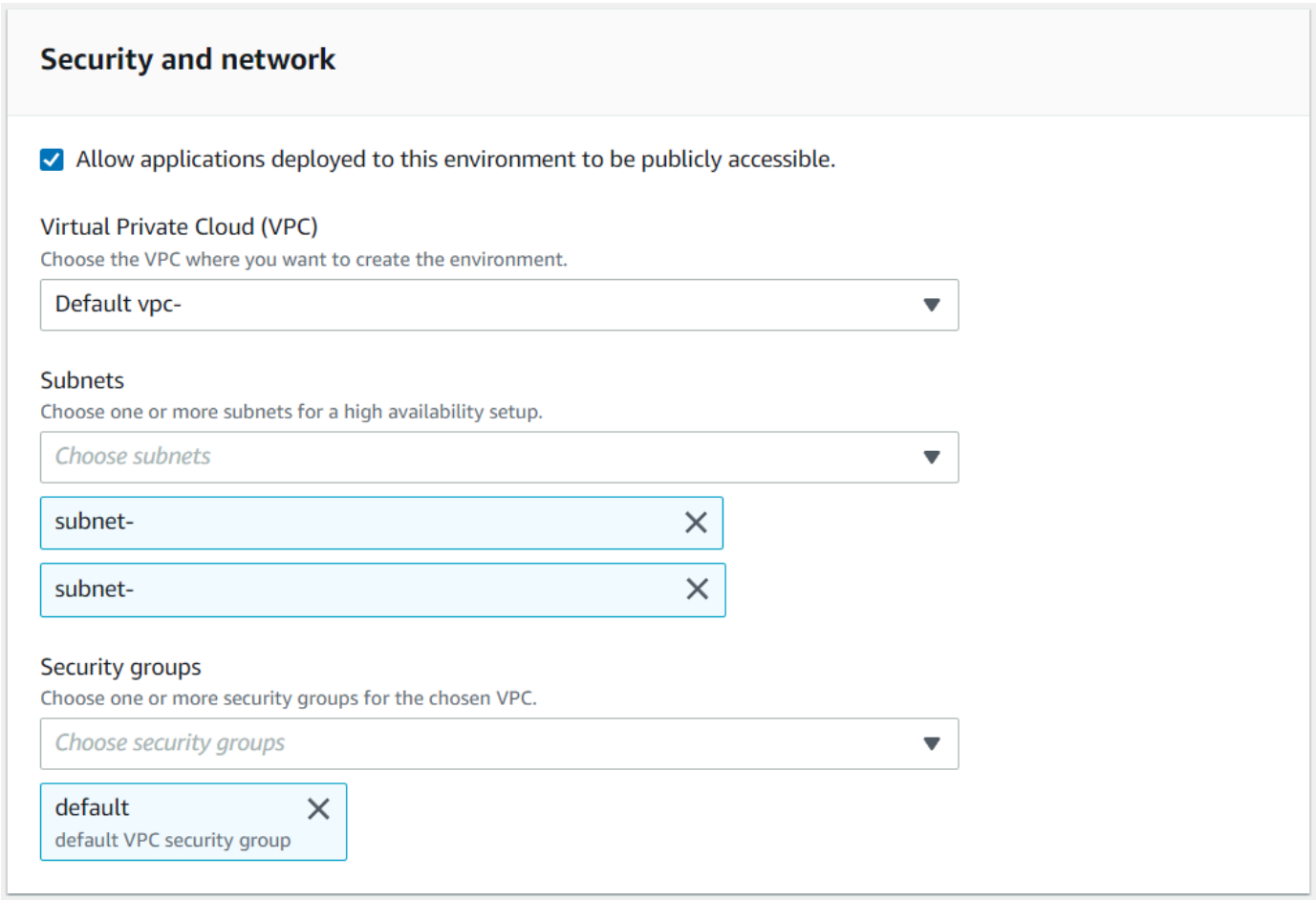


6. On the **Specify configurations** page, choose **Standalone runtime environment**.



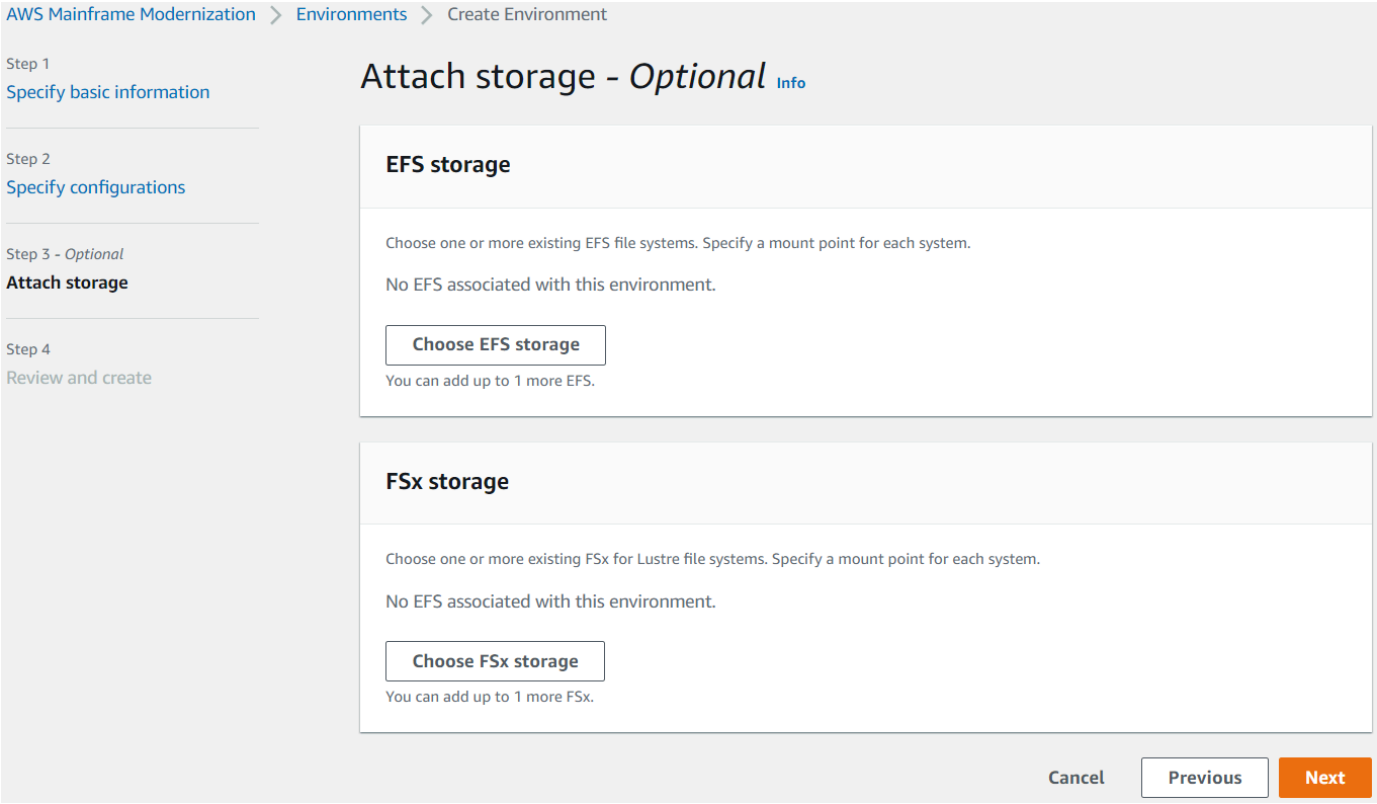
7. Under **Security and network**, make the following changes:

- 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.



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 Troubleshooting error: Cannot access an application URL.

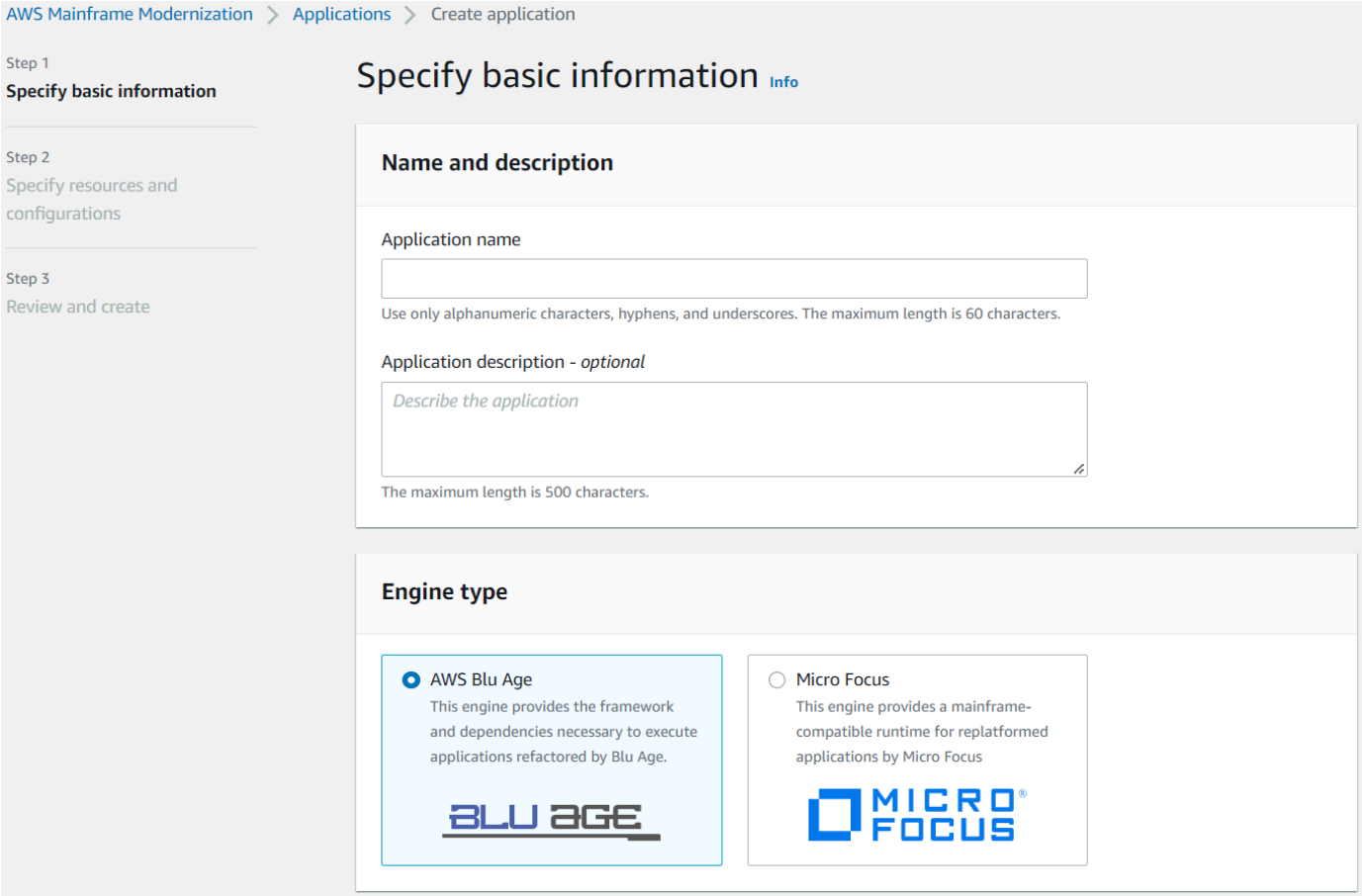
8. Choose **Next**.
9. In **Attach storage - Optional**, leave the default selections and choose **Next**.



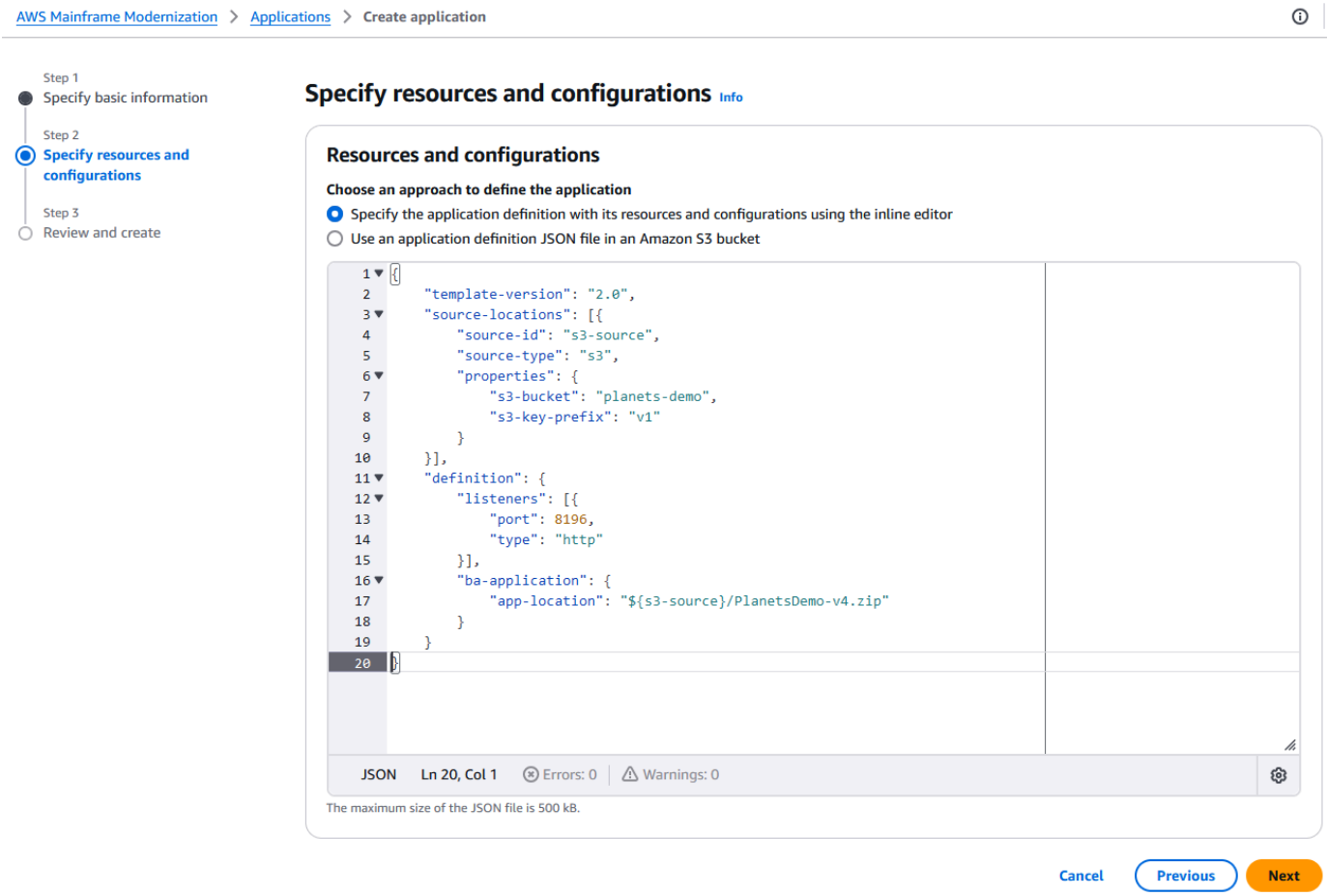
10. In **Schedule maintenance**, choose **No preference**, and then choose **Next**.
11. In **Review and create**, review the information, and then choose **Create environment**.

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 you created in the section called “Step 2: Create the application definition”.



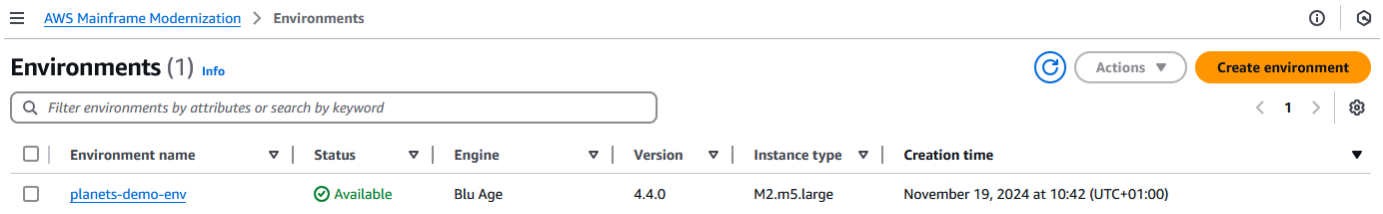
4. In **Review and create**, review your choices, and then choose **Create application**.

> **Note**
>
> If your application creation fails, check the S3 path you entered since it's case sensitive.

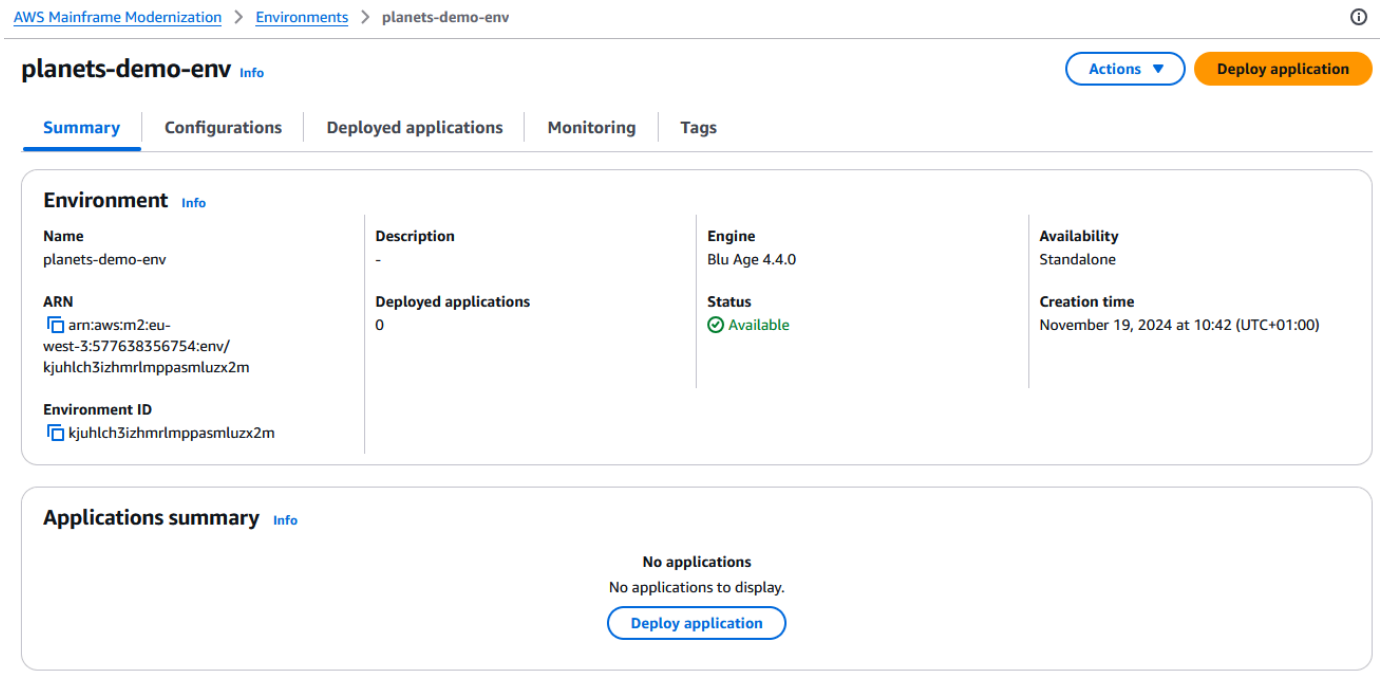
Step 5: Deploy an application

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:

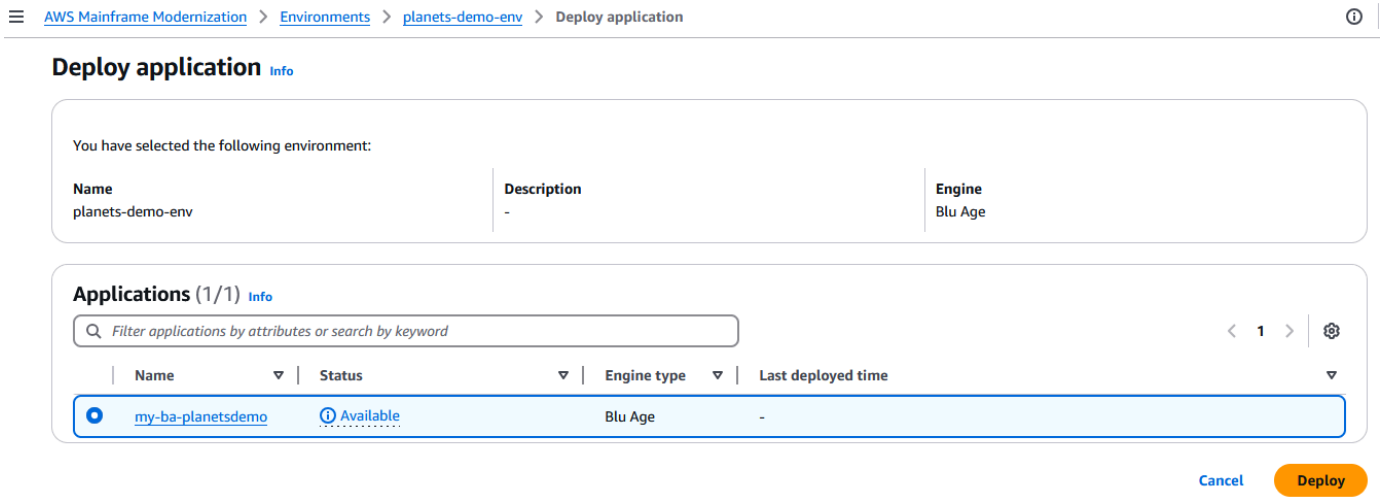
1. Navigate to **AWS Mainframe Modernization** in the AWS Management Console. In the navigation pane, choose **Environments**. The Environments list page is displayed.



2. Choose the previously created runtime environment. The environment details page is displayed.
3. Choose **Deploy application**.



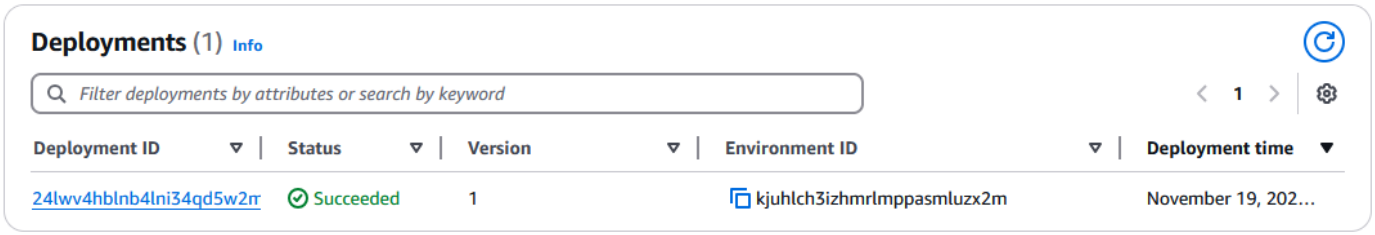
4. Choose the previously created application, then choose the version you want to deploy your application to. Then choose **Deploy**.



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**.



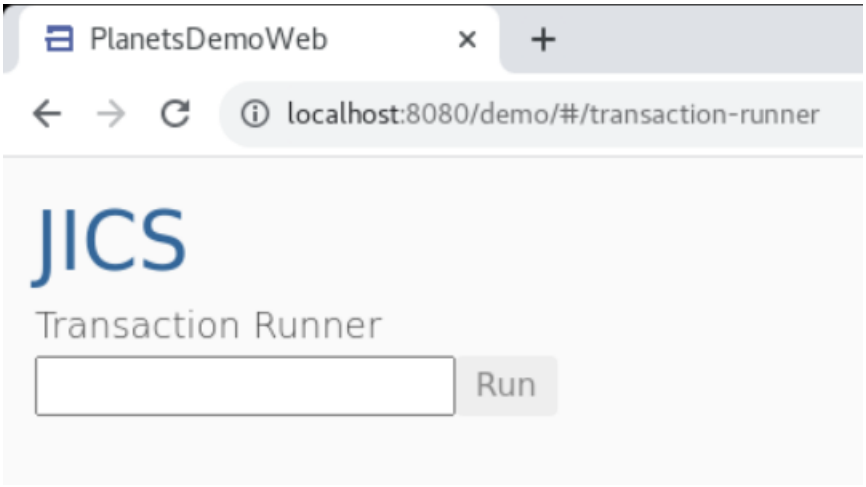
3. Choose **Actions**, and then choose **Start application**.

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.

   The JICS screen appears.



If you can't access the application, see Troubleshooting error: Cannot access an application URL.

> **Note**
>
> If the application is not accessible, and the inbound rule on security group has 'My IP' selected on port 8196, specify rule to allow traffic from LB i/p on port 8196.

Step 8: Test the application

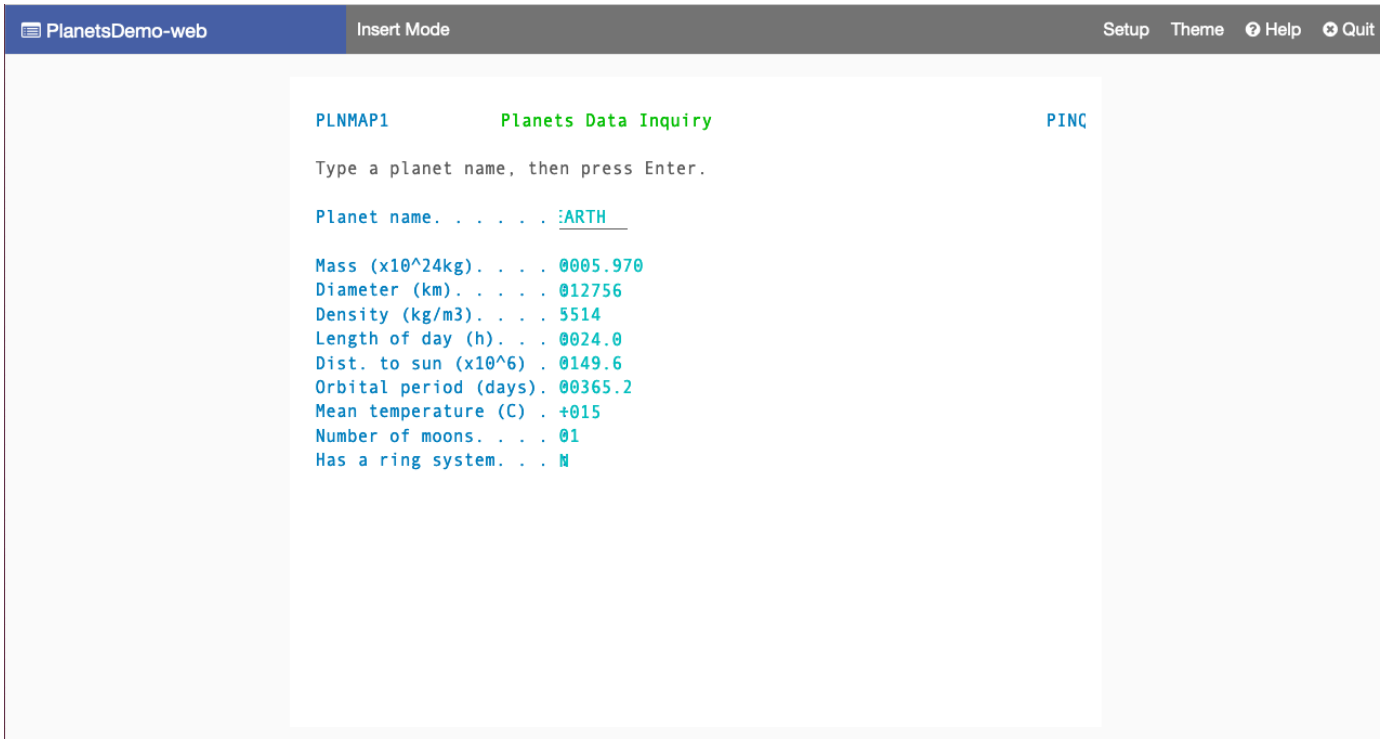
In this step, you run a transaction in the migrated application.

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.
- Delete the runtime environment. For more information, see Delete an AWS Mainframe Modernization runtime environment.

Tutorial: Set up managed runtime for Rocket Software (formerly Micro Focus)

You can deploy and run an application in AWS Mainframe Modernization managed runtime environment with the Rocket Software runtime engine. This tutorial shows how to deploy and run the CardDemo sample application in an AWS Mainframe Modernization managed runtime environment with the Rocket Software runtime engine. The CardDemo sample application is a simplified credit card application developed to test and showcase AWS and partner technology for mainframe modernization use cases.

In the tutorial, you create resources in other AWS services. These include Amazon Simple Storage Service, Amazon Relational Database Service, AWS Key Management Service, and AWS Secrets Manager.

Topics

- Prerequisites
- Step 1: Create and load an Amazon S3 bucket
- Step 2: Create and configure a database
- Step 3: Create and configure an AWS KMS key
- Step 4: Create and configure an AWS Secrets Manager database secret
- Step 5: Add the sslMode to the secret
- Step 6: Create a runtime environment
- Step 7: Create an application

- Step 8: Deploy an application
- Step 9: Import data sets
- Step 10: Start an application
- Step 11: Connect to the CardDemo CICS application
- Clean up resources
- Next steps

Prerequisites

- Make sure that you have access to a 3270 emulator to use the CICS connection. Free and trial 3270 emulators are available from third party websites. Alternatively, you can start an AWS Mainframe Modernization AppStream 2.0 Rocket Software instance and use the Rumba 3270 emulator (not available for free).

  For information about AppStream 2.0, see the section called "Tutorial: Set up AppStream 2.0 for Enterprise Analyzer and Enterprise Developer".

  > **Note**
  >
  > When creating the stack, choose the Enterprise Developer (ED) option and not Enterprise Analyzer (EA).

- Download the CardDemo sample application and unzip the downloaded file to any local directory. This directory will contain a subdirectory titled `CardDemo_runtime`.
- Identify a VPC in your account where you can define the resources created in this tutorial. The VPC will need subnets in at least two Availability Zones. For more information about Amazon VPC, see How Amazon VPC works.

Step 1: Create and load an Amazon S3 bucket

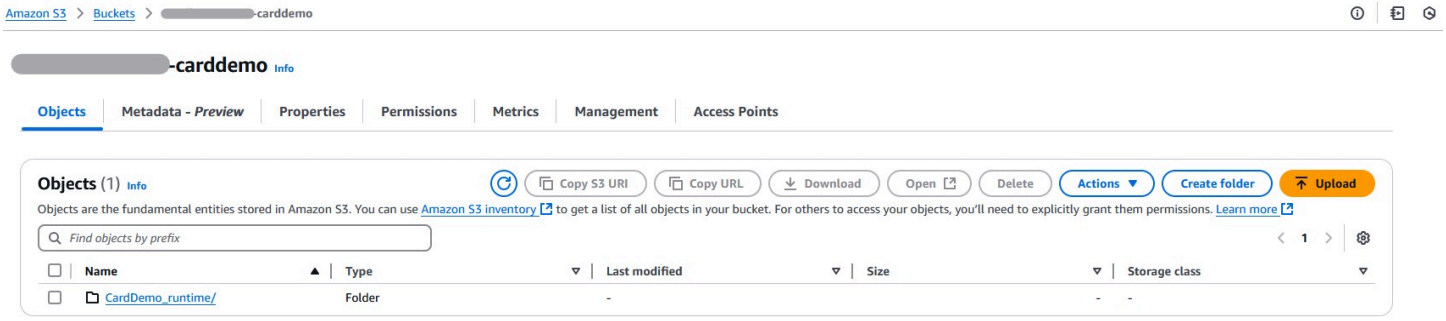
In this step, you create an Amazon S3 bucket and upload CardDemo files to this bucket. Later in this tutorial, you use these files to deploy and run the CardDemo sample application in an AWS Mainframe Modernization Rocket Software Managed Runtime environment.

> **Note**
>
> You do not have to create a new S3 bucket but the bucket that you choose must be in the same Region as other resources used in this tutorial.

To create an Amazon S3 bucket

1. Open the Amazon S3 console, and choose **Create bucket**.
2. In **General configuration**, choose the **AWS Region** where you want to build the AWS Mainframe Modernization Rocket Software Managed Runtime.
3. Enter a **Bucket name**, for example, `yourname-aws-region-carddemo`. Keep the default settings, and choose **Create bucket**. Alternatively, you can also copy settings from an existing Amazon S3 bucket and then choose **Create bucket**.
4. Choose the bucket that you just created, and then choose **Upload**.
5. In the **Upload** section, choose **Add Folder**, and then browse to the `CardDemo_runtime` directory from you local computer.
6. Choose **Upload** to start the upload process. Upload times vary based on your connection speeds.
7. When the upload completes, confirm that all files have been successfully uploaded, and then choose **Close**.

Your Amazon S3 bucket now contains the `CardDemo_runtime` folder.



For information about S3 buckets, see Creating, configuring, and working with Amazon S3 buckets.

Step 2: Create and configure a database

In this step, you create a PostgreSQL database in Amazon Relational Database Service (Amazon RDS). For the tutorial, this database contains the data sets that the CardDemo sample application uses for customer tasks regarding credit card transactions.

To create a database in Amazon RDS

1. Open the Amazon RDS console.
2. Choose the AWS Region in which you want to create the database instance.
3. From the navigation pane, choose **Databases**.
4. Choose **Create database**, and then choose **Standard create**.
5. For **Engine type**, choose **PostgreSQL**.
6. Choose an **Engine version** of 15 or higher.

   > **Note**
   >
   > Save the engine version because you need it later in this tutorial.

7. In **Templates**, choose **Free tier**.
8. Change the **DB instance identifier** to something meaningful, for example, `MicroFocus-Tutorial`.
9. Refrain from managing master credentials in AWS Secrets Manager. Instead, enter a *master* password and confirm it.

   > **Note**
   >
   > Save the username and password that you use for the database. You will store them securely in the next steps of this tutorial.

10. Under **Connectivity**, choose the **VPC** where you want to create the AWS Mainframe Modernization managed runtime environment.
11. Choose **Create database**.

To create a custom parameter group in Amazon RDS

1. In the Amazon RDS console navigation pane, choose **Parameter groups**, and then choose **Create parameter group**.
2. In the **Create parameter group** window, for **Parameter group family**, select the **Postgres** option that matches your database version.

> **Note**
>
> Some Postgres versions require a **Type**. Select **DB Parameter Group** if needed. Enter a **Group name** and **Description** for the parameter group.

3. Choose **Create**.

To configure the custom parameter group

1. Choose the newly created parameter group.
2. Choose **Actions**, and then choose **Edit**.
3. Filter on `max_prepared_transactions` and change the parameter value to 100.
4. Choose **Save Changes**.

To associate the custom parameter group with the database

1. In the Amazon RDS console navigation pane, choose **Databases**, and then choose the database instance that you want to modify.
2. Choose **Modify**. The **Modify DB instance** page appears.

> **Note**
>
> The **Modify** option is not available until the database has finished creating and backing-up, which might take several minutes.

3. On the **Modify DB instance** page, navigate to **Additional configuration**, and change the **DB parameter group** to your parameter group. If your parameter group is not available in the list, check if it was created with the correct database version.
4. Choose **Continue**, and check the summary of modifications.

5. Choose **Apply immediately** to apply the changes instantly.
6. Choose **Modify DB instance** to save your changes.

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

> **Note**
>
> You can also use an Amazon Aurora PostgreSQL database with AWS Mainframe Modernization but there is no free tier option. For more information, see Working with Amazon Aurora postgreSQL.

Step 3: Create and configure an AWS KMS key

To store credentials securely for the Amazon RDS instance, first create an AWS KMS key.

To create an AWS KMS key

1. Open the Key Management Service console.
2. Choose **Create Key**.
3. Leave the defaults of **Symmetric** for key type and **Encrypt and decrypt** for key usage.
4. Choose **Next**.
5. Give the key an **Alias** such as `MicroFocus-Tutorial-RDS-Key` and an optional description.
6. Choose **Next**.
7. Assign a key administrator by checking the box next to your user or role.
8. Choose **Next** .
9. Assign key usage permission by checking the box next to your user or role.
10. Choose **Next**.
11. On the review screen, edit the **Key policy**, then enter the following **inside the existing "Statement" array**:

```
{
     "Sid" : "Allow access for Mainframe Modernization Service",
     "Effect" : "Allow",
        "Principal" : {
           "Service" : "m2.amazonaws.com"
```

```
                },
      "Action" : "kms:Decrypt",
      "Resource" : "*"
},
```

This policy grants AWS Mainframe Modernization decrypt permissions using this specific key policy.

12. Choose **Finish** to create the key.

For more information, see Creating keys in the AWS Key Management Service Developer Guide.

Step 4: Create and configure an AWS Secrets Manager database secret

Now store the database credentials securely using the AWS Secrets Manager and AWS KMS key.

To create and configure an AWS Secrets Manager database secret

1. Open the Secrets Manager console.
2. In the navigation pane, choose **Secrets**.
3. In **Secrets**, choose **Store a new secret**.
4. Set the **Secret type** to **Credentials for Amazon RDS database**.
5. Enter the **Credentials** that you specified when you created the database.
6. Under **Encryption key**, select the key that you created in step 3.
7. In the **Database** section, select the database that you created for this tutorial, and then choose **Next**.
8. Under **Secret name**, enter a name such as `MicroFocus-Tutorial-RDS-Secret` and an optional description.
9. In the **Resource permissions** section, choose **Edit permissions**, and replace the contents with the following policy:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect" : "Allow",
            "Principal" : {
                "Service" : "m2.amazonaws.com"
```

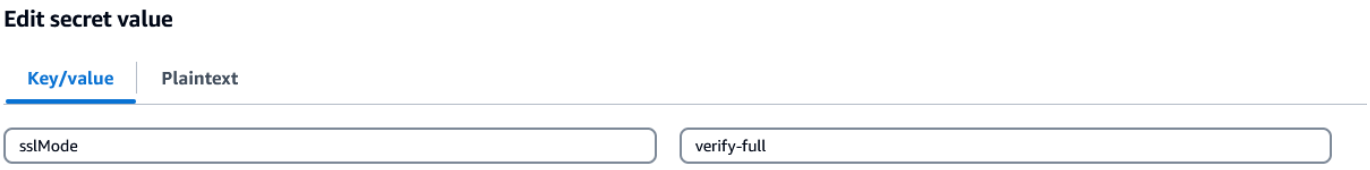
```
            },
            "Action" : "secretsmanager:GetSecretValue",
            "Resource" : "*"
        }
    ]
}
```

10. Choose **Save**.
11. Choose **Next** for the subsequent screens, and then choose **Store**.

Step 5: Add the sslMode to the secret

To add the sslMode to the secret

1. Refresh the secrets list to see the new secret.
2. Choose the newly created secret in step 4, and note the `Secret ARN` because you need it later in the tutorial.
3. In the **Overview** tab of the secret, choose **Retrieve secret value**.
4. Choose **Edit**, and then choose **Add row**.
5. Add a **Key** for `sslMode` with a **Value** of `verify-full`:

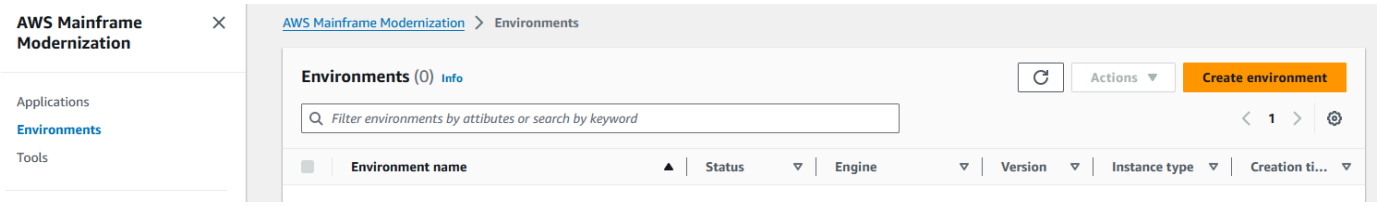


6. Choose **Save**.

Step 6: Create a runtime environment

To create a runtime environment

1. Open the AWS Mainframe Modernization console.
2. In the navigation pane, choose **Environments**. Then choose **Create environment.**



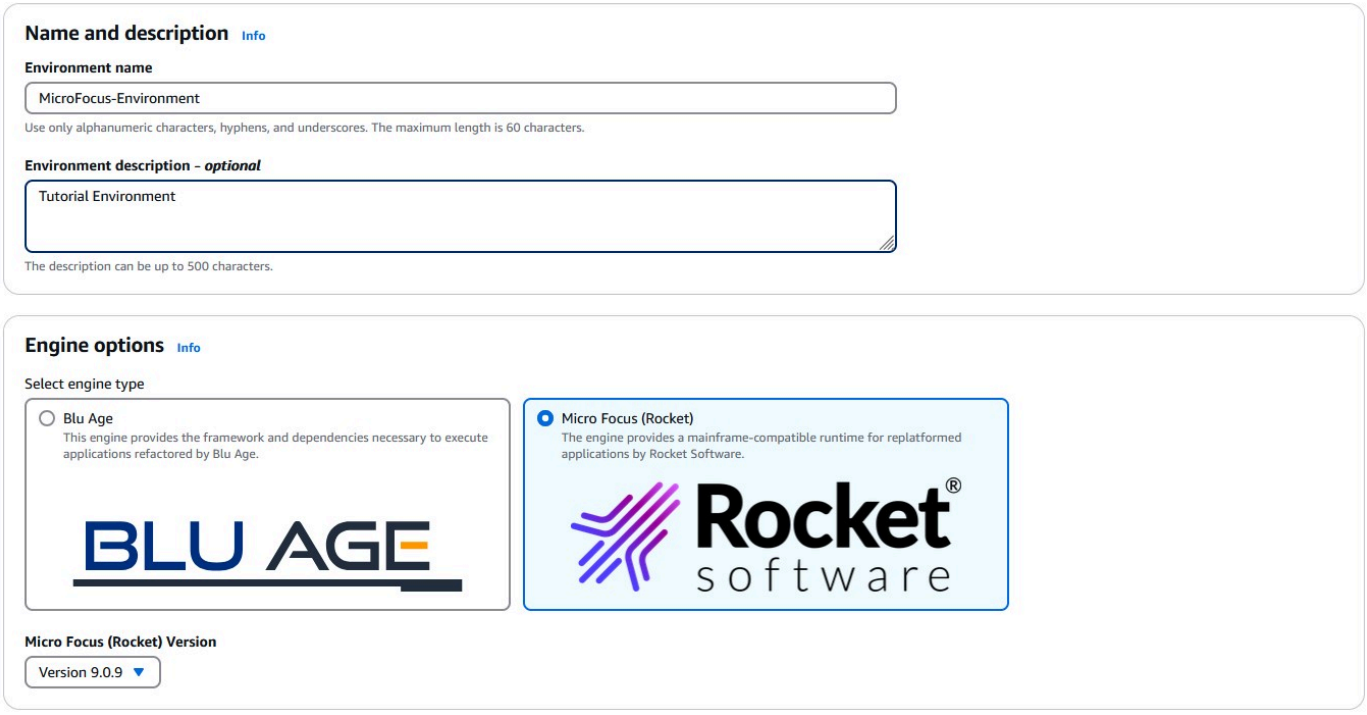
3. Under **Specify basic information**,

   a. Enter `MicroFocus-Environment` for the environment name.

   b. Under engine options, make sure **Micro Focus (Rocket)** is selected.

   c. Choose the latest **Micro Focus (Rocket) Version**.

   d. Choose **Next**.



4. Configure the environment

   a. Under **Availability**, choose **High availability cluster**.

   b. Under **Resources**, choose either **M2.c5.large** or **M2.m5.large** for the instance type, and the number of instances that you want. Specify up to two instances.

   c. Under **Security and network**, choose **Allow applications deployed to this environment to be publicly accessible** and choose at least two public subnets.

   d. Choose **Next**.

Specify configurations Info

Availability Info

Choose the availability pattern for your environment.

- Standalone runtime environment
  Sets up a single instance in a single availability zone. Does not guarantee high availability but costs less.
- High availability cluster
  Sets up redundant instances across two availability zones. Enables higher availability but costs more.



Resources

Instance type
Choose the instance type for your high availability cluster.
M2.m5.large

Desired capacity
Specify the desired number of instances.
2

Security and network

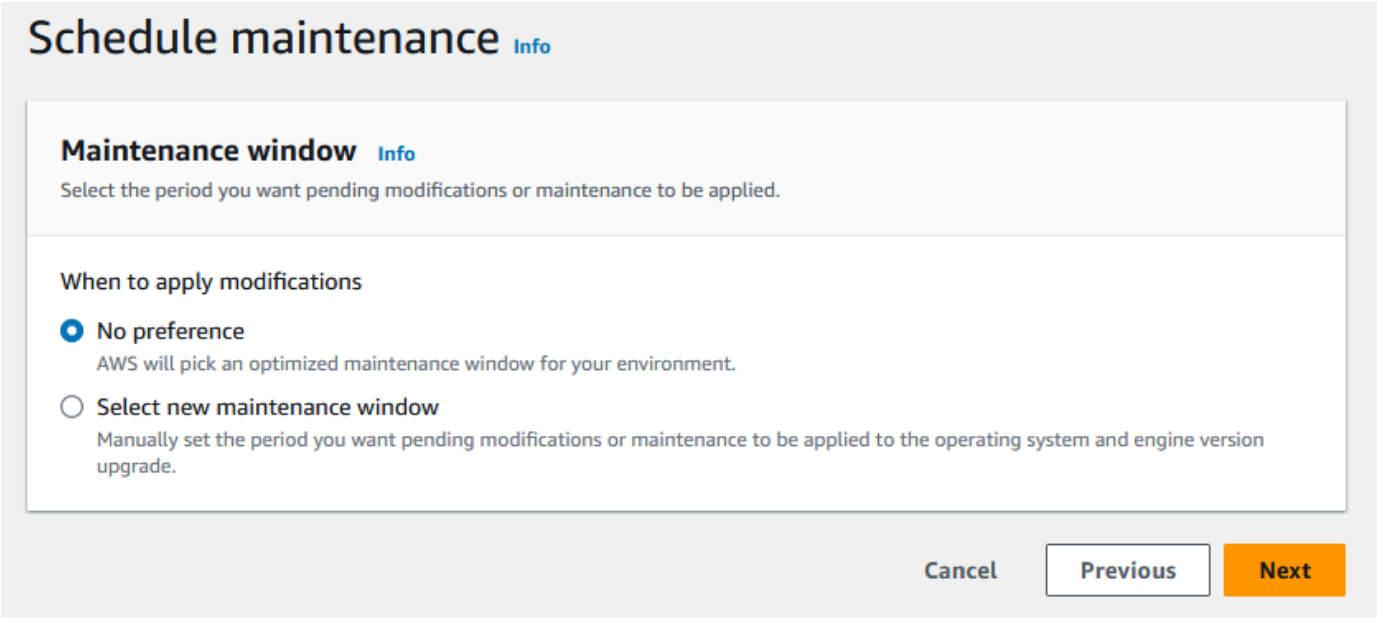
Allow applications deployed to this environment to be publicly accessible.

Virtual Private Cloud (VPC)
Choose the VPC where you want to create the environment.
Default vpc-15

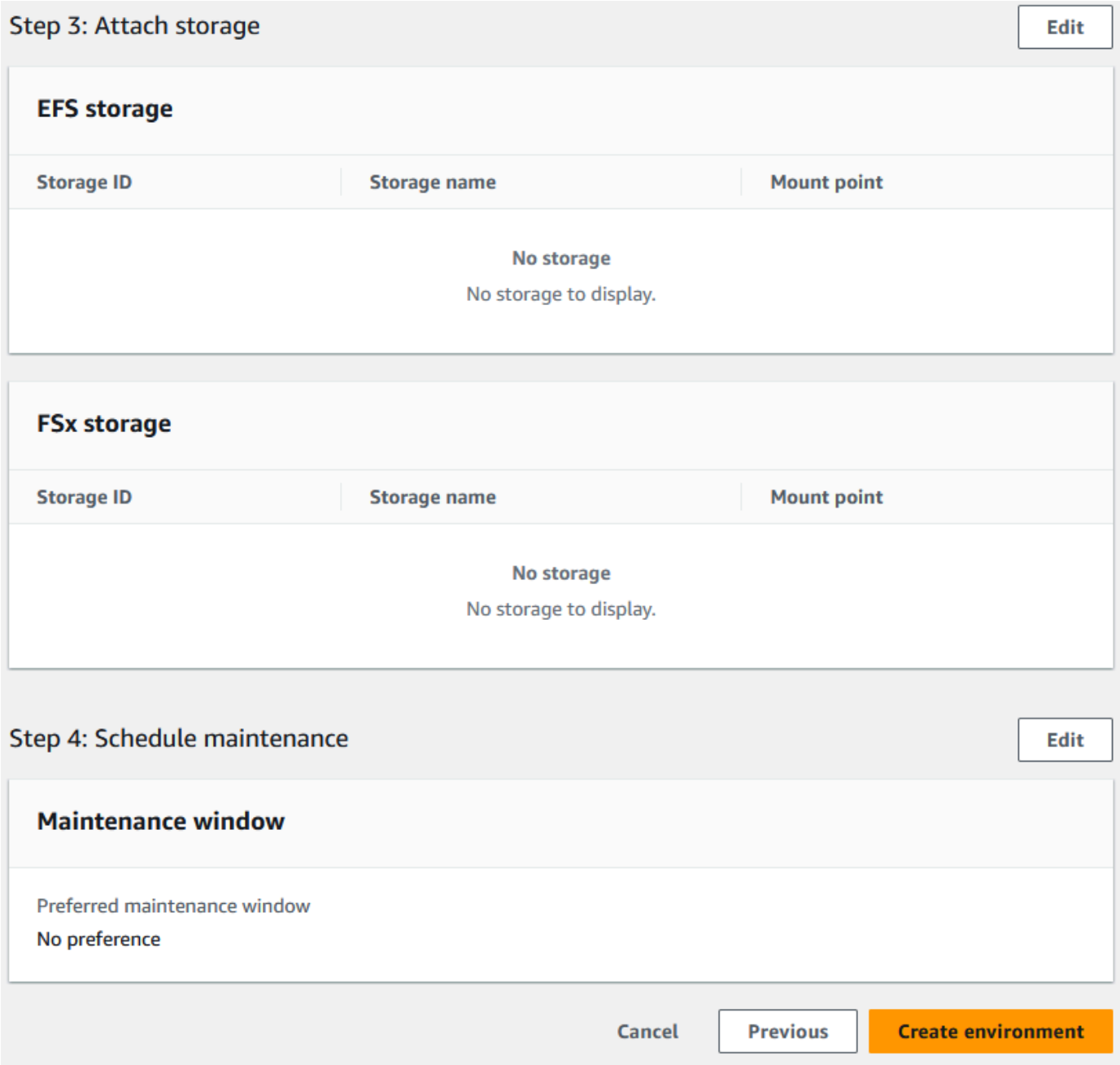
Subnets
Choose one or more subnets for a high availability setup.
Choose subnets
subnet-56f1a | us-west-2a
subnet-6685 | us-west-2b

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

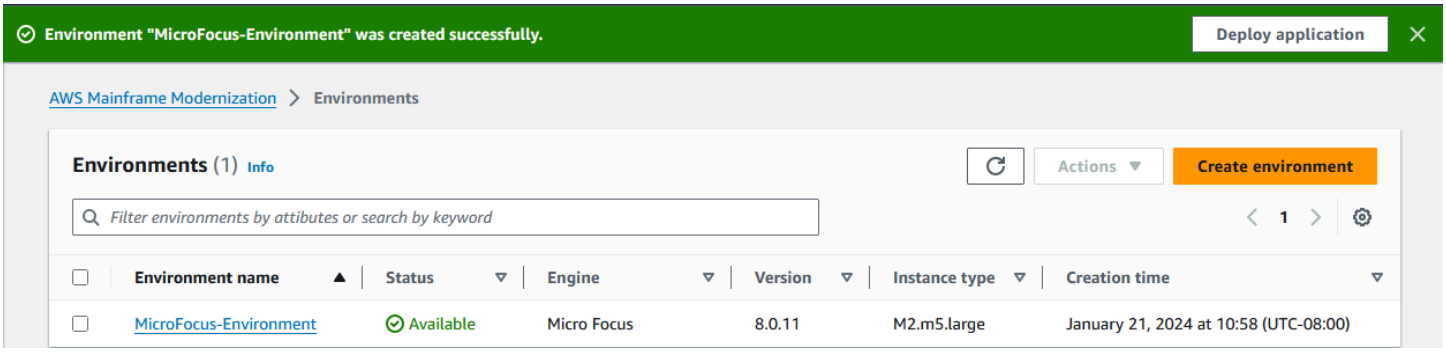
5. On the **Attach storage** page, choose **Next**.
6. On the **Schedule maintenance** page, choose **No preference** and then choose **Next**.



7. On the **Review and create** page, review all the configurations that you provided for the runtime environment, and then choose **Create environment**.



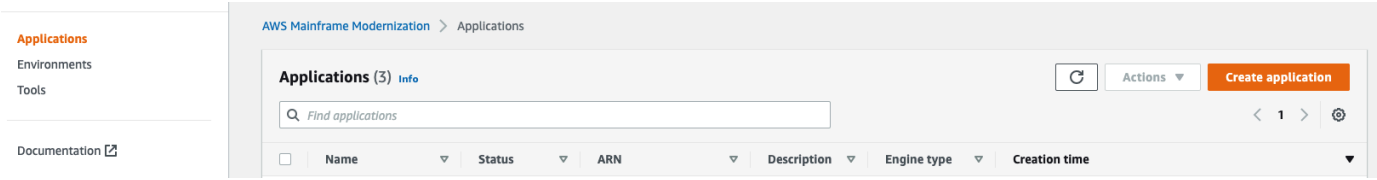
When you've created your environment, a banner appears that says `Environment` *`name`* `was created successfully`, and the **Status** field changes to **Available**. The environment creation process takes several minutes but you can continue with the next steps while it runs.



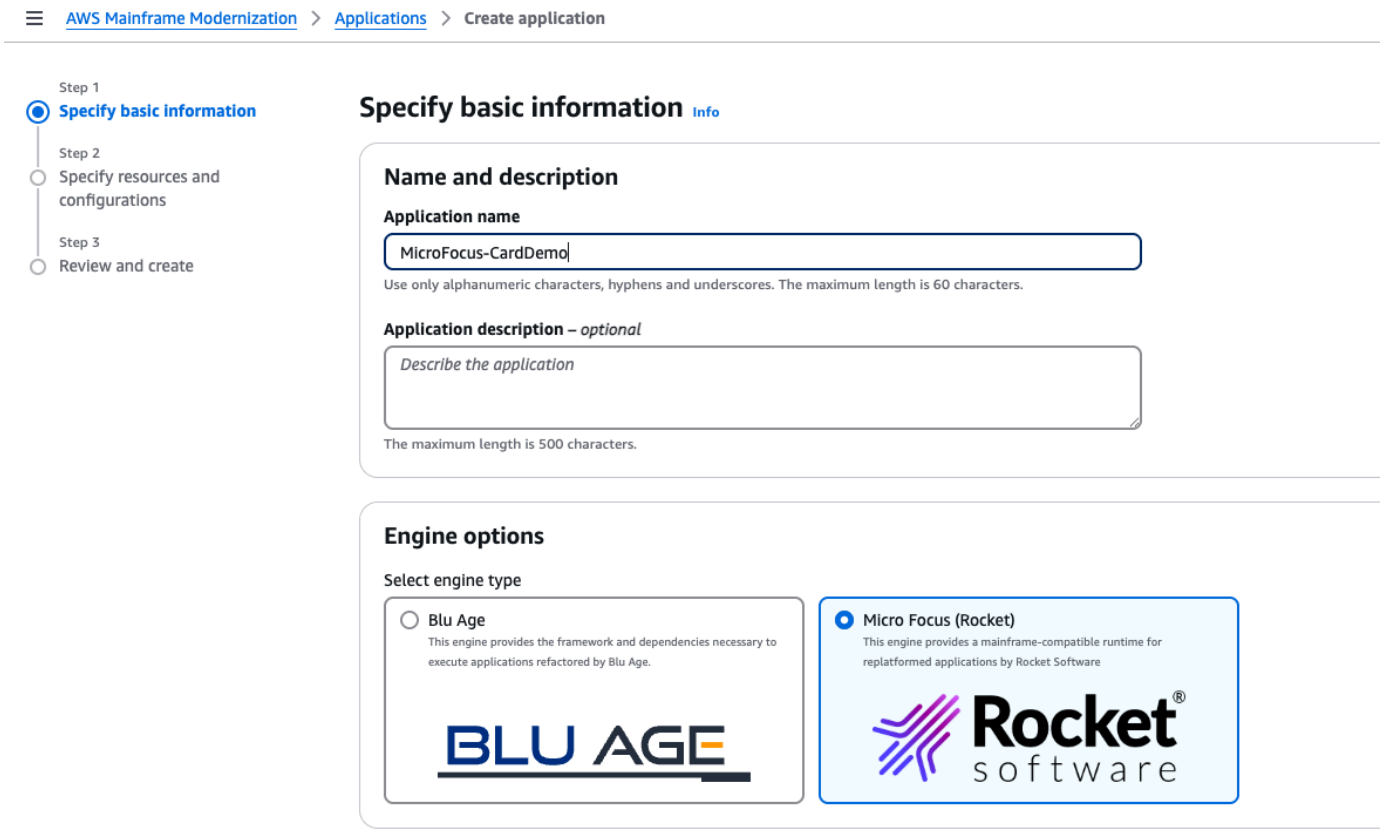
Step 7: Create an application

To create an application

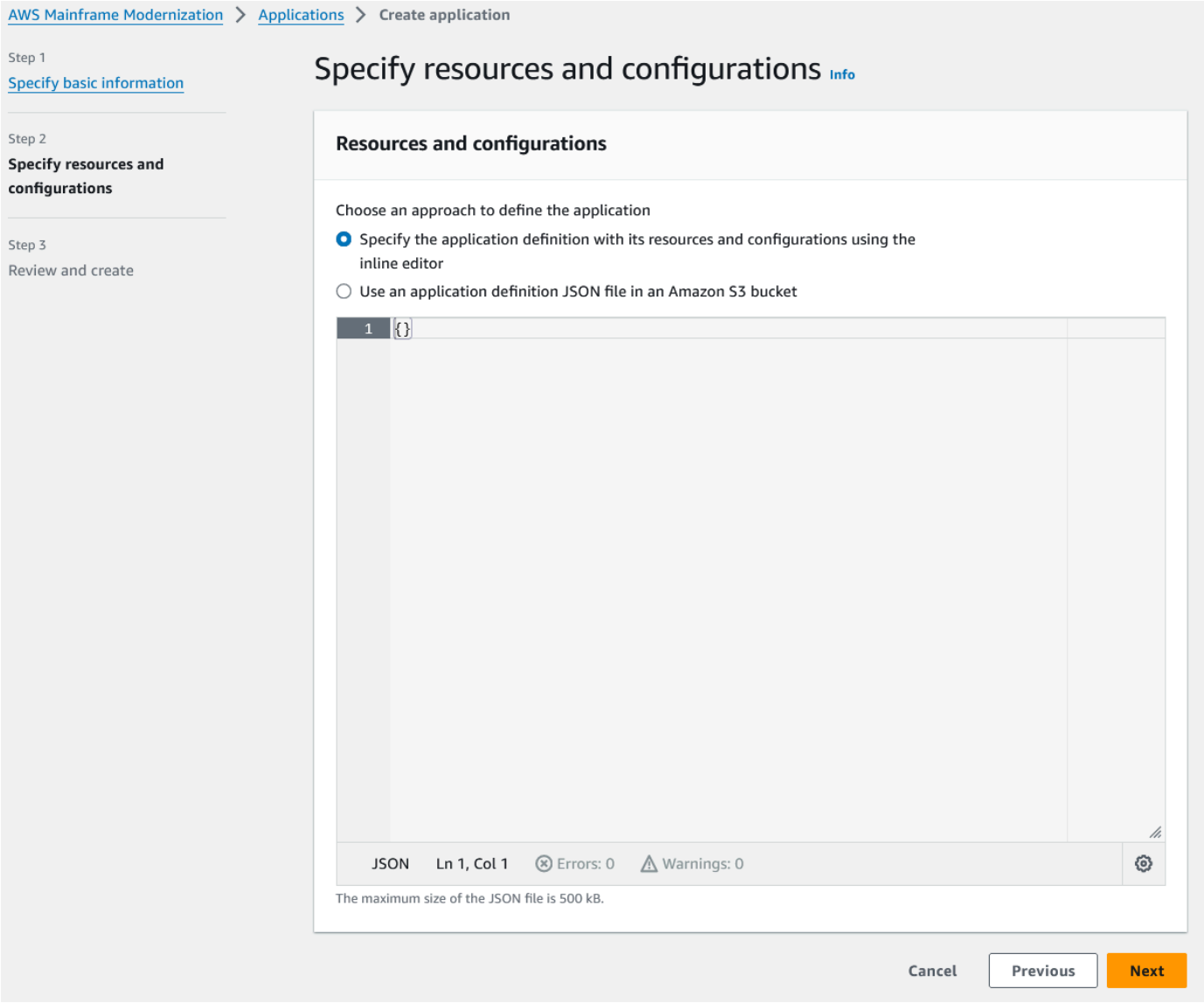
1. In the navigation pane, choose **Applications**. Then choose **Create application**.



2. On the **Create application** page, under **Specify basic information**, enter `MicroFocus-CardDemo` for the application name and under **Engine type** make sure **Micro Focus (Rocket)** is selected. Then choose **Next**.



3. Under **Specify resources and configurations**, choose the option to specify the application definition with its resources and configurations using the inline editor.



Enter the following application definition in the editor:

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

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

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

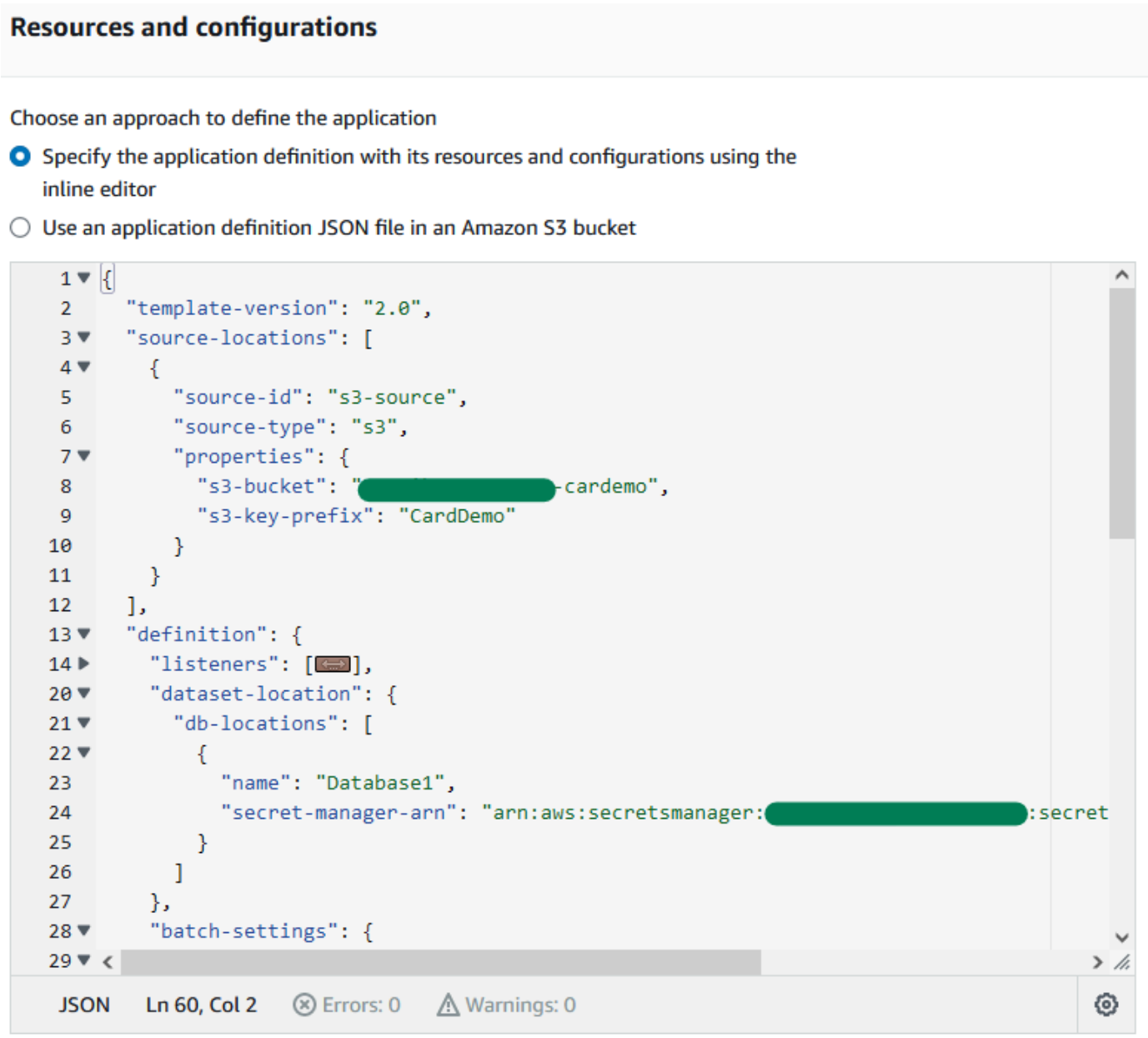
> **Note**
>
> This file is subject to change.

4. Edit the application JSON in the **properties** object of **source-locations** as follows:

   a. Replace the value for `s3_bucket` with the name of the Amazon S3 bucket that you created in Step 1.

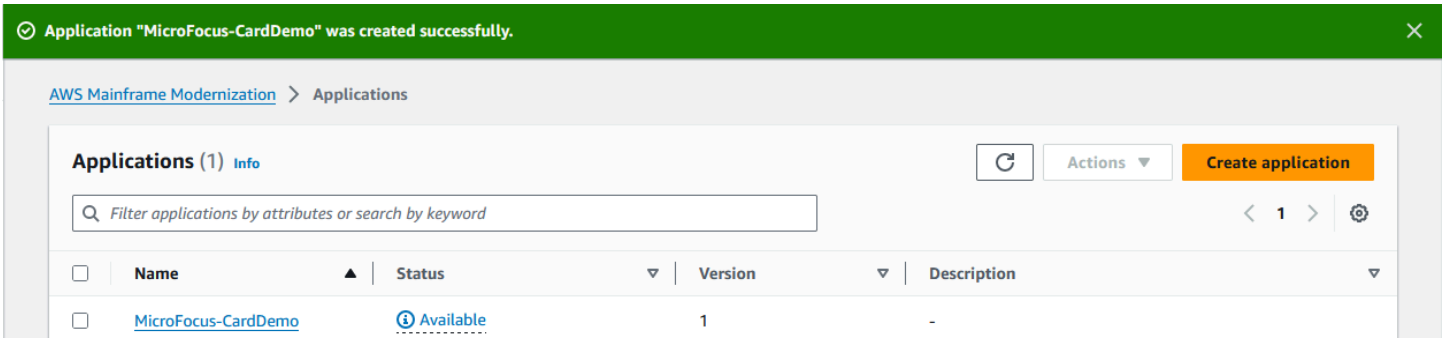
   b. Replace the value for `s3-key-prefix` with the folder (key prefix) where you uploaded the CardDemo sample files. If you uploaded the `CardDemo` directory directly to an Amazon S3 bucket, then the `s3-key-prefix` doesn't need to be changed.

   c. Replace both `secret-manager-arn` values with the ARN for the database secret that you created in Step 4.



For more information on the application definition, see [Rocket Software (formerly Micro Focus) application definition](#).

5. Choose **Next** to continue.
6. On the **Review and create** page, review the information that you provided, and then choose **Create application**.

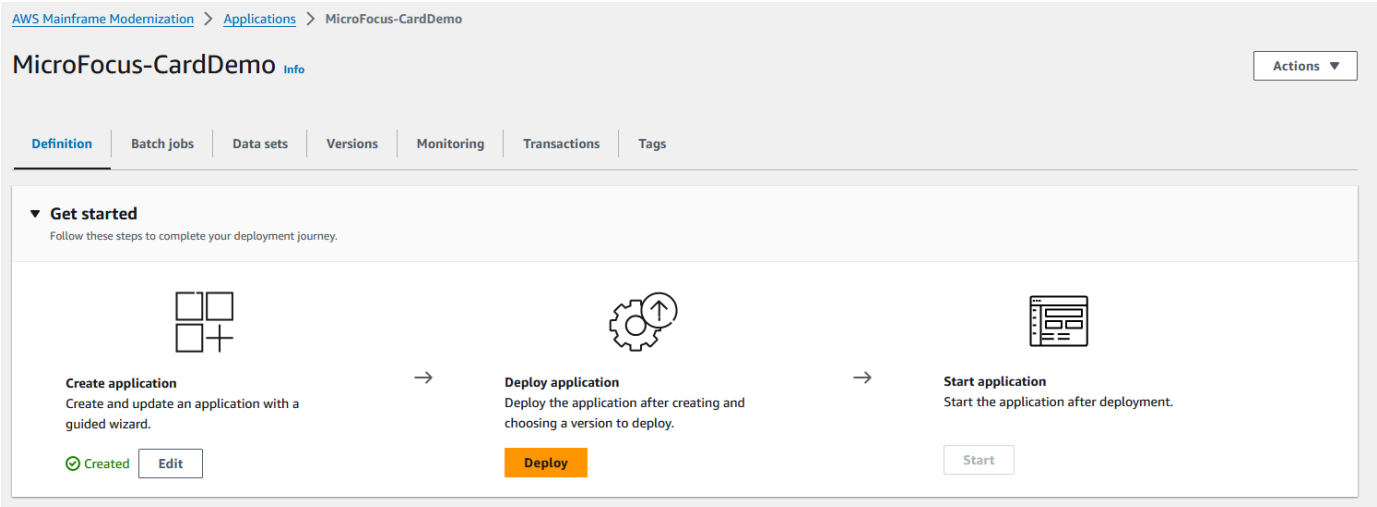


When you've created your application, a banner appears that says `Application` *`name`* `was created successfully`. And the **Status** field changes to **Available**.

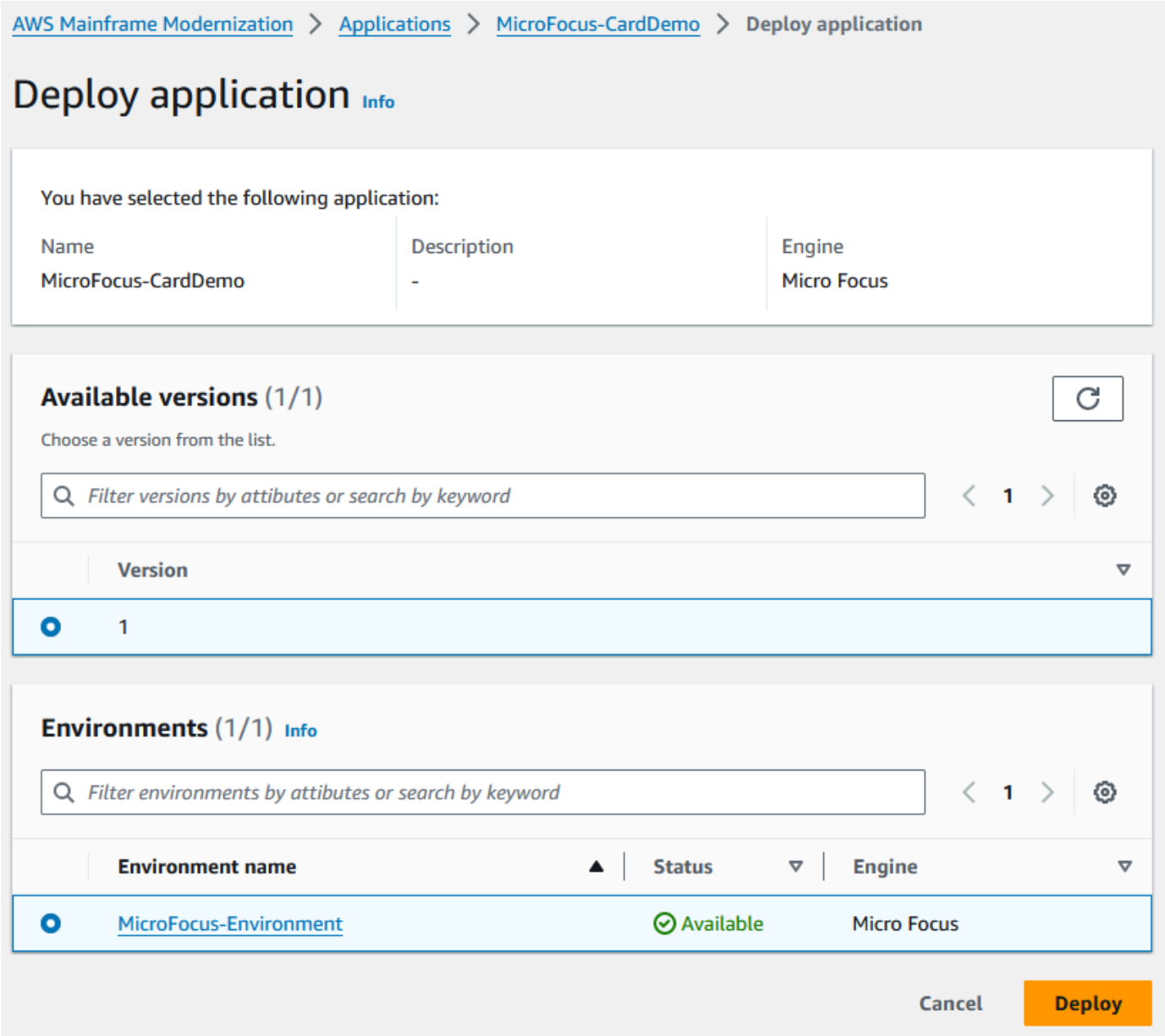
Step 8: Deploy an application

To deploy an application

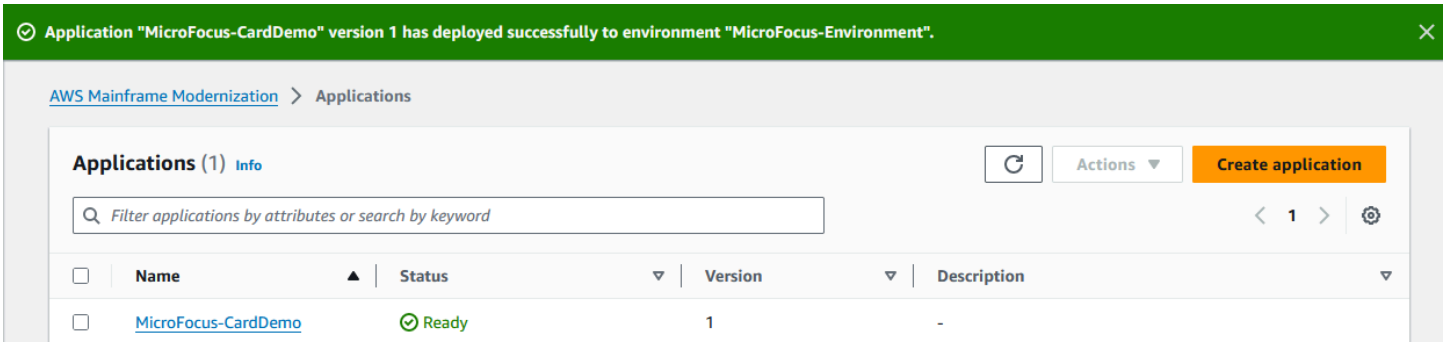
1. In the navigation pane, choose **Applications**, and then choose `MicroFocus-CardDemo`.
2. Under **Deploy application**, choose **Deploy**.



3. Choose the latest version of the application and the environment that you created previously, and then choose **Deploy**.



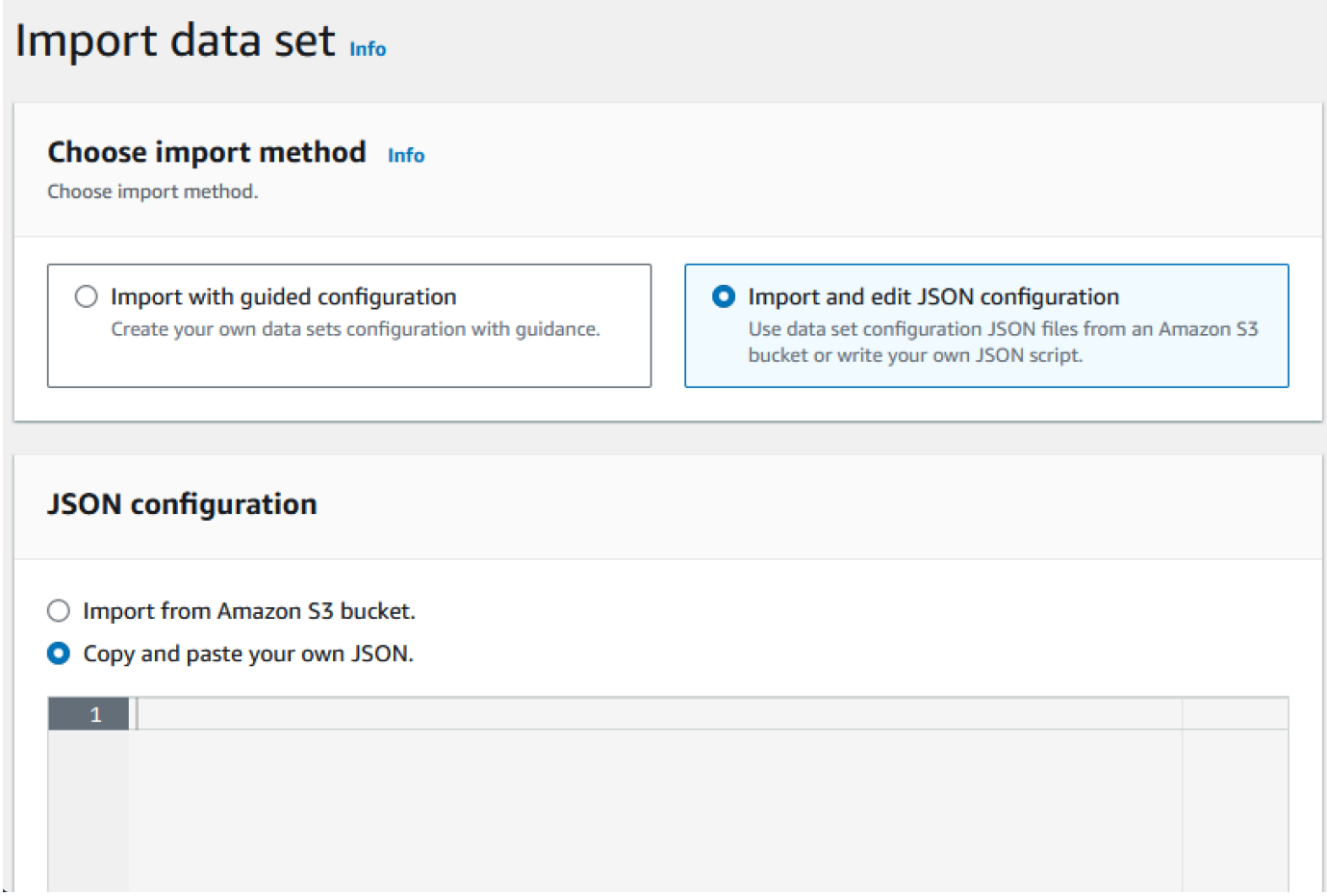
When the CardDemo application deploys successfully, the status changes to **Ready**.



Step 9: Import data sets

To import data sets

1. In the navigation pane, choose **Applications**, and then choose the application.
2. Choose the **Data sets** tab. Then choose **Import**.
3. Choose **Import and Edit JSON configuration**, and then choose the **Copy and paste your own JSON** option.



4. Copy and paste the following JSON but don't choose "Submit" yet. This JSON contains all the data sets required for the demo application but needs your Amazon S3 bucket details.

```
{
    "dataSets": [
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS",
                "relativePath": "DATA",
```

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

```
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 150,
                    "max": 150
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo_runtime/catalog/data/
AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 16,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 50,
                    "max": 50
```

```
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo_runtime/catalog/data/
AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 9,
                            "offset": 0
                        }
                    }
                },
                "recordLength": {
                    "min": 500,
                    "max": 500
                }
            },
            "externalLocation": {
                "s3Location": "s3://<s3-bucket-name>/CardDemo_runtime/catalog/data/
AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS.DAT"
            }
        },
        {
            "dataSet": {
                "storageType": "Database",
                "datasetName": "AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH",
                "relativePath": "DATA",
                "datasetOrg": {
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 11,
                            "offset": 25
```

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

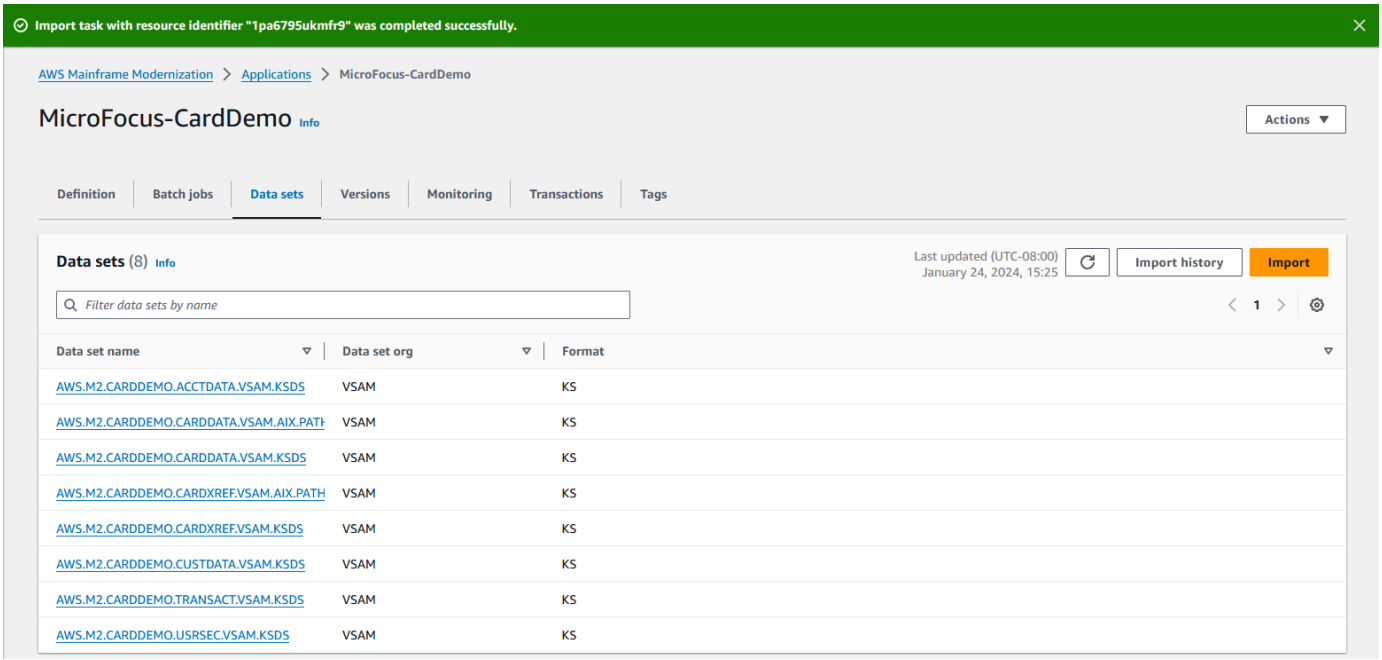
```
                    "vsam": {
                        "format": "KS",
                        "encoding": "A",
                        "primaryKey": {
                            "length": 8,
                            "offset": 0
                        }
                    }
                },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

5. Replace each occurrence of `<s3-bucket-name>` (there are eight) with the name of the Amazon S3 bucket that contains the CardDemo folder, for example, `your-name-aws-region-carddemo`.

> **Note**
>
> To copy the Amazon S3 URI for the folder in Amazon S3, select the folder, and then choose **Copy Amazon S3 URI**.

6. Choose **Submit**.

   When the import finishes, a banner appears with the following message: `Import task with resource identifier` *name* `was completed successfully.` A list of the imported datasets is shown.

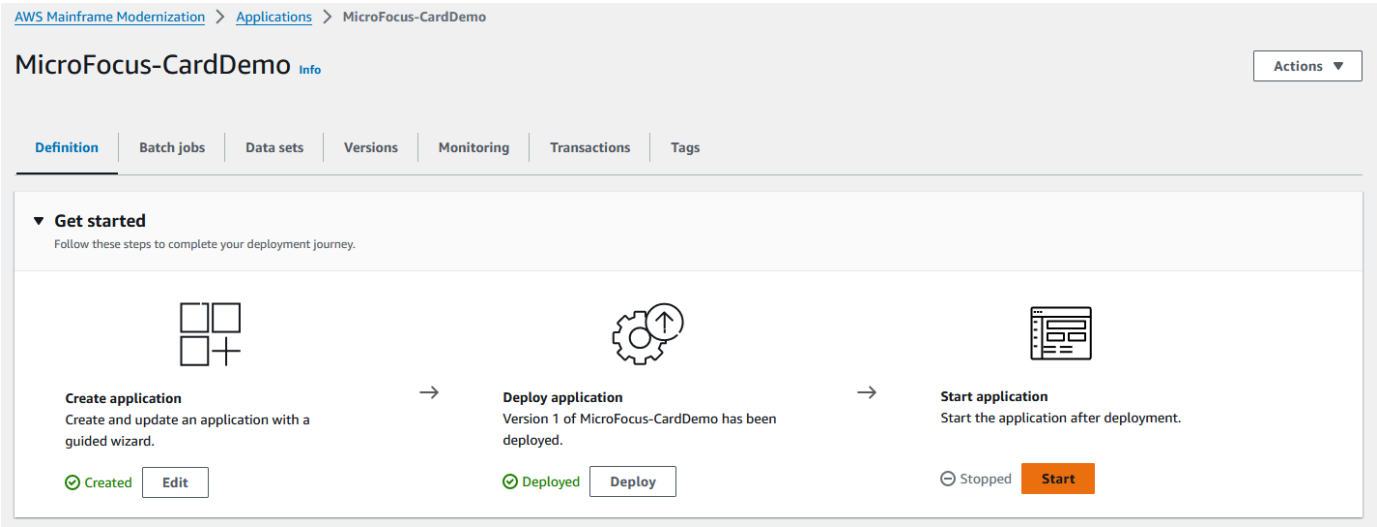


You can also view the status of all data set imports by choosing **Import History** on the **Data sets** tab.

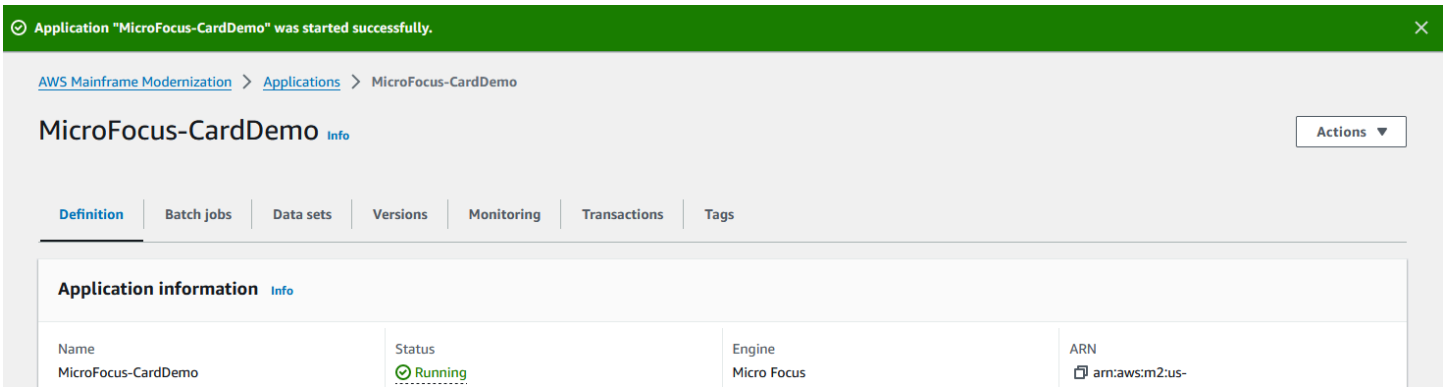
Step 10: Start an application

To start an application

1. In the navigation pane, choose **Applications**, and then choose the application.
2. Choose **Start application**.



When the CardDemo application starts to run successfully, a banner appears with the following message: `Application` *`name`* `was started successfully`. The **Status** field changes to **Running**.



Step 11: Connect to the CardDemo CICS 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 for your network interface that you will connect from.

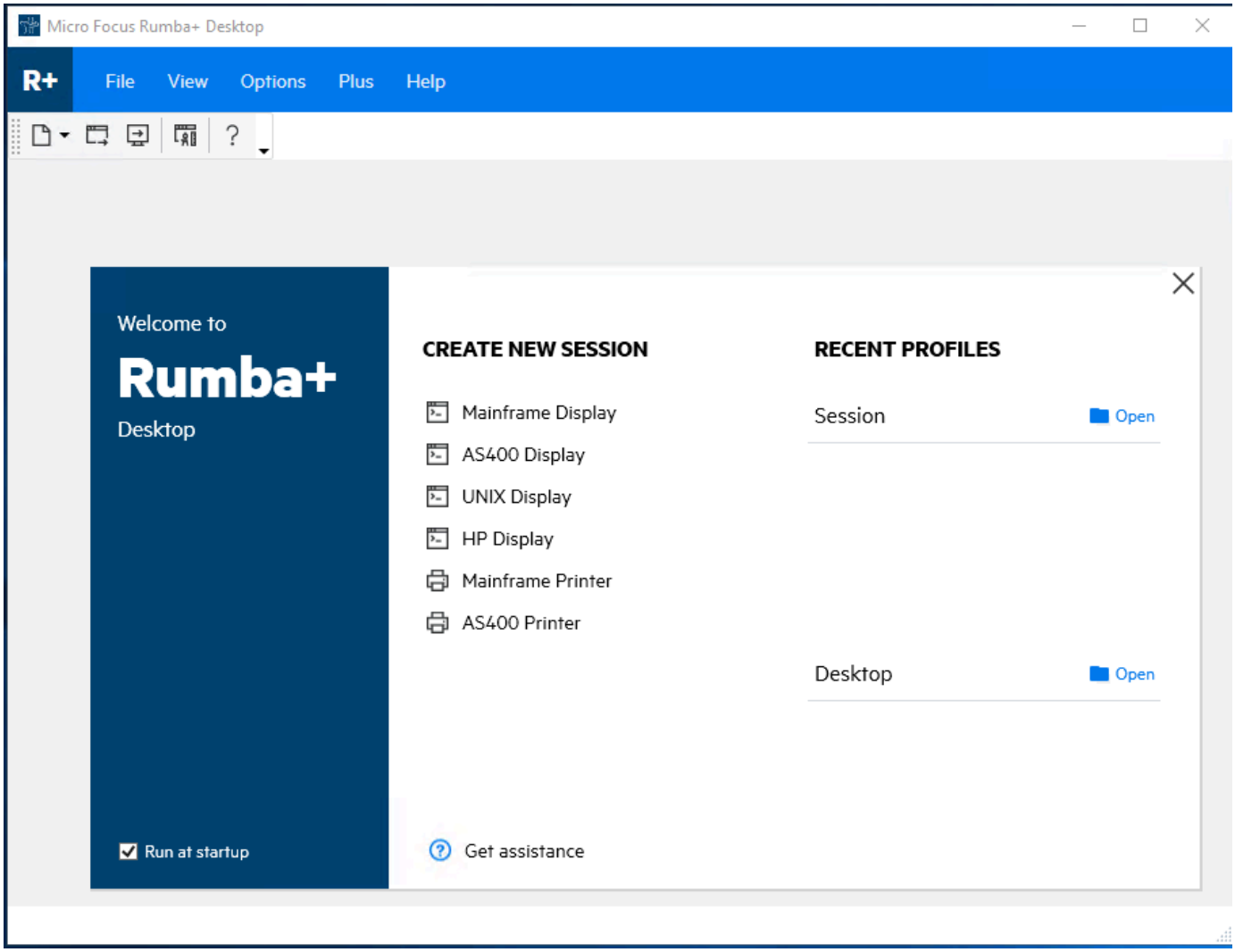
To configure the TN3270 connection, you also need the DNS hostname and the port of the application.

To configure and connect an application to mainframe using terminal emulator

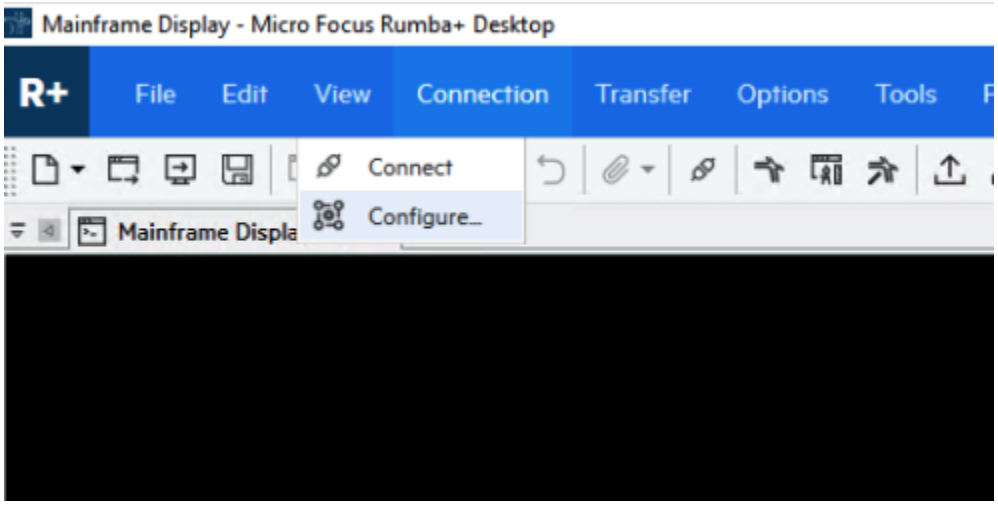
1. Open the AWS Mainframe Modernization console and choose **Applications**, and then choose `MicroFocus-CardDemo`.
2. Choose the copy icon to copy the **DNS Hostname**. Also make sure to note the **Ports** number.
3. Start a terminal emulator. This tutorial uses Micro Focus Rumba+.

   > **Note**
   >
   > The configuration steps vary by emulator.

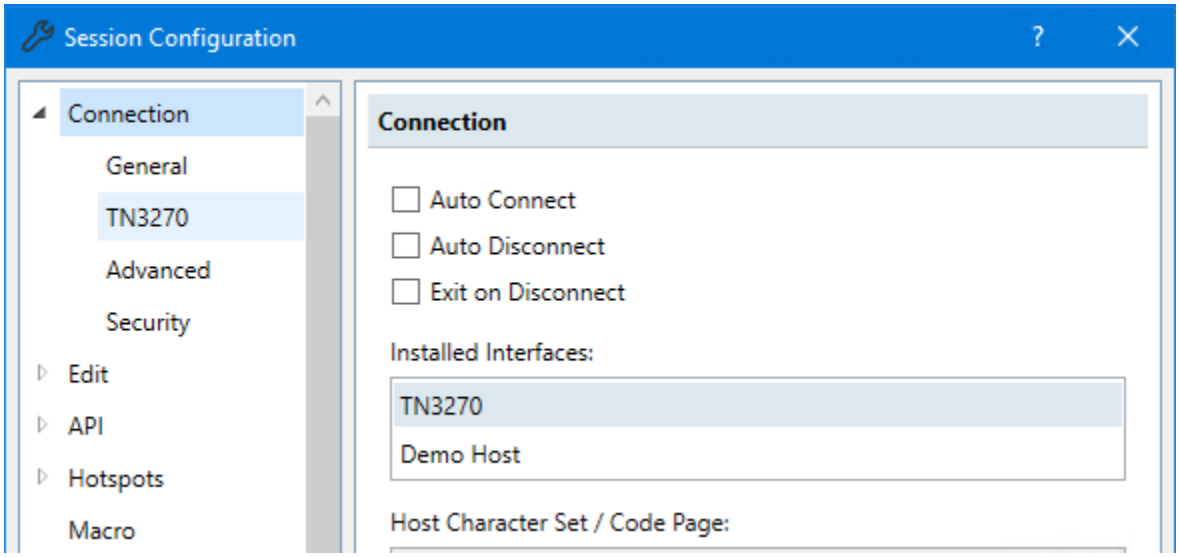
4. Choose **Mainframe Display**.



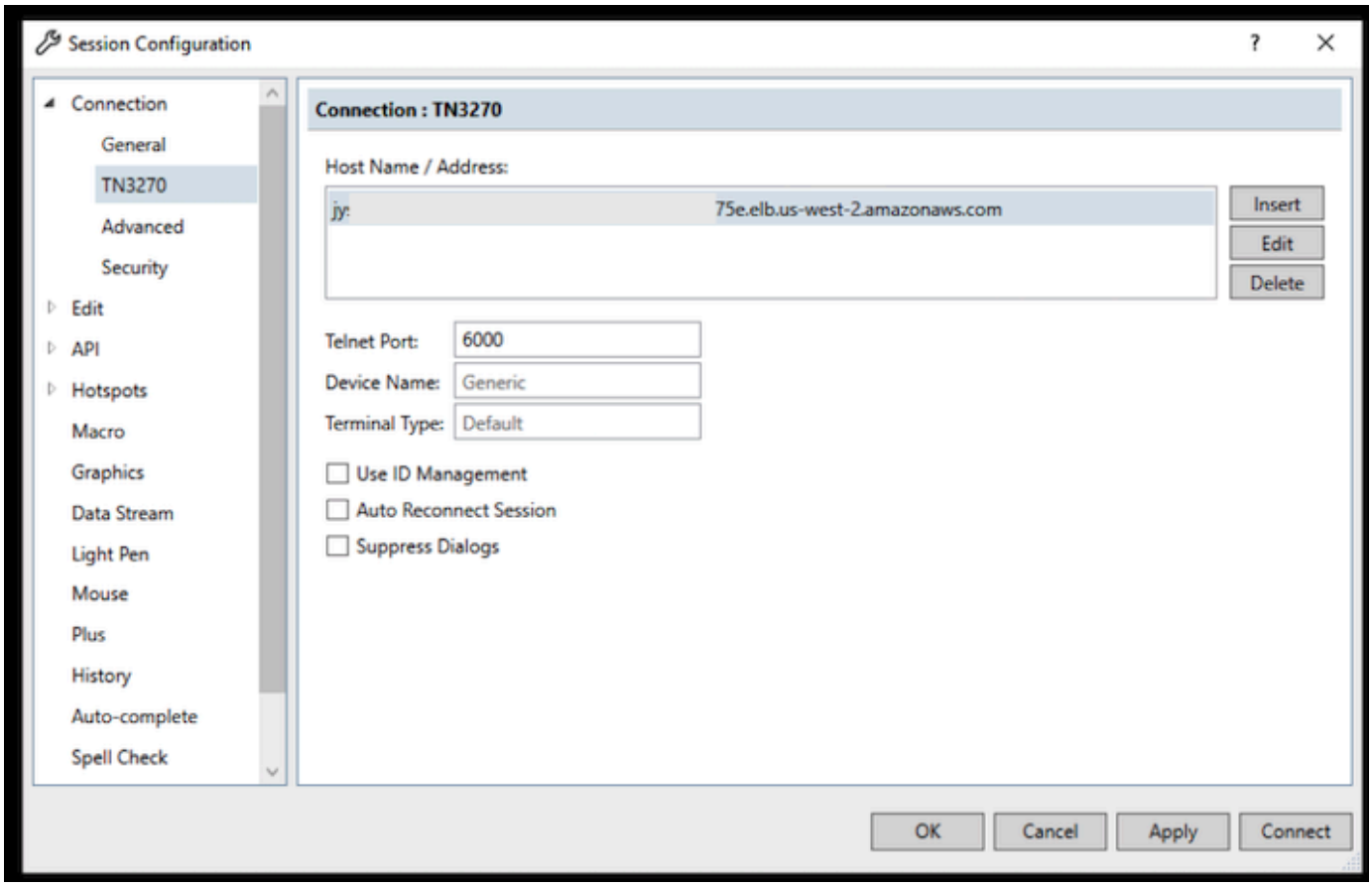
5. Choose **Connection**, and then choose **Configure.**



6. Under **Installed Interfaces**, choose TN3270, and then choose TN3270 again under the **Connection** menu.



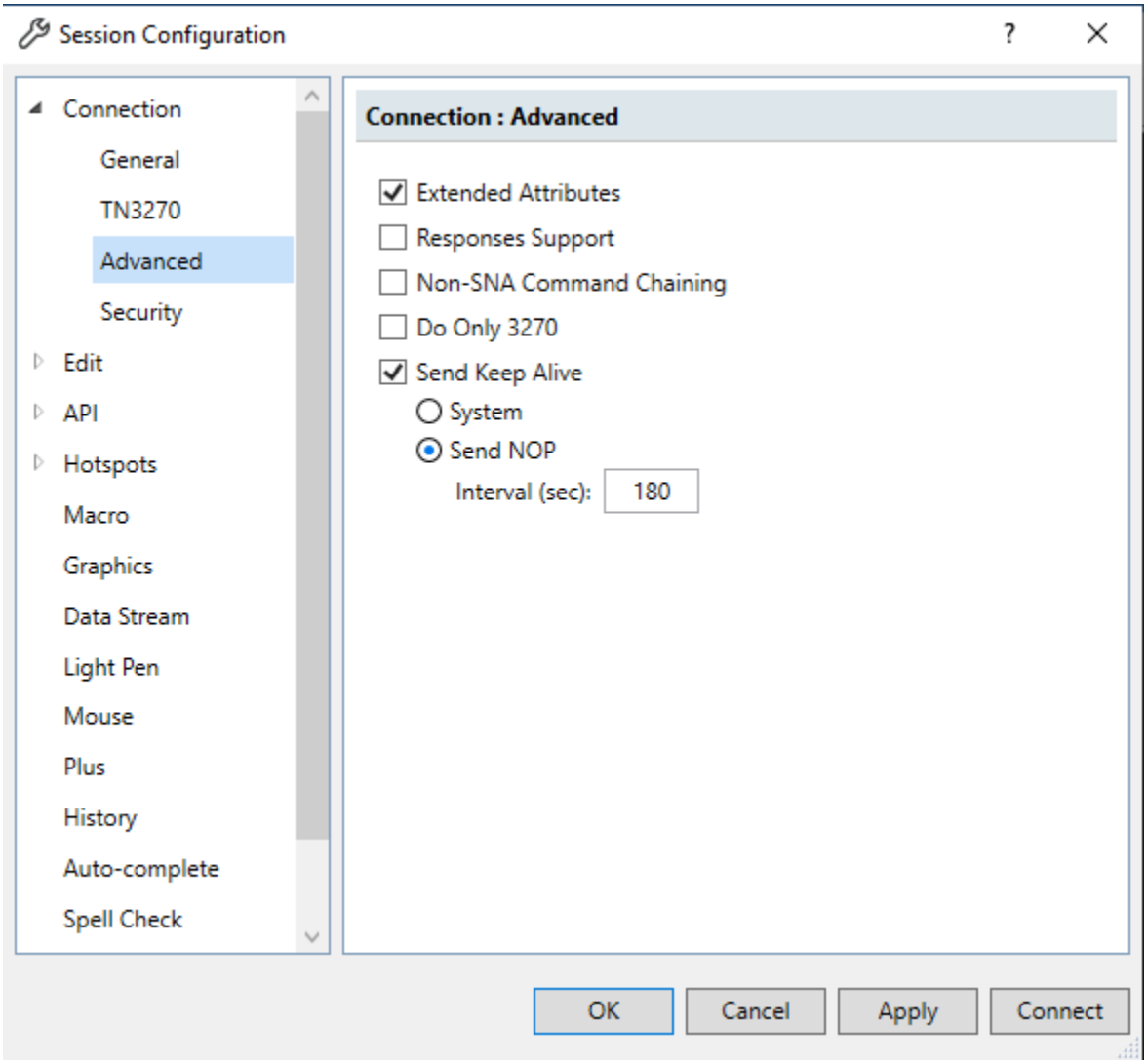
7. Choose **Insert**, and paste the `DNS Hostname` for the Application. Specify `6000` for the **Telnet Port**.



> **Note**
>
> If you are using AWS AppStream 2.0 in a browser and having difficulties with pasting values, please refer to Troubleshooting AppStream 2.0 User Issues.

8. Under **Connection**, choose **Advanced**, and then choose **Send Keep Alive** and **Send NOP**, and enter **180** for the **Interval.**

> **Note**
>
> Configuring the keep alive setting on your TN3270 terminal to at least 180 seconds helps ensure that the Network Load Balancer doesn't drop your connection.



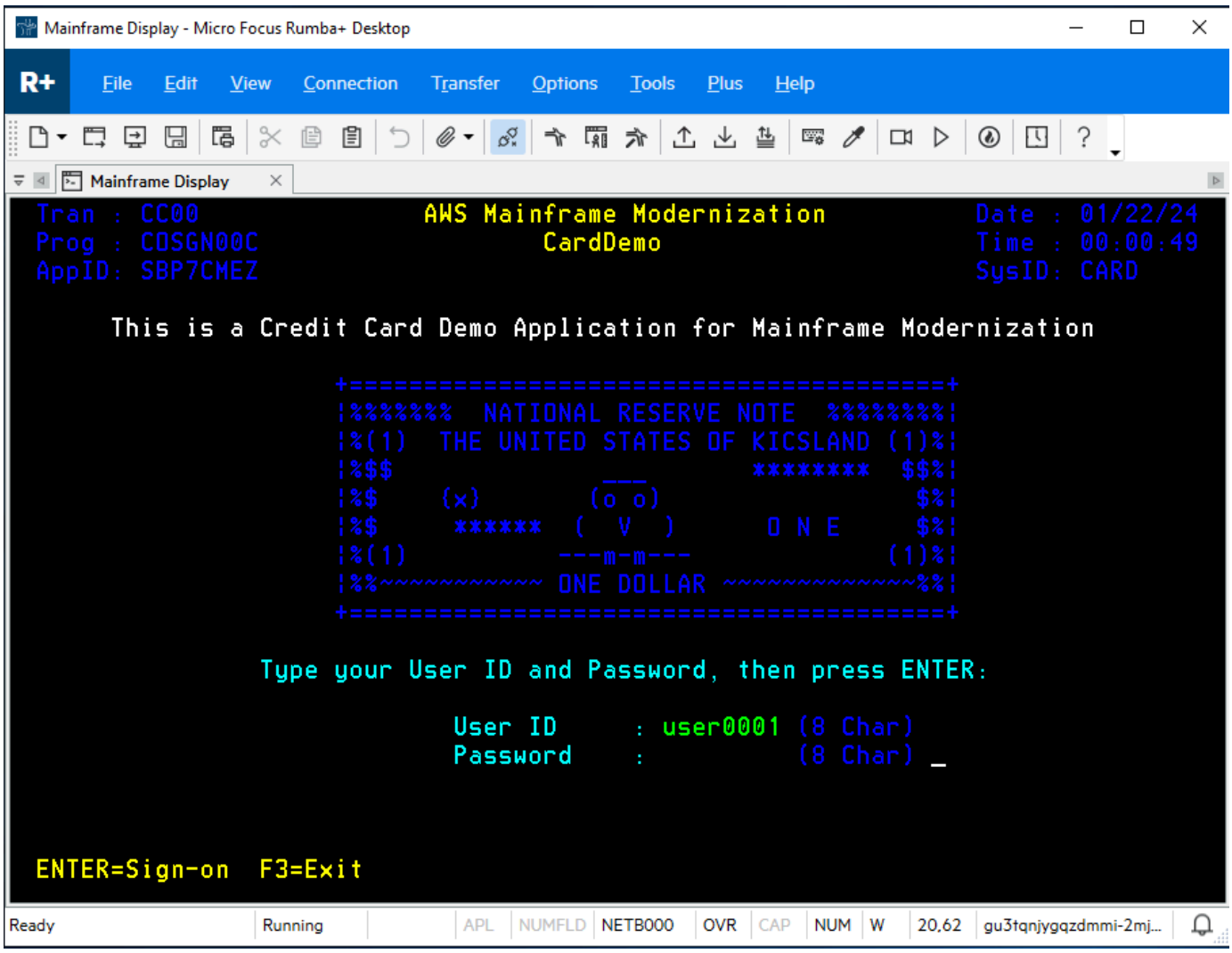
9. Choose **Connect**.

> **Note**
>
> If the connection fails:
>
> - If you are using AppStream 2.0, confirm that the VPC and security group specified for the application's environment are the same as the AppStream 2.0 fleet.
> - Use the VPC Reachability Analyzer to analyze the connection. You can access the Reachability Analyzer through the console.
> - As a diagnostic step, try adding or changing the Security Group inbound rules for the application to allow traffic for port 6000 from anywhere (i.e. CIDR Block 0.0.0.0/0). If

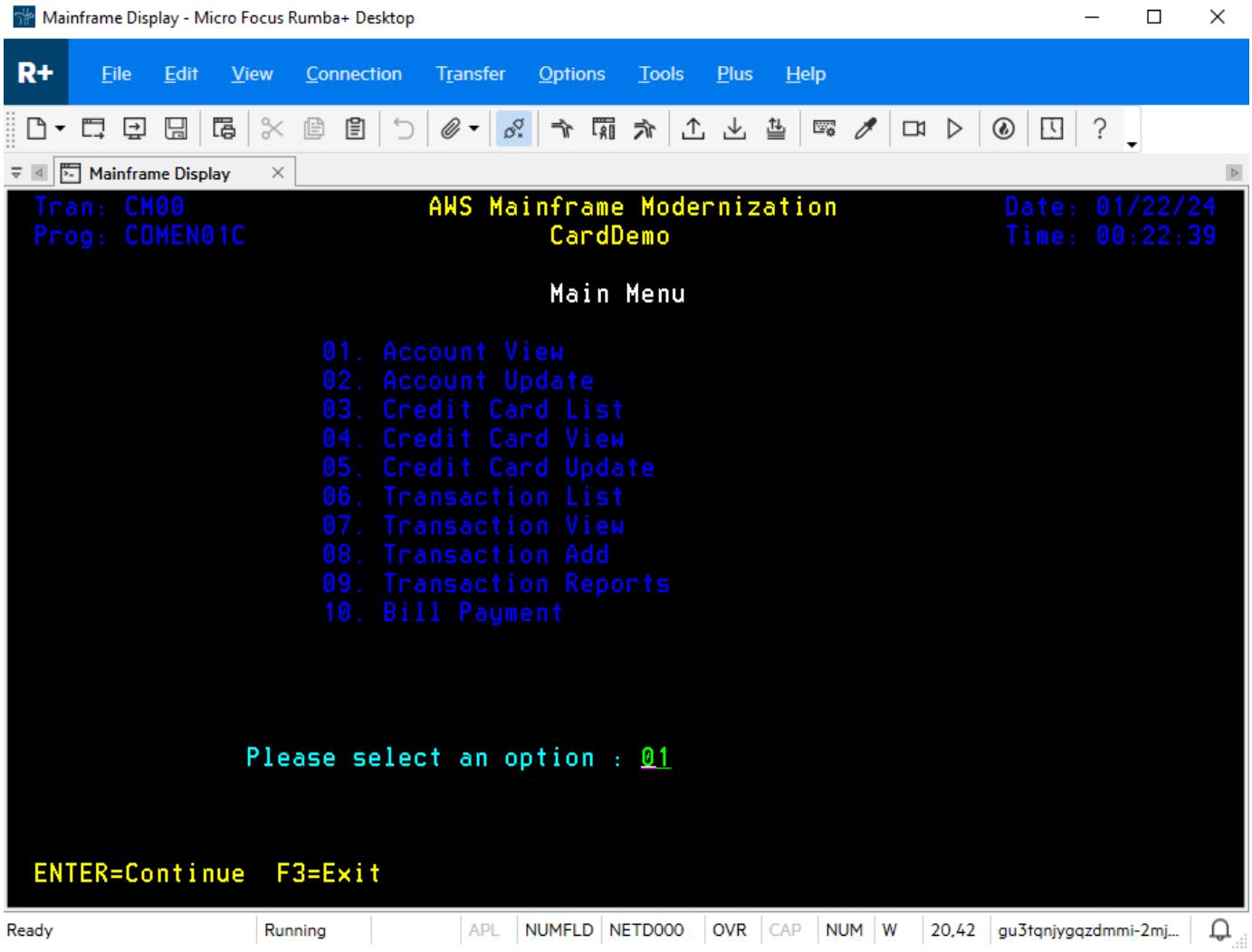
you successfully connect, then you know the security group was blocking your traffic. Change the security group source to something more specific. For more information on security groups, see Security group basics.

10. Enter USER0001 for the username and password for the password.

> **Note**
>
> In Rumba, the default for Clear is ctrl-shift-z, and the default for Reset is ctrl-r.

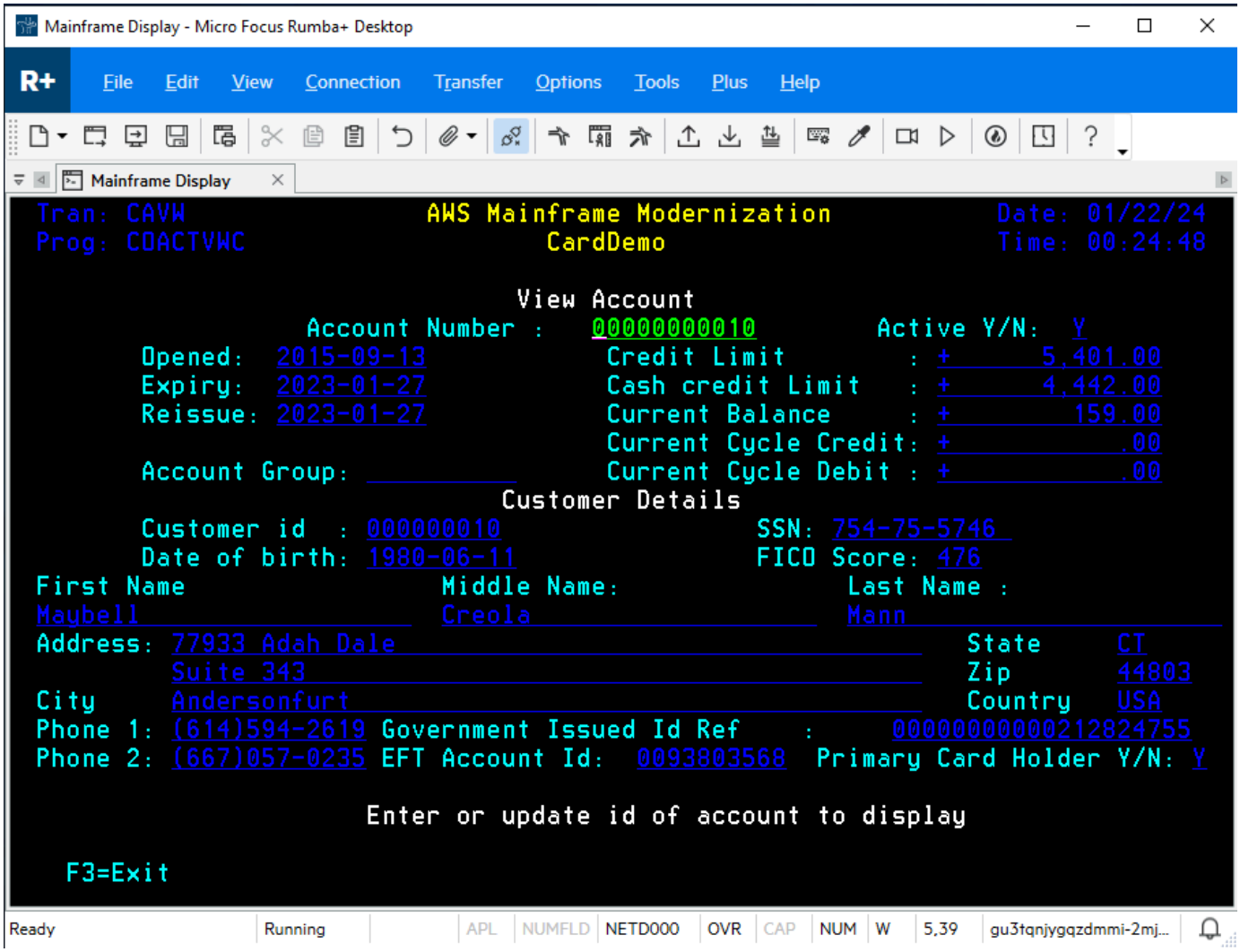


11. After you log in successfully, you can navigate through the CardDemo application.
12. Enter 01 for the Account View.



13. Enter `00000000010` for the Account Number and press **Enter** on your keyboard.

> **Note**
>
> Other valid accounts are `0000000011` and `00000000020`.



14. Press **F3** to Exit to the menu, and **F3** to exit the transaction.

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 application.
- Delete the application. For more information, see Delete an AWS Mainframe Modernization application.
- Delete the runtime environment. For more information, see Delete an AWS Mainframe Modernization runtime environment.

- Delete the Amazon S3 buckets that you created for this tutorial. For more information, see Deleting a bucket in the *Amazon S3 User Guide*.
- Delete the AWS Secrets Manager secret that you created for this tutorial. For more information, see Delete a secret.
- Delete the KMS key that you created for this tutorial. For more information, see Deleting AWS KMS keys.
- Delete the Amazon RDS database that you created for this tutorial. For more information, see Delete the EC2 instance and DB instance in the *Amazon RDS User Guide*.
- If you added a Security Group rule for port 6000, delete the rule.

Next steps

To learn how to set up a development environment for your modernized applications, see Tutorial: Set up AppStream 2.0 for use with Rocket Enterprise Analyzer and Rocket Enterprise Developer.

AWS Mainframe Modernization components lifecycle

Each component of AWS Mainframe Modernization goes through version upgrades and a development lifecycle. You can use this page as an overview to understand these components, their version upgrade plans, and how AWS Mainframe Modernization communicates the release or deprecation of these components or their versions.

Components lifecycle overview

AWS Mainframe Modernization lifecycle describes the approach and timelines for releasing and supporting AWS Mainframe Modernization service components throughout their lifecycle. Providing predictable and consistent lifecycle helps you as you plan, test, and deploy newer versions.

All AWS-provided AWS Mainframe Modernization components benefit from the product support provided by Support from the time they are released until their retirement per each component's release calendar table. You can learn more about the Support scope and activities at Compare Support Plans. During active modernization projects, we typically encourage customer support to be provided first by professional services delivery teams as per the statement of work.

AWS Mainframe Modernization releases some components with versions originating from suppliers which can be AWS itself, select AWS Partners, or communities. For each AWS Mainframe Modernization component, a version has a major version number and a minor version number. Each component has its own major and minor version numbering.

For versioned components, we have the following intents:

- To release AWS Mainframe Modernization **components newer versions on a regular basis** or per customer demand. If a component's newer version is desired and not yet available in the AWS Mainframe Modernization service, you can make an explicit request via Support Product Feature Request (PFR).
- To have AWS Mainframe Modernization component-specific versions' **end of support and retirement dates align** with the component supplier end of support dates.
- To **notify customers** approximately one year prior to the retirement of a component's major version.

While we strive to meet these guidelines, in some cases, we may retire specific versions sooner with shorter notification timeframes. For example, we may retire a version with security issues promptly with a shorter notification timeframe. We may also retire minor versions early when a minor version has significant bugs or security issues that have been resolved in a later minor version. In the unlikely event that such cases occur, we will notify customers and communicate about the plan and the timeline for retirement. Specific circumstances may dictate different timelines depending on the situation.

> **Note**
>
> Critical updates to components might be made available at any time. For example, new versions may be made available promptly for security reasons or to provide fixes for the production environments. For requests made through Support, the support plan dictates the processes, severity, and the response times.

When a component version is retired, AWS Mainframe Modernization doesn't distribute these versions to customers for new deployments. Consequently, these versions are also unsupported by Support. Customers running existing component deployments past their version retirement dates should be aware of the risks of doing so. AWS is not responsible to provide security updates, technical support, or hot fixes for retired component versions. Also, we don't automatically remove access or delete your environment's resources. We strongly encourage you check for new versions every 3 months, and upgrade all your AWS Mainframe Modernization components to recent supported versions.

Version upgrade

AWS Mainframe Modernization provides newer versions of each supported component so you can stay up-to-date with the latest maintenance updates and features. Newer versions can include bug fixes, security enhancements, and other improvements for the components. We recommend you should upgrade regularly to benefit from security fixes, bug fixes, and feature enhancements. When AWS Mainframe Modernization releases a new version, you can choose how and when to upgrade your existing deployments. There are two kinds of upgrades: major version upgrades and minor version upgrades. In general, a major engine version upgrade can introduce changes that aren't compatible with existing applications. In that case, substantial application changes may be required for a major version upgrade. In contrast, a minor version upgrade includes changes that

are mostly backward-compatible with existing applications. Little to no changes may be required for a minor version upgrade.

You should perform non-regression tests prior to performing components' version upgrades. It's best practice to use DevOps test and deployment pipelines. DevOps test pipelines can be built during modernization projects, and should be maintained to automate application testing when performing component upgrades and application code changes. You can also use blue/green deployments or canary deployment during upgrades. You can learn more about such deployments and change management at AWS Well-Architected Reliability Pillar.

AWS Mainframe Modernization Refactor with AWS Blu Age release overview

With AWS Blu Age runtime, the version follows a `Major.Minor.Patch` pattern. For example, for AWS Blu Age runtime version `4.1.0`, the major version is 4, the minor version is 1, and the patch version is 0.

We intend to release new AWS Blu Age runtime major versions when there are impactful changes to runtime or their dependencies. AWS Blu Age runtime major versions are supported for at least 12 months unless some **Common Vulnerabilities and Exposures (CVEs)** appear. The support covers for bugs in the runtime features as mentioned in our documentation. In case of Critical and High CVEs in the dependencies of the runtime (Spring, Java, Tomcat, and others), the major version support duration is reduced to 6 months for High CVEs, and 3 months for Critical CVEs from the release date of the new runtime version fixing the CVE, unless explicitly stated otherwise.

We intend to release new AWS Blu Age minor versions monthly. Customers are expected to upgrade versions regularly to obtain the latest security fixes, bug fixes, and feature enhancements. Active projects not yet in production have to adopt the latest runtime version as soon as it becomes available.

New fixes are provided in the latest minor version for the particular major version where an issue is raised. If you require new fixes, you need to upgrade to a new minor version to apply those fixes.

Patched versions for supported releases are provided only to address critical runtime defects not present in previous supported minor versions.

Alpha pre-releases are short-lived versions made available for quick iteration during delivery projects. Fixes for issues identified in alpha pre-releases are provided in the later minor versions, as no patches are delivered for Alpha pre-release versions.

You can find release dates and details about each runtime version in the the section called “AWS Blu Age release notes”.

Security scans are performed by Amazon Inspector.

Understand managed 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?
- Set up for AWS Mainframe Modernization
- Tutorial: Set up managed runtime for AWS Blu Age
- Tutorial: Set up managed runtime for Rocket Software (formerly Micro Focus)

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 Rocket Software (formerly Micro Focus) if you are using the replatforming pattern.

Topics

- Create AWS resources for a migrated application
- Create an AWS Mainframe Modernization application
- Deploy an AWS Mainframe Modernization application
- Update an AWS Mainframe Modernization application
- Delete an AWS Mainframe Modernization application
- Submit batch jobs for AWS Mainframe Modernization applications
- Cancel batch jobs for AWS Mainframe Modernization applications
- Import data sets for AWS Mainframe Modernization applications
- Export data sets for AWS Mainframe Modernization applications
- Manage transactions for AWS Mainframe Modernization applications
- Configure the Rocket Software (formerly Micro Focus) managed application

- Configure the AWS Blu Age managed application
- AWS Mainframe Modernization application definition reference
- AWS Mainframe Modernization data set definition reference

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 Amazon 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.

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 Rocket Software 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:

```
{
   "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.

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

Create an AWS Mainframe Modernization application

Use the AWS Mainframe Modernization console to create an AWS Mainframe Modernization application. Creating an application allows you to perform tasks with the migrated mainframe workload.

These instructions assume that you have completed the steps in Set up for AWS Mainframe Modernization.

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 (Rocket)** 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.

   > **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 or Rocket Software (formerly Micro Focus) application definition.
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. You need to deploy your applications on a runtime environment before performing tasks.

These instructions assume that you have completed the steps in Set up for AWS Mainframe Modernization.

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/.
2. In the AWS Region selector, choose the Region where you want to create the application.
3. On the **Applications** page, choose the application that you want to deploy.
4. Choose **Deploy application**.
5. In the **Available versions** section, choose the version that you want to deploy.

6. In the **Environments** section, choose a runtime environment where you want your application to run.
7. Choose **Deploy**.

To deploy a different version of a deployed 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 the application that you want to deploy.
4. From the **Actions** menu, choose **Stop application**.
5. After the application stops, choose **Deploy application**.
6. In the **Available versions** section, choose the version that you want to deploy. In the **Environments** section, the environment that the application is already deployed in is preselected.
7. Choose **Deploy**.

Update an AWS Mainframe Modernization application

Use the AWS Mainframe Modernization console to update an AWS Mainframe Modernization application. Updating an application creates a new version of the application.

These instructions assume that you have completed the steps in Set up for AWS Mainframe Modernization.

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 or Rocket Software (formerly Micro Focus) application definition.
6. When you're finished updating the application definition, choose **Update**.

> **Note**
>
> After you update the application, you must deploy it again. For more information, see Deploy an AWS Mainframe Modernization application.

Delete an AWS Mainframe Modernization application

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 Set up for AWS Mainframe Modernization.

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 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.

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 run of the batch job. The batch job records remain available for you to view in the details for the batch job run.

If your batch job requires access to one or more data sets, use the AWS Mainframe Modernization console 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 Set up for AWS Mainframe Modernization and in Create an AWS Mainframe Modernization application.

Topics

- Submit a batch job
- Restart a batch job

Submit a batch job

To submit a batch job

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 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**.

Restart a batch job

To restart a batch job

> **Important**
>
> A batch job restart is available on the following engine versions:
>
> - Micro Focus (Rocket) environment engine versions 8.0.6 or greater. You also need to have an EFS or FSx file system attached to your environment.
> - AWS Blu Age environment engine versions 4.3.0 or greater. You also need to have an EFS or FSx file system attached if it is HA 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 and your batch job was created.
3. On the **Applications** page, choose the application where you want to restart a batch job.
4. On the application details page, choose **Batch jobs**.
5. Select the batch job you want to restart from the generated list. Navigate to the **Actions** menu, and choose **Restart job**.

6. Specify how you want to restart the batch job. You can do the following for Micro Focus (Rocket) environment engine and AWS Blu Age environment engine:

   - For Micro Focus (Rocket) environment engine, you can either choose to **Restart from the beginning** or **Restart using steps or procsteps**.
     - **Restart from the beginning** option allows you to restart all steps of a batch job from the beginning.
     - **Restart using steps or procsteps** option allows you to choose a specific step or procedure step you want to restart, and optionally a step or procedure step you want to end.

   > **Note**
   >
   > The end step or procstep must be greater than or equal to the start step or procstep number.

   - For AWS Blu Age environment engine, you can either restart a batch job's most recent execution from a previously failed JCL/PROC step or perform a delayed restart by bypassing previously successful steps.
     - You can choose a specific **Step name** that you want to restart.
     - Optionally, you can use **Skip step** to bypass the selected step and restart from the next step in the wokflow.

7. Choose **Submit job**.

Cancel batch jobs for AWS Mainframe Modernization applications

In AWS Mainframe Modernization you can cancel batch jobs for your applications. You can 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 run of the batch job. The batch job records remain available for you to view in the details for the batch job run.

Cancel a batch job

When you cancel a batch job, it does not delete a batch job, but the running of tasks for that batch job. You can still view details of your batch job.

To cancel a batch job

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the AWS Region selector, choose the Region with the application for your batch jobs.
3. From the batch job list find and select the batch job you want to cancel.
4. Choose **Actions**, and choose **Cancel job**.
5. Choose **Cancel batch job**.

This will cancel any batch job tasks you had scheduled for running.

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 to be imported 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 Set up for AWS Mainframe Modernization and in Create an AWS Mainframe Modernization application.

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" 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**.

Export data sets for AWS Mainframe Modernization applications

With AWS Mainframe Modernization you can export data sets to use with your applications. You can specify the data sets to be exported in a JSON file stored in an Amazon S3 bucket, or you can specify data set configuration values separately. After you export the data sets, you can review the details of the export task to confirm that the data sets that you wanted were exported.

Use the AWS Mainframe Modernization console to export data sets for a AWS Mainframe Modernization application.

These instructions assume that you have completed the steps in Set up for AWS Mainframe Modernization and in Create an AWS Mainframe Modernization application.

Export a data set

To export a dataset

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 export data sets for.
4. On the application details page, choose **Data sets**.
5. Choose **Export**.
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. For more information, see the section called "Data set definition reference".

     Enter the dataset name, 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 Set up for AWS Mainframe Modernization and in Create an AWS Mainframe Modernization application.

Manage 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 Region where the application that you want to run was created.
3. On the **Applications** page, choose the application where you want to manage transactions.
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 service supports additional resource types to manage transactions for applications, and can be accessed in the console.

   - **Groups** are collection of transaction resources. You can 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 AWS Blu Age refactor engine, you specify the lists to be included at the startup. There is no limit to number of lists.
     - With Rocket Software 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 Rocket Software replatform engine.

5. Choose a transaction resource to display all the resource information. You can also view all attributes associated with your transaction resource.

Configure the Rocket Software (formerly Micro Focus) managed application

You can configure your applications with Rocket Software runtime engine to customize additional properties including integrations.

Topics

- Supported third party integrations for Rocket Software
  - Printers

Supported third party integrations for Rocket Software

To make use of third party integrations, your AWS Mainframe Modernization managed environment must use a Rocket Software engine version that supports this type of configuration. Engine versions with the postfix R (e.g., version 9.0.9.R) are supported. That means, engine version 9.0.9.R includes client installation support for the party integrations, but 9.0.9 doesn't.

Printers

Printer resources are configured through the Rocket Software application definition as described in the the section called “Printers - optional” section.

A printer definition may define a custom or service-provided exit module for the printer. Some examples of possible exit module configurations are:

1. Example of loading service provided binary.

```
...
{
    "name": "p1",
    "classes": [
        "AB"
    ],
    "description": "Using service managed LRS Queue exit module",
    "exit-module": {
```

```
        "name": "lrsprte6"
    }
},
...
```

2. Example of providing binary from S3.

```
"exit-module": {
        "name": "s3Exit",
        "module": "${s3-source}/3pa/s3Exit.so"
}
```

3. Example of providing binary from EFS.

> **Note**
>
> To use EFS mount, it must be attached during the creation of the environment, along with some additional values to be set such as `program-path`.

```
...
"batch-settings": {
    "jes-printers": [
        {
            "name": "p3",
            "classes": [
                "EF"
            ],
            "description": "Using binary from customer provided exit module on EFS
 Mount",
            "exit-module": {
                "name": "efsExit"
            }
        }
    ],
    "program-path": "$EFS_MOUNT/path/to/directory/containing/binaries/"
},
"runtime-settings": {
    "environment-variables": {
        "EFS_MOUNT": "/m2/mount/efs"
    }
```

```
}
...
```

LRS queue - optional

To use LRS queue, you must be using a Rocket Software engine that supports third party artifacts (i.e. engines ending with `.R`). In addition to setting up a printer with an exit module pointing to `lrsprte6` as the exit module's entry name, LRS queue requires an additional environment variable, as defined in the pre-existing "runtime-settings" block of the Rocket Software application definition.

LRSQ_ADDRESS

(Required) Specifies the LRS server address for the LRSQ print exit module to send to.

LRS printers - Configuring an LRS Printer requires the definition of a jes printer as specified in the the section called "Printers - optional" section.

Additionally, the LRSQ_ADDRESS must be specified as part of the `runtime-settings` field in the application definition.

```
"runtime-settings": {
    "environment-variables": {
        "LRSQ_ADDRESS": "<lrsq-address>"
    }
}
```

Configure the AWS Blu Age managed application

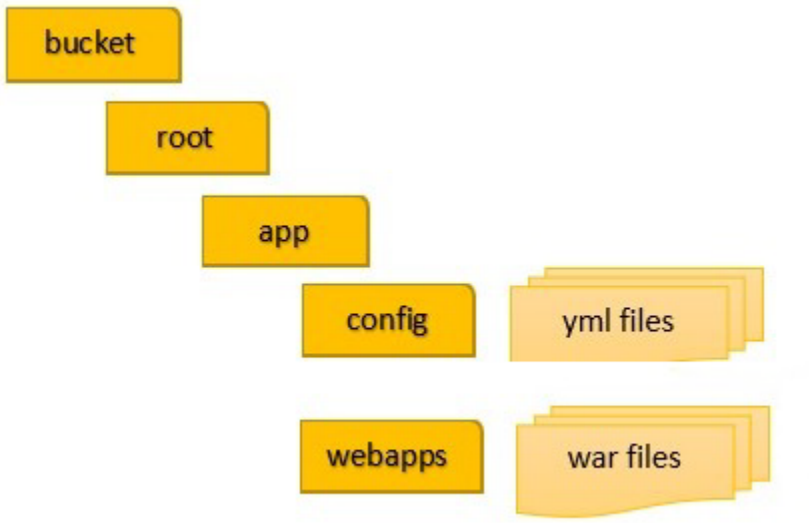
You can configure your application to include access to legacy utilities. You can customize additional properties as well. In order to understand what you can configure and where, refer the the section called "Structure of AWS Blu Age managed applications" section to understand the overall structure of an AWS Blu Age modernized application.

Topics

- Structure of AWS Blu Age managed applications
- Configure access to utilities for managed applications
- Add configuration properties for the managed application with AWS Blu Age engine

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.

extra

Contains jars that can provide additional capabilities for the modernized application.

Managing an application's Java options

To manage some java options for the application, add a properties file named `tomcat.properties` to the `application-name` folder. This file can have three properties: `xms`, which specifies the minimum Java memory consumption, `xmx`, which specifies the maximum Java memory consumption, and `dnscachettl`, that manages the cache duration for dns resolutions. The following is an example of the contents of a valid `tomcat.properties` file.

```
xms=512M
xmx=1G
dnscachettl=5
```

The values that you specify for the first 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.

The value for the third property represents the cache duration in seconds, and can have value of -1 (cache forever), or can range from 0 (never cache) to 999. In the context of managed application deployments, the default value is -1.

Configure 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

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
 spring.aws.client.datasources.primary.secret: `arn:aws:secretsmanager:us-
west-2:111122223333:secret:business-FfmXLG`

 treatLargeNumberAsInteger: false

 # Zoned mode : valid values = EBCDIC_STRICT, EBCDIC_MODIFIED, AS400
 zonedMode: EBCDIC_STRICT

 jcl.type: mvs

 # Unload properties
 # For date/time: if use database configuration is enabled, formats are ignored
 # For nbi; use hexadecimal syntaxe to specify the byte value
 unload:
  sqlCodePointShift: 384
```

```
  nbi:
    whenNull: "6F"
    whenNotNull: "00"
  useDatabaseConfiguration: false
  format:
    date: MM/dd/yyyy
    time: HH.mm.ss
    timestamp: yyyy-MM-dd-HH.mm.ss.SSSSSS
  chunkSize:500
  fetchSize: 500
  varCharIsNull: false
  columnFiller: space

# Load properties
# Batch size for DSNUTILB Load Task
load:
  sqlCodePointShift: 384
  batchSize: 500
  format:
    localDate: dd.MM.yyyy|dd/MM/yyyy|yyyy-MM-dd
    dbDate: yyyy-MM-dd
    localTime: 'HH:mm:ss|HH.mm.ss'
    dbTime: 'HH:mm:ss'

table-mappings:
  TABLE_1_NAME : LEGACY_TABLE_1_NAME
  TABLE_2_NAME : LEGACY_TABLE_2_NAME
```

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`.

```
sysPunchEncoding : cp1047
```

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:

Primary data source

This is the primary business application database.

spring.aws.client.datasources.primary.secret

(Optional) Specifies the secret in Secrets Manager that contains the data source properties.

```
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.

```
spring.aws.client.datasources.primary.dbname: target-database-name
```

spring.aws.client.datasources.primary.type

(Optional) Specifies the fully qualified name of the connection pool implementation to use. The default value is `com.zaxxer.hikari.HikariDataSource`.

```
spring.aws.client.datasources.primary.type: target-datasource-type
```

If the type of the primary data source is `com.zaxxer.hikari.HikariDataSource`, you can specify additional properties as follows:

spring.datasource.primary.[property_name]

(Optional) You can use this format to specify extra properties for configuring a primary data source connection pool implementation.

The following is an example for a primary data source of type `com.zaxxer.hikari.HikariDataSource`.

```
spring:
   datasource:
       primary:
         autoCommit: XXXX
         maximumPoolSize: XXXX
         keepaliveTime: XXXX
         minimumIdle: XXXX
         idleTimeout: XXXX
         connectionTimeout: XXXX
         maxLifetime: XXXX
```

Other utility data sources

In addition to the primary data source, you can provide other utility data sources.

spring.aws.client.utility.pgm.datasources.names

(Optional) Specifies the list of utility data source names.

```
spring.aws.client.utility.pgm.datasources.names: dsname1, dsname2, dsname3
```

spring.aws.client.utility.pgm.datasources.[dsname].secret

(Optional) Specifies the secret ARN in SSM that hosts the data source properties. Provide [dsname] in the list of names specified in `spring.aws.client.utility.pgm.datasources.names`.

```
spring.aws.client.utility.pgm.datasources.dsname1.secret: datasource-secret-ARN
```

spring.aws.client.utility.pgm.datasources.[dsname].dbname

(Optional) Specifies the target database name if the database name isn't provided directly in the database secret by using the dbname property. Provide [dsname] in the list of names specified in `spring.aws.client.utility.pgm.datasources.names`.

```
spring.aws.client.utility.pgm.datasources.dsname1.dbname: target-database-name
```

spring.aws.client.utility.pgm.datasources.[dsname].type

(Optional) Specifies the fully qualified name of the connection pool implementation to use. The default value is `com.zaxxer.hikari.HikariDataSource`. Provide [dsname] in the list of names specified in `spring.aws.client.utility.pgm.datasources.names`.

```
spring.aws.client.utility.pgm.datasources.dsname1.type: target-datasource-type
```

If the utility data source type is `com.zaxxer.hikari.HikariDataSource`, you can provide additional properties as follows:

spring.datasource.[dsname].[property_name]

(Optional) Specifies a collection of extra properties to configure a utility data source connection pool implementation. Provide [dsname] in the list of names specified in `spring.aws.client.utility.pgm.datasources.names`. Specify the properties in the following format: `property_name : value`

The following is an example for additional utility data sources of type `com.zaxxer.hikari.HikariDataSource`:

```
spring:
   datasource:
       dsname1:
         connectionTimeout: XXXX
         maxLifetime: XXXX
       dsname2:
         connectionTimeout: XXXX
         maxLifetime: XXXX
       dsname3:
         connectionTimeout: XXXX
         maxLifetime: XXXX
```

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`

```
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.

```
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.

```
# 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.

```
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.

  ```
  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.

```
# 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
```

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.

```
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.

```
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:

```
table-mappings:
 TABLE_1_NAME : LEGACY_TABLE_1_NAME
 TABLE_2_NAME : LEGACY_TABLE_2_NAME
 ...
 TABLE_*N*_NAME : LEGACY_TABLE_*N*_NAME
```

> **Note**
>
> When the utility application starts, it explicitly logs all provided mappings.

Add configuration properties for the managed application with 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.

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
- Gapwalk batchscript properties
- Gapwalk Blugen properties
- Gapwalk CL command properties
- Gapwalk CL runner properties
- Gapwalk JHDB properties
- Gapwalk JICS properties
- Gapwalk runtime properties
- Gapwalk utility program properties
- Other properties

Gapwalk application properties

bluesam.fileLoading.commitInterval

Optional. The BluSAM commit interval.

Type: number

Default: 100000

card.encoding

Optional. Card encoding: to be used with `useControlMVariable`.

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
```

blockSizeDefault

Optional. The default block size to be used for BDW bytes.

Type: number

Default: 32760

```
blockSizeDefault: 32760
```

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

jcl.checkpoint.expireTimeout

Optional. Specifies time duration to retain JCL checkpoints in the persistence provider or in-memory registry.

Type: number

Default: -1

jcl.checkpoint.expireTimeoutUnit

Optional. Specifies time duration unit for the `jcl.checkpoint.expireTimeout` property. Supported enum constant values: `java.util.concurrent.TimeUnit`.

Type: string

Default: SECONDS

Gapwalk batchscript properties

encoding

Optional. The encoding used in batchscript projects (not with groovy). Expects a valid encoding `CP1047`, `IBM930`, `ASCII`, `UTF-8`...

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 AWS Blu Age Runtime program designed for debugging purposes.

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

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/

> **Note**
>
> Currently, for deployment constraints, you must copy your dbds and psbs directories in the config directory of your application or in a subdirectory of the config directory: e.g., config/setup
>
> ```
> config
> |- setup
>    |- dbds
>    |- psbs
> ```
>
> and set in application-jhdb.yml

```
jhdb.metadata.extrapath: file: ./config/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

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

mq.queues.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 thead 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 thead 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

Default: true

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 code pointshift 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.

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

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 Rocket Software (formerly Micro Focus) application definitions and identifies all required and optional elements.

Contents

- General header section
- Definition section overview
- AWS Blu Age application definition sample
- AWS Blu Age definition details
  - Listener(s) - required
  - AWS Blu Age application - required
  - BluSAM - optional
  - AWS Blu Age message queues - optional
  - AWS Blu Age Application storage EFS config - optional
- Rocket Software (formerly Micro Focus) application definition
- Rocket Software definition details
  - Listener(s) - required
  - Data set locations - required
  - Amazon Cognito authentication and authorization handler - optional
  - LDAP and Active Directory handler - optional
  - Batch settings - required
  - CICS settings - required
  - Printers - optional
  - XA resources - optional
  - Runtime settings - optional

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.

Use the following structure to specify the template version and source locations.

```
"template-version": "2.0",
    "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket",
                "s3-key-prefix": "v1"
            }
        }
    ]
```

Note

You can use the following syntax if you want to input S3 ARN as s3-bucket :

```
"template-version": "2.0",
  "source-locations": [
      {
          "source-id": "s3-source",
          "source-type": "s3",
          "properties": {
              "s3-bucket": "arn:aws:s3:::mainframe-deployment-bucket",
              "s3-key-prefix": "v1"
          }
      }
  ]
```

template-version

(Required) Specifies the version of the application definition file. Do not change this value. Currently, the only allowed value is 2.0. Specify `template-version` with a string.

source-locations

Specifies the locations of the files and other resources that the application requires during runtime.

source-id

Specifies a name for the location. This name is used to reference the source location as needed in the application definition JSON.

source-type

Specifies the type of the source. Currently, the only allowed value is s3.

properties

Provides the details of the source location. Each property is specified with a string.

- `s3-bucket` - Required. Specifies the name of the Amazon S3 bucket where the files are stored.
- `s3-key-prefix` - Required. Specifies the name of the folder in the Amazon S3 bucket where the files are stored.

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

```
{
 "template-version": "2.0",
 "source-locations": [
        {
            "source-id": "s3-source",
            "source-type": "s3",
            "properties": {
                "s3-bucket": "mainframe-deployment-bucket-aaa",
```

```
                "s3-key-prefix": "v1"
            }
        }
    ],
    "definition" : {
        "listeners": [{
            "port": 8194,
            "type": "http"
        }],
        "ba-application": {
            "app-location": "${s3-source}/murachs-v6/"
        },
        "blusam": {
            "db": {
                "nb-threads": 8,
                "batch-size": 10000,
                "name": "blusam",
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:blusam-FfmXLG"
            },
            "redis": {
                "hostname": "blusam.c3geul.ng.0001.usw2.cache.amazonaws.com",
                "port": 6379,
                "useSsl": true,
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:bluesamredis-nioefm"
            }
        }
    }
}
```

AWS Blu Age 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:

```
"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.

```
"ba-application": {
            "app-location": "${s3-source}/murachs-v6/",
            "files-directory": "/m2/mount/myfolder",
            "enable-jics": <true|false>,
            "enable-batch-restart": <true|false>,
            "shared-app-location": "${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. The subfolder must be owned by a suitable user for use by the **Blu Age** application running inside AWS Mainframe Modernization. To achieve this, when attaching the drive to a Linux Amazon EC2 instance, a group with ID `101` and a user with ID `3001` must be created, and the desired folder must be owned by this user. *For example, this way, the `testclient` folder can be used by **Blu Age** AWS Mainframe Modernization Managed.*

```
groupadd -g 101 mygroup
useradd -M -g mygroup -p mypassword -u 3001 myuser
mkdir testclient
chown myuser:mygroup testclient
```

enable-jics

(Optional) Specifies whether to enable JICS. Defaults to true. Setting this to false prevents the JICS database from being spawned.

enable-batch-restart

(Optional) Specifies whether to enable restart feature for batch jobs. Defaults to false. For more information regarding batch restart configurations, see AWS Blu Age engine properties prefixed `jcl.checkpoint` in Configuration properties for the managed application with AWS Blu Age engine.

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 - optional

Specify the BluSAM database and Redis cache using the following structure.

```
"blusam": {
            "db": {
                "nb-threads": 8,
                "batch-size": 10000,
                "name": "blusam",
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:blusam-FfmXLG"
            },
            "redis": {
                "hostname": "blusam.c3geul.ng.0001.usw2.cache.amazonaws.com",
                "port": 6379,
                "useSsl": true,
                "secret-manager-arn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret:bluesamredis-nioefm"
            }
}
```

db

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 4: Create and configure an AWS Secrets Manager database secret.

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 4: Create and configure an AWS Secrets Manager database secret.

AWS Blu Age message queues - optional

Specify the JMS-MQ connection details for AWS Blu Age application.

```
"message-queues": [
  {
      "product-type": "JMS-MQ",
      "queue-manager": "QMgr1",
      "channel": "mqChannel1",
```

```
      "hostname": "mqserver-host1",
      "port": 1414,
      "user-id": "app-user1",
      "ssl-cipher": "*TLS12ORHIGHER",
      "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",
      "ssl-cipher": "*TLS12ORHIGHER",
      "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.

ssl-cipher

(Optional) Specifies the SSL cipher specification for the connection.

secret-manager-arn

(Optional) Specifies the Amazon Resource Name (ARN) of Secrets Manager that provides the password of the specified user.

AWS Blu Age Application storage EFS config - optional

Specify the application storage EFS Access point details using the following structure.

```
"ba-application": {
            "file-permission-mask": "UMASK002"
},
"efs-configs": [
           {
            "file-system-id": "fs-01376dfsvfvrsvsr",
            "mount-point": "/m2/mount/efs-ap2",
            "access-point-id": fsap-0eaesefvrefrewgv8"
           }
]
```

file-system-id

(Required) The ID of EFS file system that the access point applies to. Pattern: "fs-([0-9a-f]{8,40}){1,128}$"

mount-point

(Required) The mount point for the application level file system. This must be different than the environment level storage mount point.

access-point-id

(Required) The ID of the access point, assigned by Amazon EFS. Pattern: "^fsap-([0-9a-f]{8,40}){1,128}$"

file-permission-mask

(Optional) Defines the file creation mask for files created by the application process. For example, when the value is set to UMASK006, all the files will have permission 660. This will

mean that only the file owner and file group will have the read and write access, while other users don't have any permissions.

> **Note**
>
> The value set for this field is only considered when using application level EFS storage.

> **Note**
>
> When efs config is provided, files-directory must be specified in the application definition section. It must be a subfolder of the Amazon EFS mount point set up at application level.

Rocket Software (formerly Micro Focus) application definition

The following sample definition section is for the Rocket Software 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",
                "s3-key-prefix": "v1"
            }
        }
    ],
    "definition" : {
        "listeners": [{
            "port": 5101,
            "type": "tn3270"
        }],
        "dataset-location": {
            "db-locations": [{
                "name": "Database1",
                "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456"
            }]
```

```
        },
        "cognito-auth-handler": {
            "user-pool-id": "cognito-idp.us-west-2.amazonaws.com/us-west-2_rvYFnQIxL",
            "client-id": "58k05jb8grukjjsudm5hhn1v87",
            "identity-pool-id": "us-west-2:64464b12-0bfb-4dea-ab35-5c22c6c245f6"
        },
        "ldap-ad-auth-handler": {
            "ldap-ad-connection-secrets": [LIST OF AD-SECRETS]
        },
        "batch-settings": {
            "initiators": [{
                "classes": ["A", "B"],
                "description": "initiator...."
            }],
            "jcl-file-location": "${s3-source}/batch/jcl",
            "program-path": "/m2/mount/libs/loadlib:$EFS_MOUNT/emergency/loadlib",
            "system-procedure-libraries":"SYS1.PROCLIB;SYS2.PROCLIB",
            "aliases": [
               {"alias": "FDSSORT", "program": "SORT"},
               {"alias": "MFADRDSU", "program": "ADRDSSU"}
            ]
        },
        "cics-settings": {
            "binary-file-location": "${s3-source}/cics/binaries",
            "csd-file-location": "${s3-source}/cics/def",
            "system-initialization-table": "BNKCICV"
        },
        "jes-printers": [
           {
             "name": "printerName",
              "classes": [
                 "A",
                 "B"
             ],
             "description": "printer desc....",
             "exit-module": {
                 "name": "lrsprte6"
                }
             }
         ],
        "xa-resources" : [{
            "name": "XASQL",
            "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456",
            "xa-connection-type": "postgres",
```

```
            "module": "${s3-source}/xa/ESPGSQLXA64.so"
        }],
        "runtime-settings": {
          "base-configuration-location": "${s3-source}/exported.json",
          "environment-variables": {
            "ES_JES_RESTART": "N",
            "EFS_MOUNT": "/m2/mount/efs",
            "LRSQ_ADDRESS": "<lrsq-address>"
          }
        }
    }
}
```

Rocket Software definition details

The content in the definition section of the Rocket Software 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_rvYFnQIxL",
            "client-id": "58k05jb8grukjjsudm5hhn1v87",
            "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.

LDAP and Active Directory handler - optional

You can integrate your application with Active Directory (AD) or any type of LDAP server to make it possible for users of the application to use their LDAP/AD credentials for authorization and authentication.

To integrate your application with AD

1. Follow the steps described in Configuring Active Directory for Enterprise Server Security in the Micro Focus Enterprise Server documentation.
2. Create an AWS Secrets Manager secret with your AD/LDAP details for each AD/LDAP server that you want to use with your application. For information on how to create a secret, see Create an AWS Secrets Manager secret in the AWS Secrets Manager User Guide. For secret type, choose **Other type of secret** and include the following key-value pairs.

```
{
    "connectionPath"       : "<HOST-ADDRESS>:<PORT>",
    "authorizedId"         : "<USER-FULL-DN>",
    "password"             : "<PASSWORD>",
    "baseDn"               : "<BASE-FULL-DN>",
    "userClassDn"          : "<USER-TYPE>",
    "userContainerDn"      : "<USER-CONTAINER-DN>",
    "groupContainerDn"     : "<GROUP-CONTAINER-DN>",
    "resourceContainerDn"  : "<RESOURCE-CONTAINER-DN>"
}
```

Security recommendations

- For `connectionPath`, AWS Mainframe Modernization supports the LDAP and LDAP over SSL (LDAPS) protocols. We recommend using LDAPS because it is more secure and prevents credentials from appearing in network transmissions.

- For `authorizedId` and `password`, we recommend that you specify the credentials of a user with no more permissions than the most restrictive read-only and verification permissions that are required for your application to run.
- We recommend rotating the AD/LDAP credentials on a regular basis.
- Do not create AD users with the username `awsuser` or `mfuser`. These two usernames are reserved for AWS use.

The following is an example.

```
{
    "connectionPath" : "ldaps://msad4.m2.example.people.aws.dev:636",
    "authorizedId" :
 "CN=LDAPUser,OU=Users,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "password" : "ADPassword",
    "userContainerDn" : "CN=Enterprise Server Users,CN=Micro Focus,CN=Program
 Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "groupContainerDn" : "CN=Enterprise Server Groups,CN=Micro Focus,CN=Program
 Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev",
    "resourceContainerDn" : "CN=Enterprise Server Resources,CN=Micro
 Focus,CN=Program Data,OU=msad4,DC=msad4,DC=m2,DC=example,DC=people,DC=aws,DC=dev"
}
```

Create the secret with a customer-managed KMS key. You must grant AWS Mainframe Modernization the `GetSecretValue` and `DescribeSecret` permissions on the secret, and `Decrypt` and `DescribeKey` permissions on the KMS key. For more information, see Permissions for the KMS key in the AWS Secrets Manager User Guide.

3. Add the following to your application definition.

```
"ldap-ad-auth-handler": {
    "ldap-ad-connection-secrets": [LIST OF AD/LDAP SECRETS]
}
```

The following is an example.

```
"ldap-ad-auth-handler": {
    "ldap-ad-connection-secrets": ["arn:aws:secrets:1234:us-east-1:secret:123456"]
}
```

If the application is integrated with LDAP and has been started, you must provide credentials to execute at least one application related operations mentioned in the list of supported authorizations.

The LDAP/AD authentication handler is available for Micro Focus (Rocket) 8.0.11 and later versions.

> **Note**
>
> Currently, the LDAP administrator has to provide "alter" permissions on the `casstart` utility in the "OPERCMDS" enterprise server resources in their LDAP directory. This needs to be done for all the required default users (e.g., CICSUSER - if the application is CICS related) to start the application successfully.

To provide LDAP user credentials for authentication and authorization

1. Create an AWS Secrets Manager with the following keys and values:

```
{
  "username" : "<USERNAME>",
  "password" : "<PASSWORD>"
}
```

> **Important**
>
> You should have the rights to execute `DescribeSecrets` and `GetSecretValue` on the Secrets Manager being used. Also, associate a KMS key and necessary permissions for the AWS Secrets Manager, as mentioned in Choosing a AWS KMS key.

2. Choose the Secrets Manager parameter.

   AWS console

   While executing operations from AWS console, there will be an option to choose the Secrets Manager that needs to be passed.

   AWS CLI (or SDK)

   While executing operations from AWS CLI (or SDK) the API parameter `auth-secrets-manager-arn` should be passed with the Secrets Manager ARN.

Here is the list of application operations that currently support authorization:

- `StartBatchJob`
- `CancelBatchJobExecution`
- `ListBatchJobRestartPoints`

Batch settings - required

Specify the details required by the batch jobs that run as part of the application using the following structure.

```
"batch-settings": {
            "initiators": [{
                "classes": ["A", "B"],
                "description": "initiator...."
            }],
            "jcl-file-location": "${s3-source}/batch/jcl",
            "program-path": "/m2/mount/libs/loadlib:$EFS_MOUNT/emergency/loadlib",
            "system-procedure-libraries":"SYS1.PROCLIB;SYS2.PROCLIB",
            "aliases": [
               {"alias": "FDSSORT", "program": "SORT"},
               {"alias": "MFADRDSU", "program": "ADRDSSU"}
    ]
}
```

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:

```
"batch-settings": {
            "initiators": [
                {
                  "classes": ["A", "B"],
                  "description": "initiator...."
                },
                {
                  "classes": ["C", "D"],
                  "description": "initiator...."
                }
```

```
        ],
}
```

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 (Job Control Language) files that are required by the batch jobs the migrated application runs.

program-path

Specifies the path required to run batch jobs when a program in a JCL is not in the default location. The different path names are separated with a colon (:).

> **Note**
>
> The program path can only be an EFS path.

system-procedure-libraries

Specifies the default partitioned data sets that will be searched for JCL procedures. The procedure is, however, not found in the JCL or via the JCLLIB statements. These data sets must be cataloged and the catalog name must be used. And the entries are separated with a semi-colon (;).

aliases

Defines a mapping for the utility and program names used in JCL to the implementation name of the utility. AWS and 3rd party batch utilities (e.g. M2SFTP, M2WAIT, Syncsort, etc.) can optionally have aliases to eliminate the need to change the JCL. For example:

- FDSSORT Alias FDSSORT for SORT and Alias FDSICET for ICETOOL
- ADRDSSU Alias MFADRDSU for ADRDSSU
- Syncsort Alias DMXMFSRT for SORT

CICS settings - required

Specify the details required for the CICS transactions that run as part of the application using the following structure.

```
"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.

Printers - optional

Specify a jes printer using the following structure.

```
"jes-printers": [
    {
        "name": "printerName",
        "classes": [
            "A",
            "B"
        ],
        "description": "printer desc....",
        "exit-module": {
            "name": "lrsprte6",
            "module" : "program"
        }
```

```
    }
],
```

> **Note**
>
> There can be a maximum of 25 printers configured for a given application.

name

(Required) Specifies the name to associate with this printer resource. Names must be unique for each printer, and a limit of 128 alphanumeric characters can be used.

classes

(Required) Specifies the output classes applicable to this printer resource. A limit of 36 alphanumeric characters can be used.

description

(Optional) Additional descriptive text for the printer.

exit-module

(Optional) Specifies a custom module for print exit. There are no default values, if not specified, no exit module will be used. You can use either a managed print exit module or supply your own. Managed print exit modules are defined using reserved name `lrsprte6` for LRS queue or supply your own using the module parameter to specify the location and name.

The structure of `exit-module` has two components:

- `name`- (Required), if `exit-module` is used. The name of the entry of the exit module. There is a limit on the exit module entry name of up to 8 characters.
- `module` - (Optional) The S3 location of the print exit module binary.

You can see more examples of defining the exit module in the the section called “Printers” section.

XA resources - optional

Specify the details required for the XA resources that the application requires using the following structure.

```
"xa-resources" : [{
            "name": "XASQL",
            "secret-manager-arn": "arn:aws:secrets:1234:us-east-1:secret:123456",
            "xa-connection-type": "postgres",
            "module": "${s3-source}/xa/ESPGSQLXA64.so"
}]
```

Note

XA resource definition has been updated to include an optional `xa-connection-type` field. If not provided, the connection type is assumed to be “postgres”.

name

(Required) Specifies the name of the XA resource.

secret-manager-arn

(Required) Specifies the Amazon Resource Name (ARN) for the secret that contains the credentials for connecting to the database.

xa-connection-type

(Optional) Specifies the XA resource connection type.

module

(Required) 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.

Runtime settings - optional

Specify the details required for runtime settings to manage permitted environment variables using the following structure.

```
"runtime-settings": {
          "base-configuration-location": "${s3-source}/exported.json",
          "environment-variables": {
            "ES_JES_RESTART": "N",
            "EFS_MOUNT": "/m2/mount/efs"
```

```
    }
}
```

base-configuration-location

(Optional) Specifies the location for bulk import for a Micro Focus server configuration. This file should be a valid JSON and present in the same S3 location as the application artifacts location defined above. To export configuration from an existing application, see To export a region from ESCWA *section* in the *Rocket software documentation*.

environment-variables

Specifies Micro Focus supported environment variables that are applied to this application's runtime.

- `ES_JES_RESTART` is a Rocket Software environment variable that enables JCL restart processing. Optionally, you can also use `ES_ALLOC_OVERRIDE` as a Rocket Software environment variable.
- `EFS_MOUNT` is a custom environment variable that your application might use to identify where the environment's EFS mount is located.

You can access all the Rocket Software environment variables in the *Rocket Enterprise Server for UNIX guide.*

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. Data sets must be in a format that can be loaded to the selected runtime engine. In many cases this can be a sequential file but for Rocket Software(formerly Micro Focus) VSAM it will need to be in their proprietary format. The `DFCONV` utility is the suggested

method of converting the file. Specify the name of the bucket and folder in the data set definition JSON file.
For more information on Rocket Software runtime engine, see DFCONV Batch File Conversion in the *Rocket Software* documentation.
For more information on AWS Blu Age, see the section called "AWS Blu Age Runtime configuration".

Topics

- Common properties
- Sample data set request format for VSAM
- Sample data set request format for GDG base
- Sample data set request format for PS or GDG generations
- Sample data set request format for PO

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.

```
{
    "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.
- Rocket Software 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.
- Rocket Software 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.
- Rocket Software 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
    }
},
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/catalog/data/AWS.M2.VSAM.KSDS.DAT"
}
```

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) and KSDS (KS). Record format can be fixed or variable.
- Rocket Software 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

(Required) 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, and KSDS, `min` is optional and `max` is required.
- Rocket Software 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.

```
{
    common properties
        ...
        "datasetOrg": {
```

```
            "vsam": {
                common properties
                ...
                "compressed": boolean,
                common properties
                ...
            }
        }
}
```

Specify the compression property as follows:

compression

(Optional) Specifies whether indexes for this data set are stored as compressed values. If you have a large data set (typically > 100 Mb), consider setting this flag to `true`.

Sample data set request format for GDG base

- AWS Blu Age runtime engine: not supported.
- Rocket Software runtime engine: supported.

If you are importing GDG base data sets, specify gdg as the `datasetOrg`. Your JSON should resemble the following example:

```
{
    "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.
- Rocket Software 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:

```
{
    "storageType": "Database",
    "datasetName": "AWS.M2.PS.FB",
    "relativePath": "DATA",
    "datasetOrg": {
        "ps": {
            "format": "FB",
            "encoding": "A"
        }
    },
    "recordLength": {
        "min": 300,
        "max": 300
    }
},
"externalLocation": {
    "s3Location": "s3://$M2_DATA_STORE/catalog/data/AWS.M2.PS.LSEQ"
}
}
```

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 Rocket Software 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?
- Set up for AWS Mainframe Modernization
- Get started with AWS Mainframe Modernization
- Tutorial: Set up managed runtime for AWS Blu Age
- Tutorial: Set up managed runtime for Rocket Software (formerly Micro Focus)

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 Rocket Software (formerly 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
- Update an AWS Mainframe Modernization runtime environment
- Stop an AWS Mainframe Modernization runtime environment
- Restart an AWS Mainframe Modernization runtime environment
- Delete an AWS Mainframe Modernization runtime environment

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 Set up for AWS Mainframe Modernization.

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 (Rocket)** 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, 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 the network type. If you choose IPv4, AWS Mainframe Modernization environment applications serve only IPv4 requests. In the dual-stack mode, applications will serve both IPv4 and IPv6 requests. If you choose the dual-stack mode, make sure there is at least 1 VPC with IPv6-enabled subnets.

   c. Choose a Virtual Private Cloud (VPC).

   d. 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 Rocket Software engine, you can specify one subnet.

   e. 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 and outbound rules allow access from an IP address to the port you specified in the `the section called "Listener(s) - required"` property of the application definition. For more information, see Update the security groups for your Network Load Balancer in the *User Guide for Network Load Balancers*.

   f. 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.

   > **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.

g. (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.

h. Choose **Next**.

7. (Optional) On the **Attach storage** page, choose one or more Amazon EFS or Amazon FSx file systems.

   The file system mounted to an AWS Mainframe Modernization environment must be owned by a suitable user to be used by your applications that are running in the AWS Mainframe Modernization console.

   To configure these user settings, you can attach the drive to a Linux Amazon EC2 instance. Then create a group with ID `101` and a user with ID `3001`. Also, ensure the desired data folder that will be used by your application(s) must be owned by this user.

   For example, the `myFiles` folder can be used by your AWS Mainframe Modernization applications running in AWS Mainframe Modernization Managed.

   ```
   groupadd -g 101 mygroup
   useradd -M -g mygroup -p mypassword -u 3001 myuser
   mkdir myFiles
   chown myuser:mygroup myFiles
   ```

   Note

   To enable access to the file system, the following security groups rules should be configured for establishing network connectivity between the EFS and M2 environment instance:

   - **M2 environment security group** – Include an outbound rule that allows traffic over the NFS 2049 port.
   - **File system mount targets security group** – Include an inbound rule that allows traffic over the NFS 2049 port from the instance security group (listed above), and an outbound rule that allows traffic over the NFS 2049 port.

8. Choose **Next**.

9. 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.
   - 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.

   Choose **Next**.
10. 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 Set up for AWS Mainframe Modernization.

Update a runtime environment

To update 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 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 two-hour maintenance window. Any system changes are applied during this time. The maintenance window is your chance to control when modifications, and software and security patching occurs. If a maintenance event is scheduled for a given week, it begins during that two-hour maintenance window. Most maintenance events also complete during the two-hour maintenance window, although larger maintenance events might take more than a couple of hours to complete.

The two-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 2 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.

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 Set up for AWS Mainframe Modernization.

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.

> **Note**
>
> You must stop all applications before stopping environment.

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**
   >
   > To stop an environment, you can only choose to stop immediately.

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.

Restart a runtime environment

To restart an AWS Mainframe Modernization runtime environment, you follow similar steps as the stop environment section.

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**.

> **Note**
>
> The desired capacity for standalone environment can only be updated to 1. To restart a runtime environment, you can only choose to restart immediately.

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 Set up for AWS Mainframe Modernization.

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 Testing in AWS Mainframe Modernization

AWS Mainframe Modernization Application Testing provides automated functional equivalence testing for your migration projects. AWS Mainframe Modernization Application Testing accelerates migration projects by leveraging the elasticity of the cloud. You can run independent test suites on as many parallel environments as required, reducing testing timelines. Key benefits of Application Testing include testing acceleration and agility, high degrees of testing repeatability, built-in scalability and elasticity, large-scale automation, cost efficiency, and seamless integration with AWS CloudFormation for creating target test environments.

Topics

- What is AWS Mainframe Modernization Application Testing?
- AWS Mainframe Modernization Application Testing concepts
- AWS Mainframe Modernization Application Testing prerequisites
- Application Testing console workflows
- Tutorial: Set up the CardDemo sample application in AWS Mainframe Modernization Application Testing
- Tutorial: Replay and compare in AWS Mainframe Modernization Application Testing using CardDemo for AWS Blu Age deployed on Amazon EC2
- AWS Mainframe Modernization Application Testing supported data sets code pages
- Data protection in AWS Mainframe Modernization Application Testing
- How AWS Mainframe Modernization Application Testing works with IAM

What is AWS Mainframe Modernization Application Testing?

Testing impacts modernization projects significantly. AWS Application Testing, a feature of AWS Mainframe Modernization, provides automated functional equivalence testing for your migrated applications. Functional equivalence testing helps you validate that your applications on the AWS Cloud are equivalent to your applications on your mainframe. AWS Application Testing automatically compares changes to data sets, database records, and online 3270 screens between your mainframe and AWS. Moreover, Application Testing permits repeatable testing, so you can run your test scenarios many times as you update target architecture, resolve issues, and progress toward a fully migrated application. After migration, you can continue to use Application Testing for regression testing, to make sure that updates to runtime engines or other components don't

cause regressions. Application Testing is cost-efficient: target test environments are created using the user-provided CloudFormation templates, leveraging Infrastructure-as-Code (IaC) concepts. Application Testing accelerates migration projects using the elasticity of the cloud. You can run independent test suites on as many parallel environments as required, reducing testing timelines.

Topics

- Are you a first-time Application Testing user?
- Benefits of Application Testing
- Integration with AWS CloudFormation
- How Application Testing works
- Related services
- Accessing Application Testing
- Pricing for Application Testing

Are you a first-time Application Testing user?

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

- Application Testing concepts
- Tutorial: Set up CardDemo application in Application Testing
- the section called “Tutorial: Replay and compare on AWS Blu Age using CardDemo”

Benefits of Application Testing

Application Testing provides several benefits to help you in your migration process:

- Testing acceleration, agility, and flexibility.
- “Record once on mainframe, replay multiple times in AWS” testing concepts.
- IaC creation of target environments through user-supplied CloudFormation templates.
- High degrees of testing repeatability.
- Built for the cloud, with scalability and elasticity in mind.
- Large-scale testing with high degree of automation.
- Cost efficiency.

Integration with AWS CloudFormation

Application Testing uses infrastructure as code with AWS CloudFormation. This design choice simplifies and improves your testing experience. AWS CloudFormation gives you autonomy and the independence to define the better infrastructure for your needs. You can select or define for many parameters (instance size, RDS instance, optimal security group) independently. You can add resources, such as an Amazon SQS queue that you require for your application to work properly under test conditions.

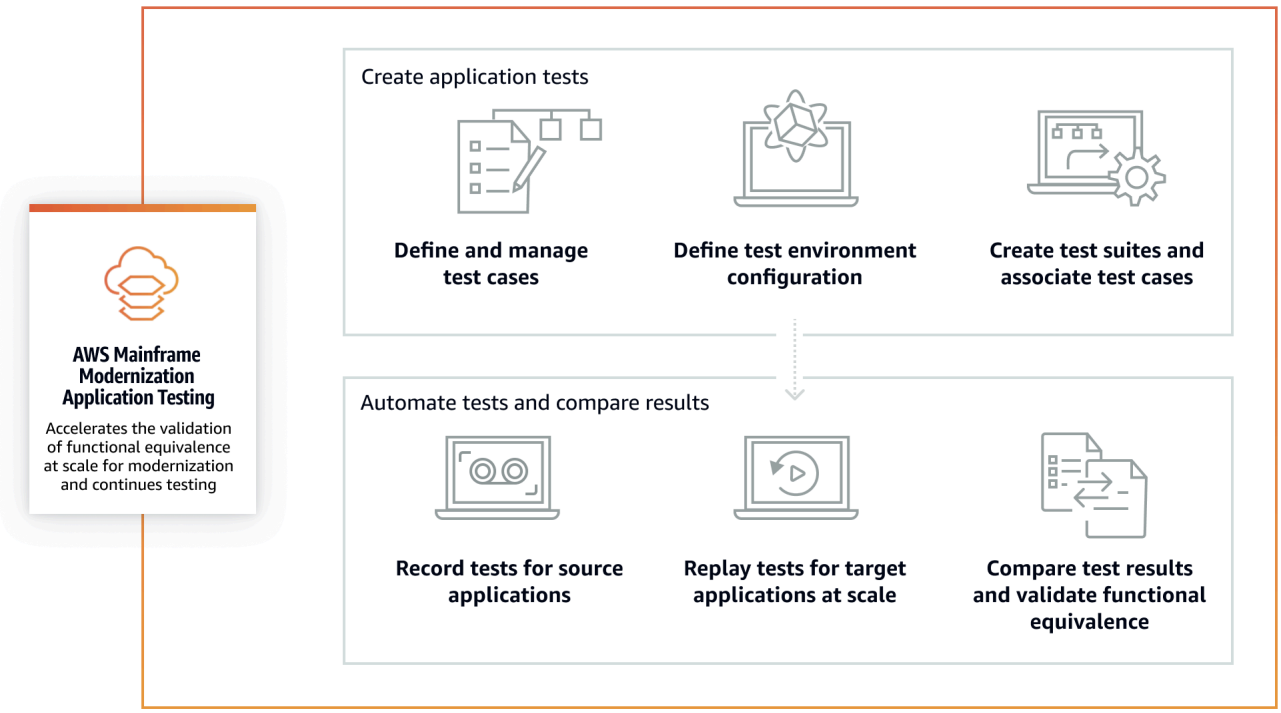
In the AWS CloudFormation templates provided for download, you will notice some common features:

- Application Testing creates a fully isolated stack, including an AWS Mainframe Modernization runtime environment and application, with its own network and security definitions. This isolated stack provides resiliency, because other actors in the same AWS account cannot interfere with testing activity. It also avoids situations where system operators modify the default VPC or security group, which can cause testing activity failures.
- The security group also allows you to control external access to the resources used in testing. For example, a database might contain confidential data.
- Full isolation prevents other actors that share the VPC from snooping on the traffic.
- It enhances performance. For example, communication between the AWS Mainframe Modernization application that the template creates and its Amazon RDS database occurs on a separate network (a private VPC), which avoids other actors slowing down traffic.

We recommend that you implement these features in the AWS CloudFormation templates you create as well.

How Application Testing works

The following figure is an overview of how Application Testing works.



- You can transfer input data from the source to AWS using *File Transfer* or your preferred tools for mainframe data transfer.
- You run the same business logic on both the source and the target.
- Application Testing automatically compares the output data (data sets, relational database changes, online 3270 screens and user interactions) from both source and target. After you run your test scenario on the mainframe, you capture the output data and transfer them to AWS, then replay the test scenario on the target. Application Testing automatically compares the output data from the test run on AWS with the output data from the source. You can see at a glance which records are identical, equivalent, different, or missing. In addition, you can define equivalence rules, so that records that are not identical but have the same business meaning are understood to be equivalent.

The workflow you follow in Application Testing consists of the following steps:

1. **Create test cases**: Test cases are the smallest unit of testing actions. When you create a test case, you also identify the data types to be compared that best represent functional equivalence between the source and target.
2. **Define test environment configuration**: Specify your environment configuration by specifying AWS CloudFormation template and additional attributes.

3. **Create test suites**: Test suites are a collection of test cases.
4. **Upload data sets on the source and replay on the target**: Capture the input and output data sets on the mainframe, and upload them to AWS. Then replay the test scenario on AWS.
5. **Compare source and target data sets**: Application Testing automatically compares the output data sets from both source and target, so you can see at a glance what is correct and what is not.

Both the final action of a test scenario and the goal of the entire process is to identify discrepancies between the source and the target test runs. Application Testing compares the source version and the target version for the data captured on all the interaction channels during the test run. It also compares the final states of the relevant data (as defined in the test cases).

Related services

Application Testing is a feature of AWS Mainframe Modernization. It also uses infrastructure as code with AWS CloudFormation to ensure testing repeatability, automation, and cost efficiency. For more information, see:

- AWS Mainframe Modernization
- AWS CloudFormation

Accessing Application Testing

You can access Application Testing console at https://console.aws.amazon.com/apptest/ or from the AWS Mainframe Modernization console by choosing **Application Testing** in the left navigation pane.

Pricing for Application Testing

Pricing for Application Testing can be found at AWS Mainframe Modernization Pricing.

AWS Mainframe Modernization Application Testing concepts

AWS Application Testing uses terms that other testing services or software packages might use with a slightly different meaning. The following sections explain how AWS Mainframe Modernization Application Testing uses this terminology.

Topics

- Test case
- Test suite
- Test environment configuration
- Upload
- Replay
- Compare
- Database comparisons
- Dataset comparisons
- Comparison status
- Equivalence rules
- Final-state data set comparison
- State-progress database comparisons
- Functional equivalence (FE)
- Online 3270 screen comparisons
- Replay data
- Reference data
- Upload, Replay, and Compare
- Differences
- Equivalencies
- Source application
- Target application

Test case

A test case is the individual most atomic unit of action in your testing workflow. Usually, a test case is used to represent an independent unit of business logic that modifies data. Comparisons will be done for each test case. Test cases are added to a test suite. Test cases contain metadata about the data artifacts (datasets, databases) which the test case modifies and about the business functions that are triggered during the test case execution: batch jobs, 3270 interactive dialogs, and others. For example, the names and code pages of datasets.

Input data → Test case → Output data

Test cases can be either online or batch type:

- **Online 3270 screen test cases** are test cases where user executes interactive screen dialogs (3270) to read, modify, or produce new business data (database and / or datasets records).
- **Batch test cases** are test cases requiring to submit a batch to read, process, and modify or produce new business data (datasets and / or database records).

Test suite

Test suites have a collection of test cases that are run in a sequential order, one by one. Replay is done at a test suite level. All test cases in the test suite are run on the target testing environment when a test suite is replayed. If there are differences after comparing reference and replay testing artifacts, the differences will be shown at the test case level.

For example, Test Suite A:

Test Case 1, Test Case 2, Test Case 3, and so forth.

Test environment configuration

Test environment configuration allows you to set up the initial set of data and configuration parameters (or resources) with CloudFormation that you need to make the test run repeatable.

Upload

Uploads are done at a test suite level. During upload, you must provide an Amazon S3 location that contains the artifacts, data sets, and relational database CDC journals from the source mainframe to be compared against. These will be considered as reference data from the source mainframe. During replay, the generated replay data will be compared against the uploaded reference data to ensure application equivalency.

Replay

Replays are done at a test suite level. During replay, AWS Mainframe Modernization Application Testing uses the CloudFormation script to create the target test environment and run the application. Data sets and database records that are modified during replay are captured and

compared against the reference data from the mainframe. Typically, you will upload on the mainframe once and then replay multiple times, until functional equivalency has been reached.

Compare

Comparisons are made automatically after a replay finishes successfully. During comparisons, the referenced data you uploaded and captured during the upload phase is compared against the replay data generated during the replay phase. Comparisons happen at an individual test case level for data sets, database records, and online screens separately.

Database comparisons

Application Testing employs a state-progress matching functionality when comparing changes in database records between the source and target applications. State-progress matching compares differences in each individual run INSERT, UPDATE, and DELETE statement, unlike comparing table rows at the end of the process. State-progress matching is more efficient than alternatives, providing faster and more accurate comparisons by only comparing changed data and detecting self-correcting errors in the transaction flow. By using CDC (Changed Data Capture) technology, Application Testing can detect individual relation database changes and compare them between the source and target.

Relation database changes are generated on source and target by the tested application code using DML (Data Modification Language) statements like **SQL INSERT**, **UPDATE**, or **DELETE**, but also indirectly when the application is using stored procedures, or when database triggers are set on some tables, or when **CASCADE DELETE** are used to guarantee referential integrity, triggering automatically additional deletions.

Dataset comparisons

Application Testing automatically compares the reference and replay data sets produced on the source (recording) and target replay) systems.

To compare data sets:

1. Start with the same input data (data sets, database) on both the source and the target.
2. Run your test cases on the source system (mainframe).
3. Capture the produced data sets and upload them to an Amazon S3 bucket. You can transfer input data sets from the source to AWS using CDC journals, screens,and data sets.

4. Specify the location of the Amazon S3 bucket where the mainframe data sets were uploaded when you uploaded the test case.

After replay is complete, Application Testing automatically compares the output reference and target data sets, showing if records are identical, equivalent, different, or missing. For example, date fields that are relative to the moment of workload execution (day + 1, end of current month, etc.) are automatically considered as equivalent. In addition, you can optionally define equivalence rules, so that records that are not identical still have the same business meaning, and are flagged as equivalent.

Comparison status

Application Testing uses the following comparison statuses: IDENTICAL, EQUIVALENT, and DIFFERENT.

IDENTICAL

The source and target data are exactly the same.

EQUIVALENT

The source and target data contain false differences considered as equivalences, such as dates or timestamps that do not affect functional equivalence when they are relative to the moment of workload execution. You can define equivalence rules to identify what these differences are. When all replayed test suites compared to their reference test suites show the status of IDENTICAL or EQUIVALENT, your test suite shows no differences.

DIFFERENT

The source and target data contains differences, such as a different number of records in a dataset, or different values in the same record.

Equivalence rules

A set of rules to identify false differences that can be considered equivalent results. Offline functional equivalence testing (OFET) inevitably causes differences for some results between the source and target systems. For example, update timestamps are different by design. Equivalence rules explain how to adjust for those differences and avoid false positives at comparison time. For example, if a date is runtime + 2 days in a particular data column, the equivalence rule describes it

and accepts a time on the target system that is runtime on target + 2 days instead of a value that strictly equals the same column in the reference uploading.

Final-state data set comparison

The end state of data sets that have been created or modified, including all changes or updates made to the data sets from their initial state. For data sets, Application Testing looks at the records in those data sets at the end of a test case run, and compare the results.

State-progress database comparisons

Comparisons of changes done to database records as a sequence of individual DML (Delete, Update, Insert) statements. Application Testing compares individual changes (insert, update, or delete a table's row) from the source database to the target database, and will identify differences for each individual change. For example, an individual INSERT statement may be used to insert in a table a row with different values on the source database compared to the target database.

Functional equivalence (FE)

Two systems are considered functionally equivalent if they produce the same results on all observable operations, given the same input data. For example, two applications are considered functionally equivalent if the same input data produces identical output data (through screens, dataset changes or database changes).

Online 3270 screen comparisons

Compares the output of the mainframe 3270 screens with the output of the modernized application web screens when the target system is running under AWS Blu Age runtime in the AWS Cloud. And it compares the output of the mainframe 3270 screens with the 3270 screens of the rehosted application when the target system is running under Rocket Software (formerly Micro Focus) runtime in the AWS Cloud.

Replay data

Replay data is used to describe the data generated by replaying a test suite on the target test environment. For example, replay data is generated when a test suite is running on an AWS Mainframe Modernization service application. Replay data is then compared to the reference data captured from the source. Every time you replay the workload in the target environment, a new generation of replay data is generated.

Reference data

Reference data is used to describe the data captured on the source mainframe. It is the reference to which replay (target) generated data will be compared. Usually, for every record on the mainframe that creates reference data, there will be many replays. This is because users typically capture the correct state of the application on the mainframe, and replay the test cases on the target modernized application to validate equivalency. If bugs are found, they are fixed and the test cases are replayed again. Often, multiple cycles of replay, fixing bugs, and replaying again to validate the occurrence. This is called the **capture once, replay multiple times** paradigm of testing.

Upload, Replay, and Compare

Application Testing operates in three steps:

- **Upload**: captures the referenced data created on the mainframe for each test case of a test scenario. These can include 3270 online screens, data sets, and database records.
  - For online 3270 screens, you must use the Blu Insights terminal emulator to capture your source workload. For more information see, Blu Insights documentation.
  - For data sets, you will need to capture the data sets produced by each test case on the mainframe by using common tools, like FTP or the dataset transfer service part of AWS Mainframe Modernization.
  - For database changes, you use theAWS Mainframe Modernization Data Replication with Precisely documentation to capture and generate CDC journals containing changes.
- **Replay**: The test suite is replayed in the target environment. All test cases specified in the test suite run. Specified data types created by the individual test cases, such as data sets, relational database changes, or 3270 screens, will be captured with automation. These data are known as replay data, and will be compared against the reference data captured during the upload phase.

  > **Note**
  >
  > The relational database changes will require DMS-specific configuration options in your initial condition CloudFormation template.

- **Compare**: the source testing reference data, and the target replay data are compared, and the results will be displayed to you as identical, different, equivalent, or missing data.

Differences

Indicates differences have been detected between the reference and replay data sets by data comparison. For example, a field in an online 3270 screen that is showing different values from a business logic standpoint between the source mainframe and the target modernized application will be considered as a difference. Another example is a upload in a data set that is not identical between source and target applications.

Equivalencies

Equivalent records are records that are different between the reference and replay data sets, but should not be treated as different from a business logic standpoint. For example, a record containing the timestamp of when the dataset was produced (workload execution time). Using customizable equivalency rules, you can instruct Application Testing to treat such false positive difference as an equivalence, even if it shows different values between reference and replay data.

Source application

The source mainframe application to be compared against.

Target application

The new or modified application on which testing is done and which will be compared to the source application to detect any defects and to achieve functional equivalence between source and target applications. The target application is typically running in the AWS Cloud.

AWS Mainframe Modernization Application Testing prerequisites

AWS Mainframe Modernization Application Testing feature in AWS Mainframe Modernization allows you to perform automated functional equivalence testing for your migration projects. To prepare for using the Application Testing in the AWS Mainframe Modernization console, do the following:

1. **Define test cases**: Define the basic units of testing you want to run and replay in a specific order, for your target application. For additional information on how to create test cases, see the section called “Create test cases in Application Testing”.

2. **Prepare CloudFormation template and input data**: Create a CloudFormation template that will be used to provision the target test environment. The variables from this template will be used for adding input data and output variable names in your AWS Mainframe Modernization application. For additional information, see Working with AWS CloudFormation template in *AWS CloudFormation User guide*.
3. **Ensure mainframe access and data capture**: Verify that you have access to the source mainframe. This will also ensure that you can capture and upload the source data generated by the applications running on the mainframe.

Application Testing console workflows

AWS Mainframe Modernization Application Testing console helps you create test cases, test suites, and test environment configurations.

Topics

- Create test cases in AWS Mainframe Modernization Application Testing
- Create test suites in AWS Mainframe Modernization Application Testing
- Create test environment configurations in AWS Mainframe Modernization Application Testing

Create test cases in AWS Mainframe Modernization Application Testing

A **test case** is an atomic unit that represents a certain action in your workflow. For additional information on various concepts, see ???.

> **Important**
>
> You need to create at least one test environment configurations first before running test cases. To create your first environment configuration, see the section called “Create test environment configurations in Application Testing”.

Topics

- Create a Batch test case
- Create an Online 3270 screen test case

Create a Batch test case

Batch test cases allow you to submit a batch to read, process, and modify or produce new business data (database and/or data set records).

To create a Batch test case

1. Open the AWS Mainframe Modernization Application Testing console at https://console.aws.amazon.com/apptest/.
2. In the AWS Region selector, choose the Region where Application Testing is available.

   > **Note**
   >
   > Application Testing is currently available in US East (N. Virginia), Asia Pacific (Sydney), Europe (Frankfurt), and South America (São Paulo) regions only.

3. In the left navigation pane, select **Test cases**.
4. In **Define test case**, enter your test case name and optional description. Choose **Batch** under Test case type.
5. Choose **Next**.
6. (Optional) On **Specify batch JCL parameters** page, add the JCL (job control language) name and your job parameters (names and values).
7. Choose **Next**.
8. On **Data source to capture** page, you can choose either **Relational database changes**, **Data sets**, or both.

   - Choose **Relational database changes** when you want the test case to modify database records.
   - Choose **Data sets** when you want the test case to modify data sets. Under **Output data sets**, add the name of your output data set.

     > **Note**
     >
     > You can add multiple data sets.

9. Choose **Next**.
10. On **Review and create** page, review all information and choose **Create test case**.

Create an Online 3270 screen test case

Online 3270 screen test cases allow you to run interactive screen dialogs (3270) to read, modify, or produce new business data (database and/or data set records).

To create an Online 3270 screen test case

1. Open the AWS Mainframe Modernization Application Testing console at https://console.aws.amazon.com/apptest/.
2. In the AWS Region selector, choose the Region where Application Testing is available.

   > **Note**
   >
   > Application Testing is currently available in US East (N. Virginia), Asia Pacific (Sydney), Europe (Frankfurt), and South America (São Paulo) regions only.

3. In the left navigation pane, select **Test cases**.
4. In **Define test case**, enter your test case name and optional description. Choose **Online 3270 screens** under Test case type.
5. Choose **Next**.

   > **Note**
   >
   > Online 3270 screen does not need you to Specify JCL parameters.

6. Choose **Next**.
7. On **Data source to capture** page, the default selection is **Online 3270 screens**. Additionally, you can choose **Relational database changes** and **Data sets**.

   - Choose **Relational database changes** when you want the test case to modify database records.
   - Choose **Data sets** when you want the test case to modify data sets. Under **Output data sets**, add the name of your output data set.

     > **Note**
     >
     > You can add multiple data sets.

8. Choose **Next**.
9. On **Review and create** page, review all information and choose **Create test case**.

Create test suites in AWS Mainframe Modernization Application Testing

Test suites are series of test cases that are run in a sequential order. Test suites are important for replaying test cases.

> **Important**
>
> Before creating test suites, you need to have at least one test case. You can create your first test case using, the section called "Create test cases in Application Testing".

For additional information on various concepts, see the section called "Application Testing concepts".

Topics

- Create a test suite
- Upload reference data
- Replay and compare

Create a test suite

Test suites allow you to run different test cases, and replay and compare them later.

To create a test suite

1. Open the AWS Mainframe Modernization Application Testing console at https://console.aws.amazon.com/apptest/.
2. In the AWS Region selector, choose the Region where Application Testing is available.

> **Note**
>
> Application Testing is currently available in US East (N. Virginia), Asia Pacific (Sydney), Europe (Frankfurt), and South America (São Paulo) regions only.

3. In the left navigation pane, select **Test cases**.
4. Choose **Create test suites**.
5. In the **Create test suites** section, find test cases from the test case library, and choose **Add selected test cases**.

> **Note**
>
> You can add up to 20 test cases in one test suite.

6. In the **Test suite** pane, enter your test suite name and optional description. Also, select from the either managed runtime or non-managed runtime, which will define how test suite configures and deconfigures an AWS Mainframe Modernization application. Optionally, add the AWS Mainframe Modernization import data set JSON S3 URI.
7. In the **Added test cases** section, stack your test cases in the order you want to upload and replay them.
8. Choose **Create test suite**.

Upload reference data

Upload mainframe reference data to AWS Application Testing. You only need to save the uploaded reference data the first time. The testing service can reuse the uploaded results from the source and compare them consecutively with the replayed results on the target.

To upload reference data

1. From the **Test suites section**, choose the test suite to upload reference data.
2. Choose **Upload**.
3. On the **Upload reference data** page, select test cases you want to replay. Complete fields for Data captured date, Database change journal S3 location, Data sets S3 location, and Choose **Upload**.

Replay and compare

Replay and compare process associates your test case to the target test environment and runs the application. You need to upload data before running the replay process.

To replay and compare

1. From the **Test suites section**, choose the test suite to replay.
2. Choose **Replay and compare**.
3. On the **Replay and compare overview** page, select your **test environment configuration** and review information. **Edit** function allows you to edit any test environment configuration fields. You can also find AWS CloudFormation parameters.
4. Under **Test cases to be replayed** section, choose test cases and place them in the order you want to replay them.
5. Choose **Replay and compare**.

Create test environment configurations in AWS Mainframe Modernization Application Testing

Test environment configurations allow you to set up the initial set of data and configuration parameters (or resources) with AWS CloudFormation that you need to make the test run repeatable.

For additional information on various concepts, see the section called “Application Testing concepts”.

Create a test environment configuration

Configure your test environment to replay and compare test cases in Application Testing.

Set up test environment configurations

1. Open the AWS Mainframe Modernization Application Testing console at https://console.aws.amazon.com/apptest/.
2. In the AWS Region selector, choose the Region where Application Testing is available.

> **Note**
>
> Application Testing is currently available in US East (N. Virginia), Asia Pacific (Sydney), Europe (Frankfurt), and South America (São Paulo) regions only.

3. In the left navigation pane, select **Test environment configurations** .

4. Choose **Create test environment configuration**.
5. On the **Create test environment configuration** pane, enter the name and description. Also add your Amazon S3 bucket that contains the CloudFormation template for Application Testing. Additionally, you can add the CloudFormation input parameters that will be used during the CloudFormation stack creation.
6. Specify your **AWS Mainframe Modernization application** that will be affected by this test configuration. Add output variable name for AWS Mainframe Modernization application ID, runtime engine (AWS Blu Age non-managed or Rocket Software (formerly Micro Focus) managed).

   > **Note**
   >
   > The output variable name for AWS Mainframe Modernization application ID should match the output variable name from the CloudFormation template for stack creation.

   > **Important**
   >
   > AWS Blu Age non-managed runtime also requires you to specify the output variable name for VPC endpoint service ID, output variable name for listener port, and output variable name for WebApp name. These names should match the output variable names from the CloudFormation template.

7. (Optional) Additional attribute such as output variable name can be defined for the Database Migration Service (DMS) task Amazon Resource Name (ARN), which is used for capturing relational database changes. Another attribute is Source database DDL S3 URI.

   > **Important**
   >
   > The output variable name should match the variable name from CloudFormation template.

8. (Optional) Customize your Key Management Service (KMS) key. For more information, see Managing access to customer managed keys in the *AWS Key Management Service Developer Guide*.
9. Choose **Create test environment configuration**.

Tutorial: Set up the CardDemo sample application in AWS Mainframe Modernization Application Testing

For this tutorial, you create an AWS CloudFormation stack that helps you set up the CardDemo sample application for replatforming with Micro Focus on AWS Mainframe Modernization managed service, and features including AWS Mainframe Modernization Application Testing. The tutorial describes a sample AWS CloudFormation template that you can use to create the stack. We also provide a zipped file of the necessary application artifacts. The example template provisions a database, a runtime environment, an application, and a fully isolated network environment.

This template creates several AWS resources. You will be billed for them if you create a stack from this template.

Prerequisites

- Download and unzip the `IC3-card-demo-zip` and `datasets_Mainframe_ebcdic.zip`. These files contain the CardDemo sample and sample datasets for use with AWS Application Testing.
- Create an Amazon S3 bucket to hold the CardDemo files and other artifacts. For example, `my-carddemo-bucket`.

Step 1: Prepare to set up CardDemo

Upload the CardDemo sample files and edit the AWS CloudFormation template that will create the CardDemo application.

1. Upload the `datasets_Mainframe_ebcdic` and `IC3-card-demo` folders that you unzipped previously to your bucket.
2. Download the `aws-m2-math-mf-carddemo.yaml` AWS CloudFormation template from your bucket. It is in the `IC3-card-demo` folder.
3. Edit the `aws-m2-math-mf-carddemo.yaml` AWS CloudFormation template as follows:
    - Change the `BucketName` parameter to the name of the bucket that you defined previously, such as `my-carddemo-bucket`.
    - Change the `ImportJsonPath` to the location in your bucket of the `mf-carddemo-datasets-import.json` file. For example, `s3://my-carddemo-bucket/IC3-card-`

demo/mf-carddemo-datasets-import.json Updating this value makes sure that the output M2ImportJson has the correct value.

- (Optional) Adapt the EngineVersion and InstanceType parameters to match your standards.

> **Note**
>
> Do not modify the M2EnvironmentId and M2ApplicationId outputs. Application Testing uses those values to locate the resources with which it will interact.

Step 2: Create all necessary resources

Run your customized AWS CloudFormation template to create all the resources you need to complete this tutorial successfully. This template sets up the CardDemo application so that you can use it in testing.

1. Log in to the AWS CloudFormation console and choose **Create stack**, then choose **With new resources (standard)**.
2. In **Prerequisite - Prepare template**, choose **Template is ready**.
3. In **Specify template**, choose **Upload a template file**, then choose **Choose file**.
4. Navigate to where you downloaded aws-m2-math-mf-carddemo.yaml and choose that file, then choose **Next**.
5. In **Specify stack details** provide a name for the stack so you can easily find it in a list and then choose **Next**.
6. In **Configure stack options**, keep the default values and choose **Next**.
7. In **Review**, check what AWS CloudFormation is creating for you, and then choose **Submit**.

It takes about 10–15 minutes for AWS CloudFormation to create the stack.

> **Note**
>
> The template is set up to append a unique suffix to the names of the resources it creates. This means that you can create multiple instances of this stack template in parallel, a key feature for Application Testing that allows you to run multiple test suites at the same time.

Step 3: Deploy and start the application

Deploy the CardDemo application that AWS CloudFormation created for you and make sure it is running.

1. Open the AWS Mainframe Modernization console and choose **Applications** from the left navigation.
2. Choose the CardDemo application, which is named something like `aws-m2-math-mf-carddemo-abc1d2e3`.
3. Choose **Actions**, then choose **Deploy application**.
4. In **Environments**, choose the runtime environment that corresponds to the application. It will have the same unique identifier appended to the end of the name. For example, `aws-m2-math-mf-carddemo-abc1d2e3`.
5. Choose **Deploy**. Wait until the application deploys successfully and is in the Ready state.
6. Choose the application, then choose **Actions** and **Start application**. Wait until the application is in the Running state.
7. In the application details page, copy the **Port** and **DNS Hostname**, which you need in order to connect to the running application.

Step 4: Import initial data

To use the CardDemo sample application, you must import an initial set of data. Complete the following steps.

1. Download the `mf-carddemo-datasets-import.json` file.
2. Edit the file in your preferred text editor.
3. Locate the `s3Location` parameter and update the value to point to the Amazon S3 bucket you created.
4. Make this same change for all occurrences of `s3Location`, then save the file.
5. Log in to the Amazon S3 console and navigate to the bucket you created earlier.
6. Upload the customized `mf-carddemo-datasets-import.json` file.
7. Open the AWS Mainframe Modernization console and choose **Applications** from the left navigation.
8. Choose the CardDemo application.

9. Choose **Data sets** and then choose **Import**.
10. Navigate to the location in Amazon S3 where you uploaded the customized JSON file and choose **Submit**.

This job imports 23 datasets. To monitor the outcome of the import job, check the console. When all datasets are successfully imported, connect to the application.

> **Note**
>
> When you use this template in Application Testing, the Output `M2ImportJson` automatically handles the import process.

Step 5: Connect to the CardDemo application

Connect to the CardDemo sample application using the 3270 emulator of your choice.

- When the application is running, use your 3270 emulator to connect to the application, specifying the DNS hostname and the port name, if necessary.

  For example, if you are using the open source c3270 emulator, your command looks like this:

  ```
  c3270 -port port-number DNS-hostname
  ```

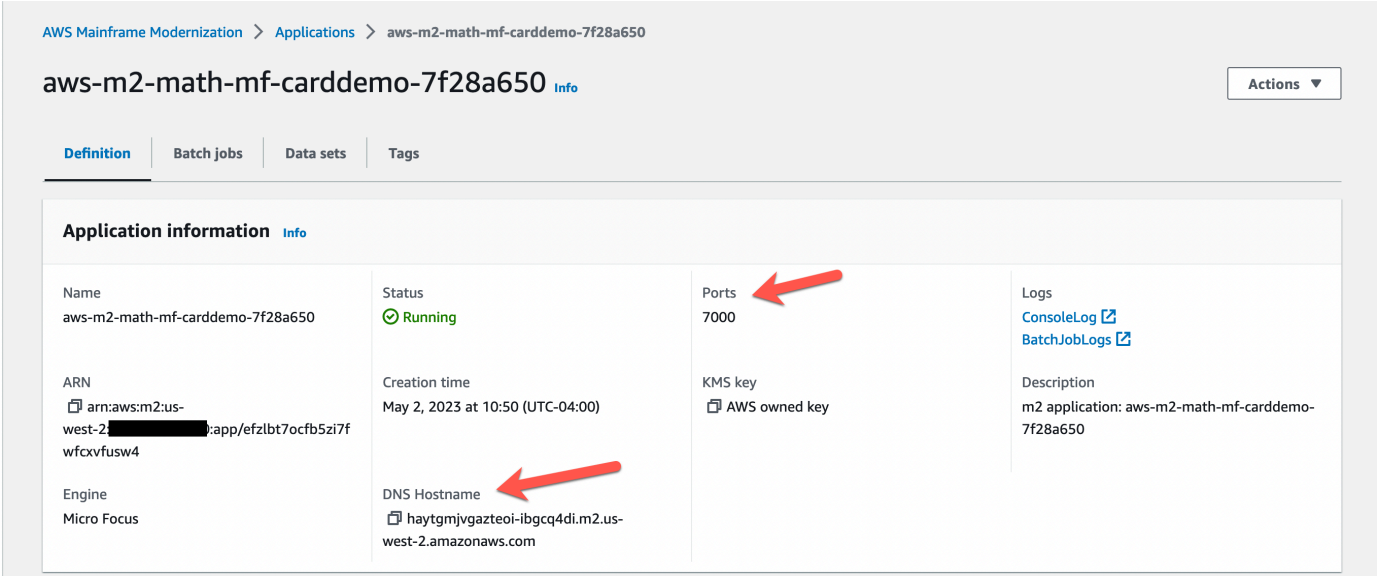
  port

  The port specified on the application detail page. For example, 6000.

  Hostname

  The DNS Hostname specified on the application detail page.

  The following figure shows where to find the port and DSN Hostname.



Tutorial: Replay and compare in AWS Mainframe Modernization Application Testing using CardDemo for AWS Blu Age deployed on Amazon EC2

In this tutorial, you will complete required steps to replay and compare testing workloads with the CardDemo application running on AWS Blu Age deployed on Amazon EC2.

Step 1: Obtain AWS Blu Age Amazon EC2 Amazon Machine Image (AMI)

Follow the instructions in the AWS Blu Age Runtime (on Amazon EC2) Setup tutorial for onboarding steps required to get access to AWS Blu Age on Amazon EC2 AMI.

Step 2: Start an Amazon EC2 instance using the AWS Blu Age AMI

1. Set up your AWS credentials.
2. Identify the location of the **3.5.0** Amazon EC2 AMI binary file (CLI only/AWS Blu Age version) from the Amazon S3 bucket:

```
aws s3 ls s3://aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1/
aws s3 ls s3://aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1/3.5.0/AMI/
```

> **Note**
>
> Application Testing feature is only available to use in 4 regions in prod (us-east-1, sa-east-1, eu-central-1 and ap-southeast-2).

3. Restore the AMI in your account with the following command:

```
aws ec2 create-restore-image-task --object-key 3.5.0/AMI/ami-0182ffe3b9d63925b.bin
 --bucket aws-bluage-runtime-artifacts-xxxxxxx-eu-west-1 --region eu-west-1 --name
 "AWS BLUAGE RUNTIME AMI"
```

> **Note**
>
> Replace the AMI bin file name and the Region where you want to create the AMI.

4. After you create an Amazon EC2 instance you can find the correct AMI ID that was restored AMI from the Amazon S3 bucket in the Amazon EC2 image catalog.

> **Note**
>
> In this tutorial, the AMI ID is **ami-0d0fafcc636fd1e6d**, and you must change this ID in the different configuration files to the one provided to you.

1. If the **aws ec2 create-restore-image-task** fails, then check your version of Python and CLI using the following command:

```
aws --version
```

> **Note**
>
> Python version must be >= 3 and CLI version must be >= 2.

2. If these versions are obsolete, the CLI must be updated. To update the CLI:
   a. Follow the instructions in Install or update the latest version of the AWS CLI.
   b. Remove CLI v1 with the following command:

```
sudo yum remove awscli
```

c. And install CLI v2 with the following commands:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
 "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

d. Finally, check version of Python and CLI with the following command:

```
aws --version
```

3. You can then redo the **aws ec2 create-restore-image-task**.

Step 3: Upload CardDemo dependent files to S3

Copy the content of folders **databases**, **file-system**, and **userdata**. Download and unzip the CardDemo applications. These three folders must be copied into one of your buckets called **amzn-s3-demo-bucket** in this documentation.

Step 4: Load databases and initialize the CardDemo application

Create a temporary Amazon EC2 instance that you will use as a **compute resource** to generate the required database snapshots for the CardDemo application. This EC2 instance will not run the CardDemo application itself, but instead generate the database snapshots that will be used later.

Start by editing the provided CloudFormation template named 'load-and-create-ba-snapshots.yml.' This is the CloudFormation template that's used to create the Amazon EC2 instance used to generate the database snapshots.

1. Generate and provide your EC2 key pair that will be used for the EC2 instance. For more information, see Create key pairs.

   Example:

```
Ec2KeyPair:
    Description: 'ec2 key pair'
    Default: 'm2-tests-us-west-2'
```

```
    Type: String
```

2. Specify the Amazon S3 path of your folder where you have put the **database** folder from the previous step:

```
S3DBScriptsPath:
    Description: 'S3 DB scripts folder path'
    Type: String
    Default: 's3://amzn-s3-demo-bucket/databases'
```

3. Specify the Amazon S3 path of your folder where you have put the **file-system** folder from the previous step:

```
S3ApplicationFilesPath:
    Description: 'S3 application files folder path'
    Type: String
    Default: 's3://amzn-s3-demo-bucket/file-system'
```

4. Specify the Amazon S3 path of your folder where you have put the **userdata** folder from the previous step:

```
S3UserDataPath:
    Description: 'S3 userdata folder path'
    Type: String
    Default: 's3://amzn-s3-demo-bucket/userdata'
```

5. Also specify an Amazon S3 path where you will save the result files to be used in the next step.

```
S3SaveProducedFilesPath:
    Description: 'S3 path folder to save produced files'
    Type: String
    Default: 's3://amzn-s3-demo-bucket/post-produced-files'
```

6. Change the AMI ID with the correct one obtained earlier in this tutorial using the following template:

```
  BaaAmiId:
    Description: 'ami id (AL2) for ba anywhere'
    Default: 'ami-0bd41245734fd20d9'
    Type: String
```

- You can optionally change the name of the three snapshots that will be created by the run of the load databases with CloudFormation. These will be visible in the CloudFormation stack as it's being created and will be used later in this tutorial. Remember to note the names used for the database snapshots.

```
SnapshotPrimary:
    Description: 'Snapshot Name DB BA Primary'
    Type: String
    Default: 'snapshot-primary'

  SnapshotBluesam:
    Description: 'Snapshot Name DB BA Bluesam'
    Type: String
    Default: 'snapshot-bluesam'

  SnapshotJics:
    Description: 'Snapshot Name DB BA Jics'
    Type: String
    Default: 'snapshot-jics'
```

> **Note**
>
> In this document, we assume that the name of the snapshots remains consistent.

7. Run the CloudFormation with CLI or AWS console using the Create Stack button and wizard. At the end of the process, you should see three snapshots in the RDS console with the name you chose followed by a unique ID. You will need these names in the next step.

> **Note**
>
> RDS will add postfixes to the snapshots names defined in the AWS CloudFormation template. Be sure to obtain the full snapshot name from RDS before proceeding to the next step.

Sample CLI command-

```
aws cloudformation create-stack --stack-name load-and-create-ba-snapshots --
template-url https://your-apptest-bucket.s3.us-west-2.amazonaws.com/load-and-
create-ba-snapshots.yml --capabilities CAPABILITY_NAMED_IAM
```

You can also check in the Amazon S3 path that you provided for **S3SaveProducedFilesPath** that the datasets have been correctly created.

Step 5: Launch AWS Blu Age runtime CloudFormation

Use CloudFormation to run the Amazon EC2 instance with the CardDemo AWS Blu Age application. You must replace some variables in the CloudFormation named `m2-with-ba-using-snapshots-https-authentication.yml` by editing the YAML file or by modifying the values in the console during launch of the CFN.

1. Modify the `AllowedVpcEndpointPrincipals` to specify which account will reach the VPC endpoint for accessing the AWS Blu Age runtime, using the following commands:

```
AllowedVpcEndpointPrincipals:
    Description: 'comma-separated list of IAM users, IAM roles, or AWS accounts'
    Default: 'apptest.amazonaws.com'
    Type: String
```

2. Change the value of variables SnapshotPrimaryDb, SnapshotBlusamDb, and SnapshotJicsDb to the name of the snapshots. Also obtain the snapshots names from RDS after they were created in the previous step.

```
SnapshotPrimary:
    Description: 'Snapshot DB cluster for DB Primary'
    Type: String
    Default: 'snapshot-primary87d067b0'

  SnapshotBluesam:
    Description: 'Snapshot DB cluster for DB Bluesam'
    Type: String
    Default: 'snapshot-bluesam87d067b0'

  SnapshotJics:
    Description: 'Snapshot DB cluster for DB Jics'
    Type: String
```

```
    Default: 'snapshot-jics87d067b0'
```

> **Note**
>
> RDS will add its own postfix to the snapshot names.

3. Provide your Amazon EC2 key pair for the EC2 instance, using this command:

```
Ec2KeyPair:
    Description: 'ec2 key pair'
    Default: 'm2-tests-us-west-2'
    Type: String
```

4. Provide the AMI ID that you have obtained during the AMI registration process for the variable **BaaAmiId**, using:

```
BaaAmiId:
    Description: 'ami id (AL2) for ba anywhere'
    Default: 'ami-0d0fafcc636fd1e6d'
    Type: String
```

5. Provide the Amazon S3 folder path that you used in the previous step to save the produced files, using the following command:

```
S3ApplicationFilesPath:
    Description: 'bucket name'
    Type: String
    Default: 's3://amzn-s3-demo-bucket/post-produced-files'
```

6. Lastly, provide the folder path of the **s3-userdata-folder-path**:

```
S3UserDataPath:
    Description: 'S3 userdata folder path'
    Type: String
    Default: 's3://amzn-s3-demo-bucket/userdata'
```

- (Optional) You can enable the **HTTPS** mode and the basic **HTTP** authentication for tomcat. Although the default settings would also work.

> **Note**
>
> By default, the HTTPS mode is disabled and set to mode HTTP in the parameter **BacHttpsMode**:

For example:

```
BacHttpsMode:
    Description: 'http or https for Blue Age Runtime connection mode '
    Default: 'http'
    Type: String
    AllowedValues: [http, https]
```

- (Optional) To enable HTTPS mode, you must change the value to HTTPS and to provide your ACM certificate ARN by changing the value of the variable **ACMCertArn**:

```
ACMCertArn:
    Type: String
    Description: 'ACM certificate ARN'
    Default: 'your arn certificate'
```

- (Optional) The basic authentication is disabled by default with the parameter **WithBacBasicAuthentication** set to false. You can enable it by setting the value to **true**.

```
WithBacBasicAuthentication:
    Description: 'false or true for Blue Age Runtime Basic Authentication '
    Default: false
    Type: String
    AllowedValues: [true, false]
```

7. When you have completed the configuration, you can create the stack by using the edited CloudFormation template.

Step 6: Testing the AWS Blu Age Amazon EC2 instance

Manually run the CloudFormation template to create the AWS Blu Age Amazon EC2 instance for the CardDemo application to make sure that it starts without errors. This is done to verify that the CloudFormation template and all prerequisites are valid, before using the CloudFormation

template with the Application Testing feature. You can then use Application Testing to automatically create the target AWS Blu Age Amazon EC2 instance during replay and compare.

1. Run the CloudFormation create stack command to create the AWS Blu Age Amazon EC2 instance, providing the **m2-with-ba-using-snapshots-https-authentication.yml** CloudFormation template you edited in the previous step:

```
aws cloudformation create-stack --stack-name load-and-create-ba-snapshots —-
template-url https://apptest-ba-demo.s3.us-west-2.amazonaws.com/m2-with-ba-using-
snapshots-https-authentication.yml --capabilities CAPABILITY_NAMED_IAM --region us-
west-2
```

> **Note**
>
> Remember to specify the correct Region where the AWS Blu Age AMI was restored.

2. Make sure that everything is working correctly by looking in the console to find the running Amazon EC2 instance. Connect to it using Session Manager.
3. After you are connected to the Amazon EC2 instance, use the following commands:

```
sudo su
cd /m2-anywhere/tomcat.gapwalk/velocity/logs
cat catalina.log
```

4. Make sure that there are no exceptions or errors in the log.
5. Next, check that the application is responding by using this command:

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

You will see the message, "Jics application is running."

Step 7: Validate previous steps were completed correctly

In the next several steps, we will use AWS Mainframe Modernization Application Testing to replay and compare datasets created by the CardDemo application. These steps rely on successful completion of all previous steps in this tutorial. Validate the following before proceeding:

1. You have successfully created the AWS Blu Age on Amazon EC2 instance through the AWS CloudFormation template.
2. The Tomcat service on the AWS Blu Age on Amazon EC2 is up and running, without exceptions.

When you get the EC2 instance running with the CardDemo application, complete the following steps on the Application Testing console to perform replay and compare for batch datasets.

Step 8: Create the test case

In this step, you create the test case that will be used to compare the datasets created in the Card Demo application.

1. Create a new test case. Give it a name and description.
2. Specify `CREASTMT.JCL` as the JCL name.
3. Add following datasets to Test case definition:

Name	CCSID	RecordFormat	RecordLength
AWS.M2.CARDDEMO.STATEMNT.PS	"037"	FB	80
AWS.M2.CARDDEMO.STATEMNT.HTML	"037"	FB	100

> **Note**
>
> Your JCL name and dataset details must match.

Step 9: Create a test suite

1. Create a new test suite, and provide a name and description for it.
2. Add the test case that you created in the previous step to your test suite.

3. Once the test suite is created, capture test cases on mainframe, and upload mainframe reference data to AWS Application Testing.
4. Choose **Create test suite**.

Step 10: Create a test environment configuration

1. Create a new test environment configuration, and provide a name and description for it.
2. Add your CloudFormation template. You can also add input parameter name and value from you CloudFormation template.
3. Choose AWS Mainframe Modernization service AWS Blu Age non-managed as your runtime.
4. Add the output variable name for name for AWS Mainframe Modernization application ID, output variable name for VPC endpoint service ID, output variable name for Listener port, and output variable name for WebApp name.

> **Note**
>
> The names of these fields should match the output variable names from the CloudFormation template that will be returned from AWS Mainframe Modernization during stack creation.

5. (Optional) Choose output variable name for DMS (Database Migration Service) task ARN and source database DDL (Database definition language) S3 URI location.
6. (Optional) Customize your Key Management Service (KMS) key. For more information, see Managing access to customer managed keys in the *AWS Key Management Service Developer Guide*.
7. Choose **Create test environment configuration**.

Step 11: Upload your input data in test suite

In this step, you run test cases on the source. To do that:

1. Download and run the datasets that originated from the mainframe run of the CardDemo application.
2. Upload the unzipped folder to your Amazon S3 bucket. This Amazon S3 bucket must be in the same Region as your other Application Testing resources.

> **Note**
>
> There should be two files with the names matching the dataset names passed in the previous test case.

3. On the **Test suite overview** page, choose the **Upload** button.
4. On the **Upload reference data** page, specify the Amazon S3 location to where you uploaded the datasets obtained from the source mainframe.
5. Choose **Upload** to start the upload process.

> **Note**
>
> Wait for the recording to complete before you perform replay and compare.

Step 12: Replay and compare

Run the test suite and test cases in the target AWS AWS Blu Age on Amazon EC2 environment. Application Testing will capture the replay produced datasets, and compare them to the reference datasets that were recorded on the mainframe.

1. Choose **Replay and compare**. It should take about three minutes to create the CloudFormation stack, and perform the comparison.

Once everything is complete, you should have comparison results with a few differences intentionally created for the purpose of this demo.

AWS Mainframe Modernization Application Testing supported data sets code pages

Use the following table to determine whether the coded character set identifier (CCSID) for your data is supported on AWS Application Testing. If your data uses an unsupported CCSID, we recommend that you either convert it to a supported CCSID or contact us for help.

CCSID	Character sets	Description
37	IBM037, IBM-037, Cp037	Host: USA, Canada (ESA), Netherlands, Portugal, Brazil, Australia, New Zealand
273	IBM273, IBM-273, Cp273	Host: Austria, Germany
277	IBM277, IBM-277, Cp277	Host: Denmark, Norway
278	IBM278, IBM-278, Cp278	Host: Finland, Sweden
280	IBM280, IBM-280, Cp280	Host: Italy
284	IBM284, IBM-284, Cp284	Host: Spain, Latin America (Spanish)
285	IBM285, IBM-285, Cp285	Host: United Kingdom
297	IBM297, IBM-297, Cp297	Host: France
300	IBM-300	JAPAN DB EBCDIC
301	IBM-301	PC data: Japan DB
437	IBM437, IBM-437, US-ASCII, ASCII, Cp437, US-ASCII	PC data: PC Base USA, many other countries
500	IBM500, IBM-500, Cp500	Host: Belgium, Canada (AS/400), Switzerland, International Latin-1
720	IBM-720	MSDOS ARABIC
737	IBM-737, x-IBM737	MSDOS GREEK
775	IBM775, IBM-775	MSDOS BALTIC
808	IBM-808	PC data: Cyrillic, Russia, with euro

CCSID	Character sets	Description
813	ISO-8859-7, ISO8859_7	ISO 8859-7: Greece
819	ISO-8859-1, ISO8859_1	ISO 8859-1: Latin-1 countries
833	IBM-833	KOREAN EBCDIC
834	IBM-834, x-IBM834	KOREAN DB EBCDIC
835	IBM-835	T-CHINESE DB EBCD
836	IBM-836	S-CHINESE EBCDIC
837	IBM-837	S-CHINESE EBCDIC
850	IBM850, IBM-850, Cp850	PC data: Latin-1 countries
855	IBM855, IBM-855, Cp855	PC data: Cyrillic
856	IBM-856, x-IBM856, Cp856	PC data: Hebrew
858	IBM00858, IBM-858, Cp858	PC data: Latin-1 countries, with euro
859	IBM-859	PC data: LATIN-9
860	IBM860, IBM-860	PC data: Portuguese
861	IBM861, IBM-861	PC data: Iceland
862	IBM862, IBM-862, Cp862	PC data: Hebrew (migration)
863	IBM863, IBM-863	PC data: Canada
865	IBM865, IBM-865, Cp865	PC data: Den/Norway
866	IBM866, IBM-866, Cp866	PC data: Cyrillic, Russia
867	IBM-867	PC data: Hebrew with euro
870	IBM870, IBM-870, Cp870	Host: Latin-2 multilingual

CCSID	Character sets	Description
871	IBM871, IBM-871, Cp871	Host: Iceland
874	x-IBM874	PC data: Thai
875	IBM-875, x-IBM875, Cp875	Host: Greece
897	IBM-897	PC data: Japan SB
912	ISO-8859-2, ISO8859_2	ISO 8859-2: Latin-2 multiling ual
915	ISO-8859-5, ISO8859_5	ISO 8859-5: Cyrillic
916	ISO-8859-8, ISO8859_8	ISO 8859-8: Hebrew
918	IBM918, IBM-918, Cp918	Host: Urdu
920	ISO-8859-9, ISO8859_9	ISO 8859-9: Latin-5 (ECMA-128, Turkey TS-5881)
921	IBM-921, x-IBM921, Cp921	PC data: Latvia, Lithuania
922	IBM-922, x-IBM922, Cp922	PC data: Estonia
923	ISO-8859-15, Cp923, ISO8859_15_FDIS	ISO 8859-15: Latin-9
924	IBM-924	ISO 8859-15: Latin-9
927	IBM-927	PC data: T-Chinese
930	IBM-930, x-IBM930, Cp930	Katakana Host: extended SBCS. Kanji Host: DBCS including 4370 user-defined characters
932	IBM-932	PC data: Japan Mix

CCSID	Character sets	Description
933	IBM-933, x-IBM933, Cp933	Host: Extended SBCS. Host: DBCS including 1880 user-defined characters and 11172 full Hangul characters
935	IBM-935, x-IBM935, Cp935	Host: Extended SBCS. Host: DBCS including 1880 user-defined characters.
937	IBM-937, x-IBM937, Cp937	Host: Extended SBCS. Host: DBCS including 6204 user-defined characters
939	IBM-939, x-IBM939, Cp939	Latin Host: extended SBCS. Kanji Host: DBCS including 4370 user-defined characters.
942	IBM-942, IBM-942C, x-IBM942, x-IBM942C, Cp942, Cp942C	PC data: Extended SBCS. PC data: DBCS including 1880 user-defined characters
943	IBM-943, IBM-943C, Shift_JIS , windows-31j, windows-932, x-IBM943, x-IBM943C, Cp943, Cp943C, MS932	PC data: SBCS. PC data: DBCS for Open environment including 1880 IBM user-defined characters
947	IBM-947	T-CHINESE BIG-5
948	IBM-948, x-IBM948, Cp948	PC data: Extended SBCS. PC data: DBCS including 6204 user-defined characters
949	IBM-949, IBM-949C, x-IBM949, x-IBM949C, Cp949, Cp949C	IBM KS Code - PC data: SBCS. IBM KS code - PC data: DBCS including 1880 user-defined characters

CCSID	Character sets	Description
950	Big5, IBM-950, x-IBM950, Cp950	PC data: SBCS (IBM BIG5). PC data: DBCS including 13493 CNS, 566 IBM selected, 6204 user-defined characters
951	IBM-951	PC data: IBM KS
954	EUC-JP, IBM-954, IBM-954C	G0: JIS X201 Roman. G1: JIS X208-1990. G1: JIS X201 Katakana. G1: JIS X212
964	EUC-TW, IBM-964, x-IBM964, Cp964	G0: ASCII. G1: CNS 11643 plane 1. G1: CNS 11643 plane 2.
970	EUC-KR, x-IBM970, Cp970	G0: ASCII. G1: KSC X5601-1989 including 1880 user-defined characters
971	IBM-971	KOREAN EUC
1006	IBM-1006, x-IBM1006, Cp1006	ISO-8: Urdu
1025	IBM-1025, x-IBM1025, Cp1025	Host: Cyrillic multilingual
1026	IBM1026, IBM-1026, Cp1026	Host: Latin-5 (Turkey)
1027	IBM-1027	JAPAN LATIN EBCD
1041	IBM-1041	PC data: Japan
1043	IBM-1043	PC data: T-Chinese
1046	IBM-1046, IBM-1046S, x-IBM1046	ARABIC - PC

CCSID	Character sets	Description
1047	IBM1047, IBM-1047	Host: Latin-1
1051	hp-roman8	HP EMULATION
1088	IBM-1088	PC data: Korea KS
1089	ISO-8859-6, ISO8859_6	ISO 8859-6: Arabic
1097	IBM-1097, x-IBM1097, Cp1097	Host: Farsi
1098	IBM-1098, x-IBM1098, Cp1098	PC data: Farsi
1112	IBM-1112, x-IBM1112, Cp1112	Host: Latvia, Lithuania
1114	IBM-1114	PC data: T-CH SB
1115	IBM-1115	PC data: S-CH GB
1122	IBM-1122, x-IBM1122, Cp1122	Host: Estonia
1123	IBM-1123, x-IBM1123, Cp1123	Host: Cyrillic Ukraine
1124	IBM-1124, x-IBM1124, Cp1124	8-bit: Cyrillic, Belarus
1140	IBM01140, IBM-1140, Cp1140	Host: USA, Canada (ESA), Netherlands, Portugal, Brazil, Australia, New Zealand, with euro
1141	IBM01141, IBM-1141, Cp1141	Host: Austria, Germany, with euro

CCSID	Character sets	Description
1142	IBM01142, IBM-1142, Cp1142	Host: Denmark, Norway, with euro
1143	IBM01143, IBM-1143, Cp1143	Host: Finland, Sweden, with euro
1144	IBM01144, IBM-1144, Cp1144	Host: Italy, with euro
1145	IBM01145, IBM-1145, Cp1145	Host: Spain, Latin America (Spanish), with euro
1146	IBM01146, IBM-1146, Cp1146	Host: United Kingdom, with euro
1147	IBM01147, IBM-1147, Cp1147	Host: France, with euro
1148	IBM01148, IBM-1148, Cp1148	Host: Belgium, Canada (AS/400), Switzerland, International Latin-1, with euro
1149	IBM01149, IBM-1149, Cp1149	Host: Iceland, with euro
1200	UTF-16BE	Unicode with character set 65535. In the absence of a byte-order mark (BOM), assumed to be UTF-16 BE (big-endian).
1202	UTF-16LE	UTF-16 LE with IBM PUA
1204	UTF-16	UTF-16 with IBM PUA
1208	UTF-8, UTF-8J, UTF8	Unicode with character set 65535. UTF-8.
1232	UTF-32BE	UTF-32 BE with IBM PUA

CCSID	Character sets	Description
1234	UTF-32LE	UTF-32 LE with IBM PUA
1236	UTF-32	UTF-32 with IBM PUA
1351	IBM-1351	JAPAN OPEN
1362	IBM-1362	KOREAN MS-WIN
1363	IBM-1363, IBM-1363C, windows-949, MS949	PC data: MS Windows Korean SBCS. PC data: MS Windows Koran DBCS including 11172 full Hangul
1364	IBM-1364	Host: Extended SBCS. Host: DBCS including 1880 user-defined characters and 11172 full Hangul characters
1370	IBM-1370	PC data: Extended SBCS, with euro. PC data: DBCS including 6204 user-defined characters, with euro
1371	IBM-1371	Host: Extended SBCS, with euro. Host: DBCS including 6204 user-defined characters, with euro
1375	Big5-HKSCS	Mixed Big-5 Ext for HKSCS
1380	IBM-1380	PC data: S-CH GB
1381	IBM-1381, x-IBM1381, Cp1381	PC data: Extended SBCS (IBM GB). PC data: DBCS (IBM GB) including 31 IBM-selected, 1880 user-defined characters

CCSID	Character sets	Description
1382	IBM-1382	S-CHINESE EUC
1383	EUC-CN, GB2312, IBM-1383, x-IBM1383, Cp1383	G0: ASCII. G1: GB 2312-80 set
1385	IBM-1385	PC data: S-CH GBK
1386	GBK, IBM-1386, windows-9 36, MS936	PC data: S-Chinese GBK and T-Chinese IBM BIG-5. PC data: S-Chinese GBK
1388	IBM-1388	Host: Extended SBCS. Host: DBCS including 1880 user-defined characters
1390	IBM-1390	Katakana Host: extended SBCS, with euro. Kanji Host: DBCS including 6205 user-defined characters
1399	IBM-1399	Latin Host: extended SBCS, with euro. Kanji Host: DBCS including 4370 user-defined characters, with euro
5050	JIS0201, JIS0208, JIS0212, JIS0201, JIS0208, JIS0212	G0: JIS X201 Roman. G1: JIS X208-1990. G1: JIS X201 Katakana. G1: JIS X212
5054	ISO-2022-JP	JAPANESE TCP
5346	windows-1250, Cp1250	MS Windows: Latin-2, version 2 with euro
5347	windows-1251, Cp1251	MS Windows: Cyrillic, version 2 with euro

CCSID	Character sets	Description
5348	windows-1252, Cp1252	MS Windows: Latin-1 countries, version 2 with euro
5349	windows-1253, Cp1253	MS Windows: Greece, version 2 with euro
5350	windows-1254, Cp1254	MS Windows: Turkey, version 2 with euro
5351	windows-1255, Cp1255	MS Windows: Hebrew, version 2 with euro
5352	windows-1256, windows-1256S, Cp1256	MS Windows: Arabic, version 2 with euro
5353	windows-1257, Cp1257	MS Windows: Baltic Rim, version 2 with euro
5354	windows-1258, Cp1258	MS Windows: Vietnamese, version 2 with euro
5488	GB18030	GB18030, 1-byte data GB18030, 2-byte data GB18030, 4-byte data
9030	IBM-838, Cp838	Host: Thai extended SBCS
9066	IBM-874, Cp874	PC data: Thai extended SBCS
9400	CESU-8	CESU-8 with IBM PUA
25546	ISO-2022-KR	KOREAN TCP
33722	IBM-33722, IBM-33722C	IBMeucJP

Data protection in AWS Mainframe Modernization Application Testing

The AWS shared responsibility model applies to data protection in AWS Mainframe Modernization Application Testing. 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. You are also responsible for 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*.

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). As a result, 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 recommend that you avoid using any confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields (for example, Name field). This includes when you work with AWS Mainframe Modernization Application Testing 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 might be used for billing or diagnostic logs. If you provide a URL to an external server, avoid using credentials information in the URL to validate your request to that server.

Data collected by the AWS Mainframe Modernization Application Testing

AWS Mainframe Modernization Application Testing collects several types of data from you:

- `Resource definition`: Resource definition indicates the data passed to Application Testing when you create or update a resource of type test case, test suite, or a test configuration.
- `Scripts for replay`: These are scripts passed to Application Testing against your AWS Mainframe Modernization application.
- `Data for comparison`: These are data sets or Database Change Data Capture (CDC) files passed to Application Testing for comparison.

AWS Mainframe Modernization Application Testing stores this data natively in AWS. The data we collect from you is stored in an AWS Mainframe Modernization Application Testing-managed Amazon S3 bucket. When you delete a resource, the associated data is removed from the Amazon S3 bucket.

When you start a test run to perform replay for testing interactive workloads, AWS Mainframe Modernization Application Testing downloads the script to an ephemeral storage backed-Amazon ECS-managed Fargate container to perform the replay. The script file is deleted once the replay is complete and the script generated output file is stored in Application Testing-managed Amazon S3 bucket in your account. The replay output file is deleted from the Amazon S3 bucket when you delete the test run.

Similarly, when you start a test run to compare files (datasets or database changes), AWS Mainframe Modernization Application Testing downloads the files to an ephemeral storage backed-Amazon ECS-managed Fargate container to perform the comparison. The downloaded files are deleted as soon as the comparison operation is complete. The comparison output data is stored in Application Testing-managed Amazon S3 bucket in your account. The output data is deleted from the S3 bucket when you delete the test run.

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 Application Testing uses for comparing files.

Data encryption at rest for the AWS Mainframe Modernization Application Testing

AWS Mainframe Modernization Application Testing integrates with AWS Key Management Service (KMS) to provide transparent server side encryption (SSE) on all dependent resources that store data permanently. Resource examples include Amazon Simple Storage Service, Amazon DynamoDB, and Amazon Elastic Block Store. AWS Mainframe Modernization Application Testing 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 test applications that require strict encryption compliance and regulatory requirements.

You can't disable this layer of encryption or select an alternate encryption type when you create test cases, test suites, or test configurations.

You can use your own customer managed key for comparison files and AWS CloudFormation templates to encrypt Amazon S3. You can use this key to encrypt all the resources created for test runs in Application Testing.

> **Note**
>
> DynamoDB resources are always encrypted using an AWS managed key in the Application Testing service account. You cannot encrypt DynamoDB resources using a customer managed key.

AWS Mainframe Modernization Application Testing uses your customer managed key for the following tasks:

- Exporting data sets from Application Testing to Amazon S3.
- Uploading comparison output files to Amazon S3.

For more information, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

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 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.

Following is an example key policy scoped down access with ViaService that allows Application Testing to write replay and comparison-generated data in your account. You should attach this policy to the IAM role when you invoke `StartTestRun` API.

Example

```
{
    "Sid": "TestRunKmsPolicy",
    "Action": ["kms:Decrypt", "kms:GenerateDataKey"],
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/TestRunRole"
    },
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": ["s3.amazonaws.com"]
        },
        "ForAnyValue:StringEquals": {
            "kms:EncryptionContextKeys": "aws:apptest:testrun"
        }
    }
}
```

For more information, see Managing access to customer managed keys in the *AWS Key Management Service Developer Guide*.

For more information about troubleshooting key access, see the *AWS Key Management Service Developer Guide*.

Specifying a customer managed key for AWS Mainframe Modernization Application Testing

When you create a test configuration, you can specify a customer managed key by entering a **KEY ID**. Application Testing uses to encrypt the data uploaded to the Amazon S3 bucket during the test run.

- **KEY ID**— A key identifier for a customer managed key. Enter a key ID, key ARN, alias name, or alias ARN.

To add your customer managed key when you create a test configuration with the AWS CLI, specify the `kmsKeyId` parameter, as follows:

```
create-test-configuration --name test \
--resources '[{
    "name": "TestApplication",
    "type": {
        "m2ManagedApplication": {
            "applicationId": "wqju4m2dcz3rhny5fpdozrsdd4",
            "runtime": "MicroFocus"
        }
    }
}]' \
--service-settings '{
    "kmsKeyId": "arn:aws:kms:us-west-2:111122223333:key/05d467z6-c42d-40ad-
b4b7-274e68b14013"
}'
```

AWS Mainframe Modernization Application Testing 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 Application Testing encryption context

AWS Mainframe Modernization Application Testing uses the same encryption context in all AWS KMS cryptographic operations related to a test run, where the key is `aws:apptest:testrun` and the value is the unique identifier of the test run.

Example

```
"encryptionContext": {
        "aws:apptest:testrun": "u3qd7uhdandgdkhhi44qv77iwq"
}
```

Using encryption context for monitoring

When you use a symmetric customer managed key to encrypt your test run, you can also use the encryption context in audit records and logs to identify how the customer managed key is being used when uploading data to Amazon S3.

Monitoring your encryption keys for AWS Mainframe Modernization Application Testing

When you use an AWS KMS customer managed key with your AWS Mainframe Modernization Application Testing resources, you can use AWS CloudTrail to track requests that AWS Mainframe Modernization Application Testing sends to Amazon S3 when uploading objects.

Encryption in transit

For test cases that define steps to test transactional workloads, the data exchanges between the Application Testing managed terminal emulator running your selenium scripts and the AWS Mainframe Modernization application endpoints are not encrypted in transit. AWS Mainframe Modernization Application Testing uses AWS PrivateLink to connect to your application endpoint to privately exchange data without exposing the traffic over the public internet.

AWS Mainframe Modernization Application Testing uses HTTPS to encrypt the service APIs. All other communication within AWS Mainframe Modernization Application Testing is protected by the service VPC or security group, as well as HTTPS.

Basic encryption in transit is configured by default, but does not apply to `TN3270` protocol based interactive workload tests.

How AWS Mainframe Modernization Application Testing works with IAM

Before you use IAM to manage access to AWS Mainframe Modernization Application Testing, learn what IAM features are available to use with AWS Mainframe Modernization Application Testing.

IAM features you can use with AWS Mainframe Modernization Application Testing

IAM feature	AWS Mainframe Modernization Application Testing support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Forward access sessions (FAS)	Yes
Service roles	No
Service-linked roles	No

To get a high-level view of how AWS Mainframe Modernization Application Testing 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 Application Testing

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 Define custom IAM permissions with customer managed 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 Application Testing

To view examples of AWS Mainframe Modernization Application Testing identity-based policies, see Identity-based policy examples for AWS Mainframe Modernization.

Resource-based policies within AWS Mainframe Modernization Application Testing

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 Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS Mainframe Modernization Application Testing

Supports policy actions: Yes

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 Application Testing actions, see Actions defined by AWS Mainframe Modernization Application Testing in the *Service Authorization Reference*.

Policy actions in AWS Mainframe Modernization Application Testing use the following prefix before the action:

```
apptest
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "apptest:CreateTestCase",
      "apptest:StartTestRun"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "apptest:List*"
```

To view examples of AWS Mainframe Modernization identity-based policies, see Identity-based policy examples for AWS Mainframe Modernization.

Policy resources for AWS Mainframe Modernization Application Testing

Supports policy resources: Yes

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). 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 Application Testing 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) in the *AWS General Reference*.

For example, an AWS Mainframe Modernization Application Testing test case has the following ARN.

```
"Resource": "arn:aws:apptest:regionId:accountId:testcase/service-generated-unique-identifier"
```

An AWS Mainframe Modernization Application Testing test configuration has the following ARN.

```
"Resource": "arn:aws:apptest:regionId:accountId:testconfiguration/service-generated-unique-identifier"
```

An AWS Mainframe Modernization Application Testing test suite has the following ARN.

```
"Resource": "arn:aws:apptest:regionId:accountId:testsuite/service-generated-unique-identifier"
```

An AWS Mainframe Modernization Application Testing test run has the following ARN.

```
"Resource": "arn:aws:apptest:regionId:accountId:testrun/service-generated-unique-identifier"
```

Not all AWS Mainframe Modernization Application Testing actions support resource-level permissions. For actions that don't support resource-level permissions, you must use the wildcard (*).

The following AWS Mainframe Modernization Application Testing actions do not support resource-level permissions.

```
ListTestCases
ListTestConfigurations
ListTestRuns
ListTestSuites
ListTagsForResource
```

To see a list of AWS Mainframe Modernization Application Testing resource types and their ARNs, see Resources defined by AWS Mainframe Modernization Application Testing in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS Mainframe Modernization Application Testing.

To view examples of AWS Mainframe Modernization identity-based policies, see Identity-based policy examples for AWS Mainframe Modernization.

Policy condition keys for AWS Mainframe Modernization Application Testing

Supports service-specific policy condition keys: Yes

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*.

To see a list of AWS Mainframe Modernization Application Testing condition keys, see Condition keys for AWS Mainframe Modernization Application Testing 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 Application Testing.

To view examples of AWS Mainframe Modernization identity-based policies, see Identity-based policy examples for AWS Mainframe Modernization.

Access control lists (ACLs) in AWS Mainframe Modernization Application Testing

Supports ACLs: No

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 Application Testing

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 Define permissions with ABAC authorization 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 Application Testing

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 Switch from a user to an IAM 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.

Forward access sessions for AWS Mainframe Modernization Application Testing

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a

different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

> **Important**
>
> These tokens give AWS Mainframe Modernization Application Testing access to customer data without your explicit agreement; for example, AWS Mainframe Modernization Application Testing delivers testing results to a customer 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 Application Testing

Supports service roles: No

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 Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

Service-linked roles for AWS Mainframe Modernization Application Testing

Supports service-linked roles: No

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.

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 offers various runtime options to suit different deployment scenarios and operational preferences. These include managed and non-managed runtimes, each with its own set of features and deployment targets.

For a comprehensive overview of the available AWS Blu Age Runtime options, including managed and non-managed versions, deployment targets, and their respective characteristics, see the section called "AWS Blu Age Runtime options" documentation.

This guide will help you understand the differences between the runtime options and choose the most suitable one for your modernization project.

Topics

- AWS Blu Age releases
- AWS Blu Age Runtime concepts
- Set up configuration for AWS Blu Age Runtime

- AWS Blu Age Runtime APIs
- Set up AWS Blu Age Runtime (non-managed)
- Modify the source code with Blu Age Developer IDE
- AWS Blu Age FAQ

AWS Blu Age releases

AWS Blu Age engine has multiple versions that you can choose from. This page is an overview of the way AWS Blu Age versioning works, what changes each version releases have, upgrading instructions for versions, how updates to AWS Blu Age are communicated with customers, and the lifecycle of these versions.

The the section called “AWS Blu Age versioning” page details information about releases and how each release can be identified by major and minor versions. The the section called “AWS Blu Age release notes” page has in-depth release notes for each major and minor version. the section called “AWS Blu Age security vulnerabilities” page mentions how AWS Blu Age handles Common Vulnerabilities and Exposures (CVE). the section called “Upgrading AWS Blu Age” details upgrading instructions for AWS Blu Age versions. And the section called “AWS Blu Age lifecycle” includes all details about end of life (EOL) dates for the AWS Blu Age Runtime major versions.

Topics

- AWS Blu Age versioning
- AWS Blu Age Runtime options
- AWS Blu Age release notes
- AWS Blu Age security vulnerabilities
- Upgrading instructions for AWS Blu Age
- AWS Blu Age lifecycle

AWS Blu Age versioning

The AWS Blu Age Transformation and Runtime products are versioned using a semver (semantic versioning) compliant scheme. To deploy your application, you need to use the corresponding runtime version compatible with your modernized code. If you have questions about what version to use, contact your AWS Blu Age delivery manager.

Releases

Each *release* is identified with a **`[Major].[Minor].[Patch]`** pattern. For example, with AWS Blu Age Runtime version `4.1.0`, the major version is 4, the minor version is 1, and the patch version is 0.

We intend to release new AWS Blu Age Runtime minor versions monthly, and new major versions when there are impactful changes to the product or its dependencies.

For details on the new features available in each version, see the section called “AWS Blu Age release notes”.

Alpha pre-releases

Each *alpha pre-release* is identified with a **`[Major].[Next_Minor].0-alpha.[pre-release]`** pattern. For example, in the pre-release `4.2.0-alpha.1`, the changes available in `alpha.1` will be released in the next minor version `4.2.0`.

Alpha pre-releases are frequent short-lived versions that are intended and available for quick iteration during the modernization projects. There is no fixed release cadence for new Alpha pre-release versions, and are made available as they are developed and tested.

For more information about versioning, upgrades, and support, see *Components lifecycle*.

> **Important**
>
> Alpha pre-releases should be used during the modernization project phase only and not for production or critical workloads.

AWS Blu Age Runtime options

AWS Blu Age offers three types of Runtime options to cater to different stages of your modernization journey and operational needs. This page describes each option, its characteristics, use cases, and how to access them.

Non-managed Runtime

With AWS Blu Age Runtime (non-managed), you can deploy your modernized application in your own AWS account , allowing you to manage your own infrastructure. This option provides both

release and pre-release versions, giving you the flexibility to operate all the technical components required to run your modernized application the way you want. You can choose between stable releases for production environments or pre-release versions for testing and development purposes.

The non-managed runtime is deployed and managed by the customer, offering more control over the runtime environment. It provides automated refactoring capabilities and is suitable for customized deployment scenarios.

When to use

The non-managed runtime is suitable for testing and production environments, and is particularly useful when specific customization of the runtime environment is required.

How to access

To request access to the Non-managed Runtime artifacts, see Onboarding AWS Blu Age Runtime.

Deployment

AWS Blu Age Runtime (non-managed) is available for deployment on:

- Amazon EC2
- Amazon ECS on Amazon EC2
- Amazon EKS on Amazon EC2
- Amazon ECS managed by AWS Fargate

Deploying on Amazon EC2 can be done directly in the instance or through a Docker containerized application, which is the preferred way when using Amazon ECS or Amazon EKS.

For detailed deployment instructions, see Set up AWS Blu Age Runtime (non-managed) documentation.

Managed Runtime

With AWS Blu Age Runtime 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.

The managed runtime features infrastructure managed by AWS, with automatic updates and patches. It offers simplified operations and maintenance. Only release versions are available in the managed runtime, ensuring stability and reliability for your production environments.

When to use

The managed runtime is ideal for production environments and is best suited for situations where you prefer a hands-off approach to runtime management.

How to access

To access the AWS Blu Age Runtime Managed version, you simply need access to the following AWS services:

- Amazon S3
- AWS Mainframe Modernization

With the appropriate account permissions, you can seamlessly interact with the managed runtime environment. This streamlined access ensures that you can leverage the full capabilities of Blu Age Runtime while benefiting from AWS's managed services.

Deployment

To learn how to set up and use the Managed Runtime, see Set up managed runtime for AWS Blu Age documentation.

Developer Runtime (BluInsights toolbox)

The Developer Runtime is accessible from BluInsights Toolbox, and is designed for development and testing phases and is frequently updated with the latest features. This runtime includes both Release versions and Alpha pre-release versions, providing developers with access to stable builds as well as cutting-edge features. Its primary purpose is to support specific test or development activities in a local development environment, typically from an IDE. Importantly, this runtime is limited to 2 hours of use, making it suitable for focused development sessions rather than long-term or production deployments.

When to use

The Developer Runtime is ideal during initial development and modernization projects, as well as for quick iterations and testing of modernized applications.

How to access

Access to the BluInsights Toolbox is provided as part of your AWS Blu Age project engagement. The Developer Runtime is available through AWS Blu Age toolbox requests. Once approved, you'll have access to specific S3 buckets: `s3://toolbox-dev-runtime-<region>`

These buckets are available in the us-east-1 and us-east-2 regions. To use the available bucket, append the region to the bucket name (e.g., `s3://toolbox-dev-runtime-us-east-1`).

For detailed instructions on how to request access and set up the necessary permissions, see Dev and Special AWS Blu Age Runtimes documentation.

Deployment

To deploy the Developer Runtime:

1. List available versions using AWS CLI:

```
aws s3 ls s3://toolbox-dev-runtime
```

2. Choose a specific version and list its contents:

```
aws s3 ls s3://toolbox-dev-runtime/[version]/
```

3. Download the runtime artifact:

```
aws s3 cp s3://toolbox-dev-runtime/[version]/gapwalk-[version]-dev.tar.gz
```

4. Extract and set up the runtime according to your project requirements.

For more detailed deployment instructions and usage guidelines, see Dev and Special AWS Blu Age Runtimes documentation.

> **Note**
>
> Ensure you have the necessary S3 read permissions on your AWS account to access these buckets. An example IAM policy can be found in the Dev and Special AWS Blu Age Runtimes documentation.

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, contact AWS Blu Age delivery services. For information about the latest Blu Insights features, see Blu Insights releases.

Topics

- Release notes 4.8.0
- Runtime release 4.8.0
- AWS Blu Age Transformation Engine 4.8.0
- Release notes 4.7.0
- Runtime release 4.7.0
- AWS Blu Age Transformation Engine 4.7.0
- Release notes 4.6.0
- Runtime release 4.6.0
- AWS Blu Age Transformation Engine 4.6.0
- Release notes 4.5.0
- Runtime release 4.5.0
- AWS Blu Age Transformation Engine 4.5.0
- Release notes 4.4.0
- Runtime release 4.4.0
- AWS Blu Age Transformation Engine 4.4.0
- Release notes 4.3.0
- Runtime release 4.3.0
- Modernization tools release 4.3.0
- Release notes 4.2.0
- Runtime release 4.2.0

- Modernization tools release 4.2.0
- Release notes 4.1.0
- Runtime release 4.1.0
- Modernization tools release 4.1.0
- Release notes 4.0.0
- Runtime release 4.0.0
- Modernization tools release 4.0.0
- Release notes 3.10.0
- Runtime release 3.10.0
- Modernization tools release 3.10.0
- Release notes 3.9.0
- Runtime release 3.9.0
- Modernization tools release 3.9.0
- Release notes 3.8.0
- Runtime release 3.8.0
- Modernization tools release 3.8.0
- Release notes 3.7.0
- Runtime release 3.7.0
- Modernization tools release 3.7.0
- Release notes 3.6.0
- Runtime release 3.6.0
- Modernization tools release 3.6.0
- Release notes 3.5.0
- Runtime release 3.5.0
- Modernization tools release 3.5.0

Release notes 4.8.0

Release date: April 23, 2025

This release of AWS Blu Age Runtime and Transformation Engines introduces key updates to enhance database transformation capabilities and performance:

- **IBM IMS database transformation** — Added support for transforming IBM IMS databases into our JHDB customized Blu Age solution.
- **In-memory cache feature** — Added in-memory cache feature that enables users to cache read-only data within memory, improving performance for data-intensive programs.

We tested this version of the AWS Blu Age Runtime with the following stack.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 22.11.0
	Npm 10.9.0
	Angular 19.1.3
Service layer	Spring Boot 3.4.2
	Spring Core 6.2.2
	Spring Session 3.4.1
	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 15.10
	Oracle 21c
Application server	Apache Tomcat 10.1.17

Runtime release 4.8.0

zOS

New features

- Introduced support for defining custom headers for secured transactions through the YML property `gapwalk-application.security.customAllowedHeaders`. This property is related to `gapwalk-application.identity` property with value `oauth`.

- Added a feature to allow customers to rebuild Blusam metadata based on the actual records in the Blusam database table
- Secured JHDB connections using AWS Secrets Manager integration

Improvements

- COBOL
  - Improved support for AT END condition on READ file statement
  - Improved support for MOVE signed numeric literals to alphanumeric fields without moving the sign
  - Improved support for the UNSTRING statement to handle non-numeric to numeric field conversions and retain low-values when the delimiter is not found and no value is moved to other variables
  - Improved the address bind operation for dynamic linkage entities
  - Improved string concatenation operations by including support for shift in/out bytes (SOSI)
  - Improved support for the COBOL ACCEPT statement with DAY-OF-WEEK source
  - Added support for SYSIN and SYSOUT as SPECIAL NAMES values
  - Improved support for STRING concatenation to handle the form feed character
- JCL - IDCAMS
  - Added support for deleting gdg generations using a base name without the GDG parameter
  - Improved support for handling the VOLUME parameter as a variant of the standard VOLUMES parameter for the DEFINE CLUSTER statement
- JCL - SORT
  - Added support for SYMNAMES with JOINKEYS command
- JCL - IEBGENER
  - Improved support for IEBGENER to handle invalid SYSIN content by introducing validation checks and automatic fallback to ICEGENER, ensuring continuous operation even with incorrect control statements
- JCL - INFUTILB
  - Improved handling of unload INTO and FROM clauses
- JCL - DSNUTILB
  - Added support for DOUBLE PRECISION data type

- JCL - Misc
  - Improved overriding file configurations using DD names in child procedure from the job
  - DSNTIAUL/INFUTILB - Enhanced customization of SYSREC padding bytes for unload utility
  - Added support for OUTDD option for DISPLAY statement redirection
  - Improved support for Fixed Block Sequential File write. When the LRECL in the COBOL program differs from the JCL LRECL, the JCL LRECL takes precedence.
  - Added support for catalog DCB attributes for the SORTOUT file, but only for permanent datasets
  - Enhanced support for conditional JCL step transformation
  - Improved support for ACCEPT with SYSIN and lines exceeding target size with trailing spaces
  - Improved support for SET statements to handle command not including variable name, command defined inside includes a member file that should be accessible within the JCL and command contains inline comments
  - Added support to retain the job level 'scriptContext' parameters in the JCL checkpoint metadata and the job context for use at restart
- Blusam
  - Improved record insertion on Blusam large indexed data sets when writing after loading an empty data set
  - Improved performance for large data sets by enabling a warm-up mechanism and introducing an optional prefetch window for records, as well as local storage for indexes and pages
  - Improved support for Export Data Set to handle a larger dataset with AWS Key Management Service
- SQL
  - Improved support for JDBC BLOB data type
- Message Queue Integration
  - Added support for User Identifier on MQ PUT command when XA is activated
  - Added support for concurrent MQ connections when XA is activated
- IMS
  - Added a YML property `jhdb.keepParent` to control whether IMS ISRT calls populate _parent and _logicalparent fields in the database when set to true
  - Improved robustness by removing message queue for invalid transaction code

AS400

New features

- Added in-memory cache feature that enables users to cache read-only data within memory, improving performance for data-intensive programs.

Improvements

- CL
  - Improved support of QRYSLT parameter in OPNQRYF command to handle RANGE keyword, to parse value when string columns have purely numeric values and to handle empty string values
  - Added support for FTP SENDPASV command
  - Improved support for query formation in command DSPDBR executed using Qcmdexc
  - Improved the support of MONMSG exception to propagate it properly in the execution stack
  - Improved record copying between tables with varying column counts and data types on CPYF
  - Improved message monitoring priority: command-level messages now correctly take precedence over procedure-level ones
  - Improved support for CPYF command to consider the library defined by BLU4IVLIB
  - Improved support of MonMsgs in command CHKOBJ to handle the case when file doesn't exist in library
- Screen
  - Improved handling of BigDecimal values with more than 15 digits by building the value manually and then turning it into a number to be displayed rather than letting the Number be rounded
- Misc
  - Improved support for UserSpace with Replace as NO when UserSpace already exists
  - Enhanced BLU4IVLIB functionality to support both retrieval and writing of records to overridden libraries
  - Improved support for multi-dimensional array fields using DIM and OCCURS operations
  - Enhanced support for low-values passed as key field values in DAO operation
  - Removed conflict on MONMSG instruction for messages that can be handled by a local and a global monitor

- Improved setting FileStatus on Duplicate Record Insert
- Reduced userspace lock conflict and reduced creation failure risk by using registry first to check if userspace exists
- Improved support for readPriorEqual() skipping records after delete()
- Improved records locking mechanism under commitment control
- Improved handling of RPG statement RETURN in subroutines
- Improved the support of user info in SharedContext

AWS Blu Age Transformation Engine 4.8.0

zOS

Improvements

- COBOL
  - Improved Printer mode generation for multiple files.
  - Improved support for SPECIAL-NAMES to handle optional end-dot and multiple dots.
  - Improved support of COPY with relative path /REPLACE OFF/COPY-REPLACING statements.
  - Added support for COL as valid field name.
  - Improved parsing to handle spaces after SPECIAL-NAMES, X-COMPUTER and DECLARATIVES.
  - Added support for S01 to S05 and C01 to C12 as valid paragraph names.
  - Added support for LENG function as alias of LENGTH function.
  - Improved Copybooks parsing when cobolMaxCodeWidth configuration value differs from their COBOL includer
  - Improved support for parsing expression with comma and operation e.g. MIN(A * B, (3 + 1) / 2, 3 + 4).
  - Added support for IS INITIAL PROGRAM in PROGRAM ID clause.
  - Added support for Level 88 condition on SEARCH ALL statement.
  - Added support for REPORT SECTION.
- SQL
  - Added support for DDL transformation from IBM IMS database.

AS400

Improvements

- CL
  - Enhanced support of CL by setting the transformation to JAVA as the default support
  - Improved the transformation to groovy script by adding the use of a metamodel
  - Added keywords CONTINUE and FOR to list of reserved keywords
- DDS
  - Improved support of DSPF fields defined relatively to the prior one
  - Improved support for DDS character field with keyword VARLEN
  - Improved transformation/generation of LF entity DAOs by generating the explicit setting of renamed fields in the convert and updateJPAFromVo methods.
- RPG
  - Improved support of PREFIX keyword parameter to handle cases where values don't have quotes
  - Improved handling of nested method calls in array indices
- COBOL400
  - Improved the generation of record entities imported from a DSPF with directive COPY DDS-ALL-FORMATS
- SQL
  - Improved the transformation of SQL WHERE clauses with non-enclosed OR conditions.

Release notes 4.7.0

Release date: March 12, 2025

This release of AWS Blu Age Runtime and Transformation Engines introduces key updates to core dependencies: AWS Blu Age Runtime has been upgraded to use the latest Spring Boot version 3.4.2, and modernized frontend applications now utilize the latest Angular version 19.

We've tested this version of the AWS Blu Age Runtime with the following stack. Other versions might also be compatible.

Component	Version tested

Java	Java 17
Presentation layer	Node JS 22.11.0
	Npm 10.9.0
	Angular 19.1.3
Service layer	Spring Boot 3.4.2
	Spring Core 6.2.2
	Spring Session 3.4.1
Persistence layer	Spring statemachine 4.0.0
	PostgreSQL engine 14
Oracle 21c	
Application server	Apache Tomcat 10.1.17

Runtime release 4.7.0

zOS

New features

- Added new YML property `dataSimplifier.doubleFloatingType` to define which format to use when encoding and decoding the floating-point numbers - Allowed values are `IEEE_754` (default) / `HFP` (for Hexadecimal Floating-Point format)
- Added support of decimal scaling position P on the numeric PICTURE

Improvements

- COBOL
  - Improved support for INSPECT statement with AFTER INITIAL and BEFORE INITIAL clauses
  - Upgraded Numeric Edited Type support with optimized byte conversion enhancing overall data handling efficiency

  - Added support for FUNCTION MIN & MAX with alphanumeric comparison
  - Improved support for FUNCTION INTEGER for negative values
- PL/I
  - Enhanced support of the PUT statement with FBA/VBA SYSPRINT dataset
- JCL - SORT
  - Added support of the OUTREC option SQZ with the subparameters SHIFT, LEAD, TRAIL, MID, PREBLANK, and PAIR
  - Enhanced cataloging DCB attributes for the SORTOUT file
- JCL - DSNUTILB
  - Enhanced support for DECIMAL EXTERNAL data type
  - Improved support for the NULLIF condition related to another column's values/position
  - Improved support for BYTEA columns
- JCL - INFUTILB
  - Improved support for keywords CURRENT [DATE/TIME/TIMESTAMP] in UNLOAD operation
- JCL - Improved support for qualified return code of called PROCs
- Data Processing
  - Enhanced numeric comparison logic when processing blank-valued fields redefined from alphanumeric to numeric types, ensuring correct evaluation against zero
- Blusam
  - Improved support for START command to handle partial key searches using segments of the primary key, providing more flexible record retrieval capabilities
- SQL
  - Added support for input parameter passed in ON clause
  - Added support for REPEAT function
  - Improved support for UPDATE statement with an assignment involving column name
  - Improved SQLCODE and SQLSTATE handling in multi-row cursor fetches
  - Added support for DB2 XML function for DB2 database target
- Message Queue Integration
  - Enhanced error handling in MQ GET operations, providing clearer status indication MQCC_FAILED for empty message bodies

- Enhanced support for MQ PUT operations by handling MQRFH2 header in XA transaction context
- Improved support for AWS Blu Age Runtime secrets to handle additional properties for JMS MQ
- Enhanced JMS MQ connection pool configuration capacity

- Misc
  - System Integration - Added support for the schema environment in DFSRRC00 program calls
  - Added compatibility for IMS transaction when Blusam is disabled
  - Improved support for end-of-file condition on a sequential file to align to legacy rule 'EOF is considered an unsuccessful read'

AS400

New features

- Added support for RUNSQLSTM command to execute SQL queries stored in an external source file

Improvements

- CL
  - Added file locking mechanism through the ALCOBJ/DLCOBJ commands
  - Improved the CPYF command functionality to maintain record sequence integrity during file copying
  - Improved CPYF command to support output as a flat file
  - Enhanced parameter handling to automatically adjust data length during program calls
  - Improved parameter support to handle range reference
  - Added support for MONMSG CPF2105 on DLTDTAARA command
  - Added support of the QTEMP library for command CHKOBJ on DATAAREA
  - Enhanced support for the ADDLFM command, improving logical file member management capabilities
  - Added support for generic data area names on DLTDTAARA command
  - Added support to SBMJOB to handle passing switches via SWS parameter

  - Improved the data area writing logic by putting the read and write calls into a synchronized block
  - Enhanced data area concurrency management to prevent simultaneous writes through the implementation of a granular locking mechanism, eliminating errors during parallel operations
  - Added support for keyword *ALL in DTAARA parameter for RTVDTAARA command
- Screen
  - Improved the color of a white attribute in the front end
- RPG
  - Improved support for TESTB operation to better handle different encodings
  - Improved support for EDITC operation with edit code 'Y' and 'P'
  - Improved support for EDITCODE for codes K, Q and Z
  - Implement support for Prototype ExtProc keyword support on local procedure with parameters
  - Handled parameters passed by value for external procedures
- COBOL
  - Improved SORT statement to handle targeted library
  - Truncated a partition when the associated DAO is opened in OUTPUT mode
  - Added support for PREFIX keyword
- DataQueue
  - Added keystore and truststore configuration options for RabbitMQ data and hybrid message queues, enabling secure messaging in production deployments
  - Optimized server restart performance by eliminating redundant data queue existence check
  - Streamlined architecture by removing DataQueueRegistry and DataQueueConnectionProvider components
  - Simplified data queue deletion through direct handling without registry validation
  - Improved queue creation process by ensuring proper exchange creation and queue-to-exchange binding
- Misc
  - Added transaction request size validation to prevent system overload related to excessive range requests
  - Improved lock management on job resource cleaning

- Improved the DAO locking mechanism to prevent locking a record when the file is opened in INPUT mode
- Optimized performance for dynamic entities mapping
- Improved handling of JPA persistence context to enhance performance on DAO operations
- Improved data area support by adding timestamp columns to handle record metadata
- Improved data area reliability by implementing synchronized read-write operations to prevent concurrent access issues
- Improved support for DateHelper#moveDate to support date-to-timestamp conversions

- DDS
  - Improved support of Select/Omit conditions combined with synthetic operation such as SST or CONCAT
- SQL
  - Optimized the SQLExecutorBuilder to handle partial record fetching more efficiently, particularly when retrieving multiple rows with fetch limits exceeding one

AWS Blu Age Transformation Engine 4.7.0

zOS

Improvements

- COBOL
  - Added support for CODE-SET statement
  - Added support for option NOT END OF PAGE from WRITE statement
  - Added support for a new syntax on ALTER
  - Added support of multiple line replacement for COPY REPLACING statement
  - Improved parsing of CHARACTERS keyword implied at different clause definition
  - Improved support for alternate keys involved in input-output section when the key's names are duplicated but have different parents
  - Improved support for IF condition with comparison of large fields against spaces
  - Improved logical file reference static resolution by making it case insensitive
  - Enhanced code transformation to automatically generate LINAGE-COUNTER implicit fields for each FD entry that contains a LINAGE clause

- Improved support for IDCAMS Cluster definition with a name enclosed by simple quotes
- Improved support of PERFORM statement for Basic format, with TIMES phrase format and format involving section's qualification
- Enabled the transformation of SORT statement with multiple GIVING files and added an exception to track this case on execution
- Improved the transformation of RENAMES when specified on a group using the REDEFINES clause

- PL/1
  - Added Support for option DATA or LIST for PUT STRING
  - Added support for Multiple Entry program support

AS400

Improvements

- CL
  - Improved the parsing of parameters for QCMDEXC to be able to execute ALCOBJ/DLCOBJ through QCMDEXC
  - Added support for ElseIf statement on CL
  - Added support for builtin function %SWITCH to set the job switches.
- DDS
  - Improved support of program-described DSPF record to handle dummy record in input specifications
  - Improved the DAO generation WHERE clause conditions in the case of an CREATE INDEX SQLDDS file
- RPG
  - Added support for ZEROES initialization INZ(*ZEROS) on Unsigned Integer
  - Improved support of command EXCEPT with a file type of Workstation and SQL Index
  - Improved support of RPG internal data type for numeric fields defined externally
  - Improved support of built-in %ERROR to get the current error status after an operation on a file
  - Improved support of COMMIT file keyword to handle multi-format logical file
  - Enhanced SELECT/OMIT processing for multi-format logical files

- Improved handling MOVEA for setting array with blanks, moving Fixed arrays of Packed, Zoned and Binary to another array of the same type.
- Improved array handling for Z-ADD *ZEROS and MOVEA *ZEROS
- Improved MOVE/MOVEL operations when moving from numeric and character combinations
- Implement support for RPGLE Prototype ExtProc keyword support on local procedure with parameters
- Improved handling of *HIVAL figurative constant in assignment statements
- Improved the support for PREFIX keyword to handle character replacement

- COBOL400
  - Improved support for REWRITE and UPDATE operation on Dynamic access mode
  - Added support for ACCEPT Statement FOR clause in CBLLE
  - Improved support of built-in %STATUS to get the current status of a file after an operation
- Misc
  - Improved transformation of programs which define data structures that share the same name as one of the fields in the specified PF/LF
  - Improved array access generation on kanji variables

Release notes 4.6.0

Release date: January 24, 2025

We've tested this version of the AWS Blu Age Runtime with the following stack. Other versions might also be compatible.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 22.11.0
	Npm 10.9.0
	Angular 18
Service layer	Spring Boot 3.3.5

	Spring Core 6.1.14
	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 14
	Oracle 21c
Application server	Apache Tomcat 10.1.17

Runtime release 4.6.0

zOS

Improvements

- COBOL
  - Enhanced `WRITE ADVANCING` capabilities with improved accuracy for sequential file line writing, supporting multiple contexts (`BEFORE>`, `AFTER`, and Implicit uses) and full `PAGE` statement implementation
  - Enhanced support for `FILLER` for cases when a nested table FILLER is used as a group with a table as child
  - Improved access to child of ambiguous parents inside one segment
  - Added support for Numeric Edited type with picture='-----'
  - Improved display handling of BINARY type data
- PL/I
  - Improved conversion of binary literal values in assignment statements
- JCL — SORT
  - Improved support for consecutive `OVERLAY` parameters in the same `OUTFIL` statement
- JCL — DSNUTILB
  - Optimized loading mechanisms, resulting in 25% faster data retrieval times
  - Enhanced support for XA transactions for external business data-sources
- JCL — INFUTILB
  - UNLOAD - Added support of FLOAT8 data type
- JCL — IDCAMS

  - Optimized handling of return codes for IDCAMS commands
  - Added support to delete all GDG generations based on GDG base name
  - Added support for files deletion without NONVSAM parameter
- JCL — Misc
  - Enhanced Batch Restart Metadata handling to improve workflow status management during restart mode
- Blusam
  - Added support of TTL for Blusam cache in both Ehcache and Redis implementations
  - Improved support for DEPENDING ON field on COBOL File Description FD for Blusam KSDS file
  - Enhanced thread safety in Redis Blusam read operations for simultaneous multi-job execution
  - Improved Blusam schema creation for better robustness regarding database user privileges
  - Improved padding to the right on variable block concatenated input dataset READ
- BAC
  - Added support for dataset creation in Multi-schema mode, including a new "Schema" column to indicate schema association for each dataset
- MFS
  - Improved propagation of user info from Front-end to shared context, ensuring proper propagation to JHDB context
  - Added support for IBM MQ IMS information header on XA transactions
- SQL
  - Enhanced SQLCODE handling to set 305 during cursor fetch when all column values are NULL
  - Added support for IN clause involving OCCURS parameter for WHERE conditions
  - Added support for DECLARE GLOBAL temporary table statements
  - Extended DB2 SQL support for midnight 24H DB2 specific timestamp format thru dedicated conversions at execution according targeted database engine
- Misc
  - Enhanced IBM930 charset to allow both Unicode characters U+2014 and U+2015 to correspond to x'44x4A' in EBCDIC
  - TDQUEUE - Refactored SQS implementation to support multi-threading
  - Improved GDG dataset name resolution to allow customer to archive files with the same GDG prefix (e.g. A.B.C.G0002V00 is current file and A.B.C.G0001V00.1236 is an archive file)

- Enhanced `SQLConverter::toPgmDate/Time/Timestamp` to align the date computation according to the legacy format

AS400

New features

- Added support for dynamically created AS400 tables for flat files and duplicated entities, enabling access to tables created via CL commands like CRTPF, CRTDUPOBJ, and CPYF
- Added a service to support library list through a registry that handles the default library for every tables

Improvements

- CL
  - CLRPFM - Improved handling of member when the command is called for QTEMP library
  - SMBJOB - Improved support of parameter `PARM` to handle dynamically built argument
  - CPYFRMIMPF - Added support for parameter `TIMFMT`, `ERRRCDFILE`, and `ERRRCDOPT`
  - CPYFRMIMPF - Improved support of database alphanumeric values that contain single quotes
  - CPYF - Refined the command query construction for multi-member `FROM` files with `TOMBR(*ALL)`
  - CPYF - Enhanced support to handle `FMTOPT` parameter for `MAP DROP`
  - CPYTOIMPF - Improved support of parameter `FROMFILE` to handle `MEMBER` of the table
  - RTVUSRPRF - Added support for parameter `RTNUSRPRF`
  - DSPDBR - Overhaul the command to match expected legacy behavior of printing out information about views the exist on a table, as well as the library and member they're a part of
  - DSPFD - Improved support of parameter `FILE`
  - DSPFD - Improved support of parameter `TYPE MBR` output to include additional values: mbfile, mblib, mbfcdt, mfccn
- Screen
  - Improved cursor position priority for `DSPATR(PC)`

- Improved the validation of subfile record fields by ignoring the front-end validation of “protected” fields
- Improved support for initializing records in workstation with multiple array fields sharing component names
- Enhanced support for response indicators in DSPF keywords (SFLMSG, SFLMSGID, CHANGE and command keys)

- RPG
  - Enhanced program cycle support for a better handling of the fields read from primary/secondary files
  - Added support for Split Control Field for primary/secondary files reading
  - Enhanced %SUBST built-in method to handle double-byte fields in compare statements
  - Improved support of ZERO indicator for MVR operation
- DDS
  - Added support of multi-format logical files with record format that refer to the same physical record
- DataQueue
  - Improved job interruption handling for jobs waiting on data queue messages by cleaning up the consumer during interrupts
  - Migrated from RabbitMQ to Spring-AMQP for better channel management and thread scaling
- Misc
  - Improved SQLExecutorBuilder to support queries with multiple white spaces and open braces without leading spaces
  - Improved DAO support to handle correctly the cursor positioning while changing the reading direction
  - Refined key initialization after retrieve and delete operations to ensure proper removal of related records before inserting updated records
  - Optimized DAO mapper generated code to improve time execution performances

AWS Blu Age Transformation Engine 4.6.0

zOS

Improvements

- COBOL
  - Improved parsing of RESERVE clause with optional AREA/AREAS literal
  - Enhanced COBOL support with optional DATA DIVISION declaration, supporting streamlined test cases
  - Improved special names paragraph by adding support for ALPHABET, SYMBOLIC, and CLASS clauses, switches, and FORMFEED variable
  - Added support for SYSIN as a Mnemonic Name in ACCEPT statements
  - Enhanced PICTURE clause support for "$", "0", "CR", "DB" symbols in PIC logical size calculations
  - Improved USE statement transformation for multiple file scenarios
  - Enhanced ALTER statement transformation for multiple alterations
  - Added support for figurative constants ZERO HIGH-VALUE LOW-VALUES in delimited by clause
- SQL
  - Improved transformation of default value for postgreSQL target to handle quotes around the CURRENT_TIMESTAMP default value
  - Handle WITH CHECK OPTION clause of SQL views

AS400

Improvements

- DDS
  - Improved support of multi-format logical files that refer to the same physical record multiple times
- RPG
  - Enhanced MOVE and MOVEL operations to better handle padding zeros
  - Enhanced handling of nested function calls in evaluations and conditions
- COBOL400
  - Added support for transforming the IN keyword in SELECT statements
  - Improved support for missing dots in data description entries, aligning with the latest COBOL version where dots are assumed when missing

- Enhanced cursor positioning on `REWRITE` operations
- Enhanced support for `START` statement to lock the record at current file position
- Improved support for compiler directive `COPY DDS` to generate all input/output data structure

- Misc
  - StateMachines - Improved transformation to enhance composite states declaration in alignment with stateless4j paradigm
  - Improved sanitization for LF files containing special characters
  - Improved support of figurative `*ALL` with hexadecimal values
  - Improved `MOVE` operation support for implicit conversion from numeric to character types
  - Optimized report bean generation to sort by associated printer name, preventing duplicate or conflicting names
  - Improved support of keyword `EXTFILE` combined with `USROPN` to handle literal value and format `libname/filename`

Release notes 4.5.0

Release date: December 20, 2024

This release of AWS Blu Age Runtime and AWS Blu Age Transformation Engines includes the following key features.

- **JCL support** — It is now possible to generate and execute JCL scripts on the fly within the runtime context. This feature adds flexibility and automation in batch job processing. We've updated the support for JCL utilities in the runtime, with a set of improvements to SORT, ICETOOL, INFUTILB, and IDCAMS (see details in the following sections). These enhancements offer more robust and efficient data processing capabilities.
- **Binding Directories and Activation Groups Support for AS/400 Modernized Applications** — Binding Directories enhance system organization by managing exported procedure references, while Activation Groups streamline execution context management. These features improve precision and reliability, robust resource management, and optimized system interactions. The result is a more resilient, organized, and efficient system for modernized AS400 applications.
- **Dependencies updates:** — Update of all frontend frameworks (BAC/JAC & modernized applications) to the long-term support (LTS) versions. The update of Angular from v17 to v18 introduces a new reactivity model and streamlined state management, reducing complexity and

improving application maintenance for developers. Node.JS has also been updated from v20 to v22.

We've tested this version of the AWS Blu Age Runtime with the following stack. Other versions might also be compatible.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 22.11.0
	Npm 10.9.0
	Angular 18
Service layer	Spring Boot 3.3.5
	Spring Core 6.1.14
	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 14
	Oracle 21c
Application server	Apache Tomcat 10.1.17

Runtime release 4.5.0

zOS

New features

- JCL — Added ability to invoke a batch job from online programs. We added a service to handle JCL script stored in a dedicated TDQueue when a modernized program generates it on the fly. This service makes it possible to reconstruct the JCL message, refactors this message into a groovy script, and runs this groovy script.
- ADABAS — Added support for ADABAS program. With this support, the runtime emulates the ADABAS commands for database access (available for Oracle only).

Improvements

- COBOL
  - Enhanced support of DISPLAY statement leveraging the NO ADVANCING option
  - Increased accuracy in management of currency signs allowing the user to benefit from a more accurate transformed COBOL structure
  - Improved support for value assignment when moving an unsigned field to a signed field and vice versa
  - Improved support for block size for GDG files and concatenated files
- CICS
  - Added support for OpenStatus and EnableStatus of Blusam datasets
  - Added support for the `SET DATASET` command
- JCL — SORT
  - Improved handling of data set record size
  - Improved support for the `OUTFIL` statement to produce output files containing only the records from the input files according values specified in STARTREC and `ENDREC` options
  - Enhanced support of `OVERLAY` statements
  - Improved support for the `OUTREC` statement to handle a variant of the `EDIT` option. We now support `EDIT(...)` in addition to `EDIT=(...)`
  - Added support for the pattern(p,m,f,OPERATOR,p2,m2,f2) in arithmetical operations
  - You can use the `DUMMY` file clause of the `SORT` program from a JCL to handle empty input files and benefit from empty file generation
- JCL — ICETOOL
  - Enhanced support for the `SORT FIELDS=COPY` statement through the `SORT` program
- JCL — INFUTILB
  - Improved support for record size computing if it is not specified in the JCL and the property DFSIGDCB is disabled
  - Improved UNLOAD with INTO clause for DECIMAL by updating the precision and scale according to the into clause fields
  - Improved format method in VarcharFormatter
  - Enhanced support with a new configurable option that allows users to control how VARCHAR fields are handled during data unloading as regards as padding behavior ensuring flexibility and accuracy in data extraction processes.

- JCL — IDCAMS
  - Improved deletion for file with wildcard suffix and name defined either directly either enclosed by parenthesis or by simple quotes
  - Improved accuracy to leverage MAXCC return code
- JCL — IKJEFT01 - Added feature flag `systin.encoding` (default = `ASCII`) to support specific encoding for SYSTSIN file dataset
- JCL — Improved support for BDW property for a generated output file in a JCL step and subsequent steps use the same file system as input and `DISP=PASS`
- MF
  - Improved support for 2-byte header for Record Sequential file
  - Improved handling of return codes for the DELETE command
  - Improved Write Advancing Line for Record Sequential file
- Redis
  - Improved Redis template initialization for JCL checkpoints and Jics TSQueues
  - Improved Redis dataset record lock information accessibility and readability
- SQL
  - Improved parsing of FOREIGN KEY with REFERENCES clause
  - Provided an extendable caching feature to store original legacy graphic types in the database, enhancing data traceability and facilitate graphic compute
  - Enhanced parsing support of CASE WHEN pattern of SQL queries across runtime utilities
  - Improved SQL Postgres Blu Age built-in function gwdecimal the runtime relies on to fit with DECIMAL DB2 built-in function.
- Misc
  - Enhanced support of NumericEditedType using SIGN operand
  - Improved primary datasource configuration generation in SpringBootLauncher in the modernized application
  - Enhanced flexibility to segregate application logs from path related to the called job.
  - Improved support for Blank value in comparing fields from NumberUtils
- FILE — Improved support of variable blocks data sets in the underlying files
- MQ — Improved MQ connection management for high availability environment ready
- Improved MQ Queue compatibility by adding support for non-JMS clients to enhance encoding and character set handling

- Improved support for ANSI Control characters for Ebcdic file

AS400

New features

- Added support for Exported Data within Bound Programs
- Added ILE specific support for the division by zero

Improvements

- COBOL400
  - Improved support of EOF in File Status
  - Increase the precision support of Cobol START statement to support EQUAL keyword into KEY IS clause
- CL
  - Added support for command UPDENVPARM
  - CRTPF - Added support for table accessed with a partition
  - RCVF - Improved support of logical files with override
  - FTP - Improved support of logical I/O files with OVRDBF and enhanced OUTPUT log and added support for I/O files in the working directory
  - CPYFRMIMPF - Added support for parameters `ERRRCDFILE`, `TIMFMT`, `ERRRCDOPT`
  - CPYF - Improved QTEMP partition creation
  - CPYF - Added monitoring message when the *FROM file is empty
  - OVRPRTF - Added support for new parameters: `PAGESIZE`, `OUTQ`, `DEV`, `LIP`, `CPI`, `OVRFLOW`, `LVLCHK`, `FORMTYPE`, `HOLD`
  - Increased accuracy when using the `FMTOPT` parameter with `MAP` *and* `DROP` options in the `CPYF` command to allow copying data from a source file with extra columns to a target file
  - Increased accuracy in managing the mapping of file system path wildcard patterns in the `RMVLNK` command
  - The `RMVM` (Remove Virtual Machine) command has been enhanced to handle `DROP` partition tables ensuring complete cleanup of related resources.
  - OPNQRYF - Improved support of parameter *FILE for command

  - Implemented CPF0000 handling to encompass all CPFx messages
  - CHGDTAARA - Added support for *ALL keyword to change the entire data area
- Screen
  - Improved tables/subfile displaying by increasing accuracy for scrolling and position/priority of cursor
  - Improved `CHECK(RZ)` and `CHECK(RB)` functionality for non-numeric and non-signed fields
  - Improved support of help screen feature for keyword `HLPARA`
- RPG
  - Improved support of built-in `%SubDt`
  - Improved support for procedure using a local data structure which is externally-described
  - Added support for optional error code parameter `QMHSNDPM`, `QMHRMVPM` , and `QMHRCVPM`
  - Enhanced support of `%SUBST` built-in method to better handle double bytes fields.
  - Added support for built-in %TLOOKUP and its variants (%TLOOKUPGE, %TLOOKUPGT, %TLOOKUPLE, %TLOOKUPLT)
- Dataarea
  - Improved support for OUT operation when factor1 is blank
  - Improved concurrent reads on the same data area
  - Added configuration variable `blu4iv.dtaara.library.disable` to disable libraries for data area
  - Extended support to leverage named libraries thru data area operations enabling user to structure data area location as he wish.
- DataQueue
  - Improved RabbitMQ channel usage
  - Improved RabbitMQ Consumer to only attempt to cancel the consumer once
  - Improved data queue retrieve from RabbitMQ by only attempting basicGet when wait time is 0
- Misc
  - User Space - Improved behavior when multiple jobs attempt to retrieve the same userspace simultaneously
  - Improved support of uncommitted record deletion under commitment control
  - Entity - Improved support for consecutive omits as OMIT carries implicit `AND` meaning

- Added support for camel case in entities mappers setters to handle customs named defined through additional refactoring
- Improved propagation of user information from AS400 environment transactions thru whole application.
- Improved accuracy when ending a job scheduled by Quartz in case of interruption case.
- Improved Commitment Control support to make it program-scope

AWS Blu Age Transformation Engine 4.5.0

zOS

Improvements

- JCL - Improved groovy generation for KSDS dataset based on LISTCAT parsing
- COBOL
  - Improved parsing of `COPY-REPLACING` statement to handle replacement of qualified subfield when ambiguity for this subfield name is present
  - Improved support for `SYSOUT` defined in `SPECIAL-NAMES` statement
  - Improved support of figurative ZEROES in `ADD n TO ZERO` statement
  - Improved support for `REPLACE` statement to handle multi-line issue by flattening multi-line keys and text blocks
  - Improved support for arithmetic operations ADD/SUBTRACT/MULTIPLY/DIVIDE with `GIVING` clause
  - Initiated parsing support of REPORT SECTION and its related actions (INITIATE, TERMINATE, GENERATE report)
- Misc - Improve weather report generation and robustness

AS400

Improvements

- DDS
  - Improved support of implicit length of type DATE
  - Improved support of stop-zero-suppression character on keyword EDITWORD

  - Improved support of column name DESC as it is a reserved word in DB
- RPG
  - Improved support of built-in %TIME
  - Improved generation of EVALR statements to handle assignment from a string value to a variable of shorter length with a better right-adjustement
  - Enhanced SQL parsing around options setting
  - Improved support for PSDS initialization in NOMAIN RPGLE programs
  - Improved support of keyword LIKE to define a DDS numeric field as Packed, no matter its external description
  - Improved file name sanitizing by replacing “$” by “DL”
  - Improved support of built-in %SUBST to handle double byte values
- COBOL400
  - Screen - Improved support of DSPF record around I/O operations
- CL
  - Improved renaming of reserved variable names
  - Improved support of Select/Omits conditions to handle multiple formats file
- Misc
  - Reduced duplicated entities around file operations (EOF, FOUND, EQUAL)
  - Improved generation of JRXML files for QPRINT, a standard printer on AS/400. When it is used, the created JSON file will not contain any reference to the program or the file. Only one JRXML file is generated (QPRINT-QPRINT.jrxml)
  - Improved the display of additional message information for components displaying messages from program queue

Release notes 4.4.0

Release date: November 13, 2024

This release of AWS Blu Age Runtime and Transformation Engines focuses on upgrading critical dependencies and supported technologies while boosting performance in multiple functionalities. Some key features and changes in this release include:

- **Dependencies updates**: Console applications (BAC and JAC), and modernized applications are now running on Bootstrap 5. The AWS Blu Age Runtime is now powered by Spring Boot 3.3.5 framework.
- **Performance**: Improved the performance of the state machines execution (up to 10x faster), thanks to a new implementation that overcomes a performance degradation after upgrading the Spring State Machine library from version 2.5.1 to 4.0.0. This upgrade was not optional as 2.5.1 version was no more maintained and contains Critical and High CVEs. It includes a runtime state machine implementation on the platform to a new library, with a lightweight and efficient state machine implementation, free of CVE, and with better overall performance.
- **Simplification of database access**: Completed a significant overhaul of the components used to access the database, including DAOs, JPA entities, DDS DataSimplifier entities, and Mappers. This redesign was driven by the need to provide better support for the OVRDBF (Override Database File) feature common in AS400 projects. It allows to handle more cases with a simplified architecture for the generated code.

We tested this version of the AWS Blu Age Runtime with the following stack. Other component versions might also be compatible.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 18.18
	Npm 9.8
	Angular 17
Service layer	Spring Boot 3.3.5
	Spring Core 6.1.14
	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 14
	Oracle 21c
Application server	Apache Tomcat 10.1.17

For more information about the changes included in this release, see the following sections.

Runtime release 4.4.0

zOS

New features

- COBOL - Added support for JSON GENERATE statement
- COBOL - Added support for control blocks
- MF - Added support for FCDREG compiler directive
- Blusam - Added feature VSAM file-sets with an implementation based on database schema - Only PostgresSQL supported
- Blusam - Added support for handling TTL (Time to live) for Blusam cached data items (Redis cache engine)
- JCL - IDCAMS - Added new property `idcams.encoding.forced` to force charset used to decode SYSIN card
- JICS - Extended the `jics.db.dataScriptLocation` property from `application-main.yml` to accept a list of file and folder paths. The order of the list matters. The first SQL file is executed first and so on. When a folder is executed, the SQL scripts it contains are executed in no definite order.
- Added support of CEE3ABD utility

Improvements

- Blusam - Improved loading time and memory footprint from legacy large data sets to Blusam for customers using PostgreSQL engine (we observed up to an 8-fold increase in loading speed for large data sets)
- Blusam - Improved exportDataSetToS3 API with Credentials Support
- Blusam - Improved LISTCAT uploading files for data sets creation
- Blusam - Improved support for Dynamic READ using explicit KEY
- Blusam - Improved the write-behind mechanism logic
- JCL - Enhanced JES support to improve file locking in parallel execution
- JCL - Added support for statement `INCLUDE MEMBER`

- JCL - DNSUTILB - Improved support for duplicate key to handle special case when primary key contains spaces
- JCL - DSNUTILB - Improved LoadTask to optimize performance when loading GRAPHIC data
- JCL - INFUTILB - Added support for `fetchsize` when `chunksize`is not defined
- JCL - INFUTILB - Improved support to query returning empty resultset
- JCL - INFUTILB - Improved robustness when processing data in CHUNK
- JCL - INFUTILB - Improved support for unload with nullable field
- JCL - INFUTILB - Improved support for numeric type
- JCL - INFUTILB - Improved unload for Nullable Field
- JCL - SORT - Improved support for OUTREC syntax
- JCL - SORT - Improved parsing of statement DATE1
- JCL - SORT - Improved support of clause INREC PARSE with RDW
- JCL - SORT - Improved fields formatting using editing masks
- JCL - SORT - Improved support of 'SubString' in OUTREC
- JCL - SORT - Improved support for MF compatible CARD
- JCL - UNLOAD - Improved support of field size with Postgresql
- JCL - IDCAMS - Improved performance for File Loading VSAM data set by introducing bulk mode
- PL/1 - Enhances support for NumericEditedType formatting to prevent scale discrepancies
- IMS - Improved support for IMS Database _right column in NodeSorter
- CICS - Improved command `RECEIVE MAP` with `SET` and no `INTO`
- BMS - Improved support of field initial value
- SQL - Improved DateTimeFormat parsing for `ddMMMyy` pattern
- COBOL - Improved support for NumericEditedType value when decimal point not considered while getting value
- Improved support for reading variable-length field in line-sequential file
- Improved support for record size inheritance from dataset catalog for GDG files
- Improved support for printing report by allowing customizable advancing lines
- Improved initialization of record data for Variable Block (VB) Files

GS21

New features

- Screen - Added support for PSAM files
- Screen - Added support for ATTR2
- Added support for AIM (Advanced Information Manager) ecosystem.
- Added PED support in AIM

Improvements

- Improved BitUtils signatures to handle RangeReference
- Improved support for DummyFileConfiguration to add recordSize / rdw / bdw / blksize / blkszlim attributes
- Improved support for VPOINT statement to handle the case of a record not found
- Added robustness when accessing record byte array
- Improved JEF charset character mapping
- Improved support for handling arrays and conditions in JDBC mapping
- Improved support for SQL requests in the different NDB statements, better handling the variations of SQL syntaxes using constants for each part of an SQL query.
- Improved support for GS21 PackedType last nibble to be C, D or F for numeric validation
- Screen - Improved support for ACSAPI and DefaultPsamController for SPA and ENTER
- Screen - Enhanced support of ACSAPI and NDB verbs

AS400

New features

- Added support for Multi-record format Database files
- Redesigned the AS400 Database access framework
  - Enhanced capabilities around file overriding
  - Removed obsolete components and reduce complexity
  - Streamlined the code generated from legacy programs
  - Integrated DAOCycleManager component into the Blu4iv plugin, enabling us to leverage the AS400-specific features of our custom runtime.
- JOB - Improved support for job management (Quartz) to add the ability to interrupt a job/group of jobs. Added a REST API endpoint to interrupt a job with the specified execution id (unique to

each job since it is a primary key). Upon successful interruption, the runtime update the status of the job to “INTERRUPTED”.

- Added support for utility program CEERAN0
- Added support for passive mode. Added the YAML `configuration gapwalk-application.cl:ftpservice:passive` to enable passive mode
- Added feature to create QTEMP sessions and delay QTEMP cleaning
- Added support for compilation feature BNDDIR to define explicit dependencies between programs
- Added support for Activation Groups mechanism

Improvements

- CL - Improved RMVMSG command on program message queue to handle *PREV keyword
- CL - Improved support for overrides in OPNQRYF
- CL - Added support for the MSGLEN and SECLVLLEN parameters for the RTVMSG command
- CL - Improved support for CRTDUPOBJ to manage case when NEWOBJ is not passed and added support for generic table names
- CL - Improved support of FTP to handle parameters GET, RMTSYS and BINARY
- CL - Improved CLRPFM query performance and added an option to use TRUNCATE instead of DELETE
- CL - Improved SBMJOB to properly handle USER parameter to use it as the USER when a job is submitted
- CL - Improved DLTOVR command support to handle the case of *ALL
- Data Area - Improved support for Blu4DataArea by adding logging for Exception handling
- Data Area - Improved support for Blu4DataArea to fetch a new DataAreaDao instance for every thread
- Data Area - Improved data area locks, avoiding locks at record level and instead using the newly implemented locking mechanism
- Data Area - Data Area write operation now continues with execution when a lock isn't acquired and an error indicator is provided
- Report - Improved support for report output path/naming convention for the printed reports. Allowed customers to customize the report output path and the name as well. The customer can specify their own path and naming convention without impacting any other project.

- JOB - Improved support for job management (Quartz) to update job status in the event of abnormal job termination. e.g.: ‘Shutdown’ or ‘abnormal shutdown’ of Tomcat
- Screen - Improved handling of numerical value on field with edit word with minus
- Screen - Improved rendering popup with only titleColorTop
- Screen - Improved support for help information retrieving to handle case when general help item is not found
- Screen - Improved displaying the 'additional information' screen when pressing F1 on subfile message line
- Screen - Improved display of message line footers for SFLMSG
- Screen - Improved Front End to remove a Record in its entirety when a new record overlaps it
- Queuing - Improved RabbitMQ message retrieval to consume fewer resources
- Queuing - Improved RabbitMQ Data Queue implementation to only retrieve one message at a time.
- SQL - Improved SQLExecutorBuilder handling of SQLCODE for dynamic CREATE and DROP table queries
- SQL - Improved support of OVRDBF on query
- SQL - Improved SQLExecutorBuilder so that OVRDBF overrides are applied to prepared statements
- RPG - Improved support for Input and Output specifications of Program described Disk files
- RPG - Improved support for Primary and Secondary File Read with MR (Matching Records) indicator. The retrieve order of a DAO Cycle with match fields has been improved.
- RPG - Improved support for Primary and Secondary files. Improvement on the update Primary files and output Secondary files update/write code generation.
- RPG - Added support for RETURN statement in free-form format
- RPG - Improved transformation and runtime handling of numeric decimal assignments,
- RPG - Improved generation of binary variables
- RPG - Improved support for EDITC
- RPG - Improved handling of local data area
- Improved support of DDS fields shared by multiple device type (DISK, WORKSATION, PRINTER)
- Improved override handling so that overrides on PFs will no longer affect LFs
- Improved Blu4ivWebController not to re-set the username and userid to default values
- Improved index adjusting during record reads when the read direction changes

- Improved cursor placement on record reads after update/delete operations
- Improved support of reading on a multi-entities DAO when the reading direction changes
- Improved support for User Spaces to avoid instance to be reused by all threads instead of each thread having its own instance
- Improved support of multi-threading concurrent access on record read
- Improved the storing of the username/userid in SharedContext through YML configuration
- Improved Locked Records Release with Updated Values
- Added support for OPM compiler specific behavior for NEXT SENTENCE statement

Transversal capabilities

New features

- Added new metadata.ini property `legacy.compilerto` specify the legacy compiler of the artifacts to transform. The support of some COBOL statements, like NEXT SENTENCE, is different depending on the value you set.
  - "ZOS" for a z/OS legacy system.
  - "ILE" or "OPM" for AS400 system. Default = "ILE" when `legacy.system` = "as400"

Improvements

- Front-End - Redesigned the screen field components to expand the range of supported field types. This enhancement enables the runtime to accommodate a wider variety of user input and data requirements involved in AS400.
- Improved method `isValid()` for separate sign byte on ZonedType
- Improved support for `StringConcatenationBuilder::withPointer` for concatenation involving CRLF
- Improved support for specific double bytes encoding to make them thread-safe
- Improved state machine performances by integrating a new framework
- Improved algorithm for assignment optimization to prevent unexpected rewriting

AWS Blu Age Transformation Engine 4.4.0

zOS

Improvements

- LISTCAT - Improved parser to prevent duplicate entries
- LISTCAT - Improved support of ESDS to file system in JCL / Groovy
- CICS - Improved support for LENGTH OF for CICS statements

AS400

Improvements

- DDS Record generation enhancement
  - Improved the support of DDS record to generate entities that correspond to the DDS record structure
  - Provided support for shared fields and mapping functions that match better with the legacy
  - Improved the handling of both externally-described and program-described files
- RPG - Improved RPG detection for module with only free form
- RPG - Improved support for COPY statement to ignore keyword `*LIBL/` as prefix to locate an application copybook
- RPG - PF - Improved support for input specification with physical records from pfile
- RPG - Added support of On-Exit statement
- RPG - Improved support of LikeRec keyword
- RPG - Improved mapping of renamed DSPF fields
- CL - Improved field name resolving
- COBOL - Improved support of conversion from hexadecimal to character
- Improved support for Decimal type generation
- Improved support of FIXME message for unsupported legacy code (display whole legacy line)
- Improved performance on AWS Transformation Engine (AS400 parsing step)
- Improved support of Keyword LikeRec to align it with File Specifications
- Improved support of built-in function %Diff

- Added support for special character Currency sign on DSPF label

Release notes 4.3.0

Release date: September 16, 2024

This release of AWS Blu Age Runtime and Modernization Tools focuses on extending the capabilities and coverage to modernize mainframe functionalities. Some key features and changes in this release include:

- **CICS**: Additional support to exchange data from the terminals, and run transactions with incoming data by supporting the SEND MAP command with Map Reference.
- **JCL**: New capability that allows to restart the most recent execution of a batch job from a previously failed JCL/PROC step, or trigger a delayed restart by bypassing previously executed steps. This provides greater control over batch processing using persisted step-level checkpoints.
- **AS400**: Additional library support, enhanced performance and robustness of commonly used commands such as CPYF, OVRDBF, SBMJOB, and OPNQRYF and many more.

We tested this version of the AWS Blu Age Runtime with the following stack. Other component versions might also be compatible.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 18.18
	Npm 9.8
	Angular 17
Service layer	Spring Boot 3.2.5
	Spring Core 6.1.5
	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 14

	Oracle 21c
Application server	Apache Tomcat 10.1.17

For more information about the changes included in this release, see the following sections.

Runtime release 4.3.0

zOS

New features

- CICS - Added Support for Map Reference in SEND MAP command
- CICS - Added support for RECEIVE command and support for running transaction with data from the `JicsTransactionRunner` screen
- Added support for IIH header for the JMS messages
- COBOL - Added support for multiple embedded spaces in Pseudo-text for REPLACING statement
- COBOL - Added support for JSON PARSE statement
- Blusam - Added support for KMS to feature “Export dataset”
- BAC - Added the configuration of `application-main.yaml` to define record size to filter loaded masks matching with this record size
- JCL - INFUTILB - Added support for the keyword INTO as part of BMC control statement
- GS21 - Added SOSI handling for JEF encoding
- GS21 - JCL - Added KDJBR14 as an alias of IEFBR14
- GS21 - JCL - Added KQCAMS as an alias of IDCAMS
- MF - Added support for COBOL MF Compatible File dependingOn field support
- MF - Added support for SORT mechanism for COBOL MF Compatible file
- MF - Added support for COBOL MF Compatible open non-optional missing file

Improvements

- JCL - DSNUTILB - Improved LOAD operation with ZONED DECIMAL Type
- JCL - DSNUTILB - Added support of duplicate key
- JCL - DSNUTILB - Added support for rollback mechanism on LOAD command

- JCL - INFUTILB - Improved UNLOAD with new properties FETCHSIZE and CHUNKSIZE
- JCL - IKJEFT1A - Enhanced SYSTSIN file reading by adding current charset
- JCL - DFSORT - Added support for option DATE4 & DATE5
- JCL - DFSORT - Added support for case of Variable Block type as input and Fixed Block type as output
- JCL - DFSORT - Added support for ALTSEQ
- JCL - Enhanced checkpoint metadata with job web identifier
- JCL - Improved Batch restart checkpoint purge for REDIS
- IMS - Implemented EXPRESS function for PURGE command
- IMS - Added support for options PCBNAME and LIST for PCB statement
- COBOL - Added support for GO TO statement without target
- CICS - Improved support for INTO statement with RecordAdaptable in READQ TS
- CICS - Improved support for INQUIRE TRANSACTION command
- CICS - Improved support for setBytes in READNEXT command
- CICS - Improved support for START command without CHANNEL option
- CICS - Added Support for Reference type for Inquire TSQueue
- CICS - Improved support for RECEIVE MAP command when map and mapset are Reference
- CICS - Improved support for options FROM and LENGTH for RECEIVE MAP command
- CICS - Added support of RecordAdaptable attribute
- CICS - Improved support for RECEIVE command to handle overflow
- CICS - Added support for slice rule in CICS statements
- CICS - Improved support for linkage structures DFHCOMMAREA and DFHEIBLK. The transformation engine supports more implicit definitions
- CICS - Added support for options START, NEXT and END for INQUIRE CONNECTION command
- CICS - Added support for both type ‘int’ and ‘reference’ for option LENGTH of RECEIVE command
- CICS - Enhanced support for parsing INQUIRE NETNAME command
- CICS - Added support for group name for JicsQueueBuilder
- Blusam - Added support for indexed file starting with generic key
- Blusam - Improved Blusam loaders
- BAC - Improved support for data synchronization in multi-instance environment when Redis is used to centralize cached values, including both actual data and locks

- BAC - Improved UI (style, logo, checkbox)
- BAC and JAC - Added the configuration of `application-main.yaml` to retrieve the username and the password of the default super admin user in the secret from AWS Secrets Manager by specifying the ARN
- BAC and JAC - Upgrading dependency to Bootstrap 5
- Improved JCL checkpoints and JICS TSQueues Redis template configuration
- Improved support for Size of Pointer depending on AMode
- Added support for zero comparison on NumericEditedType
- Enforced Slf4j MDC properties before logging
- Improved support to file reading to handled multiple empty line
- MF - Improved support for initializing pointer variables for COBOL MF compiler directive initPtr
- Redis - Improved feature GwFileLock on concurrence aspect through an implementation based on Redisson

AS400

New features

- CL - Added support for CHGPF command
- RPG - Added support for functions %HOURS, %MINUTES and %SECONDS
- COBOL - Added support of SORT file with Blu4IV DAO architecture

Improvements

- CL - Improved PgmClose to be registered as a program and accept a variety of objects for OPNID param
- CL - Refactored RTVMBRD to handle multiple libraries and members
- CL - Added support for TOLIB param on MOVOBJ command
- CL - Improved support of partition on CPYFRMSTMF command
- CL - Added support for SNDMSG parameter TOUSR
- CL - Improved support of OVRDBF command
- CL - Improved performance for OVRDBF command - Update Default Values for srcfile and member

- CL - Improved file copy with CPYF command
- CL - Re-engineered CPYF command to be more robust and better handle QTEMP, CRTFILE, FROMRCD & TORCD, MBROPT, and FMTOPT(*MAP* & DROP)
- CL - Improved support for CPYF command for cases where FROMFILE & TOFILE have mismatched columns
- CL - Improved CPYF *NOCHK's* handling of columns with different names when REPLACE is specified
- CL - Added empty implementation for CRTDUPOBJ command on logical files
- CL - Handled substring indexing issue with CHGDTAARA command
- CL - Improved support of SBMJOB command
- CL - Made OverrideManager and OpnqryfHelper mappings case insensitive
- Screen - Improved the initial focus of the first editable field when a cursor is not specified
- Screen - Improved focus position after closing and when using help menu
- Screen - Improved cursor focus after pressing page up/down in table component
- Screen - Improved support for multiple field error messages and focus
- Screen - Improved line number computation for subfile fields
- Screen - Improved support of sub files initialized using SFLINZ
- Screen - Improved support for numeric only entry
- Screen - Improved handling of WINDOW keyword in DSPF with 3 parameters
- Screen - Improved position of footer for table with records containing more than 1 line
- Screen - Improved page navigation for rotation message sticking on Page Up/Down
- Improved EDITC functionality for edit code 3
- Improved Blu4iv data area lock mechanism to do nothing when there is no lock to unlock instead of throwing exception
- Added support to return the number of rows affected in StraightQueryBuilder
- Improved QTEMP log mechanism
- Improved DAOManager reads/writes/deletes for use cases on file overrided by a different file + library

Transversal capabilities

New features

- Added a centralized way to manage the SSL/TLS-related system properties by configuration, allowing the use of AWS Secrets Manager
- Enhanced configuration of IBMMQ resources with AWS Secrets Manager
- JCL - Added the temporary location configuration for Runtime resolved groovy files through the YML property tempFilesDirectory and added the capability to specify whether to purge contents of the temporary files folder at application startup through the YML property cleanTempFilesDirectoryAtStartup
- Add AWS secrets for all Redis credentials

Improvements

- Improved conversion from type alphanumeric to type numeric edited
- Improved DataUtils::isNumeric check for PackedType
- Enhanced log file timestamping
- Handled separate sign in ZonedType.decodeAsString
- COBOL - Improved support of statement INITIALIZE
- Improved support of DataUtils.compareAlphInt to handle leading and trailing spaces for AS400 and ZOS
- SQL - Improved implicit read-only cursor runtime validation
- SQL - Improved Metadata caching mechanism
- Remove Jics/Blusam database connection from Gapwalk Application `application-main.yml`

Modernization tools release 4.3.0

zOS

New features

- GS21 - Add support for COBOL GS21 CONSTANT SECTION
- GS21 - Added JEF encoding to available charsets

Improvements

- CICS - Added support for parsing DOCUMENT CREATE command

- CICS - Added support to parse CICS WEB EXTRACT command
- CICS - Added support for parsing WEB WRITE command
- CICS - Added transformation support for DB2CONN SIGNIN and PLAN
- CICS - Enhanced support for parsing SEND MAP command by ignoring option TERMINAL
- CICS - Enhanced support for parsing RETURN command by ignoring option ENDACTIVITY
- MFS - Improved support to Generate MFS files with specific extension
- COBOL - Improved support for REPLACE statement
- COBOL - Handled dynamic path and MF compiler directive
- COBOL - Improve support for OMITTED value in CALL Statement
- COBOL - Improved multi-dimensional fields access to support signed value
- COBOL - Added support for clause OF for FILE STATUS statement
- COBOL - Improved parsing of statement RESULT-SET-LOCATOR
- JCL - IDCAMS - Added support for RECORDS abbreviation

AS400

New features

- CL - Added support for pointer based and defined variables in CL transformation
- CL - Added support for special characters in DCLF
- Added support for retrieve call stack (QWVRCSTK) API

Improvements

- RPG - Improved transformation of procedure parameters using `likeds` keyword
- RPG - Review support of keyword EXTNAME
- RPG - Improved support literal value *ALL
- RPG - Improved support for output specification and program-described files
- DDS - Improved resolution of DDS fields in a LF that references a PF that references a Dictionary PF
- Screen - Cleared indicators when CLEAR statement is used to clear a record from DSPF
- CL - Improved transformation/generation of CL params with element lists

Transversal capabilities

Improvements

- SQL - Improved generation of SQL queries containing N with tilde character
- COBOL - Improved support of the statement LENGTH OF for group fields
- COBOL - Improved support of REDEFINED fields using copybooks

Release notes 4.2.0

Release date: July 10, 2024

This release of AWS Blu Age Runtime and Modernization Tools is focused on performance and security. Some key features and changes in this release are:

- We improved transformation performance, especially for large projects with over 30 million lines of code. We implemented a set of improvements and the results we obtained showed a time reduction of over 150%, and runs that completed in minutes instead of hours. The key improvement we implemented is the configuration of a timeout mechanism to limit the maximum time allocated for analysis so as to skip files with detected issues. We mark skipped files so that you can investigate them later if necessary.
- We added support for a distributed lock management system for AS400 projects. In a High Availability environment (multi-node) where multiple instances of the application target the same database, maintaining data consistency throughout the life cycle of these instances is a significant challenge. To effectively address this challenge, we added Redis as a shared and external caching server to coordinate among all instances when running in batch mode.
- We added a new dynamic pagination feature for the table component. The goal of this feature is to improve the response time and reduce memory usage for tables with a large number of rows. This feature allows the table component to only load part of the data, and to fetch more records on demand as you navigate through the pages. To further improve the experience, the platform also supports the prefetching of data. This new dynamic pagination feature provides a more efficient and responsive user experience for applications with large data sets.
- To address a key challenge that comes up frequently, we added support for nested COBOL programs. Previously, the workaround for modernizing nested COBOL programs involved manually separating programs into different files, linking them through the linkage section, and making them call each other with the necessary arguments. This process was not only time-

consuming but also error-prone. You can now modernize nested COBOL programs without the need for manual separation.

We tested this version of the AWS Blu Age Runtime with the following stack. Other component versions might also be compatible.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 18.18
	Npm 9.8
	Angular 17
Service layer	Spring Boot 3.2.4
	Spring Core 6.1.5
	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 14
	Oracle 21c
Application server	Apache Tomcat 10.1.17

For more information about the changes included in this release, see the following sections.

Runtime release 4.2.0

zOS

New features

- DB2 - Added support for stored procedure invocation without schema qualifier in the SQL query
- COBOL - Added support for HEX-OF function
- COBOL - Added support for nested programs

- COBOL - Added support for FUNCTION TEST-DATE-YYYYMMDD and TEST-DAY-YYYYDDD
- CICS - Added support for option UCTRANST in the SET TERMINAL command
- CICS - Added support for the INQUIRE DB2CONN command
- BluSam - Added support for key deletion on dynamically accessed VSAM
- IMS - Added support for the TERM command
- BAC - Added authorization checks on all BAC REST endpoints
- BAC - Added configuration through `application-main.yaml` to define a record size to filter loaded masks that match that record size
- BAC and JAC : Added configuration through `application-main.yaml` to retrieve the username and the password of the default super admin user in the secret from `command` by specifying the ARN

Improvements

- JCL - SORT - Enhanced support for OMIT clause to handle conditions with Shiftin and ShiftOut characters
- JCL - SORT - Improved support for the BDW field
- JCL - SORT - Improved support for multiple GDG concatenations with the BDW field
- JCL - DFSORT - Added support for INREC PARSE STARTAFT / STARTAT clauses
- JCL - IEBGENER - Enhanced recordSize handling for output files
- JCL - INFUTILB - Disabled NULL INDICATOR based on YML- FIX GRAPHIC CASE
- JCL - Improved support for FormatterParser to handle constants in the OUTREC field
- JCL - Enhanced load data for graphic type in DSNUTILB program utility
- JCL - SORT - Enhanced support for Zoned Decimal format
- JCL - SORT - Enhanced support for the OMIT clause to handle conditions with Shiftin and ShiftOut characters
- MQ - Improved the handling of MQ connection to fit several business workflows
- CICS - Enhanced support of pointer reference for EXEC CICS READ SET (ptr-ref) statements
- COBOL - Improved support for ADDRESS OF linkage section record
- COBOL - Added support for EXP and EXP10 functions
- COBOL - Improved support for the REPLACE statement using copybook
- COBOL - Improved multidimensional field access to support signed values

- MF COBOL - Added support for variable-format sequential files
- IMS - Improved reading of the configuration of IMS YML files to make it possible to use environment variables
- IMS - Handled additional ways of specifying the segment number
- IMS - Added robustness when an IMS program is called from a programatically-started transaction
- IMS - Improved the search criteria SSA build to take the current length of the WHERE clause into account if the implied segment length is not provided
- IMS - Improved reading of the configuration of IMS YML files to allow the use of environment variables
- Improved support for the VALUE clause in NumericEditedType
- Improved support for string concatenation to handle the case when the first string to be concatenated is empty, blank, or spaces

AS400

New features

- Added support for pagination inside the Table component; projects can use this feature to decrease the response time and size when a Table component with a large number of rows is loaded
- Added library support for SQL queries on the AS400 application; because libraries are converted to partitions in modern applications, we adapted the runtime to rewrite the queries accordingly
- RPG - Added support for the QTEMP library for SQL queries
- RPG - Added encoding in the CONVERT function to handle empty input values
- RPG - Added support for the %HOURS, %MINUTES, and %SECONDS functions
- CL - Added the CHGPFM command
- CL - Added support for the *FROMLIB keyword in the CRTDUPOBJ command
- CL - Added support for table and partition creation for table names exceeding 9 characters
- CL - Added support for deletion of flat files in subfolders for the DLTF command

Improvements

- Screen - Improved ErrorMessage to bind with specific field and add to ArrayMessageLine

- Screen - Improved errormsg cursor
- Screen - Improved ArrayMessageLine to not be included in Tab Order
- Screen - Improved display of error message arrays for AS400 screen
- SQL - Improved cursor support to commit Transaction upon closing to avoid deadlocks on partition creation
- CL - Added support for the PgmCall command and improved the QCMDEXC unsupported pattern
- CL - Improved support for the CHKOBJ command to handle OBJTYPE PGM
- CL - Improved multi-library support for CPYF and other CL commands that deal with libraries and partitions
- CL - Added support for passing a program name variable in the CALL PGM command
- CL - Handled the case for default type of Object type
- CL - Added multi-library support for the CRTDUPOBJ command
- CL - Enhanced database connection handling on multiple commands
- CL - Improved support for RMVLNK to handle the case when a file or directory isn't found and the CPF0000 monitor message
- CL - Improved CLRPFM to take the library into account when removing records
- CL - CPYF - Improved command to support the QTEMP library, FmtOpt(*NoChk) parameter, and control character
- CL - Fixed handling of quotation marks and missing parameters in the RMVLNK and CPY commands
- RPG - Enhanced variable scoping; DataArea is now in working scope instead of linkage scope
- RPG - Improved DAO read queries to run without a transaction to avoid deadlocks
- Enhanced MQ messaging lookup by adding a trim to MSGQ on DB lookup
- Removed unnecessary transaction declarations on database connection support
- Improved the update of Quartz job status in case of exception
- Added support to handle the case when an indicator array isn't initialized

Transversal capabilities

New features

- Redis - Added global Redis configuration for all Redis caches

- Added session-tracking functionality to make it possible to store session-tracking information (session ID, associated username, creation timestamp, and node ID) by persisting the data to Redis
- Added temporary location configuration for runtime resolved groovy files through the YML property `tempFilesDirectory`; also added the capability to specify whether to purge contents of the temporary files folder at application startup through the YML property `cleanTempFilesDirectoryAtStartup`

Improvements

- Enhanced support for connection pool implementation configuration properties for utility data sources
- Improved support for printer mode and ANSI carriage control based on the usage of ADVANCING clauses and WRITE BEFORE clauses
- Updated Angular version on front-end application for modernized projects
- Enhanced secret manager URL syntax construction for DB2
- Enhanced the DataUtils.compareAlphInt method to add support for trailing spaces
- Improved SQL support for blob type output
- Added robustness for job triggers through post/script endpoint

Modernization tools release 4.2.0

zOS

New features

- CICS - Added support for parsing WEB CICS commands
- CICS - Added support for the transformation of the MONITOR command
- CICS - Added support for parsing the CICS command SEND MRO
- COBOL - Added support for parsing the NO REWIND statement
- COBOL - Added support for number type of option UCTRANST in the CICS command SET TERMINAL
- COBOL - Add supported for the MULTIPLE FILE clause in I-O-SECTION
- CSD - Added support for the transformation of multiple CSD files

- CSD - Added support for the generation of jicsFileAix.json from multiple CSD files
- IDCAMS - Added support for the creation of a relative record data set (RRDS)

Improvements

- Improved performance when computing SQL masks
- COBOL - Improved parsing of useless RESERVE clause in FILE-CONTROL
- COBOL - Improved parsing of SECTION and CLASS
- COBOL - Improved DFHRESP handling
- COBOL - Enhanced support for EXIT PARAGRAPH through perform
- IMS - Improved support for segment names specified by using double parentheses
- IMS - Enriched the generation of status codes when SCHD and TERM are invoked
- COBOL - Improved generation of DEPENDING ON fields
- COBOL - Improved transformation of TO_TIMESTAMP DB2 builtin function

AS400

New features

- Added support for converting alphanumeric fields as CHAR in SQL scripts
- COBOL400 - Added support for program-described DATABASE files

Improvements

- DDS - Improved support for ALIAS name
- Enhanced support for type float without initial value
- COBOL 400 - Improved size compute for signed zoned type

Transversal capabilities

Improvements

- Improved error ID reporting around DDS and SQL parsing
- Improved code generation on condition branches
- Improved performance on weather report generation

Release notes 4.1.0

Release date: May 31, 2024

This release of AWS Blu Age Runtime and Modernization Tools is focused on performance and security. Some key features and changes in this release are:

- **Transformation and performance**: To allow projects with a large codebase (+50M lines of code) to transform successfully, we have optimized the performance and memory footprint of the whole transformation mechanism.
- **BAC/JAC**: Security at AWS is the highest priority. Applications modernized with AWS Blu Age must comply with security standards. We have made some major upgrades to the BluSam Administration Console (BAC) and the JICS Administration Console (JAC) to make them more secure:
  - Updated the application to Angular v17.
  - In addition to the native support for AWS Cognito, we added generic support for OAuth that will enable more flexibility to let customers use the identity provider of their choice.
  - Configured and extended the security features using appropriate headers.
- *AS400 - Multi-node support for database lock mechanism.* Provided the possibility to plug a shared and external caching server (Redis) to run a batch application on multiple instances, like managed AWS Mainframe Modernization.

This version of the Blu Age runtime has been tested with the following stack. Other versions might also be compatible.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 18.18
	Npm 9.8
	Angular 16.1
Service layer	Spring Boot 3.2.5
	Spring Core 6.1.5

	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 14
	Oracle 21c
Application server	Apache Tomcat 10.1.17

For more information about the changes included in this release, see the following sections.

Runtime release 4.1.0

zOS

New features

- Added configuration for dynamic OAuth2 provider handling. Introduced SECRET_OAUTH2_PROVIDER_NAME_KEY to specify provider. Updated secret retrieval method to handle multiple providers. Ensured secrets are securely retrieved from AWS Secrets Manager.
- Added support for DB2 SSL properties on AWS Secrets Manager to make it possible for you to define an SSL certificate (sslTrustStoreLocation) and a password (sslTrustStorePassword) to unlock the keystore file.
- Added support for external business data sources.
- JCL - Added support for checkpoint mechanism for batch restart.
- JCL - Added support for DCB parameters record size and RDW.
- JCL - Added dynamic folder-name configuration for generated temporary files.
- REDIS - Added pool configuration in Redis configuration for JICS.
- REDIS - Added database index in Redis configuration for Catalog and JICS.
- BatchScript - Added propagation of step name for running program executions.
- CICS - Added support for the ADDRESS SET command.
- CICS - Added support for PURGE MESSAGE and JUSTIFY.

Improvements

- JCL - INFUTILB - Enhanced support for disabling the null indicator based on YML property.
- JCL - INFUTILB - Improved support for the CHAR/BPCHAR datatype.

- JCL - ICEGENER - Added support for copying multiline input streams into files.
- JCL - IEBGENER - Improved support for handling conversion from Variable Block to Fixed Block files.
- JCL - DFSORT - Improved support for multi-digit parameters on operation DATE.
- JCL - DFSORT - Added support for the INCLUDE=ALL clause.
- JCL - Improved support for the SORT utility to handle the BDW field in output.
- JCL - Improved support for DD concatenation.
- JCL - Improved support for Input Stream.
- JCL - DSNUTILB - Improved support for the NULLIF() statement.
- JCL - INFUTILB - Added support for unloading data with the NOPAD option.
- JCL - INFUTILB - Enhanced support for current date in INFUTILB.
- JCL - Added file existence and size checks before using a file.
- JCL - GDG - Improved the handling of sub-directories for GDG.
- MQ - Improved connection opening in the JMS implementation.
- MQ - Improved data length setting of GET message for XA datasoure.
- MQ - Decomposed CMQV standard copybook to prevent compilation errors and refactoring uses.
- BluSam - Improved support for delete requests for non-existent data sets.
- Improved support for the ALLOCATE statement.
- Improved robustness of TS-QUEUE Naming.
- BatchScript - Enhanced preservation of previous step return code in job re-execution.
- Dataset - Improved the file existence check when a file exists and is temporary.
- Dataset - Improved the concurrency when locating GDG files to delete.
- Dataset - Added support for getting GDG Dataset record size.
- CICS - Improved support for the SUSPENDED option in the INQUIRE TASK LIST command.
- CICS - Improved support for LOAD SET using the ADDRESS OF statement.
- CICS - Improved unhandled CICS arguments REMOTESYSTEM when CICS INQUIRE.
- CICS - Enhanced support for the GETMAIN command to handle the SET option with a pointer defined with the OF keyword.
- JICS - Improved robustness for the jicsXAPrepare() method by adding the transaction state check.
- JICS XA - Added a check for transaction state and enhanced transaction thread termination.
- BAC - Enhanced role-based authentication on client side and refactored/centralized all API calls.

- BAC - Implemented a feature to block public access to BAC and JAC based on configuration
- BAC - Upgrade of the dependencies: Angular 17.
- BAC - Improved security integration with OAuth2.
- BAC - Enhanced hibernate generated DDL.
- BAC - Improved export data set mechanism.
- JAC - Updated to Angular 17 and reporting all specifics work from BAC (ROLE, sadmin conf, XSRF, logout).
- COBOL - Added support for the CHAR and ORD-MIN functions.
- Enhanced FileFactory to keep catalog record size in MOD disposition.
- Enabled logging using MDC for JICS transactions.
- Improved SQLCA > SQLSTATE produced for stored procedures generating ad-hoc result sets.
- Improved support for task scheduling related to last Spring upgrade.

AS400

New features

- Added multi-node support for database record locks using Redis.
- Added support for BINARY CHARACTER for the DDS type.
- CL - Added support for custom report file generation.
- RPG - Added support for the RENAME keyword on primary/secondary files.

Improvements

- Improved database support for handling the CTID column with a JOIN clause.
- Improved cursor position for multiple DSPATR(PC).
- Improved logging on read exception.
- Improved Quartz job logging to include job properties to MDC.
- Improved support for the AS400 help screen.
- CL - Improved support for the RMVJOBSCDE command to accept entry numbers with trailing spaces.
- CL - Improved support for the RMVJOBSCDE command to remove a job schedule using a generic job name.

- CL - Improved support for the SAVOBJ command to order records by table key.
- CL - Improved support for the CPYF command to establish a new connection for DB queries.
- CL - Improved insertion of inquiry messages in queue message with SNDPGMMSG.
- CL - Improved job queue configuration to specify default job queue.
- CL - Improved the CRTPF command to support the QTEMP library and the RCDLEN parameter.
- CL - Improved support for the CHKOBJ command - Check for partition with library.
- CL - Improved RTVMGS to send CPF2407 and CPF2419 when file/ID not found.
- CL - Improved CPYTOIMPF and CPYFRMIMPF interpretation of legacy formatting parameters.
- CL - Added support for OVRPRTF parameter USRDTA.
- CL - Improved the CPYTOIMPF CL command to establish a new connection to avoid closing existing result sets.
- CL - Improved CHGDTAARA so that it no longer modifies the data area length when it updates the content.
- CL - Improved ClCommand database connection handling.
- Optimized interaction between the front end and the back end.
- COBOL - Updated transformation to handle FILLER in copybooks.
- Improved additional message information display for custom messages sent to the front end.
- Updated the default value for the selector in app.component.ts.
- Improved text splitting in split-dynamic-field display.
- Improved the display of error message with multiple writes followed by a read.

Transversal capabilities

New features

Added support for the dynamic configuration of OAuth2 provider secret.

Improvements

- Printing - Improved QCMDEXC parameter support for handling quotation marks and improved report name formation
- Improved support for delimited syntax on RecordAdaptable.
- Enhanced InspectBuilder error logging to add context about source string.

- DataSimplifier - added robustness for ByteArray affectation.
- Enhanced MDC logging with new runtime attributes.

Modernization tools release 4.1.0

zOS

New features

- Added support for multiple CSD file transformations
- COBOL - Added support for the CICS ALLOCATE statement.
- COBOL - Added support for ON SIZE ERROR in the ADD CORRESPONDING statement.
- COBOL - Added support for EXIT PARAGRAPH.

Improvements

- COBOL - Improved support for -INC copybook.
- COBOL - Enhanced support for FILLER initialization.
- COBOL - Improved support for figurative values comparison.
- COBOL - Enhanced support for WHEN ANY in consecutive WHEN clauses lacking intermediary code blocks.
- COBOL - Improved support for figurative constant.
- COBOL - Improved support for packed type size computation.
- COBOL - Improved unhandled CICS argument KEEP for SPOOLCLOSE.
- COBOL - Improved generation for the TEST-NUMVAL function.
- COBOL - Improved Java generation arguments on INSPECT framework support.
- CICS - Improved support for defining DFHCOMMAREA.

AS400

New features

- RPG - Added an error-catching mechanism to generate the (incomplete) DDS so it won't block program generation.
- Added support for the INCLUDE file description specification keyword.

Improvements

- RPG - Improved full-free parsing.
- RPG - Added robustness with error catching.
- RPG - Improved initialization of field/DS with export keyword.
- RPG - Improved DAO operation to handle indicators.
- RPG - Handled the default value of PERRCD with CTDATA.
- RPG - Upgraded the Free-RPG parser to log a unique error per parsing rule.
- PRTF - Handled name collision between PRTF and JRXML.
- COBOL - Improved support of the LIKE keyword.

Transversal capabilities

Improvements

- Added robustness for ErrorID API
- Performance optimization for large project transformation. For example: timeout to skip blocked files, re-use of the classification from Blu Insights, and better memory allocations.
- Optimized the memory footprint during COBOL/PL1 transformation.
- Fixed CVE on third-party (jQuery and bootstrap).
- Managed timeoutParser options in TC.
- Improved the multiple spaces rewriting on SQL queries.
- Improved Read Only Cursor with sensitivity attribute.

Release notes 4.0.0

Release date: April 8, 2024

For instructions on how to migrate from AWS Blu Age Runtime 3.10.0 to 4.0.0, see the section called “Migrating from 3.10.0 to 4.0.0”.

This release of AWS Blu Age Runtime and Modernization Tools is focused on upgrading critical dependencies and supported technologies while boosting performance in multiple functionalities. Some key features and changes in this release are:

- • **Upgrade from Spring Boot 2.7 to 3.2.4, Spring Core 5.3 to 6.1.5, and Tomcat 9.0 to 10.1.17** to provide improved security, performance, and maintainability by using versions that are actively being patched and maintained.
- **Lazy loading on front-end application** to build faster large projects with more than 2000 screens and reduce the displaying initialization from 10 s to 300 ms.
- **Support for DBCS display on front-end application**for enhancement of the support of double-byte characters to provide a new font that handles double-byte and single-byte characters, prevent single-byte input in a double-byte field, and handle fields with mixed double-byte and single-byte characters.
- **Thread monitoring feature for AS400 Online application** to run AS400 application with parallelization.
- **Improved performance on context and RunUnit initialization** by adding a configurable mechanism to pre-initialize program context reducing the impact of loading complex structures inherent in legacy complexity.

This version of the AWS Blu Age Runtime was tested with the following stack. Other versions might also be compatible.

Component	Version tested
Java	Java 17
Presentation layer	Node JS 18.18
	Npm 9.8
	Angular 16.1
Service layer	Spring Boot 3.2.4
	Spring Core 6.1.5
	Spring statemachine 4.0.0
Persistence layer	PostgreSQL engine 14
	Oracle 21

Application server	Apache Tomcat 10.1.17

For more information about the changes included in this release, see the following sections.

Runtime release 4.0.0

zOS

New features

- Added support for include statement '-INC CPYNAME'.
- CICS - Added support for PUSH/POP HANDLE statement.
- COBOL - Added support for "ASSIGN TO DYNAMIC".
- Added support for DB2 UNLOAD using INFUTILB.
- Added support for keyword SEQNUM in an OVERLAY of INREC statement.

Improvements

- SORT - Added support for special chars (parenthesis and asterisks) in sort string literals C'....'.
- SORT - Improved support for OUTFIL NOMATCH-(..) argument.
- SORT - Added support for SYMNAMES data definition.
- SORT - Improved handling of TO= and LENGTH= arguments.
- SORT - Improved handling on MOD disposition.
- SORT - Added support for HIT=NEXT argument.
- Enhanced ICEGENER to add support for specific output file encoding.
- INFUTILB - Enhanced support for WITH UR clause.
- INFUTILB - Enhanced support for unload when writeNullIndicator is false.
- DSNUTILB - Enhanced robustness to load step when the NULLIF keyword is after an optional SQL keyword.
- DSNUTILB - Enhanced support for isolate column name.
- DSNUTILB - Added support to load an empty file into a table.
- DNSUTILB - Added support for MOD disposition for the DNSUTILB SYSDISC file.
- IDCAMS - Enhanced comments support.

- JCL - Added support for column with double quote in LoadTask.
- JCL - Enhanced UNLOAD SQL query handling regarding whites paces removing.
- JCL - Enhanced response of Groovy script when an exception occurs in processing to assure a JSON format.
- JCL - Improved check file disposition in the case of DISP=NEW and DISP=OLD.
- JCL - Enhanced support to handle multiple GDG generation reference with special character in GDG base name.
- JCL - Enhanced support to load a dummy file.
- JCL - Enhanced support for tempFilesDirectory YML parameter.
- JCL - Improved JSON return when it is needed to escape double quotes within a string element .
- JCL - Enhanced FileUtils to support GDG base name.
- JCL - Enhanced DSNTEP program for DB2 multiple queries execution.
- Added support for Spring beans.
- Enhanced SQLConverter to avoid rectifying wrong dates.
- Improved JicsTimeBuilder handling of YYYYDDD.
- Allowed custom jars to be accessible from groovy.
- IMS - Enhanced navigation across records in the IMS database implementation.
- IMS - Enhanced CBLTDLI to be able to launch program use purge.
- IMS - DFSRRC00 able to pass the params from groovy to backend program.
- Added support for JICS command that was not invoked through a transactionRunner.
- JICS - Improved performance by using configurable cache.
- BluSam - Add support for disabling warmup BluSam when opening to enhance performance for large dataset.
- BluSam- Improved delete/rename behaviour on regular BluSam data sets.
- BluSam - Enhanced performance on record operations.
- Enhanced datasimplifier for the methods determining if a string is low value.
- Enhanced support for Packed-Decimal & sorting order issue.
- Enhanced configuration of DB2 as primary data-source with AWS Secrets.
- Enhanced FileSystem API to expose the file status.
- Enhanced DynamicFileBuilder read stream input with lineSeparator.

- Enhanced datasimplifier for the methods determining if a string is low value when deals with CUSTOM930 charset.
- SQL - Improved SQL Stored Procedure Output Processing.
- SQL - Improved lambda mapping for multiple table with aliases.
- COBOL - Improved support fro LENGTH OF statement.
- COBOL - Added support for TRANSFORM statement.
- COBOL - Added support for 9 new mathematical functions.
- COBOL - Improved support for INTEGER-OF-DAY FUNCTION.
- COBOL - Improved support for 88 level involving figurative value.
- COBOL - Improved transformation for SET ADDRESS statement.

AS400

New features

- Removed duplicated indicator entities.
- Added support for DBCS characters.
- Introduced handling of HELP keyword for subfile record control.
- Added configuration parameter to toggle column name capitalization & split comment column content on pipe char.
- Added support for using 0x0c as last nibble for Packed type fields.
- RPG - Handled prototypes declared with ExtProc('system').
- CL - Handled 'CLEAR' parameter of cl-command RMVMSG + introduce in-memory non-program message queues.
- CL - Handled generic statements being passed to SBMJOB CMD() calls.
- CL - Added command STRCMTCTL and ENDCMTCTL. Modified locking mechanism and cleaning up of transactions and locks.
- CL - Added support for RCDDLM parameter for CPYTOIMPF command.
- CL - Added handling of padding zeros in SAVOBJ command.
- CL - Added handling of libraries included in the qualified name of the OBJ parameter for RTVOBJD.
- CL - Added support for CPYTOIMPF command params STRDLM, STRESCCHR, and RMVBLANK.

- CL - Enhanced RTVMGS to send CPF2407 and CPF2419 when file/id not found.
- CL - Improved RCVF command to receive records from any provided library in DEV parameter.

Improvements

- Changed default values for Blu4iv task executor to allow better scaling by default.
- Parameterhelper modified to convert list of strings and ElementaryRangeReference to String.
- Enhanced CTID to handle not existing column in POSTGRE.
- Added robustness to support user space API "QUSPTRUS".
- Added support for User Spaces APIs QUSRUSAT and QUSCUSAT.
- Enhanced support for User Space API (QUSPTRUS) without error code.
- Added support for CRON Job Scheduling using Quartz.
- Enhanced support of RPG program cycle.
- Improved Blu4iv transaction management.
- Record locking of files under commitment control within same transaction has been improved.
- Improved handling of subfile initialization.
- Improved display of scroll indicators for Message Lines.
- Prevented trailing zeros on numbers sent through data queue.
- Improved Additional Message Information Screen.
- Improved JPA write operations to consider current library.
- Improved behavior of ProgramJobExecutor when executing programs without parameters.
- Added functionality to directly pass arguments from front end links to back end scripts.
- Improved transaction handling for jobs metadata.
- CL - Added support for param SECLVL in RTVMSG.
- CL - Added empty implementation for CLRLIB.
- CL - Improved CPYFRMIMPF support for copying from both database and CSV.
- CL - Improved CPYFRMIMPF implementation to ignore extra columns.
- CL - Improved CPYTOIMPF and CPYFRMIMPF interpretation of legacy formatting parameters.
- CL - Added param removeDecimalPoint to format numeric values in SAVOBJ.
- CL - Improved RCVF command to properly handle EOF condition.
- CL - RTVSYSVAL - Implementation SYSVAL = QDATETIME.

- CL - OVRDBF command modified to get field as default table name.
- CL - RTVJOBA Unavailable value for param : USRLIBL.
- CL - Handled leading slashes in SNDPGMMSG MSGF param.
- CL - Improved support for wildcards in sourcefile in command DSPFFD.
- CL - Improved handling of param PGMQ in RCVMSG and SNDPGMMSG.
- CL - Made RTVMSG param MSG optional to align with legacy docs.

Transversal capabilities

New features

- Improved capability when passing parameter at USING clause of OPEN cursor.
- Performance: Improved pre-initialization of context and RunUnit for performance tuning.

Improvements

- Improved the mechanism to dump low values from UNLOAD command of INFUTILB utility program.
- Added support current schema option on datasources secret manager.
- Enhanced runtime to not consider parameters passed at open cursor when they are not needed.
- Improved numeric format validation for numeric fields.
- Improved SQL Diagnostic in highly parallel execution environment.
- Introduced unicode for codepage byte sequence (FE FD).
- DataSimplifier performance optimization - Enhanced assign statements.
- DataSimplifier performance optimization - Improve default value for numeric type initialization to prevent useless BigDecimal usage.

Modernization tools release 4.0.0

zOS

New features

- Added support handling Abend PROGRAM.
- Improved support to generate AIX dataset.

- COBOL - Added support for JUSTIFIED clause on ALPHANUMERIC/ALPHABETIC/GRAPHIC fields.

Improvements

- Improved PURGETHRESH attribute handling for TRANSCLASS resource definitions.
- Improved support for data definition and MOVE statement.
- CICS - Enhanced support for DELAY command on option MILLISECS.
- Improved SQL lambda mapping for multiple tables with aliases.
- Improved support for parent field finding.
- Improved SQLCA sqlstate set for COMMIT and ROLLBACK operation.
- COBOL - Enhance parsing by commenting obsolete paragraphs
- COBOL - Enhanced support for REPLACING clause.
- COBOL - Added support for mathematical functions ASIN ACOS LOG TAN.
- COBOL - Added support for multiple AFTER statements in PERFORM VARYING.
- COBOL - Enhanced support for RENAMES (level 66) fields.
- COBOL - Enhanced LENGTH OF method to get length at a specific index in an array field.
- COBOL - Added support for multiple AFTER clauses in PERFORM VARYING statements.
- COBOL - Enhanced support for RENAMES clause.
- COBOL - Enhanced support of PICTURE keyword.
- COBOL - Enhanced support for Level 88 field parsing.
- COBOL - Improved goto depending condition with table data items.

AS400

New features

- Added functionality to pass arguments to direct front end java calls.
- CL - Improved %SST generation including support for *LDA with CL→Java.
- RPG - Added support Program-Described record for DISK files.

Improvements

- Improved display file, resolve referenced fields with the "REFFLD" keyword.

- Improved support of display file keyword SETOF-CSRLOC.
- Removed files from the commitment control after closing.
- Ensured consistent behavior for concurrent Read and Write Operations on a table when performed by the same program.
- Handled assignment to substring of SizePrefixedAlphanumericType.
- Handled passing data structure to procedure with varying-length string parameter.
- Improved retention of invalid numeric values upon onBlur event and creation of event listeners for valid fields only.
- Improved error messages on screens and highlighting of fields with invalid input.
- Improved handling of screen fields conditioned on indicators.
- Enabled scrolling with mouse wheel.
- Added support for function keys for Help screen.
- Improved support for long text in split-dynamic-field component.
- Improved handling of multi-record LF files when renaming records.
- CL - Improved RTVJOBD command to handle LF files (views).
- CL - Improved OVRDBF command when used on a multi record LF.
- RPG - Handled scenario where procedure defines a variable with same name as renamed param.
- RPG - Improved handling of *ZEROS when initializing signed binaryInteger.
- RPG - Improved handling of pointers to non-local (reference) variables.
- RPG - Improved handling of ELSEIF statements following IFxx statements.
- RPG - Added support for Fields defined with LIKE on prototype.
- RPG - Improved the support for LIKE keyword of a field created by LIKEREC.
- RPG - Improved generation of operators with figuratives.
- RPG - Improved parsing for array expression xxx(*) and support it in %lookup.
- RPG - Improved LookUp operation code with high and equal (or low and equal) indicators.
- RPG - Improved free form parsing.
- RPG - Improved parsing of I-card named constants that follow I-card record formats.
- RPG - Improved support for type INTEGER and UNSIGNED.
- COBOL - Added support INDIC clause of DSPF format in COPY DDS statement.
- COBOL - Improved grammar for DISPLAY and ACCEPT statements to unblock transformation and generation.

- COBOL - Added support fro DISK files.
- COBOL - Improved DDS display files support programs.
- COBOL - Added support for LIKE clause.
- COBOL - Added support for Program-Described DISK file.
- COBOL - Added support for file name with suffix.

Transversal capabilities

New features

- Handled the Lazy loading of Map Components of web projects.

Improvements

- Improved java generation of SQL indicators parameters.
- Improved capacity to handle variables involved in SET DB2 statement.
- Improved raise of error at end of fetched cursor when output is a single entity array.
- Managed path in Linux.
- Data Migrator manage vulnerabilities and remove unused dependencies.

Release notes 3.10.0

This release of AWS Blu Age Runtime and Modernization Tools is focused on core baseline upgrades and improvements across the product striving to increase performance and robustness in all transformation and execution steps. Some key features and changes in this release are:

- Version upgrade from Java 8 to Java 17, increasing security and performance, and allowing customers to deploy and run applications implemented in a more modern language and to use recent third-party framework versions.
- Additional support for managing large shared memory spaces between users or jobs, storing data reusable after application or instance restart.
- Faster access to large data sets in Blusam using a pagination mechanism that makes it possible to retrieve a subset of records incrementally.

For more information about the changes included in this release, see the following sections.

Runtime release 3.10.0

This runtime is based on Java17, Spring2.7, and Angular16.

zOS

New features

- Blusam - Added support for large data sets through a paginated mechanism where indexes are stored and loaded using pages

Improvements

- Enhanced DataUtils.compare to handle lower precedence conversion from string to number
- Added support to check that no ByteRange is created with improper values through YML property dataSimplifier.byteRangeBoundsCheck
- Enhanced removeSOSI() to support the initialization of a GraphicAlphanumericType with an empty character
- Added robustness for job operation and secure GDG state read
- Blusam - Added support for clearing Ehcache of Blusam data sets through a new method named CoreBluesamManager.removeCache()
- Blusam - Improved delete/rename behavior for regular Blusam data sets
- Redis - Enhanced support for unlocking data sets and clearing record lock
- JICS - Improved the error message for failed requests
- JCL - Added support for ControlM variable concatenation based on dot character
- JCL - Added support for Write ADVANCING (ADV) for GDG files
- JCL - Enhanced support for current generation number after delete all GDG files
- JCL - Enhanced support for rdw/recordSize reading from catalog at dataset creation
- JCL - Added support to update the resource object (from AbstractSequentialFile) when opening the file with the size of data output record
- JCL - Improved IDCAMS performance
- JCL - Enhanced support for PRINT STATEMENT by adding "CHAR" as alias of "CHARACTER"
- SORT - Enhanced support for copy operation from a Blusam fixed-length dataset to a dataset with variable length
- SORT - Enhanced sort grammar to handle some specific statements

AS400

New features

- Added support for User Spaces and its related APIs
- Added support for TOMSGQ parameter of SNDPGMMSG and implemented message queues
- CL - Added support for FILE and SPLFNAME params for the OVRPRTF command
- CL - Added support for handling libraries for corresponding partition table with the CPYF command
- CL - Added support for handling the CHGCURLIB command and considering the current library when building queries
- CL - Added support for handling the cl command as part of the call stacktrace

Improvements

- Improved MessageHandlingBuilder for better handling of the call stack trace entry
- Improved parallel execution of the contextPreconstruct feature
- Improved display attributes when a record is created by SFLINZ
- Improved SAVOBJ to allow the handling of multiple output files
- Improved groovy programs handling by adding them to programCallStack when they are called from a Java program
- Improved top positioning detection of help modal
- Improved toPgmQ functionality when toMsgQ param is provided for SNDPGMMSG
- Improved fetching of predefined messages and functionality of message loader
- Improved CPYTOIMPF handling of delimiter characters in content
- Improved release lock on READ record

Transversal capabilities

New features

- Added a translation for system messages on Front-End
- Added a new method in ExecutionContext to return the program call stack
- Set a line separator (for data simplifier) regardless of the actual environment

- Added the possibility to configure the SQL model JSON path

Improvements

- Improved the comparison method DataUtils.compareAlphInt() when padding is involved
- Creation of a flag to allow custom behavior on exception in cursor queries
- Improved graphic LOWVALUES db conversion

Third party

- Upgrade to mitigate CVE-2024-21634, CVE-2023-34055, CVE-2023-34462, IN1-JAVA-ORGSPRINGFRAMEWORKSECURITY-5905484, CVE-2023-46120, CVE-2023-6481, CVE-2023-6378, CVE-2023-5072)

Modernization tools release 3.10.0

zOS

Improvements

- COBOL - Added support for ABS function
- JCL - Enhanced variable scope: attached to STEP instead of JOB
- Enhanced cursor parameter injection for low/high value
- Improved CSD parsing, notably for remote TRANSACTIONS

AS400

Improvements

- Removed blank check for Control Level Indicator
- Added support for external name for IMPORT/EXPORT keywords
- Added support for %LEN on fields
- CL - Added support for new operators for the CLLE language
- CL - Added support for nested IF
- COBOL - Improved handling of the START command when used with multiple keys

- DSPF - Improved handling of cursor position with record number
- DSPF - Improved the formatting for signed numeric, numeric only fields, and fields with large scale
- DSPF - Improved the determination of the title for Screen General Help
- DSPF - Improved support of Input/Output specifications
- DSPF - Improved handling of grouping separators during validation of numeric field
- Improved mapping output/DDS records
- Improved printer file REFFLT keyword ability to resolve referenced fields
- RPG - Enhanced support for “ALL free” statements
- RPG - Improved condition parsing and added support for handling CABXX without result TAG
- RPG - Improved input specification handling of numeric fields
- RPG - Improved handling of procedure calls within IF/ELSEIF/WHEN conditions
- RPG - Improved handling of READ command when called on a dspf file
- RPG - Improve support for files referring to a non-existing DDS
- Improve handling of REFFLD when passed a physical record format name
- Added support to use 'return' as a db column name

Transversal capabilities

New features

- Oracle - Made it possible to define users than SYS to store built-in functions

Improvements

- Upgraded Java version from v8 to v17
- Improved SQL condition with Cluster column name
- Added support for ORDER BY clauses from view

Release notes 3.9.0

This release of AWS Blu Age Runtime and Modernization Tools is focused on multiple transversal enhancements across the product striving to increase performance in high-availability

architectures, along with new capabilities to raise jobs executions to the next level. Some key features and changes in this release are:

- Version upgrade from Angular 13 to Angular 16, increasing security and giving access to new features that improve the performance in customer's online applications.
- Add support of cross job features in AS400, with the main high-light that jobs can send inquiry messages synchronously among them, enabling decoupling in modernized jobs.
- Performance improvements on the usage of Redis, including connection pool optimization, high security on connection and upgraded dataset locking mechanism.

For more information about the changes included in this release, see the following sections.

Runtime release 3.9.0

zOS

New features

- Sort program: Updated VSAM inputs with fixed length
- JHDB DB: Added configurable timeout

Improvements

- Enhanced support for line separator to stream if used in files concatenation
- Enhanced support to open concatenated sequential files. Initialize DataSetIndex after opening of the file
- Enhanced support for virtual decimal separator when a NumericEditedType is affected to a numeric value
- Enhanced support for NumericEditedType on negative values
- IDCAMS: SYSIN cards are now read using the "encoding" property defined in application-utility-pgm.yml
- IDCAMS: Updated grammar to support FILE(..) argument in DEFINE CLUSTER statement
- INFUTILB: Added support for DFSIGDCB argument to override DCB parameters of DD SYSREC
- INFUTIL: Enhanced support for "DFSIGDCB YES" parameter
- Improved SPLICE to handle huge input file

- DFSORT: Improved remark fields handling
- DFSORT: Added support for (signed / unsigned) free form numeric format (SFF/UFF)
- SORT: Added parsing support for OPTION PRINT and OPTION ROUTE statements
- SORT/ICEMAN: Added support for enclosed division operations (field with DIV operator)
- Enhanced support for CICS READ using generic key
- Function StringUtils.chargraphic fixed to remove SOSI from a graphic type
- Enhance performance on DataUtils.isDoubleByteEncoding
- JCL: Enhanced support for KEEP disposition mode for a temporary data set. The system changes the disposition to PASS
- JCL: Handles DCB parameters dynamically
- JCL: Enhanced SUM FIELDS outputs for incorrect values
- JCL: CommonDDUtils::getContent now searches for the recordSize in the catalog
- JCL: Read rdw/recordSize attributes from catalog at dataset creation
- JCL: Added support for DCB=.MYDD to copy DCB parameters of a DD into another in same job step
- JCL: Improved record size inheritance system
- JCL: Added (Redis) exclusive dataset lock
- Redis: Added SSL support for standalone mode
- Redis: Added synchronized Redis lock count with lock
- Redis: Supported Pool parameters for Redis lock
- Redis: Optimized metadata refresh with Redis
- Redis: Improved redis cluster support
- Improvement on open locks with IO mode
- Improved datasets locks performance and clear unused locks
- Enhanced path of the dataset during unregister file
- Improved pre-fetch window cache invalidation
- Added support for thread safe utility datasource provider usage
- Enhanced datasetState nullity check
- Enhanced support for not reopening already opened data sets
- Added robustness for job final operation

- Enhanced support for indexes order for keys allowing duplicates
- Enhanced support for skip list serialization order
- Added support for debug dump feature to help diagnose indexes order issues
- Enhanced support for metadata refresh
- Enhanced support for Blusam bulk read

AS400

New features

- Creates an application-context registry
- Support for DSPF keyword CLRL(NO) Support record locks monitoring
- Support for keyed DataQueue
- Support for INQUIRY messages for batch jobs
- Added support for Program-described Printer file for AS400 COBOL
- Handles RMVJOBSCDE cl command
- Improvement for RUNSQL/DLYJOB
- CHKOBJ: Raising legacy error code for parameter LIB
- SNDPGMMSG: Supports string parameters
- RTVDTAARA: Improved substring in LDA
- DSPFD: FILE param supported added for specific file name
- RUNQRY: Support for sql file in QRY PARAM
- CRTDUPOB: Support to copy the data between data areas
- SBMJOB: Converts instruction to use JobQueueManager
- OPNQRYF: Added support for Qtemp library
- CRTDUPOBJ: Improved logic for copying partition content
- CRTDUPOBJ: Added support for Qtemp for views
- RTVSYSVAL: Support for SYSVAL value, QDATFMT in CL command
- CHKOBJ: Added support for OUTQ
- RTVJOBA: Supports SWS param
- SNDPGMMSG and RCVMSG: Additional parameters supported MSGF, MSGFLIB, MSGDTA, MSGTYPE, KEYVAR, MSGKEY, MSGID

Improvements

- Improved WORKSTATION I/O cards supports
- Improved handling of set message overlaying previous message
- Supports additional message information on array-messageline
- Improved standalone array wrapper access inside EVAL, SortA, figuratives
- Improve DAOs cleaning when online application ends
- Added support for additional date formats and improve handling of string inputs
- Improved CVTDAT handling of SYSVAL by adding system value helper class Decode and build parameters from CL command SbmJob
- Removed package com.netfective.bluage.gapwalk.rt.blu4iv from gapwalk-cl-command component scan
- Improved the support of predefined messages for message queue API
- Improved the support of retrieveSubfileRecord for record written in another program
- Improved the support of immediate messages for message queue API
- Improved handling of local data area when submitting a job
- Starts JobQueues automatically when the server starts
- Uses applicationContext configuration to decode params for SBMJOB
- Improvement on system-supplied error messages
- Allows RTVMSG to search for .properties files in nested sub-directories
- Handles reset of entities bound to bad/invalid pointers
- Improved MessageHandlingBuilder to display msgId and MsgFile name as strings for RCVMSG
- Improved withMsgFileName method of the message queuing API
- Improved data area lock mechanism
- RTVMBRD: Support for lower and upper case for parameter FILE
- CRTDUPOBJ: Improved handling of views
- CPYTOSTMF: Improved handling of connection
- CPYF: Improvement in handling directory name while copying from a flat file
- RCVF: Properly handles DEV/RCDFMT parameters and transformation of RCDFMT for groovy and java
- RCVF: Handles subsequent calls and avoid resetting the cursor
- CPYF: Added support for writing from flat files

- CRTDUPOBJ: Added handling of new obj with Qtemp library
- CHGDTAARA: Increased data area max length from 256 to 2000
- SAVOBJ: Ensure records saved are in insertion order
- RTVDTAARA: Values retrieved (not to be trimmed)
- CHKOBJ: Returns correct monitor messages when member does not exist
- RTVDTAARA: Added support of LDA substring
- RTVDTAARA: Returns whitespaces up to the length of variable specified in the RTNVAR param
- RTVDTAARA: Supports integer parameters for start and length and support latest transformation format
- CHGDTAARA: Added support for parameter that includes lower and upper bounds
- CHKOBJ: Handles VIEW value for parameter object type
- CHKOBJ: result set to true irrespective of member if the view exists

Transversal capabilities

New features

- Handles generating reports to .txt files
- Added currentSchema XA datasource property to secret manager
- Add database.cursor.raise.already.opened.error YAML property to enable framework to raise SQLCODE error 502 when already opened cursor is opening

Improvements

- Added gapwalk poms to AWS Blu Age on Amazon EC2 packaging
- Uses the new signal handler paradigm by default
- Add support for lock when disposition is MOD or OLD
- Added cache to store database date time patterns
- Improved check function of PackedType
- Improve DataUtils.setTo functions for Records with VariableSizeArray
- Handles MQ SYNCPOINT option as regard as run unit
- Enabled framework to set SQLCODE on rollback transaction
- Added automatic driver class name according to engine key secret

- Program/Transaction timeout
- Restore cursor position after Rollback when accessing cursor

Third party

- Upgrade SnakeYAML, Redisson and Amazon SDK, remove YamlBeans (mitigate CVE-2022-25857, CVE-2023-24621, CVE-2023-42809, CVE-2023-44487)

Modernization tools release 3.9.0

zOS

Improvements

- Enhanced support for XML-TEXT as source for target of type String
- Enhanced STM to UML workflow to support X/(Y/Z) division pattern
- JHDB DB: Accepts ROLLBACK call before any database update
- JHDB DB: Accepts ROLLBACK even if transaction is terminated (NOP)
- JCL: Improved step validation function
- SORT: Handles SUM function with zone decimal negative values
- COBOL: Adds support for single/double quote escaping in string literals

AS400

Improvements

- Improved built-in function %editc handling of edit code X by adding leading zeros
- Improved handling of input only fields initial value
- Added action keys to help dialogs
- Footer record of dynamic table appearing at the bottom
- Handled START command without KEY PHASE for files that specify an actual RECORD-KEY
- Added default value for float and NumberUtils::pow type
- Added support defining a variable using LIKE(IN)
- Updated FOR loop handling to support omitting optional elements

- Updated RPG parsing to associate records with CTDATA array name
- Improved handling of indicators for CABxx statements
- Supports optional parameter on COMMIT keyword
- Improved FORMAT Keyword support in LF
- Managed LOOKUP operation code with high and equal (or low and equal) indicators
- Handled PF key name declared within double quotes
- Improved the handling of EDTCDE X to not suppress leading zeroes
- Improved support for MSGCON in printer file not generating unnamed labels
- Field CONTENT is shared by multiple data structures
- Handled ERRSFL parameter in combination with SFLMSG/SFLMSGID
- Improved main code before proc declaration scope of full free rpg
- Added parsing conditioned control specification
- Improved support for setErrSfl() method in dataholdermapper
- Improved type resolution for internally created variables
- Improved support for Z-ADD opcode
- Improved the handling of constant field with DFT value
- Improve the support of integer field inside program status ds
- Handled indicator assignment in ENTRY params
- Improved the filter of keywords propagated through ref/reffield keyword
- Supported unnamed DataArea data structure
- Improved handling of pointer data type
- Handled elements of array used to define variables with LIKE keyword support array access in output field
- Improved support for signed numeric, only displaying digits
- Supported logical relationship on O card
- Test case for %CHAR on alphanumeric
- Supported control specification keyword main
- EDTCDE with two parameters in printer file
- Improved FullFreeRPG parsing
- Enhanced the dynamic table to ensure the footer is correctly positioned
- Added support for initializing numeric types with ALL figurative constant

- Improved handling of multiple RPG logical files referencing the same physical file
- Improve the detection of modified fields in a modern screen
- Modal synchronization with dynamic fields
- Improved the handling of output only signed numeric field
- Improve WORKSTATION I/O cards supports

Transversal capabilities

New features

- Data Migrator Tool: Added ebcdicFilesWithVarcharInVB property to allow taking VARCHAR 2-byte length into account when reading bytes
- Implemented a common API to log error
- Implementation of BluAgeErrorDictionaryUtils and use of common API to log error and/or info in COBOL2Model, RPGCycleBuilder, Definitions2Model and FieldsProcessor
- Improved SQL grammar to support different isolation clause definition

Improvements

- Upgraded Angular version to v16
- Angular: Upgraded ajv version from 6 to 8.9

Third party

- Upgraded Groovy to version 2.4.15

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.

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 set up 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 ERRMSGID and CHKMSGID
- 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 Blusam secret performance
- Improved handling of database connections in SQL framework
- Standardized AWS multi-datasource secret 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

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 Â¥ 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 built-ins
- 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
- Improvedsupport 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

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, and others.
- 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.

Runtime release 3.7.0

Topics

- zOS

- AS400
- Transversal capabilities

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)
- UseSsl 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)

Modernization tools release 3.7.0

Topics

- zOS
- AS400
- Transversal capabilities

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)

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.

Runtime release 3.6.0

Topics

- zOS

- AS400
- Transversal capabilities

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 multi line 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)

Modernization tools release 3.6.0

Topics

- zOS
- AS400
- Transversal capabilities

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 PSDS 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)

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 datasets metadata and import datasets from S3 buckets.

For more information about the changes included in this release, see the following sections.

Runtime release 3.5.0

Topics

- zOS

- AS400
- Transversal capabilities

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 dataset 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 Fiedl.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 ResulSet. (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)

Modernization tools release 3.5.0

Topics

- zOS
- AS400
- Transversal capabilities

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 programs (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 ebcdic 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 Â¬ character by ! in case of Â¬ 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)

AWS Blu Age security vulnerabilities

Common Vulnerabilities and Exposures (CVE) is a reference list for publicly known cybersecurity vulnerabilities. Each entry contains an identification number, a description, and at least one public reference.

We recommend that you always upgrade to the latest AWS Blu Age version release to be protected against known vulnerabilities. Security scans are continuously performed with Amazon Inspector and findings are classified as per their severity in NIST.

The following list details the CVEs fixed in each available minor version, that result from the use of dependencies:

Version	CVE
4.8.0	CVE-2025-22228
4.7.0	CVE-2025-25193, CVE-2025-24970, CVE-2024-47535, CVE-2024-38820, CVE-2024-38809, CVE-2024-22262, CVE-2022-45868
4.6.0	CVE-2024-12801, CVE-2024-12798, CVE-2024-50379, CVE-2024-56337
4.5.0	CVE-2024-47535, CVE-2024-52316, CVE-2024-47535, CVE-2024-38827
4.4.0	CVE-2024-38820, CVE-2024-38821, CVE-2024-38809, CVE-2024-38816, CVE-2024-47554, CVE-2024-6484, CVE-2024-6485
4.3.0	CVE-2024-43788, CVE-2022-25898, CVE-2021-30246, CVE-2024-21484, CVE-2024-34750
4.2.0	CVE-2020-11023, CVE-2023-26364 , CVE-2019-11358, CVE-2020-11022,

Version	CVE
	CVE-2021-23358 , CVE-2017-18214, CVE-2022-24785, CVE-2022-31129, CVE-2023-48631
4.1.0	CVE-2024-29025, CVE-2024-23080, CVE-2024-22262, CVE-2024-30171, CVE-2024-29857, CVE-2024-30172
4.0.0	CVE-2016-1000027, CVE-2022-1471, CVE-2024-1597, CVE-2024-22243, CVE-2024-22233, CVE-2024-22234, CVE-2024-22259, CVE-2024-22257, CVE-2024-29131, CVE-2024-29133

Note

For details on CVEs fixed in previous release versions, please contact your AWS Blu Age delivery manager

Upgrading instructions for AWS Blu Age

This page contains instructions for upgrading the AWS Blu Age version.

Common upgrades

In most of the cases, when upgrading the AWS Blu Age Runtime (non-managed) version, you should replace the artifacts (WARs, configuration files, scripts, etc.) of your previous version with the ones provided in the new one and restart your application. Make sure to perform extensive regression tests of your modernized applications once you upgrade. You may also contact your AWS Blu Age delivery manager for specific instructions applicable to your application.

To upgrade the AWS Blu Age Runtime (managed) version, see *Managed runtime environments*.

Some upgrades may require additional configuration to ensure compatibility. In that case, follow the instructions for that specific upgrade.

Migrating from 3.10.0 to 4.0.0

The main change in 4.0.0 is the migration from Spring Boot 2.7 to Spring Boot 3.2 and from Tomcat 9 to Tomcat 10.

Code changes

This section lists changes required to make the modernized code compatible with AWS Blu Age Runtime 4.0.0. You can skip this section if you decide to launch a new generation using the 4.0.0 version on Blu Insights (Transformation Center).

POM changes

Group	ArtifactId	Change
org.slf4j	slf4j-api	Remove (is a transitive dependency)
org.yaml	snakeyaml	Remove (is a transitive dependency)
org.springframework.boot	spring-boot-starter-web	- Upgrade spring.boot.version to 3.2.4 - Remove exclusion of log4j-to-slf4j
org.springframework.boot	spring-boot-starter-jta-atomikos	Change to com.atomikos:transactions-spring-boot3-starter:6.0.0
org.apache.commons	commons-dbcp2	Upgrade to 2.10.0
org.postgresql	postgreql	Upgrade to 42.7.2
com.microsoft.sqlserver	mssql-jdbc	Upgrade to 12.4.2.jre11
com.oracle.database.jdbc	ojdbc8	Change to ojdbc11 version 23.3.0.23.09

Migrate from Javax to Jakarta

The tomcat upgrade comes with a migration from the Javax Java package to Jakarta. **Make sure to update your imports accordingly from javax.* to jakarta.***.

Nearly all the old referenced classes in the Javax package can be found in Jakarta. Known exceptions to this are the `javax.sql` and `javax.xml` packages, which are still unchanged.

Atomikos change

Due to the dependency change referenced above, references to `org.springframework.boot.jta.atomikos.AtomikosDataSourceBean` must be changed to `com.atomikos.spring.AtomikosDataSourceBean`.

PostgreSQL dialect removal

The custom class `PostgreSQLDialect.java` is removed. References to it in the main launcher must be removed too.

Deployment (AWS Blu Age Runtime (non-managed))

Tomcat

This version is compatible with Tomcat `10.1.17`. Upgrading the Tomcat server to this version is required to run the Blu Age Runtime `4.0.0`. Make sure to port the old configuration changes (notably the Catalina properties).

Shared dependencies

The runtime shared folder contains the up-to-date dependencies.

Extra dependencies

If you used extra dependencies (not included on the runtime), you might need to update them. The readme file in the extra folder lists the supported versions.

AWS Blu Age lifecycle

This section defines end of life (EOL) dates for the AWS Blu Age Runtime major versions. This enables you to plan for version upgrades so that you can stay up to date with the latest maintenance and features. To upgrade your version, see the section called “Upgrading AWS Blu Age”.

We recommend you check for new versions every 3 months and upgrade to recent versions frequently. For each upgrade, you should perform non regression tests of your modernized applications prior to production or critical deployments.

> **Note**
>
> End of life dates may be subject to change due to critical security remediations. For more details, see *Components lifecycle*.

AWS Blu Age Runtime End of Life (EOL)

The following table summarizes EOL date for each major version.

Major version	End of Life date
Version 3	July 8th 2024
Version 4	Yet to be released

> **Note**
>
> EOL date for major version 4 will be aligned with the availability for the next major version.

To understand the support model of minor versions, refer *Components lifecycle*.

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
- AWS Blu Age structure of a modernized application
- What are data simplifiers in AWS Blu Age

- AWS Blu Age Blusam
- Available programs in utility web application
- AWS Blu Age Blusam Administration Console

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
- Execution environments
- Statelessness and session handling
- High availability and statelessness

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 What are data simplifiers in AWS Blu Age.

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 APIs.

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. 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, concatenated, VSAM (through the **Blusam** set of libraries), and GDG Data Set organizations,
- 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 (Redis OSS)
- 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:

- 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 to retrieve the corresponding fully qualified names for AWS Blu Age runtime elements and Third party fully qualified name mappings 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.

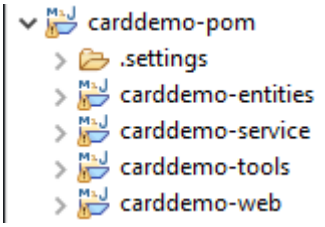
Topics

- Artifacts organization
- Running and calling programs
- Write your own program
- Fully qualified name mappings

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 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.



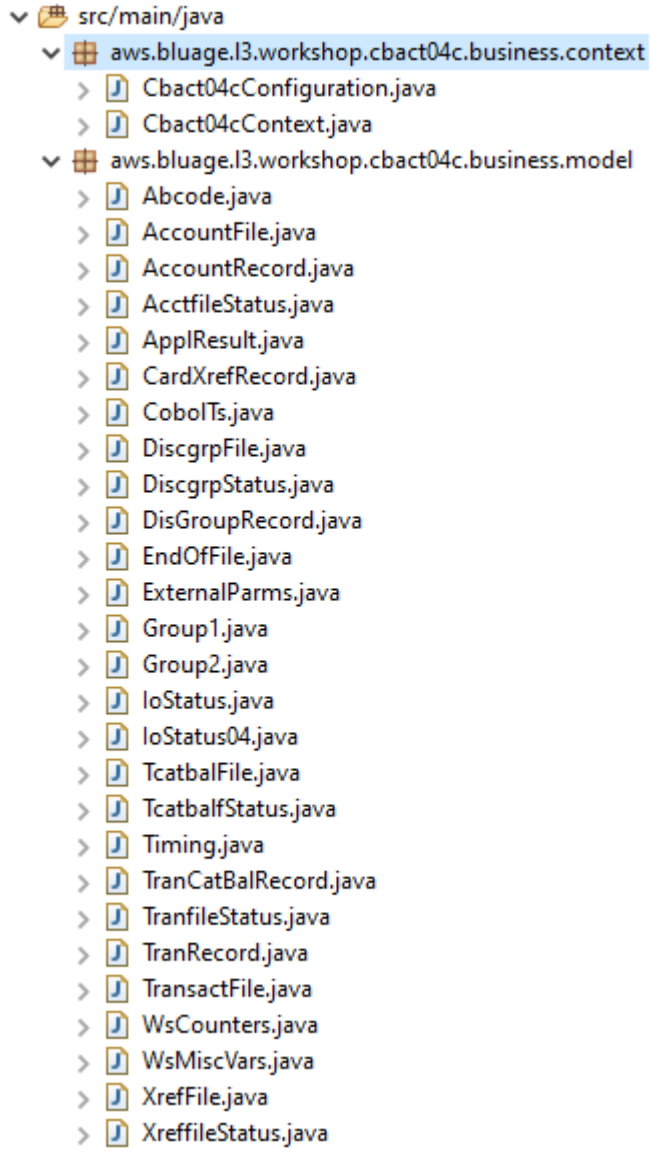
The basic organization relies on the following structure:

- 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 refactoring, 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.

Cbact04cConfiguration.java

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

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

/**
 * Creates Datasimplifier configuration for the Cbact04cContext context.
 */
@org.springframework.context.annotation.Configuration
@Lazy
public class Cbact04cConfiguration {

    @Bean(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.

Cbact04cContext.java

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

    @Autowired
    private TcatbalFile tcatbalFile;

    @Autowired
    private XrefFile xrefFile;

    @Autowired
    private DiscgrpFile discgrpFile;

    @Autowired
    private AccountFile accountFile;

    @Autowired
    private TransactFile transactFile;

    private IndexedFile tcatbalFileFile;

    private IndexedFile xrefFileFile;

    private IndexedFile discgrpFileFile;

    private IndexedFile accountFileFile;

    private SequentialFile transactFileFile;

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

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

  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 What are data simplifiers in AWS Blu Age.

DiscgrpFile.java

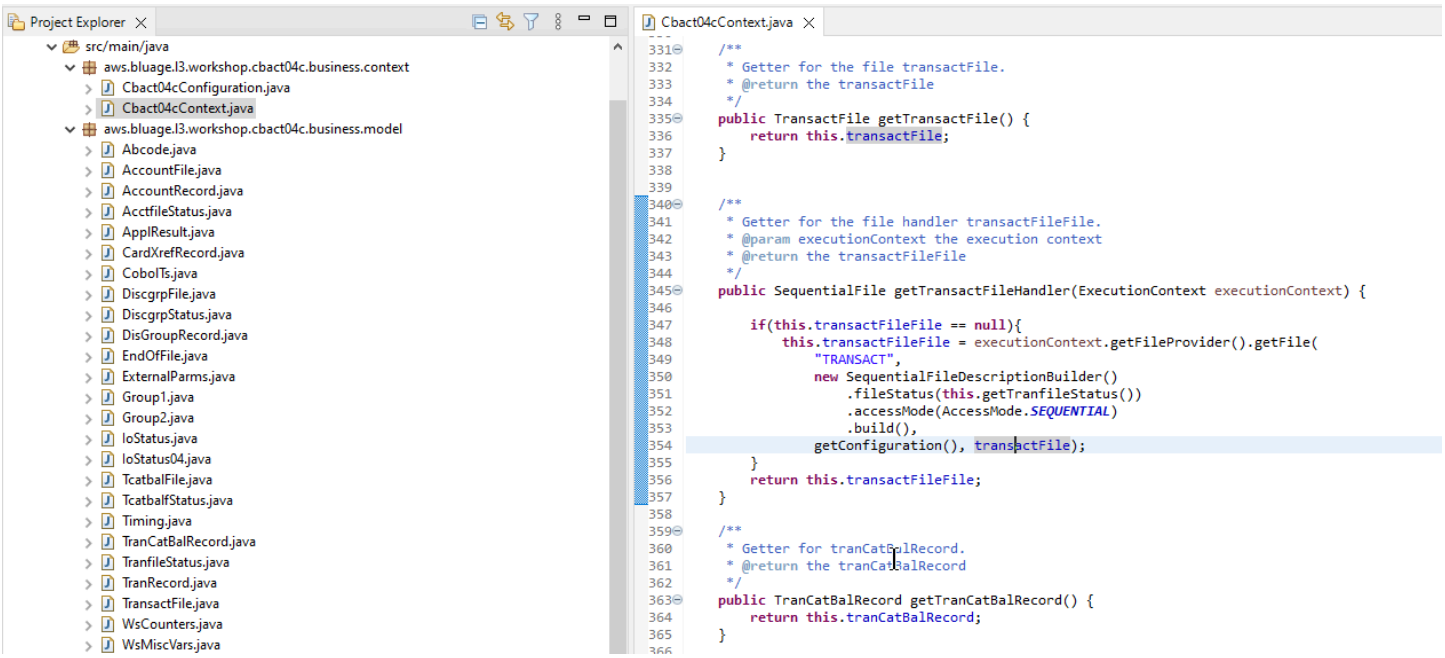
```java
package aws.bluage.l3.workshop.cbact04c.business.model;

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

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

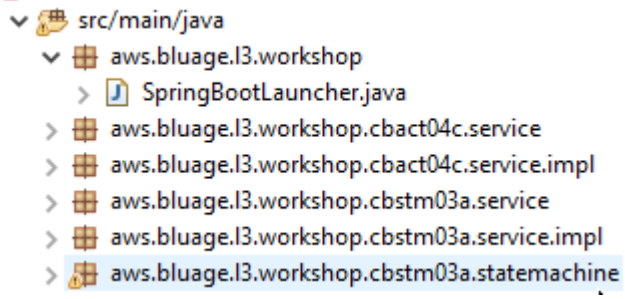
    private final Group root = new Group(getData());
    private final Group fdDiscgrpRec = new Group(root);
    private final Group fdDiscgrpKey = new Group(fdDiscgrpRec);
    private final Elementary fdDisAcctGroupId = new Elementary(fdDiscgrpKey,new AlphanumericType(10));
    private final Elementary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

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:



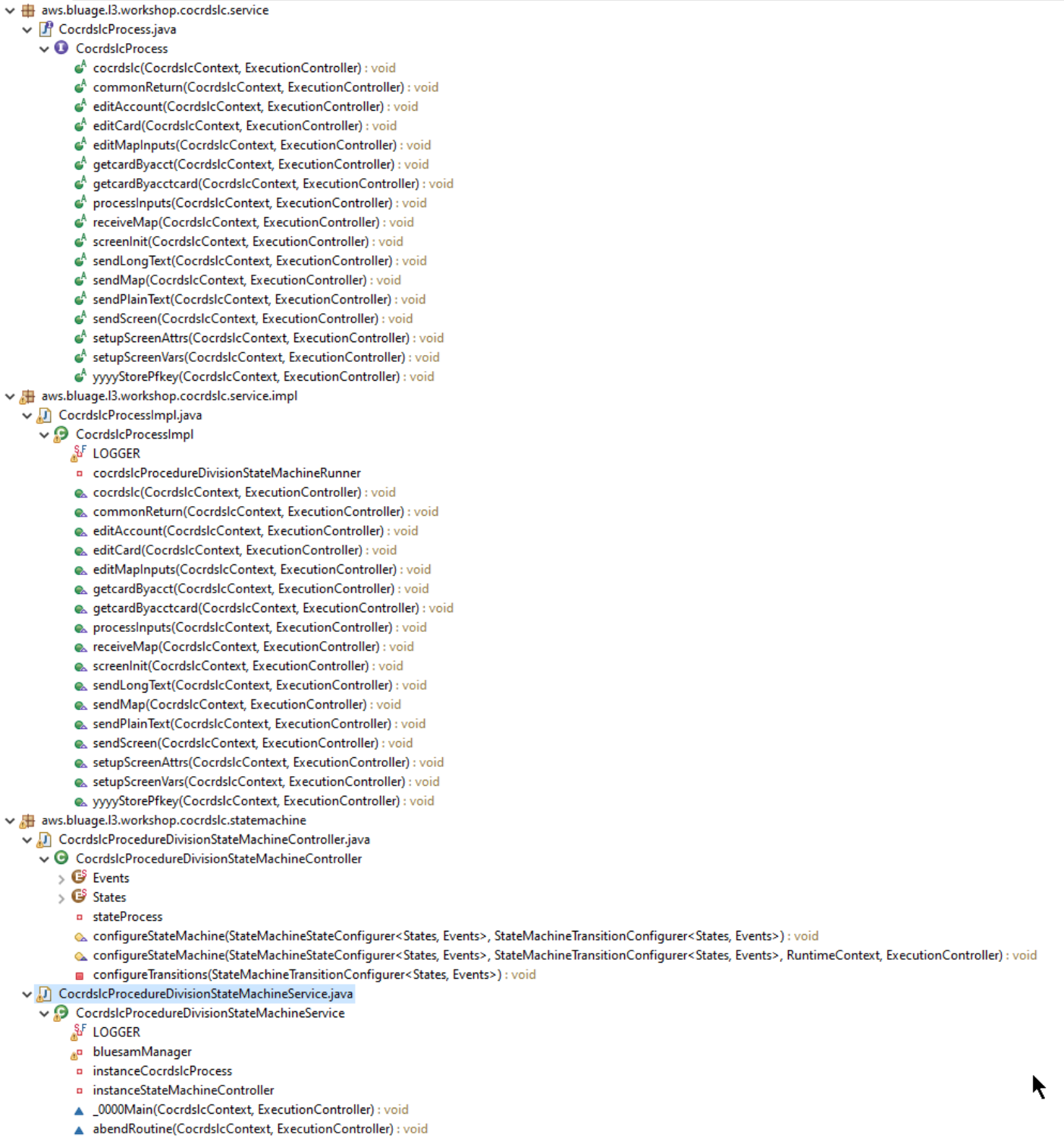
This class is notably responsible for:

- 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.

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

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 *Program*Process, which has business methods to handle the business logic, preserving the legacy execution control flow.

- *base package*.*program*.service.impl: contains a class named *Program*ProcessImpl, 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:

CocrdslcProcessImpl.java

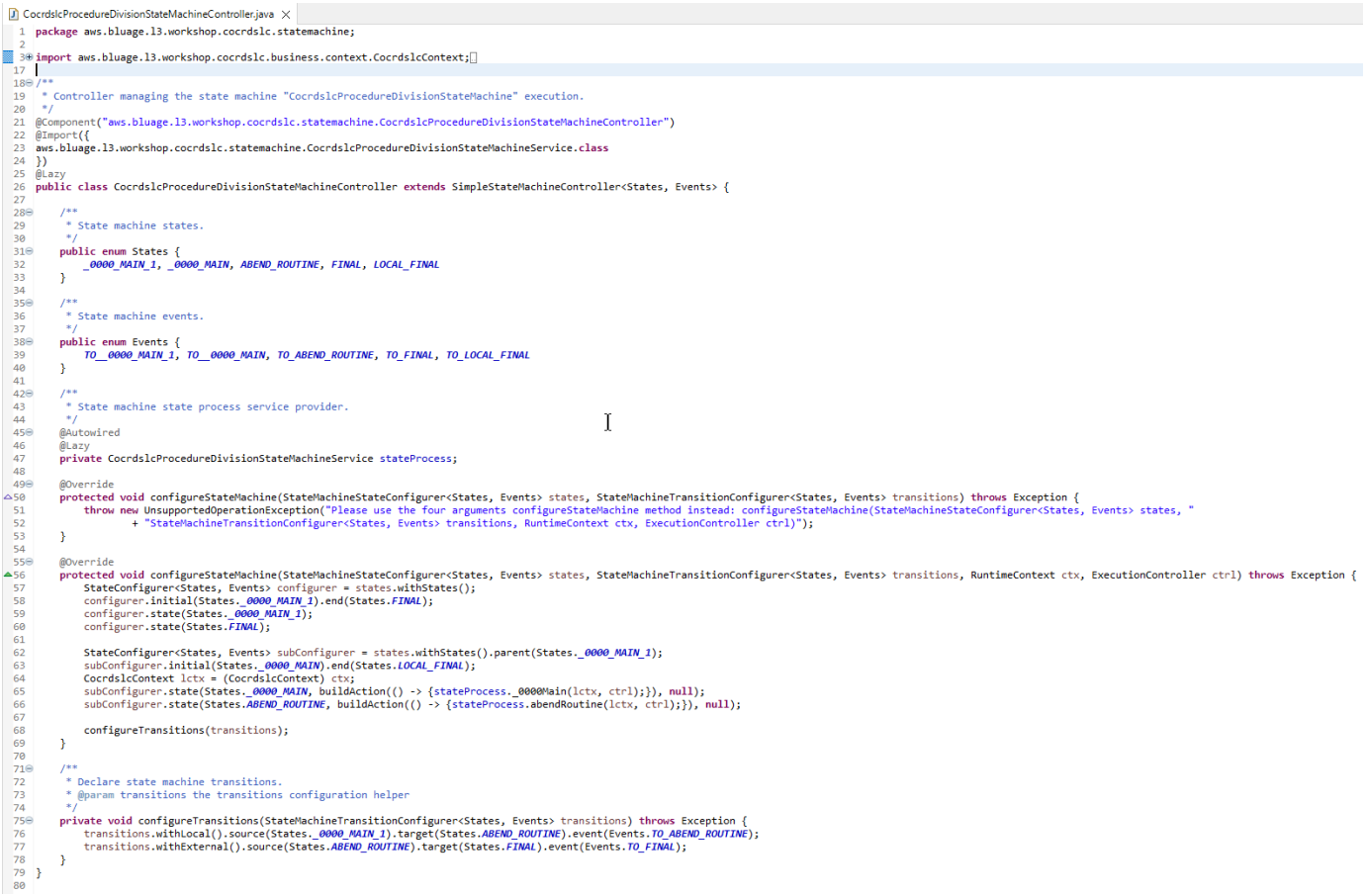
```
210     /**
211      * Process operation sendScreen.
212      *
213      * @param ctx
214      * @param ctrl
215      */
216     @Override
217     public void sendScreen(final CocrdslcContext ctx, final ExecutionController ctrl) {
218         ctx.getCcWorkAreas().setCcardNextMapset(ctx.getWsLiterals().getLitThismapset());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

- *base package*.*program*.statemachine: 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:

  - *Program*ProcedureDivisionStateMachineController: 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.



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

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

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

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

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

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

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

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

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

        configureTransitions(transitions);
    }

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

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:

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

- *Program*`ProcedureDivisionStateMachineService`: 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:

CocrdslcProcedureDivisionStateMachineService.java

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

CocrdslcProcedureDivisionStateMachineService.java

```
     *
     * @param ctx
     * @param ctrl
     */
    void abendRoutine(CocrdslcContext ctx, ExecutionController ctrl) {
        if (DataUtils.isLowValue(ctx.getAbendData().getAbendMsgReference())) {
            ctx.getAbendData().setAbendMsg("UNEXPECTED ABEND OCCURRED.");
        }
        ctx.getAbendData().setAbendCulprit(ctx.getWsLiterals().getLitThispgm());
        SendTextBuilder.newInstance(ctx.getDfheiblk(), ctx)
        .withData(ctx.getAbendData())
        .withLength(134)
        .execute();
        HandleAbendBuilder.newInstance(ctx.getDfheiblk(), ctx).cancel().execute().handleException();
        AbendBuilder.newInstance(ctx.getDfheiblk(), ctx).withAbendCode("9999").execute().handleException();

        /*
        Ver: CardDemo_v1.0-15-g27d6c6f-68 Date: 2022-07-19 23:12:33 CDT */
        instanceStateMachineController.sendEvent(Events.TO_FINAL);

    }
}
```

The statemachine service also makes calls to the process service implementation described previously:

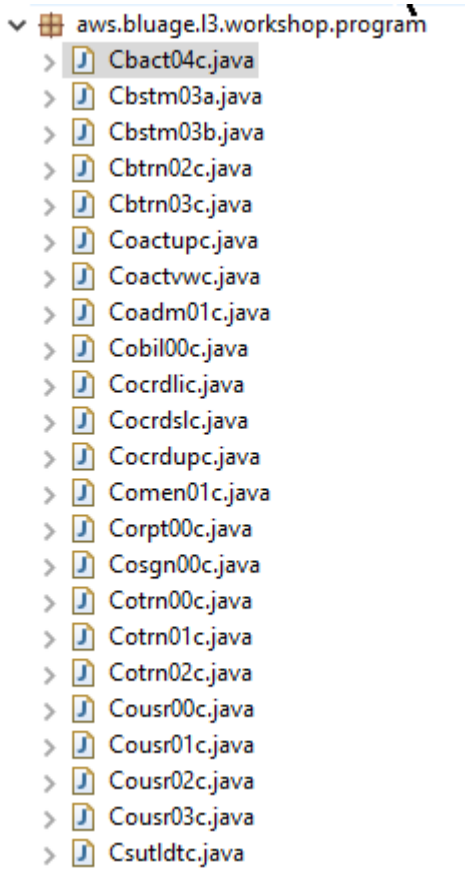
CocrdslcProcedureDivisionStateMachineService.java

```
        /*
        ************************************************************
        COMING FROM CREDIT CARD LIST SCREEN
        SELECTION CRITERIA ALREADY VALIDATED
        ************************************************************ */
    } else if (ctx.getCarddemoCommarea().isCdemoPgmEnter() && DataUtils.compare(ctx.getCarddemoCommarea().getCdemoFromProgramReference(), ctx.getWsLiterals().getLitCclistpgmReference()) == 0) {
        ctx.getWsMiscStorage().setInputOk(true);
        ctx.getCcWorkAreas().setCcAcctIdN(ctx.getCarddemoCommarea().getCdemoAcctId());
        ctx.getCcWorkAreas().setCcCardNumN(ctx.getCarddemoCommarea().getCdemoCardNum());
        instanceCocrdslcProcess.getcardByacctcard(ctx, ctrl);
        instanceCocrdslcProcess.sendMap(ctx, ctrl);
        instanceCocrdslcProcess.commonReturn(ctx, ctrl);
    } else if (ctx.getCarddemoCommarea().isCdemoPgmEnter()) {

        /*
        ************************************************************
        COMING FROM SOME OTHER CONTEXT
        SELECTION CRITERIA TO BE GATHERED
        ************************************************************ */
        instanceCocrdslcProcess.sendMap(ctx, ctrl);
        instanceCocrdslcProcess.commonReturn(ctx, ctrl);
    } else if (ctx.getCarddemoCommarea().isCdemoPgmReenter()) {
        instanceCocrdslcProcess.processInputs(ctx, ctrl);
        if (ctx.getWsMiscStorage().isInputError()) {
            instanceCocrdslcProcess.sendMap(ctx, ctrl);
            instanceCocrdslcProcess.commonReturn(ctx, ctrl);
        } else {
            instanceCocrdslcProcess.getcardByacctcard(ctx, ctrl);
            instanceCocrdslcProcess.sendMap(ctx, ctrl);
            instanceCocrdslcProcess.commonReturn(ctx, ctrl);
        }
    } else {
        ctx.getAbendData().setAbendCulprit(ctx.getWsLiterals().getLitThispgm());
        ctx.getAbendData().setAbendCode("0001");
        DataUtils.setToBlank(ctx.getAbendData().getAbendReasonReference());
        ctx.getWsMiscStorage().setWsReturnMsg("UNEXPECTED DATA SCENARIO");
        instanceCocrdslcProcess.sendPlainText(ctx, ctrl);
    }

    /*
    If we had an error setup error message that slipped through
    Display and return */
    if (ctx.getWsMiscStorage().isInputError()) {
        ctx.getCcWorkAreas().setCcardErrorMsg(ctx.getWsMiscStorage().getWsReturnMsg());
        instanceCocrdslcProcess.sendMap(ctx, ctrl);
        instanceCocrdslcProcess.commonReturn(ctx, ctrl);
    }
    instanceCocrdslcProcess.commonReturn(ctx, ctrl);
```

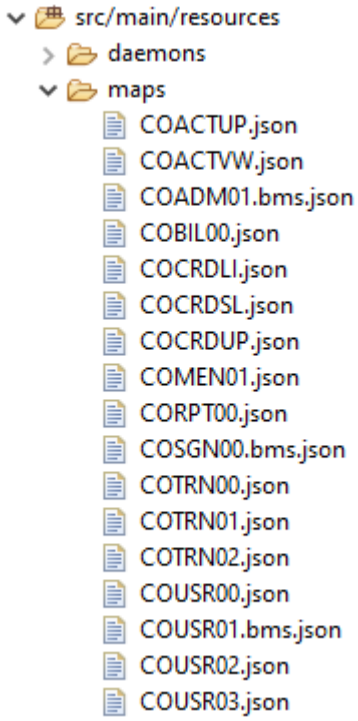
In addition to that, a package named *`base package`*`.program` plays a significant role, as it gathers one class per program, which will serve as the program entry point (more details about this later on). Each class implements the `Program` interface, marker for a program entry point.



Other artifacts

- BMS MAPs companions

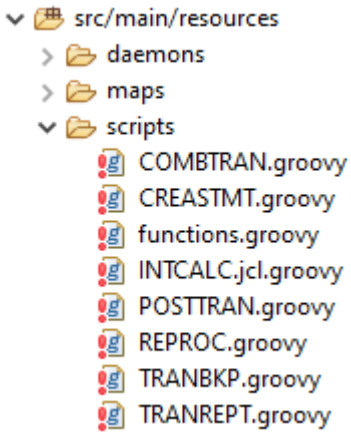
In addition to program related artifacts, the service project can contain other artifacts for various purposes. In the case of the modernization of a CICS online application, the modernization process produces a json file and puts in the map folder of the /src/main/resources folder:



The Blu Age runtime consumes those json files to bind the records used by the SEND MAP statement with the screen fields.

- Groovy Scripts

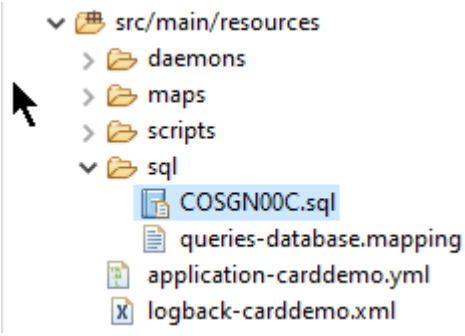
  If the legacy application had JCL scripts, those have been modernized as groovy scripts, stored in the /src/main/resources/scripts folder (more on that specific location later on):



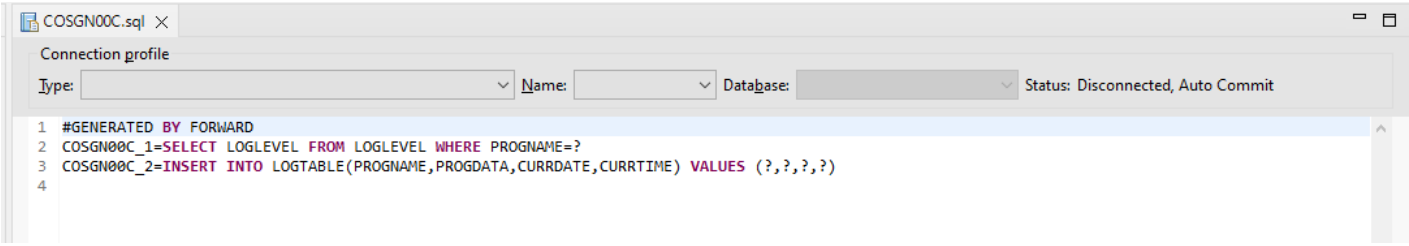
  Those scripts are used to launch batch jobs (dedicated, non-interactive, cpu-intensive data processing workloads).

- SQL files

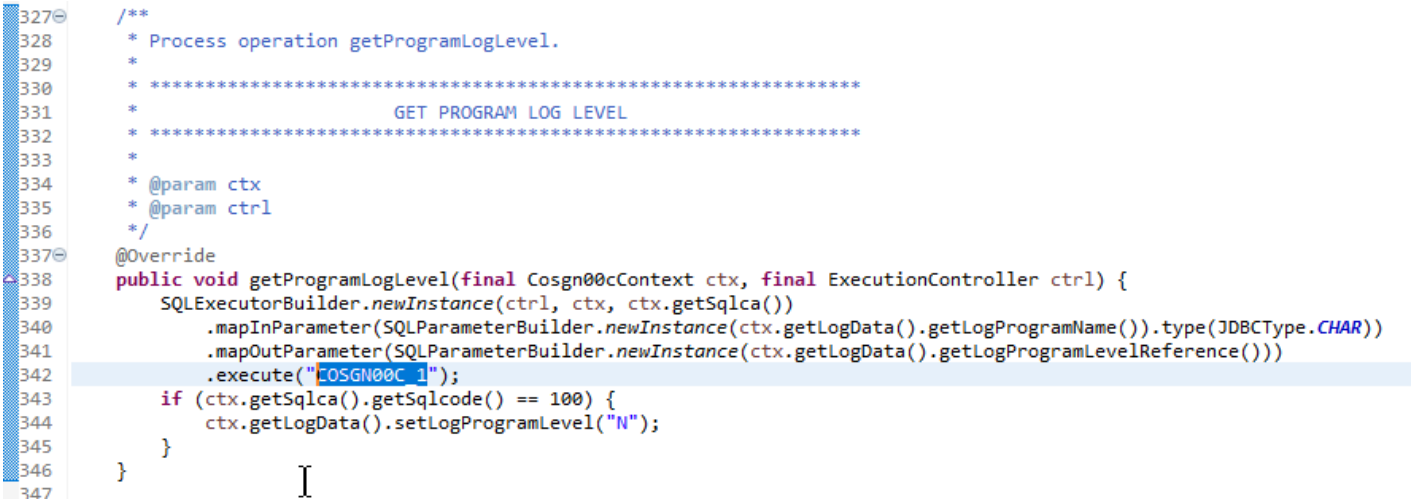
If the legacy application was using SQL queries, the corresponding modernized SQL queries have been gathered in dedicated properties files, with the naming pattern *program*.sql, where *program* is the name of the program using those queries.



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:

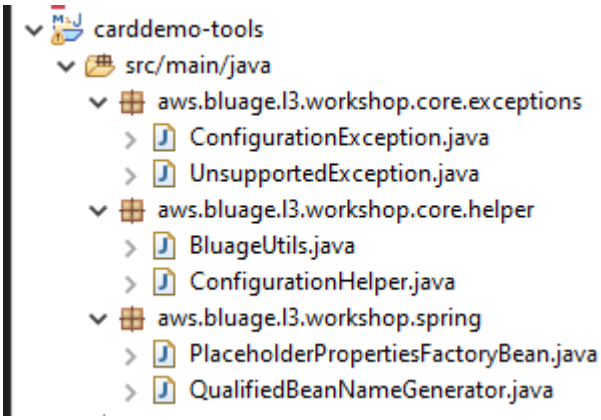


For instance, the COSGN00C program is executing the query with key "COSGN00C_1" (the first entry in the sql file):



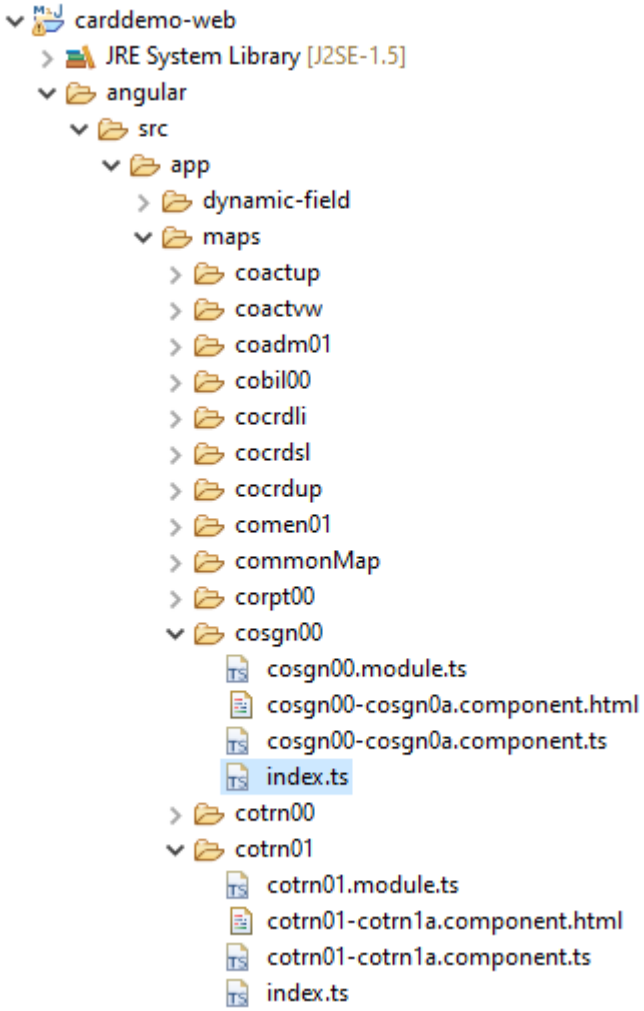
Utilities project contents

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 front end aspect of the application The service project, which relies on the utility and entities projects, provides the backend services. The link between the front end 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:

Cbact04c.java

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

import aws.bluage.l3.workshop.SpringBootLauncher;

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

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

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

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

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

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:

Cbstm03aProcessImpl.java

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

- 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 COBOL source file:

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

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

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 responsability 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:

REPROC.groovy

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

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:

REPROC.groovy

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

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.

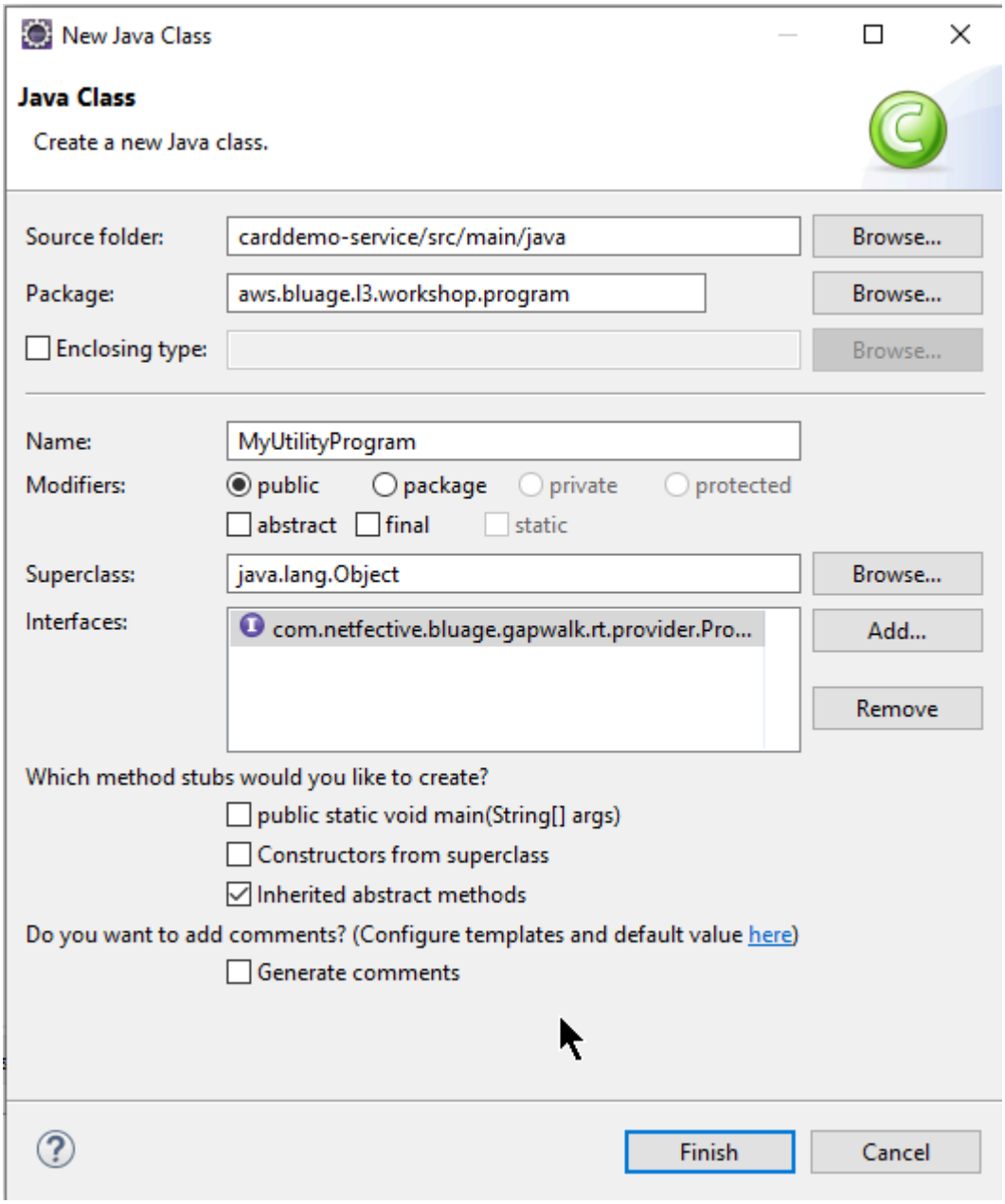
In that case, you might have to write the missing program, in java, by yourself (assuming you have enough knowledge about what the program expected behaviour should be, the memory layout of the program's arguments if any, and so on.) Your java program must comply with the program mechanics described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

MyUtilityProgram.java

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

import java.util.Set;

public class MyUtilityProgram implements Program {

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

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

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

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

    }

}
```

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

import aws.bluage.l3.workshop.SpringBootLauncher;

@Component
public class MyUtilityProgram implements Program {
```

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:

```
public class MyUtilityProgram implements Program {

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

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:

```
@Component
public class MyUtilityProgram implements Program {

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


    public ConfigurableApplicationContext getSpringApplication() {

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

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.

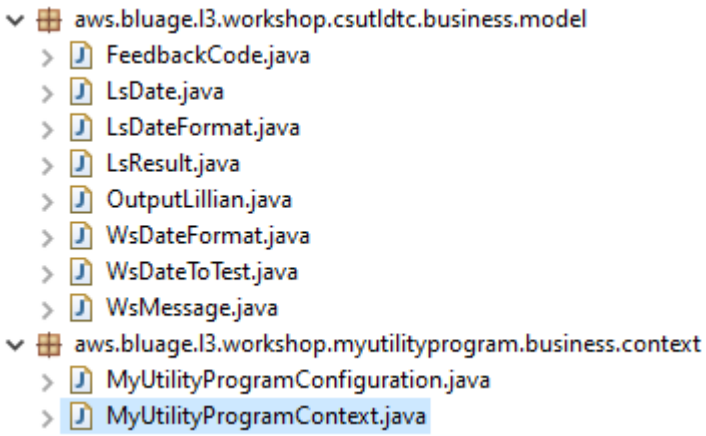
Referring to Program related classes, you can see that a program requires at least two companion classes:

- 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.

MyUtilityProgramConfiguration.java

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

import java.nio.charset.Charset;

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

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

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

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

```

Notes:

- General naming convention is *ProgramName*Configuration.
- It must use the @org.springframework.context.annotation.Configuration and @Lazy annotations.
- The bean name usually follows the *ProgramName*ContextConfiguration 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:

MyUtilityProgramContext.java

```java

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

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

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

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

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

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

}

```

Notes

- 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 *ProgramName*Context.
- 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 strcutures 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:

MyUtilityProgram.java

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

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

    @Autowired
    BeanFactory beanFactory;

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

    private static final String programIdentifier = "MYUTILPG";

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

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

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


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

```

Notes:

- The getContext method must be implemented strictly as shown, using a delegation to the getOrCreate method of the `ProgramContextStore` class and the auto wired 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).

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

Short name	Fully qualified name
`CallBuilder`	`com.netfective.bluage.gapwalk.runtime.statements.CallBuilder`
`Configuration`	`com.netfective.bluage.gapwalk.datasimplifier.configuration.Configuration`
`ConfigurationBuilder`	`com.netfective.bluage.gapwalk.datasimplifier.configuration.ConfigurationBuilder`

Short name	Fully qualified name
ExecutionController	com.netfective.bluage.gapwalk.rt.call.ExecutionController
ExecutionControllerImpl	com.netfective.bluage.gapwalk.rt.call.internal.ExecutionControllerImpl
File	com.netfective.bluage.gapwalk.rt.io.File
MainProgramRunner	com.netfective.bluage.gapwalk.rt.call.MainProgramRunner
Program	com.netfective.bluage.gapwalk.rt.provider.Program
ProgramContextStore	com.netfective.bluage.gapwalk.rt.context.ProgramContextStore
ProgramRegistry	com.netfective.bluage.gapwalk.rt.provider.ProgramRegistry
Record	com.netfective.bluage.gapwalk.datasimplifier.data.Record
RecordEntity	com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity
RuntimeContext	com.netfective.bluage.gapwalk.rt.context.RuntimeContext
SimpleStateMachineController	com.netfective.bluage.gapwalk.rt.statemachine.SimpleStateMachineController

Short name	Fully qualified name
StateMachineController	com.netfective.bluage.gapwalk.rt.statemachine.StateMachineController
StateMachineRunner	com.netfective.bluage.gapwalk.rt.statemachine.StateMachineRunner

Third party fully qualified name mappings

Short name	Fully qualified name
@Autowired	org.springframework.beans.factory.annotation.Autowired
@Bean	org.springframework.context.annotation.Bean
BeanFactory	org.springframework.beans.factory.BeanFactory
@Component	org.springframework.stereotype.Component
ConfigurableApplicationContext	org.springframework.context.ConfigurableApplicationContext
@Import	org.springframework.context.annotation.Import
@Lazy	org.springframework.context.annotation.Lazy

What are data simplifiers in AWS Blu Age

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 callers 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
- Data binding and access
- FQN of discussed Java types

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.

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:

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

```
public class Tst2 extends RecordEntity {

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

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

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

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

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

```
     */
    public ElementaryRangeReference getF2Reference() {
        return f2.getReference();
    }

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

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

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:

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

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 `index * (size in bytes of child)`. They are used to represent COBOL "OCCURS" constructs.

Primitives

In some modernization cases, "Primitives" may also be used to present independent, "root" data items. Those are very similar in use to `RecordEntity` but don't come 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 comm area, 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 instantiating 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.

Short Name	Fully Qualified Name
`Alphanumeric`	`com.netfective.bluage.gapwalk.datasimplifier.elementary.Alphanumeric`

Short Name	Fully Qualified Name
AlphanumericType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.AlphanumericType
BinaryType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.BinaryType
Bindable	com.netfective.bluage.gapwalk.datasimplifier.data.Bindable
Elementary	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Elementary
ElementaryRangeReference	com.netfective.bluage.gapwalk.datasimplifier.entity.ElementaryRangeReference
Filler	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Filler
Group	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Group
PackedType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.PackedType
Primitive	com.netfective.bluage.gapwalk.datasimplifier.elementary.Primitive

Short Name	Fully Qualified Name
RangeReference	com.netfective.bluage.gapwalk.datasimplifier.entity.RangeReference
RangeType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.RangeType
Record	com.netfective.bluage.gapwalk.datasimplifier.data.Record
RecordAdaptable	com.netfective.bluage.gapwalk.datasimplifier.data.RecordAdaptable
RecordEntity	com.netfective.bluage.gapwalk.datasimplifier.entity.RecordEntity
Repetition	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Repetition
Union	com.netfective.bluage.gapwalk.datasimplifier.data.structure.Union
ZonedDecimal	com.netfective.bluage.gapwalk.datasimplifier.elementary.ZonedDecimal
ZonedType	com.netfective.bluage.gapwalk.datasimplifier.metadata.type.ZonedType

AWS Blu Age Blusam

On mainframe systems (referred to in the following topic as "legacy"), business data is often stored using VSAM (Virtual Storage Access Method). The data is stored in "records" (byte arrays), belonging to a "data set".

There are four data set organizations:

- **KSDS**: Key-Sequenced data sets - records are indexed by a primary key (no duplicate keys allowed) and optionally, additional "alternate" keys. All key values are subsets of the record byte array, each key being defined by:
  - an offset (0-based, 0 being the start of the record byte array content, measured in bytes)
  - a length (expressed in bytes)
  - whether it tolerates duplicate values or not
- **ESDS**: Entry-Sequenced data sets - records are accessed mostly sequentially (based on their insertion order in the data set) but can be accessed using additional alternate keys;
- **RRDS**: Relative Records data sets - records are accessed using "jumps", using relative record numbers; The jumps can be done either forward or backward;
- **LDS**: Linear data sets - no records there, simply a stream of bytes, organized in pages. Mainly used for internal purposes on legacy platforms.

When modernizing legacy applications, using AWS Blu Age refactoring approach, modernized applications are no longer intended to access VSAM stored data, while preserving the data access logic. The Blusam component is the answer: it allows importing data from legacy VSAM data sets exports, provides an API for the modernized application to manipulate them along with a dedicated administration web application. See the section called “Blusam Administration Console”.

> **Note**
>
> Blusam only supports KSDS, ESDS, and RRDS.

The Blusam API makes it possible to preserve data access logic (sequential, random, and relative reads; insert, update, and delete records), whereas the components architecture, relying on a mix of caching strategies and RDBMS-based storage, permits high throughput I/O operations with limited resources.

Blusam infrastructure

Blusam relies on PostgreSQL RDBMS for data sets storage, both for raw records data and keys indexes (when applicable). The favorite option is to use the Amazon Aurora PostgreSQL compatible engine. The examples and illustrations in this topic are based on this engine.

> **Note**
>
> At server startup, the Blusam runtime checks for the presence of some mandatory technical tables and create them if they cannot be found. As a consequence, the role used in the configuration to access the Blusam database must be granted the rights to create, update, and delete the database tables (both rows and the tables definitions themselves). For information about how to disable Blusam, see the section called “Blusam configuration”.

Caching

In addition to the storage itself, Blusam operates faster when coupled to a cache implementation.

Two cache engines are currently supported, EhCache and Redis, each having its own use-case:

- EhCache : Standalone embedded volatile local cache
  - **NOT** eligible for AWS Mainframe Modernization managed environment deployment.
  - Typically used when a single node, like a single Apache Tomcat server, is used to run the modernized applications. For instance, the node might be dedicated to host batch jobs tasks.
  - **Volatile**: The EhCache cache instance is volatile; its content will be lost on the server shutdown.
  - **Embedded**: The EhCache and the server share the same JVM Memory Space (to be taken into account when defining the specifications for the hosting machine).
- Redis: Shared persistent cache
  - Eligible for AWS Mainframe Modernization managed environment deployment.
  - Typically used in multi-nodes situations, in particular when several servers are behind a load-balancer. The cache content is shared amongst all nodes.
  - The Redis is persistent and not bound to the nodes life-cycles. It is running on its own dedicated machine or service (for example, Amazon ElastiCache). The cache is remote to all nodes.

Locking

To deal with concurrent access to data sets and records, Blusam relies on a configurable locking system. Locking can be applied to both levels: data sets and records:

- Locking a data set for write purpose will prevent all others clients from performing write operations to it, at any level (data set or record).
- Locking at the record level for write will prevent other clients from performing write operations on the given record only.

Configuring the Blusam locking system should be done accordingly to the cache configuration:

- If EhCache is chosen as cache implementation, no further locking configuration is required as the default in-memory locking system should be used.
- If Redis is chosen as cache implementation, then a Redis-based locking configuration is required, to allow concurrent access from multiple nodes. The Redis cache used for locks does not have to be the same as the one used for data sets. For information about configuring a Redis-based locking system, see the section called “Blusam configuration”.

Blusam intrinsics and data migration from legacy

Storing data sets: records and indexes

Each legacy data set, when imported to Blusam, will be stored to a dedicated table; each row of the table represents a record, using two columns:

- The numeric ID column, big integer type, that is the table primary key, and is used to store the Relative Byte Address (RBA) of the record. The RBA represents the offset in bytes from the start of the data set, and begins at 0.
- The byte array record column, that is used to store the raw record's content.

See for example the content of a KSDS data set used in the CardDemo application:

```
1  SELECT * FROM public.aws_m2_carddemo_acctdata_vsam_ksds
2  ORDER BY id ASC
```

Data output | Messages | Notifications

	id [PK] bigint	record bytea
1	0	[binary data]
2	300	[binary data]
3	600	[binary data]
4	900	[binary data]
5	1200	[binary data]
6	1500	[binary data]
7	1800	[binary data]
8	2100	[binary data]
9	2400	[binary data]
10	2700	[binary data]
11	3000	[binary data]
12	3300	[binary data]
13	3600	[binary data]

- This particular data set has fixed length records, the length being 300 bytes (hence the collection of ids being multiples of 300).
- By default, the pgAdmin tool used to query PostgreSQL databases doesn't show byte array column contents, but prints a [binary data] label instead.
- The raw record content matches the raw data set export from the legacy, without any conversion. In particular, no character set conversion occurs; that implies that alphanumeric portions of the record will have to be decoded by modernized applications using the legacy character set, most likely an EBCDIC variant.

Regarding the data set metadata and keys indexes: each data set is associated with two rows in the table named `metadata`. This is the default naming convention. To learn how to customize it, see the section called “Blusam configuration”.

	name [PK] text	metadata oid
1	AWS_M2_CARDDEMO_ACCTDATA_VSAM_KSDS	66320
2	AWS_M2_CARDDEMO_ACCTDATA_VSAM_KSDS___internal	66321

- The first row has the data set name as the value of the *name* column. The *metadata* column is a binary column that contains a binary serialization of the general metadata of the given data set. For details, see the section called “General data set metadata attributes”.
- The second row has the data set name with the suffix `___internal'` as the value of the *name* column. The *metadata* column binary content depends on the "weight" of the data set.
  - For small/medium data sets, the content is a compressed serialization of:
    - definition of the keys used by the data set; the primary key definition (for KSDS) and alternate keys definitions if applicable (for KSDS / ESDS)
    - the key indexes if applicable (KSDS / ESDS with alternate keys definitions): used for indexed browsing of records; the key index maps a key value to the RBA of a record;
    - records length map: used for sequential / relative browsing of records;
  - For Large/Very Large data sets, the content is a compressed serialization of:
    - definition of the keys used by the data set; the primary key definition (for KSDS) and alternate keys definitions if applicable (for KSDS / ESDS)

Additionally, large/very large data sets indexes (if applicable) are stored using a pagination mechanism; index pages binary serializations are stored as rows of a dedicated table (one table per data set key). Each page of indexes is stored in a row, having the following columns:

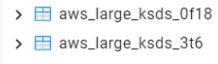
- id: technical identifier of the indexes page (numeric primary key);
- firstkey: binary value of the first (lowest) key value stored in the indexes page;
- lastkey: binary value of the last (highest) key value stored in the indexes page;
- metadata: binary compressed serialization of the indexes page (mapping key values to records RBAs).

	id [PK] bigint	firstkey bytea	lastkey bytea	metadata oid
1	1	[binary data]	[binary da...	6458928
2	2	[binary data]	[binary da...	6458929
3	3	[binary data]	[binary da...	6458930
4	4	[binary data]	[binary da...	6458931
5	5	[binary data]	[binary da...	6458932
6	6	[binary data]	[binary da...	6458933
7	7	[binary data]	[binary da...	6458934
8	8	[binary data]	[binary da...	6458935
9	9	[binary data]	[binary da...	6458936

The table name is a concatenation of the data set name and the key internal name, which contains information about the key, such as the key offset, whether the key accepts duplicates (set to true to allow duplicates), and the key length. For example, consider a data set named "AWS_LARGE_KSDS" that has the following two defined keys:

- primary key [offset: 0, duplicates: false, length:18]
- alternate key [offset: 3, duplicates: true, length: 6]

In this case, the following tables store the indexes related to the two keys.



Optimizing I/O throughput using write-behind mechanism

To optimize insert / update / delete operations performances, the Blusam engine relies on a configurable write-behind mechanism. The mechanism is built upon a pool of dedicated threads that deal with persistence operations using bulk update queries, to maximize I/O throughput towards the Blusam storage.

The Blusam engine collects all update operations done on records by applications and build records lots that are being dispatched for treatment to the dedicated threads. The lots are then being persisted to the Blusam storage, using bulk update queries, avoiding the usage of atomic persistence operations, ensuring the best possible usage of network bandwidth.

The mechanism uses a configurable delay (defaults to one second) and a configurable lot size (defaults to 10000 records). The build persistence queries are executed as soon as the first of the two following conditions is met:

- The configured delay has elapsed and the lot is not empty
- The number of records in the lot to be treated reaches the configured limit

To learn how to configure the write-behind mechanism, see the section called “Optional properties”.

Picking up the proper storage scheme

As shown in the previous section, the way data sets are being stored depends on their "weight". But what is considered as small, medium or large for a data set? When to pick the paginated storage strategy rather than the regular one?

The answer to that question depends on the following.

- The amount of available memory on each of the servers hosting the modernized applications that will use those data sets.
- The amount of available memory on cache infrastructure (if any).

When using non-paginated indexes storage scheme, the full key indexes and records sizes collections will be loaded into the server memory at data set opening time, for each data set. In addition, if caching is involved, all data set records might be pre-loaded into cache with the regular approach, which might lead to memory resource exhaustion on the cache infrastructure side.

Depending on the number of defined keys, the length of the key values, the number of records and the number of data sets opened at the same time, the amount of consumed memory can be roughly evaluated for the given known use-cases.

To learn more, see the section called “Estimating the memory footprint for a given data set”.

Blusam migration

Once the proper storage scheme has been selected for a given data set, the blusam storage must be populated by migrating legacy data sets.

To achieve this, one has to use **raw binary exports** of the legacy data sets, without any charset conversion being used during the export process. When transferring data set exports from the

legacy system, make sure not to corrupt the binary format. For example, enforce binary mode when using FTP.

The **raw binary exports** contain only the records. The import mechanism does not need the keys/indexes exports as all keys/indexes are being re-computed on the fly by the import mechanism.

Once a data set binary export is available, several options to migrate it to Blusam exist:

On AWS Mainframe Modernization managed environment:

- Import data sets by using the dedicated feature. See the section called “Import data sets for applications”.

or

- Use the data set bulk import facility. See the section called “Data set definition reference” and the section called “Sample data set request format for VSAM”.

or

- Use a groovy script to import data sets, using dedicated loading services.

> **Note**
>
> Importing LargeKSDS and LargeESDS on Mainframe Modernization managed environments is only possible using groovy scripts for now.

On AWS Blu Age Runtime on Amazon EC2:

- Import data set by using the the section called “Blusam Administration Console”.

or

- Use a groovy script to import data sets, using dedicated loading services.

Import data sets using Groovy scripts

This section will help you writing groovy scripts to import legacy data sets into Blusam.

It starts with some mandatory imports:

```
import com.netfective.bluage.gapwalk.bluesam.BluesamManager
import com.netfective.bluage.gapwalk.bluesam.metadata.Key;
import com.netfective.bluage.gapwalk.rt.provider.ServiceRegistry
import java.util.ArrayList; //used for alternate keys if any
```

After that, for each data set to import, the code is built upon the given pattern:

1. create or clear a map object
2. fill the map with required properties (this varies with data set kinds -- see below for details)
3. retrieve the proper loading service to be used for data set kind in the service registry
4. run the service, using the map as argument

There are 5 service implementations that can be retrieved from the service registry, using the following identifiers:

- `"BluesamKSDSFileLoader"`: for small/medium sized KSDS
- `"BluesamESDSFileLoader"` for small/medium sized ESDS
- `"BluesamRRDSFileLoader"`: for RRDS
- `"BluesamLargeKSDSFileLoader"`: for large KSDS
- `"BluesamLargeESDSFileLoader"`: for large ESDS

Whether to pick the regular vs large version of service for KSDS/ESDS depends on the size of the data sets and the storage strategy you want to apply for it. To learn how to pick the proper storage strategy, see the section called “Picking up the proper storage scheme”.

To be able to successfully import the data set into Blusam, the proper properties must be provided to the loading service.

Common properties:

- Mandatory (for all kinds of data sets)

- "bluesamManager" : expected value is `applicationContext.getBean(BluesamManager.class)`
- "datasetName" : name of the data set, as a String
- "inFilePath" : path to the legacy data set export, as a String
- "recordLength": the fixed record length or 0 for variable record length data set, as an integer

- Optional
  - **Not supported for Large data sets:**
    - "isAppend" : a boolean flag, indicating that the import is happening in append mode (appending records to an existing blusam data set).
    - "useCompression" : a boolean flag, indicating that compression will be used to store metadata.
  - **Only for Large data sets:**
    - "indexingPageSizeInMb" : the size in megabytes of each index page, for each of the keys of the data set, as a strictly positive integer

Data set kind dependant properties:

- KSDS/Large KSDS:
  - mandatory
    - "primaryKey" : the primary key definition, using a `com.netfective.bluage.gapwalk.bluesam.metadata.Key` constructor call.
  - optional:
    - "alternateKeys" : a List (`java.util.List`) of alternate key definitions, built using `com.netfective.bluage.gapwalk.bluesam.metadata.Key` constructor calls.
- ESDS/Large ESDS:
  - optional:
    - "alternateKeys" : a List (`java.util.List`) of alternate key definitions, built using `com.netfective.bluage.gapwalk.bluesam.metadata.Key` constructor calls.
- RRDS:
  - none.

Key constructor calls:

- `new Key(int offset, int length)`: creates a Key object, with given key attributes (offset and length) and no duplicates allowed. This variant should be used to define a primary key.
- `new Key(boolean allowDuplicates, int offset, int length)`: creates a Key object, with given key attributes (offset and length) and duplicates allowing flag.

The following Groovy samples illustrate various loading scenarios.

Loading a large KSDS, with two alternate keys:

```
import com.netfective.bluage.gapwalk.bluesam.BluesamManager
import com.netfective.bluage.gapwalk.bluesam.metadata.Key;
import com.netfective.bluage.gapwalk.rt.provider.ServiceRegistry
import java.util.ArrayList;

// Loading a large KSDS into Blusam
def map = [:]
map.put("bluesamManager", applicationContext.getBean(BluesamManager.class));
map.put("datasetName", "largeKsdsSample");
map.put("inFilePath", "/work/samples/largeKsdsSampleExport");
map.put("recordLength", 49);
map.put("primaryKey", new Key(0, 18));
ArrayList altKeys = [new Key(true, 10, 8), new Key(false, 0, 9)]
map.put("alternateKeys", altKeys);
map.put("indexingPageSizeInMb", 25);
def service = ServiceRegistry.getService("BluesamLargeKSDSFileLoader");
service.runService(map);
```

Loading a variable record length ESDS, with no alternate keys:

```
import com.netfective.bluage.gapwalk.bluesam.BluesamManager
import com.netfective.bluage.gapwalk.bluesam.metadata.Key;
import com.netfective.bluage.gapwalk.rt.provider.ServiceRegistry

// Loading an ESDS into Blusam
def map = [:]
map.put("bluesamManager", applicationContext.getBean(BluesamManager.class));
map.put("datasetName", "esdsSample");
map.put("inFilePath", "/work/samples/esdsSampleExport");
map.put("recordLength", 0);
def service = ServiceRegistry.getService("BluesamESDSFileLoader");
service.runService(map);
```

Variable record length data sets exports will contain the mandatory Record Decriptor Word (RDW) information to allow records splits at reading time.

Loading a fixed record length RRDS:

```
import com.netfective.bluage.gapwalk.bluesam.BluesamManager
import com.netfective.bluage.gapwalk.bluesam.metadata.Key;
import com.netfective.bluage.gapwalk.rt.provider.ServiceRegistry

// Loading a RRDS into Blusam
def map = [:]
map.put("bluesamManager", applicationContext.getBean(BluesamManager.class));
map.put("datasetName", "rrdsSample");
map.put("inFilePath", "/work/samples/rrdsSampleExport");
map.put("recordLength", 180);
def service = ServiceRegistry.getService("BluesamRRDSFileLoader");
service.runService(map);
```

Loading data sets in **Multi-schema mode**:

Multi-schema mode: In some legacy systems, VSAM files are organized into file sets, allowing programs to access, and modify data within specified partitions. Modern systems treat each file set as a schema, enabling similar data partitioning and access control.

To enable **Multi-schema mode** in the `application-main.yml` file refer to the section called "Blusam configuration". In this mode, data sets can be loaded into a specific schema using a Shared Context which is an in-memory registry for runtime information. To load a data set into a specific schema, prefix the data set name with the relevant schema name.

Loading a KSDS file into a specific schema for **Multi-schema mode**:

```
import com.netfective.bluage.gapwalk.bluesam.BluesamManager
import com.netfective.bluage.gapwalk.bluesam.metadata.Key;
import com.netfective.bluage.gapwalk.rt.provider.ServiceRegistry
import java.util.ArrayList;
import com.netfective.bluage.gapwalk.rt.shared.SharedContext;

// Loading a KSDS into Blusam
def map = [:]
String schema = "schema1";
String datasetName = schema+"|"+"ksdsSample";
```

```
SharedContext.get().setCurrentBlusamSchema(schema);
schema = SharedContext.get().getCurrentBlusamSchema();
map.put("bluesamManager", applicationContext.getBean(BluesamManager.class));
map.put("datasetName", datasetName);
map.put("inFilePath", "/work/samples/ksdsSampleExport");
map.put("recordLength", 49);
map.put("primaryKey", new Key(0, 18));
map.put("indexingPageSizeInMb", 25);
def service = ServiceRegistry.getService("BluesamKSDSFileLoader");
service.runService(map);
```

Loading a Large KSDS file into a specific schema for **Multi-schema mode**:

```
import com.netfective.bluage.gapwalk.bluesam.BluesamManager
import com.netfective.bluage.gapwalk.bluesam.metadata.Key;
import com.netfective.bluage.gapwalk.rt.provider.ServiceRegistry
import java.util.ArrayList;
import com.netfective.bluage.gapwalk.rt.shared.SharedContext;

// Loading a Large KSDS into Blusam
def map = [:]
String schema = "schema1";
String datasetName = schema+"|"+"largeKsdsSample";
SharedContext.get().setCurrentBlusamSchema(schema);
schema = SharedContext.get().getCurrentBlusamSchema();
map.put("bluesamManager", applicationContext.getBean(BluesamManager.class));
map.put("datasetName", datasetName);
map.put("inFilePath", "/work/samples/LargeKsdsSampleExport");
map.put("recordLength", 49);
map.put("primaryKey", new Key(0, 18));
map.put("indexingPageSizeInMb", 25);
def service = ServiceRegistry.getService("BluesamLargeKSDSFileLoader");
service.runService(map);
```

In addition, a configuration entry (to be set in the `application-main.yml` configuration file) can be used to fine tune the import process:

- `bluesam.fileLoading.commitInterval`: a strictly positive integer, definining the commit interval for regular ESDS/KSDS/RRDS import mechanism. **Does not apply to Large data sets imports.** Defaults to 100000.

Blusam configuration

Configuring Blusam happens in the `application-main.yml` configuration file (or in the `application-bac.yml` configuration file for the stand-alone deployment of the Blusam Administration Console -- BAC -- application).

Blusam has to be configured on two aspects:

- Blusam storage and caches access configuration
- Blusam engine configuration

Blusam storage and caches access configuration

For information about how to configure access to Blusam storage and caches using either secrets managers or datasources, see the section called “AWS Blu Age Runtime configuration”.

> **Note**
>
> Regarding the access to the Blusam storage, the credentials used will point to a connection role, with according privileges. For the Blusam engine be able to operate as expected, the connection role must have the following privileges:

- connect to the database
- create / delete / alter / truncate tables and views
- select / insert / delete / update rows in tables and views
- execute functions or procedures

Blusam engine configuration

Disabling Blusam support

First, let's mention that it is possible do completely disable Blusam support, by setting the `bluesam.disabled` property to `true`. An information message will be displayed in the server logs at application startup to remind Blusam disabling:

```
BLUESAM is disabled. No operations allowed.
```

No further configuration about Blusam is required in that case and any attempt to use Blusam related features (either programmatically or through REST calls) will raise an `UnsupportedOperationException` in the Java code execution, with a relevant explanation message about Blusam being disabled.

Blusam engine properties

The Blusam engine configuration properties are regrouped under the bluesam key prefix:

Mandatory properties

- `cache`: to be valued with the chosen cache implementation. Valid values are:
  - `ehcache`: For local embedded ehcache usage. See the related use case restrictions above.
  - `redis`: For shared remote redis cache usage. This is the preferred option for the AWS Mainframe Modernization managed use case.
  - none: To disable storage caching
- `persistence`: to be valued with pgsql (PostgreSQL engine: minimal version 10.0 – recommended version >=14.0
- datasource reference: `<persistence engine>.dataSource` will point to the dataSource definition for the connection to the Blusam storage, defined elsewhere in the configuration file. Commonly it's being named `bluesamDs` .

> **Note**
>
> Whenever Redis is used as cache mechanism, either for data or locks (see below), access to the Redis instances is to be configured. For details, see the section called “Available Redis cache properties”.

Optional properties

Blusam Locks: the properties are prefixed with `locks`

- `cache`: only usable value is `redis` , to specify that the redis-based locking mechanism will be used (to be used when blusam storage cache is redis-based as well). If the property is missing or not set to `redis` , the default in-memory locks mechanism will be used instead.

- `lockTimeOut`: a positive long integer value, giving the timeout expressed in milliseconds before an attempt to lock an already locked element is marked as failed. Defaults to `500` .
- `locksDeadTime`: a positive long integer value, representing the maximum time, expressed in milliseconds, an application can hold a lock. Locks are automatically marked as expired and released after that elapsed time. Defaults to `1000` ;
- `locksCheck`: a string, used to define the locking check strategy used by the current blusam lock manager, about expired locks removal. To be picked amongst the following values:
  - `off`: no checks are performed. Discouraged, as dead locks might happen.
  - `reboot`: checks are performed at reboot or application start time. All expired locks are released at that time. This is the default.
  - `timeout`: checks are performed at reboot or application start time, or when a timeout expires during an attempt to lock a data set. Expired locks are released immediately.

Write-behind mechanism: the properties are prefixed with `write-behind` key:

- `enabled`: `true` (default and recommended value) or `false` , to enable or disable the write-behind mechanism. Disabling the mechanism will greatly impact write performance and is discouraged.
- `maxDelay`: a maximal duration for the threads to be triggered. Defaults to `"1s"` (one second). Keeping the default value is generally a good idea, unless specific conditions require this value to be tuned. In any case the value should be kept low (under 3 seconds). The format for the delay string is: `<integer value><optional whitespace><time unit>` where `<time unit>` is to be picked amongst the following values:
  - `"ns"`: nanoseconds
  - `"µs"`: microseconds
  - `"ms"`: milliseconds
  - `"s"`: seconds
- `threads`: the number of dedicated write-behind threads. Default to `5` . You need to adjust this value according to the computing power of the host running the Blusam engine. It's not relevant to use a much higher value, hoping for performance increase as the limiting factor will become the storage RDBMS ability to deal with numerous concurrent batch queries. Recommended values are usually in the range 4-8.
- `batchSize`: a positive integer representing the maximal number of records in a lot that will be dispatched for bulk treatment to a thread. Its value must be between 1 and 32767. Defaults to

10000 . Using 1 as value defeats the purpose of the mechanism which is to avoid using atomic update queries; the suitable minimal value to use is around 1000 .

Embedded EhCache fine-tuning: the properties are prefixed with ehcache key:

- resource-pool:
  - size: allocated memory size for the embedded cache, expressed as a string. Defaults to "1024MB" (1 gigabyte). To be adjusted with regards to the available memory of the machine hosting the Blusam engine and the size of the datasets being used by the application. The format of the size string is: <integer value><optional whitespace><memory unit> where <memory-unit> is to be picked amongst the following values:
    - B: bytes
    - KB: kilobytes
    - MB: megabytes
    - GB: gigabytes
    - TB: terabytes
  - heap: true or false , to indicate whether the cache will consume JVM heap memory or not. Defaults to true (fastest option for cache performance, but cache storage consumes memory from the JVM on-heap RAM memory). Setting this property to false will switch to Off-Heap memory, which will be slower, due to required exchanges with the JVM heap.
- timeToLiveMillis: The duration (in Milliseconds) for which a cache entry remains in the cache before being considered expired and removed. If this property is not specified, cache entries will not automatically expire by default.

Multi-schema configuration properties

- multiSchema: false (default value) or true, to disable or enable the Multi-schema mode for Blusam - Available starting version 4.4.0.
- pgsql:
  - schemas: A list of schema names that the application will utilize in Multi-schema mode for Blusam.
  - fallbackSchema: The fallback schema name for use in Multi-schema mode. If a data set is not found in the current schema context, this schema will be used for Blusam-related operations on that data set.

Sample configuration snippet:

```
dataSource:
  bluesamDs:
    driver-class-name: org.postgresql.Driver
    ...
    ...
bluesam:
  locks:
    lockTimeOut: 700
  cache: ehcache
  persistence: pgsql
  ehcache:
    resource-pool:
      size: 8GB
  write-behind:
    enabled: true
    threads: 8
    batchsize: 5000
  pgsql:
    dataSource : bluesamDs
```

Sample configuration snippet (with Multi-schema mode enabled for Blusam):

```
dataSource:
  bluesamDs:
    driver-class-name: org.postgresql.Driver
    ...
    ...
bluesam:
  locks:
    lockTimeOut: 700
  cache: ehcache
  persistence: pgsql
  ehcache:
    resource-pool:
      size: 8GB
  write-behind:
    enabled: true
    threads: 8
    batchsize: 5000
  multiSchema: true
  pgsql:
```

```
    dataSource : bluesamDs
    schemas:
      - "schema1"
      - "schema2"
      - "schema3"
    fallbackSchema: schema3
```

> **Note**
>
> Blusam metadata schemas, including schemas listed in the `application-main.yml` file for Multi-schema mode, are created in the blusam database if they don't exist and the user has sufficient privileges.

Blusam Administration Console

The Blusam Administration Console (BAC) is a web-application, used to administrate the Blusam storage. For information about the BAC, see the section called “Blusam Administration Console”.

Appendix

General data set metadata attributes

General data set metadata serialization attributes list:

- name (of the data set)
- type (KSDS, LargeKSDS, ESDS, LargeESDS or RRDS)
- cache warm-up flag (whether the data set should be preloaded in cache at server startup or not)
- compression usage flag (whether to store records in a compressed or raw format)
- creation date
- last modification date
- fixed length record flag (whether the data set records are all having the same length or not)
- record length -- only meaningful for fixed record length
- page size (used to customize the paginated sql queries used to preload cache when required)
- size (size of the data set - cumulated length of the records)
- last offset (offset i.e. RBA of the latest record added to the data set)
- next offset (next avaliable offset for adding a new record to the data set)

- if meaningful, definition of the keys used by the data set; each key being defined by its kind (primary or part of the alternate keys collection) and three attributes:
  - offset : position in the record of the starting byte of the key value;
  - length : length in bytes of the key value. Thus the key value is the byte array which is the subset of the record starting at `key offset` and ending at position `key offset + length - 1`;
  - duplicates allowed flag: whether the key accepts duplicates or not (set to true to allow duplicates).

Estimating the memory footprint for a given data set

For small to medium sized data sets, the metadata (sizes and indexes for various keys) will be fully loaded into memory. Allocating proper resources for the machine hosting the server used to run modernized applications requires to figure out the memory consumption induced by the Blusam data sets, in particular regarding metadata. This section give practical answers to concerned operators.

The given formulas only apply to Blusam small to medium data sets, not using the "Large" storage strategy.

Blusam data set metadata

For a Blusam data set, metadata are split into two parts:

- core metadata : holds global information about the data set. The memory footprint of this can be considered as negligeable compared to the internal metadata.
- internal metadata: holds information about the records sizes and key indexes; when a data set is not empty, this is what consumes memory when loaded into the application server hosting modernized applications. The sections below detail how the consumed memory grows with the number of records.

Calculating Internal Metadata footprint

Records sizes map

First, the internal metadata stores a map to hold the size of every record (as an integer) given its RBA (relative byte address — stored as a long number).

The memory footprint of that data structure is, in bytes: `80 * number of records`

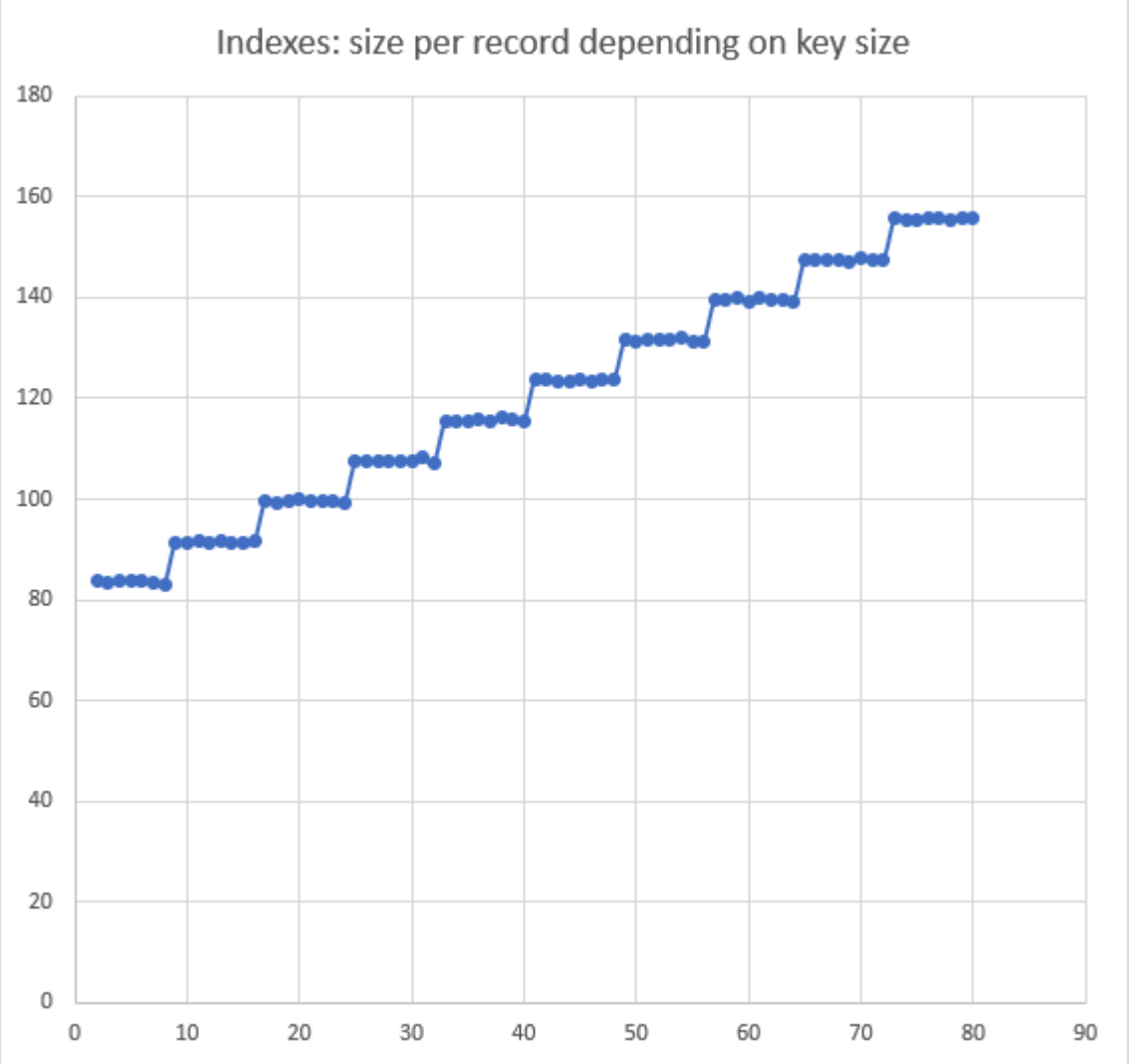
This applies to all data set kinds.

Indexes

Regarding the indexes for either the primary key of KSDS or alternate keys on both ESDS and KSDS, the calculation of the footprint depends on two factors:

- the number of records in the data set;
- the size of the key, in bytes.

The graphic below shows the size of the key index per record (y-axis) based on the size of the key (x-axis).



The corresponding formula for evaluating the footprint for a given key index of a data set is:

```
index footprint = number of records * ( 83 + 8 (key length / 8))
```

where ' / ' stands for the integer division.

Examples:

- data set 1:
  - number of records = 459 996
  - key length = 15 therefore (key length / 8) = 1
  - index footprint = 459 996 * (83 + (8*1)) = 41 859 636 bytes (= 39 MB approx.)
- data set 2:
  - number of records = 13 095 783
  - key length = 18 therefore (key length / 8) = 2
  - index footprint = 13 095 783 * (83 + (8*2)) = 1 296 482 517 bytes (= 1.2 GB approx.)

The total footprint for a given data set is the sum of all the footprints for all keys indexes and the footprint for the records sizes map.

For instance, taking the example data set 2, that has only a single key, the global footprint is:

- Records sizes map: 13 095 783 * 80 = 1 047 662 640 bytes
- Key Indexes : 1 296 482 517 bytes (see above)
- Total footprint = 2 344 145 157 bytes (= 2.18 GB approx.)

Available programs in utility web application

Utility web application provide support for various legacy platform utility programs, such as IDCAMS, INFUTILB, SORT, and so on. To configure access to the application, see Configure access to utilities for managed applications.

List of programs

- JCLBCICS utility - Used by batch to set bluesam dataset status to open/enabled or closed/disabled.

JCLBCICS utility

JCLBCICS is a JCL utility program designed to set bluesam dataset to open/enabled or closed/disabled. An open/enabled status will block access to the dataset from batch programs while a closed/disabled status makes the dataset unavailable for accessing to JICS online services.

Usage

- JCLBCICS changes STATUS column in Jics FILE_TABLE table and OPEN_STATUS column in Bluesam BLUESAM_STATUS table based on groovy configuration on DD name.

```
.open(ddName) -> ENABLED in Jics FILE_TABLE table, OPEN in Bluesam BLUESAM_STATUS
 table
.close(ddName) -> DISABLED in Jics FILE_TABLE table, CLOSED in Bluesam BLUESAM_STATUS
 table
```

- DD name size is configurable globally in `application-utility-pgm.yml` configuration file.

```
jclbcics.ddname.size: 7
```

- Global DD name size can be overridden in individual step by providing the overridden size with the following lines in groovy, then use stepParams as parameters for that step.

```
TreeMap stepMapTransfo = [:]
Map stepParams = ["MapTransfo":stepMapTransfo]
stepParams["MapTransfo"]["JCLBCICS_OVERRIDDEN_SIZE"] = '7'
...
.withParameters(stepParams)
.runProgram("JCLBCICS")
```

- When setting DD name size, max effective DD name size is 8.
- If ddName length is greater than the provided DD name size, it will be truncated from the end to match the DD name size.
- Wild card is supported in ddName If * (asterisk) is appended to the end of ddName or ddName length is less than 8.

```
.open("DTSNAME*")
```

Example code

```
// DD name with overridden size of 7 bytes
 def stepSTEP007(Object shell, Map params, Map programResults) {
   shell.with {
     if (checkValidProgramResults(programResults)) {
       TreeMap stepMapTransfo = [:]
       Map stepParams = ["MapTransfo":stepMapTransfo]
       stepParams["MapTransfo"]["JCLBCICS_OVERRIDDEN_SIZE"] = '7'
       return execStep("STEP007", "JCLBCICS", programResults, {
         mpr
           .withDatasetsConfiguration(new DatasetsConfiguration()
           .close("DTSNAME"))
           .withParameters(stepParams)
           .runProgram("JCLBCICS")
        })
     }
   }
}
```

AWS Blu Age Blusam Administration Console

The Blusam Administration Console (BAC) is a secure web-application for handling Blusam data sets. This guide covers the BAC user interface. For remote management through REST endpoints, see the section called “Blusam application console REST endpoints ”.

Topics

- Deploying the BAC
- Using the BAC
- LISTCAT JSON format

Deploying the BAC

The BAC is available as a secured single web application, using the web-archive format (.war). It is intended to be deployed alongside the BluAge Gapwalk-Application, in an Apache Tomcat application server, but can also be deployed as a standalone application. The BAC inherits the access to the Blusam storage from the Gapwalk-Application configuration if present.

The BAC has its own dedicated configuration file, named `application-bac.yml`. For configuration details, see the section called “BAC dedicated configuration file”.

The BAC is secured. For details about security configuration, see the section called "Configuring security for the BAC".

BAC dedicated configuration file

Standalone deployment: If the BAC is deployed alone the Gapwalk-Application, the connection to the Blusam storage must be configured in the application-bac.yml configuration file.

Default values for data sets configuration used to browse data set records must be set in the configuration file. See the section called "Browsing records from a data set". The records browsing page can use an optional mask mechanism that makes it possible to show a structured view on a record's content. Some properties impact the records view when masks are used.

The following configurable properties must set in the configuration file. The BAC application does not assume any default value for these properties.

Key	Type	Description
`bac.crud.limit`	integer	A positive integer value giving the maximum number of records returned when browsing records. Using `0` means *unlimited*. Recommend ed value: `10` (then adjust the value data set by data set on the browsing page, to fit your needs).
`bac.crud.encoding`	string	The default character set name, used to decode records bytes as alphanume ric content. The provided charset name must be java compatible (please see the java documentation for supported charsets). Recommended value: the legacy charset used on the legacy platform where data

Key	Type	Description
		sets are coming from; this will be an EBCDIC variant most of the times.
`bac.crud.initCharacter`	string	The default character (byte) used to init data items. Two special values can be used: `"LOW-VALUE"` , the 0x00 byte (recommended value) and `"HI-VALUE"` , the 0xFF byte. Used when masks are applied.
`bac.crud.defaultCharacter`	string	The default character (byte), as a one character string, used for padding records (on the right). Recommended value: `" "` (space). Used when masks are applied.
`bac.crud.blankCharacter`	string	The default character (byte), as a one character string, used to represent blanks in records.Recommended value: `" "` (space). Used when masks are applied.
`bac.crud.strictZoned`	boolean	A flag to indicate which zoned mode is used for the record. If `true`, the Strict zone mode will be used; if `false`, the Modified zoned mode will be used. Recommended value: `true`. Used when masks are applied.

Key	Type	Description
bac.crud.decimalSeparator	string	The character used as decimal separator in numeric edited fields (used when masks are applied).
bac.crud.currencySign	string	The default character, as a one character string, used to represent currency in numeric edited fields, when formatting is applied (used when masks are applied).
bac.crud.pictureCurrencySign	string	The default character, as a one character string, used to represent currency in numeric edited fields pictures (used when masks are applied).

The following sample is a configuration file snippet.

```
bac.crud.limit: 10
bac.crud.encoding: ascii
bac.crud.initCharacter: "LOW-VALUE"
bac.crud.defaultCharacter: " "
bac.crud.blankCharacter: " "
bac.crud.strictZoned: true
bac.crud.decimalSeparator: "."
bac.crud.currencySign: "$"
bac.crud.pictureCurrencySign: "$"
```

Configuring security for the BAC

Configuring security for the BAC relies on the mechanisms detailed in this documentation page. The authentication scheme is OAuth2, and configuration details for Amazon Cognito or Keycloak are provided.

While general setup can be applied, some specifics about the BAC need to be detailed here. The access to the BAC features is protected using a role-based policy and relies on the following roles.

- ROLE_USER:
  - Basic user role
  - No import, export, creation, or deletion of data sets allowed
  - No control over caching policies
  - No administration features allowed
- ROLE_ADMIN:
  - Inherits ROLE_USER permissions
  - All data set operations allowed
  - Caching policies administration allowed

Installing the masks

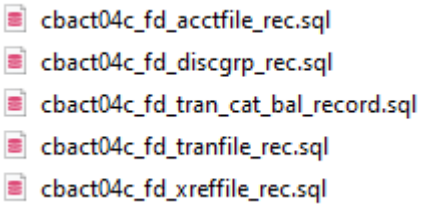
In Blusam storage, data sets records are stored in a byte array column in the database, for versatility and performance considerations. Having access to a structured view, using fields, of the business records, based on application point of view is a convenient feature of the BAC. This relies on the SQL masks produced during the BluAge driven modernization process.

For the SQL masks to be generated, please make sure to set the relevant option (`export.SQL.masks`) in the configuration of the BluInsights Transformation Center to true:



The masks are part of the modernization artifacts that can be downloaded from BluInsights for a given project. They are SQL scripts, organized by modernized programs, giving the applicative point of view on data sets records.

For example, using the AWS CardDemo sample application, you can find in the downloaded artifacts from the modernization result of this application, the following SQL masks for the program CBACT04C.cbl:



Each SQL mask name is the concatenation of the program name and the record structure name for a given data set within the program.

For example, looking at the [CBACT04C.cbl program, the given file control entry:

```
    FILE-CONTROL.
        SELECT TCATBAL-FILE ASSIGN TO TCATBALF
               ORGANIZATION IS INDEXED
               ACCESS MODE  IS SEQUENTIAL
               RECORD KEY   IS FD-TRAN-CAT-KEY
               FILE STATUS  IS TCATBALF-STATUS.
```

is associated with the given FD record definition

```
       FILE SECTION.
       FD  TCATBAL-FILE.
       01  FD-TRAN-CAT-BAL-RECORD.
           05 FD-TRAN-CAT-KEY.
              10 FD-TRANCAT-ACCT-ID             PIC 9(11).
              10 FD-TRANCAT-TYPE-CD             PIC X(02).
              10 FD-TRANCAT-CD                  PIC 9(04).
           05 FD-FD-TRAN-CAT-DATA               PIC X(33).
```

The matching SQL mask named `cbact04c_fd_tran_cat_bal_record.SQL` is the mask that gives the point of view of the program CBACT04C.cbl on the FD record named `FD-TRAN-CAT-BAL-RECORD`.

Its content is:

```
-- Generated by Blu Age Velocity
-- Mask : cbact04c_fd_tran_cat_bal_record

INSERT INTO mask (name, length) VALUES ('cbact04c_fd_tran_cat_bal_record', 50);
  INSERT INTO mask_item (name, c_offset, length, skip, type, options, mask_fk) VALUES
 ('fd_trancat_acct_id', 1, 11, false, 'zoned', 'integerSize=11!fractionalSize=0!
signed=false', (SELECT MAX(id) FROM mask));
  INSERT INTO mask_item (name, c_offset, length, skip, type, options, mask_fk) VALUES
 ('fd_trancat_type_cd', 12, 2, false, 'alphanumeric', 'length=2', (SELECT MAX(id) FROM
 mask));
  INSERT INTO mask_item (name, c_offset, length, skip, type, options, mask_fk)
 VALUES ('fd_trancat_cd', 14, 4, false, 'zoned', 'integerSize=4!fractionalSize=0!
signed=false', (SELECT MAX(id) FROM mask));
  INSERT INTO mask_item (name, c_offset, length, skip, type, options, mask_fk) VALUES
 ('fd_fd_tran_cat_data', 18, 33, false, 'alphanumeric', 'length=33', (SELECT MAX(id)
 FROM mask));
```

Masks are stored in the Blusam storage using two tables:

- mask: used to identify masks. The columns of the mas table are:
  - name: used to store mask identification (used as primary key, so must be unique)
  - length: size in bytes of the record mask
- mask_item: used to store mask details. Every elementary field from a FD record definition will produce a row in the mask_item table, with details on how to interpret the given record part. The columns of the mask_item table are:
  - name: name of the record field, based on the elementary name, using lowercase and replacing dash with underscore
  - c_offset: 1-based offset of the record sub-part, used for the field content
  - length: length in bytes of the record sub-part, used for the field content
  - skip: flag to indicate whether the given record part should be skipped or not, in the view presentation
  - type: the field kind (based on its legacy picture clause)
  - options: additional type options -- type-dependant
  - mask_fk: reference to the mask identifier to attach this item to

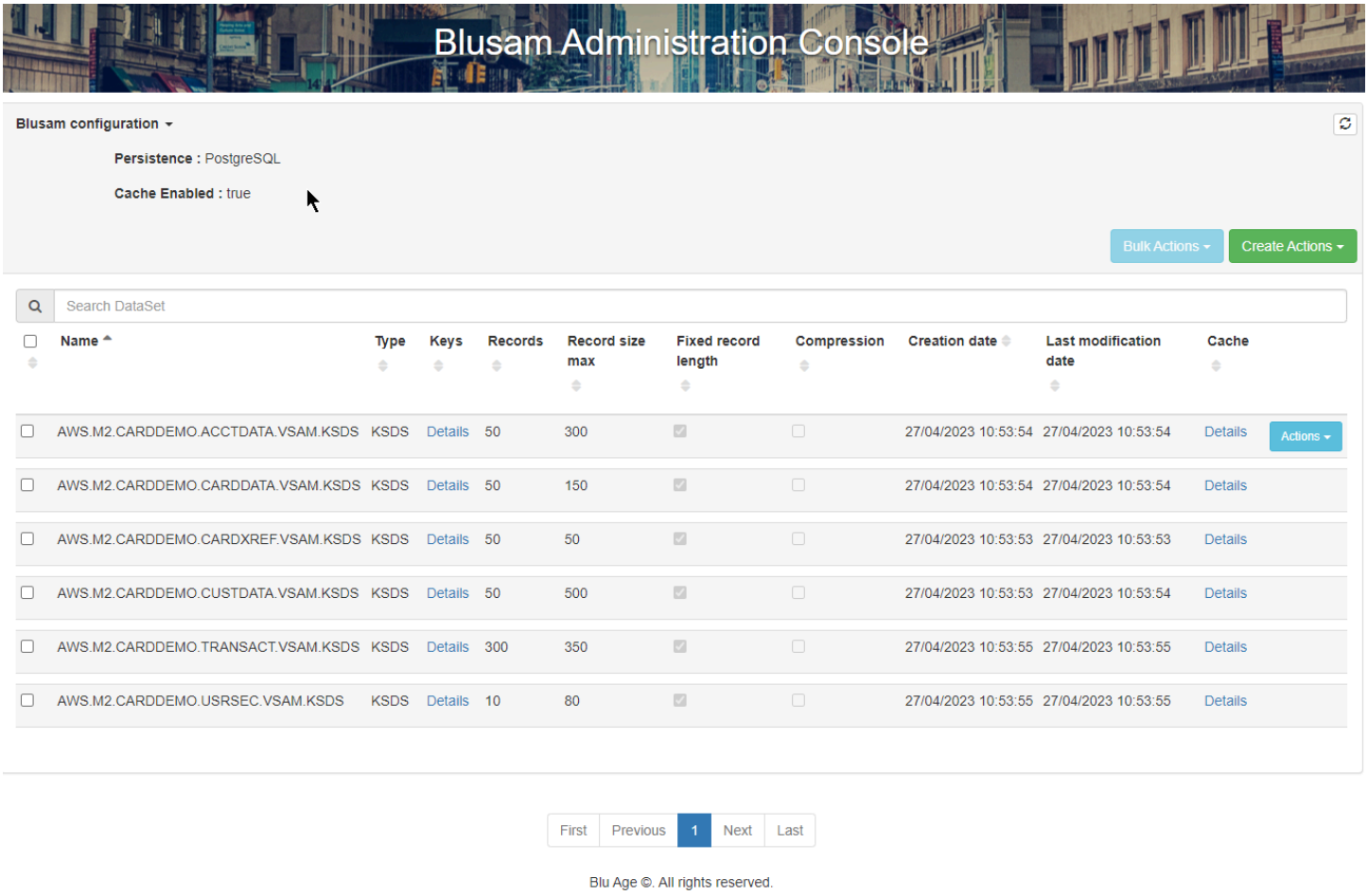
Note the following:

- SQL masks represent a point of view from a program on records from a data set: several programs might have a different point of view on a given data set; only install the masks that you find relevant for your purpose.
- A SQL mask can also represent the point of view from a program based on a 01 data structure from the WORKING STORAGE section, not only from a FD record. The SQL masks are organized into sub-folders according to their nature:
  - FD record based masks will be located in the sub-folder named `file`
  - 01 data structure based masks will be located in the sub-folder named `working`

  While FD records definitions always match the record content from a data set, 01 data structures might not be aligned or might only represent a subset from a data set record. Before you use them, inspect the code and understands the possible shortcomings.

Using the BAC

Because the BAC is secured and delivers permissions to use features based on the user role, the first step to access the application is to authenticate yourself. After the authentication step, you'll be redirected to the home page. The home page presents the paginated list of data sets found in the Blusam storage:



To return to the home page with the data sets listing, choose the Blu Age logo in the upper left corner of any page of the application. The following image shows the logo.



The foldable header, labelled "BluSam configuration", contains information about the used BluSam storage configuration:

- `Persistence`: the persistent storage engine (PostgreSQL)
- `Cache Enabled`: whether the storage cache is enabled

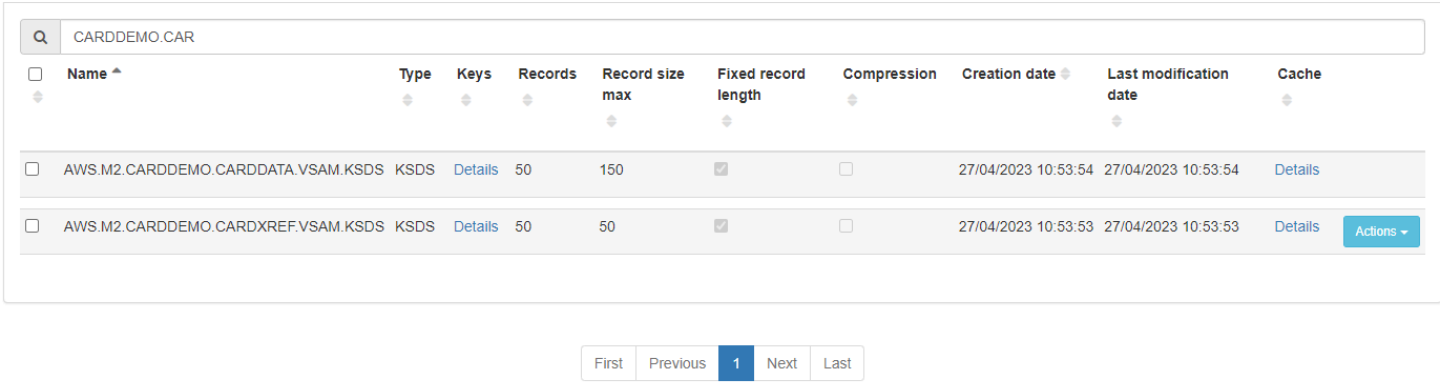
On the right side of the header, two drop-down lists, each one listing operations related to data sets:

- **Bulk actions**
- **Create actions**

To learn about the detailed contents of these lists, see the section called "Existing data set operations".

The **Bulk Actions** button is disabled when no data set selection has been made.

You can use the search field to filer the list based on the data sets names:



The paginated list that follows shows one data set per table row, with the following columns:

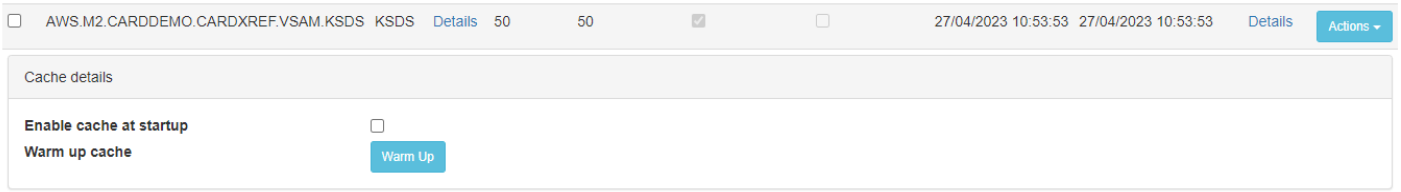
- Selection checkbox: A checkbox to select the current data set.
- Name: The name of the data set.
- Type: The type of the data set, one of the following:
  - KSDS
  - ESDS
  - RRDS
- Keys: A link to show or hide details about the keys (if any). For example, the given KSDS has the mandatory primary key and one alternative key.



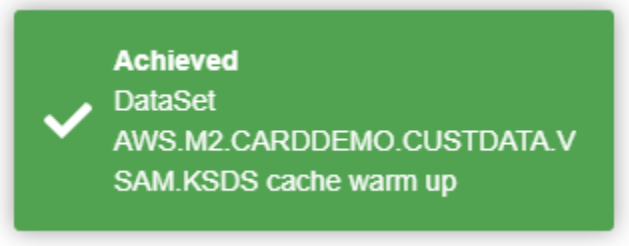
  There is one row per key, with the following columns. None of the fields are editable.

  - Key nature: either a primary key or an alternative key
  - Name: the name of the key
  - Unique: whether the key accepts duplicate entries

  - Offset: offset of the key start within the record
  - Length: length in bytes of the key portion in the record
- Records: The total number of records in the data set.
- Record size max: The maximal size for records, expressed in bytes.
- Fixed record length: A checkbox that indicates whether the records are fixed length (selected) or variable length (unselected).
- Compression: A checkbox that indicates whether compression is applied (selected) or not (unselected) to stored indexes.
- Creation date: The date when the data set was created in the Blusam storage.
- Last modification date: The date when the data set was last updated in the Blusam storage.
- Cache: A link to show or hide details about the caching strategy applied to this dataset.



  - Enable cache at startup: A checkbox to specify the startup caching strategy for this data set. If selected, the data set will be loaded into cache at startup time.
  - Warm up cache: A button to load the given data set into cache, starting immediately (but hydrating the cache takes some time, depending on the data set size and number of keys). After the data set gets loaded into cache, a notification like the following one appears.

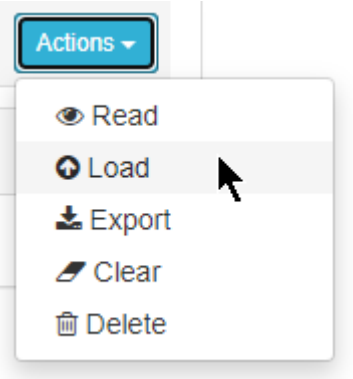


- Actions: A drop-down list of possible data sets operations. For details, see the section called "Existing data set operations".

At the bottom of the page, there is a regular paginated navigation widget for browsing through the pages of the list of data sets.

Existing data set operations

For each data set in the paginated list, there is an **Actions** drop-down list with the following content:



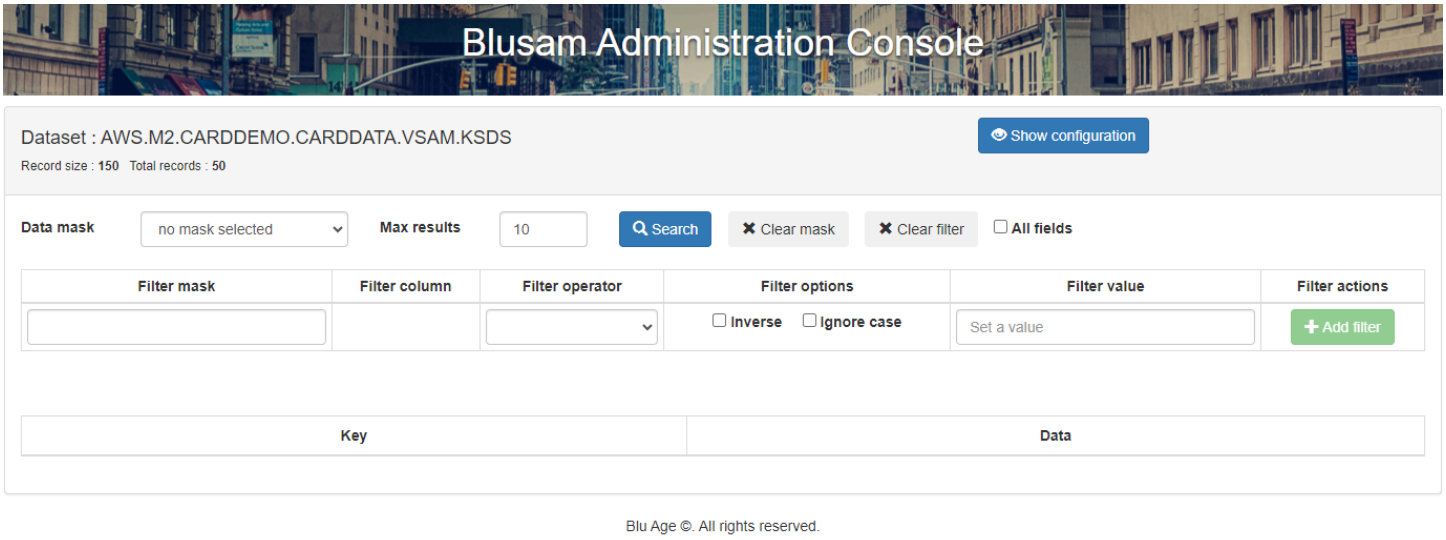
Each item in the list is an active link that makes it possible to perform the specified action on the data set:

- Read: browse records from the data sets
- Load: import records from a legacy data set file
- Export: export records to a flat file (compatible with legacy systems)
- Clear: remove all records from the data set
- Delete: remove the data set from the storage

Details for each action are provided in the following sections.

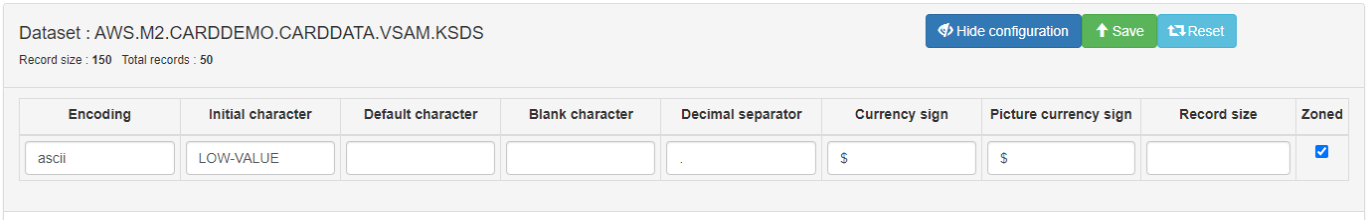
Browsing records from a data set

When you choose the **Read** action for a given data set, you get the following page.



The page is made of:

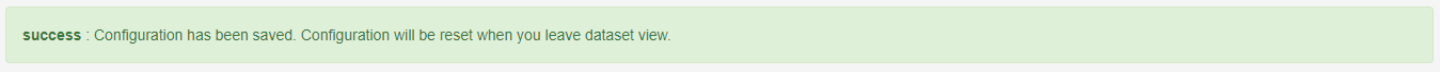
- a header, with:
  - Dataset: the data set name
  - Record size: the fixed record length, expressed in bytes
  - Total Records: the total number of records stored for this data set
  - Show configuration button (on the right side): a toggle button to show/hide the data set configuration. At first, the configuration is hidden. When using the button, the configuration you see the configuration, as shown in the following image.

    When configuration is shown, two new buttons: Save and Reset, used respectively to:
    - save the configuration for this data set and current work session
    - reset the configuration to default values for all fields.
  - A list of configurable properties to tailor the browsing experience for the given data set.

The configurable properties match the configuration properties described in the section called "BAC dedicated configuration file". Refer to that section to understand the meaning of each column and applicable values. Each value can be redefined here for the data set and saved for the

work session (using the Save button). After you save the configuration, a banner similar to the one shown in the following image appears.



The banner states that the work session ends when you leave the current page.

There is an extra configurable property that is not documented in the configuration section: Record size. This is used to specify a given record size, expressed in bytes, that will filter the applicable masks to this data set: only masks whose total length matches the given record size will be listed in the Data mask drop-down list.

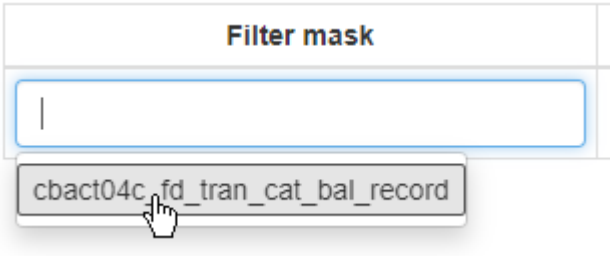
Retrieving records from the data set is triggered by the Search button, using all options and filters nearby.

First line of options:

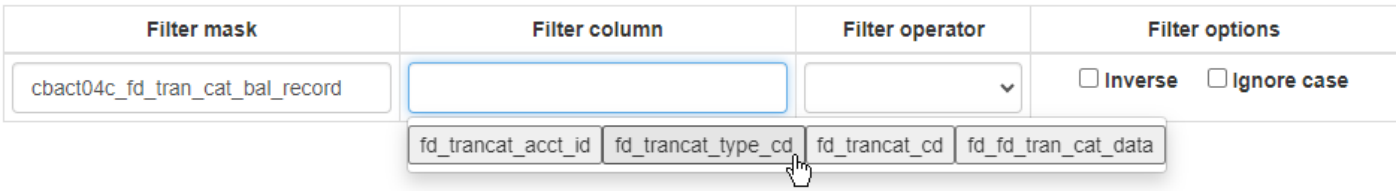
- the Data mask drop-down list show applicable masks (respecting the record size). Please note that, matching the record size is not enough to be an effective applicable mask. The mask definition must also be compatible with the records contents. The Data mask picked here has
- Max results: limits the number of records retrieved by the search. Set to 0 for unlimited (paginated) results from the data set.
- Search button: launch the records retrieval using filters and options
- Clear mask button: will clear the used mask if any and switch back the results page to a raw key/data presentation.
- Clear filter button: will clear the used filter(s) if any and update the results page accordingly.
- All fields toggle: When selected, mask items defined with `skip = true` are shown anyway, otherwise mask items with `skip = true` are hidden.

Next lines of filters: It is possible to define a list of filters, based on the usage of filtering conditions applied to fields (columns) from a given mask, as shown in the following image.

- Filter mask: The name of the mask to pick the filtering column from. When you choose the field, the list of applicable masks appears. You can choose the mask you want from that list.

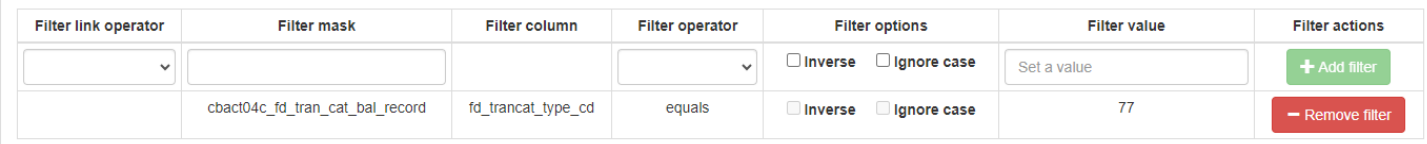


- Filter column: The name of the field (column) from the mask, used to filter records. When you choose the field, the list of mask columns appears. To fill the **Filter column** field, choose the desired cell.



- Filter operator: An operator to apply to the selected column. The following operators are available.
  - equals to: the column value for the record must be equal to the filter value
  - starts with: the column value for the record must start with the filter value
  - ends with: the column value for the record must end with the filter value
  - contains: the column value for the record must contain the filter value
- Filter options:
  - Inverse: apply the inverse condition for the filter operator; for instance, 'equals to' is replaced by 'not equals to';
  - Ignore case: ignore case on alphanumeric comparisons for the filter operator
- Filter value: The value used for comparison by the filter operator with the filter column.

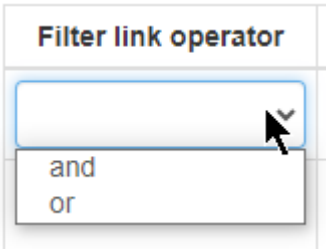
Once the minimal number of filter items are set (at least: Filter mask, filter column, filter operator and Filter value must be set), the Add Filter button is enabled, and clicking on it creates a new filter condition on the retrieved records. Another empty filter condition row is added at the top and the added filter condition has a Remove filter button that can be used to suppress the given filter condition:



When you launch the search, the filtered results appear in a paginated table.

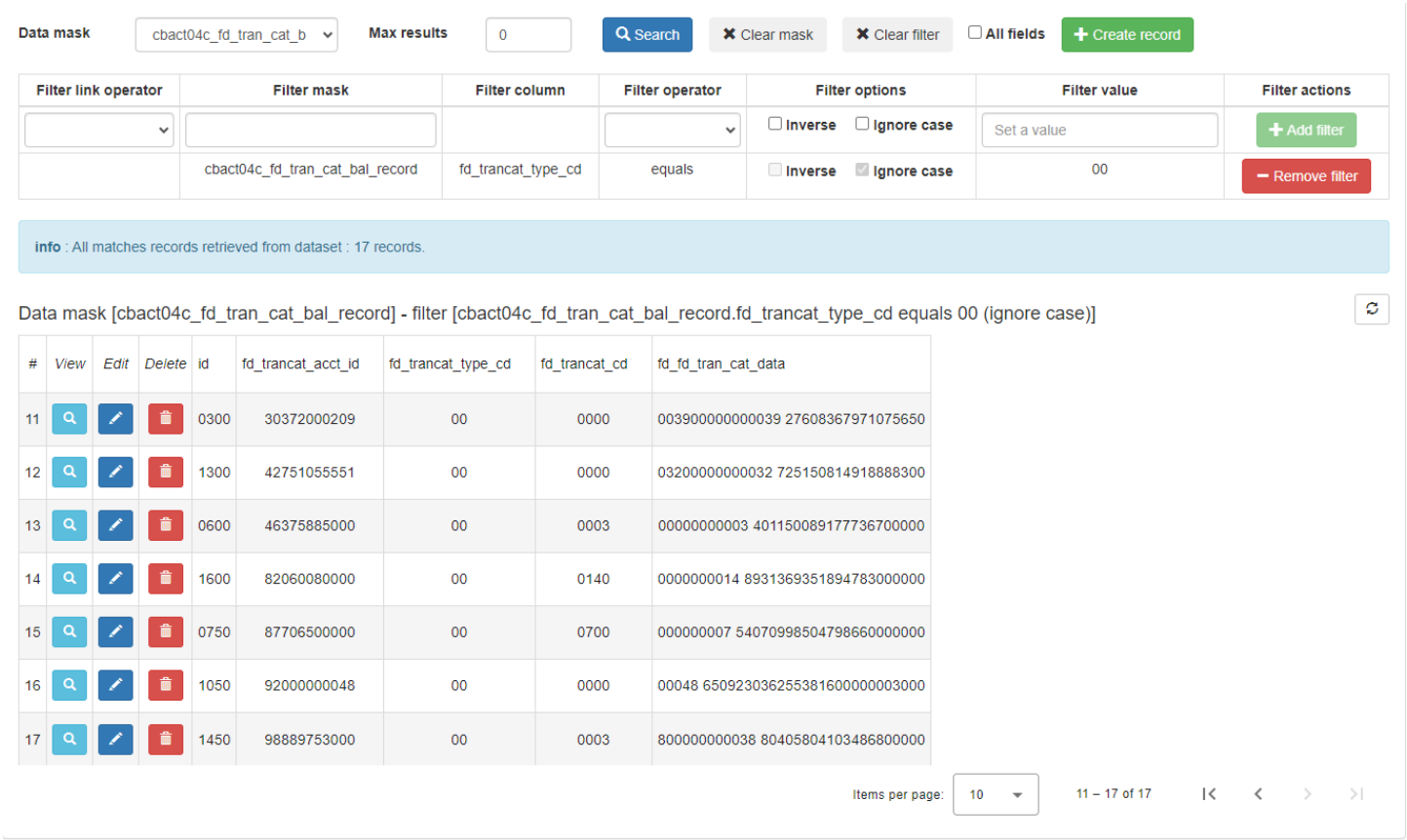
Note

- Successive filters are linked by an **and** or an **or**. Every new filter definition starts by setting the link operator, as shown in the following image.



- There might not be any records that match the given filter conditions.

Otherwise, the results table looks like the one in the following image.



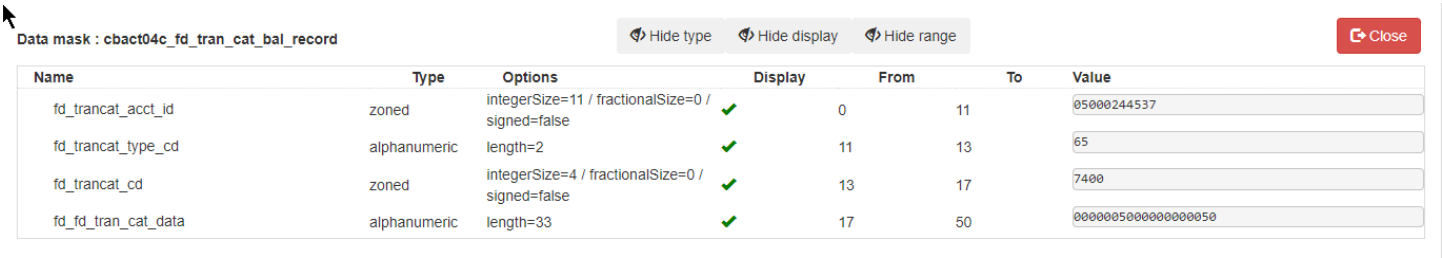
A header indicates the total number of records that match the filter conditions. After the header, you see the following.

- Reminder of the used data mask (if any) and the filter conditions.
- A refresh button that you can use to trigger the refresh of the whole results table with latest values from the Blusam storage (as it might have been updated by another user for instance).

For each retrieved record, the table has a row that shows the result of applying the data mask to the records' contents. Each column is the interpretation of the record sub-portion according to the column's type (and using the selected encoding). To the left of each row, there are three buttons:

- a magnifying glass button: leads to a dedicated page showing the detailed record's contents
- a pen button: leads to a dedicated edit page for the record's contents:
- a trashcan button: used to delete the given record from the blusam storage

Viewing the record's contents in detail:



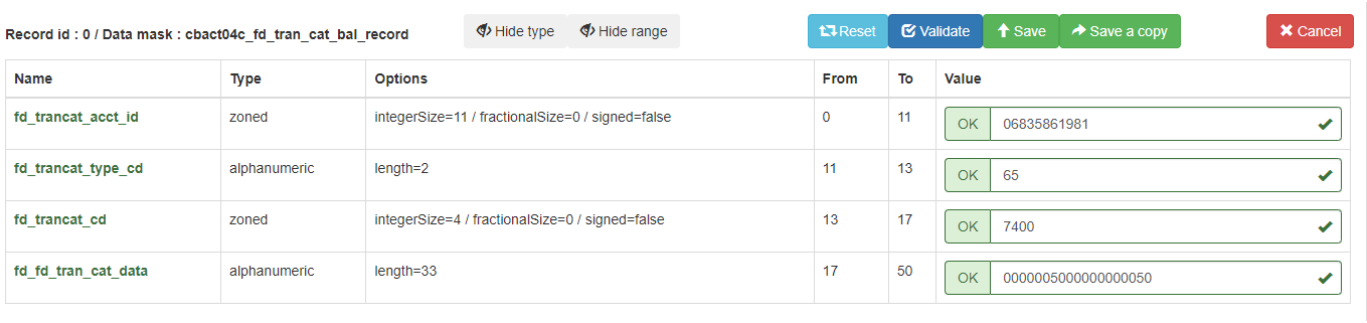
- Three toggle buttons for hiding or showing some columns:
  - Hide/show the type
  - Hide/show the display flag
  - Hide/show the range
- To leave this dedicated page and go back to the results table, choose **Close**.
- Each row represents a column from the data mask, with the following columns:
  - Name: the column's name
  - Type: the column's type
  - Display: the display indicator; a green check will be displayed if the matching mask item is defined with `skip = false`, otherwise a red cross will be displayed
  - From & To: the 0-based range for the record sub-portion
  - Value: the interpreted value of the record sub-portion, using type and encoding

Editing the record's contents:

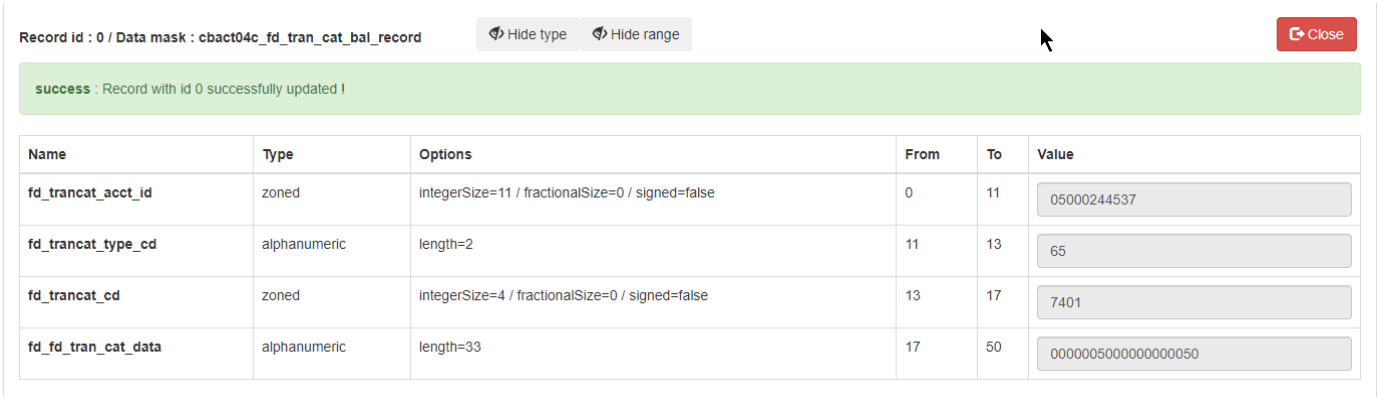


The editing page is similar to the view page described above, except that the mask items values are editable. Three buttons control the update process:

- Reset: resets the editable values to the initial record values (prior to any edition);
- Validate: validates the input, with regards to the mask item type. For each mask item, the result of the validation will be printed using visual labels (0K and checkbox if validation succeeded, ERR0R and red cross if validation failed, alongside an error message giving hints about the validation failure). If the validation succeeded, two new buttons will appear:
  - Save: attempt to update the existing record into Blusam storage
  - Save a copy: attempt to create a new record into Blusam storage



- If saving the record to the storage is successful, a message is displayed and the page will switch to a read-only mode (mask items values cannot be edited anymore):



- If for any reason the record persistence to the storage fails, an error message is displayed in red, providing a failure reason. The most common case of failures are that storing the record would lead to a key corruption (invalid or duplicate key). For an illustration, see the following note.
- To exit, choose the **Close** button.

- Cancel: Ends the editing session, closes the page, and takes you back to the records list page.

Note:

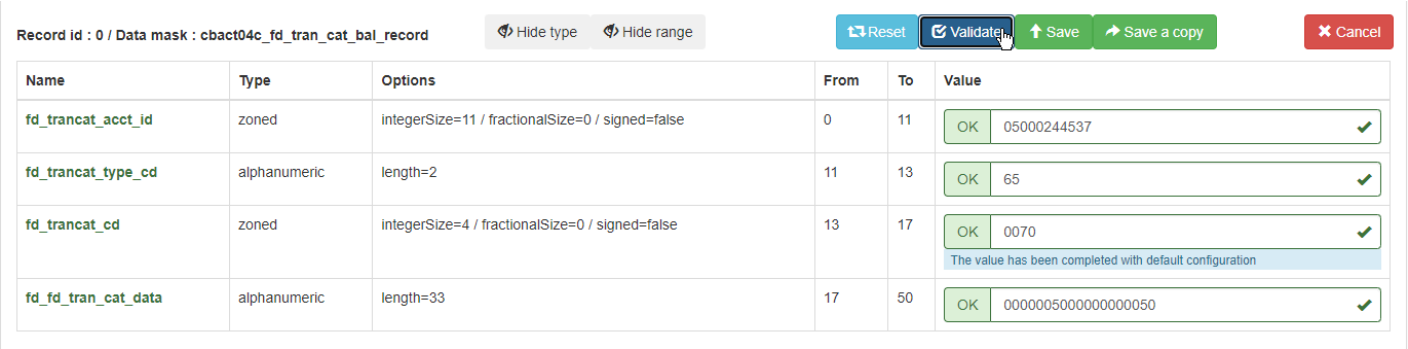
- The validation mechanism only checks that the mask item value is formally compatible with the mask item type. For example, see this failed validation on a numeric mask item:



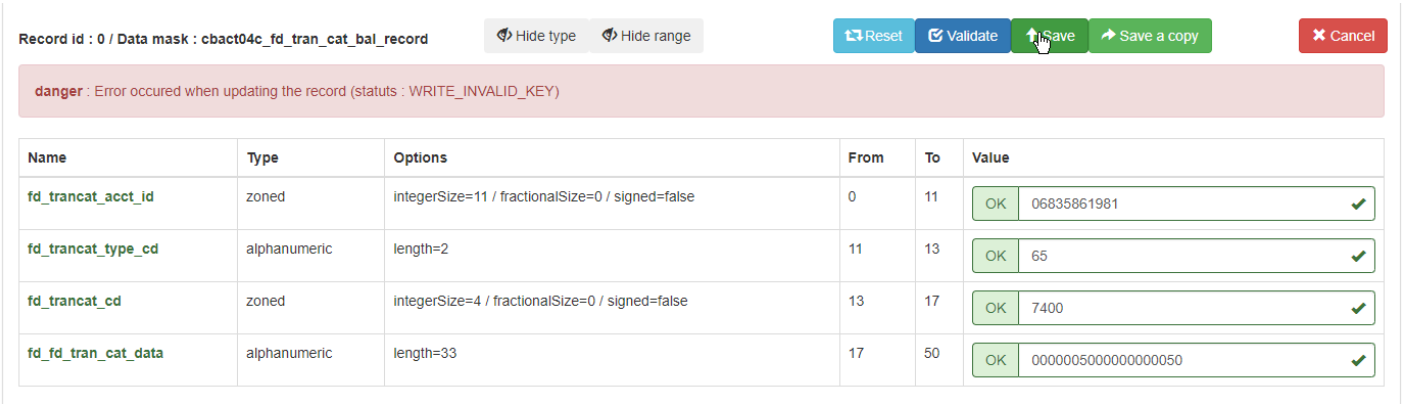
- The validation mechanism might try to auto-correct invalid input, displaying an informational message in blue to indicate that the value has been automatically corrected, according to its type. For example, inputting 7XX0 as the numeric value in the numeric `fd_trncat_cd` mask item:



Calling validation leads to the following:

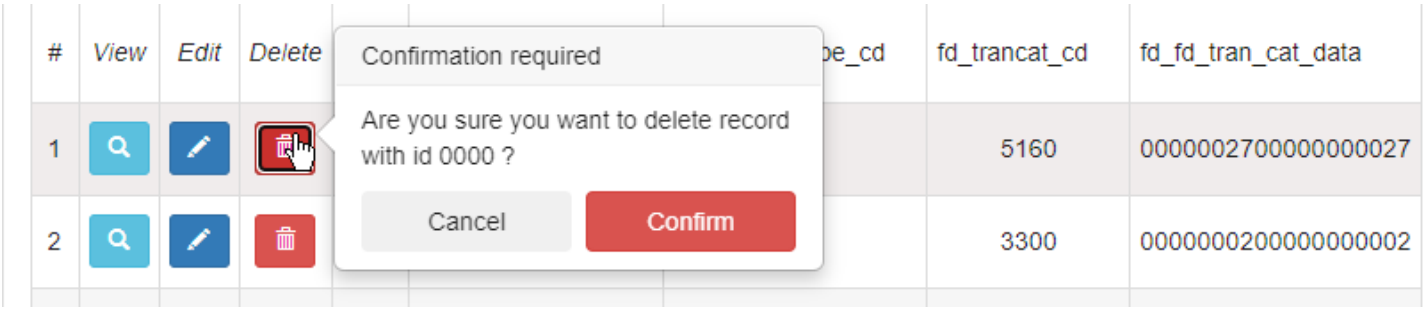


- The validation mechanism does not check whether the given value is valid in terms of key integrity (if any unique key is involved for the given data set). For instance, despite validation being successful, if provided values lead to an invalid or duplicate key situation, the persistence will fail and an error message will be displayed:



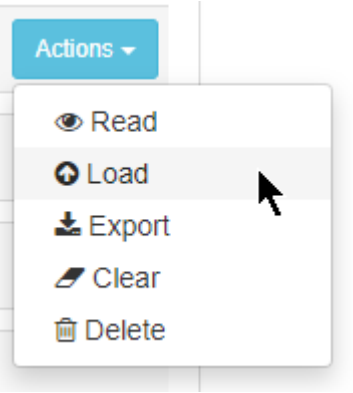
Deleting a record:

To delete a record, choose the trashcan button:

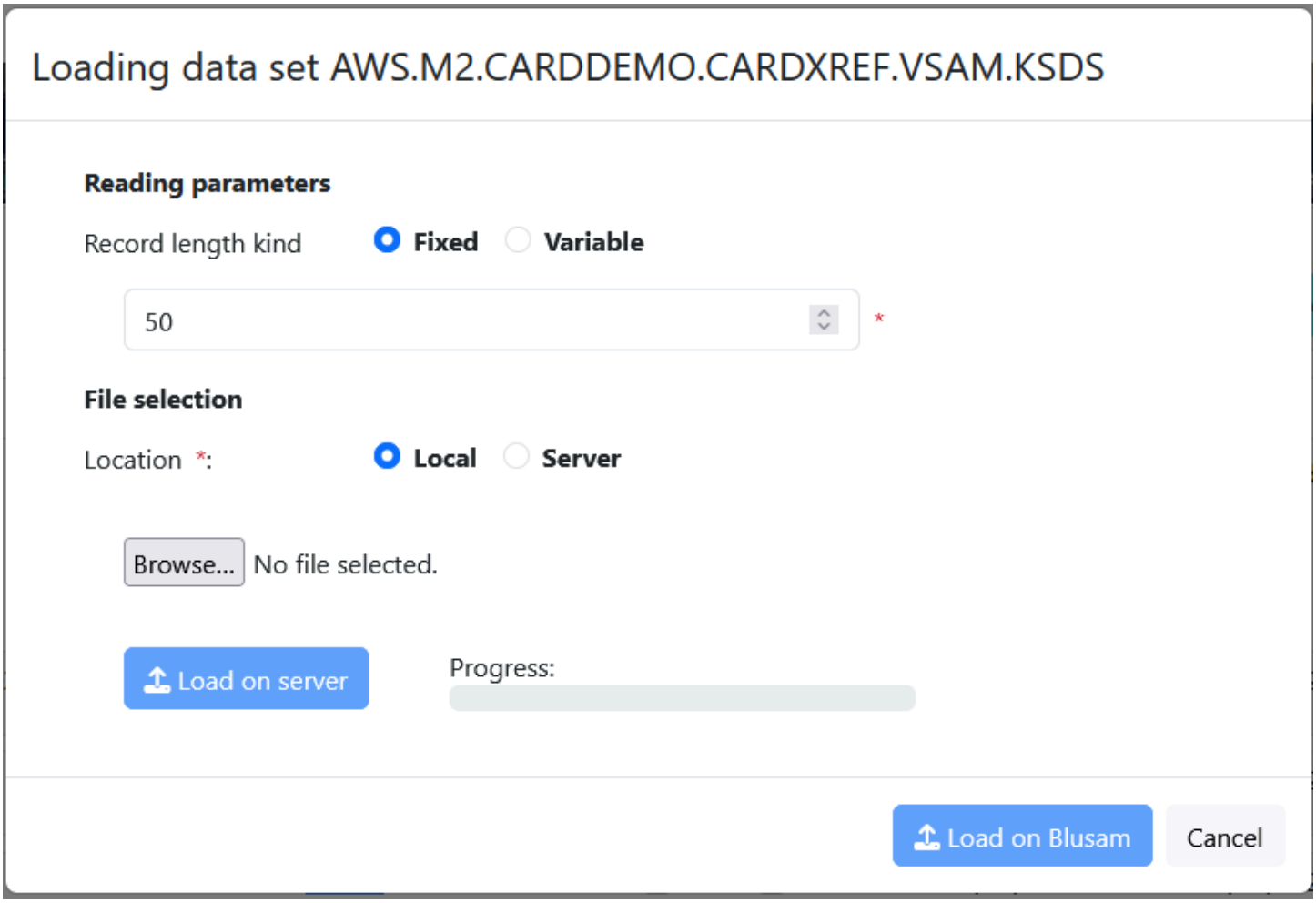


Loading records into a data set

To loading records into a data set, choose **Actions**, then choose **Load**.



A window with load options appears.

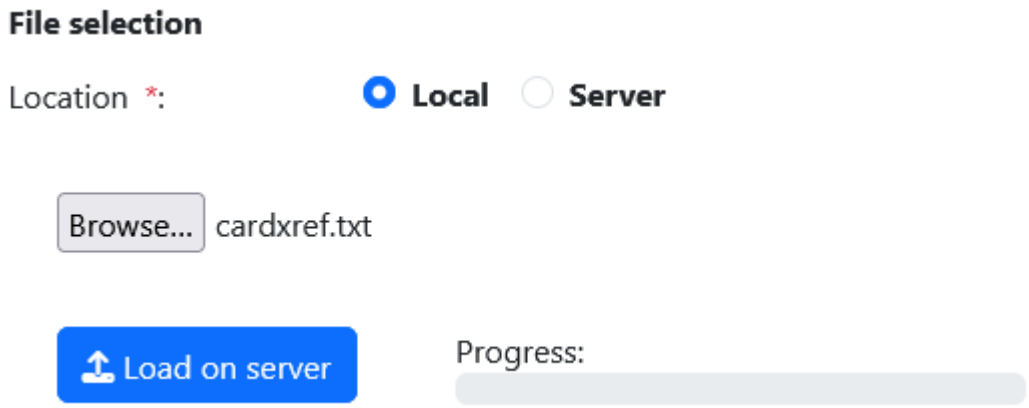


At first, both the **Load on server** and **Load on Blusam** buttons are disabled.

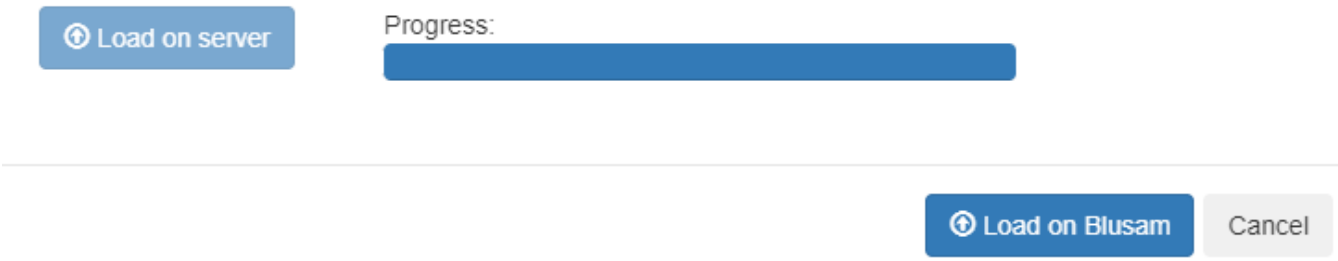
Reading parameters:

- Record length kind:

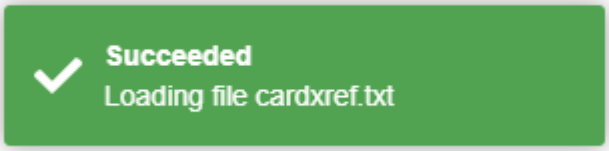
- Fixed or Variable record length: use the radio-button to specify whether the legacy data set export uses fixed length records or variable length records (the records are expected to start with RDW bytes). If you choose Fixed, the record length must be specified (in bytes) as a positive integer value in the input field. The value should be pre-filled by the information coming from the data set. If you choose Variable, the given input field disappears.
- File selection:
  - Local: choose the data set file from your local computer, using the file selector below (Note: the file selector uses your browser's locale for printing its messages -- here in french, but it might look different on your side, which is expected). After you make the selection, the window is updated with the data file name and the **Load on server** button is enabled:



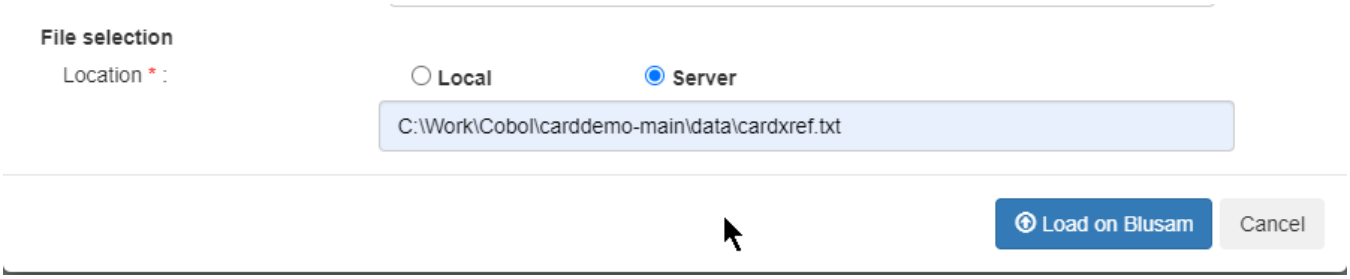
Choose **Load on server**. After the progress bar reaches its end, the **Load on Blusam** button gets enabled:



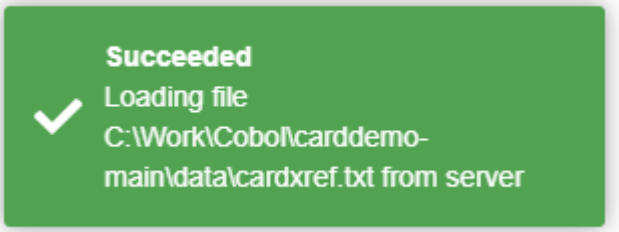
To complete the load process to the Blusam storage, choose the **Load on Blusam**. Otherwise, choose **Cancel**. If you choose to go on with the load process, a notification will appear in the lower right corner after the loading process is completed:



- Server: choosing this option makes an input field appear while the **Load on server** button disappears. The input field is where you must specify the path to the data set file on the Blusam server (this assumes that you have transferred the given file to the Blusam server first). After you specify the path, **Load on Blusam** gets enabled:

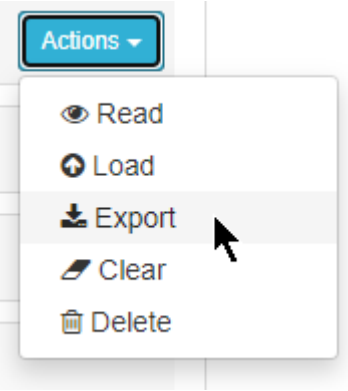


To complete the loading process, Choose **Load on Blusam**. Otherwise, choose **Cancel**. If you choose to proceed with the loading, a notification appears after the loading process is complete. The notification is different from the load from the browser as it displays the data file server path followed by the words **from server**:

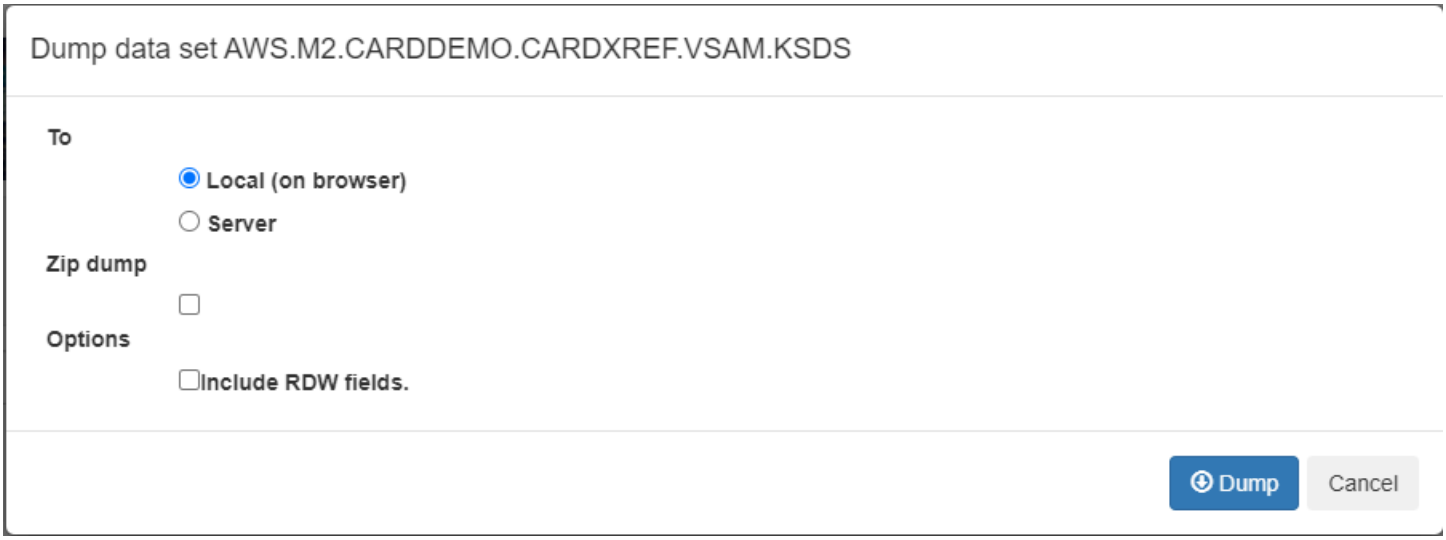


Exporting records from a data set

To export data set records, choose **Actions** in the current data set row, then choose **Export**:



The following pop-up window appears.



Options:

To : a radio button choice, to pick the export destination, either as a download in the browser (**Local (on browser)**) or to a given folder on the **Server** hosting the BAC application. If you choose to export using the **Server** choice, a new input field will be displayed:



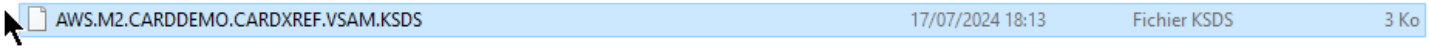
As the red asterisk on the right of the input field indicates, it is mandatory to provide a valid folder location on the server (the Dump button will be inactive while no folder location has been provided).

To export to the server, you must have the sufficient access rights for the server file system, if you plan to manipulate the exported data set file after the export.

Zip dump: a checkbox that produces a zipped archive instead of a raw file.

Options: To include a Record Descriptor Word (RDW) at the beginning of each record in the exported data set in the case of variable length record data set, choose **Include RDW fields**.

To launch the data set export process, choose **Dump**. If you choose to export to browser, check the download folder for the export data set file. The file will have the same name as the data set:

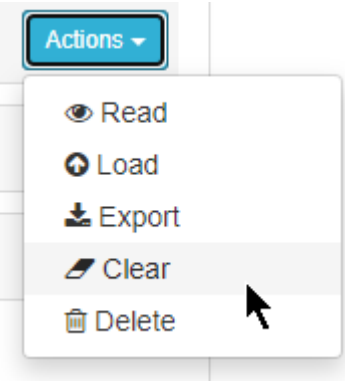


Note:

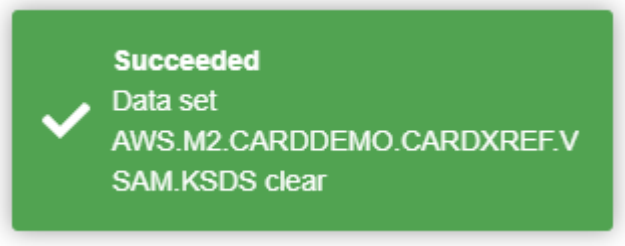
- For KSDS, the records will exported following the primary key order.
- For ESDS and RRDS, the records will be exported following the RBA (Relative Byte Address) order.
- For all data sets kinds, records will be exported as raw binary arrays (no conversion of any kind happening), ensuring direct compatibility with legacy platforms.

Clearing records from a data set

To clear all records from a data set, choose **Actions**, then choose **Clear**:

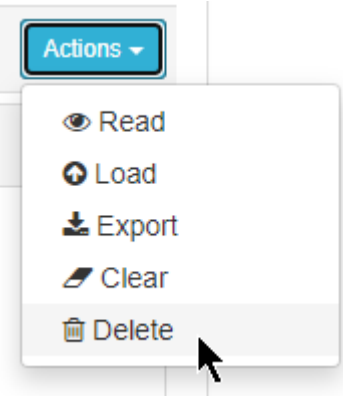


After all records are removed from a data set, the following notification appears.

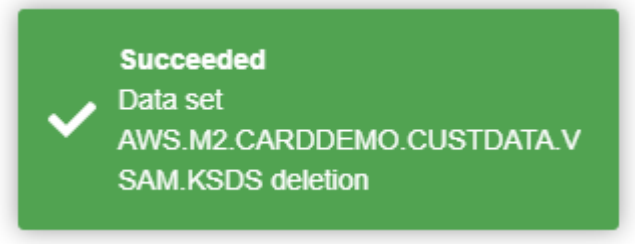


Deleting a data set

To delete a data set, choose **Actions**, then choose **Delete**:



After you delete a data set, the following notification appears:

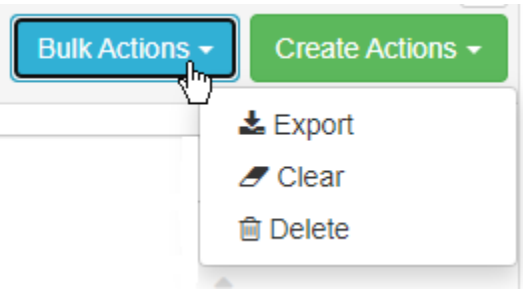


Bulk operations

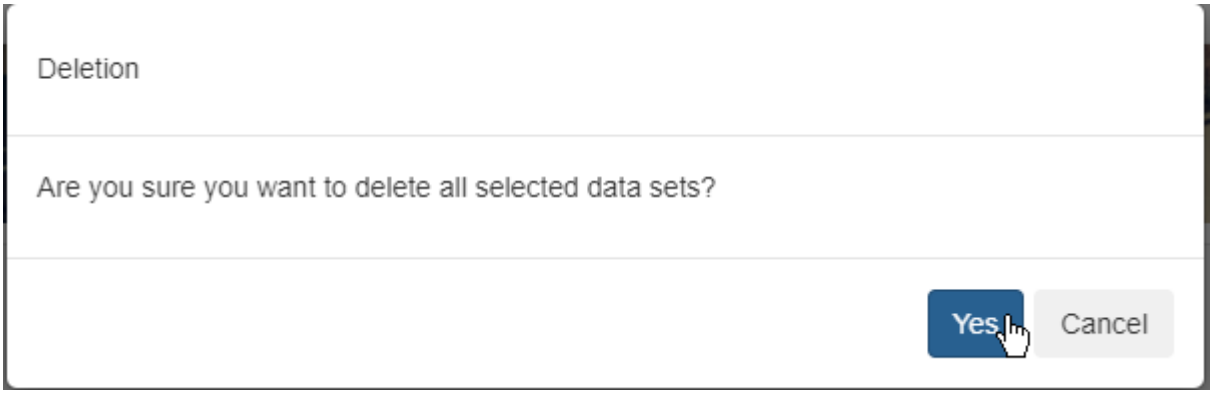
Three bulk operations are available on data sets:

- Export
- Clear
- Delete

Bulk operations can only be applied to a selection of data sets (at least one data set needs to be selected); selecting data sets is done through ticking selection checkboxes on the left of data sets rows, in the data sets list table. Selecting at least one data set will enable the Bulk Actions drop down list:



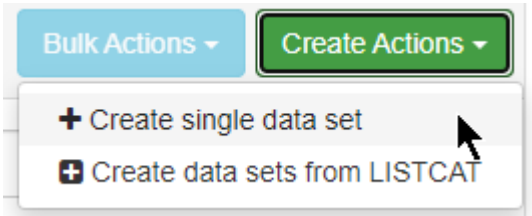
Apart from the fact that the given actions apply on a selection of data sets rather than a single one, the actions are similar to those described above, so please refer to dedicated actions documentation for details. The pop-up windows text contents will be slightly different to reflect the bulk nature. For instance, when trying to delete several data sets, the pop-up window will look like:



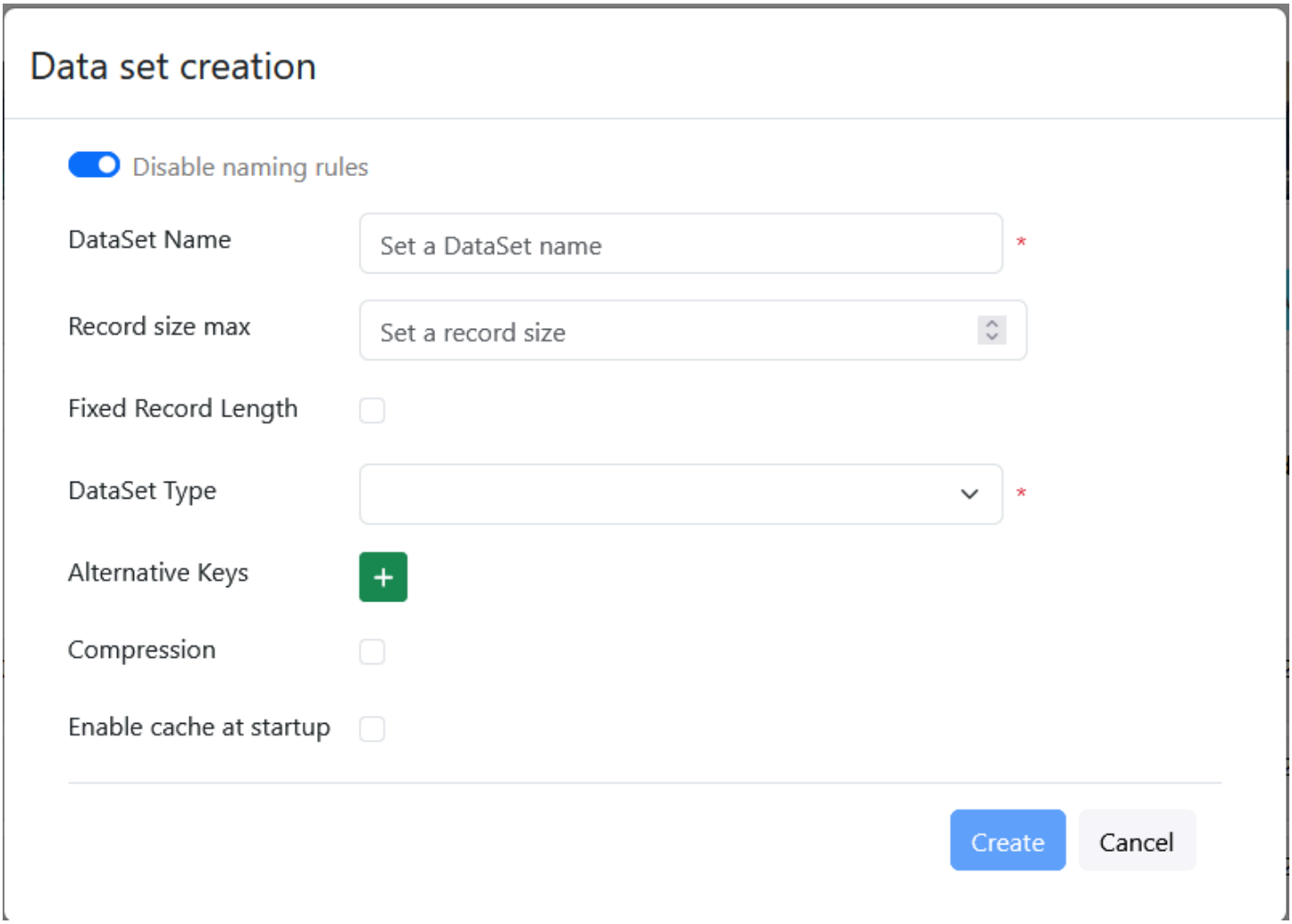
Creating operations

Create a single data set

Choose **Actions**, then choose **Create single data set**:

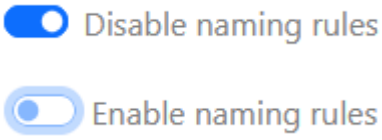


The data set creation form will then be displayed as a pop-up window:

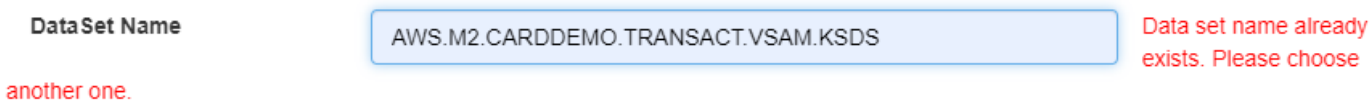


You can specify the following attributes for the data set definition:

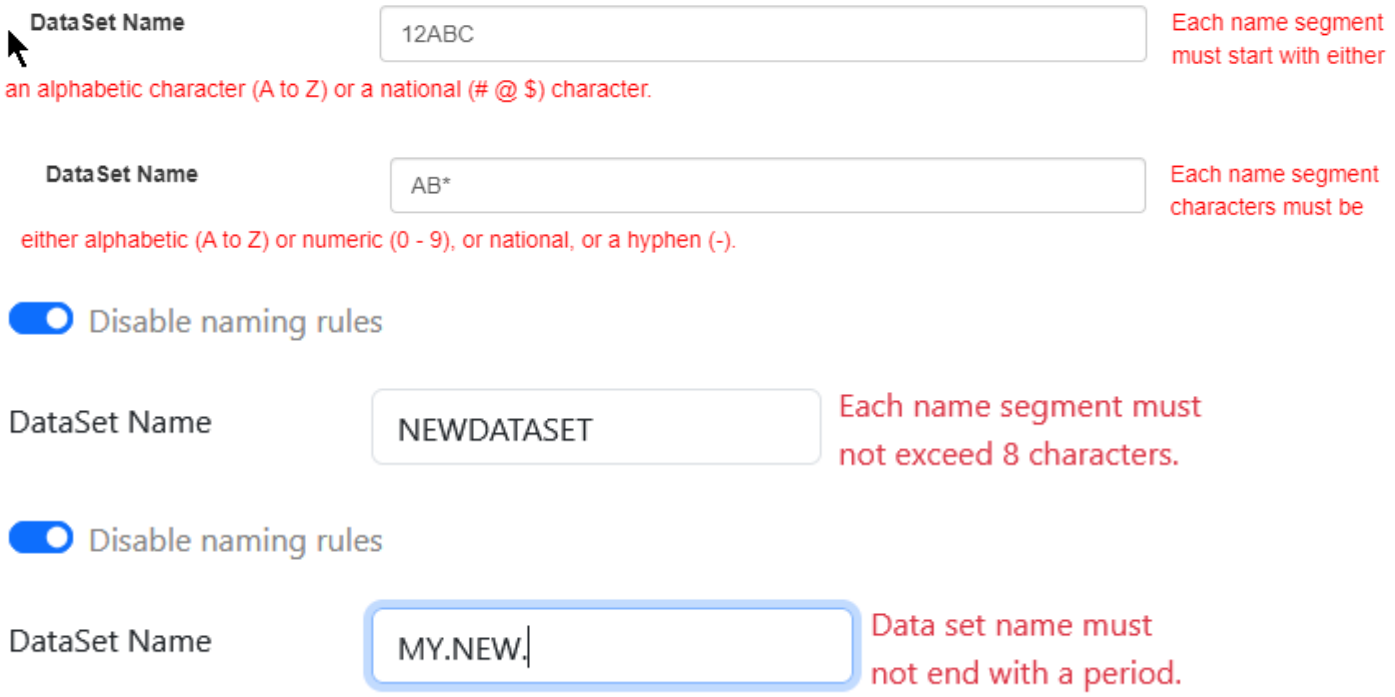
- Enabling and disabling naming rules: Use the 'Disable naming rules / Enable naming rules' toggle widget to disable and enable data set naming conventions. We recommend that you leave the toggle on the default value, with enabled data set naming rules (the toggle widget should display "Disable naming rules"):

- Data Set name: The name for the data set. If you specify a name that is already in use, the following error message appears.

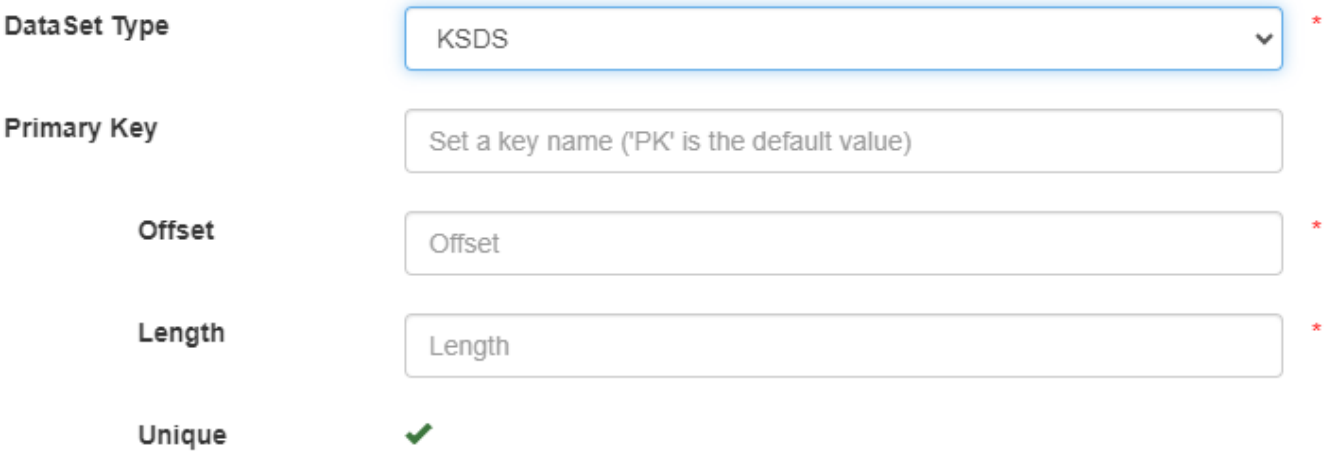
The name must also respect the naming convention if it is enabled:



- Record size max: This must be a positive integer representing the record size for a data set with fixed-length records. You can leave it blank for data sets with variable-length records .
- Fixed length record: A check box to specify whether the record length is fixed or variable. If selected, the data set will have fixed-length records, otherwise the record length will be variable.

  When you import legacy data to a variable length records data set, the provided legacy records must contain the Record Descriptor Word (RDW) that gives the length of each record.
- Data set Type: A drop-down list for specifying the current data set type. The following types are supported.
  - ESDS
  - LargeESDS
  - KSDS

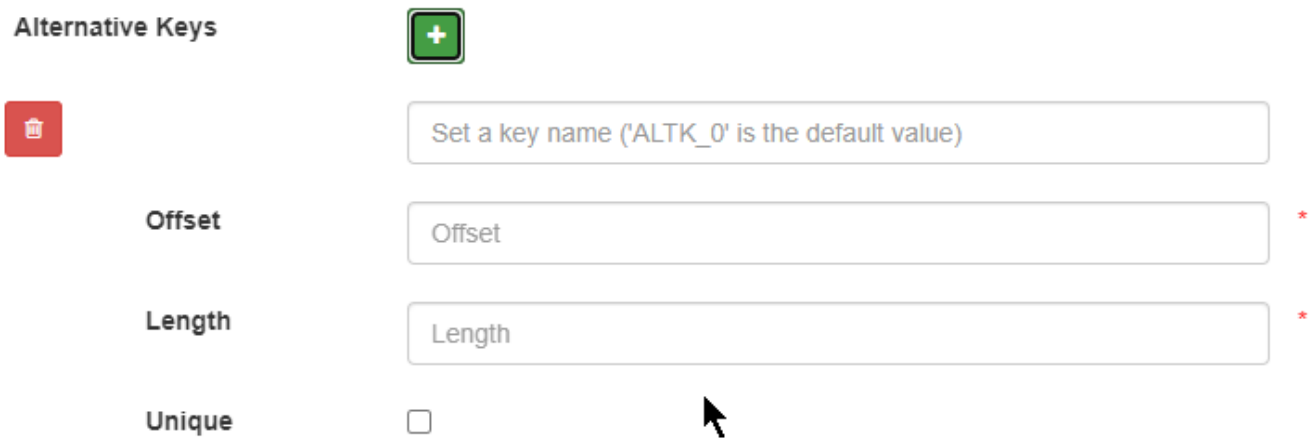
  For KSDS, you must specify the primary key:



For the primary key, specify the following:

- Name: This field is optional. The default is **PK**.
- Offset: The 0-based offset of the primary key within the record. The offset must be a positive integer. This field is required.
- Length: The length of the primary key. This length must be a positive integer. This field is required.

For KSDS and ESDS, you can optionally define a collection of alternate keys, by choosing the Plus button in front of the Alternate Keys label. Each time you choose that button, a new alternate key definition section appears in the data set creation form:



For each alternative key, you need to provide:

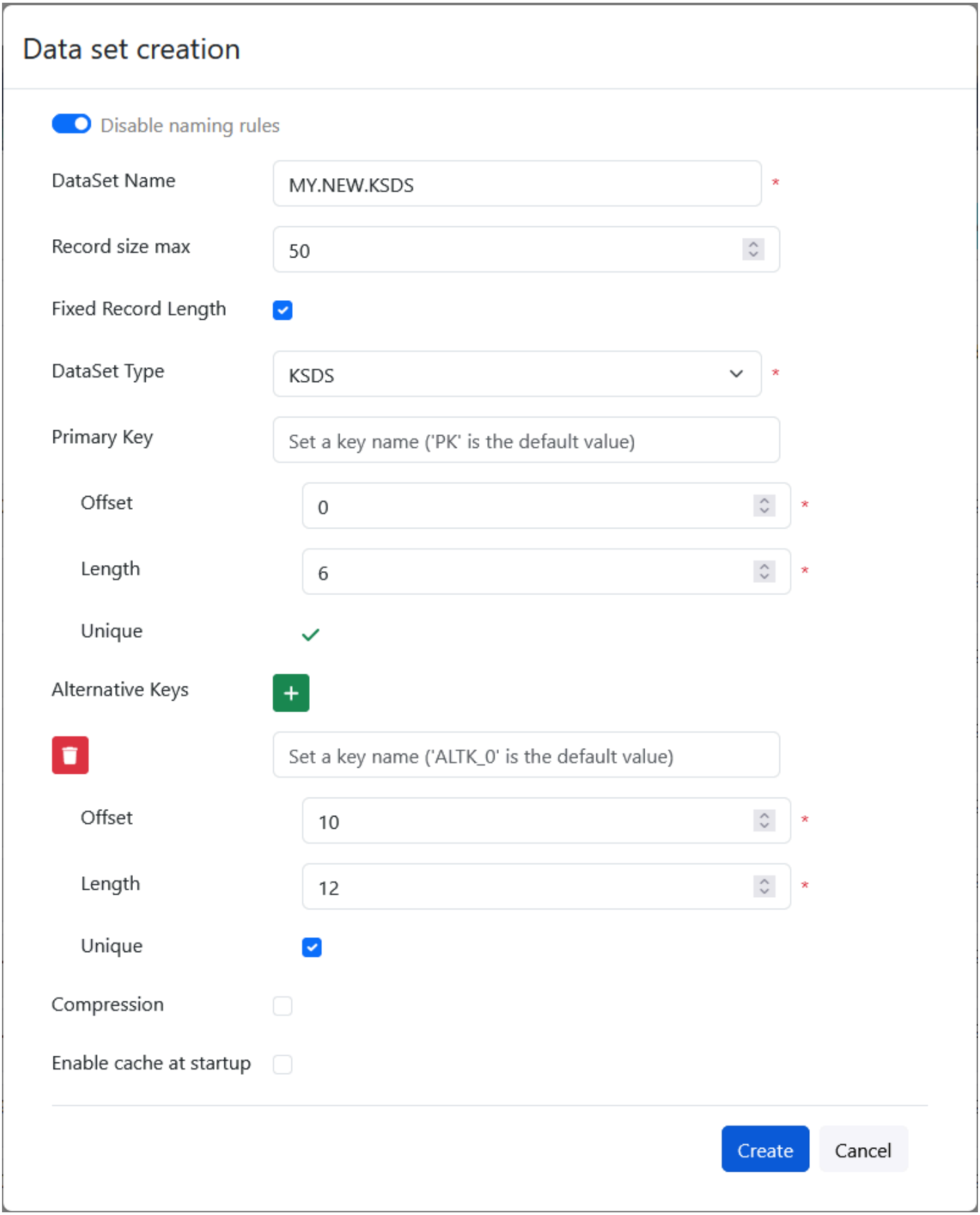
- Name: This field is optional. The default value is **ALTK_#**, where # represents an auto-incremented counter that starts at 0.

- Offset: The 0-based offset of the alternative key within the record. Must be a positive integer. This field is required.
- Length: The length of the alternative key. This length must be a positive integer. This field is required.
- Unique: A checkbox to indicate whether the alternative key will accept duplicate entries. If selected, the alternative key will be defined as unique (NOT accepting duplicate key entries). This field is required.

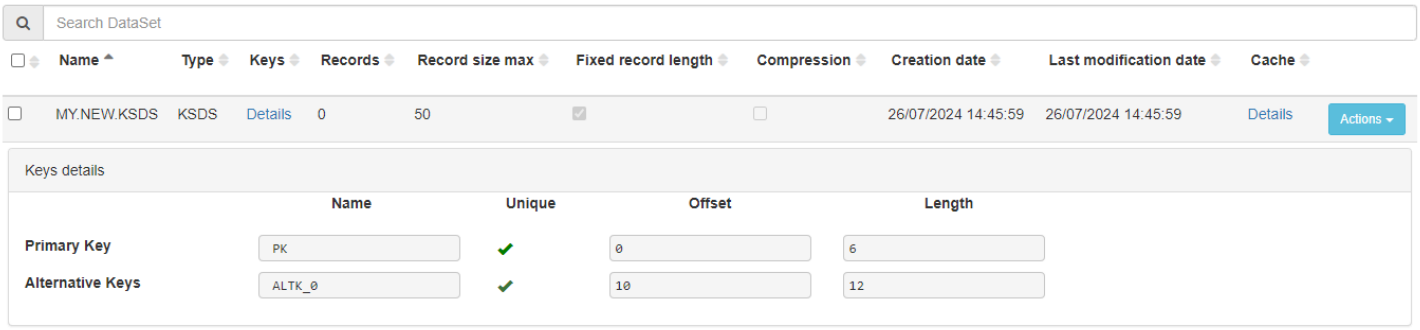
  To remove the alternate key definition, use the trashcan button on the left.

- Compression: A checkbox to specify whether compression will be used to store the data set.
- Enable cache at startup: A checkbox to specify whether the data set should be loaded into cache at application startup.

After you specify the attribute definitions, choose **Create** to proceed:


Data set creation
Disable naming rules
DataSet Name
MY.NEW.KSDS
Record size max
50
Fixed Record Length
DataSet Type
KSDS
Primary Key
Set a key name ('PK' is the default value)
Offset
0
Length
6
Unique
Alternative Keys
Set a key name ('ALTK_0' is the default value)
Offset
10
Length
12
Unique
Compression
Enable cache at startup
Create
Cancel

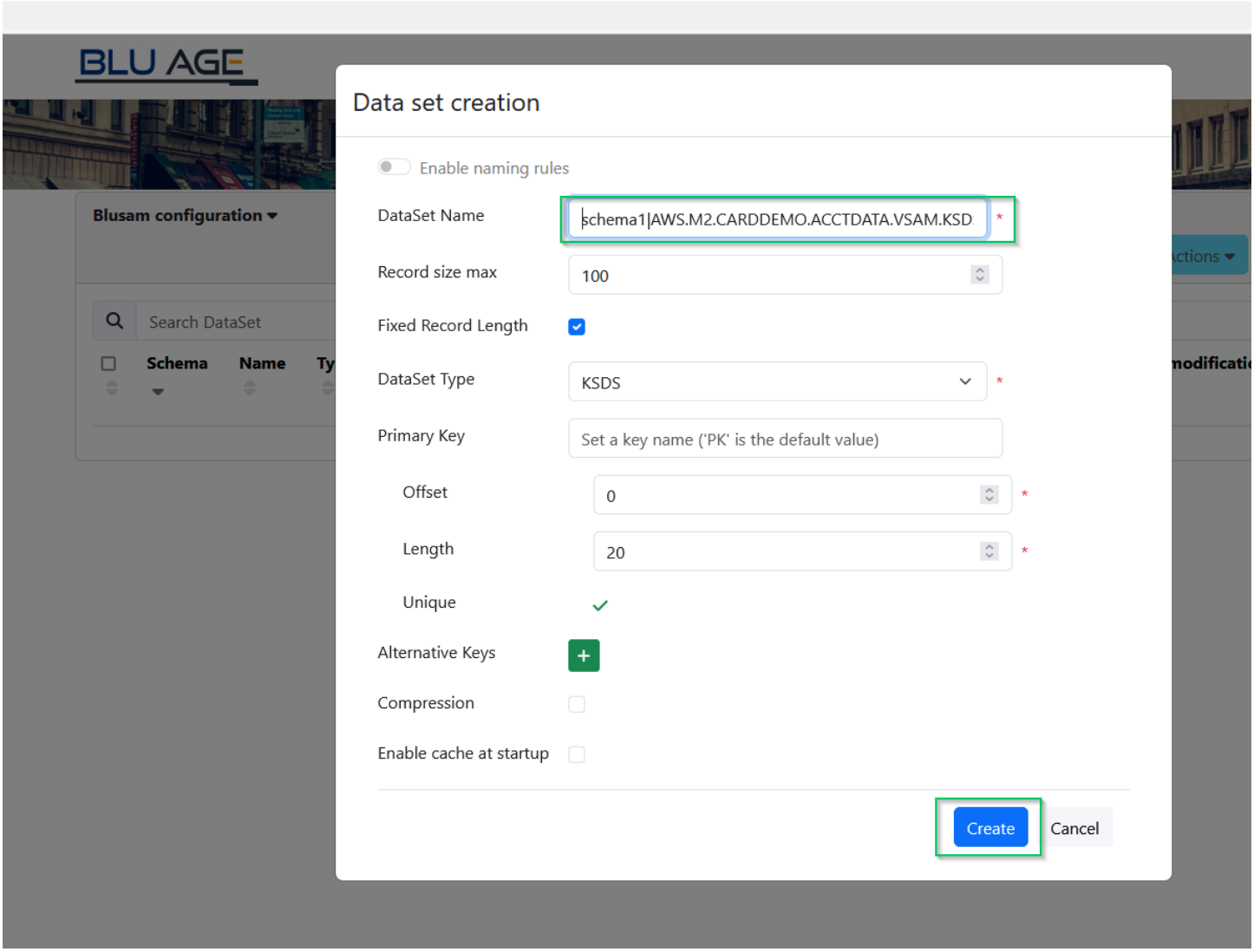
The creation window will be closed and the home page showing the list of data sets will be displayed. You can view the details of the newly created data set.



Create a single data set in Multi-schema mode

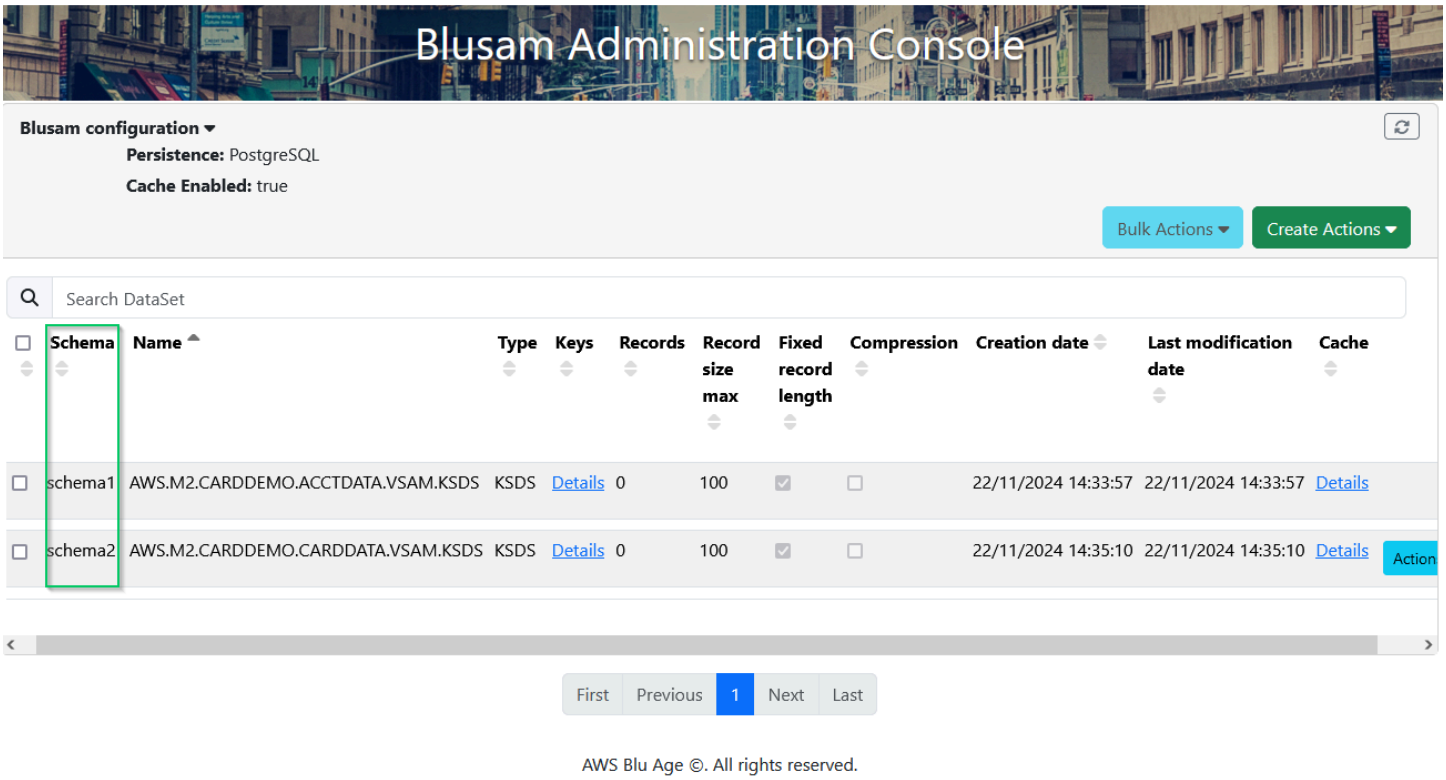
A data set can be created in a Multi-schema mode by prefixing the data set name with the schema name followed by a pipe (|) symbol (e.g., `schema1|AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS`).

> **Note**
>
> The Schema used for creating the data set should be specified in the `application-main.yml` configuration. For more information, see the section called “Multi-schema configuration properties ”.



If no schema prefix is provided, the data set will get created in the default schema specified in the Blusam datasource URL in Blusam Datasource configuration. If no schema is specified in the Blusam datasource URL, then 'public' schema is used by default.

> **Note**
>
> In Multi-schema mode, BAC console displays the schema information of the data set in the first column.



Create data sets from LISTCAT

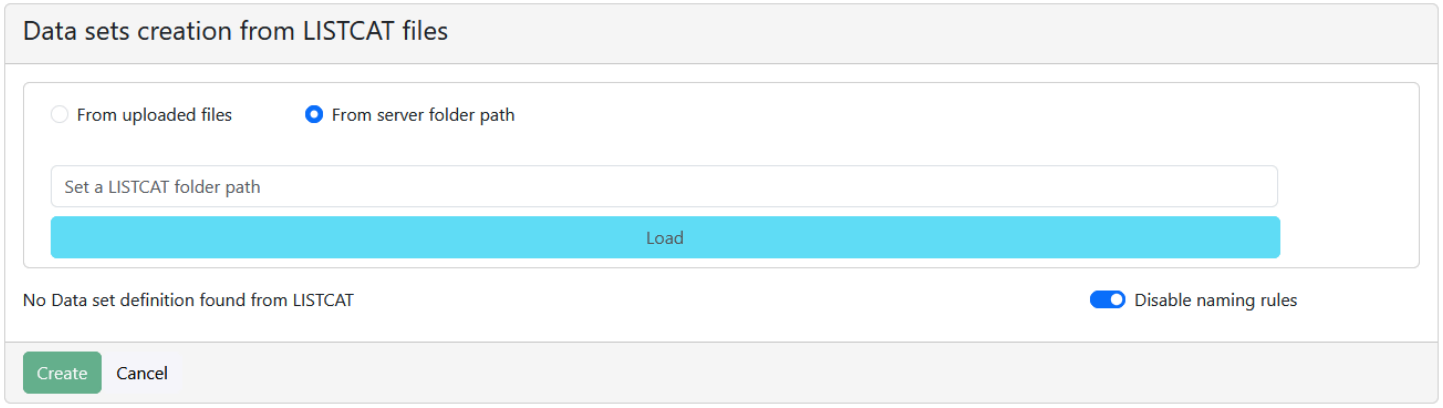
This feature makes it possible to take advantage of the LISTCAT JSON files created during the BluAge transformation process using BluInsights Transformation Center as the result of parsing LISTCAT export from the legacy platforms: LISTCAT exports are parsed and transformed into JSON files that hold the data set definitions (names, data set type, keys definitions, and whether the record length is fixed or variable).

Having the LISTCAT JSON files makes it possible to create data sets directly without having to manually enter all the information required for data sets. You can also create a collection of data sets directly instead of having to create them one by one.

If no LISTCAT JON file is available for your project (for example, because no LISTCAT export file was available at transformation time), you can always manually create one, provided you adhere to the LISTCAT JSON format detailed in the appendix.

From the Create Actions drop-down list, choose **Create data sets from LISTCAT**.

The following dedicated page will be displayed:



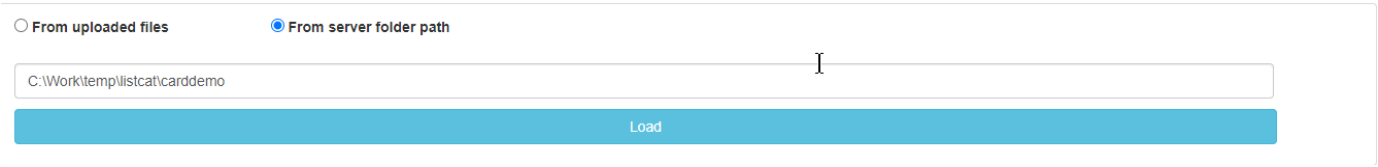
At this stage, the **Load** button is disabled, which is expected.

Use the radio buttons to specify how you want to provide the LISTCAT JSON files. There are two options:

- You can use your browser to upload the JSON files.
- You can select the JSON files from a folder location on the server. To choose this option, you must first copy the JSON files to the given folder path on the server with proper access rights.

To use JSON files on the server

1. Set the folder path on the server, pointing at the folder containing the LISTCAT JSON files:



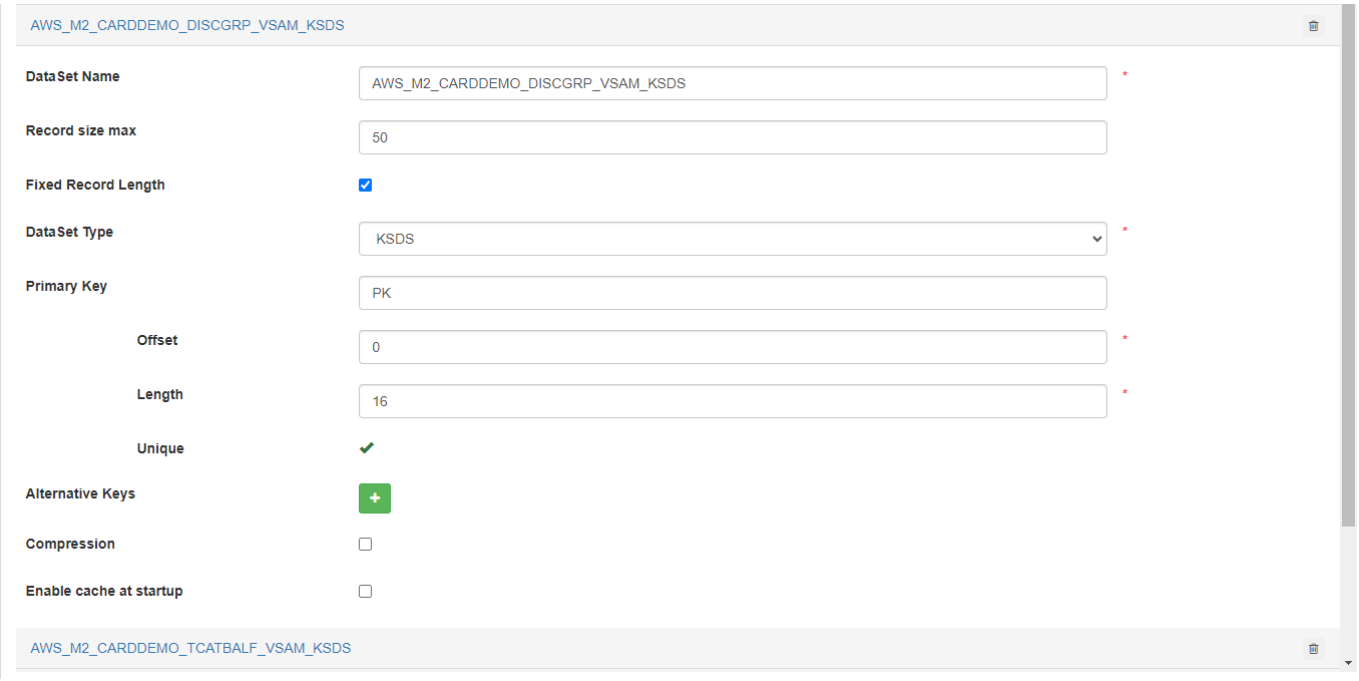
2. Choose the **Load** button. All recognized data set definitions will be listed in a table:



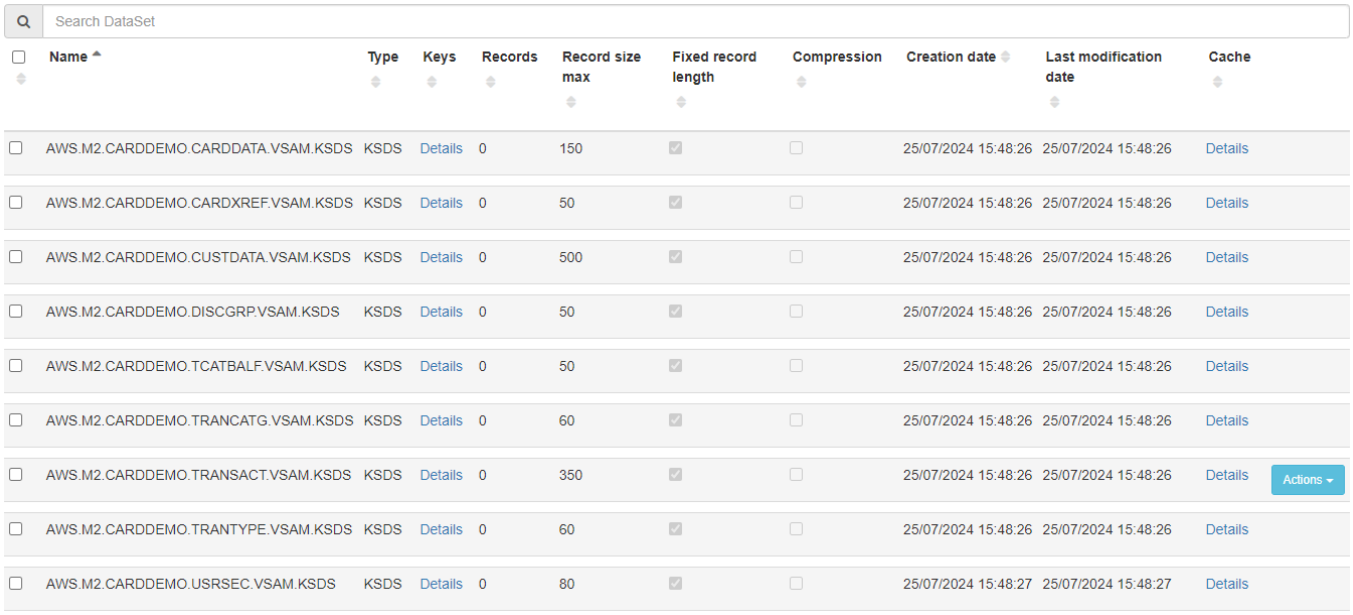
Each row represents a data set definition. You can use trashcan button to remove a data set definition from the list.

> **Important**
>
> The removal from the list is immediate, with no warning message.

3. The name on the left is a link. You can choose it to show or hide the details of the data set definition, which is editable. You can freely modify the definition, starting on the basis of the parsed JSON file.

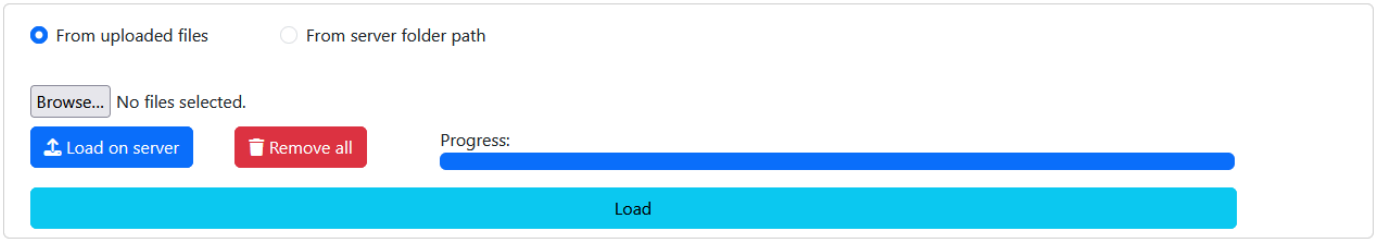


4. To create all data sets, choose **Create**. All data sets will be created, and will be displayed on the data sets results page. The newly created data sets will all have 0 records.



To upload files to the server

1. This option is similar to using the files from the server folder path, but in this case you must first upload the files using the file selector. Select all files to upload from your local machine, then choose **Load on server**.



2. When the progress bar reaches the end, all files have been successfully uploaded to the server and the **Load** button is enabled. Choose the **Load** button and use the discovered data set definitions as explained previously.

LISTCAT JSON format

The LISTCAT JSON format is defined by the following attributes:

- optional "catalogId": identifier of the legacy catalog as a String, or "default" for the default catalog.
- "identifier": the data set name, as a String.

- "isIndexed": a boolean flag to indicate KSDS: true for KSDS, false otherwise.
- "isLinear": a boolean flag to indicate ESDS: true for ESDS, false otherwise.
- "isRelative": a boolean flag to indicate RRDS: true for RRDS, false otherwise
- **Note**: "isIndexed", "isLinear", and "isRelative" are mutually exclusive.
- "isFixedLengthRecord": a boolean flag: set to true if fixed length records data set, false otherwise.
- "avgRecordSize": Average record size in bytes, expressed as a positive integer.
- "maxRecordSize": Maximal Record size in bytes, expressed as an integer. Should be equal to avgRecordSize for fixed length record size.
- for KSDS only: Mandatory primary Key definition (as nested object)
  - labelled "primaryKey"
  - "offset": 0-based bytes offset for the primary key in the record.
  - "length": length in bytes of the primary key.
  - "unique": must be set to true for primary key.
- for KSDS/ESDS, collection of alternate keys (as collection of nested objects):
  - labelled "alternateKeys"
  - For each alternate key:
    - "offset": 0-based bytes offset for the alternate key in the record.
    - "length": length in bytes of the alternate key.
    - "unique": must be set to true for alternate key, if the key does not accept duplicate entries, false otherwise.
- if no alternate keys are present, provide an empty collection:

```
alternateKeys: []
```

The following is a sample KSDS LISTCAT JSON file.

```
{
  "catalogId": "default",
  "identifier": "AWS_M2_CARDDEMO_CARDXREF_VSAM_KSDS",
  "isIndexed": true,
  "isLinear": false,
  "isRelative": false,
  "isFixedLengthRecord": true,
  "avgRecordSize": 50,
```

```
  "maxRecordSize": 50,
  "primaryKey": {
    "offset": 0,
    "length": 16,
    "unique": true
  },
  "alternateKeys": [
    {
      "offset": 25,
      "length": 11,
      "unique": false
    }
  ]
}
```

Set up configuration for AWS Blu Age Runtime

The AWS Blu Age Runtime 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 AWS Blu Age Runtime also contains additional optional web applications, that can be opted-in if needed.

Topics

- Application configuration basics
- Application precedence
- JNDI for databases
- AWS Blu Age Runtime secrets
- Other files (groovy, sql, etc.)
- Additional web application
- Enable properties for AWS Blu Age Runtime
- Available Redis cache properties in AWS Blu Age Runtime
- Configure security for Gapwalk applications

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.yaml`
- `application-`*`profile`*`.yaml` (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.

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

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

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

The following example shows a typical client YAML file.

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

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

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

For information about the content of the YAML files, see Enable properties for AWS Blu Age Runtime.

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.

AWS Blu Age Runtime 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 YAML configuration so the secret content is picked up on the fly at Apache Tomcat startup.

Secrets for Aurora

Aurora database configuration (for JICS, Blusam, customer db, and so on) 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 can 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.

YAML references to secrets

The `application-main.yml` can reference the secret ARN for various resources:

JICS database

JICS database credentials with `spring.aws.jics.db.secret`

```
spring:
   aws:
     jics:
       db:
         dbname: jics
         secret: arn:aws:secretsmanager:XXXX
```

Supported JICS database secret keys:

Secret key	Secret key description
host	The host name
port	The port
dbname	The name of the database
username	The username
password	The password
engine	Database engine: Postgres, Oracle, Db2, Microsoft SQL Server
currentSchema	Specific schema to use (Db2 support only)
sslConnection	Whether to use SSL connection (Db2 support only)
sslTrustStoreLocation	The location of the truststore on the client (Db2 support only)
sslTrustStorePassword	The password for the truststore on the client (Db2 support only)

> **Note**
>
> The name of the database is either supplied in the secret or in the yaml reference `spring.aws.jics.db.dbname`.

Blusam database

Blusam database credentials with `spring.aws.client.bluesam.db.secret`

```
spring:
   aws:
     client:
       bluesam:
         db:
           dbname: bluesam
           secret: arn:aws:secretsmanager:XXXX
```

Supported Blusam database secret keys:

Secret key	Secret key description
host	The host name
port	The port
dbname	The name of the database
username	The username
password	The password
engine	Database engine: Postgres

> **Note**
>
> The name of the database is either supplied in the secret or in the yaml reference `spring.aws.client.bluesam.db.dbname`.

Client database

The client `application-profile.yml` can reference the secret ARN for the client database. This requires an additional property to list the datasource names `spring.aws.client.datasources.names`. For each datasource name `ds_name` specify the secret ARN in the following property: `spring.aws.client.datasources.ds_name.secret`. Example:

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

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.

Supported client database secret keys:

Secret key	Secret key description
host	The host name
port	The port
dbname	The name of the database
username	The username

Secret key	Secret key description
password	The password
engine	Database engine: Postgres, Oracle, Db2, Microsoft SQL Server
currentSchema	Specific schema to use (Db2 support only)
sslConnection	Whether to use SSL connection (Db2 support only)
sslTrustStoreLocation	The location of the truststore on the client (Db2 support only)
sslTrustStorePassword	The password for the truststore on the client (Db2 support only)

PGM utility database

The `application-utility-pgm.yml` can reference the secret ARN for various resources.

- `spring.aws.client.datasources.primary`
  - `secret`

    Secret ARN for the application database.

Type: string

- `type`

Fully qualified name of the connection pool implementation to use.

Type: string

Default: `com.zaxxer.hikari.HikariDataSource`

- `spring.aws.client.utility.pgm.datasources`
  - `names`

List of data source names.

Type: string

- `dsname`
  - `dbname`

Name of the host.

Type: string

- `secret`

Secret ARN of the host database.

Type: string

- `type`

Fully qualified name of the connection pool implementation to use.

Type: string

Default: `com.zaxxer.hikari.HikariDataSource`

For a multi-datasources secret:

```
spring:
   aws:
     client:
       primary:
         secret: arn:aws:secretsmanager:XXXX
         type: dataSourceType
       utility:
         pgm:
           datasources:
             names: dsname1,dsname2,dsname3
               dsname1:
                 dbname: dbname1
                 secret: arn:aws:secretsmanager:XXXX
                 type: dataSourceType
               dsname2:
```

```
    dbname: dbname2
    secret: arn:aws:secretsmanager:XXXX
    type: dataSourceType
  dsname3:
    dbname: dbname3
    secret: arn:aws:secretsmanager:XXXX
    type: dataSourceType
```

No XA supported secret keys

- engine (postgres/oracle/db2/mssql)
- port
- dbname
- currentSchema
- username
- password
- url
- sslConnection
- sslTrustStoreLocation
- sslTrustStorePassword

For `postgres` only the `sslMode` secret key value (`disable/allow/prefer/require/verify-ca/verify-full`) and the `spring.aws.rds.ssl.cert-path` YAML property make it possible to connect with SSL.

XA supported secret keys

If the client database is using XA, the sub xa-properties are supported through secret values.

- host
- port
- dbname
- currentSchema
- username
- password
- url

- sslConnection (true/false)
- sslTrustStoreLocation
- sslTrustStorePassword

However, for other xa-properties (for example `maxPoolSize` or `driverType`), the regular YAML key `spring.jta.atomikos.datasource.XXXX.unique-resource-name` must still be supplied.

The secret value overrides the YAML properties.

Default Super Admin BAC and JAC

You can also configure application-main.yml to retrieve the username and the password of the default super admin user in the secret from AWS Secrets Manager by specifying the ARN. The following example shows how to declare this secret in a YAML file.

```
spring:
   aws:
     client:
       defaultSuperAdmin:
         secret: arn:aws:secretsmanager:XXXX
```

Supported default super admin database secret keys:

Secret key	Secret key description
username	The username.
password	The password.

OAuth2

You can also configure `application-main.yml` to retrieve the OAuth2 client secret from AWS Secrets Manager by specifying the provider and ARN. The default value for the provider property is Amazon Cognito. The following is an example configuration for the OAuth2 provider Keycloak:

```
spring:
   aws:
     client:
```

```
      provider: keycloak
      keycloak:
        secret: arn:aws:secretsmanager:XXXX
```

In this example, the client-secret for the OAuth2 provider Keycloak is retrieved from the specified ARN in AWS Secrets Manager. This configuration supports multiple providers by dynamically resolving the provider name and corresponding secret ARN.

Supported OAuth2 secret keys:

Secret key	Secret key description
client-secret	The secret generated by the authorization server during the process of application registration.

Secret manager for Redis caches

The `application-main.yml` file can reference the secret ARN for Redis caches. The supported one are:

- Gapwalk Redis credentials with `spring.aws.client.gapwalk.redis.secret`
- Bluesam Redis credentials with `spring.aws.client.bluesam.redis.secret`
- Bluesam locks Redis credentials with `spring.aws.client.bluesam.locks.redis.secret`
- Dataset catalog Redis credentials with `spring.aws.client.dataset.catalog.redis.secret`
- JICS Redis credentials with `spring.aws.client.jics.redis.secret`
- Session Redis credentials with `spring.aws.client.jics.redis.secret`
- Session tracker Redis credentials with `spring.aws.client.session.tracker.redis.secret`
- JICS TS Queues Redis credentials with `spring.aws.client.jics.queues.ts.redis.secret`
- JCL checkpoint Redis credentials with `spring.aws.client.jcl.checkpoint.redis.secret`
- Gapwalk files locks Redis credentials with `spring.aws.client.gapwalk.files.locks.redis.secret`

- Blu4IV locks Redis credentials with `spring.aws.client.blu4iv.locks.redis.secret`

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

```
spring:
   aws:
     client:
       gapwalk:
         redis:
           secret: arn:aws:secretsmanager:XXXX
       bluesam:
         locks:
           redis:
             secret: arn:aws:secretsmanager:XXXX
         redis:
           secret: arn:aws:secretsmanager:XXXX
       dataset:
         catalog:
           redis:
             secret: arn:aws:secretsmanager:XXXX
       jics:
         redis:
           secret: arn:aws:secretsmanager:XXXX
       session:
         tracker:
           redis:
             secret: arn:aws:secretsmanager:XXXX
       jics:
         queues:
           ts:
             redis:
               secret: arn:aws:secretsmanager:XXXX
       jcl:
         checkpoint:
           redis:
             secret: arn:aws:secretsmanager:XXXX
       gapwalk:
         files:
           locks:
             redis:
               secret: arn:aws:secretsmanager:XXXX
       blu4iv:
         locks:
```

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

Supported Redis secret keys:

Secret key	Secret key description
hostname	The Redis server hostname.
port	The Redis server port.
username	The username.
password	The password.

Secret manager for SSL password settings

The `application-main.yml` file can reference the secret ARN for SSL password settings. The following is supported.

- Gapwalk SSL credentials with `spring.aws.client.ssl.secret`

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

```
spring:
   aws:
     client:
       ssl:
         secret: arn:aws:secretsmanager:XXXX
```

Secret key	Secret key description
trustStorePassword	The truststore password.
keyStorePassword	The keystore password.

Secret manager for IBM MQ password settings

The `application-main.yml` file can reference the secret ARN for IBM MQ settings. The following is supported.

- IBM MQ connections are defined as a list, and so are the credentials:

  `mq.queues.jmsMQQueueManagers[N].secret:`

  N starts at 0 for the first connection.

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

```
mq.queues.jmsMQQueueManagers[0].secret: Secret-0-ARN
mq.queues.jmsMQQueueManagers[1].secret: Secret-1-ARN
```

For information about secret ARNs, see What's in a Secrets Manager secret?

Properties defined in the secret will override their corresponding values in the `jmsMQ` YAML configuration.

If `queueManager` is set in the secret, it will override the `mq.queues.jmsMQQueueManagers[N].jmsMQQueueManager` value in the YAML file.

Secret key	Secret key description
queueManager	The IBM MQ queue manager name.
appName	The IBM MQ application name.
channel	The IBM MQ channel name.
host	The IBM MQ hostname.
port	The IBM MQ port.
userId	The IBM MQ user name.
password	The IBM MQ user password.
maxPoolSize	The IBM MQ maximum pool size.

Secret key	Secret key description
sslCipherKey	The IBM MQ SSL cipher suite.

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 or a client YAML 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 property is not found, the files will be taken from the default location mentioned above. The lookup will first be done with the tomcat working directory as a root, and lastly in the application war file.

Additional web application

The AWS Blu Age Runtime 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.yaml` as its configuration.
- `gapwalk-cl-command.war`: contains support for AS/400 utility programs and uses `application-cl-command.yaml` as its configuration.
- `gapwalk-hierarchical-support.war`: contains IMS/MFS transaction support and uses `application-jhdb.yaml` as its configuration

Enable properties for AWS Blu Age Runtime

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. You can use this page to learn about the available properties for AWS Blu Age Runtime and how to enable them.

Topics

- YML notation
- Quick start / Use cases
- Available properties for the main application
- Available properties for optional web applications
- Available properties for the client application

YML notation

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

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

Quick start / Use cases

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

- Default application-main.yml

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

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

###################################
##### Spring configuration #####
###################################
spring:
  quartz:
    auto-startup: false
    scheduler-name: Default
    properties:
      org.quartz.threadPool.threadCount: 1
  jta:
    enabled: false
    atomikos.properties.maxTimeout : 600000
    atomikos.properties.default-jta-timeout : 100000
  jpa:
# DISABLE OpenEntityManagerInViewInterceptor
    open-in-view: false
```

```
    # Fix Postgres JPA Error:
    # Method org.postgresql.jdbc.PgConnection.createClob() is not yet implemented.
    properties.hibernate.temp.use_jdbc_metadata_defaults : false
####################################
##### Jics tables configuration #####
####################################

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

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

##########################
###### Level 2 cache #####
##########################
#        cache:
#          use_second_level_cache: true
#          use_query_cache: true
#          region:
#            factory_class: org.hibernate.cache.ehcache.EhCacheRegionFactory
#      javax:
#        persistence:
#          sharedCache:
#            mode: ENABLE_SELECTIVE
##########################
###### Redis settings #####
##########################
  session:
    store-type: none #redis

# Secret manager configuration for global Redis cache
    aws:
      client:
        gapwalk:
```

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

########################################
##### 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
    url : jdbc:postgresql://localhost/bluesam
    username : bluesam
    password : bluesam
    type : org.postgresql.ds.PGSimpleDataSource

########################################
##### Embedded Bluesam configuration #####
########################################
bluesam :
  remote : false
  cache : ehcache
  persistence : pgsql
  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

jics.disableSyncpoint : false
#jics.initList:
#jics.parameters.datform: DDMMYY
#jics.parameters.applid: VELOCITY
#jics.parameters.sysid: CICS
#jics.parameters.eibtrmid: TERM
#jics.parameters.userid: MYUSERID
#jics.parameters.username: MYUSERNAME
#jics.parameters.opid: XXX
#jics.parameters.cwa.length: 0
#jics.parameters.netname: MYNETNAME
#jics.parameters.jobname: MJOBNAME
#jics.parameters.sysname: SYSNAME

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

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

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

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

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

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

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

###################################
#####     SSL configuration    #####
###################################
gapwalk.ssl.enabled : true
gapwalk.ssl.trustStore : "./config/clientkey.jks"
gapwalk.ssl.trustStorePassword : mysslcertifpassword

#################################
##### MQ settings #####
#################################
mq.queues: jmsmq
mq.queues.jmsMQQueueManagers[0].jmsMQQueueManager: QM1
mq.queues.jmsMQQueueManagers[0].jmsMQAppName: Gapwalk
mq.queues.jmsMQQueueManagers[0].jmsMQChannel: DEV.APP.SVRCONN
mq.queues.jmsMQQueueManagers[0].jmsMQHost: localhost
mq.queues.jmsMQQueueManagers[0].jmsMQPort: 1415
mq.queues.jmsMQQueueManagers[0].jmsMQUserid: app
mq.queues.jmsMQQueueManagers[0].jmsMQSSLCipher: "*TLS12ORHIGHER"
mq.queues.jmsMQQueueManagers[1].jmsMQQueueManager: QM2
mq.queues.jmsMQQueueManagers[1].jmsMQAppName: Gapwalk
mq.queues.jmsMQQueueManagers[1].jmsMQChannel: DEV.APP.SVRCONN
mq.queues.jmsMQQueueManagers[1].jmsMQHost: localhost
mq.queues.jmsMQQueueManagers[1].jmsMQPort: 1415
mq.queues.jmsMQQueueManagers[1].jmsMQUserid: app

###############################
##### 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

#########################################################
#####  BLOCKSIZE DEFAULT VALUE  ####
#########################################################
#blockSizeDefault: 32760

#####################################
#####  JOBQUEUE CONFIGURATION    ####
#####################################
jobqueue:
  api.enabled: false
  impl: none # possible values: quartz, none
```

```
  schedulers:  # list of schedulers
    -
      name: queue1
      threadCount: 5
    -
      name: queue2
      threadCount: 5

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

########################################
#####  JCL Batch Restart Mechanism  ####
########################################
jcl:
checkpoint:
enabled: false
#expireTimeout: -1
#expireTimeoutUnit: SECONDS # Supported values: java.util.concurrent.TimeUnit
#provider: redis

----
```

- Use variable length files with LISTCAT commands

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

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

- Provide Null Byte Indicator Value in LOAD/UNLOAD utility

```
# 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.

Key	Type	Default value	Description	Release version
`logging.config`	Path	classpath:logback-main.xml	Standard key for the reference to the logback configuration file. Other standard logging keys are available too.	
`spring.jta.enabled`	boolean	false	Standard key. If the datasource support mode	

Key	Type	Default value	Description	Release version
			is not static-xa, spring JTA transacti ons autoconfi guration must be disabled.	
`datasourc e.jicsDs + -driver-class-name + -url + -username + -password + -type`	Standard spring datasource with subkeys		Contains the connection information for the Jics database. Alternately, use of AWS secrets is strongly encouraged, as explained in the section called “JICS database”.	
`datasourc e.bluesam Ds + -driver-class-name + -url + -username + -password + -type`	Standard spring datasource with subkeys		Contains the connection information for the Blusam database. Alternately, use of AWS secrets is strongly encouraged, as explained in the section called “Blusam database”.	

Key	Type	Default value	Description	Release version
`bluesam.disabled`	boolean	false	Whether to disable Blusam entirely.	
`bluesam.cache`	string		If not set, the Blusam cache will not be used. Possible values (cache implementations) are cache and redis (the section called "Redis cache properties").	
`bluesam.maxBluesamDisablingThreadpoolSize`	number	10	Specifies maximum threadpool size used to disable bluesam datasets for batch processing.	4.5.0
`bluesam.bluesamStatusPollingInterval`	number	1000	Specifies time (in millisecond) to wait between each iteration when polling bluesam status to check online activities.	4.5.0

Key	Type	Default value	Description	Release version
`bluesam.maxBluesamStatusPollingRetry`	number	3	Specifies maximum number of retries when polling bluesam status is failing.	4.5.0
`bluesam.checkBluesamStatus`	boolean	false	Specifies whether or not to check bluesam dataset status before accessing it.	4.5.0
`spring.aws.client.bluesam.redis.secret`	string	null	Specifies the credential secret ARN for Bluesam Redis cache, see the section called “AWS Blu Age Runtime secrets”.	
`spring.aws.client.bluesam.locks.redis.secret`	string	null	Specifies the credential secret ARN for Bluesam locks Redis cache, see the section called “AWS Blu Age Runtime secrets”.	

Key	Type	Default value	Description	Release version
`forcedDate`	string		Forces the date to the date provided if there is one.	
`frozenDate`	boolean	true	Specifies whether to freeze the date. Applies only if `forcedDate` is also set.	
`externalSort.threshold`	datasize (example: 12 MB)		The sort threshold: when to switch to external (merge) sort.	
`blockSizeDefault`	number	32760	The default block size to be used for BDW bytes.	
`jics.parameters.datform`	string	MMDDYY	The date form.	

Key	Type	Default value	Description	Release version
`jics.initList`	string		The initialize JICS list, separated by commas. If present, it defines comma-separated names of lists to activate at Apache 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.	
`jics.parameters.applid`	string	VELOCITY	The applied to identify application in JICS (at least 4 characters, no max length).	

Key	Type	Default value	Description	Release version
jics.parameters.sysid	string	CICS	The system identification (SYSID).	
jics.parameters.eibtrmid	string	TERM	The terminal identifier (4 characters maximum, 1 minimum).	
jics.parameters.userid	string		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.	
jics.parameters.username	string	MYUSERNAME	The username (10 characters maximum, 1 minimum).	
jics.parameters.netname	string	MYNETNAME	The network name (8 characters maximum, 1 minimum).	
jics.parameters.opid	string	XXX	The 3-character operator identification.	

Key	Type	Default value	Description	Release version
`jics.parameters.jobname`	string	MJOBNAME	The job name.	
`jics.parameters.sysname`	string	SYSNAME	The AS400 system name (sysname).	
`jics.parameters.cwa.length`	number	0	The length of the common work area (CWA).	
`jics.parameters.charset`	string	CP037	JICS globally used character set.	
`jics.parameters.tsqimpl`	string	bluesam	JICS Temporary Storage Queue (TSQ) implementation (allowed values are`bluesam` /`memory` /`redis`)	
`jics.queues.ts.redis.*`	Supported Redis properties		Specifies configuration properties for the JICS TS Queues Redis server, see the section called "Supported Redis properties".	

Key	Type	Default value	Description	Release version
`spring.aws.client.jics.queues.ts.redis.secret`	string	null	Specifies the credential secret ARN for JICS TS Queues Redis server, see the section called “AWS Blu Age Runtime secrets”.	
`lockTimeout`	number	500	The lock timeout, in milliseconds.	
`sqlCodePointShift`	number		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 can specify384 to match Unicode character `\u0180`.	

Key	Type	Default value	Description	Release version
`sqlIntegerOverflowAllowed`	boolean	false	Specifies whether to allow the SQL integer overflow, meaning whether placing larger values in the host variable is allowed.	
`database.cursor.overflow.allowed`	boolean	true	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).	
`reportOutputPath`	string	`/reports`	The report output path.	

Key	Type	Default value	Description	Release version
`spring.session.store-type`	string	none	The session cache for high-availability environments. Possible values are `none` or `redis`. Default is `none`.	
`stopExecutionWhenProgNotFound`	boolean	true	Specifies whether to stop running if a program isn't found. If set to `true`, interrupts the run if a program is not found.	
`forceHR`	boolean	false	Specifies whether to use Human Readable SYSPRINT, either on console or file output.	
`rollbackOnRTE`	boolean	false	Specifies whether to rollback implicit run unit transaction on runtime exceptions.	

Key	Type	Default value	Description	Release version
`sctThreadLimit`	long	5	The thread limit for triggering scripts.	
`dataSimplifier.onInvalidNumericData`	string	reject	How to react when decoding invalid numeric data. Allowed values are`reject` /`tolerates paces` /`tolerates paceslowvalues` /`toleratemost` . Default is`reject`.	
`filesDirectory`	string		The directory for batches input/output files.	
`ims.messages.extendedSize`	boolean	false	Specifies whether to set the extended size on IMS messages.	
`defaultKeepExistingFiles`	boolean	false	Specifies whether to set the dataset default previous value.	

Key	Type	Default value	Description	Release version
`jics.db.ddlScriptLocation`	string		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` .	
`jics.db.schemaTestQueryLocation`	string		Location of the sql file that should contain a unique query that returns the number of objects in the jics schema (if any).	

Key	Type	Default value	Description	Release version
`jics.db.dataScriptLocation`	string		Defines the path to SQL scripts used for initializing the JICS database. Accepts a comma-separated list of files and directories, allowing for multiple scripts and folders to be specified.	

Key	Type	Default value	Description	Release version
`jics.db.dataTestQueryLocation`	string		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.	
`jics.data.dataJsonInitLocation`	string			
`jics.xa.agent.timeout`	number			

Key	Type	Default value	Description	Release version
query.useConcatCondition	boolean	false	Specifies whether key condition is built by key concatenation or not.	
system.qdecfmt	string			
disposition.checkexistence	boolean	false	Specifies whether to release a check on file existence for Dataset with DISP SHR or OLD.	
useControlMVariable	boolean	false	Specifies whether to use control-M specification for variable replacement.	
card.encoding	string	CP1145	Card encoding: to be used withuseControlMVariable .	

Key	Type	Default value	Description	Release version
mapTransfo.prefixes	string	&,@,%%	List of prefixes to be used when transforming controlM variables. Each one separated by comma.	
checkinputfilesize	boolean	false	Specifies whether to release a check if the file size is a multiple of record size.	
stepFailWhenAbend	boolean	true	Specifies whether to raise an abend if a step fails or completes execution.	
bluesam.fileLoading.commitInterval	number	100000	The bluesam commit interval.	
uppercaseUserInput	boolean	true	Specifies whether user input must be in uppercase.	

Key	Type	Default value	Description	Release version
jhdb.lterm	string		Allow you to force a common logical terminal ID in the case of an IMS emulation. If not set then sessionId is used.	
jhdb.identificationCardData	string		Used to hard-code some "operator identification card data" to the MID field designated by the CARD parameter. Blank by default, no input restriction.	
encoding	string	ASCII	The encoding used in projects (not in groovy files). Expects a valid encodingCP1047,I	8...
cl.configuration.context.encoding	string	CP297	The encoding of CL files. Expects a valid encodingCP1047,I Default value isCP297	8...

Key	Type	Default value	Description	Release version
cl.zonedMode	string	EBCDIC_STRICT	The mode for encoding or decoding control language (CL) commands. Allowed values areEBCDIC_STRICT/EBCDIC_MODIFIED/AS400.	
ims.programs	string		List of IMS programs to use. Separate each parameter with a semicolon (;) and each transaction with a comma (,). For example:PCP008,T008;PCP054,PCT054;PCP066,PCT066;PCP068,PCT068;	
jhdb.configuration.context.encoding	string	CP297	The JHDB (Java Hierarchical Database) encoding. Expects a valid encoding stringCP1047,IBM	

Key	Type	Default value	Description	Release version
`jhdb.metadata.extrapath`	string	file:./setup/	A configuration parameter that specifies an extra, runtime-specific root folder for psbs and dbds folders.	

Key	Type	Default value	Description	Release version
jhdb.checkpointPersistence	string	none	The checkpoint persistence mode. Allowed values arenone /add /end. Useadd to persist checkpoints when a new one is created and added to the registry. Useend 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	

Key	Type	Default value	Description	Release version
			of checkpoin ts, it can have some effects on performances.	
`jhdb.chec kpointPath`	string	file:./setup/	If`jhdb.chec kpointPer sistence` is notnone 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 (checkpoi nt.dat) located in provided folder. Note that only checkpoin t data (scriptId, stepId, database position and checkpoint area) are concerned by this backup.	

Key	Type	Default value	Description	Release version
`jhdb.navigation.cachenexts`	number	5000	The cache duration (in milliseconds) used in hierarchical navigation for an RDBMS.	
`jhdb.use-db-prefix`	boolean	true	Specifies whether to enable a database prefix in hierarchical navigation for an RDBMS.	
`jhdb.query.limitJoinUsage`	boolean	true	Specifies whether to use the limit join usage parameter on RDBMS graphs.	
`taskExecutor.corePoolSize`	number	5	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.	

Key	Type	Default value	Description	Release version
`taskExecutor.maxPoolSize`	number	10	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).	
`taskExecutor.queueCapacity`	number	50	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)	

Key	Type	Default value	Description	Release version
`taskExecutor.allowCoreThreadTimeOut`	boolean	false	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).	
`jics.runUnitLauncherPool.enable`	boolean	false	Specifies whether to activate the run unit launcher pool in JICS.	
`jics.runUnitLauncherPool.size`	number	20	The run unit launcher pool size in JICS.	
`jics.runUnitLauncherPool.validationInterval`	number	1000	The interval between each run of the task that adjusts the size of the pool.	

Key	Type	Default value	Description	Release version
`jics.runUnitLauncherPool.parallelism`	number	2	The number of threads used to produce the missing instances in the queue when the adjustment task runs.	
`context.preconstruct.enable`	boolean	false	Specifies whether to activate pre construction of program context.	
`context.preconstruct.frequencyInMillis`	number	100	The interval between each run of the task that adjusts the size of the pool.	
`context.preconstruct.parallelism`	number	5	The number of threads used to produce the missing instances in the queue when the adjustment task runs.	

Key	Type	Default value	Description	Release version
`context.preconstruct.minInstances`	number	2	The number of instances that will be created the first time a context is needed.	
`spring.aws.application.credentials`	string	null	Load the AWS credentials from the credential profiles file in JICS.	
`jics.queues.sqs.region`	string	eu-west-1	The AWS Region for Amazon Simple Queue Service, used in JICS.	
`jics.jcl.rt.encoding`	string	CP037	The encoding of the JCL scripts written in the dedicated JICS queue.	
`jics.jcl.rt.queue`	string	JICS	The name of the queue to which JCL scripts can be written line by line at runtime.	
`mq.queues.sqs.region`	string	eu-west-3	The AWS Region for the AWS SQS MQ service.	

Key	Type	Default value	Description	Release version
quartz.scheduler.stand-by-if-error	boolean	false	Specifies whether to trigger job execution if the job scheduler is in standby mode. If true, When enabled job execution is not triggered.	
databaseStatistics	boolean	false	Specifies whether to allow SQL builders to collect and display statistics information.	
dbDateFormat	string	yyyy-MM-dd	The db target date format.	
dbTimeFormat	string	HH:mm:ss	The db target time format.	
dbTimestampFormat	string	yyyy-MM-dd HH:mm:ss.SSSSSS	The db target timestamp format.	

Key	Type	Default value	Description	Release version
dateTimeFormat	string	ISO	The dateTimeFormat describes how to spill database date time timestamp type into data simplifier entities. Allowed values areISO /EUR /EUR /USA /LOCAL	
localDateFormat	string		List of local date formats.Separate each format with\.	
localTimeFormat	string		List of local time formats. Separate each format with\	
localTimestampFormat	string		List of local timestamp formats. Separate each format with\.	
pgmDateFormat	string	yyyy-MM-dd	The date time format.	
pgmTimeFormat	string	HH.mm.ss	The time format used for pgm (programs) execution.	

Key	Type	Default value	Description	Release version
pgmTimestampFormat	string	yyyy-MM-dd-HH.mm.ss.SSSSSS	The timestamp format.	
cacheMetadata	boolean	true	Specifies whether to cache database metadata.	
forceDisableSQLTrimStringType	boolean	false	Specifies whether to disable trim of all sql string parameters.	
fetchSize	number		The fetchSize value for cursors. Use when fetching data using chunks by load/unload utils.	
check-groovy-file	boolean	true	Specifies whether to check groovy files content before registering.	
qtemp.uuid.length	number	9	The QTEMP unique id length.	

Key	Type	Default value	Description	Release version
`qtemp.dblog`	boolean	false	Whether to enable QTEMP Database logging.	
`qtemp.cleanup.threshold.hours`	number	0	To specify when`qtemp.dblog` is enabled. The db partition lifetime (in hours).	
`sort.function`	string		The sort function name for the blu4iv database.	
`invalidDataTolerence`	boolean	true	Specifies whether invalid data is tolerated for packed type.	
`program.timeout`	number	-1	Specifies a timeout for any program/ transaction execution in seconds. After this time, the system will try to interrupt the program.	

Key	Type	Default value	Description	Release version
`gapwalk.line.separator`	string	null	Specifies line separator type in gapwalk. The allowed values are WIN (CRLF) / UNIX (LF) / LINUX (LF). Other values are ignored and System line.separator property is used.	
`enableActivePgmIdCache`	boolean	false	Specifies whether to enable active program ID local cache. Use carefully this feature because JICS resources can be shared amongst programs and users. Those resources can be changed externally by any administrators and the local cache put in place might be invalidated.	

Key	Type	Default value	Description	Release version
`mq.queues.default.syncpoint`	boolean	false	Specifies the default behavior for MQ PUT commands when neither MQPMO_SYNCPOINT nor MQPMO_NO_SYNCPOINT are set. When set to true, it acts as `MQPMO_SYNCPOINT` and messages are NOT directly committed during the PUT command. When set to false, it acts as `MQPMO_NO_SYNCPOINT` and messages are directly committed during the PUT command.	

Key	Type	Default value	Description	Release version
dataSimplifier.byteRangeBoundsCheck	boolean	false	When set to true, it ensures that no ByteRange is created with improper values. The default is false.	
file.stdoutIntoLogger	boolean	false	Specifies whether to enable writing to logger instead of the default system output stream in the default SYSPRINT and SYSPUNCH files.	
tempFilesDirectory	string	null	Specifies the name of the folder location of the temporary files that are generated.	

Key	Type	Default value	Description	Release version
`cleanTempFilesDirectoryAtStartup`	boolean	true	Specifies whether to purge the contents of the temporary files folder at application startup.	

Key	Type	Default value	Description	Release version
tempFolderPattern	string	null	Specifies a pattern that will be used to dynamically build the name of the temporary folder based on the following predefined and customizable information. HOST: the host name. JOBID: the ID of the job. HASHCODE: the hash code of the job context. TIMESTAMP: the pattern to use when getting the timestamp. Target name of the temporary folder is TMP_DIR_{tempFolderPattern}. For example, in the case of the following	

Key	Type	Default value	Description	Release version
			pattern, the name will start with the job ID and end with the “timestamp”: tempFolderPattern: JOBID,HOST=xxxxx,HASHCODE,TIMESTAMP=yyyymmddhhmmss. If the property `tempFolderPattern` is not added to the YAML file or is empty, the name of the temporary folder will be "TMP_DIR_" + this.hashCode() (DefaultJobContext).	
`database.cursor.raise.already.opened.error`	boolean	false	Specifies whether to enable raising SQLCODE error 502 when an already opened cursor is opening.	

Key	Type	Default value	Description	Release version
`jics.spool.smtp.hostname`	string	null	Specifies the SMTP server host. Example: `smtp.xxx.com`	
`jics.spool.smtp.port`	string	null	Specifies the SMTP server port. Example: 25	
`jics.spool.smtp.password`	string	null	Specifies the login password of the SMTP server.	
`jics.spool.smtp.username`	string	null	Specifies the username of the SMTP server.	
`jics.spool.smtp.debug`	boolean	false	Specifies the debug mode for the SMTP server.	
`gapwalk-application.security`	string	`disabled`	Toggle global security configuration (XSS, CORS, CSRF, OAUTH authentication...). Allowed values are `disabled` and `enabled`.	

Key	Type	Default value	Description	Release version
gapwalk-application.identity	string	null	Global authentication method. Recommended value is oauth. Allowed values are json and oauth. This option is required when gapwalk-application.security is enabled.	
gapwalk-application.security.issuerUri	string	null	The issuer URI of the identity provider (IdP). This option is required when gapwalk-application.identity is oauth.	
gapwalk-application.security.allowedOrigins	string[]	null	The list of origins to allow. This option requires gapwalk-application.identity to be set to oauth.	

Key	Type	Default value	Description	Release version
`gdgDirectoryPath`	string	`output/gdg`	GDG directory path is the directory where the gdg files are stored.	4.6.0
`gapwalk-application.security.claimGroupName`	string	`cognito:groups`	The claim attribute that contains the list of all the groups a user belongs to. Use `cognito:groups` for Amazon Cognito, or any other string for a foreign IdP.	
`gapwalk-application.security.userAttributeName`	string	`username`	The claim attribute name used to identify a user request. Use `username` for Amazon Cognito, `preferred_username` for Keycloak, or any other string for a foreign IdP.	

Key	Type	Default value	Description	Release version
gapwalk-application.security.localhostWhitelistingEnabled	boolean	true	Specifies whether to enable authentication from any localhost requests.	
gapwalk-application.defaultSuperAdminUserName	string	sadmin	When gapwalk-application.security is disabled, specifies the default local super user name.	
gapwalk-application.defaultSuperAdminUserPwd	string	sadmin	When gapwalk-application.security is disabled, specifies the default local super user password.	
gapwalk-application.security.filterURIs	string	disabled	Toggle filtering URIs configuration. Allowed values are disabled and enabled.	

Key	Type	Default value	Description	Release version
`gapwalk-application.security.blockedURIs`	string[]	null	The list of URIs to block. This option is required when `gapwalk-application.security.filterURIs` is `enabled`.	
`jics.redis.*`	Supported Redis properties		Specifies configuration properties for the JICS Redis server connection factory, see the section called "Supported Redis properties".	
`spring.aws.client.jics.redis.secret`	string	null	Specifies the credential secret ARN for JICS Redis server connection factory, see the section called "AWS Blu Age Runtime secrets".	

Key	Type	Default value	Description	Release version
`jcl.checkpoint.enabled`	boolean	false	Specifies if JCL checkpoint mechanism is enabled to allow job restart. JCL checkpoints are created and saved to the in-memory registry at the start of each step or main program invocation. All the step level checkpoints are persisted at the end of the job, if persistence provider is defined.	
`jcl.checkpoint.expireTimeout`	number	-1	Specifies time duration to retain JCL checkpoints in the persistence provider or in-memory registry.	

Key	Type	Default value	Description	Release version
`jcl.checkpoint.expireTimeoutUnit`	string	SECONDS	Specifies time duration unit for the `jcl.checkpoint.expireTimeout` property. Supported enum constant values: java.util.concurrent.TimeUnit.	
`jcl.checkpoint.provider`	string	null	Specifies JCL checkpoint mechanism persistence provider. Allowed values are `redis`.	
`jcl.checkpoint.redis.*`	Supported Redis properties		Specifies configuration properties for JCL checkpoint mechanism's REDIS persistence provider, see the section called "Supported Redis properties".	

Key	Type	Default value	Description	Release version
`spring.aws.client.jcl.checkpoint.redis.secret`	string	null	Specifies the credential secret ARN for JCL checkpoint mechanism's Redis persistence provider, see the section called "AWS Blu Age Runtime secrets".	
`gapwalk.ssl.enabled`	boolean	false	Indicated to set the following `gapwalk.ssl.*` properties to the current JVM system properties if there are not already set at application start.	
`gapwalk.ssl.trustStore`	string	null	Set the value to the system property `javax.net.ssl.trustStore` if not already set at application start.	

Key	Type	Default value	Description	Release version
gapwalk.ssl.trustStorePassword	string	null	Set the value to the system property javax.net.ssl.trustStorePassword if not already setup at application start. Alternately, use of AWS secrets is strongly encouraged, as explained in the section called "Secret manager for SSL password settings".	
gapwalk.ssl.trustStoreType	string	null	Set the value to the system property javax.net.ssl.trustStoreType if not already setup at application start.	

Key	Type	Default value	Description	Release version
`gapwalk.ssl.keyStore`	string	null	Set the value to the system property `javax.net.ssl.keyStore` if not already setup at application start.	
`gapwalk.ssl.keyStorePassword`	string	null	Set the value to the system property `javax.net.ssl.keyStorePassword` if not already setup at application start. Alternately, use of AWS secrets is strongly encouraged, as explained in the section called “Secret manager for SSL password settings”.	

Key	Type	Default value	Description	Release version
mq.queues	string	sqs	Specifies which supported queue brocker to use among sqs using Amazon SQS, rabbitmq using on-prem Rabbit MQ or jms using on-prem IBMMQ.	
mq.queues.jmsMQQueueManagers[N]			When mq.queues is jms, enables to specify an IBM MQ connection list. mq.queues.jmsMQQueueManagers[0] for the first connection, mq.queues.jmsMQQueueManagers[1] for the second and so on.	
mq.queues.jmsMQQueueManagers[N].jmsMQQueueManager	string	null	The IBMMQ queue manager name.	

Key	Type	Default value	Description	Release version
mq.queues.jmsMQQueueManagers[N].jmsMQAppName	string	null	The IBMMQ application name.	
mq.queues.jmsMQQueueManagers[N].jmsMQChannel	string	null	The IBMMQ channel name.	
mq.queues.jmsMQQueueManagers[N].jmsMQHost	string	null	The IBMMQ hostname.	
mq.queues.jmsMQQueueManagers[N].jmsMQPort	number	null	The IBMMQ port.	
mq.queues.jmsMQQueueManagers[N].jmsMQUserid	string	null	The IBMMQ user name.	

Key	Type	Default value	Description	Release version
`mq.queues.jmsMQQueueManagers[N].jmsMQPassword`	string	null	The IBMMQ user password. Alternately, use of AWS secrets is strongly encouraged, as explained in the section called “Secret manager for IBM MQ password settings”.	
`mq.queues.jmsMQQueueManagers[N].jmsMQMaxPoolSize`	number	0	The IBMMQ maximum pool size . With 0, an infinite number of physical connections are enabled.	
`mq.queues.jmsMQQueueManagers[N].jmsMQSSLCipher`	string	null	The IBMMQ SSL cipher suite. An example could be `"*TLS120RHIGHER"` . Refer to the official documentation TLS CipherSpecs and CipherSuites in IBM MQ classes for JMS for more details.	

Key	Type	Default value	Description	Release version
mq.queues.non.jms.client	boolean	false	Indicate whether target client for sending messages to is non-JMS. Native MQ format will be used for non-JMS client while RFH2 format will be used for JMS.	4.5.0
			When mq.queues is rabbitmq, The IBMMQ hostname.	
mq.queues.rabbitMQHost			The Rabbit MQ hostname.	
mq.queues.rabbitMQVirtualHost			The Rabbit MQ virtual hostname.	
mq.queues.rabbitMQPort			The Rabbit MQ port.	
mq.queues.rabbitMQUsername			The Rabbit MQ user.	

Key	Type	Default value	Description	Release version
mq.queues.rabbitMQPassword			The Rabbit MQ password.	
mf.runtime.switch.N	boolean	true	Enables null insertion for MF nature line-sequential files.	4.4.0
mf.runtime.switch.T	boolean	false	Enables insertion of tab characters in MF nature line-sequential files.	4.4.0
gapwalk.database.support.useSavePointToRestoreFail	boolean	false	Enables transaction recovery in case of failure by using savepoints on insert queries. Enabling this property may impact database performance. You can override this setting for specific queries using the query-to-database mapping configuration.	4.6.0

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.

Key	Type	Default value	Description	Release version
`logging.config`	Path	classpath:logback-utility.xml	Standard key for the reference to the logback configuration file. Other standard logging keys are available too.	
`spring.jta.enabled`	boolean	false	Standard key. If the datasource support mode is not static-xa, spring JTA transactions auto configuration must be disabled.	
`spring.datasource.primary.jndi-name`	string	jdbc/primary	The JNDI name (Java Naming And Directory Interface) for the primary	

Key	Type	Default value	Description	Release version
			datasource, if using JNDI.	
`primary.datasource -driver-class-name -url -username -password`	Standard spring datasource with subkeys		Contains the connection information for the application database, if not using JNDI. Must have the same configuration as in the modernized application YAML file. Alternately, use of AWS secrets is strongly encouraged, as explained in the section called “Client database”.	
`encoding`	string	ASCII	The encoding used in utility programs. Expects a valid encoding `CP1047,IBM930,A`	
`sysPunchEncoding`	string	ASCII	The syspunch encoding character set. Expects a valid encoding`CP1047,I`	8.

Key	Type	Default value	Description	Release version
`systin.encoding`	string	ASCII	The encoding character set of SYSTIN file dataset. Expects a valid encoding`CP1047,I`	8.
`zonedMode`	string	EBCDIC_STRICT	The mode for encoding or decoding zoned data types. Allowed values are`EBCDIC_STRICT` `/EBCDIC_MODIFIED` `/AS400`.	
`idcams.encoding.forced`	string		The encoding used in IDCAMS utility program. Expects a valid encoding `CP1047`, `IBM930`, `ASCII`, `UTF-8`.	4.4.0
`unload.chunkSize`	number	0	Chunk size used for unload utility.	

Key	Type	Default value	Description	Release version
`unload.computeRecordSizeIfNull`	boolean	false	Determines whether to compute the record size if not specified. If specified, the value remains unchanged.	
`unload.sqlCodePointShift`	number	0	The SQL code pointshift for unload utility. Runs the shifting character s process. Required when your target database from DB2 is Postgresql.	
`unload.columnFiller`	string	space	The unload utility column filler.	
`unload.varCharIsNull`	boolean	false	Use this parameter in INFTILB program, if set to`true` then all not nullable fields with blank (space) values returns an empty string.	

Key	Type	Default value	Description	Release version
`unload.useDatabaseConfiguration`	boolean	false	Specifies whether to use the date or time configuration from application-main.yml in unload utility.	
`unload.format.date`	string	MM/dd/yyyy	If `unload.useDatabaseConfiguration` is enabled, the date format to use in the unload utility.	
`unload.format.time`	string	HH.mm.ss	If `unload.useDatabaseConfiguration` is enabled, the time format to use in the unload utility.	
`unload.format.timestamp`	string	yyyy-MM-dd-HH.mm.ss.SSSSSS	If `unload.useDatabaseConfiguration` is enabled, the timestamp format to use in the unload utility.	

Key	Type	Default value	Description	Release version
`unload.nbi.whenNull`	hexadecimal	6F	The Null Byte Indicator (NBI) value to add when value from database is null.	
`unload.nbi.whenNotNull`	hexadecimal	00	The Null Byte Indicator (NBI) value to add when value from database is not null.	
`unload.nbi.writeNullIndicator`	boolean	false	Specifies whether to write out the null indicator in the unload output file.	
`unload.bmc.useInto`	boolean	false	Specifies whether to handle INTO bmc control keyword for unload utility.	
`unload.fetchSize`	number	0	Allows you to tune the fetch size when handling cursors in the unload utility.	

Key	Type	Default value	Description	Release version
unload.noPad	boolean	true	Indicates that variable-length character (VARCHAR) fields are to be unloaded without any padding to the maximum length.	4.5.0
treatLargeNumberAsInteger	boolean	false	Specifies whether to treat large numbers as Integer. They are treated as BigDecimal by default.	
load.batchSize	number	0	The load utility batch size.	
load.format.localDate	string	dd.MM.yyyy\dd/MM/yyyy\yyyy-MM-dd	The load utility local date format to use.	
load.format.localTime	string	HH:mm:ss\HH.mm.ss	The load utility local time format to use.	
load.format.dbDate	string	yyyy-MM-dd	The load utility database format to use.	

Key	Type	Default value	Description	Release version
`load.format.dbTime`	string	HH:mm:ss	The load utility database time to use.	
`load.sqlCodePointShift`	number	0s	The SQL code pointshift for load utility. Runs the shifting characters process. Required when your target database from DB2 is Postgresql.	
`load.applyRollback`	boolean	false	Set this parameter to `true` to indicate that you want the service to roll back table changes if it encounters an error while loading data into the database.	
`forcedDate`	string		Forces the date to the date provided if there is one.	

Key	Type	Default value	Description	Release version
`frozenDate`	boolean	true	Specifies whether to freeze the date. Applies only if `forcedDate` is also set.	
`jcl.type`	string	mvs	.jcl file type. Allowed values are `jcl` /`vse`. The IDCAMS utility PRINT/REP RO commands return 4 if the file is empty for non-vse jcl.	
`hasGraphic`	boolean	false	Whether the INFUTILB utility needs to handle GRAPHIC DB2 columns.	
`convertGr aphicData ToFullWid th`	boolean	true	Specifies whether to convert graphic data to full-widt h format.	

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.

Key	Type	Default value	Description
`logging.config`	Path	classpath:logback-utility.xml	Standard key for the reference to the logback configuration file. Other standard logging keys are available too.
`spring.jta.enabled`	boolean	false	Standard key. If the datasource support mode is not static-xa, spring JTA transactions auto configuration must be disabled.
`spring.datasource.primary.jndi-name`	string	jdbc/primary	The JNDI name (Java Naming And Directory Interface) for the primary datasource, if using JNDI.
`primary.datasource + -driver-class-name + -url + -username + -password`	Standard spring datasource with subkeys		Contains the connection information for the application database, if not using JNDI. Must have the same configuration as in the modernized application YAML file. Alternately, use of AWS secrets is strongly encouraged, as explained in the

Key	Type	Default value	Description
			section called “Client database”.
encoding	string	ASCII	The encoding used in utility programs. Expects a valid encodingCP1047,IBM930,ASC
zonedMode	string	EBCDIC_STRICT	The mode for encoding or decoding zoned data types. Allowed values areEBCDIC_ST RICT /EBCDIC_MO DIFIED /AS400.
commands-off	string		List of commands to turn off, separated by comma. Allowed values arePGM_BASIC ,RCVMSG,SNDRCVF,CHGVAR,Q Useful when you want to disable or overwrite an existing program. PGM_BASIC is a specific AWS Blu Age Runtime program designed for debugging purposes.
forcedDate	string		Forces the date to the date provided if there is one.

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.

Key	Type	Default value	Description
`logging.config`	Path	classpath:logback-utility.xml	Standard key for the reference to the logback configuration file. Other standard logging keys are available too.
`spring.jta.enabled`	boolean	false	Standard key. If the datasource support mode is not static-xa, spring JTA transactions auto configuration must be disabled.
`jhdb.configuration.context.encoding`	string		The JHDB (Java Hierarchical Database) encoding. Expects a valid encoding stringCP1047,IBM930,ASCII,
`jhdb.checkpointPersistence`	string	none	The checkpoint persistence mode. Allowed values arenone /add /end. Useadd to persist checkpoints when a new one is created and added to the

Key	Type	Default value	Description
			registry. Useend 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 checkpoin ts will be serialize d 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.

Available properties for the client application

Your modernized application may require specific property configurations for the client Spring application. These properties initialize beans from classes packaged in runtime JAR files. The `application-profile.yaml` file, where the profile value is set during application generation, allows you to configure these properties. The following table lists the key/value parameters available for configuring the client web application that uses beans from classes packaged in the Gapwalk runtime

Key	Type	Default value	Description	Release version
`blu4iv.dt aara.libr`	boolean	false	Controls the usage of library in the context of data area	4.5.0

Key	Type	Default value	Description	Release version
`ary.disab le`			operations. If set to true, library usage is disabled for data area operation s, but this does not affect the usage of QTemp. If set to false, library is considered when performing CRUD operation s for data area.	

Available Redis cache properties in AWS Blu Age Runtime

You can use this document to learn about the Redis caches in AWS Blu Age Runtime, along with Gapwalk configuration, supported Redis properties and how `application-main.yml` file can reference secret ARN for Redis caches.

Redis caches in AWS Blu Age Runtime

Redis servers can be used as caches for various features in the AWS Blu Age Gapwalk application, such as:

AWS Blu Age Runtime features that use Redis caching	Description
Blusam cache	A Redis Blusam cache for reading records efficiently, using a write-behind strategy, to optimize write-intensive workloads encounter ed on batch payloads.

AWS Blu Age Runtime features that use Redis caching	Description
Blusam locks	A cache for distributed locks for datasets and records.
Dataset catalog	The catalog dataset cache.
Session cache	A Redis cache for HttpSession. The cache stores the username, the state of the dialogue with the Angular frontend, and specific 'dialect' (BMS, MFS, AS400) information.
Session tracker	A cache of active sessions with associated username and session-creation-time informati on.
JICS cache	A cache for JICS resource definitions.
TS queues	Storage for TS queues.
JCL checkpoint	JCL checkpoint cache.
Gapwalk file locks	A cache for distributed file locks by job.
Blu4iv locks	Storage for Blu4iv record locks.

Redis Gapwalk configuration

The global Redis configuration is used if `redis` is specified as the caching mechanism and no Redis configuration is provided for the specific feature. This configuration makes it possible for you to use the same configuration for multiple Redis caches simultaneously.

In the following example the Blusam datasets cache and JICS cache use the `gapwalk.redis` (`redis.server1`) configuration because their cache type is set to `redis`, and no implicit Redis properties are specified under the section called “JICS resource definitions” and the section called “JICS resource definitions”. However, the Blusam locks cache will use a different Redis configuration (`redis.server2`) because its Redis properties are explicitly defined.

```
...

 gapwalk:
   redis:
     hostName: redis.server1
  port: 6379
 ...

 bluesam:
   # Redis bluesam cache
   cache: redis
   # Redis locks cache
   locks:
     cache: redis
  hostName: redis.server2
  port: 6379
 ...
 # Redis jics cache
 jics:
   resource-definitions:
     store-type: redis
 ...
```

To enable the global Redis configuration, add the following configuration in `main-application.yml`.

```
gapwalk:
  redis:
    hostName: localhost
    port: 6379
    mode: standalone                    # Optional
    username:                           # Optional
    password: ""                        # Optional
    useSsl: false                       # Optional
    database: 0                         # Optional
    maxTotal: 128                       # Optional
    maxIdle: 128                        # Optional
    minIdle: 16                         # Optional
    testOnBorrow: true                  # Optional
    testOnReturn: true                  # Optional
    testWhileIdle: true                 # Optional
    testOnCreate: true                  # Optional
```

```
    minEvictableIdleTimeMillis: 60000       # Optional
    timeBetweenEvictionRunsMillis: 30000    # Optional
    numTestsPerEvictionRun: -1              # Optional
    blockWhenExhausted: true                # Optional
    nettyThreads: 32                        # Optional
    subscriptionsPerConnection: 10          # Optional
    subscriptionConnectionPoolSize: 100     # Optional
    pageSizeInBytes: 8192                   # Optional
    readTimeout: 2000                       # Optional
```

Supported Redis properties

The following table shows the Redis properties that are supported for global and specific Redis caches on AWS Blu Age Runtime.

Property name	Required?	Description	Values	Default
`mode`	No	The Redis running mode.	`standalone | cluster`	`standalone`
`hostname`	Yes	The hostname or IP address of the Redis server.	string	null
`port`	Yes	The port number on which the Redis server is listening for connections.	int	null
`username`	No	The username for authentic ation.	string	null
`password`	No	The password for authentic ation.	string	empty string

Property name	Required?	Description	Values	Default
`useSsl`	No	Specifies whether to enable SSL/TLS encryption for the Redis connection.	boolean	false
`database`	No	The Redis database number to use. Redis supports multiple logical databases, and this property specifies which one to use.	int	0
`maxTotal`	No	The maximum number of connections allowed in the Redis connection pool.	int	128
`maxIdle`	No	The maximum number of idle connections allowed in the Redis connection pool.	int	128

Property name	Required?	Description	Values	Default
`minIdle`	No	The minimum number of idle connections to maintain in the Redis connection pool.	int	16
`testOnBorrow`	No	A boolean value indicating whether to validate connections before borrowing them from the pool.	boolean	true
`testOnReturn`	No	A boolean value indicating whether to validate connections before returning them to the pool.	boolean	true
`testWhileIdle`	No	A boolean value indicating whether to validate idle connections in the pool periodically.	boolean	true

Property name	Required?	Description	Values	Default
testOnCreate	No	A boolean value indicating whether to validate connections when they are created.	boolean	true
minEvictableIdleTimeMillis	No	The minimum amount of time (in milliseconds) that an idle connection must remain in the pool before it can be evicted.	long	60000L
timeBetweenEvictionRunsMillis	No	The time (in milliseconds) between successive runs of the idle connection evictor thread.	long	30000L
numTestsPerEvictionRun	No	The maximum number of connections to test during each run of the idle connection evictor thread.	int	-1

Property name	Required?	Description	Values	Default
`blockWhenExhausted`	No	A boolean value indicating whether to block and wait for a connection to become available when the pool is exhausted.	boolean	true
`nettyThreads`	No	The number of Netty threads to use for handling Redis connections.	int	32
`subscriptionsPerConnection`	No	The maximum number of subscriptions allowed per Redis connection.	int	10
`subscriptionConnectionPoolSize`	No	The maximum number of connections allowed in the Redis subscription connection pool.	int	100
`pageSizeInBytes`	No	The default page size in bytes for Redis operations.	long	262144000

Property name	Required?	Description	Values	Default
readTimeout	No	The read timeout in milliseconds for Redis operations.	long	2000
timeToLiveMillis	No	The duration (in Milliseconds) for which a cache entry remains in the cache before being considered expired and removed. If this property is not specified, cache entries will not automatically expire by default.	long	-1

Redis cache properties

Redis Blusam cache

```
bluesam:
   cache: redis
 # If the following redis properties are not specified gapwalk.redis configuration will
 be used for this cache
   redis:
     hostName: localhost
     port: 6379
     mode: standalone                        # Optional
     username:                               # Optional
     password: ""                            # Optional
     useSsl: false                           # Optional
```

```
    database: 0                              # Optional
    maxTotal: 128                            # Optional
    maxIdle: 128                             # Optional
    minIdle: 16                              # Optional
    testOnBorrow: true                       # Optional
    testOnReturn: true                       # Optional
    testWhileIdle: true                      # Optional
    testOnCreate: true                       # Optional
    minEvictableIdleTimeMillis: 60000        # Optional
    timeBetweenEvictionRunsMillis: 30000     # Optional
    numTestsPerEvictionRun: -1               # Optional
    blockWhenExhausted: true                 # Optional
    nettyThreads: 32                         # Optional
    subscriptionsPerConnection: 10           # Optional
    subscriptionConnectionPoolSize: 100      # Optional
    pageSizeInBytes: 8192                    # Optional
    readTimeout: 2000                        # Optional
    timeToLiveMillis: 60000                  # Optional
```

Redis Blusam cache

```
bluesam:
   locks:
     cache: redis
 # If the following redis properties are not specified gapwalk.redis configuration will
 be used for this cache
       hostName: localhost
       port: 6379
       mode: standalone                        # Optional
       username:                               # Optional
       password: ""                            # Optional
       useSsl: false                           # Optional
       database: 0                             # Optional
       maxTotal: 128                           # Optional
       maxIdle: 128                            # Optional
       minIdle: 16                             # Optional
       testOnBorrow: true                      # Optional
       testOnReturn: true                      # Optional
       testWhileIdle: true                     # Optional
       testOnCreate: true                      # Optional
       minEvictableIdleTimeMillis: 60000       # Optional
       timeBetweenEvictionRunsMillis: 30000    # Optional
       numTestsPerEvictionRun: -1              # Optional
```

```
      blockWhenExhausted: true                # Optional
      nettyThreads: 32                        # Optional
      subscriptionsPerConnection: 10          # Optional
      subscriptionConnectionPoolSize: 100     # Optional
      pageSizeInBytes: 8192                   # Optional
      readTimeout: 2000                       # Optional
```

Session cache

```
spring:
  session:
    store-type: redis
# If the following redis properties are not specified gapwalk.redis configuration will
be used for this cache
jics:
  redis:
    hostName: localhost
    port: 6379
    mode: standalone                        # Optional
    username:                               # Optional
    password: ""                            # Optional
    useSsl: false                           # Optional
    database: 0                             # Optional
    maxTotal: 128                           # Optional
    maxIdle: 128                            # Optional
    minIdle: 16                             # Optional
    testOnBorrow: true                      # Optional
    testOnReturn: true                      # Optional
    testWhileIdle: true                     # Optional
    testOnCreate: true                      # Optional
    minEvictableIdleTimeMillis: 60000       # Optional
    timeBetweenEvictionRunsMillis: 30000    # Optional
    numTestsPerEvictionRun: -1              # Optional
    blockWhenExhausted: true                # Optional
    nettyThreads: 32                        # Optional
    subscriptionsPerConnection: 10          # Optional
    subscriptionConnectionPoolSize: 100     # Optional
    pageSizeInBytes: 8192                   # Optional
    readTimeout: 2000                       # Optional
```

JICS resource definitions

```
jics:
   resource-definitions:
     store-type: redis
 # If the following redis properties are not specified gapwalk.redis configuration will
 be used for this cache
   redis:
     hostName: localhost
     port: 6379
     mode: standalone                        # Optional
     username:                               # Optional
     password: ""                            # Optional
     useSsl: false                           # Optional
     database: 0                             # Optional
     maxTotal: 128                           # Optional
     maxIdle: 128                            # Optional
     minIdle: 16                             # Optional
     testOnBorrow: true                      # Optional
     testOnReturn: true                      # Optional
     testWhileIdle: true                     # Optional
     testOnCreate: true                      # Optional
     minEvictableIdleTimeMillis: 60000       # Optional
     timeBetweenEvictionRunsMillis: 30000    # Optional
     numTestsPerEvictionRun: -1              # Optional
     blockWhenExhausted: true                # Optional
     nettyThreads: 32                        # Optional
     subscriptionsPerConnection: 10          # Optional
     subscriptionConnectionPoolSize: 100     # Optional
     pageSizeInBytes: 8192                   # Optional
     readTimeout: 2000                       # Optional
```

JICS TS queues

```
jics:
   parameters:
     tsqimpl: redis
 # If the following redis properties are not specified gapwalk.redis configuration will
 be used for this cache
   queues:
     ts:
       redis:
         hostName: localhost
```

```
        port: 6379
        mode: standalone                        # Optional
        username:                               # Optional
        password: ""                            # Optional
        useSsl: false                           # Optional
        database: 0                             # Optional
        maxTotal: 128                           # Optional
        maxIdle: 128                            # Optional
        minIdle: 16                             # Optional
        testOnBorrow: true                      # Optional
        testOnReturn: true                      # Optional
        testWhileIdle: true                     # Optional
        testOnCreate: true                      # Optional
        minEvictableIdleTimeMillis: 60000       # Optional
        timeBetweenEvictionRunsMillis: 30000    # Optional
        numTestsPerEvictionRun: -1              # Optional
        blockWhenExhausted: true                # Optional
        nettyThreads: 32                        # Optional
        subscriptionsPerConnection: 10          # Optional
        subscriptionConnectionPoolSize: 100     # Optional
        pageSizeInBytes: 8192                   # Optional
        readTimeout: 2000                       # Optional
```

Session tracker

```
session-tracker:
   store-type: redis
 # If the following redis properties are not specified gapwalk.redis configuration will
 be used for this cache
   redis:
     hostName: localhost
     port: 6379
     mode: standalone                       # Optional
     username:                              # Optional
     password: ""                           # Optional
     useSsl: false                          # Optional
     database: 0                            # Optional
     maxTotal: 128                          # Optional
     maxIdle: 128                           # Optional
     minIdle: 16                            # Optional
     testOnBorrow: true                     # Optional
     testOnReturn: true                     # Optional
     testWhileIdle: true                    # Optional
```

```
    testOnCreate: true                      # Optional
    minEvictableIdleTimeMillis: 60000       # Optional
    timeBetweenEvictionRunsMillis: 30000    # Optional
    numTestsPerEvictionRun: -1              # Optional
    blockWhenExhausted: true                # Optional
    nettyThreads: 32                        # Optional
    subscriptionsPerConnection: 10          # Optional
    subscriptionConnectionPoolSize: 100     # Optional
    pageSizeInBytes: 8192                   # Optional
    readTimeout: 2000                       # Optional
```

JCL checkpoint

```
jcl:
   checkpoint:
     provider: redis
 # If the following redis properties are not specified gapwalk.redis configuration will
 be used for this cache
   redis:
     hostname: localhost
     port: 6379
     mode: standalone                       # Optional
     username:                              # Optional
     password: ""                           # Optional
     useSsl: false                          # Optional
     database: 0                            # Optional
     maxTotal: 128                          # Optional
     maxIdle: 128                           # Optional
     minIdle: 16                            # Optional
     testOnBorrow: true                     # Optional
     testOnReturn: true                     # Optional
     testWhileIdle: true                    # Optional
     testOnCreate: true                     # Optional
     minEvictableIdleTimeMillis: 60000      # Optional
     timeBetweenEvictionRunsMillis: 30000   # Optional
     numTestsPerEvictionRun: -1             # Optional
     blockWhenExhausted: true               # Optional
     nettyThreads: 32                       # Optional
     subscriptionsPerConnection: 10         # Optional
     subscriptionConnectionPoolSize: 100    # Optional
     pageSizeInBytes: 8192                  # Optional
     readTimeout: 2000                      # Optional
```

Gapwalk file locks

```
filesLocks:
   enabled: true
   retryTime: 1000
   MaxRetry: 5
   provider: redis
 # If the following redis properties are not specified gapwalk.redis configuration will
 be used for this cache
   redis:
     hostName: localhost
     port: 6379
     mode: standalone                          # Optional
     username:                                 # Optional
     password: ""                              # Optional
     useSsl: false                             # Optional
     database: 0                               # Optional
     pool:
       maxTotal: 128                           # Optional
       maxIdle: 128                            # Optional
       minIdle: 16                             # Optional
       testOnBorrow: true                      # Optional
       testOnReturn: true                      # Optional
       testWhileIdle: true                     # Optional
       testOnCreate: true                      # Optional
       minEvictableIdleTimeMillis: 60000       # Optional
       timeBetweenEvictionRunsMillis: 30000    # Optional
       numTestsPerEvictionRun: -1              # Optional
       blockWhenExhausted: true                # Optional
       nettyThreads: 32                        # Optional
       subscriptionsPerConnection: 10          # Optional
       subscriptionConnectionPoolSize: 100     # Optional
       pageSizeInBytes: 8192                   # Optional
       readTimeout: 2000                       # Optional
```

Blu4iv locks

```
blu4iv.lock: redis
blu4iv.lock.timeout: 10 #(in millisecondes)
 # If the following redis properties are not specified gapwalk.redis configuration
will be used for this cache
blu4iv.lock.redis:
```

```
      hostName: localhost
      port: 6379
      mode: standalone                          # Optional
      username:                                 # Optional
      password: ""                              # Optional
      useSsl: false                             # Optional
      database: 0                               # Optional
      maxTotal: 128                             # Optional
      maxIdle: 128                              # Optional
      minIdle: 16                               # Optional
      testOnBorrow: true                        # Optional
      testOnReturn: true                        # Optional
      testWhileIdle: true                       # Optional
      testOnCreate: true                        # Optional
      minEvictableIdleTimeMillis: 60000         # Optional
      timeBetweenEvictionRunsMillis: 30000      # Optional
      numTestsPerEvictionRun: -1                # Optional
      blockWhenExhausted: true                  # Optional
      nettyThreads: 32                          # Optional
      subscriptionsPerConnection: 10            # Optional
      subscriptionConnectionPoolSize: 100       # Optional
      pageSizeInBytes: 8192                     # Optional
      readTimeout: 2000                         # Optional
```

Dataset catalog

```
datasimplifier:
   catalogImplementation: redis
   # If the following redis properties are not specified gapwalk.redis configuration
 will be used for this cache
   redis:
     hostName: localhost
     port: 6379
     mode: standalone                         # Optional
     username:                                # Optional
     password: ""                             # Optional
     useSsl: false                            # Optional
     database: 0                              # Optional
     maxTotal: 128                            # Optional
     maxIdle: 128                             # Optional
     minIdle: 16                              # Optional
     testOnBorrow: true                       # Optional
     testOnReturn: true                       # Optional
```

```
    testWhileIdle: true                     # Optional
    testOnCreate: true                      # Optional
    minEvictableIdleTimeMillis: 60000       # Optional
    timeBetweenEvictionRunsMillis: 30000    # Optional
    numTestsPerEvictionRun: -1              # Optional
    blockWhenExhausted: true                # Optional
    nettyThreads: 32                        # Optional
    subscriptionsPerConnection: 10          # Optional
    subscriptionConnectionPoolSize: 100     # Optional
    pageSizeInBytes: 8192                   # Optional
    readTimeout: 2000                       # Optional
```

Secret manager for Redis caches

The `application-main.yaml` file can reference the secret ARN for Redis caches. For information about how to integrate AWS Secrets Manager to securely retrieve Redis connection details at runtime, see the section called “AWS Blu Age Runtime secrets”.

Configure security for Gapwalk applications

The following topics describe how to secure Gapwalk applications.

It is your responsibility to provide the right configuration to ensure that the use of the AWS Blu Age framework is secure.

All security-related features are disabled by default. To enable authentication (and CSRF,XSS,CSP, and so on), set `gapwalk-application.security` to `enabled` and `gapwalk-application.security.identity` to `oauth`.

Topics

- Configure URI accessibility for Gapwalk applications
- Configure authentication for Gapwalk applications

Configure URI accessibility for Gapwalk applications

This topic describes how to configure the filtering of URIs for Gapwalk applications. This feature does not require an identity provider (IdP).

To block a list of URIs, add the following two lines to the `application-main.yml` of your modernized application, replacing *`URI-1`*, *`URI-2`*, and so on, with the URIs that you want to block.

```
gapwalk-application.security.filterURIs: enabled
gapwalk-application.security.blockedURIs: URI-1, URI-2, URI-3
```

Configure authentication for Gapwalk applications

To configure OAuth2 authentication for your Gapwalk application, you need to set up an identity provider (IdP) and integrate it with your application. This guide covers the steps for using Amazon Cognito or Keycloak as your IdP. With Amazon Cognito, you can update your application's configuration file with the Cognito user pool details. With Keycloak, you can control access to your application's APIs and resources based on the user's assigned roles.

Topics

- Configure Gapwalk OAuth2 authentication with Amazon Cognito
- Configure Gapwalk OAuth2 authentication with Keycloak

Configure Gapwalk OAuth2 authentication with Amazon Cognito

This topic describes how to configure OAuth2 authentication for Gapwalk applications using Amazon Cognito as an identity provider (IdP).

Prerequisites

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 is required for Gapwalk authentication. For more information, see the Amazon Cognito Developer Guide.

The ClientRegistration information:

client-id

The ID of the ClientRegistration. In our example it will be PlanetsDemo.

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) that contains the keys for validating the JSON web signature issued by the authorization server.

redirect URI

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

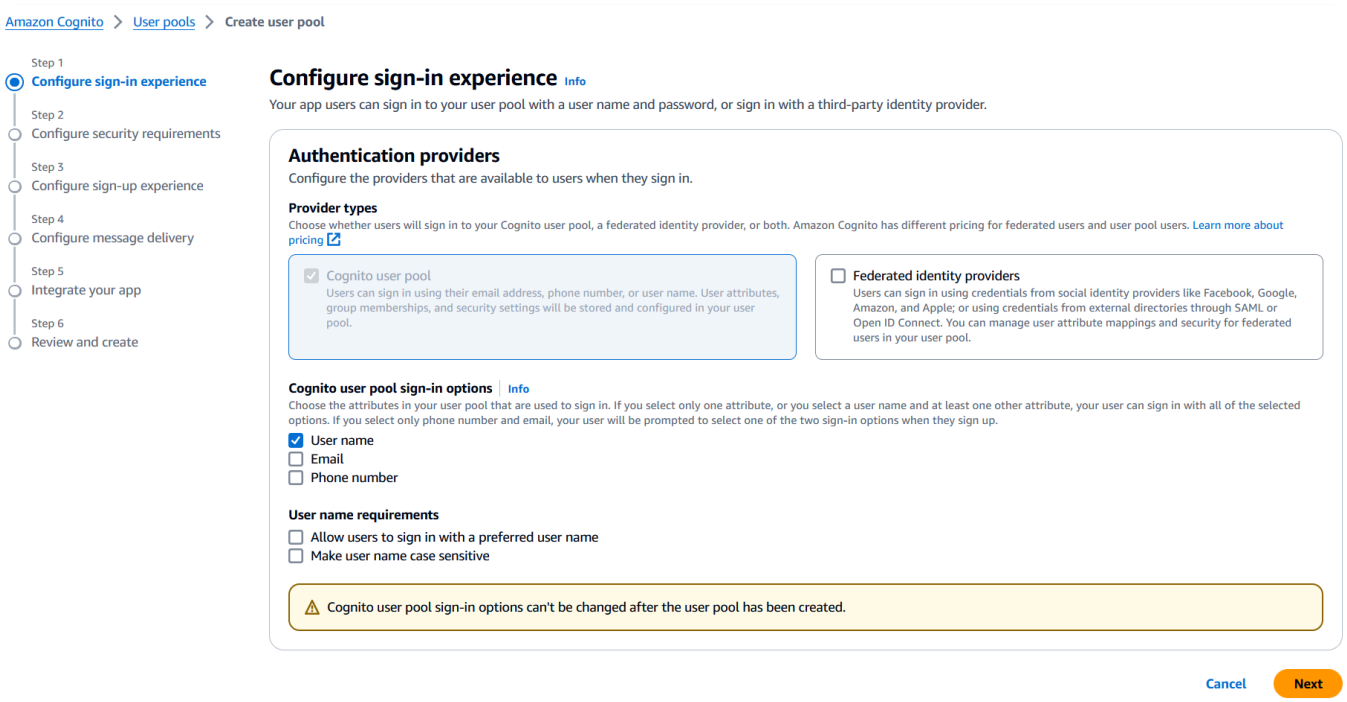
Amazon Cognito setup

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.

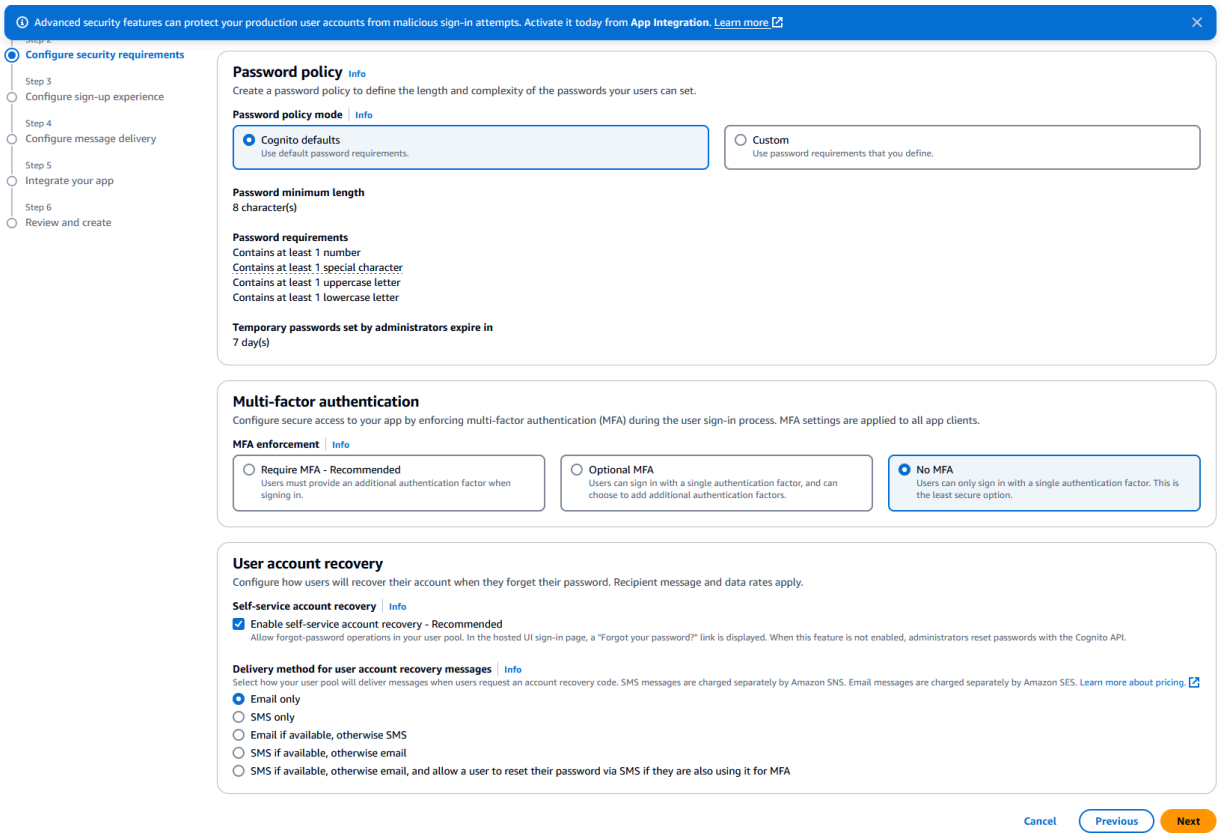
> **Note**
>
> If you are using another IdP, you can skip this step.

Create user pool

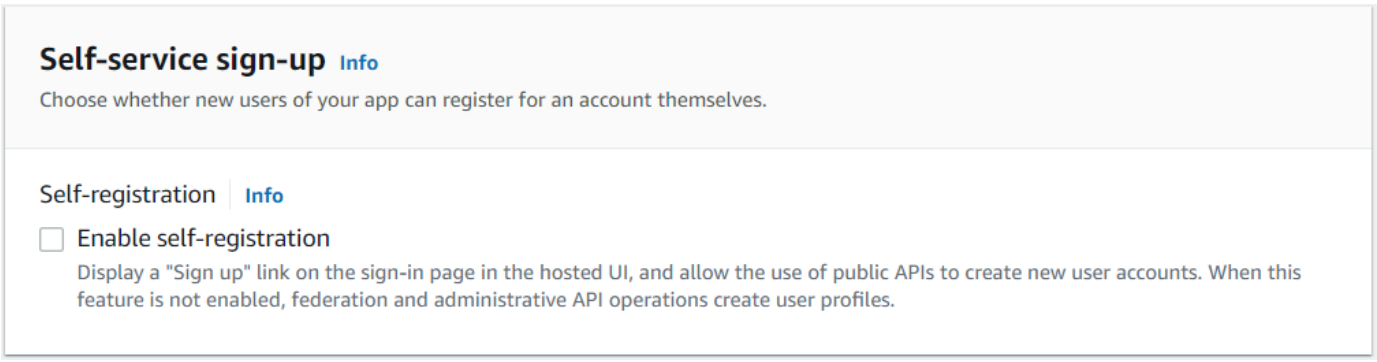
1. Go to Amazon Cognito in the AWS Management Console and authenticate using your AWS credentials.
2. Choose **User Pools**.
3. Choose **Create a user pool**.
4. In **Configure sign-in experience**, keep the **Cognito user pool** default provider type. You can choose one or multiple **Cognito user pool sign-in options**; for now, choose **User name**, then choose **Next**.



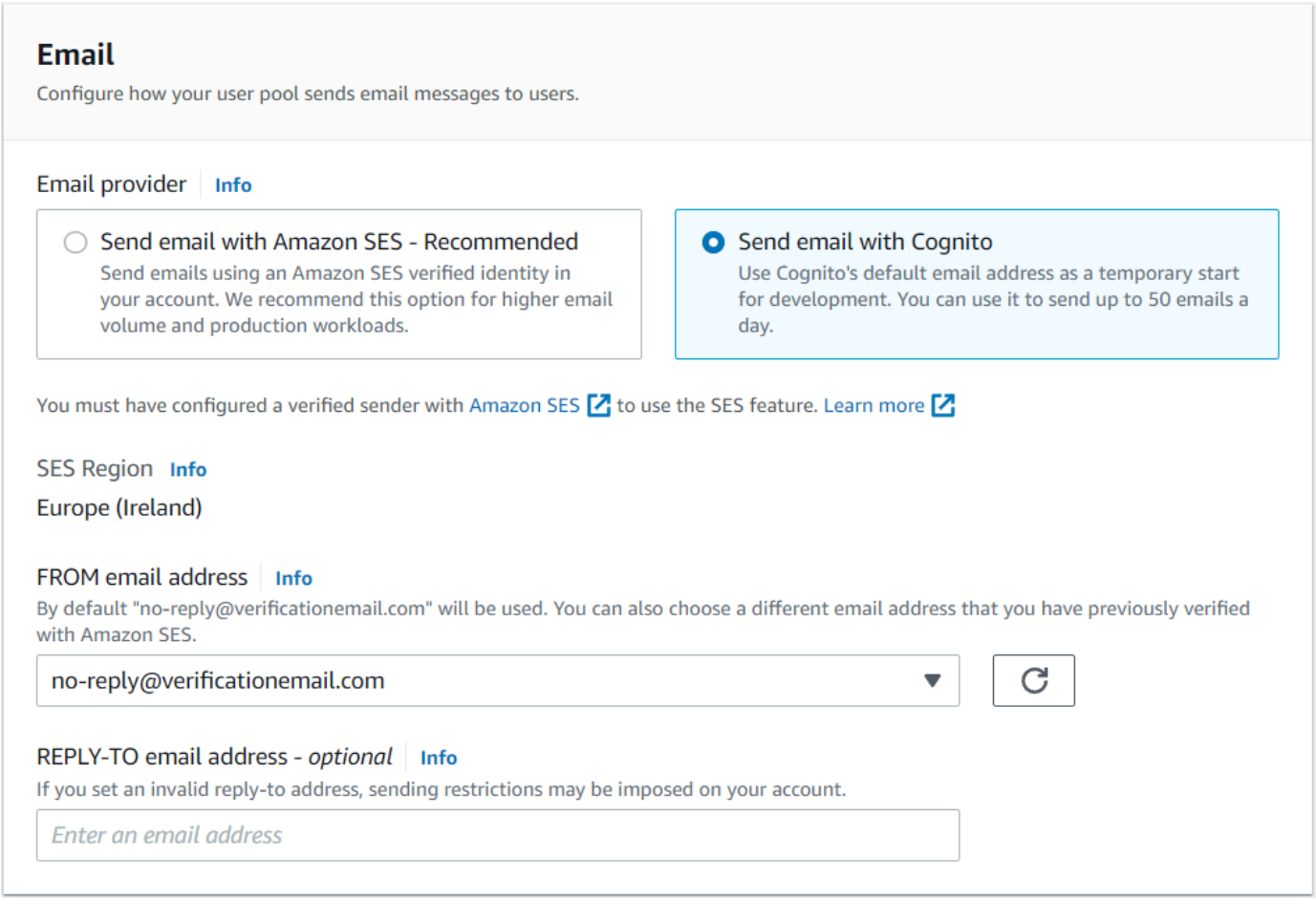
5. In **Configure security requirements**, keep the defaults and disable **Multi-factor authentication** by choosing **No MFA**, and then choose **Next**.



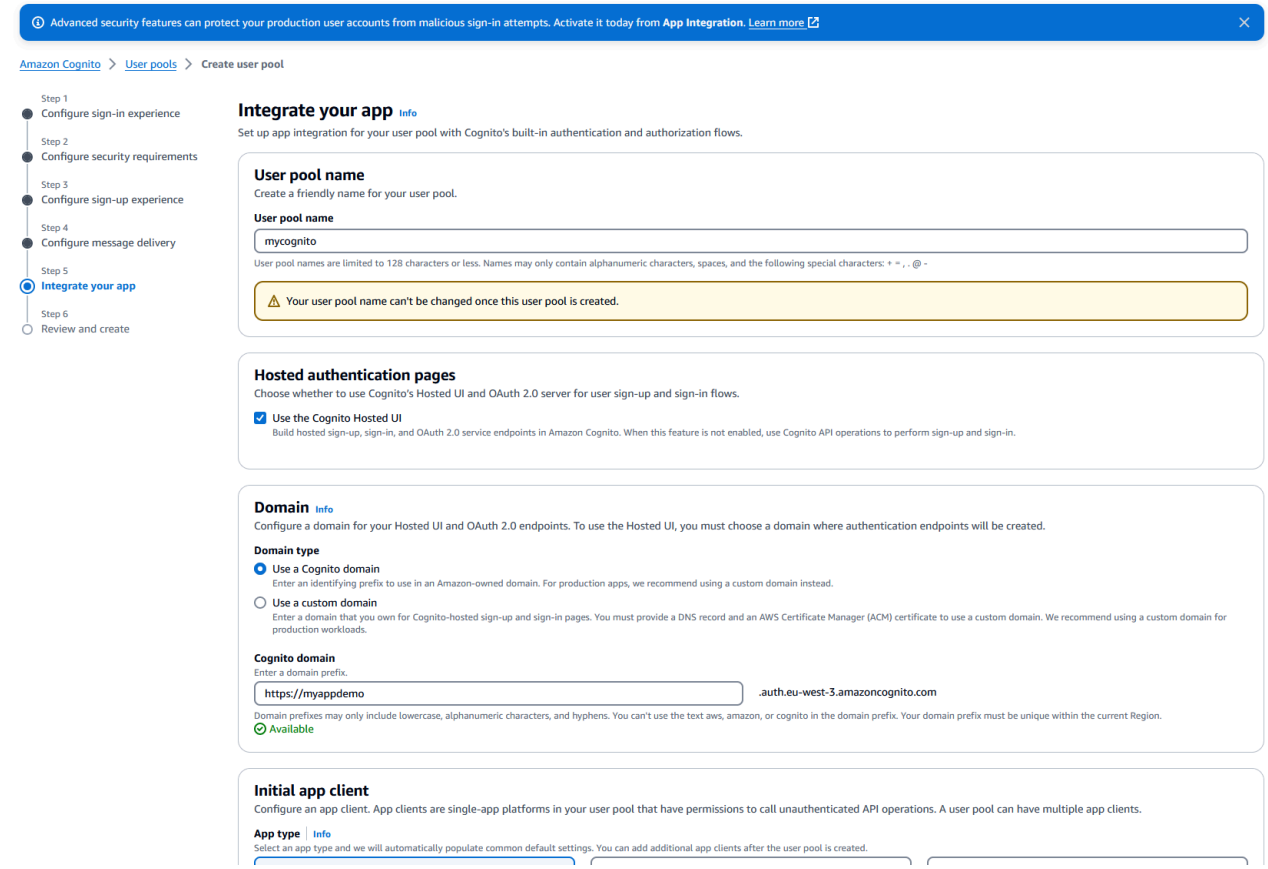
6. As a security measure, disable **Enable self-registration**, and then choose **Next**.



7. Choose **Send email with Cognito**, and then choose **Next**.



8. In **Integrate your app**, specify a name for your user pool. In **Hosted authentication pages**, choose **Use the Cognito Hosted UI**.



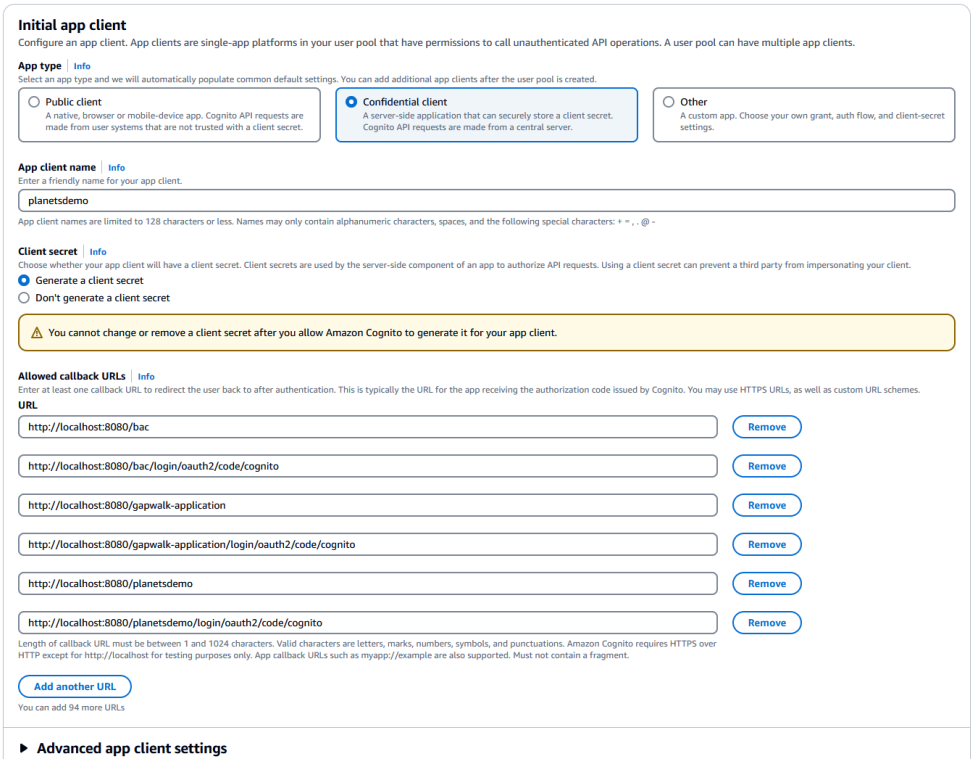
9. For simplicity, in **Domain**, choose **Use a Cognito domain** and enter a domain prefix; for example, `https://planetsdemo`. The demo app must be added as a client.

   a. In **Initial app client**, choose **Confidential client**. Enter an app client name, such as **`planetsdemo`**, and then choose **Generate a client secret**.

   b. In **Allowed callback URL** enter the URL to redirect the user to after authentication. The URL must end with `/login/oauth2/code/cognito`. For example, for our application and backend Gapwalk and BAC applications:

```
http://localhost:8080/bac
      http://localhost:8080/bac/login/oauth2/code/cognito
      http://localhost:8080/gapwalk-application
      http://localhost:8080/gapwalk-application/login/oauth2/code/cognito
      http://localhost:8080/planetsdemo
      http://localhost:8080/planetsdemo/login/oauth2/code/cognito
```

   You can edit the URL later.



c. In **Allowed sign-out URLs** enter the URL of the sign-out page that you want Amazon Cognito to redirect to when your application signs users out. For example, for backend Gapwalk and BAC applications:

```
http://localhost:8080/bac/logout
http://localhost:8080/gapwalk-application/logout
http://localhost:8080/planetsdemo/logout
```

You can edit the URL later.

d. Keep the default values in the **Advanced app client settings** and **Attribute read and write permissions** sections.

e. Choose **Next**.

10. In **Review and create**, verify your choices, and then choose **Create user pool**.

For more information, see Create user pool.

User creation

Because self-registration is disabled, create a Amazon Cognito user. Navigate to Amazon Cognito in the AWS Management Console. Choose the user pool you created, and 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**.

Role creation

In the **Groups** tab, create 3 groups (SUPER_ADMIN, ADMIN, and USER), and associate your user to one or more of these groups. These roles are later mapped to ROLE_SUPER_ADMIN, ROLE_ADMIN and ROLE_USER by the Gapwalk application to make it possible to access some restricted API REST calls.

Integrate Amazon Cognito into the Gapwalk application

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

```
gapwalk-application.security: enabled
gapwalk-application.security.identity: oauth
gapwalk-application.security.issuerUri: https://cognito-idp.<region-id>.amazonaws.com/
<pool-id>
gapwalk-application.security.domainName: <your-cognito-domain>
gapwalk-application.security.localhostWhitelistingEnabled: false

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: "<redirect-uri>"
        provider:
          cognito:
            issuer-uri: ${gapwalk-application.security.issuerUri}
```

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

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 that 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 then 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 AWS Region ID where you created your Amazon Cognito user and user pool. Example: `eu-west-3`.
6. For *`redirect-uri`* enter the URI that you specified for **Allowed callback URL**. In our example it is `http://localhost:8080/planetsdemo/login/oauth2/code/cognito`.

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

Configure Gapwalk OAuth2 authentication with Keycloak

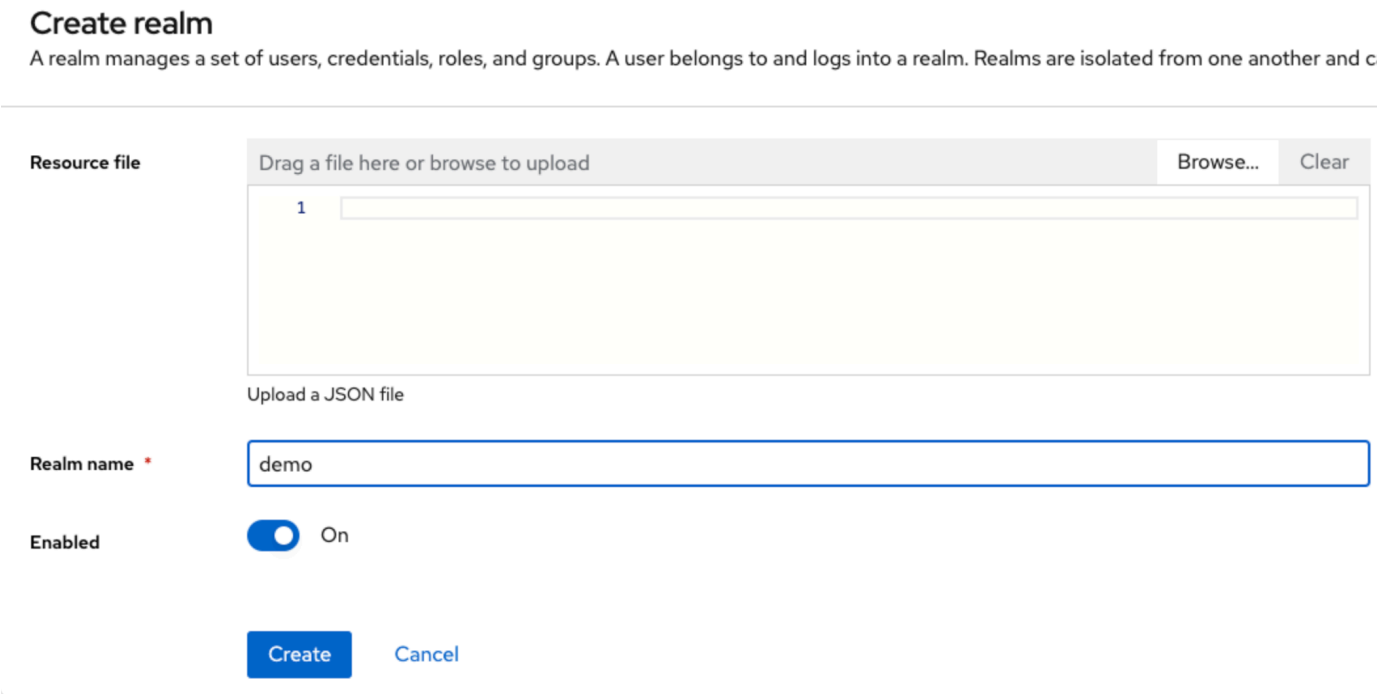
This topic describes how to configure OAuth2 authentication for Gapwalk applications using Keycloak as an identity provider (IdP). In this tutorial we use Keycloak 24.0.0.

Prerequisites

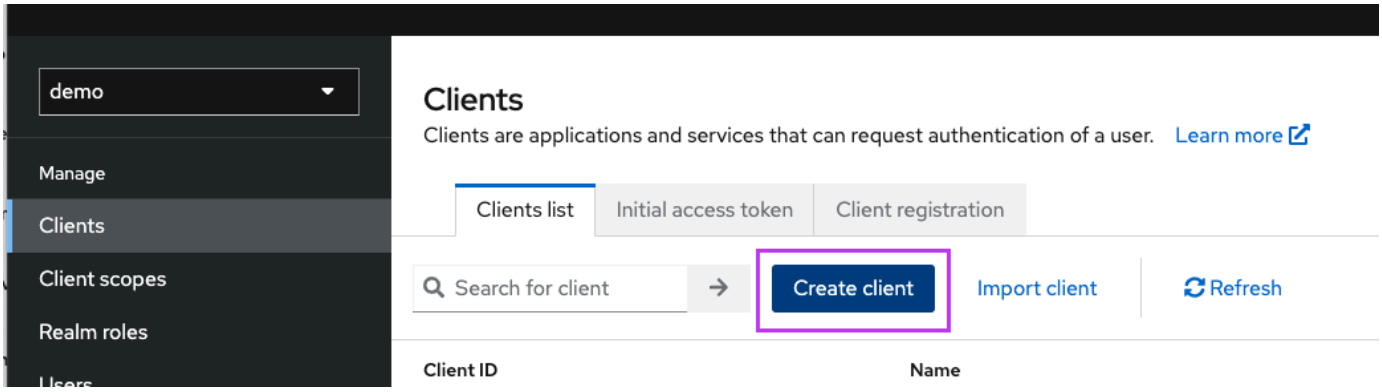
- Keycloak
- Gapwalk application

Keycloak setup

1. Go to your Keycloak dashboard in your web browser. The default credentials are admin/admin. Go to the top left navigation bar, and create a realm with the name **`demo`**, as shown in the following image.



2. Create a client with the name **`app-demo`**.



   Replace `localhost:8080` with the address of your Gapwalk application

General settings

Client ID * app-demo

Name

Description

Always display in UI Off

Access settings

Root URL http://localhost:8080

Home URL

Valid redirect URIs
- http://localhost:8080/*
- https://localhost:8080/*

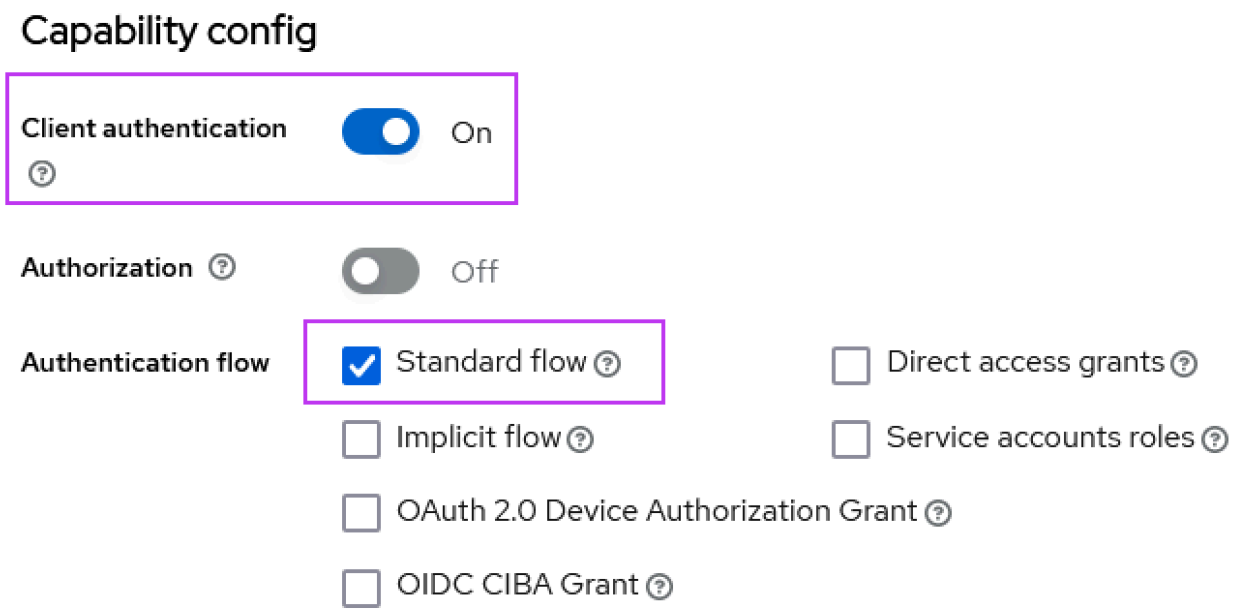
Add valid redirect URIs

Valid post logout redirect URIs
- http://localhost:8080/*
- https://localhost:8080/*

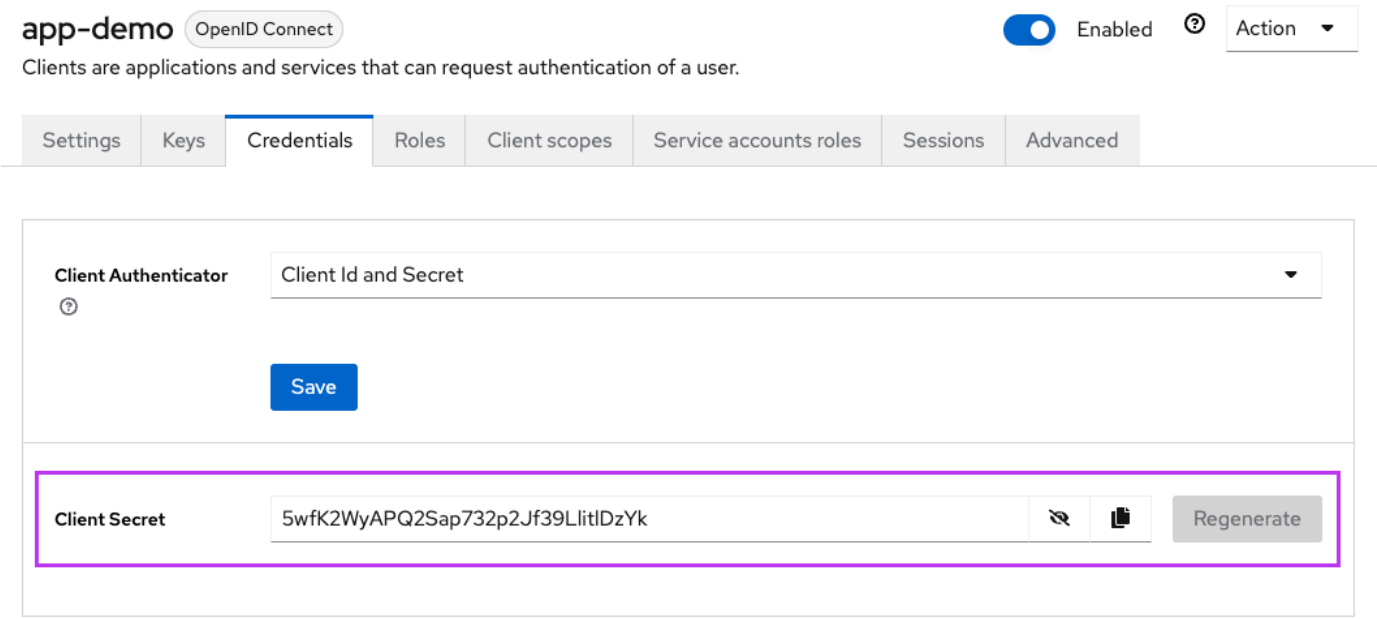
Add valid post logout redirect URIs

Web origins
- +

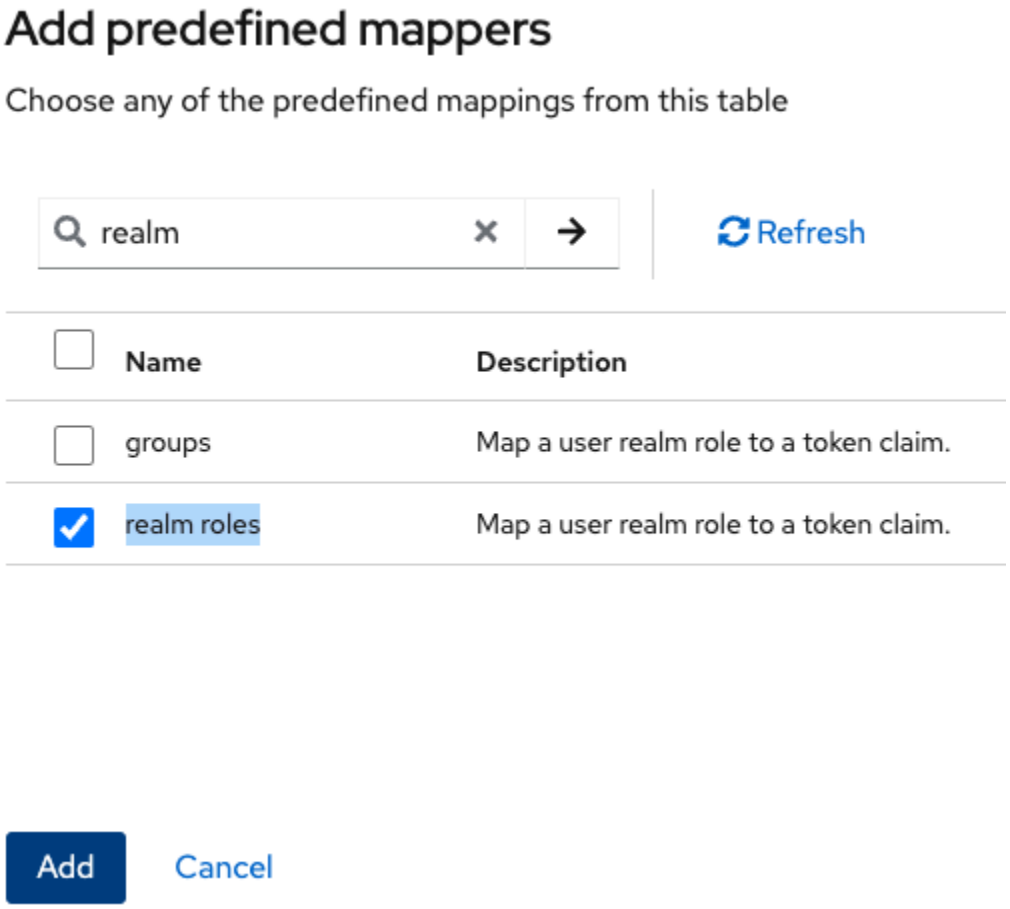
Add web origins



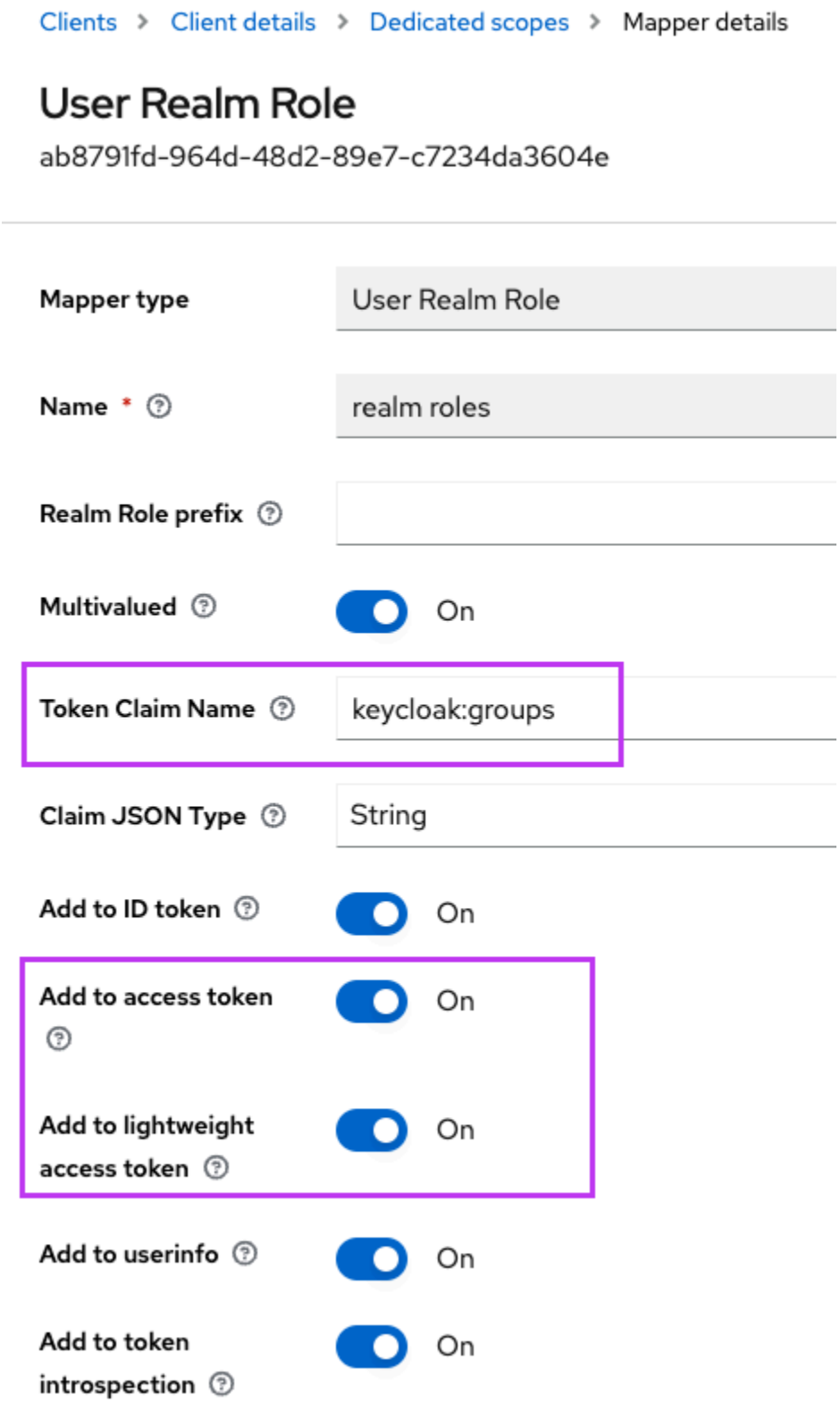
3. To get your client secret, choose **Clients**, then **app-demo**, then **Credentials**.



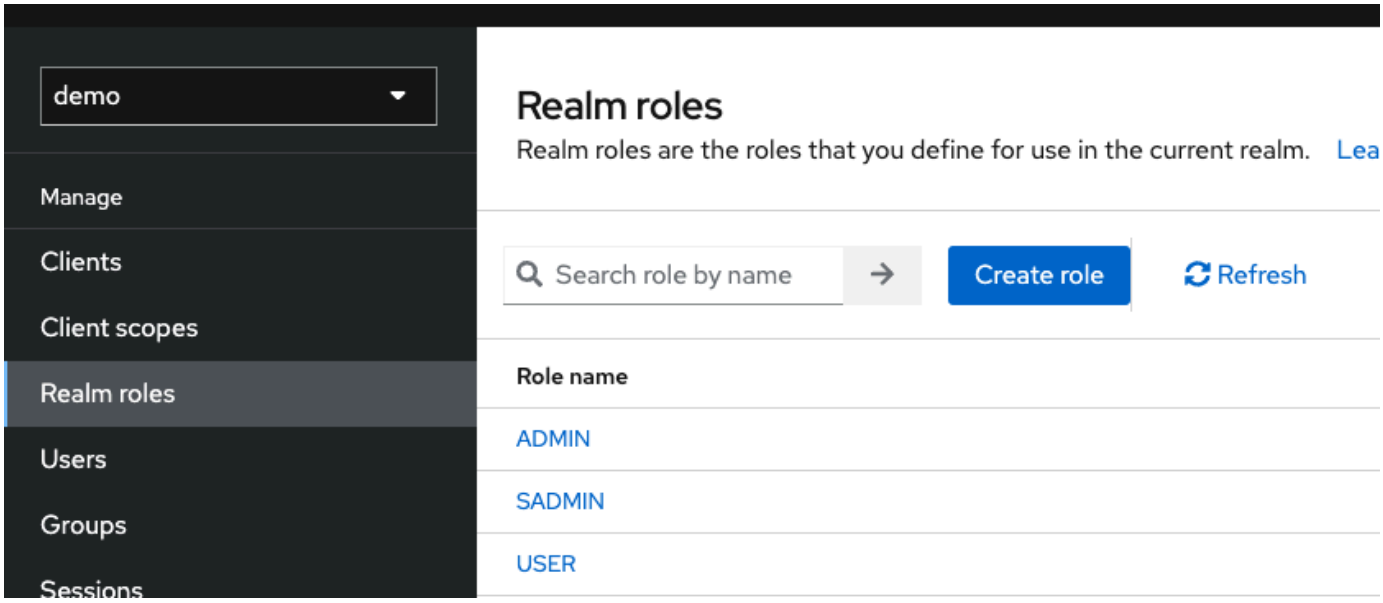
4. Choose **Clients**, then **Client scopes**, then **Add predefined mapper**. Choose **realm roles**.



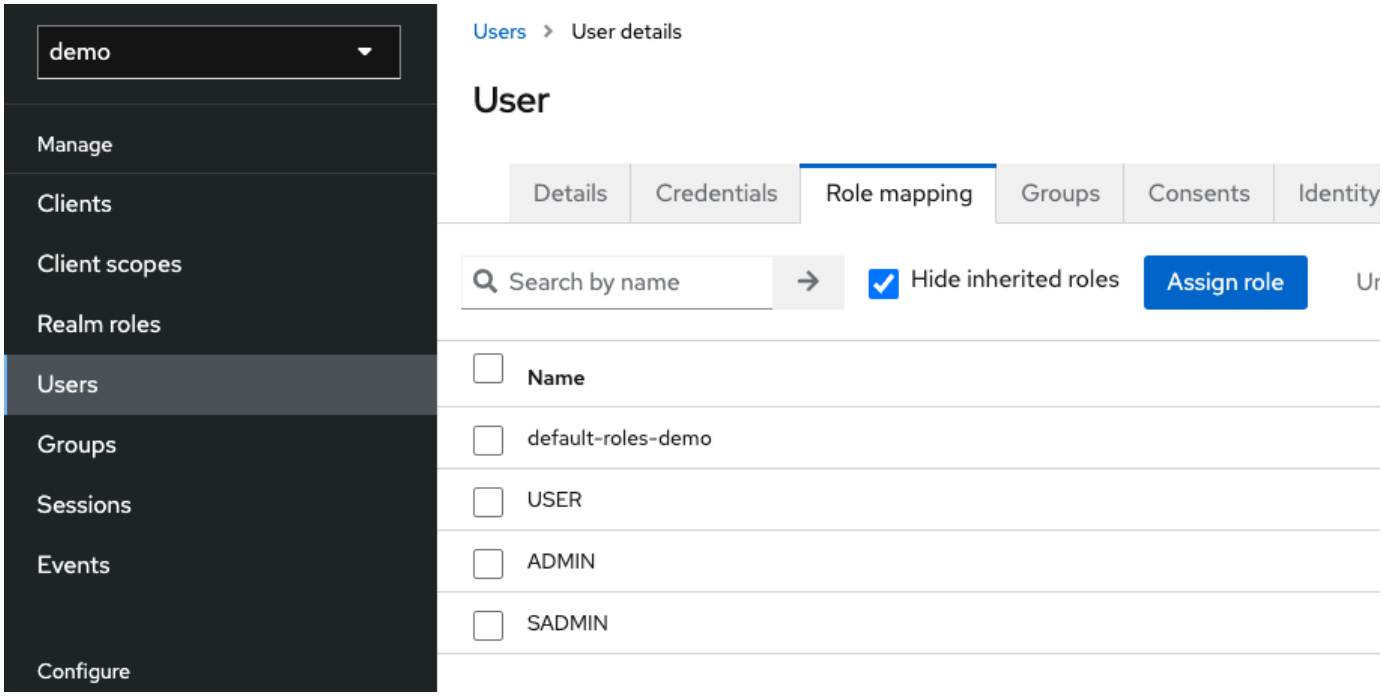
5. Edit your realm role with the configuration shown in the following image.



6. Remember the defined **Token Claim Name**. You'll need this value in the Gapwalk settings definition for the `gapwalk-application.security.claimGroupName` property.



7. Choose **Realms roles**, and create 3 roles: **`SUPER_ADMIN`**, **`ADMIN`**, and **`USER`**. These roles are later mapped to `ROLE_SUPER_ADMIN`, `ROLE_ADMIN`, and `ROLE_USER` by the Gapwalk application to be able to access some restricted API REST calls.



Integrate Keycloak into the Gapwalk application

Edit your `application-main.yml` as follows:

```
gapwalk-application.security: enabled
```

```
gapwalk-application.security.identity: oauth
gapwalk-application.security.issuerUri: http://<KEYCLOAK_SERVER_HOSTNAME>/realms/
<YOUR_REALM_NAME>
gapwalk-application.security.claimGroupName: "keycloak:groups"

gapwalk-application.security.userAttributeName: "preferred_username"
# Use "username" for cognito,
#     "preferred_username" for keycloak
#      or any other string
gapwalk-application.security.localhostWhitelistingEnabled: false

spring:
  security:
    oauth2:
      client:
        registration:
          demo:
            client-id: <YOUR_CLIENT_ID>
            client-name: Demo App
            client-secret: <YOUR_CLIENT_SECRET>
            provider: keycloak
            authorization-grant-type: authorization_code
            scope: openid
            redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"
        provider:
          keycloak:
            issuer-uri: ${gapwalk-application.security.issuerUri}
            authorization-uri: ${gapwalk-application.security.issuerUri}/protocol/
openid-connect/auth
            jwk-set-uri: ${gapwalk-application.security.issuerUri}/protocol/openid-
connect/certs
            token-uri: ${gapwalk-application.security.issuerUri}/protocol/openid-
connect/token
            user-name-attribute: ${gapwalk-application.security.userAttributeName}
      resourceserver:
        jwt:
          jwk-set-uri: ${gapwalk-application.security.issuerUri}/protocol/openid-
connect/certs
```

Replace *<KEYCLOAK_SERVER_HOSTNAME>*, *<YOUR_REALM_NAME>*, *<YOUR_CLIENT_ID>*, and *<YOUR_CLIENT_SECRET>* with your Keycloak server hostname, your realm name, your client ID, and your client secret.

AWS Blu Age Runtime 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

- Available endpoints for user when building URLs
- Endpoints for Gapwalk application in AWS Blu Age
- Blusam application console REST endpoints
- Manage JICS application console in AWS Blu Age
- Data structures for AWS Blu Age user

Available endpoints for user when building URLs

This topic lists the URLs with root paths for endpoints. 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.

Endpoints for Gapwalk application in AWS Blu Age

In this topic, learn about the endpoints for the Gapwalk web application. These use the root path `/gapwalk-application`.

Topics

- Batch jobs (modernized JCLs and alike) related endpoints
- Metrics endpoints
- Other endpoints
- Job queues related endpoints

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
- Launch a script synchronously
- Launch a script asynchronously
- Listing triggered scripts
- Retrieving job execution details
- Listing asynchronously launched scripts that can be killed
- Listing synchronously launched scripts that can be killed
- Killing a given job execution
- Listing existing checkpoints for restartability

- Restarting a job (synchronously)
- Restarting a job (asynchronously)
- Setting thread limit for asynchronous job executions

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).

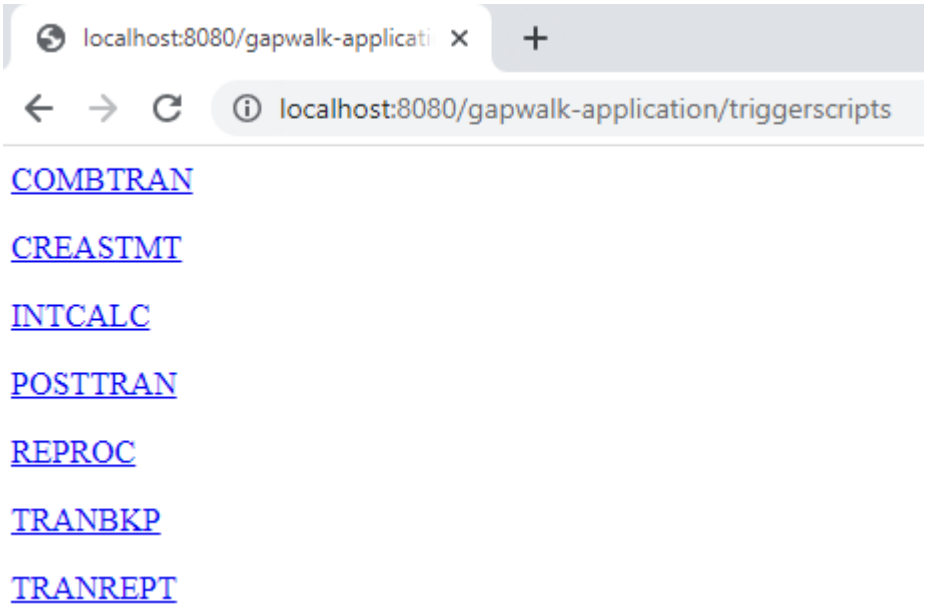
Sample response:

```
<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 launch-able script -- see sample below).

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



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):

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

> **Note**
>
> 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.

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) 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 predecessor steps above the checkpoint).

- Supported method: GET
- Path: `/restarts/{scriptId}/{jobId}`
- Arguments:
  - scriptId (optional - string): the script being restarted.
  - jobId (optional - string): the unique identifier of a job execution.
- Returns a JSON formatted list of existing restart points, that can be used to restart a job whose execution did not come to and end properly or trigger a delayed restart by bypassing previously executed steps. 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}/{scriptId}/{skipflag}`
- Arguments:
  - hashcode (integer - mandatory): restart the most recent execution of a job, using the provided hashcode as checkpoint value (see the `/restarts` endpoint above to learn how to retrieve a valid checkpoint value).
  - scriptId (optional - string): the script being restarted.
  - skipflag (optional - boolean): skip execution of selected (checkpoint) step and issue a restart from immediate successor step (if any).
- Returns: see `/script` return description above.

Restarting a job (asynchronously)

- Supported method: GET
- Path: `/triggerrestart/{hashcode}/{scriptId}/{skipflag}`
- Arguments:
  - hashcode (integer - mandatory): restart the most recent execution of a job, using the provided hashcode as checkpoint value (see the `/restarts` endpoint above to learn how to retrieve a valid checkpoint value).

- scriptId (optional - string): the script being restarted.
- skipflag (optional - boolean): skip execution of selected (checkpoint) step and issue a restart from immediate successor step (if any).
- Returns: see `/triggerscript` 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 en 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 list registered programs or services, discover health status, and manage JICS transactions.

Topics

- Listing registered programs
- Listing registered services
- Health status
- Listing available JICS transactions
- Launch a JICS transaction
- Launch a JICS transaction (alternative)
- List active sessions

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:

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

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:

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

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.
- Returns a `com.netfective.bluage.gapwalk.rt.shared.web.TransactionResultBean` JSON serialization, representing the outcome of the JICS transaction launch.

For more information about the details of the structure, see Transaction launch outcome structure.

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.

List active sessions

- supported methods: GET,POST
- path: `/activesessionlist`
- Arguments: none
- Returns a list of `com.netfective.bluage.gapwalk.application.web.sessiontracker.SessionTrackerObje` in JSON serialization, representing the list of active user sessions. When session tracking is disabled, an empty list will be returned.

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.

Operations on job queues are handled through the following dedicated endpoints. You can invoke these operations from the Gapwalk Application URL with the following root URL: `http://server:port/gapwalk-application/jobqueue`.

Topics

- List available queues
- Start or restart a job queue
- Submit a job for launch
- List all submitted jobs
- Release all jobs that are "on hold"
- Release all jobs that are "on hold" for a given job name
- Release a given job for a job number
- Submit a job on repeating schedule
- List all submitted repeating jobs
- Cancel the scheduling of a repeating job

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:

```
{"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
- Path: `/submit`
- Argument: mandatory as request body, a JSON serialization of a `com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage` object. For more information, see Submit job and schedule job input.
- Returns a JSON containing the original `SubmitJobMessage` and a log indicating if the job has been submitted or not.

List all submitted jobs

- Supported method: GET
- Path: `/list-jobs?status={status}&size={size}&page={page}&sort={sort}`
- Arguments:
  - page: Page number to retrieve (default = 1)
  - size: Size of the page (default = 50, max = 300)
  - sort: The order of the Jobs. (default = “executionId”). “executionId” is currently the only supported value
  - status: (optional) If present, it will filter on the status.
- Returns a list of all scheduled jobs, as a JSON string. For a sample response, see List of scheduled jobs response.

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 occurred. 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 granted 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.

Submit a job on repeating schedule

Schedule a job that will be executed with a repeating schedule.

- Supported method: POST
- Path: `/schedule`
- Argument: the request body must contain a JSON serialization of a `com.netfective.bluage.gapwalk.rt.jobqueue.SubmitJobMessage` object.

List all submitted repeating jobs

- Supported method: GET
- Path: `/schedule/list?status={status}&size={size}&page={page}&sort={sort}`
- Arguments:
  1. page: Page number to retrieve (default = 1)
  2. size: Size of the page (default = 50, max = 300)
  3. sort: The order of the Jobs. (default = “id”). “id” is the only supported value for now.
  4. status: (optional) If present, it will filter on the status. Possible values are the one mentioned in section 1.
  5. status: (optional) If present, it will filter on the status. Possible values are the one mentioned in section 1.
  6. Returns a list of all scheduled jobs, as a JSON string.

Cancel the scheduling of a repeating job

Removes a job that was created on a repeating schedule. The job scheduling status is set to INACTIVE.

- Supported method: GET
- Path: `/schedule/remove/{schedule_id}`
- Argument: `schedule_id`, the identifier of the scheduled job to remove.

Blusam application console REST endpoints

In this section, you can learn about the Blusam application console, which is an API designed to simplify the management of modernized VSAM datasets. Endpoints for the Blusam web application use the root path `/bac`.

Topics

- Data sets related endpoints
- Bulk data sets related endpoints
- Records
- Masks

- Other
- BAC user-management endpoints

Data sets related endpoints

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

Topics

- Create a data set
- Upload a file
- Load a data set (POST)
- Load a data set (GET)
- Load a data set from an Amazon S3 bucket
- Export a data set to an Amazon S3 bucket
- Clear a data set
- Delete a data set
- Count data set records

Create a data set

You can use this endpoint to create a data set definition.

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- 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:

```
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:


```
{
    "dataSet": {
      "name": "DATASET",
      "checked": false,
      "nbRecords": 0,
      "keyLength": -1,
      "recordSize": 10,
      "compression": false,
      "fixLength": true,
      "type": "ESDS",
      "cacheEnable": false,
      "cacheWarmup": false,
      "cacheEviction": "100ms",
      "creationDate": 1686744961234,
      "modificationDate": 1686744961234,
      "records": [],
      "primaryKey": {
        "name": "PK",
        "offset": null,
        "length": null,
        "columns": null,
        "unique": true
      },
      "alternativeKeys": [
        {
          "offset": 10,
          "length": 10,
          "name": "ALTK_0"
        }
      ],
      "readLimit": 0,
      "readEncoding": null,
      "initCharacter": null,
      "defaultCharacter": null,
      "blankCharacter": null,
      "strictZoned": null,
      "decimalSeparator": null,
      "currencySign": null,
      "pictureCurrencySign": null
    },
    "message": null,
    "result": true
```

```
}
```

Upload a file

You can use this endpoint to upload files to the server. The file is stored in a temporary folder that corresponds to each specific user. Use this endpoint every time you need to upload a file.

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- 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 (POST)

After you use `createDataSet` to create the data set definition, you can load records that are associated with the uploaded file to a specific data set.

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- 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 (GET)

- Supported methods: GET
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/rest/bluesamservice/loadDataSet`

- Arguments:

  name

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

  dataset file

  (required, string): the data set file name.
- 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
- Requires authentication and the ROLE_ADMIN role.
- 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
- Requires authentication and the ROLE_ADMIN role.
- 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.

  kmsKeyId

  (optional, string): the AWS KMS ID to be used for encryption of the exported data set to 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.

- Supported methods: POST, GET
- Requires authentication and the ROLE_ADMIN role.
- 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 definition and records.

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- 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 with a data set.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- 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 (GET)
- Export data sets (POST)
- Create multiple data sets
- List all data sets
- Direct list all data sets

- Direct list all data sets by page
- Stream data set
- Delete all data sets
- Get data set definitions from listcat file
- Get data set definitions from uploaded list cat file
- Get a data set
- Load listcat from JSON file

Export data sets (GET)

- Supported methods: GET
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/bluesamservice/exportDataSet`
- Arguments:

  datasetName

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

  datasetOutputFile

  (required, string): the path of the folder where you want to store the exported dataset on the server.

  rdw

  (required, boolean): whether you want the record descriptor word (RDW) to be part of the exported records. If the data set has fixed length records, the value of this parameter is ignored.
- Returns the status of the request and the path to the file containing the exported data set (if any). If the dataset is null in the response, that means the system was not able to locate a data set with the given name.

Export data sets (POST)

- Supported methods: POST
- Requires authentication and the ROLE_USER role.

- Path: `/api/services/rest/bluesamservice/exportDataSet`
- Arguments:

  dumpParameters

  (required, BACReadParameters): Bluesam read parameters.
- Returns the status of the exported data set.

Create multiple data sets

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- 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
- Requires authentication and the ROLE_USER role.
- 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
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/bluesamservice/directListDataSet`
- Arguments: None
- Returns: the status of the request and the list of the data sets.

Direct list all data sets by page

- Supported methods: GET
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/bluesamservice/directListDataSetByPage`
- Arguments:

  datasetName

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

  pageNumber

  (required, int): the page number.

  pageSize

  (required, int): the page size.
- Returns: the status of the request and the list of the data sets.

Stream data set

- Supported methods: GET
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/rest/bluesamservice/streamDataset`
- Arguments:

  datasetName

  (required, string): the name of the data set.
- Returns: A stream of the requested data sets.

Delete all data sets

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/rest/bluesamservice/removeAll`
- Arguments: None
- Returns: a boolean that represents the status of the request.

Get data set definitions from listcat file

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- 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
- Requires authentication and the ROLE_ADMIN role.

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

Get a data set

- Supported methods: GET
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/bluesamservice/getDataSet`
- Arguments:

  name

  (required, string): the name of the data set.
- Returns the requested data set.

Load listcat from JSON file

- Supported methods: GET
- Requires authentication and the ROLE_ADMIN role.
- 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
- Read a data set

- Delete a record
- Update a record
- Save a record
- Validate a record
- Get a record tree

Create a record

You can use this endpoint to create a new record.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- 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

You can use this endpoint to read a data set.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- 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

You can use this endpoint to delete a record from a data set.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- 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

You can use this endpoint to update a record associated with a data set.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- 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

You can use this endpoint to save a record to a data set and using a mask.

- Supported methods: POST

- Requires authentication and the ROLE_USER role.
- 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.

Validate a record

Use this endpoint to validate a record.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/crud/validateRecord`
- Arguments:

  dataset

  (required, DataSet): the data set object
- Returns the status of the request and the data set with the records.

Get a record tree

Use this endpoint to get the hierarchical tree of a record.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/crud/getRecordTree`
- Arguments:

  dataset

  (required, DataSet): the data set object

record

(required, Record): the record to fetch

- Returns the status of the request and the hierarchical tree of the requested record.

Masks

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

Topics

- Load masks
- Apply mask
- Apply mask filter

Load masks

You can use this endpoint to retrieve all the masks that are associated with a specific data set.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/crud/loadMasks`
- Path variables:
  recordSize: .../loadMasks/{recordSize}

  (optional, numeric): the record size, filter loaded masks that match this record size
- Arguments:
  dataset

  (required, DataSet): the data set object
- Returns the status of the request and the list of the masks.

Apply mask

You can use this endpoint to apply a mask to a specific data set.

- Supported methods: POST

- Requires authentication and the ROLE_USER role.
- 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

You can use this endpoint to apply a mask and a filter to a specific data set.

- Supported methods: POST
- Requires authentication and the ROLE_USER role.
- 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
- Check cache enabled
- Enable cache

- Check allocated RAM cache
- Check persistence
- Check supported data set types
- Check server health

Check warm up cache

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

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN role.
- 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.

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

Enable cache

- Supported methods: POST
- Requires authentication and the ROLE_ADMIN and ROLE_SUPER_ADMIN roles.
- Path: `/api/services/rest/bluesamservice/enableDisableCache/{enable}`
- Arguments:

enable

(required, boolean): if set to true, it will enable caching.

- Returns None

Check allocated RAM cache

You can use this endpoint to retrieve the allocated RAM cache memory.

- Supported methods: GET
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/bluesamservice/allocatedRamCache`
- Arguments: None
- Returns: the size of the memory as a string

Check persistence

- Supported methods: GET
- Requires authentication and the ROLE_USER role.
- Path: `/api/services/rest/bluesamservice/persistence`
- Arguments: None
- Returns: the persistence used as a string

Check supported data set types

- Supported methods: GET
- Path: `/api/services/rest/bluesamservice/getDataSetTypes`
- Requires authentication and the ROLE_USER role.
- 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/bluesamserver/serverIsUp`
- Arguments: None
- Returns: None. HTTP response status code 200 indicates that the server is up and running.

BAC user-management endpoints

Use the following endpoints to manage user interactions.

Topics

- Log a user in
- Verify whether at least one user exists in the system
- Record a new user
- Get user info
- List users
- Delete a user
- Log the current user out

Log a user in

- Supported method: POST
- Path: `/api/services/security/servicelogin/login`
- Arguments: None
- Returns the JSON serialization of a `com.netfective.bluage.bac.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": "some-admin",
     "password": null,
     "roles": [
       {
         "id": 0,
         "roleName": "ROLE_ADMIN"
```

```
      }
    ]
  }
```

Verify whether at least one user exists in the system

- Supported method: GET
- Path: `/api/services/security/servicelogin/hasAccount`
- Arguments: None
- Returns the boolean value `true` if at least one user other than the default super admin user has been created. Returns `false` otherwise.

Record a new user

- Supported method: POST
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/security/servicelogin/recorduser`
- Arguments: the JSON serialization of a `com.netfective.bluage.bac.entities.SignOn` object that represents the user to be added to the storage. The roles for the user must be defined, otherwise the user might not be able to use the BAC facility and endpoints.
- Returns the boolean value `true` if the user was successfully created. Returns `false` otherwise.
- Sample request JSON:

```
{
    "login": "simpleuser",
    "password": "simplepassword",
    "roles": [
      {
        "id": 2,
        "roleName": "ROLE_USER"
      }
    ]
  }
```

The following are the two valid values for `roleName`:

- `ROLE_ADMIN`: can manage Blusam resources and users.

- ROLE_USER: can manage Blusam resources but not users.

Get user info

- Supported method: GET
- Path: `/api/services/security/servicelogin/userInfo`
- Arguments: None
- Returns the username and role of the currently connected user

List users

- Supported method: GET
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/security/servicelogin/listusers`
- Arguments: None
- Returns a list of `com.netfective.bluage.bac.entities.SignOn`, serialized as JSON.

Delete a user

> **Important**
>
> This action cannot be undone. The deleted user won't be able to connect to the BAC application again.

- Supported method: POST
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/security/servicelogin/deleteuser`
- Arguments: the JSON serialization of a `com.netfective.bluage.bac.entities.SignOn` object that represents the user to be removed from the storage.
- Returns the boolean value `true` if the user was successfully removed.

Log the current user out

- Supported method: GET
- Path: `/api/services/security/servicelogout/logout`
- Arguments: None
- Returns the JSON message `{"success":true}` if the current user was successfully logged out. The related HTTP session will be invalidated.

Manage JICS application console in AWS Blu Age

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 application console web application use the root path `/jac/`.

Topics

- JICS resources management
- Other
- JAC users management endpoints

JICS resources management

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

Topics

- List JICS LISTS and GROUPS
- Retrieve JICS resources
- List JICS GROUPS
- List JICS GROUPS for a given LIST
- LIST JICS resources for a given GROUP
- LIST JICS resources for a given GROUP (alternative using a name)

- Editing the owned GROUPS of several LISTS
- Delete a LIST
- Delete a GROUP
- Delete a TRANSACTION
- Delete a PROGRAM
- Delete a FILE
- Delete a TDQUEUE
- Delete a TSMODEL
- Delete elements
- Create a LIST
- Create a GROUP
- Common RESOURCES creation considerations
- Create a TRANSACTION
- Create a PROGRAM
- Create a FILE
- Create a TDQUEUE
- Create a TSMODEL
- Create elements
- Update a LIST
- Update a GROUP
- Common RESOURCES update considerations
- Update a TRANSACTION
- Update a PROGRAM
- Update a FILE
- Update a TDQUEUE
- Update a TSMODEL
- Update elements
- Upsert elements
- Retrieve elements

- JICS CRUD operation

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 and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/listJicsListsAndGroups`
- Arguments: None
- 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
    }
  ]
```

Retrieve JICS resources

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/retrieveJicsResources`
- Arguments: A JSON payload that represents the JICS resources that you want to retrieve. This is the JSON serialization of a `com.netfective.bluage.jac.entities.request.RetrieveOperationRequest` object.
- Returns: A list of serialized JicsResource objects. The objects are returned in no particular order and are of different types, like PROGRAM, TRANSACTION, FILE, and so on.

List JICS GROUPS

- Supported method: GET
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/listJicsGroups`
- Arguments: None
- Returns a list of serialized JicsContainer objects (GROUPS) as JSON. The GROUPS are returned without their owning LIST information.

Sample response:

```
[
    {
      "jacType": "JACGroup",
      "name": "MURACHS",
```

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

List JICS GROUPS for a given LIST

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/listGroupsForList`
- Arguments: a JSON payload, representing the JICS LIST whose GROUPS you'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 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 and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/listResourcesForGroup`
- Arguments: a JSON payload, representing the JICS GROUP whose resources you'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 returned in no particular order and are of different types, like PROGRAM, TRANSACTION, FILE, and so on.

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 you'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, like PROGRAM, TRANSACTION, FILE, and so on.

Editing the owned GROUPS of several LISTS

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER

- Path: `/api/services/rest/jicsservice/editGroupsList`
- Arguments: a JSON representation of a collection of LISTS with children GROUPS;

  Sample request:

```
[
  {
        "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" belonged to the LIST named "MURACHS". With this editing, you "add" the group named "TEST" to the LIST named "MURACHS".
- Returns a boolean value. If the value is 'true', the LISTS modifications were successfully persisted to the underlying JICS storage.

Delete a LIST

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully operated on the underlying JICS storage.

Delete a GROUP

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully operated on the underlying JICS storage.

Delete a TRANSACTION

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully operated on the underlying JICS storage.

Delete a PROGRAM

- Supported method: POST
- Requires authentication and one the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully operated on the underlying JICS storage.

Delete a FILE

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully operated on the underlying JICS storage.

Delete a TDQUEUE

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully operated on the underlying JICS storage.

Delete a TSMODEL

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully operated on the underlying JICS storage.

Delete elements

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/deleteElements`
- Arguments: A JSON payload that represents the JICS elements to delete.
- Returns a boolean value where `true` indicates that the deletion was successfully operated in the underlying JICS storage.

Create a LIST

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully 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 and the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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: you try to create the transaction 'ARIT' in the 'TEST' group, but a transaction with that name already exists in that group.

```
{
```

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

You receive the following error response:

```
{
    "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(JicsServiceI
```

Create a TRANSACTION

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully created in the underlying JICS storage.

Create a PROGRAM

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully created in the underlying JICS storage.

Create a FILE

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully created in the underlying JICS storage.

Create a TDQUEUE

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully created in the underlying JICS storage.

Create a TSMODEL

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 where `true` indicates that the creation of elements was successfully operated in the underlying JICS storage.

Create elements

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/createElements`
- Arguments: a JSON payload that represents the JICS elements to create.
- Returns a boolean value. If the value is 'true', the elements were successfully created in the underlying JICS storage.

Update a LIST

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 the children into account.
- Returns a boolean value. If the value is 'true', the LIST was successfully 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 deactivating a lot of resources with a single operation, over several GROUPS.

Update a GROUP

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully 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 deactivating 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 and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully updated in the underlying JICS storage.

Update a PROGRAM

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully updated in the underlying JICS storage.

Update a FILE

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully updated in the underlying JICS storage.

Update a TDQUEUE

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully updated in the underlying JICS storage.

Update a TSMODEL

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- 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 was successfully updated in the underlying JICS storage.

Update elements

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/updateElements`
- Arguments: A JSON payload that represents the elements to update.
- Returns a boolean value where `true` indicates that the elements update was successfully operated in the underlying JICS storage.

Upsert elements

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/upsertElements`
- Arguments: A JSON payload that represents the elements to upsert.

- Returns a boolean value where `true` indicates that the elements upsert was successfully operated in the underlying JICS storage.

Retrieve elements

- Supported method: GET
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/retrieveElements`
- Arguments: None
- Returns a list of all serialized JICS resources.

JICS CRUD operation

- Supported method: POST
- Requires authentication and one of the following roles: ROLE_ADMIN, ROLE_SUPER_ADMIN, ROLE_USER
- Path: `/api/services/rest/jicsservice/jicsCrudOperation`
- Arguments: a JSON payload that represents the JICS resources you're looking for. This is the JSON serialization of a `com.netfective.bluage.jac.entities.request.JicsCrudOperationRequest` object.
- Returns a JSON payload that represents the response. This is the JSON serialization of a `com.netfective.bluage.jac.entities.request.JicsCrudOperationResponse` object.

Other

Topics

- JICS server health status

JICS server health status

- Supported method: GET
- Path: `/api/services/rest/jicsserver/serverIsUp`

- Arguments: None
- Returns: None. An HTTP STATUS 200 response indicates that the server is up and running.

JAC users management endpoints

Use the following endpoints to manage user interactions.

Topics

- Logging a user
- Testing if at least an user exists in the system
- Recording a new user
- User info
- Listing users
- Deleting a user
- Logout the current user

Logging a user

- Supported method: POST
- Path: `/api/services/security/servicelogin/login`
- Arguments: None
- 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": "some-admin",
    "password": null,
    "roles": [
      {
        "id": 0,
        "roleName": "ROLE_ADMIN"
      }
    ]
```

```
}
```

Testing if at least an user exists in the system

- Supported method: GET
- Path: `/api/services/security/servicelogin/hasAccount`
- Arguments: None
- Returns the boolean value `true` if at least one user other than the default super admin user has been created. Returns `false` otherwise.

Recording a new user

- Supported method: POST
- Requires authentication and the ROLE_ADMIN role.
- 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.
- Returns the boolean value `true` if the user was successfully created. Returns `false` otherwise.

Sample request:

```
{
    "login": "simpleuser",
    "password": "simplepassword",
    "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.

User info

- Supported method: GET
- Path: `/api/services/security/servicelogin/userInfo`
- Arguments: None
- Returns the username and roles of the currently connected user.

Listing users

- Supported method: GET
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/security/servicelogin/listusers`
- Arguments: None
- Returns a list of `com.netfective.bluage.jac.entities.SignOn`, serialized as JSON.

Deleting a user

- Supported method: POST
- Requires authentication and the ROLE_ADMIN role.
- Path: `/api/services/security/servicelogin/deleteuser`
- Arguments: the JSON serialization of a `com.netfective.bluage.jac.entities.SignOn` object that represents the user to be removed from the storage.
- Returns the boolean value `true` if the user was successfully removed.

> **Important**
>
> This action cannot be undone. The deleted user won't be able to connect to the JAC application again.

Logout the current user

- Supported method: GET
- Path: `/api/services/security/servicelogout/logout`

- Arguments: None
- Returns the JSON message `{"success":true}` if the current user was successfully logged out. The related HTTP session will be invalidated.

Data structures for AWS Blu Age user

You can learn about various data structures for AWS Blu Age engine in the following section.

Topics

- Job execution details message structure
- Transaction launch outcome structure
- Transaction launch record outcome structure
- Possible status of a job on a queue
- Submit job and schedule job input
- List of scheduled jobs response
- List of repeating jobs response

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:

```
{
    "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 as 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 and schedule job input

The submit job and schedule 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 for submit job:

```
{
    "messageQueueName":null,
    "scheduleDate":null,
    "scheduleTime":null,
    "programName":"PTA0044",
    "programParams":
     {"wmind":"B"},
    "localDataAreaValue":"",
    "userName":"USER1",
    "jobName":"PTA0044",
    "jobNumber":9,
    "jobPriority":5,
    "executionDate":"20181231",
    "jobQueue":"queue1",
    "jobOnHold":false
}
```

Sample input for schedule job:

```
{
     "scheduleCron": "*/2 * * * * ?",
     "programName":"LOGPGM",
     "programParams": {
         "cl_sbmjob_param_json": "[\"./output/schedule-job-log.txt\", \"Every 2
 seconds!\"]"
     },
     "localDataAreaValue":"",
     "userName":"PVO",
     "jobName":"LOGGERJOB",
     "jobPriority":5,
     "jobQueue":"queue1",
     "scheduleMisfirePolicy": 4,
     "startTime": "2003/05/04 07:00:00.000 GMT-06:00",
     "endTime": "2003/05/04 07:00:07.000 GMT-06:00"
 }
```

jobNumber

if the job number 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

Will be passed to the program as a map.

scheduleMisfirePolicy

Defines the strategy used when a trigger is misfired. The following are the possible values:

1. Release the first misfire and discard the other misfires.
2. Submit a job on hold for the first misfire and discard the other misfires.
3. Discard the misfire.
4. Release all misfires. The job queue will run all jobs.

List of scheduled jobs response

This is the structure of the list-jobs job queue endpoint. 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.


```
[
  {
    "jobName": "PTA0044",
    "userName": "USER1",
    "jobNumber": 9,
    "jobPriority": 5,
    "status": "HOLD",
    "jobDelay": 0,
    "startDate": null,
    "endDate": null,
    "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,
      "scheduleCron": null,
      "save": false,
      "scheduleMisfirePolicy": 4,
      "omitdates": null
    },
    "executionId": 1,
    "jobScheduledId": 0,
    "jobScheduledAt": null
  },
  {
    "jobName": "PTA0044",
    "userName": "USER1",
    "jobNumber": 9,
    "jobPriority": 5,
    "status": "COMPLETED",
    "jobDelay": 0,
    "startDate": "2022-10-13T22:48:34.025+00:00",
    "endDate": "2022-10-13T22:52:54.475+00:00",
    "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,
      "scheduleCron": "*/20 * * * * ?",
      "save": false,
      "scheduleMisfirePolicy": 4,
      "omitdates": null
    },
    "executionId": 2,
    "jobScheduledId": 0,
    "jobScheduledAt": null
  }
```

```
]
```

List of repeating jobs response

This is the structure of the /schedule/list job queue endpoint.

```
[
  {
    "id": 1,
    "status": "ACTIVE",
    "jobNumber": 1,
    "userName": "PVO",
    "msg": {
      "messageQueueName": null,
      "scheduleDate": null,
      "scheduleTime": null,
      "startTime": "2024/03/07 21:12:00.000 UTC",
      "endTime": "2024/03/07 21:13:59.000 UTC",
      "programName": "LOGPGM",
      "programParams": {"cl_sbmjob_param_json": "[\"./output/schedule-job-log.txt\",
 \"Every 20 seconds!\"]"},
      "localDataAreaValue": "",
      "userName": "PVO",
      "jobName": "LOGGERJOB",
      "jobNumber": 1,
      "jobScheduleId": 1,
      "jobPriority": 5,
      "executionDate": null,
      "jobQueue": "queue1",
      "jobOnHold": false,
      "scheduleCron": "*/20 * * * * ?",
      "save": false,
      "scheduleMisfirePolicy": 4,
      "omitdates": null
    },
    "lastUpdatedAt": "2024-03-07T21:11:13.282+00:00",
    "lastUpdatedBy": ""
  }
]
```

Set up AWS Blu Age Runtime (non-managed)

This section explains the steps to set up AWS Blu Age Runtime (non-managed) on your AWS infrastructure. Before you set up your AWS Blu Age Runtime (non-managed) for applications, understand the prerequisites, Regions and buckets, and the CloudWatch alarm setup to configure and manager your runtime environment.

Topics

- AWS Blu Age Runtime prerequisites
- Onboarding AWS Blu Age Runtime
- Infrastructure setup requirements for AWS Blu Age Runtime (non-managed)
- AWS Blu Age Runtime artifacts
- Deploy AWS Blu Age Runtime on Amazon EC2
- Deploy AWS Blu Age Runtime on containers on Amazon ECS and Amazon EKS
- Test the PlanetsDemo application

AWS Blu Age Runtime prerequisites

AWS Blu Age Runtime (non-managed) is available in several the section called "AWS Blu Age release notes" release versions. If you have ongoing modernization projects, you might need incremental versions of the runtime for implementation and testing purposes. To define your needs, contact your AWS Blu Age delivery manager.

Before you begin the AWS Blu Age Runtime (non-managed) onboarding process, do the following:

- Make sure that you have an AWS account.
- Make sure that you have a modernized application refactored with AWS Blu Age.
- Choose an AWS Region and one of the compute options that are supported for AWS Blu Age Runtime (non-managed).
- Choose the AWS Blu Age Runtime version that you want to use.
- Review the section called "Infrastructure setup requirements" and validate the additional components required to run the AWS Blu Age Runtime (non-managed).

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

Onboarding AWS Blu Age Runtime

To get started, create an AWS Support case to request onboarding to access AWS Blu Age Runtime. Include in your request your AWS account ID, the AWS Region that you want to use, and a compute choice, and AWS Blu Age Runtime version. If you aren't sure which version you need, contact your AWS Blu Age delivery manager. If you already have code source of application generated by AWS Mainframe Modernization Refactoring tools, note the `gapwalk.version` tag value into `pom.xml` file within your modernized code base.

> **Note**
>
> The AWS Blu Age Runtime is available in two main varieties: Alpha pre-releases and Releases. To determine which release to use, see the section called "AWS Blu Age versioning", or contact your AWS Blu Age delivery manager.

Regions and buckets for AWS Blu Age Runtime (non-managed)

We store the AWS Blu Age Runtime (non-managed) artifacts in different Amazon S3 buckets by Region and by compute choice. To access the bucket for your AWS Region for AWS Blu Age Runtime (non-managed), use the name listed in the following table.

AWS Region	Release bucket	Alpha pre-release bucket
US East (Ohio)	aws-bluage-runtime-artifacts-055777665268-us-east-2	aws-bluage-runtime-artifacts-dev-055777665268-us-east-2
US East (N. Virginia)	aws-bluage-runtime-artifacts-139023371234-us-east-1	aws-bluage-runtime-artifacts-dev-139023371234-us-east-1

AWS Region	Release bucket	Alpha pre-release bucket
US West (N. California)	aws-bluage-runtime-artifacts-788454048782-us-west-1	aws-bluage-runtime-artifacts-dev-788454048782-us-west-1
US West (Oregon)	aws-bluage-runtime-artifacts-836771190483-us-west-2	aws-bluage-runtime-artifacts-dev-836771190483-us-west-2
Canada (Central)	aws-bluage-runtime-artifacts-637423580979-ca-central-1	aws-bluage-runtime-artifacts-dev-637423580979-ca-central-1
Europe (Ireland)	aws-bluage-runtime-artifacts-925278190477-eu-west-1	aws-bluage-runtime-artifacts-dev-925278190477-eu-west-1
Europe (London)	aws-bluage-runtime-artifacts-767397831990-eu-west-1	aws-bluage-runtime-artifacts-dev-767397831990-eu-west-1
Europe (Paris)	aws-bluage-runtime-artifacts-673009995881-eu-west-3	aws-bluage-runtime-artifacts-dev-673009995881-eu-west-3
Europe (Frankfurt)	aws-bluage-runtime-artifacts-485196800481-eu-central-1	aws-bluage-runtime-artifacts-dev-485196800481-eu-central-1
Europe (Stockholm)	aws-bluage-runtime-artifacts-654654484534-eu-north-1	aws-bluage-runtime-artifacts-dev-654654484534-eu-north-1
Europe (Milan)	aws-bluage-runtime-artifacts-654654328338-eu-south-1	aws-bluage-runtime-artifacts-dev-654654328338-eu-south-1

AWS Region	Release bucket	Alpha pre-release bucket
Europe (Spain)	aws-bluage-runtime-artifacts-905417994954-eu-south-2	aws-bluage-runtime-artifacts-dev-905417994954-eu-south-2
South America (São Paulo)	aws-bluage-runtime-artifacts-737536804457-sa-east-1	aws-bluage-runtime-artifacts-dev-737536804457-sa-east-1
Asia Pacific (Tokyo)	aws-bluage-runtime-artifacts-445578176276-ap-northeast-1	aws-bluage-runtime-artifacts-dev-445578176276-ap-northeast-1
Asia Pacific (Seoul)	aws-bluage-runtime-artifacts-381492221498-ap-northeast-2	aws-bluage-runtime-artifacts-dev-381492221498-ap-northeast-2
Asia Pacific (Osaka)	aws-bluage-runtime-artifacts-905418229615-ap-northeast-3	aws-bluage-runtime-artifacts-dev-905418229615-ap-northeast-3
Asia Pacific (Singapore)	aws-bluage-runtime-artifacts-767397774613-ap-southeast-1	aws-bluage-runtime-artifacts-dev-767397774613-ap-southeast-1
Asia Pacific (Sydney)	aws-bluage-runtime-artifacts-726160321909-ap-southeast-2	aws-bluage-runtime-artifacts-dev-726160321909-ap-southeast-2
Asia Pacific (Mumbai)	aws-bluage-runtime-artifacts-905418353577-ap-south-1	aws-bluage-runtime-artifacts-dev-905418353577-ap-south-1
Africa (Cape Town)	aws-bluage-runtime-artifacts-992382777663-af-south-1	aws-bluage-runtime-artifacts-dev-992382777663-af-south-1

AWS Region	Release bucket	Alpha pre-release bucket
Israel (Tel Aviv)	aws-bluage-runtime-artifacts-471112516508-il-central-1	aws-bluage-runtime-artifacts-dev-471112516508-il-central-1

Using the AWS CLI to list the contents of the bucket

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

```
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 (non-managed) runtime, such as the following for a release bucket:

```
PRE 3.10.0/
PRE 4.0.0/
```

Or the following for a build bucket:

```
PRE 4.1.0-alpha.8/
PRE 4.1.0-alpha.9/
```

We recommend that you use the latest version available. If that isn't possible, then use the runtime version that was validated during the application refactoring phase. To list the available frameworks for a specific version, run the following command:

```
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 following are two examples.

For a release bucket:

```
aws s3 ls s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/4.0.0/
Framework/
```

The command returns a list of frameworks, such as:

```
2024-04-08 16:11:19  152040176 aws-bluage-runtime-4.0.0.tar.gz
2024-04-08 16:11:50         45 aws-bluage-runtime-4.0.0.tar.gz.checksumSHA256
2024-04-08 16:11:52  176518889 aws-bluage-webapps-4.0.0.tar.gz
2024-04-08 16:12:28         45 aws-bluage-webapps-4.0.0.tar.gz.checksumSHA256
```

For a build bucket:

```
aws s3 ls s3://aws-bluage-runtime-artifacts-dev-139023371234-us-
east-1/4.1.0-alpha.9/Framework/
```

The command returns a list of frameworks, such as:

```
2024-04-09 20:23:34  152304534 aws-bluage-runtime-4.1.0-alpha.9.tar.gz
2024-04-09 20:24:05         45 aws-bluage-runtime-4.1.0-alpha.9.tar.gz.checksumSHA256
2024-04-09 20:24:07  176262381 aws-bluage-webapps-4.1.0-alpha.9.tar.gz
2024-04-09 20:24:42         45 aws-bluage-webapps-4.1.0-alpha.9.tar.gz.checksumSHA256
```

Download the framework

You can download the framework for example to upgrade the AWS Blu Age 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-1/4.0.0/Framework/ . --recursive`

This command produces the following output:

```
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/4.0.0/
Framework/aws-bluage-runtime-4.0.0.tar.gz.checksumSHA256 to ./aws-bluage-
runtime-4.0.0.tar.gz.checksumSHA256
```

```
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/4.0.0/
Framework/aws-bluage-webapps-4.0.0.tar.gz.checksumSHA256 to ./aws-bluage-
webapps-4.0.0.tar.gz.checksumSHA256
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/4.0.0/Framework/aws-
bluage-webapps-4.0.0.tar.gz to ./aws-bluage-webapps-4.0.0.tar.gz
download: s3://aws-bluage-runtime-artifacts-139023371234-us-east-1/4.0.0/Framework/aws-
bluage-runtime-4.0.0.tar.gz to ./aws-bluage-runtime-4.0.0.tar.gz
```

You can list the framework files as follows:

```
ls -l
```

This command produces the following output:

```
total 230928
-rw-rw-r-- 1 cloudshell-user cloudshell-user 152040176 Apr  8 16:11 aws-bluage-
runtime-4.0.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user        45 Apr  8 16:11 aws-bluage-
runtime-4.0.0.tar.gz.checksumSHA256
-rw-rw-r-- 1 cloudshell-user cloudshell-user 176518889 Apr  8 16:11 aws-bluage-
webapps-4.0.0.tar.gz
-rw-rw-r-- 1 cloudshell-user cloudshell-user        45 Apr  8 16:12 aws-bluage-
webapps-4.0.0.tar.gz.checksumSHA256
```

Note

Access to artifacts may be temporarily interrupted, and versions may be removed for security reasons. We strongly recommend that you store the artifacts you use in your own account. The local version should be used for reference in your internal architectures.

Infrastructure setup requirements for AWS Blu Age Runtime (non-managed)

This topic describes the minimum infrastructure configuration required to run AWS Blu Age Runtime (non-managed). The following procedures describe how to set up AWS Blu Age Runtime (non-managed) on your compute of choice to deploy a modernized application on the AWS Blu Age Runtime. The resources that you create must be in an Amazon VPC that has a subnet that is dedicated to your application domain.

Topics

- Infrastructure requirements
- Amazon EC2 instance types for AWS Blu Age Runtime (on Amazon EC2)
- Running AWS Blu Age Runtime on Amazon EC2
- Running AWS Blu Age Runtime on Amazon ECS on Amazon EC2
- Running AWS Blu Age Runtime on Amazon EKS on Amazon EC2
- Running AWS Blu Age Runtime on Amazon ECS managed by AWS Fargate

Infrastructure requirements

Create a security group

If you plan to work on Amazon EC2 instances on Amazon EKS, skip this procedure because the Amazon EKS cluster creation process creates a security group on your behalf. Use that security group in the following procedures instead of creating a new one.

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, under **Security**, choose **Security groups**.
3. In the central pane, choose **Create security group**.
4. In the **Security group name** field, enter **`M2BluagePrivateLink-SG`**.
5. In the **Inbound rules** section, choose **Add rule**.
6. For **Type**, choose HTTPS.
7. For **Source** enter your VPC CIDR.
8. In the **Outbound rules** section, choose **Add rule**.
9. For **Type**, choose HTTPS.
10. For **Destination**, enter **`0.0.0.0/0`**.
11. Choose **Create security group**.

Create an Amazon VPC endpoint

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, under **Virtual private cloud**, choose **Endpoints**.
3. In the central pane, choose **Create endpoint**.

4. In the **Services** section, enter **SQS** in the search field, and then select the Amazon SQS service that corresponds to your Region.
5. In the **VPC** section, select the Amazon VPC that you created in the previous step.
6. In the **Subnets** section, select the subnet that you created for your application domain.
7. In the **Security groups** section, select the security group from the previous procedure.
8. Choose **Create endpoint**.

Create an IAM policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, under **Access management**, choose **Policies**.
3. In the central pane, choose **Create policy**.
4. In the **Policy editor** section, choose **JSON**.
5. Replace all of the JSON that you see in the editor with the following JSON.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sqs:GetQueueUrl",
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> If you need further details to customize your policy, contact your AWS Blu Age delivery manager or account manager.

6. Choose **Next**.

7. Enter a name for the policy, then choose **Create policy**.

Create an IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, under **Access management**, choose **Roles**.
3. In the central pane, choose **Create role**.
4. In the **Use case** section, depending on your compute choice, choose one of the following:
   - **EC2** (for Amazon EC2 and Amazon EKS on Amazon EC2)
   - **Elastic Container Service** and then **EC2 Role for Elastic Container Service** (for Amazon ECS on Amazon EC2)
   - **Elastic Container Service** and then **Elastic Container Service Task** (for Amazon ECS managed by Fargate)
5. Choose **Next**.
6. In the search box, enter the name of the policy that you created earlier.
7. Select the checkbox to the left of your policy.

   > **Note**
   >
   > If you can't add a policy, finish creating the role and then update the role to add the policy.

8. Choose **Next**.
9. Enter a name for the role, then choose **Create role**.

Amazon EC2 instance types for AWS Blu Age Runtime (on Amazon EC2)

The following is a list of the Amazon EC2 instance types that you can use for AWS Blu Age Runtime (on Amazon EC2) when creating Amazon EC2 instances or when defining Amazon EKS worker nodes.

Check that the instance of your interest is available in the desired region you plan to deploy.

```
t3.small
t3.medium
```

```
t3.large
t3.xlarge
t3.2xlarge
t2.small
t2.medium
t2.large
t2.xlarge
t2.2xlarge
r7a.medium
r7a.large
r7a.xlarge
r7a.2xlarge
r7a.4xlarge
r7a.8xlarge
r7a.12xlarge
r7a.16xlarge
r7a.24xlarge
r7a.32xlarge
r7a.48xlarge
r7a.metal-48xl
r7i.large
r7i.xlarge
r7i.2xlarge
r7i.4xlarge
r7i.8xlarge
r7i.12xlarge
r7i.16xlarge
r7i.24xlarge
r7i.48xlarge
r7i.metal-24xl
r7i.metal-48xl
r6i.xlarge
r6i.large
r6i.4xlarge
r6i.2xlarge
r5b.xlarge
r5b.large
r5b.2xlarge
r3.xlarge
m6i.xlarge
m6i.large
m6i.8xlarge
m6i.4xlarge
m6i.2xlarge
```

```
m6i.16xlarge
m5zn.xlarge
m5zn.large
m5zn.3xlarge
m5zn.2xlarge
m5.xlarge
m5.large
m5.8xlarge
m5.4xlarge
m5.2xlarge
m5.16xlarge
m5.12xlarge
c6i.xlarge
c6i.large
c6i.8xlarge
c6i.4xlarge
c6i.2xlarge
c6i.16xlarge
c5.xlarge
c5.large
c5.9xlarge
c5.4xlarge
c5.2xlarge
c5.18xlarge
c5.12xlarge
```

Running AWS Blu Age Runtime on Amazon EC2

To create an Amazon EC2 instance, use the following steps.

Create an Amazon EC2 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Launch instance**.
3. For **Instance type**, choose one of the types listed in the section called "Amazon EC2 instance types for AWS Blu Age Runtime (on Amazon EC2)".
4. In the **Key pair** section, either choose an existing key pair or create a new one.
5. In the **Network settings** section, choose **Select existing security group**.
6. For **Common security groups**, choose **M2BluagePrivateLink-SG**.
7. Expand the **Advanced details** section.

8. For **IAM instance profile**, choose the IAM role that you created earlier.
9. Choose **Launch instance**.

Install the application on the Amazon EC2 instance

1. When the state of the Amazon EC2 instance changes to **Running**, connect to the instance.
2. Install the following software components on the instance:

   - Java Runtime Environment (JRE) 17.
   - Apache Tomcat 10.
   - AWS Blu Age Runtime (on Amazon EC2). Install the AWS Blu Age runtime at the root of Apache Tomcat installation folder (some files will be added while others will be overwritten).

   To install the additional webapps delivered alongside the AWS Blu Age Runtime archive, set up a secondary instance of the Apache Tomcat server, and decompress the webapps archive at that location. For detailed instructions, see the section called “AWS Blu Age Runtime artifacts”.

Running AWS Blu Age Runtime on Amazon ECS on Amazon EC2

1. Create an Amazon ECS cluster, with **Amazon EC2 instances** as an underlying infrastructure. See Getting started with Windows on Amazon EC2 in the Amazon Elastic Container Service Developer Guide.
2. Specify the IAM role that you created in the previous steps.
3. Choose one of the instance types listed in the section called “Amazon EC2 instance types for AWS Blu Age Runtime (on Amazon EC2)”.
4. In **Network settings for Amazon EC2 instances**, choose the security group that you created in the previous steps.

Running AWS Blu Age Runtime on Amazon EKS on Amazon EC2

1. Create an Amazon EKS cluster. See Creating an Amazon EKS cluster in the *Amazon EKS User Guide*.
2. As mentioned previously, a security group is created on your behalf. You can use that security group when you create the Amazon VPC endpoint.

3. Create a node group. Specify the IAM role that you created in the previous steps.
4. Choose one of the instance types listed in the section called "Amazon EC2 instance types for AWS Blu Age Runtime (on Amazon EC2)".
5. Amazon EKS will automatically assign the security group to the spawned Amazon EC2 instances.

Running AWS Blu Age Runtime on Amazon ECS managed by AWS Fargate

Create an Amazon ECS cluster with **AWS Fargate (serverless)** as an underlying infrastructure. See Getting started with Fargate in the *Amazon Elastic Container Service Developer Guide*.

AWS Blu Age Runtime artifacts

AWS Blu Age Runtime artifacts are the components for deploying and running modernized applications. This document outlines the different types of artifacts available, their storage locations, and how to access them.

AWS Blu Age Runtime (non-managed) artifacts

Artifacts access and storage

The AWS Blu Age Runtime artifacts for non-managed deployments are stored in region-specific S3 buckets. Each version has its own dedicated folder, allowing for easy version management and access.

There are two types of buckets:

Release buckets

Release buckets contain directories for the most recently deployed versions and follow the naming convention: `aws-bluage-runtime-artifacts-<accountId>-<region>`.

Pre-release buckets

Pre-release buckets contain directories for alpha versions corresponding to the latest short-lived incremental pre-releases and follow the naming convention: `convention: aws-bluage-runtime-artifacts-dev-<accountId>-<region>`.

Access to the Production and to the Pre-release buckets are granted independently. For more information on how to request access, and more details about the S3 bucket organization, see the section called "Onboarding AWS Blu Age Runtime ".

Artifacts contents

In both Release and Pre-release buckets, you'll find:

aws-bluage-runtime-x.y.z.tar.gz

This archive, where x.y.z represents the version number (`major.minor.patch` as per semantic versioning, see the section called “AWS Blu Age versioning”), and contains the core AWS Blu Age Runtime components essential for executing AWS Blu Age applications, including:

- **Gapwalk**: A crucial component of the AWS Blu Age Runtime, designed to bridge the gap between legacy applications and modern cloud-native environments. It serves as a compatibility layer that allows applications modernized by AWS Blu Age to run effectively on contemporary platforms.
- **bluage.bin**: The core binary file of the AWS Blu Age Runtime. This file is critical for the operation of the runtime.
- All necessary libraries and supporting files for the AWS Blu Age Runtime operation.

aws-bluage-webapps-x.y.z.tar.gz

This archive, where x.y.z follows the same versioning scheme as above, includes web applications and libraries necessary for managing and controlling AWS Blu Age deployments:

- **BAC** (Blusam console) WAR file, which is used for monitoring the Blusam database.
- **JAC** (JICS console) WAR file, used for monitoring the JICS database.
- Necessary supporting libraries.

Additional files

- Checksum files that let you verify integrity for both Blu Age archives following the naming convention:
  - For Runtime: `aws-bluage-runtime-x.y.z.tar.gz.checksumSHA256`
  - For Webapps: `aws-bluage-webapps-x.y.z.tar.gz.checksumSHA256`
- CVE report files (For the release versions only) list the present CVEs on this version and follow the naming convention:
  - For Runtime: `Bluage-Runtime-x.y.z-CVEs.txt`
  - For Webapps: `Bluage-Webapps-x.y.z-CVEs.txt`

For details on how security vulnerabilities are addressed, see AWS Mainframe Modernization Refactor with AWS Blu Age release overview.

> **Note**
>
> While we strive to release our products without CVEs, new CVEs may appear later on. The CVE report file is updated regularly to reflect the latest status.

Developer AWS Blu Age Runtime artifacts

Artifacts access and storage

The AWS Blu Age Developer Runtime artifacts are stored in dedicated S3 buckets. This runtime includes both Release and Alpha pre-release versions. Access to these artifacts is managed through AWS Blu Age toolbox requests. Once your request is processed and approved, you'll be granted access to the appropriate bucket from the AWS account specified in your request.

Developer Runtime bucket

The main bucket for the Developer Runtime is: `s3://toolbox-dev-runtime`

For more detailed information on requesting access and understanding the bucket structure, see Dev and Special AWS Blu Age Runtimes documentation.

Artifact contents

Developer runtime artifacts typically include:

gapwalk-x.y.z-dev.tar.gz

This archive contains the development version of the Gapwalk component, which is a crucial part of the AWS Blu Age Runtime. It's designed to bridge legacy applications with modern cloud-native environments.

gapwalk-runtime-x.y.z-javadoc.zip

This zip file contains the JavaDoc documentation for the Gapwalk runtime. JavaDoc provides detailed API documentation, which is especially useful for developers working on integrating or extending the Gapwalk runtime.

gapwalk-webapps-x.y.z-javadoc.zip

Similar to the runtime JavaDoc, this zip file contains the JavaDoc documentation specifically for the Gapwalk web applications. This documentation is crucial for developers working with or customizing the web-based components of the Gapwalk system.

Deploy AWS Blu Age Runtime on Amazon EC2

You can learn how to set up AWS Blu Age Runtime (non-managed) on Amazon EC2, how to update the runtime version, how to monitor your deployment by using Amazon CloudWatch alarms, and how to add licensed dependencies with the topics in this section. These instructions are applicable when you create Amazon EC2 instances as well as when you use Amazon ECS on Amazon EC2 or Amazon EKS on Amazon EC2.

Topics

- Set up AWS Blu Age Runtime (non-managed) on Amazon EC2
- Upgrade the AWS Blu Age Runtime on Amazon EC2
- Set up AWS Blu Age Runtime (on Amazon EC2) Amazon CloudWatch alarms
- Set up licensed dependencies in AWS Blu Age Runtime on Amazon EC2

Set up AWS Blu Age Runtime (non-managed) on Amazon EC2

This topic explains how to set up and deploy the PlanetsDemo sample application using AWS Blu Age Runtime (non-managed) on Amazon EC2.

Topics

- Prerequisites
- Setting up
- Test the deployed application

Prerequisites

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

- Configure the AWS CLI by following the steps in Configuring the AWS CLI.
- Complete the section called “AWS Blu Age Runtime prerequisites” and the section called “Onboarding AWS Blu Age Runtime ”.

- Create an Amazon EC2 instance using one of the supported instance types. 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.

> **Note**
>
> Throughout this guide, the Tomcat installation path is assumed to be `/m2-anywhere/tomcat-gapwalk/velocity`. Ensure you use this path when following the instructions below or adapt the following instruction to the path of your choice.

- Download and extract AWS Blu Age Runtime (on Amazon EC2). Copy the contents of the velocity directory to `/m2-anywhere/tomcat-gapwalk/velocity`. Make sure to place the `bluage.bin` file exactly in the location specified by CATALINA_HOME environment variable described under CATALINA_HOME and CATALINA_BASE in the Apache Tomcat documentation. For instructions on how to retrieve the AWS Blu Age Runtime artifacts, including information about storage, access, and content, see the section called "AWS Blu Age Runtime artifacts".
- Download the PlanetsDemo application archive.
- Unzip the archive and upload the application to an Amazon S3 bucket of your choice.
- Create an Amazon Aurora PostgreSQL database for JICS. The AWS Blu Age Runtime will automatically execute the `PlanetsDemo-v1/jics/sql/initJics.sql` script during the first startup. For information about how to create an Amazon Aurora PostgreSQL database, see Creating and connecting to an Aurora PostgreSQL DB cluster.

Setting up

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

1. Connect to your Amazon EC2 instance and go to the `conf` folder under your Apache Tomcat 10 installation folder. Open the `catalina.properties` file for editing and replace the line that starts with `common.loader` with the following line.

```
common.loader="${catalina.base}/lib","${catalina.base}/lib/
*.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar","${catalina.home}/
shared","${catalina.home}/shared/*.jar","${catalina.home}/extra","${catalina.home}/
extra/*.jar"
```

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

3. 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.

4. Copy the content of `PlanetsDemo-v1/config/` folder to `/m2-anywhere/tomcat-gapwalk/velocity /config/`.
5. Provide the connection information for the database that you created as part of the prerequisites in the following snippet in the `application-main.yml` file. For more information see, Creating and connecting to an Aurora PostgreSQL DB cluster.

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

6. Start your Apache Tomcat server and verify the logs.

```
/m2-anywhere/tomcat-gapwalk/velocity/startup.sh

tail -f /m2-anywhere/tomcat-gapwalk/velocity/logs/catalina.log
```

If you find error codes that start with a C followed by a number, such as CXXXX, note the error messages. For example, error code C5102 is a common error indicating an incorrect infrastructure configuration.

Test the deployed application

For an example of how to test the PlanetsDemo application, see the section called “Test the PlanetsDemo application”.

Upgrade the AWS Blu Age Runtime on Amazon EC2

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

Topics

- Prerequisites
- Upgrade the AWS Blu Age Runtime in the Amazon EC2 instance
- Upgrade the AWS Blu Age Runtime in a container

Prerequisites

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

- To check if there are specific instructions for your version, see the section called “Upgrading AWS Blu Age”.
- Complete the section called “AWS Blu Age Runtime prerequisites” and the section called “Onboarding AWS Blu Age Runtime ”.
- Ensure that you have an Amazon EC2 instance that contains the latest AWS Blu Age Runtime. 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 that you want to upgrade to. For more information, see the section called “ Set up AWS Blu Age Runtime (non-managed)” The framework consists of two binary files: `aws-bluage-runtime-x.x.x.x.tar.gz` and `aws-bluage-webapps-x.x.x.x.tar.gz`.

Upgrade the AWS Blu Age Runtime in the Amazon EC2 instance

Complete the following steps to upgrade the AWS Blu Age Runtime.

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. 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. Therefore, 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.

```
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 Apache Tomcat services by using the following commands.

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

7. Replace the content of `<your-tomcat-path>/shared/` with the content of `aws-bluage-runtime-x.x.x.x/velocity/shared/`.
8. Replace `<your-tomcat-path>/webapps/gapwalk-application.war` with `aws-bluage-runtime-x.x.x.x/velocity/webapps/gapwalk-application.war`.
9. Replace the war files in `<your-tomcat-path>/webapps/`, namely `bac.war` and `jac.war`, with the same files from `aws-bluage-webapps-x.x.x.x/velocity/webapps/`.
10. Start the Apache Tomcat services by running the following commands.

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

11. Check the logs.

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

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

The following message should appear.

```
Jics application is running
```

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

The following message should appear.

```
Jics application is running
```

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

The response should be empty.

The AWS Blu Age runtime is successfully upgraded.

Upgrade the AWS Blu Age Runtime in a container

Complete the following steps to upgrade the AWS Blu Age Runtime.

1. Rebuild your Docker image with the desired AWS Blu Age Runtime version. For instructions, see the section called “Set up AWS Blu Age Runtime (non-managed) on Amazon EC2”.
2. Push your Docker image to your Amazon ECR repository.
3. Stop and restart your Amazon ECS or Amazon EKS service.
4. Check the logs.

The AWS Blu Age Runtime is successfully upgraded.

Set up AWS Blu Age Runtime (on Amazon EC2) Amazon CloudWatch alarms

You can set up CloudWatch to receive your application log and add an alarm to warn you of possible errors. This allows you to have more visible notifications whenever your deployed applications encounter exceptions. The following sections help you understand and learn about the configuration of CloudWatch logging and alarm setup.

Deployment of CloudWatch logging

By default, the AWS Blu Age Runtime contains a logging file named `logback-cloudwatch.yml`. This file is referenced in the `application-main.yml` file, but this reference is commented out.

```
# logging:
```

```
#  config: classpath:logback-cloudwatch.xml
```

Both files are in the config folder, and by uncommenting the above lines, the feature can be activated. CloudWatch logging can be configured, as explained in the following sections.

Configuration of CloudWatch logging

The default `logback-cloudwatch.xml` file has the following contents.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration>
<configuration>

    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS,UTC}  %level --- [%thread{15}]
 %logger{40} : %msg%n%xThrowable</pattern>
        </encoder>
    </appender>

    <appender name="cloudwatch"
 class="com.netfective.bluage.runtime.cloudwatchlogger.CloudWatchAppender">
        <logGroup>BluAgeRuntimeOnEC2-Logs</logGroup>
        <logStream>%date{yyyy-MM-dd,UTC}.%instanceId.%uuid</logStream>
        <layout>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS,UTC}  %level --- [%thread{15}]
 %logger{40} : %msg%n%xThrowable</pattern>
        </layout>
        <appender-ref ref="console" />
    </appender>

    <root level="INFO">
        <appender-ref ref="cloudwatch" />
    </root>
</configuration>
```

Everything outside the <appender name="cloudwatch"/> element is standard logback configuration. There are two appenders in this file: a console appender to send logs to the console and a CloudWatch appender to send logs to CloudWatch.

The `level` attribute in the `root` element specifies the logging level of the entire application.

The required values inside the tag <appender name="cloudwatch"/> are:

- <logGroup/>:Sets the name of the log group in CloudWatch. If the value is not specified it defaults to `BluAgeRuntimeOnEC2-Logs`. If the log group doesn't exist it will be created automatically. This behavior can be changed through configuration, which is covered below.
- <logStream/>: Sets the name of the logStream (inside of the log group) in CloudWatch.

Optional values:

- <region/>: Overrides the Region that the log stream will be written to. By default, logs go to the same Region as the EC2 instance.
- <layout/>: The pattern the log messages will use.
- <maxbatchsize/>: The maximum number of log messages to send to CloudWatch per operation.
- <maxbatchtimemillis/>: The time in milliseconds to allow for CloudWatch logs to be written.
- <maxqueuewaittimemillis/>: The time in milliseconds to try to insert requests in the internal log queue.
- <internalqueuesize/>: The maximum size of the internal queue.
- <createlogdests/>: Create log group and log stream if they don't exist.
- <initialwaittimemillis/>: The amount of time that you want the thread to sleep on startup. This initial wait allows for an initial accrual of logs.
- <maxeventmessagesize/>: The maximum size of a log event. Logs that exceed this size won't be sent.
- <truncateeventmessages/>: Truncate messages that are too long.
- <printrejectedevents/>: Enable the emergency appender.

CloudWatch setup

In order for the above configuration to correctly push logs to CloudWatch, update your Amazon EC2 IAM instance profile role to grant it additional permissions for the `BluAgeRuntimeOnEC2-Logs` log group and its log streams:

- `logs:CreateLogStream`
- `logs:DescribeLogStreams`
- `logs:CreateLogGroup`
- `logs:PutLogEvents`

- `logs:DescribeLogGroups`

Alarm setup

Thanks to CloudWatch logs, you can then configure different metrics and alarms, depending on your application and your needs. Specifically, you can set up proactive alarms for usage alerts, in order to be warned in the case of errors that might put your application in a grace period (and in the end, prevent it from working at all). To achieve this, you can add a metric concerning the "Error C5001" string in the logs, which highlights errors in the connection to the AWS Blu Age control system. You can then define an alarm that reacts to this metric.

Set 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
- Overview
- Set up the dependencies for JAC and BAC webapps

Prerequisites

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

- Complete the section called “AWS Blu Age Runtime prerequisites” and the section called “Onboarding AWS Blu Age Runtime ”.
- Make sure that you have an Amazon EC2 instance containing the latest AWS Blu Age Runtime (on Amazon EC2). 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 sources.

Oracle database

Supply an Oracle database driver. We tested the AWS Blu Age Runtime (on Amazon EC2) functionality with version **ojdbc11-23.3.0.23.09.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.jakarta.client-9.3.4.1.jar**, but a more recent version might be compatible.

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

- bcprov-jdk15to18-1.76.jar
- bcpkix-jdk15to18-1.76.jar
- bcutil-jdk15to18-1.76.jar

DDS Printer files

Supply the Jasper reports library (https://community.jaspersoft.com/download-jaspersoft/community-edition). 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.js8.jar
- javax.inject-1.jar
- jcommon-1.0.23.jar
- jfreechart-1.0.19.jar
- commons-beanutils-1.9.4.jar
- commons-collections-3.2.2.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 <your-tomcat-path>/extra/ folder.

```
cd <your-tomcat-path>/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
```

6. Verify the logs.

Set up the dependencies for JAC and BAC webapps

1. If your JICS database is hosted on Oracle, then you need to provide the Oracle database driver in <your-tomcat-path>/extra.
2. Create the folder if it is not present already.
3. Stop and restart your Apache Tomcat server.
4. Verify the logs.

Deploy AWS Blu Age Runtime on containers on Amazon ECS and Amazon EKS

You can use the topics in this section to learn how to set up AWS Blu Age Runtime on containers to deploy it on Amazon ECS (managed by Amazon EC2 or AWS Fargate), and Amazon EKS managed by Amazon EC2, how to update the runtime version, how to monitor your deployment by using Amazon CloudWatch alarms, and how to add licensed dependencies.

> **Note**
>
> This is not compatible with Amazon EKS managed by AWS Fargate.

Topics

- Set up AWS Blu Age Runtime on container
- Upgrade the AWS Blu Age Runtime on container
- Set up Amazon CloudWatch alarms for AWS Blu Age Runtime on container
- Set up licensed dependencies in AWS Blu Age Runtime on container

Set up AWS Blu Age Runtime on container

This topic explains how to set up and deploy the PlanetsDemo sample application using AWS Blu Age Runtime on a docker container.

AWS Blu Age Runtime on container is available for Amazon ECS managed by Amazon EC2, Amazon ECS managed by AWS Fargate, and Amazon EKS managed by Amazon EC2. It isn't compatible with Amazon EKS managed by AWS Fargate.

Topics

- Prerequisites
- Setting up
- Test the deployed application

Prerequisites

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

- Configure the AWS CLI by following the steps in Configuring the AWS CLI.
- Complete the section called “AWS Blu Age Runtime prerequisites” and the section called “Onboarding AWS Blu Age Runtime ”.
- Download the AWS Blu Age Runtime binaries. For instructions, see the section called “Onboarding AWS Blu Age Runtime ”.
- Download the Apache Tomcat 10 binaries.

- Download the PlanetsDemo application archive.
- Create an Amazon Aurora PostgreSQL database for JICS, and run the `PlanetsDemo-v1/jics/sql/initJics.sql` query on it. For information about how to create an Amazon Aurora PostgreSQL database see, Creating and connecting to an Aurora PostgreSQL DB cluster.

Setting up

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

1. After downloading the Apache Tomcat binaries, extract the contents, and go to the `conf` folder. Open the `catalina.properties` file for editing and replace the line that starts with `common.loader` with the following line.

```
common.loader="${catalina.base}/lib","${catalina.base}/lib/
*.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar","${catalina.home}/
shared","${catalina.home}/shared/*.jar","${catalina.home}/extra","${catalina.home}/
extra/*.jar"
```

2. Compress the Apache Tomcat folder by using the tar command to build a `tar.gz` archive.
3. Prepare a Dockerfile to build your custom image based on the provided runtime binaries and Apache Tomcat server binaries. See the following example Dockerfile. The goal is to install Apache Tomcat 10, followed by AWS Blu Age Runtime (for Amazon ECS managed by AWS Fargate) extracted at the root of Apache Tomcat 10 installation directory, and then to install the sample modernized application named PlanetsDemo.

> **Note**
>
> The contents of install-gapwalk.sh and install-app.sh scripts, which are used in this example Dockerfile, are listed after the Dockerfile.

```
FROM --platform=linux/x86_64 amazonlinux:2

RUN mkdir -p /workdir/apps
WORKDIR /workdir
COPY install-gapwalk.sh .
COPY install-app.sh .
RUN chmod +x install-gapwalk.sh
RUN chmod +x install-app.sh
```

```
# Install Java and AWS CLI v2-y
RUN yum install sudo java-17-amazon-corretto unzip tar -y
RUN sudo yum remove awscli -y
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
 "awscliv2.zip"
RUN sudo unzip awscliv2.zip
RUN sudo ./aws/install

# Installation dir
RUN mkdir -p /usr/local/velocity/installation/gapwalk
# Copy PlanetsDemo archive to a dedicated apps dir
COPY PlanetsDemo-v1.zip /workdir/apps/

# Copy resources (tomcat, blu age runtime) to installation dir
COPY tomcat.tar.gz /usr/local/velocity/installation/tomcat.tar.gz
COPY aws-bluage-runtime-4.x.x.tar.gz /usr/local/velocity/installation/gapwalk/
gapwalk.tar.gz

# run relevant installation scripts
RUN ./install-gapwalk.sh
RUN ./install-app.sh

EXPOSE 8080
EXPOSE 8081
# ...

WORKDIR /bluage/tomcat.gapwalk/velocity
# Run Command to start Tomcat server
CMD ["sh", "-c", "sudo bin/catalina.sh run"]
```

The following are the contents of `install-gapwalk.sh`.

```
# Vars
TEMP_DIR=/bluage-on-fargate/tomcat.gapwalk/temp

# Install
echo "Installing Gapwalk and Tomcat"
sudo rm -rf /bluage-on-fargate
mkdir -p ${TEMP_DIR}
# Copy Blu Age runtime and tomcat archives to temporary extraction dir
sudo cp /usr/local/velocity/installation/gapwalk/gapwalk.tar.gz ${TEMP_DIR}
```

```
sudo cp /usr/local/velocity/installation/tomcat.tar.gz ${TEMP_DIR}
# Create velocity dir
mkdir -p /bluage/tomcat.gapwalk/velocity
# Extract tomcat files
tar -xvf ${TEMP_DIR}/tomcat.tar.gz -C ${TEMP_DIR}
# Copy all tomcat files to velocity dir
cp -fr ${TEMP_DIR}/apache-tomcat-10.x.x/* /bluage/tomcat.gapwalk/velocity
# Remove default webapps of Tomcat
rm -f /bluage-on-fargate/tomcat.gapwalk/velocity/webapps/*
# Extract Blu Age runtime at velocity dir
tar -xvf ${TEMP_DIR}/gapwalk.tar.gz -C /bluage/tomcat.gapwalk
# Remove temporary extraction dir
sudo rm -rf ${TEMP_DIR}
```

The following are the contents of `install-app.sh`.

```
#!/bin/sh

APP_DIR=/workdir/apps
TOMCAT_GAPWALK_DIR=/bluage-on-fargate/tomcat.gapwalk

unzip ${APP_DIR}/PlanetsDemo-v1.zip -d ${APP_DIR}
cp -r ${APP_DIR}/webapps/* ${TOMCAT_GAPWALK_DIR}/velocity/webapps/
cp -r ${APP_DIR}/config/* ${TOMCAT_GAPWALK_DIR}/velocity/config/
```

4. Provide the connection information for the database that you created as part of the prerequisites in the following snippet in the `application-main.yml` file, which is located in the `{TOMCAT_GAPWALK_DIR}/config` folder. For more information see, Creating and connecting to an Aurora PostgreSQL DB cluster.

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

5. Build and push the image to your Amazon ECR repository. For instructions, see Pushing a Docker image in the Amazon Elastic Container Registry User Guide. Then, depending on your situation, either create an Amazon EKS pod or an Amazon ECS task definition using your Amazon ECR image, and deploy it to your cluster. For example on creating these, see Creating

a task definition using the console in the *Amazon Elastic Container Service (Amazon ECS) Developer Guide* and Deploy a sample application in the *Amazon EKS User Guide*.

6. Specifically, for **Amazon ECS managed by AWS Fargate** case, when creating the Task definition, use the IAM role you created as part of the initial Infrastructure setup. Then, while creating the service, expand the **Networking** section, and configure the VPC, subnets, and security group that you created as part of the initial Infrastructure setup. See, Infrastructure setup requirements for AWS Blu Age Runtime (non-managed).

Test the deployed application

For an example of how to test the PlanetsDemo application, see the section called "Test the PlanetsDemo application".

Upgrade the AWS Blu Age Runtime on container

This guide describes how to upgrade the AWS Blu Age Runtime on container. To do this, you need to first complete some prerequisites, and then work with Docker image to upgrade the AWS Blu Age Runtime.

Topics

- Prerequisites
- Upgrade the AWS Blu Age Runtime

Prerequisites

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

- Complete the section called "AWS Blu Age Runtime prerequisites" and the section called "Onboarding AWS Blu Age Runtime ".
- Download the version of the AWS Blu Age Runtime that you want to upgrade to. For more information, see the section called "Onboarding AWS Blu Age Runtime ". The framework consists of two binary files: `aws-bluage-runtime-x.x.x.x.tar.gz` and `aws-bluage-webapps-x.x.x.x.tar.gz`.

Upgrade the AWS Blu Age Runtime

Complete the following steps to upgrade the AWS Blu Age Runtime.

1. Rebuild your Docker image with the desired AWS Blu Age Runtime version. For instructions, see the section called “Set up AWS Blu Age Runtime on container”.
2. Push your Docker image to your Amazon ECR repository.
3. Stop and restart your Amazon ECS or Amazon EKS service.
4. Verify the logs.

The AWS Blu Age Runtime is successfully upgraded.

Set up Amazon CloudWatch alarms for AWS Blu Age Runtime on container

You can set up CloudWatch to have more visible notifications whenever your deployed applications encounter exceptions. This helps you to monitor your application log redirected to CloudWatch, and add an alarm to warn you of possible errors.

Alarm setup

With CloudWatch logs, you can configure any number of metrics and alarms, depending on your application and your needs.

Specifically, you can set up proactive alarms for usage alerts directly during your cluster creation, so that you get notified when errors occur. To highlight errors in the connection to the AWS Blu Age control system, add a metric concerning the string "Error C" in the logs. You can then define an alarm that reacts to this metric.

Set up licensed dependencies in AWS Blu Age Runtime on container

This topic describes how to set up additional licensed dependencies that you can use with AWS Blu Age Runtime on container.

Topics

- Prerequisites
- Overview

Prerequisites

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

- Complete the section called “AWS Blu Age Runtime prerequisites” and the section called “Onboarding AWS Blu Age Runtime ”.

- Get the following dependencies from their source.

Oracle database

Supply an Oracle database driver. For example, **ojdbc11-23.3.0.23.09.jar**.

IBM MQ connection

Supply an IBM MQ client. For example, **com.ibm.mq.jakarta.client-9.3.4.1.jar**.

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

- bcprov-jdk15to18-1.76.jar
- bcpkix-jdk15to18-1.76.jar
- bcutil-jdk15to18-1.76.jar

DDS Printer files

Supply the Jasper reports library (https://community.jaspersoft.com/download-jaspersoft/community-edition). For example, **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.js8.jar
- javax.inject-1.jar
- jcommon-1.0.23.jar
- jfreechart-1.0.19.jar
- commons-beanutils-1.9.4.jar
- commons-collections-3.2.2.jar

Overview

To install the dependencies, complete the following steps.

1. Copy any of the above dependencies as required to your Docker image build folder.
2. If your JICS database is hosted on Oracle, provide the Oracle database driver in *your-tomcat-path*/extra.
3. On your Dockerfile, copy these dependencies to *your-tomcat-path*/extra.
4. Build your Docker image, and then push it to Amazon ECR.
5. Stop and restart your Amazon ECS or Amazon EKS service.
6. Check the logs.

Test the PlanetsDemo application

To check the status of the deployed PlanetsDemo application, run the following commands after you replace `load-balancer-DNS-name`, `listener-port`, and `web-binary-name` with the correct values for your setup.

```
curl http://load-balancer-DNS-name:listener-port/gapwalk-application/
```

If the application is running, you see the following output message: `Jics application is running`.

Next, run the following command.

```
curl http://load-balancer-DNS-name:listener-port/jac/api/services/rest/jicsservice/
```

If the application is running, you see the following output message: `Jics application is running`.

```
Jics application is running
```

If you have configured Blusam, you can expect an empty response when you run the following command.

```
curl http://load-balancer-DNS-name:listener-port/bac/api/services/rest/bluesamserver/
serverIsUp
```

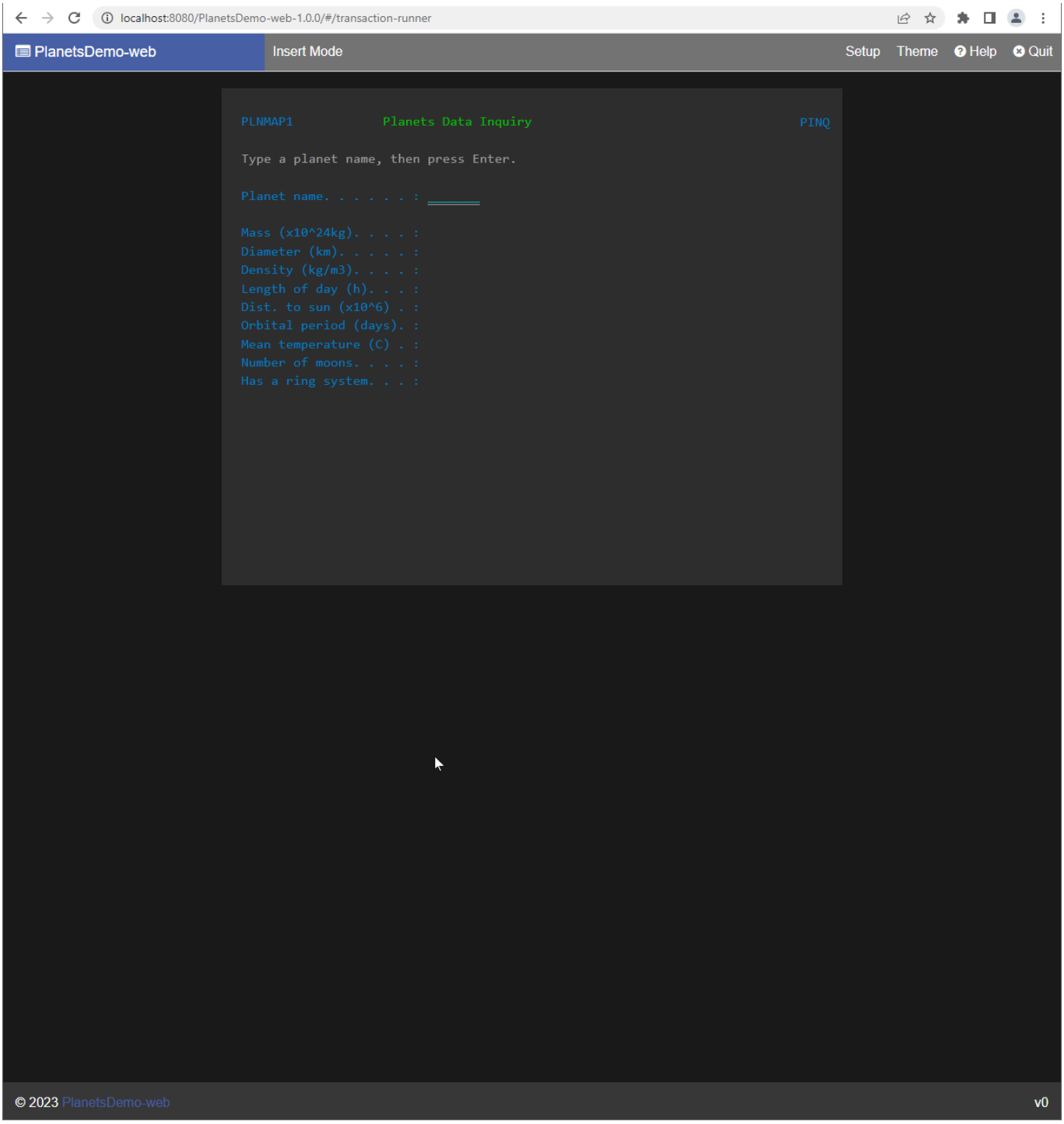
Note the name of the web binary (PlanetsDemo-web-1.0.0, if unchanged). To access the PlanetsDemo application, use a URL of the following format.

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

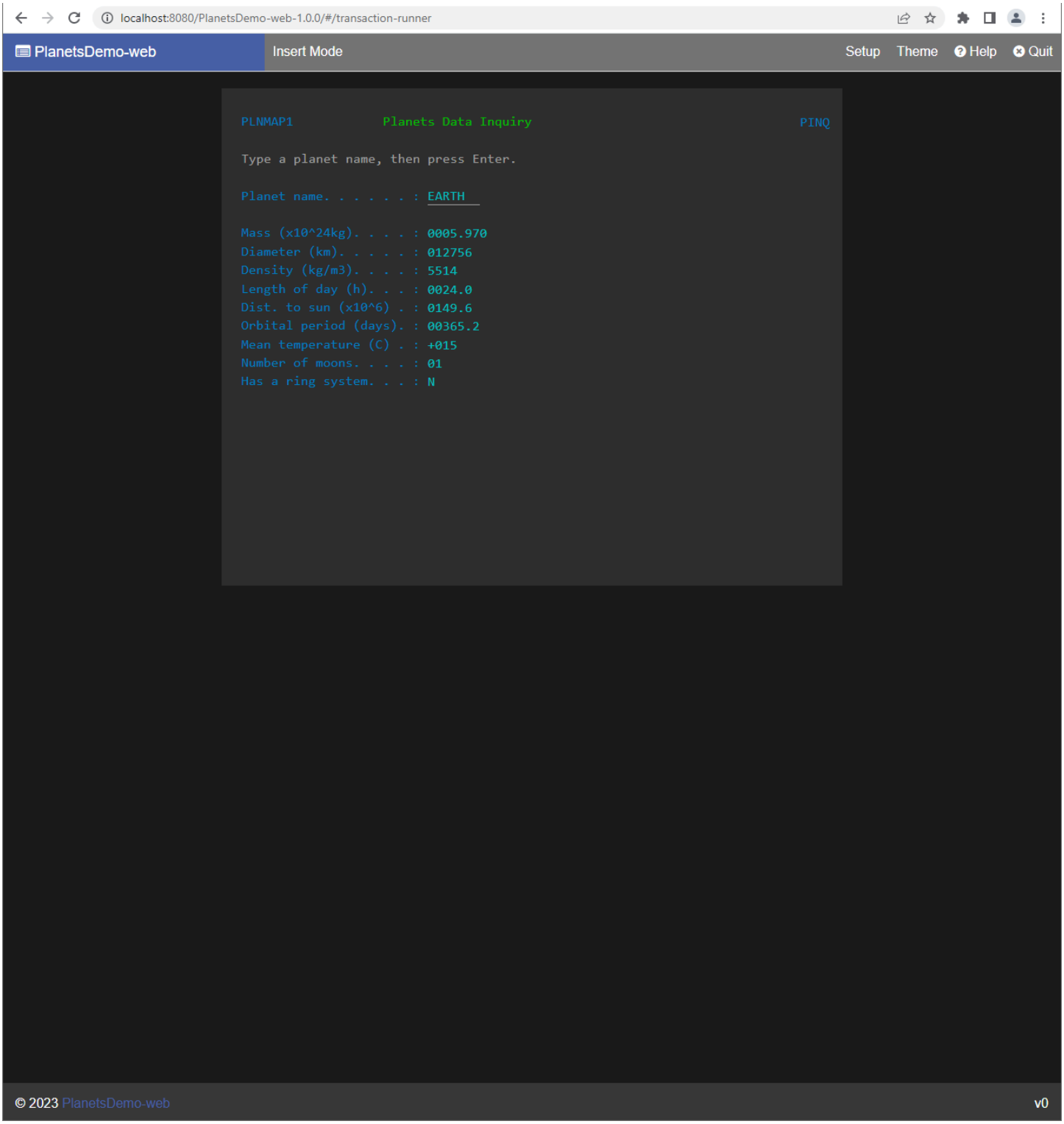
After the PlanetsDemo application starts, the home page is displayed.



Enter PINQ in the text box and then press Enter. The data inquiry page is displayed.



For example, enter EARTH in the PlanetsDemo name field, and then press Enter. The page for the planet you entered is displayed.



AWS Blu Age Runtime is available in the following regions: US East (Ohio), US East (N. Virginia), US West (N. California), US West (Oregon), Canada (Central), Europe (Ireland) Region, Europe (London) Region, Europe (Paris) Region, Europe (Frankfurt), Europe (Stockholm) Region, Europe (Milan) Region, Europe (Spain) Region, South America (São Paulo), Asia Pacific (Tokyo), Asia Pacific (Seoul),

Asia Pacific (Osaka), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Mumbai), Africa (Cape Town), and Israel (Tel Aviv).

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
- Tutorial: Use AWS Blu Age Developer on AppStream 2.0

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

- Prerequisites
- Step 1: Create an Amazon S3 bucket
- Step 2: Attach a policy to the S3 bucket

- Step 3: Upload files to the Amazon S3 bucket
- Step 4: Download AWS CloudFormation templates
- Step 5: Create the fleet with AWS CloudFormation
- Step 6: Access an instance
- Clean up resources

Prerequisites

For first time users, do this:

1. Navigate to the AppStream 2.0 console at https://console.aws.amazon.com/appstream2/home.
2. Choose **Get Started**.
3. Choose **Skip**.

> **Important**
>
> Amazon AppStream 2.0 uses IAM roles to manage your AppStream 2.0 resources and AWS will create these roles when you do this.

Then, 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. For more information on launching and connecting to an Amazon EC2 instance, see Get started with Amazon EC2.

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. For more information on buckets, see Creating a bucket.

Step 2: Attach a policy to the S3 bucket

Attach the following policy to the bucket that you create for this tutorial. For more information on attaching a policy to S3 bucket, see Adding a bucket policy.

Make sure to replace `amzn-s3-demo-bucket` with the actual name of the bucket that you create.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "AllowAppStream2.0ToRetrieveObjects",
        "Effect": "Allow",
        "Principal": {
            "Service": "appstream.amazonaws.com"
        },
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }]
}
```

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

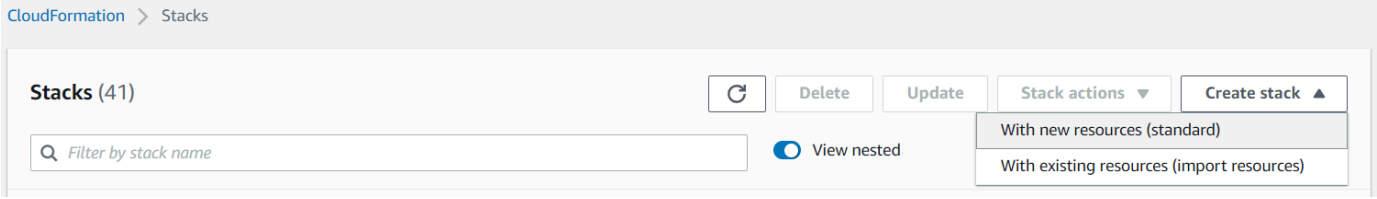
Download the following AWS CloudFormation templates. You need these templates to create and populate the AppStream 2.0 fleet.

- 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

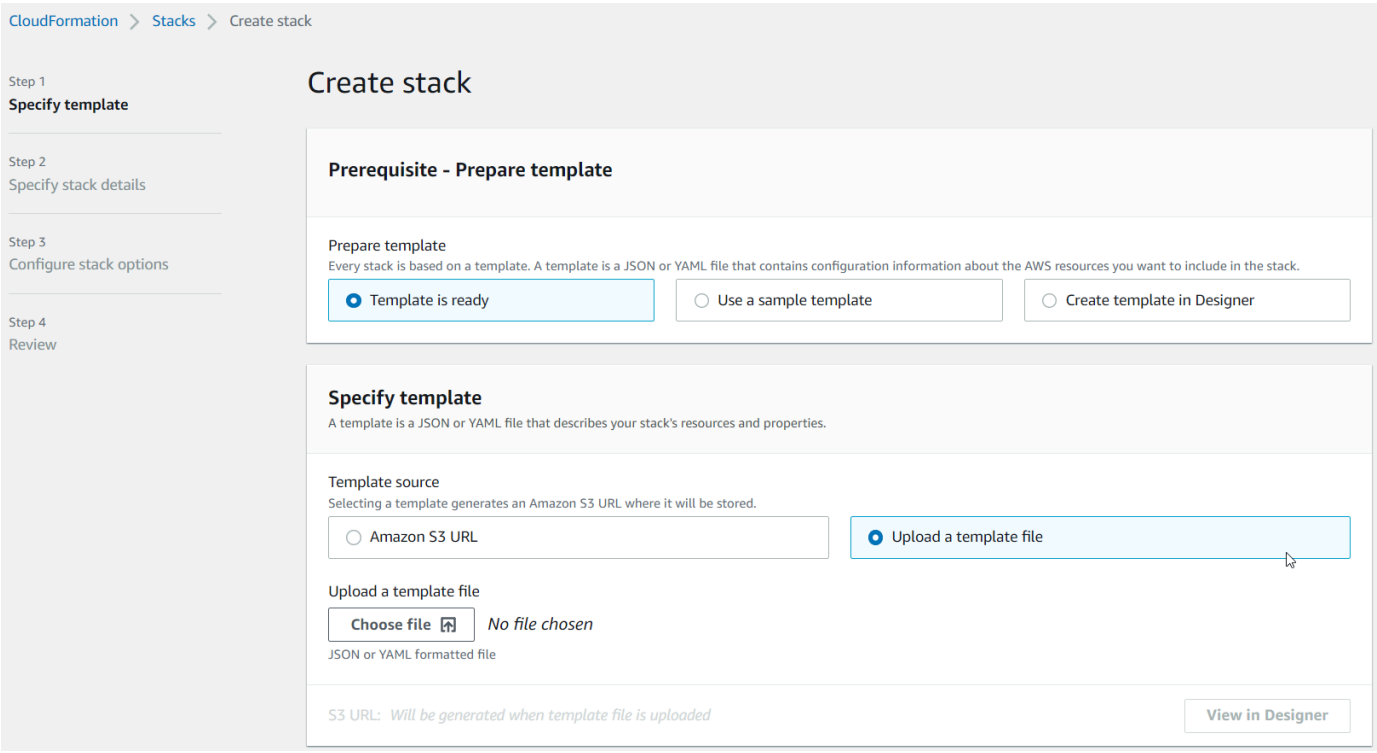
Step 5: Create the fleet with AWS CloudFormation

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 **Choose an existing template** 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.

> **Note**
>
> The two subnets of security group need to be in different availability zones.

6. Choose **Next**.
7. Navigate down the page and choose **I acknowledge that AWS CloudFormation might create IAM resources with custom names.**.
8. Choose **Next**.
9. Review the details, and choose **Submit**.
10. After you create the fleet, create CloudFormation stacks with all 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.

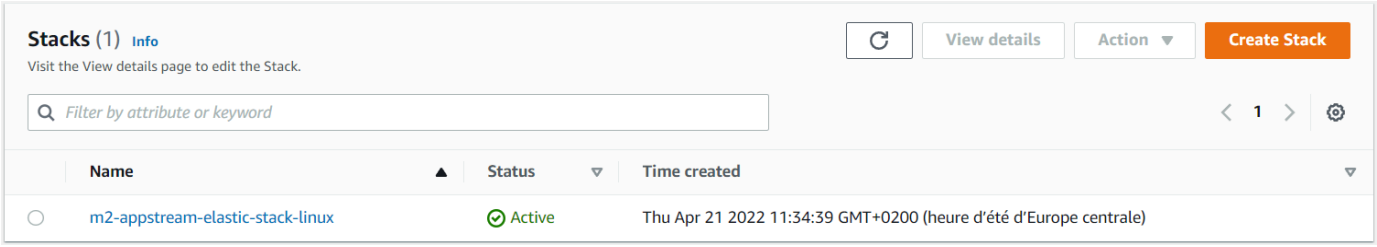
> **Note**
>
> The downloaded templates expect to find assets in an S3 bucket with a folder structure called `appstream/bluage/developer-ide/`. The bucket must be in the same AWS Region as the fleet that you created.

> **Important**
>
> Run all the CloudFormation scripts downloaded in step 4 to set up your application correctly.

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 the stack, and then choose **Associate fleet**.
3. In the prompt, choose the fleet you created and started previously.
4. Choose **Associate**.
5. Choose the associated stack and from the **Actions** menu, choose **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 8: Create the JICS database
- Step 9: Start and test the application
- Step 10: Debug the application
- Clean up resources

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. Set **Public access** to **Yes**.
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.

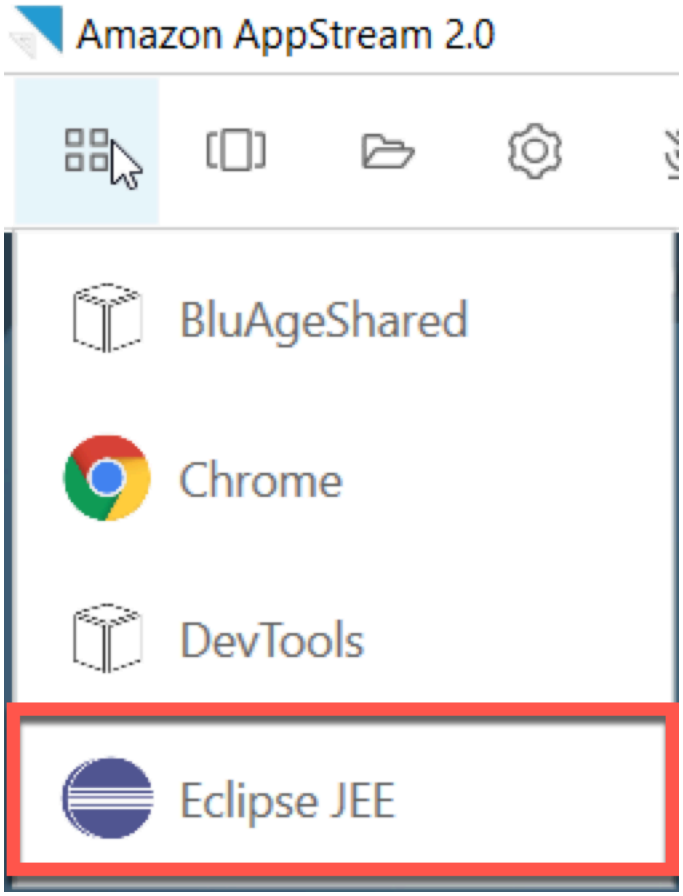
1. Choose **Applications**, from the menu bar, and then choose **Terminal**.
2. Enter the following command:

```
~/_install-velocity-runtime.sh
```

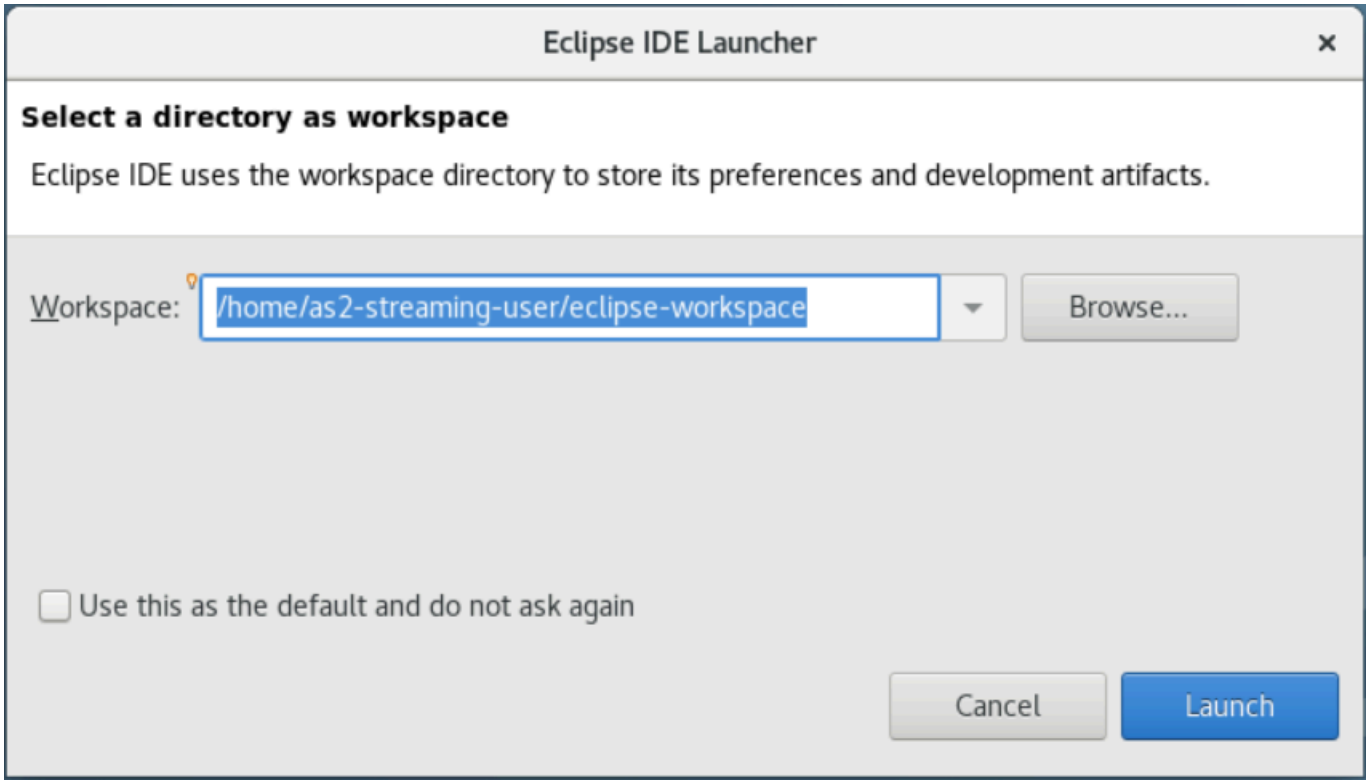
Step 4: Start the Eclipse IDE

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**.



Optionally, 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:

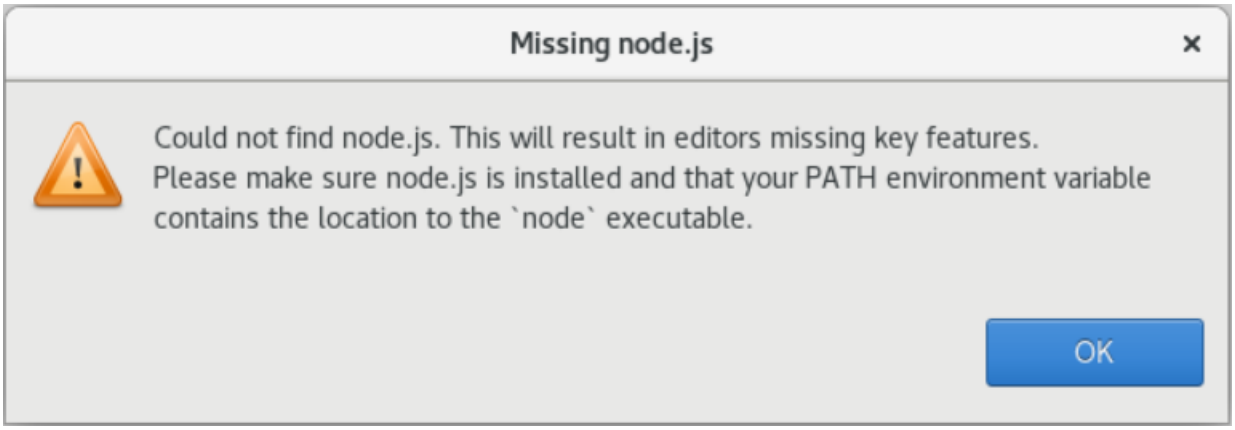
   ```
   cat ~/runtime-version.txt
   ```

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:

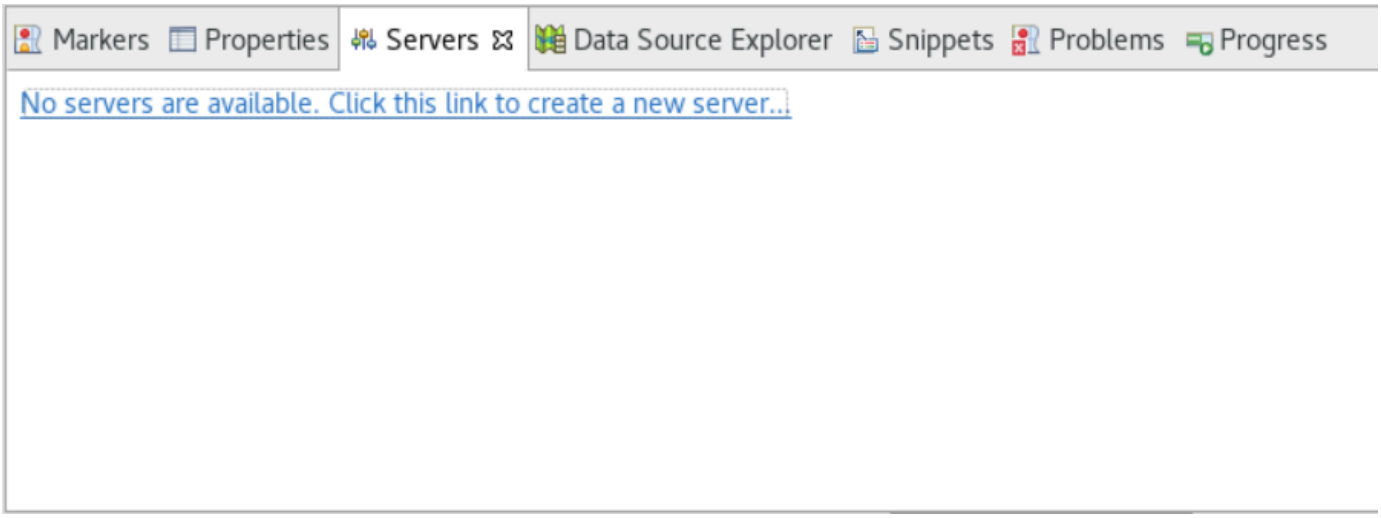
   

   Wait until the end of the build. You can follow the build in the **Progress** view.
7. In Eclipse, select the project and choose **Run as**. Then choose **Maven install**. After the Maven installation succeeds, it creates the `war` file under `PlanetsDemoPom/PlanetsDemo-web/target/PlanetsDemo-web-1.0.0.war`.

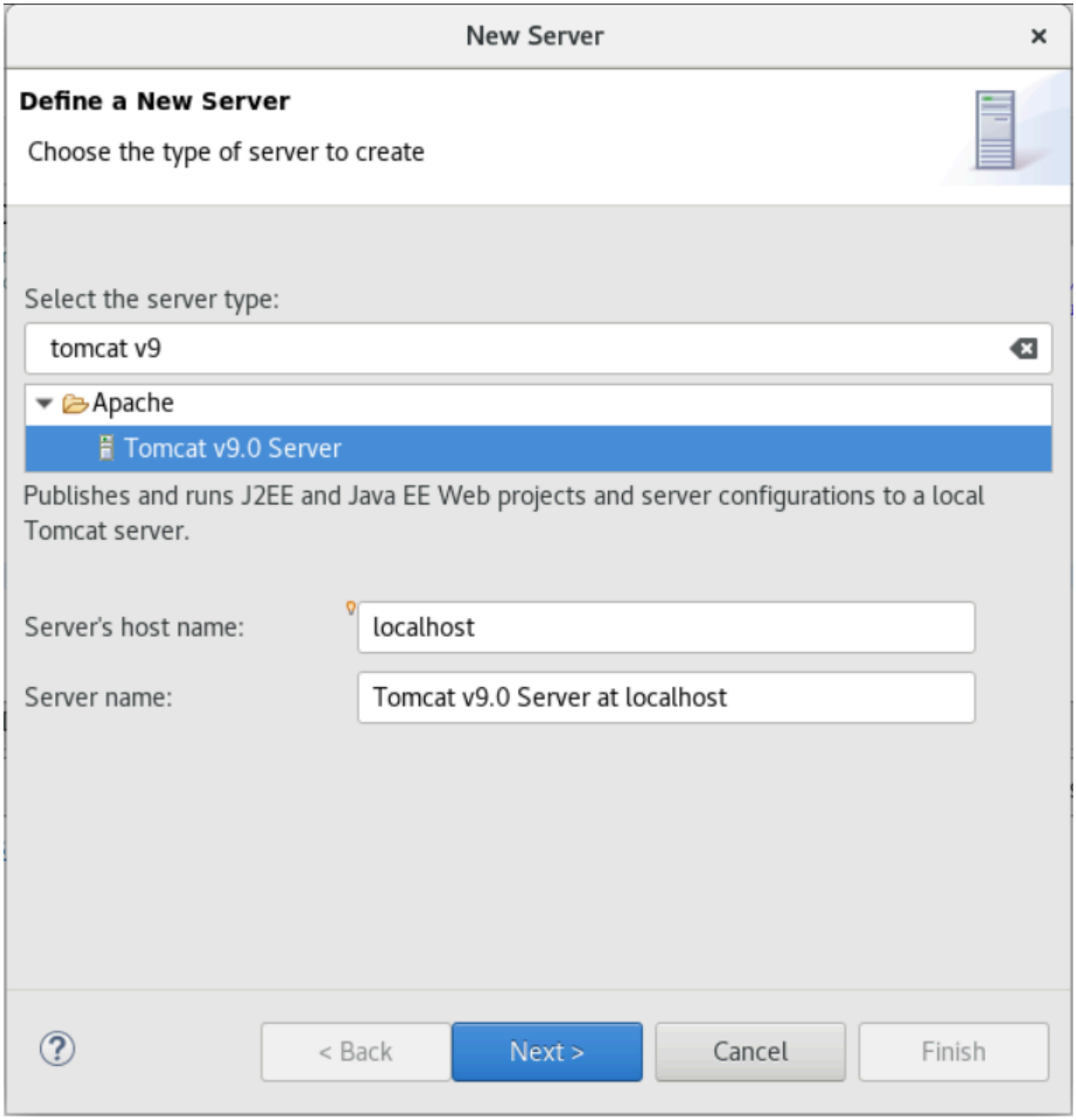
Step 6: Configure a Tomcat server

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:

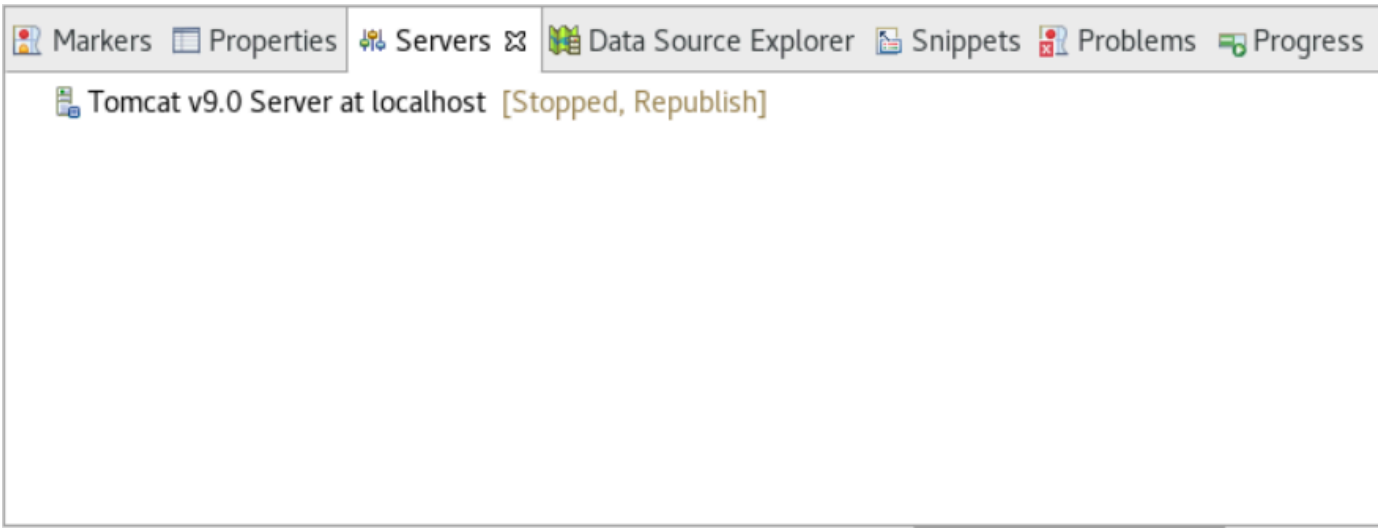


2. Choose **No servers are available. Click this link to create a new server...**. The **New Server**wizard appears. In the **Select the server type** field of the wizard, enter **tomcat v9** , and choose **Tomcat v9.0 Server**. Then choose **Next**.



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**.

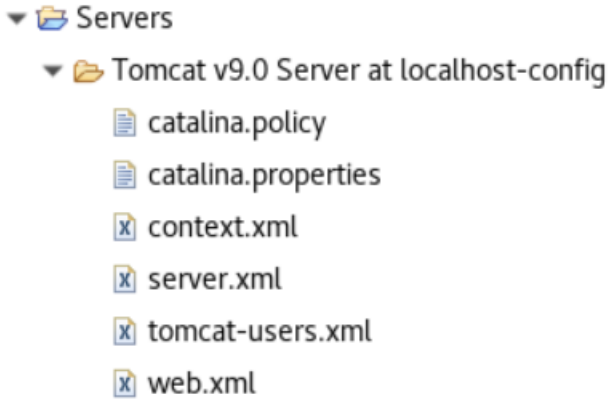
   A **Servers** project is created in the workspace, and a Tomcat v9.0 server is now available in the **Servers** view. This is where the compiled application will be deployed and started:



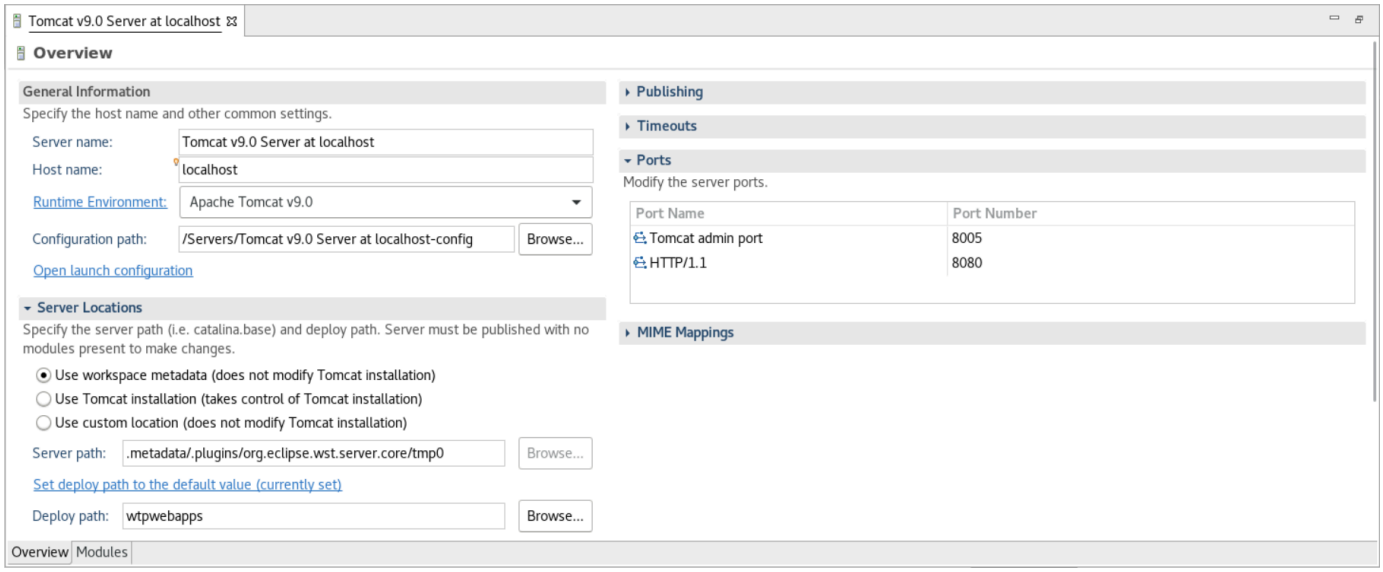
Step 7: Deploy to Tomcat

In this step, you deploy the Planets demo application to the Tomcat server so you can run the application.

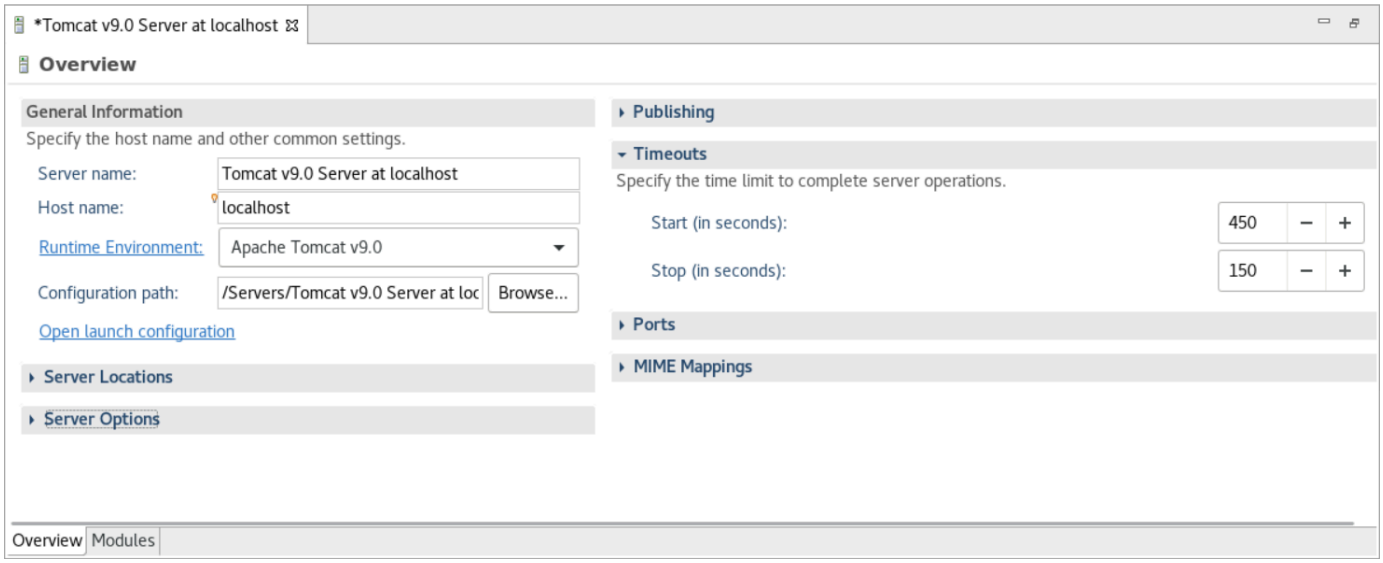
1. Select the `PlanetsDemo-web` file and choose **Run As > Maven install**. Select `PlanetsDemo-web` again and choose **Refresh** to ensure that the npm-compiled frontend is properly compiled to a .war and noticed by Eclipse.
2. Upload the PlanetsDemo-runtime.zip to the instance, and unzip the file at 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:



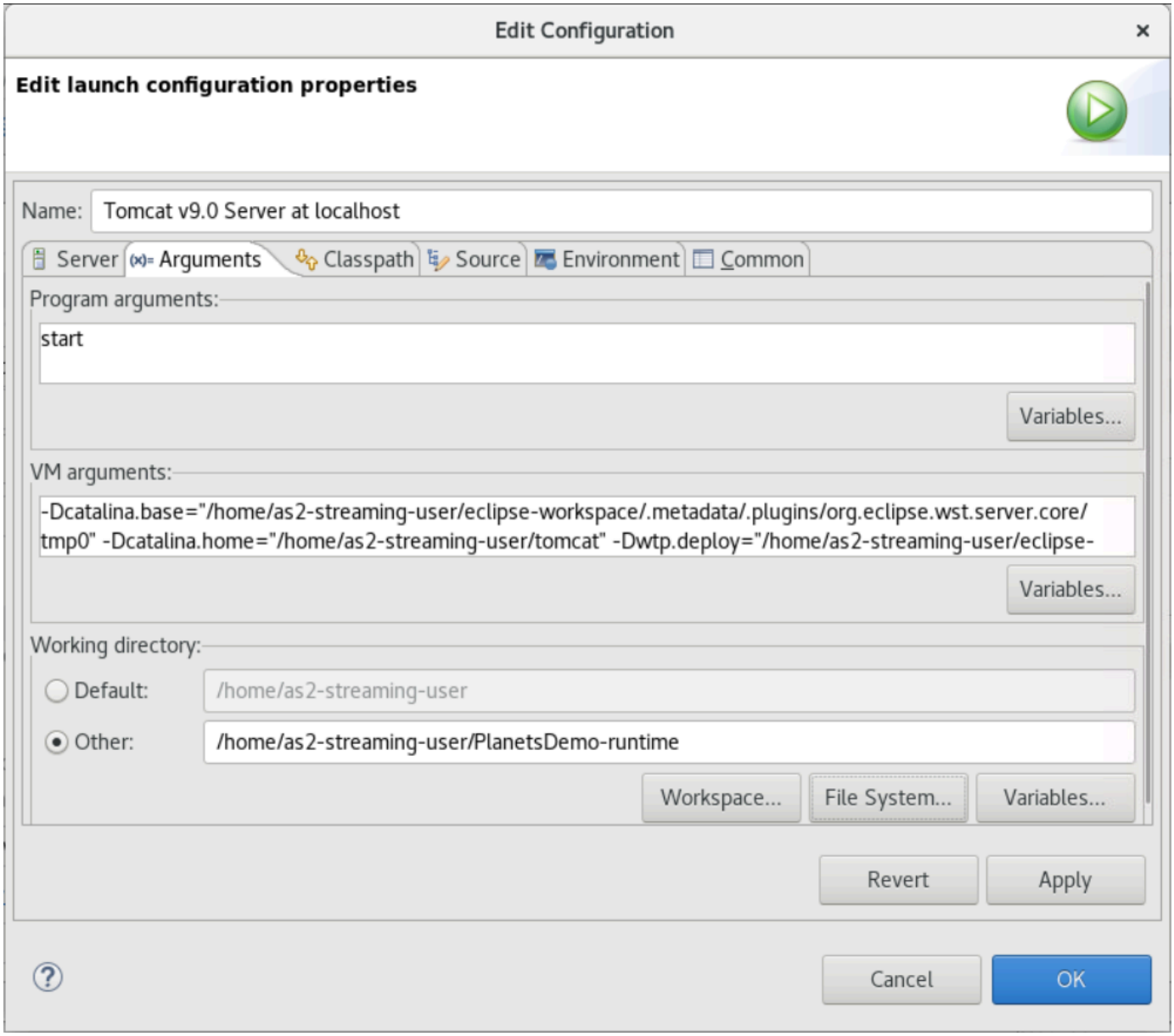
4. Open the `tomcat v9.0` server entry in the Servers view. The server properties editor appears:



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**.



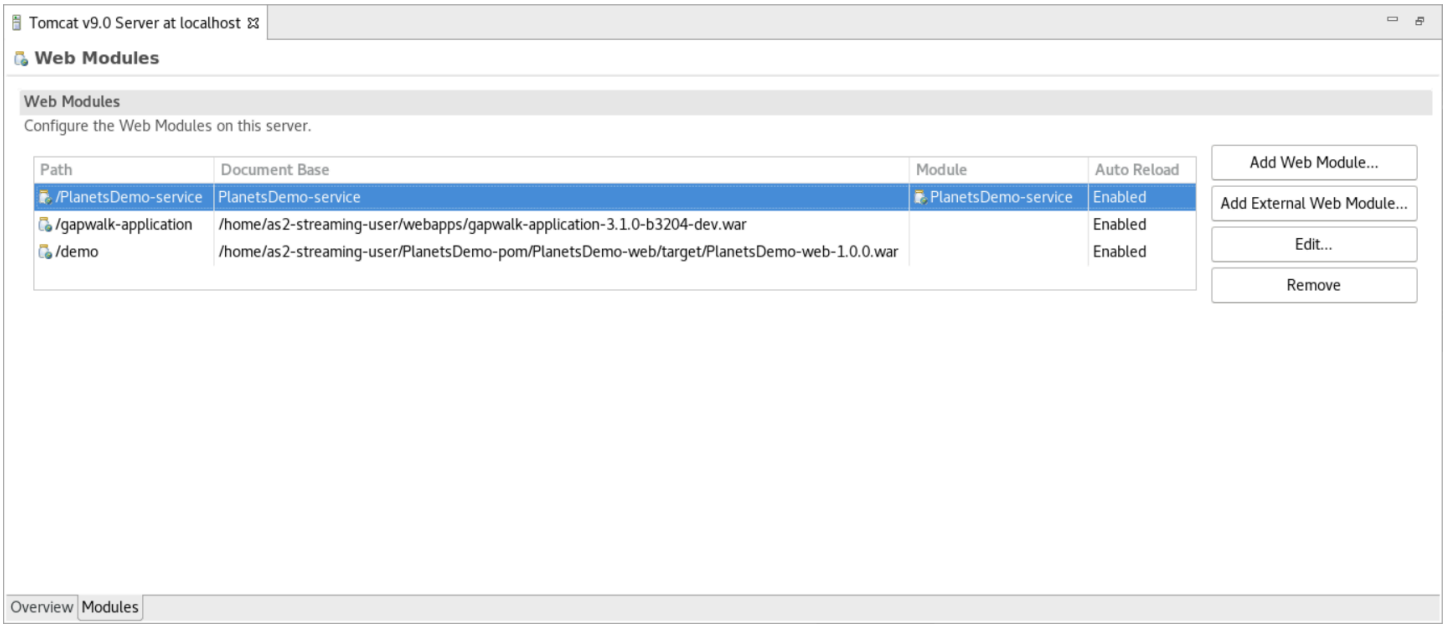
7. Choose the **Modules** tab of the server properties editor and make the following changes:

   - 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.



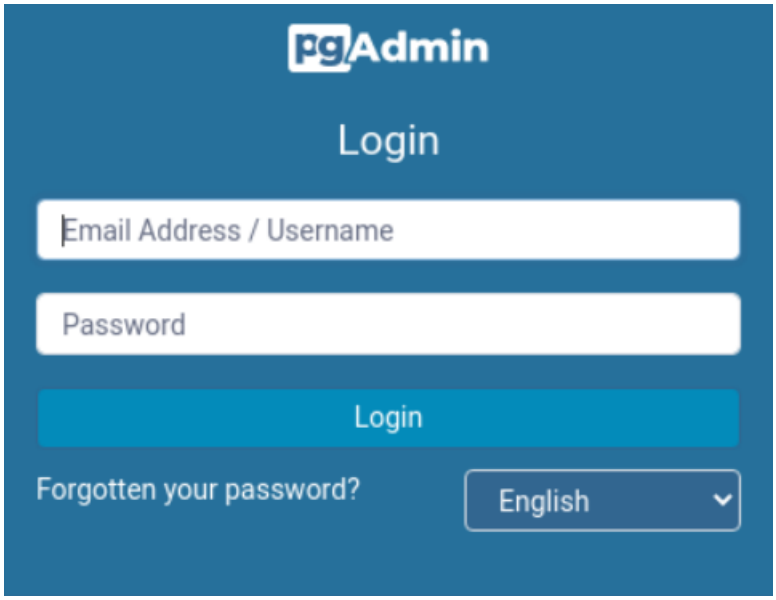
Step 8: Create the JICS database

In this step, you connect to the database that you created in Step 1: Create a database.

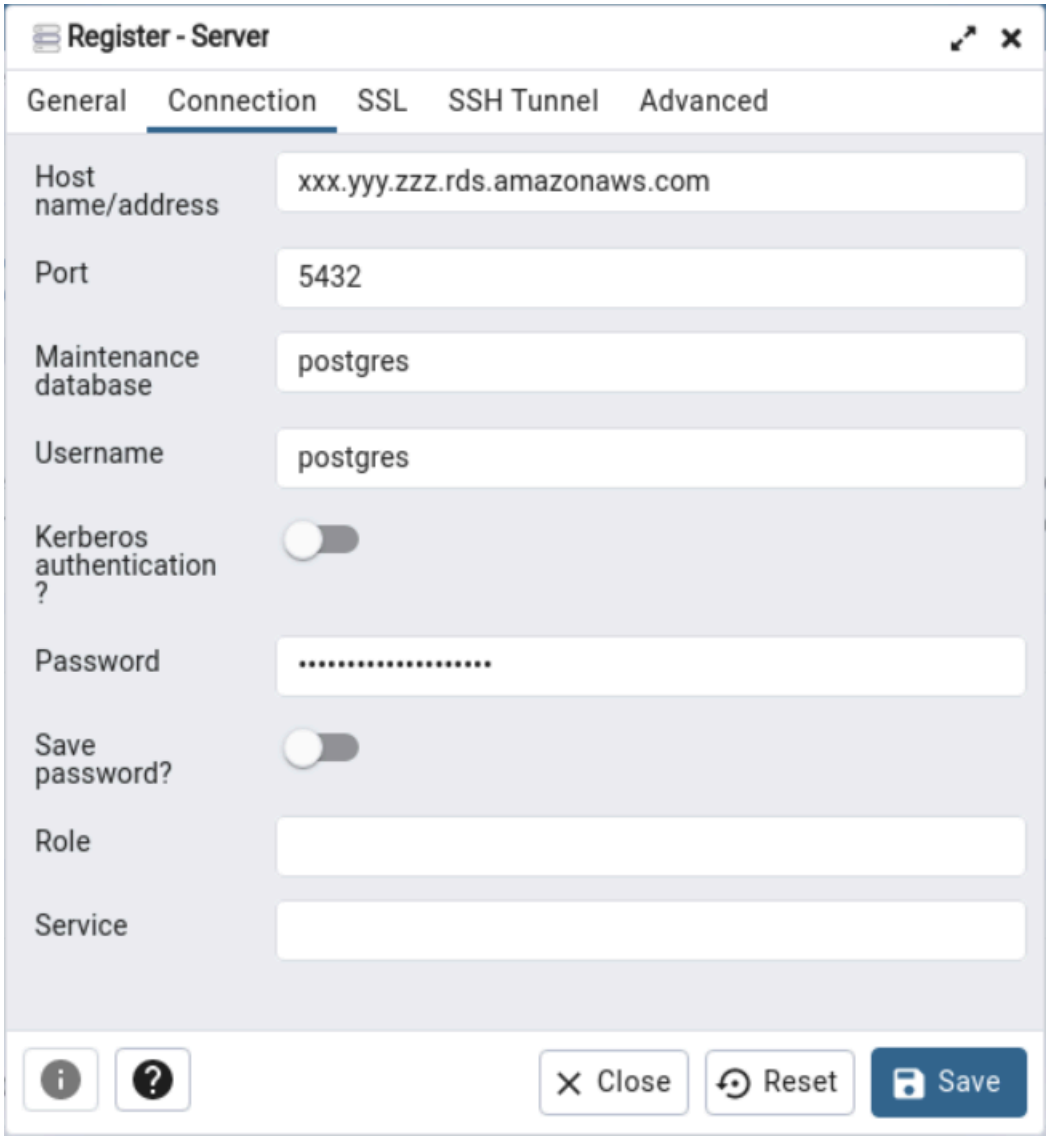
1. From the AppStream 2.0 instance, issue the following command in a terminal to launch `pgAdmin`:

   ```
   ./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.

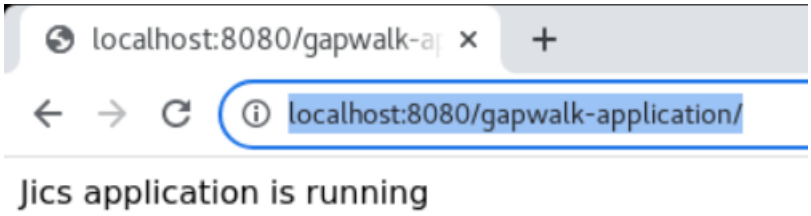


4. When you connect to the database server, use **Object > Create > Database** and create a new database named **jics**.
5. Edit the database connection information that the demo app used. This information is defined in `PlanetsDemo-runtime/config/application-main.yml`. Search for the `jicsDs` entry. 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**.

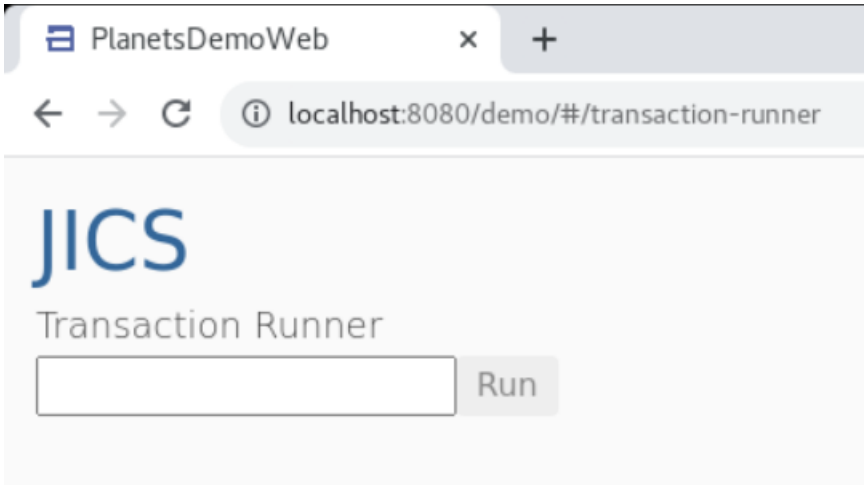
Step 9: Start and test the application

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.

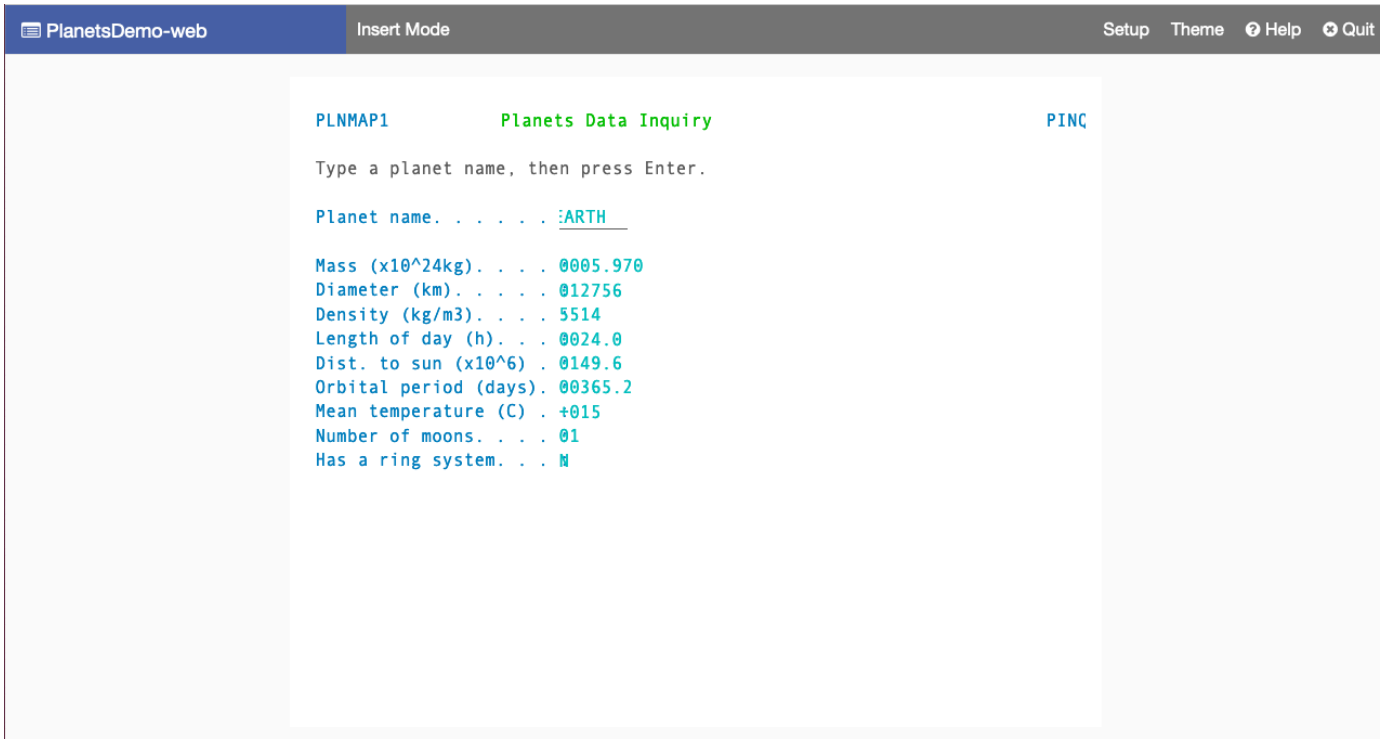


4. To start the application transaction, enter `PINQ` in the input field, and choose **Run** (or press Enter).

   The demo app screen should appear.



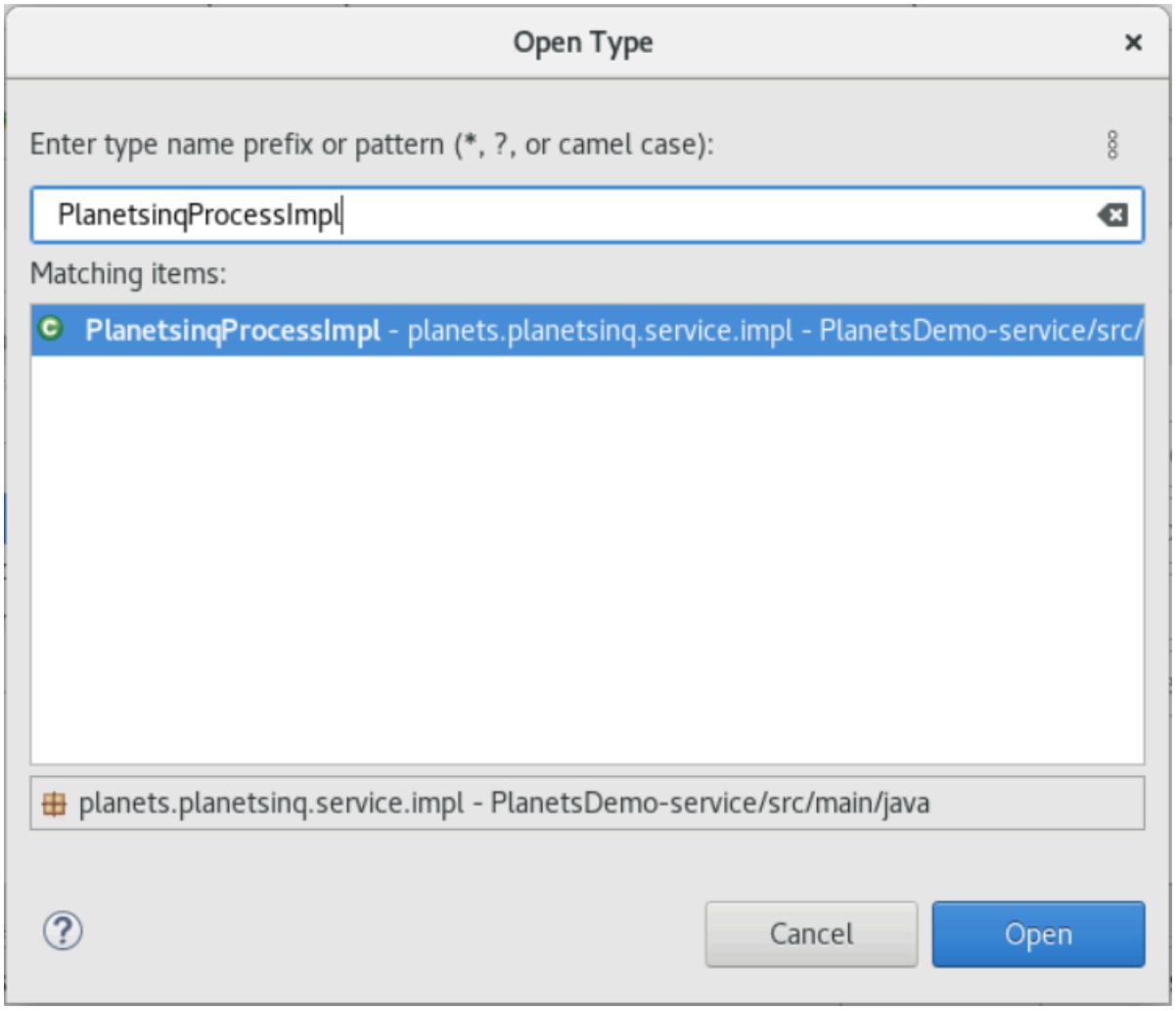
5. Type a planet name in the corresponding field and press Enter.



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. For more information, see Deleting a DB instance.

AWS Blu Age FAQ

General

1. **What is the main purpose of AWS Blu Age refactoring capability?**

   The refactoring capability refactors legacy monolithic code into java using contemporary distributed applications using modern languages and frameworks, following an automated refactoring pattern. This pattern involves automatically analyzing legacy code, understanding its functionality, and converting it into equivalent modern code while preserving business logic. The process includes modernizing not just the code, but also the entire application stack, dependencies, and infrastructure using automated tools and processes. The solution aims to speed up modernization while maintaining functional equivalence and performance. This includes transforming both application code and associated databases and data stores, while implementing cloud best practices and design patterns.

2. **Which mainframe applications are supported by AWS Blu Age?**

   AWS Blu Age currently supports the modernization of IBM z/OS mainframe applications written in COBOL, PL/I, JCL (Job Control Language) and relying on CICS (Customer Information Control System) transaction manager, BMS (Basic Mapping Support) screens, IMS MFS Screens, DB2 databases, IMS databases, Flat files, GDG (Generation data groups) and VSAM (Virtual Storage Access Method) data files. For more details, see AWS Blu Insights.

3. **What mainframe languages can AWS Blu Age modernize?**

   AWS Blu Age transforms COBOL and PL/I code to Java, JCLs to Groovy, screens (BMS or MFS) to HTML (with Sass) and JavaScript (Angular applications – React is not supported for now), enabling the modernization of legacy mainframe applications to cloud-native

architectures. These technologies are chosen for their widespread adoption, robust ecosystem, and cloud-native capabilities. Angular provides a modern, responsive user interface layer that replaces legacy green-screen interfaces. It enables the creation of dynamic, user-friendly web applications that can be accessed across different devices and platforms. Its component-based architecture supports maintainable and scalable front-end development. The transformation results in distributed applications that follow modern architectural patterns and best practices.

4. **How does AWS Blu Age balance legacy constraints with cloud benefits?**

   AWS Blu Age achieves balance by preserving critical business logic and functionality while introducing cloud-native capabilities. It ensures that modernized applications maintain necessary legacy business logic while taking advantage of cloud scalability, agility, and modern operational practices. This approach helps organizations maintain business continuity while gaining the benefits of cloud infrastructure.

5. **What role does service-oriented architecture play in the modernized application?**

   Service-oriented architecture plays a fundamental role in breaking down monolithic applications into more manageable modular components. AWS Blu Age creates service-oriented and object-oriented applications that facilitate better maintainability and scalability. This architectural approach enables organizations to achieve greater business efficiency and prepare for potential future microservices adoption.

6. **What aspects of the application stack are included in the refactoring process?**

   The refactoring process includes the complete software stack: application code, dependencies, databases, and infrastructure (e.g. options for caching, messaging support, etc). It covers the transformation of legacy programming languages, database systems, data files, and associated infrastructure components. This comprehensive approach ensures all aspects of the application are modernized cohesively, resulting in a fully transformed modern application stack.

7. **Does the AWS Blu Age modernization process eliminate the need for any testing or quality assurance checks on the modernized Java application?**

   No, the AWS Blu Age modernization process doesn't eliminate the need for any testing or quality assurance checks on the modernized Java application.

8. **What does AWS Blu Age JAC stand for?**

   JAC stands for JICS Administration Console

9. **How can I access the AWS Blu Age tooling?**

AWS Blu Age tooling is accessible through the AWS Console via AWS Mainframe Modernization (M2) Refactor, with feature access based on your accreditation level. Start with the Transformation Center to assess automatic Java refactoring of your source code. For detailed guidance, refer to the AWS Blu Insights documentation. After modernization, you can deploy applications using either managed or non-managed runtime options. For more information on these deployment choices, see AWS Mainframe Modernization documentation.

10. **How to size (workload and timeline) a project?**

    See AWS Blu Insights Estimates for more information on this or work with your Account Manager.

11. **Are there specific requirements to maintain Java AWS Blu Age migrated solutions?**

    No, there are no specific requirements to maintain Java AWS Blu Age migrated solutions.

12. **What are the technical specifications and compatibility of AWS Blu Age generated code?**

    AWS Blu Age generated code is designed with specific technical characteristics and broad compatibility. While it doesn't support JPA, it uses direct SQL execution with externalized queries. The code relies on runtime-specific libraries for functional equivalence, web services generation, and MQ implementations. The generated code can be imported into any Java IDE for development, testing, building, and deployment, though required libraries must be imported accordingly. While Maven is integrated by default with AWS Mainframe Modernization service for build processes, alternative tools like Gradle can be used by modifying the packaging format after transformation. The platform offers flexibility in terms of development tools and source control, with training available for development teams managing the code. For more information, see AWS Blu Age Runtime high level architecture.

AWS Blu Age Runtime

1. **Where can I find information about AWS Blu Age Runtime?**

   Refer the Set up AWS Blu Age Runtime (non-managed) documentation about non-managed runtime that details set-up process onboarding, retrieving artifacts, deployment, etc.

2. **Where can I find AWS Blu Age Runtime for Developers?**

   The AWS Blu Age Runtime for Developers is available in Blu Age Toolbox for L3 Certified individuals.

3. **Are the AWS Blu Age JAR dependencies uploaded to the client's Maven Repository for local development?**

   Libraries can be imported into EC2 using an AMI which can be used for configuring Development, Test & Production environment. Training & enablement will be given to the team to maintain/enhance the generated application code. For more information, see AWS Blu Age Runtime high level architecture.

4. **What does the term "Gapwalk" refer to in the distributed AWS Blu Age Runtime jars?**

   For information on Gapwalk, see AWS Blu Age Runtime artifacts .

5. **How to request access to the AWS Blu Age Runtime non-managed?**

   Follow instructions on Onboarding AWS Blu Age Runtime to request access to AWS Support center.

6. **What are the supported Runtimes for AWS Blu Age refactored applications?**

   To explore the full range of runtime choices for your modernized applications, we recommend reviewing the Blu Age Runtime Options guide.

7. **When is the AWS Blu Age Runtime used?**

   An AWS Blu Age Runtime is necessary to support the execution of AWS Blu Age refactored applications. A runtime is necessary during AWS Blu Age-based refactoring projects for testing the refactored applications. Once refactoring project is over, a runtime is also needed for maintaining, testing, and running AWS Blu Age refactored applications in production.

8. **How does AWS distribute new releases for AWS Blu Age Runtime?**

   For M2 Managed Runtime, updates, including patches, minor, and major versions, are made available in the AWS Console and AWS CLI. They include OS updates, engine, and dependency changes, typically within 30 days of general availability. AWS is responsible for supported components and applies updates to AWS Mainframe Modernization instances automatically. And it is the same case for other environments like Custom Runtime, Linux AMI, and on-premises.

9. **How often are new major and minor versions of AWS Blu Age runtime released?**

   New versions are released once a month or two, and customers can decide when and how to upgrade their runtime instances. For more information, see AWS Blu Age versioning page.

10. **How does AWS provide support for AWS Blu Age Runtime?**

Support is provided through AWS Support, where issues are addressed by raising a ticket, and the standard SLA applies. For more information, see AWS Mainframe Modernization components lifecycle.

11. **What does the AWS Mainframe Modernization AWS Blu Age Runtime entail?**

    The AWS Blu Age Runtime includes toolbox libraries for accelerating modernization, facilitating cloud integrations, and improving code quality and maintainability. It also enables more modernization automation by facilitating transitions between legacy architectures and cloud architectures. The runtime provides support for handling legacy verbs and data structures memory representations using java idioms. It allows building modernized applications based on object-oriented programming techniques and able to reproduce legacy control flows. It modernizes legacy VSAM data sets or IMS hierarchical databases support using a relational database such as Amazon Aurora. It provides java replacements for legacy system utilities (IDCAMS, IEBGENER, DFSORT,etc), and legacy transaction management systems (CICS, IMS). It facilitates cloud integrations with caching in Amazon ElastiCache and support for AWS messaging solutions (SQS, Kinesis).

12. **Does AWS Blu Age Runtime support non-x86 computer architectures?**

    Currently, AWS Blu Age Runtime only support x86-based computer architectures and compute. AWS Blu Age Runtime doesn't support ARM-based and Graviton-based compute.

13. **How can customers stay informed about AWS Blu Age Runtime versions, including notifications of new releases and access to version history and release notes?**

    New versions of AWS Blu Age Runtime are uploaded to our official release page. We recommend checking this page regularly, ideally every 3 months, for the latest versions and updates. Regarding access to version history and release notes, availability depends on the end-of-life (EOL) date for each major version. For detailed information on EOL dates, version upgrade planning, and access to historical information, see AWS Blu Age lifecycle.

14. **What are the main components of AWS Blu Age Runtime high-level architecture?**

    The AWS Blu Age Runtime architecture comprises two main component types. First are Java libraries (jar files) stored in a shared folder (accessible to the application server classloader) that provide legacy constructs and statements support. Second are web applications (war files) containing Spring-based applications that provide frameworks and services to modernized programs. The runtime also includes: a Programs Registry that collects all programs for invocation and cross-program calls and a Scripts Registry that collects all modernized jobs

scripts. These components work together to provide a unified REST-based entry point and execution framework for modernized applications. The Runtime and the modernized application are deployed together in an application server (e.g. Tomcat).

15 **How to configure the shared folder holding AWS Blu Age Runtime artifacts?**

The AWS Blu Age Runtime artifacts (jars) must be gathered in a shared folder, accessible to the application server classloader. For a tomcat server, the configuration is made by modifying the regular configuration file named catalina.properties. For instance, if you created the shared folder as a folder named “shared”, in the tomcat folder, you will need to modify the common.loader entry in the catalina.properties to make the shared folder accessible to the tomcat classloader, as such:

```
common.loader="${catalina.base}/lib","${catalina.base}/lib/*.jar","${catalina.home}/
lib","${catalina.home}/lib/*.jar","${catalina.home}/shared","${catalina.home}/shared/
*.jar"
```

16 **How does AWS Blu Age Runtime handle statelessness and session management?**

AWS Blu Age Runtime implements statelessness and session management through multiple mechanisms. For HTTP sessions, it uses cookie-based identification with external cache storage for user context. Sessions can be stored in various datastores including Amazon ElastiCache, Redis cluster, or in-memory maps. The statelessness design ensures that most non-transient states are stored externally in a common 'single source of truth', enabling high availability and horizontal scaling. This approach, combined with load balancing and shared sessions, allows distribution of user-facing dialog across multiple nodes.

17 **What role do web applications play in the AWS Blu Age Runtime environment?**

Web applications in AWS Blu Age Runtime serve multiple key functions. They provide execution frameworks that reproduce legacy environments and transaction monitors (like JCL batches, CICS, IMS). They offer REST-based entry points through the `gapwalk-application.war` for triggering and controlling transactions, programs, and batches. Additionally, they provide emulation of OS-provided programs and specialized 'driver' programs that legacy applications depend on for accessing services like IMS DB or user dialogs through MFS.

18 **How are programs registered and managed in AWS Blu Age Runtime?**

Programs in AWS Blu Age Runtime are registered through a ProgramRegistry system that populates during server startup. Each program implements the Program interface and is marked as a Spring component. Programs are registered using their identifiers, with multiple

entries possible if a program has several identifiers. The registration process is automatic and logged in Tomcat logs. The ProgramRegistry enables other programs and scripts to locate and call registered programs, maintaining the modularity and interconnectivity of the modernized system.

19 **How is configuration managed in AWS Blu Age Runtime applications?**

Configuration in AWS Blu Age Runtime is managed through YAML files using Spring Boot framework capabilities. Two main configuration files are used: application-main.yml for framework configuration and `application-profile.yml` for client-specific options. The system follows Spring's precedence logic, allowing configuration overrides through various means. Additional configuration can be provided through JNDI for databases and command-line parameters, offering flexibility in configuration management. Loggers configuration is done using logback xml configuration files.

20 **What role do secrets managers play in the AWS Blu Age Runtime configuration?**

Secrets managers in AWS Blu Age Runtime secure sensitive configuration data like database credentials and Redis cache passwords. They allow storage of critical data in AWS secrets and reference them in YAML configuration files. The system supports different types of secrets, including database secrets that automatically populate all relevant fields and single-password secrets for password-protected resources. This approach enhances security by keeping sensitive data separate from application configuration.

21 **How can developers write their own programs compatible with AWS Blu Age Runtime?**

Developers can create AWS Blu Age Runtime-compatible programs by implementing the Program interface and following specific patterns. The program must be declared as a Spring component, implement required methods, and be properly registered in the ProgramRegistry. Developers need to create companion context and configuration classes, handle program identifiers, and ensure proper integration with the Spring framework. The implementation should follow AWS Blu Age Runtime conventions for program structure and execution.

22 **How does AWS Blu Age Runtime handle program execution errors?**

AWS Blu Age Runtime handles program execution errors through multiple mechanisms. For batch jobs, it captures execution status, exit codes, and detailed error information in the job execution details. Error handling includes specific exit codes (-1 for technical errors, -2 for service program failures) and detailed logging in Tomcat logs. The system can be configured to rollback transactions on runtime exceptions and provides options for error

notification and recovery. Error details are accessible through REST endpoints for monitoring and troubleshooting.

23. **What AWS Blu Age Runtime monitoring capabilities are available for batch jobs?**

    AWS Blu Age Runtime provides monitoring capabilities for batch jobs through various endpoints. It tracks job execution status, start/end times, execution mode, and detailed results. The system offers endpoints for listing triggered scripts, retrieving job execution details, and monitoring currently running jobs. Metrics' endpoints provide JVM statistics, session counts, and detailed batch execution metrics. The platform also supports pagination and time-based filtering of monitoring data.

24. **How are AWS Blu Age Runtime job execution statuses tracked and managed?**

    Job execution statuses are tracked through a comprehensive status system that includes states like DONE, TRIGGERED, RUNNING, KILLED, and FAILED. Each job execution receives a unique identifier for tracking and maintains detailed execution information including start time, end time, caller information, and execution results. The system provides REST endpoints for querying job status, managing running jobs, and retrieving execution history. Status information persists in server memory and can be purged based on age for resource management.

25. **How does AWS Blu Age Runtime handle external system interactions?**

    The runtime handles external system interactions through various mechanisms, including REST endpoints for service integration, support for message queues (SQS, RabbitMQ, IBM MQ), and database connectivity options. It provides emulation of legacy system interactions through specialized components, supports SSL/TLS for secure communications, and includes features for handling external file systems. The system also supports integration with external authentication providers and can be configured to interact with various third-party services.

26. **How is authentication handled in AWS Blu Age Runtime?**

    AWS Blu Age Runtime supports multiple authentication methods, with OAuth2 being the primary mechanism. It can integrate with identity providers like Amazon Cognito or Keycloak. Authentication configuration is managed through the main configuration file named application-main.yml, where security settings, identity providers, and authentication methods can be defined. The system supports features like XSS protection, CORS, CSRF, and can be configured for both global security and specific endpoint security. For development, a local authentication system with default super admin credentials is also available.

27. **How does AWS Blu Age Runtime ensure high availability?**

AWS Blu Age Runtime ensures high availability through several mechanisms. It implements statelessness by storing non-transient states in external shared storage, enabling multiple application instances to work together. The system supports load balancing and shared sessions, allowing requests to be distributed across multiple nodes. For data storage, it can utilize highly available databases and caching systems. The architecture supports automatic fail-over and can be deployed across multiple availability zones for increased reliability.

28. **What component is used to reproduce CICS distributed transactions with AWS Blu Age applications?**

    The AWS Blu Age Runtime provides a dedicated endpoint to allow existing JICS transactions to be invoked as part of a global transaction (XA support). The underlying two phases commit support relies on the Atomikos software component.

29. **What is the AWS Blu Age name of the classes that are used to define specific program behavior?**

    Each program is bound to a dedicated Configuration class which allows to specify the program specific behaviors. For more information on naming and location conventions, see AWS Blu Age structure of modernized application

30. **Which encoding has the following character sequence order: space, lowercase characters, uppercase characters, numerals?**

    Charsets belonging to the EBCDIC variants family (such as CP1047, CP297, etc).

31. **How do you operate AWS Blu Age managed Runtime?**

    With the AWS Management Console, the AWS CLI, or the AWS APIs

32. **What are the pricing dimensions for AWS Blu Age Runtime?**

    AWS Mainframe Modernization-core-hours (See AWS Mainframe Modernization pricing).

33. **What is the mechanism used to pass raw data through HTTP to the program endpoints?**

    Base64 encoded strings.

34. **How does a user launch a batch job run?**

    Using an HTTP call to one of the dedicated batch endpoint (see batch endpoints documentation page).

35. **Which AWS Blu Age Runtime endpoint is the main entry point from the main web front-end application?**

```
/transaction
```

36. **What does AWS Blu Age JICS stand for?**

    The AWS Blu Age JICS is the runtime component used to support the modernization of CICS resources. The resources definitions are stored in a dedicated data store. To administer them, use either the REST API or the JICS application console. For information, see Manage JICS application console in AWS Blu Age.

37. **What AWS Blu Age Runtime caching mechanisms are available?**

    AWS Blu Age Runtime supports multiple caching mechanisms, including Redis and EhCache. Redis is recommended for production environments, providing shared persistent caching across multiple nodes. EhCache is available for standalone deployments with embedded volatile local caching. The system supports caching for various components, including Blusam data, session information, JICS resources, and temporary storage queues. Cache configuration can be customized for different use cases and performance requirements.

38. **How do we estimate the price of an AWS Mainframe Modernization AWS Blu Age Runtime deployment?**

    AWS provides estimates to customers based on their requirements and target architecture.

39. **What is the AWS Mainframe Modernization AWS Blu Age Runtime price?**

    AWS Mainframe Modernization offers two pricing models for AWS Blu Age: a Managed Runtime option that includes the runtime, compute resources, internal storage, and automation, and a Non-managed Runtime option that covers the AWS Blu Age runtime itself only. For AWS deployments, both use a pay-as-you-go pricing structure. For the most up-to-date and detailed pricing information, it's recommended to consult the official AWS Mainframe Modernization Pricing page.

40. **What if we need to deploy an AWS Blu Age refactored application on an infrastructure not listed in the supported runtime?**

    If you need to deploy an AWS Blu Age refactored application on an infrastructure not listed in the supported runtime, several options are available. First, check if your infrastructure is compatible with existing deployment options like Amazon EKS Anywhere or other container orchestration platforms. If so, you may be able to use the AWS Blu Age Runtime (non-managed). For non-compatible infrastructures, we recommend consulting with an AWS mainframe specialist to explore custom solutions or potential adaptations. You can also submit a Product

Feature Request (PFR) for expanded infrastructure support. Alternative billing options may be available for non-standard deployments. Contact your AWS representative to discuss your specific needs and the best approach for your environment.

41 **How is the AWS Blu Age Runtime licensed? Is it open source?**

AWS Blu Age Runtime is not open source. It's AWS IP distributed as a cloud-native service. There are two deployment options:

a. AWS Blu Age Managed, the runtime is deployed into a dedicated AWS managed service, taking advantage of an all-preconfigured and ready for deployment environment without setup nor Administration.

b. AWS Blu Age Non Managed, which can be deployed into your own bespoke AWS architecture based on Amazon EC2 or Amazon ECS/AWS Fargate, that you have to provision and setup by yourself. Both options incur runtime fees, which are included in the project estimates provided to you. As this is a managed service with Support access, you don't need the source code. For more details on pricing, see AWS Mainframe Modernization Pricing page.

42 **How are changes and upgrades to AWS Blu Age frameworks and libraries managed ?**

AWS Blu Age frameworks and libraries are updated through regular code generation and deployment processes. These updates are managed as part of the AWS Mainframe Modernization lifecycle, which includes version upgrades and support from the AWS Blu Age team or certified partners. For detailed information on versioning, upgrade processes, and support timelines, please refer to the AWS Mainframe Modernization lifecycle documentation.

Data

1. **Which database options are available for the modernized applications, regarding the modernization of the legacy database?**

   The modernized applications can use several modern database options including: PostgreSQL, Amazon Aurora, RDS for PostgreSQL, Oracle database, MS-SQL, and IBM Db2. These options provide flexibility in choosing the most appropriate database system based on specific requirements, while leveraging the benefits of modern database management systems and cloud-native features.

2. **What is the transformation coverage of IBM Db2 for z/OS to Postgres DDL?**

   Full transformation (including database constraints).

3. **Does AWS Blu Age support Group Data Generation (GDG)?**

   Yes, using GDG in batches is supported, with the support of relative and absolute generations and automatic clean-up strategies.

4. **Does AWS Blu Age support concatenated data sets?**

   Yes, using concatenated data sets in batches is supported. With concatenation in action, several data sets can be read as a single data set. Please note that the Blusam data sets cannot be part of a concatenation.

5. **What is the process applied on SQL queries?**

   Adjusted during code transformation, depending on the target database.

6. **Which options apply if there are multiple databases for an application?**

   Configure the target database for each query and define all the databases in the application and in Apache Tomcat.

7. **Can Blusam be disabled?**

   Yes, in the main configuration file, and no database is required (for more information, see Blusam configuration documentation page).

8. **Which AWS Blu Age API is used to replace databases such as IMS DB?**

   The JHDB (Java Hierarchical DataBase) API.

9. **Which AWS Blu Age product can be used to migrate legacy data and databases to a modern relational database management system (RDBMS)?**

   AWS Blu Age DB modernization Tool (Data Migrator).

10. **What is AWS Blu Age Data Simplifier and what problem does it solve in modernization?**

    Data Simplifier is a core library in AWS Blu Age that addresses the challenge of handling legacy memory access patterns in Java. It provides constructs to support low-level memory access, legacy data types (like zoned, packed, alphanumeric), and mixed structured/raw memory access that are common in mainframe applications but not natively available in Java. The library exposes these features through familiar Java patterns like getters/setters and class-based APIs, making them accessible to Java developers while maintaining legacy functionality.

11. **How does AWS Blu Age handles legacy memory layouts and data structures?**

AWS Blu Age handles legacy memory layouts through the Record interface, which provides an abstraction of byte arrays with fixed size. For structured data like COBOL '01 data items', it uses RecordEntity subclasses that are automatically generated during modernization. These classes maintain the hierarchical structure of the legacy data, with each element having a parent-child relationship. The system supports both raw memory access and structured access patterns, preserving the flexibility of legacy systems while providing a modern programming interface.

12. **How does AWS Blu Age deals with VSAM data sets modernization?**

    The Blusam component is providing the support for the modernization of the VSAM data sets, with a dedicated API, endpoints and an administration web-application (BAC: Blusam Administration Console). Blusam relies on a relational database as backend (PostgreSQL, either using RDS or Aurora).

Transformation

1. **Were can I found details about the transformation process?**

   See AWS Blu Insights documentation.

2. **What are the names of the AWS Blu Age generated modules?**

   Service, entities, web, and tools.

3. **Why was Java/Spring chosen as one of the target technologies for AWS Blu Age?**

   Java/Spring was chosen as a target technology because of its widespread adoption, large talent pool, and robust enterprise capabilities. The Java ecosystem offers extensive libraries, frameworks, and tools that support modern application development. Spring framework provides enterprise-grade features, cloud-native capabilities, and follows industry best practices, making it ideal for modernized applications.

4. **What is the name of the parent project that contains the AWS Blu Age generated modules?**

   The name of the parent project is suffixed by “-pom” and can be defined in the Transformation Center using the Transform property named project.

5. **How does AWS Blu Age manage legacy scheduler modernization, if provided?**

   Legacy scheduler assets are not being modernized by AWS Blu Age. They’re being taken into account during the assessment phase, to help identify possible missing artifacts.

6. **What is the requirement for debugging the generated code with AWS Blu Age?**

   Any integrated development environment (IDE) supporting Java, such as Eclipse, JetBrain, or VisualCode.

Deployment

1. **Which environments are available to deploy the modernized application with AWS Blu Age?**

   Windows Server, Linux server, and Docker Linux container.

2. **Can AWS Blu Age refactored applications run on any infrastructure?**

   While AWS Blu Age refactored applications are not designed to run on any infrastructure, they offer significant flexibility in deployment options. These applications can be deployed on various compute platforms, including cloud managed services, serverless compute, and on-premises infrastructure. AWS Blu Age provides both managed and non-managed runtime options, allowing organizations to choose between fully-managed convenience and customized control based on their specific needs and requirements. This flexibility enables easy movement across supported infrastructures, making AWS Blu Age refactored applications highly adaptable to different deployment environments. For more details, see AWS Blu Age Runtime options documentation.

3. **Which MQ configuration does AWS Blu Age support?**

   SQS, IBM WebSphere MQ.

4. **Into which application servers can a user deploy Java business application logic with AWS Mainframe Modernization non-managed runtime?**

   Apache Tomcat, version greater or equal to 10.1.

5. **How does the refactored application integrate with other AWS services like Amazon Aurora?**

   The modernized application integrates with AWS services by supporting transformation to cloud-native database solutions like Amazon Aurora and RDS for PostgreSQL. AWS Blu Age ensures integration between modernized applications and AWS services, enabling organizations to use cloud capabilities. This integration extends to both data storage and application services within the AWS ecosystem. Beyond database storage, AWS Blu Age Runtime integrates with various AWS services including Amazon ElastiCache for Redis caching, AWS Secrets Manager for configuration management, and AWS Mainframe Modernization for deployment. It supports

Amazon EC2, Amazon EKS, ECS managed by Fargate for container deployment. The system can utilize AWS Identity and Access Management for authentication, Amazon Simple Storage Service for storage, and supports integration with other AWS services through configuration and service connectors.

6. **How does the refactored application ensure scalability requirements are met?**

   The solution ensures scalability by transforming applications into cloud-native architectures that can use the AWS elastic infrastructure. It implements modern design patterns and best practices that enable horizontal and vertical scaling. The service-oriented approach allows for independent scaling of components. The modernized applications can take advantage of cloud services' inherent scalability features.

7. **What happens after the source code refactoring is completed?**

   After source code refactoring, two main steps occur. First, the refactored application is built. Second, the application is deployed and monitored in AWS Mainframe Modernization AWS Blu Age Runtime. The deployment can be done either in an AWS-managed environment (AWS Mainframe Modernization managed runtime) where infrastructure is managed with automation, or in your AWS account (AWS Mainframe Modernization AWS Blu Age non-managed Runtime) where customers manage their own infrastructure. The non-managed option can be deployed on various platforms, including Amazon EC2, ECS on EC2 or on Fargate, EKS on EC2.

8. **How can I deploy and run an application modernized with AWS Blu Age on a custom Amazon Linux AMI, without using the AWS Mainframe Modernization (M2) managed service?**

   This can be achieved by deploying the application using AWS Blu Age Runtime (non-managed) on Amazon EC2. The process involves creating a Java/Spring application with a dependency on the AWS Blu Age Runtime library and deploying it on a custom Amazon Linux AMI. For detailed instructions on this approach, see Set up AWS Blu Age Runtime (non-managed) on Amazon EC2.

9. **Is there an Amazon Machine Image (AMI) available? Is there a Docker image available?**

   - **AMI**: No, due to customers' needs to customize and set up their environment as they prefer, there is no AMI available. Customers can retrieve the AWS Blu Age artifacts and set up their instance according to their requirements.
   - **Docker Image**: No, there is no available docker image, out of the box, but the Set up AWS Blu Age Runtime on container page explains how to build and deploy your own docker image based on AWS Blu Age Runtime binaries, to a suitable container management system.

10. **Can customer package and run an AWS Blu Age application as a Docker container?**

It is not possible for M2 Managed Runtime but it is for customer defined environment based on an Amazon Linux AMI and for on-premises or other cloud providers.

11 **How can I know the resource ARN of the SQS policy required to run AWS Blu Age non-managed if I want to scope it down?**

To determine the specific SQS policy resource ARN for running AWS Blu Age non-managed with a scoped-down policy, please consult the delivery team or Technical Account Manager (TAM). They can provide account-specific guidance. For general information on SQS policies, refer to the AWS SQS Policy documentation.

12 **How does job scheduling work with batch?**

It is integrated with Control-M /Stone branch or with any other Distributed scheduler.

Replatforming applications with Rocket Software (formerly Micro Focus)

This guide covers the end-to-end process of replatforming mainframe applications using AWS Mainframe Modernization solutions on AWS. It describes all tasks and includes information on configuring and operating AWS Mainframe Modernization runtime on Amazon EC2 from initial setup and analysis to building, testing, and deploying your modernized applications on AWS. It also covers advanced topics like working with legacy data structures, using templates and predefined projects, and setting up automation for streaming sessions.

Topics

- Set up Rocket Software (formerly Micro Focus) (on Amazon EC2)
- Set up Automation for Rocket Enterprise Analyzer (formerly Micro Focus) and Rocket Enterprise Developer Streaming Sessions
- View data sets as tables and columns in Rocket Enterprise Developer (formerly Micro Focus Enterprise Developer)
- Edit data sets using Rocket Software (formerly Micro Focus) Data File Tools in Enterprise Developer
- Tutorials for Rocket Software (formerly Micro Focus)
- Available batch utilities in AWS Mainframe Modernization

Set up Rocket Software (formerly Micro Focus) (on Amazon EC2)

AWS Mainframe Modernization provides several Amazon Machine Images (AMIs) that include Rocket Software (formerly Micro Focus) licensed products. These AMIs allow you to quickly provision Amazon Elastic Compute Cloud (Amazon EC2) instances to support Rocket Software 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 for setting up Rocket Software (formerly Micro Focus) (on Amazon EC2)

- Create the Amazon VPC endpoint for Amazon S3
- Request the allowlist update for the account
- Create the AWS Identity and Access Management role
- Grant License Manager the required permissions
- Subscribe to the Amazon Machine Images
- Launch an AWS Mainframe Modernization Rocket Software (formerly Micro Focus) instance
- Subnet or VPC with no internet access

Prerequisites for setting up Rocket Software (formerly Micro Focus) (on Amazon EC2)

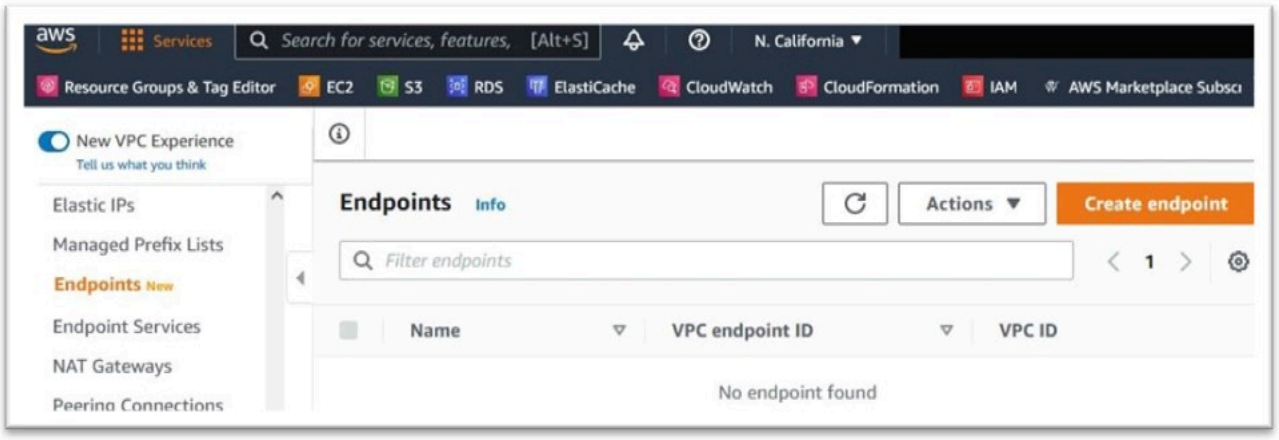
When you set up Rocket Software (on Amazon EC2), make sure you meet the following prerequisites.

- 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.

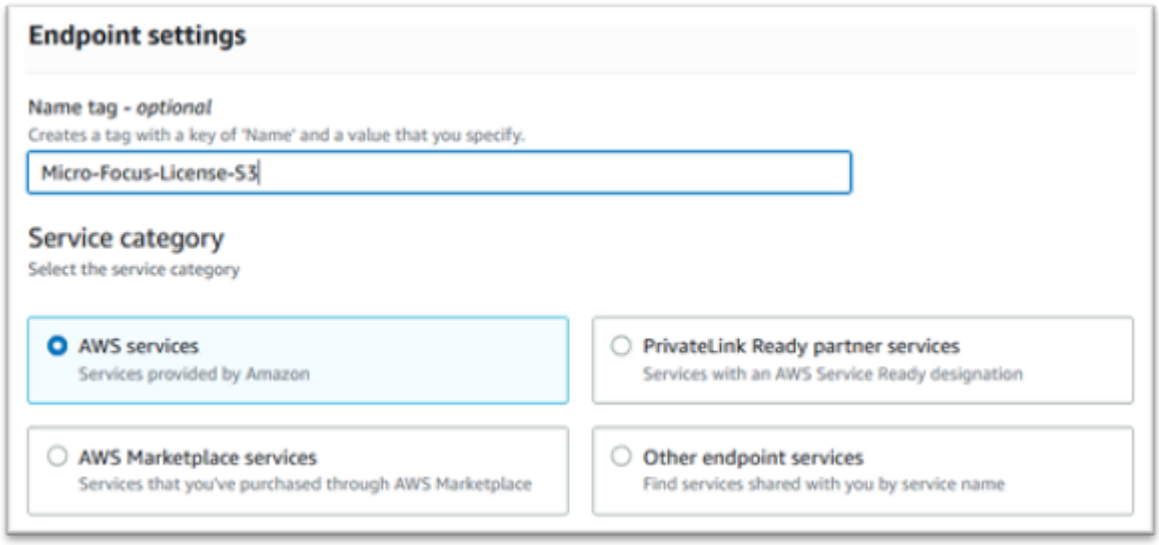
Create the Amazon VPC endpoint for Amazon S3

In this section, you create a Amazon VPC endpoint for Amazon S3 to use. Setting up this endpoint will help you later when setting up internet access for VPC.

1. Navigate to Amazon VPC in the AWS Management Console.
2. In the navigation pane, choose **Endpoints**.
3. Choose **Create endpoint**.

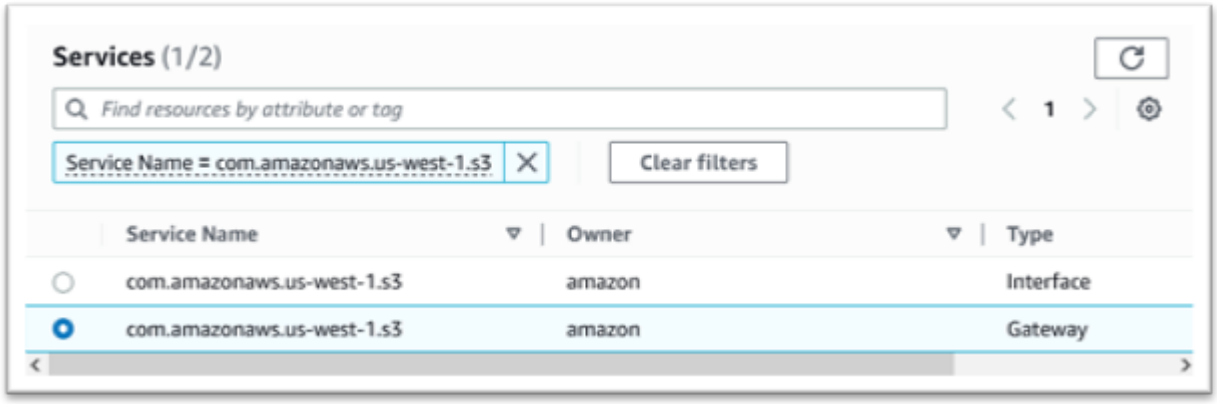


4. Enter a meaningful name tag, for example: “Micro-Focus-License-S3”.
5. Choose **AWS Services** as the Service Category.

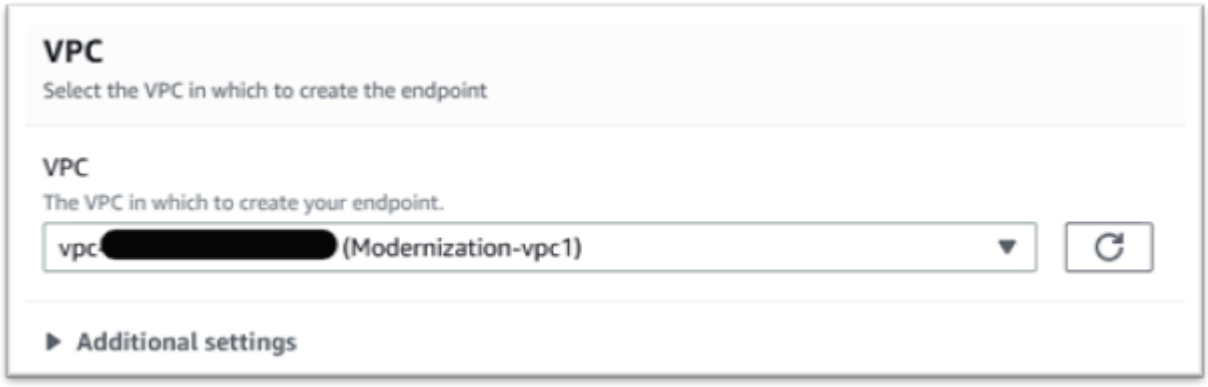


6. Under **Services** search for the Amazon S3 Gateway service: **com.amazonaws.[region].s3**.

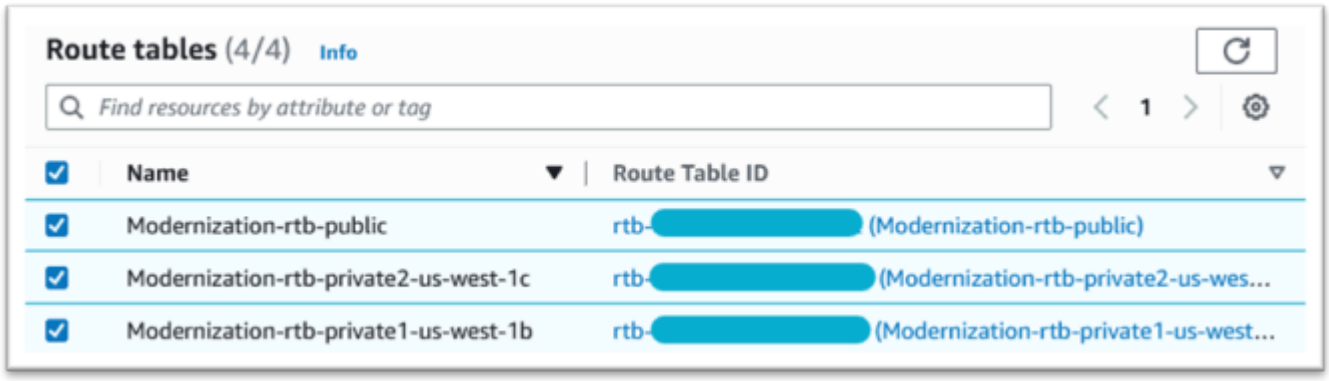
   For `us-west-1` this would be: `com.amazonaws.us-west-1.s3`.
7. Choose the **Gateway** service.



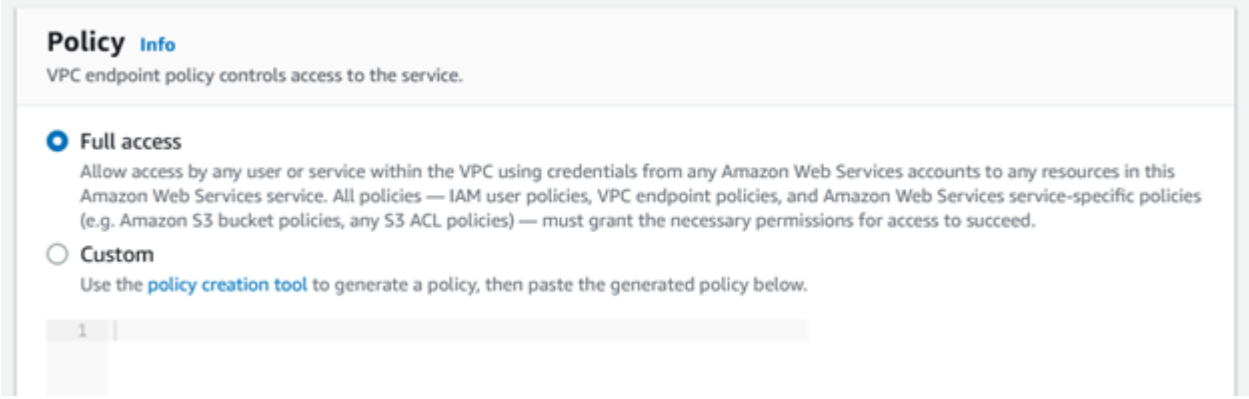
8. For VPC choose the VPC you will be using.



9. Choose all of the route tables for the VPC.



10. Under **Policy** choose **Full Access**.



> **Note**
>
> If you decide to create a custom policy, make sure it has access to the Amazon S3 bucket `s3://aws-supernova-marketplace-<region>-prod`.

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. This is the `vpce-xxxxxxxxxxxxxxxxx` id for the **com.amazonaws.[region].s3 Gateway** endpoint.
- The number of licenses required across all Rocket Software 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**
>
> Reach out to your AWS representative or AWS Support who will open the support ticket for the Allowlist request on your behalf. It can't be requested directly by you and the request may take several days to complete.

Create 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 Rocket Software 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.

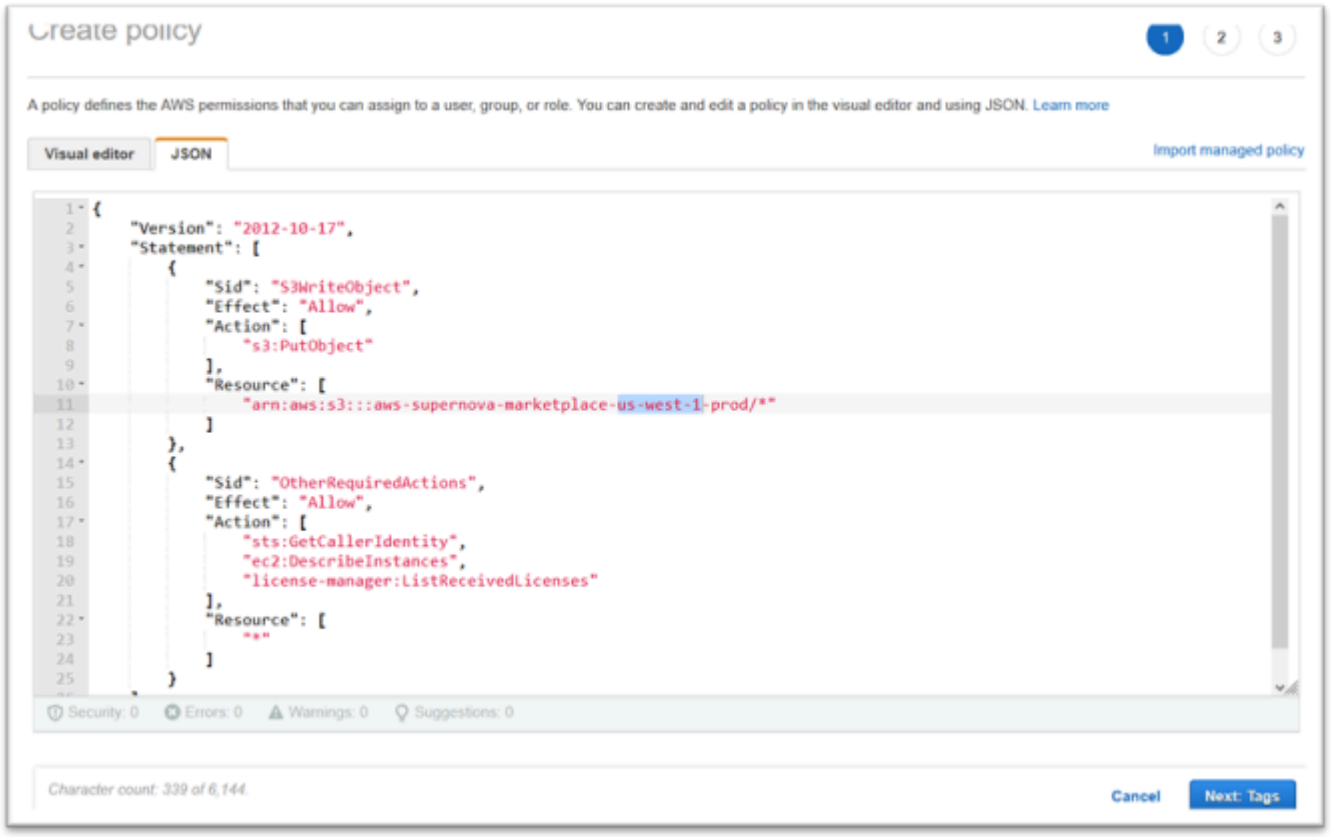


4. Replace `us-west-1` in the following JSON with the AWS Region where the Amazon S3 endpoint was defined, then copy and paste the JSON into the policy editor.

```
{
    "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": [
                "*"
            ]
        }
    ]
```

```
}
```

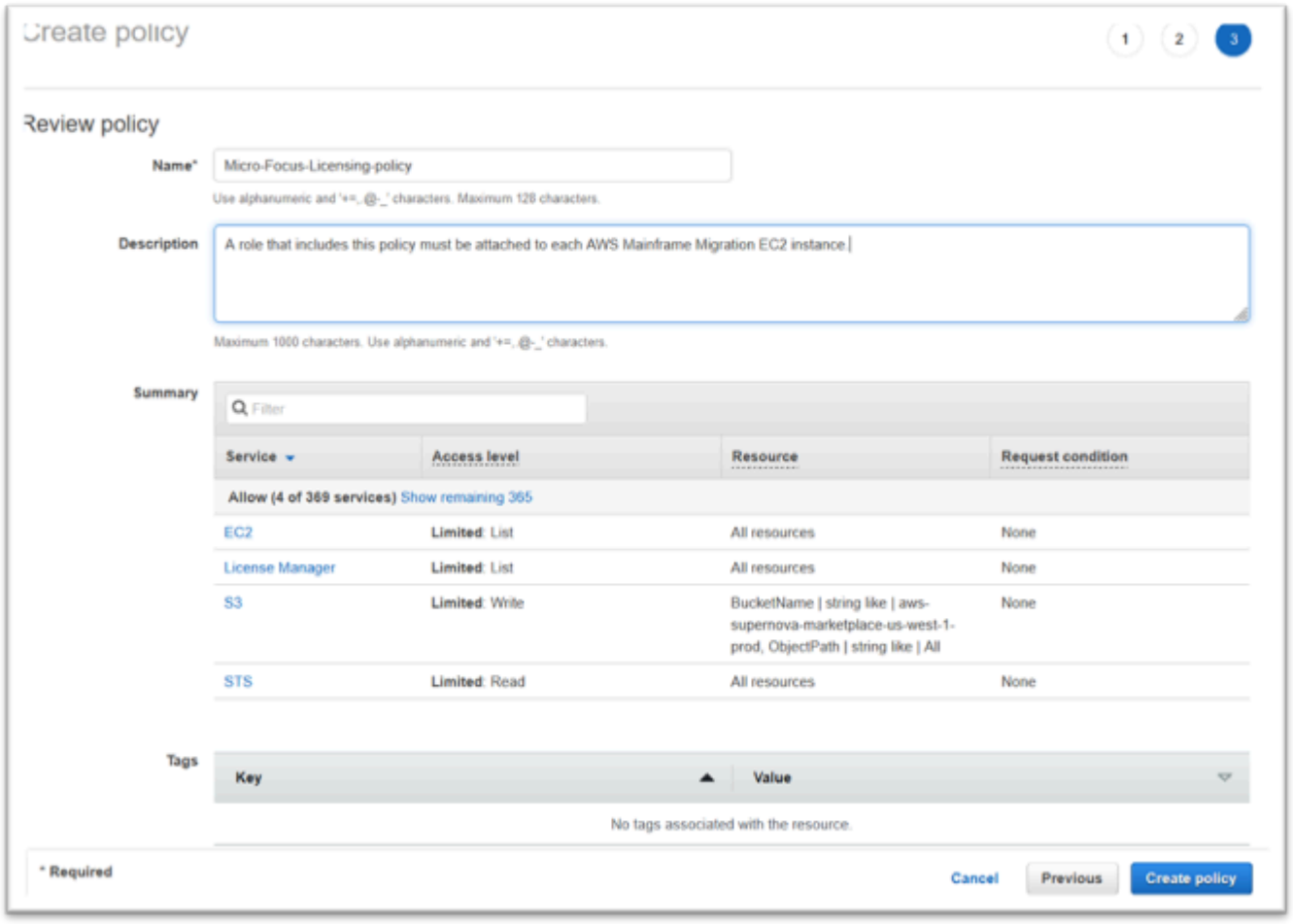
> **Note**
>
> The Actions under the Sid `OtherRequiredActions` do not support resource-level permissions and must specify `*` in the resource element.



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.”

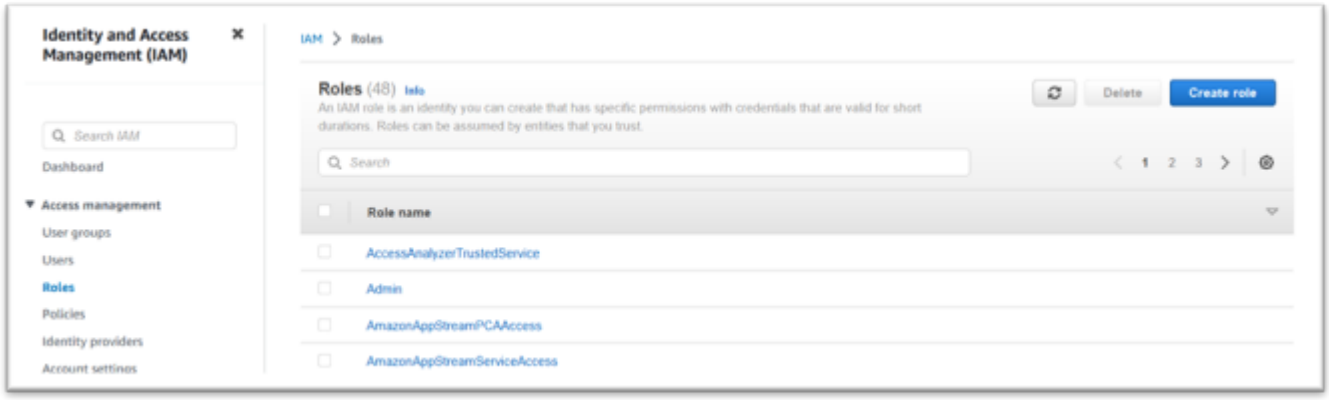


8. Choose **Create Policy**.

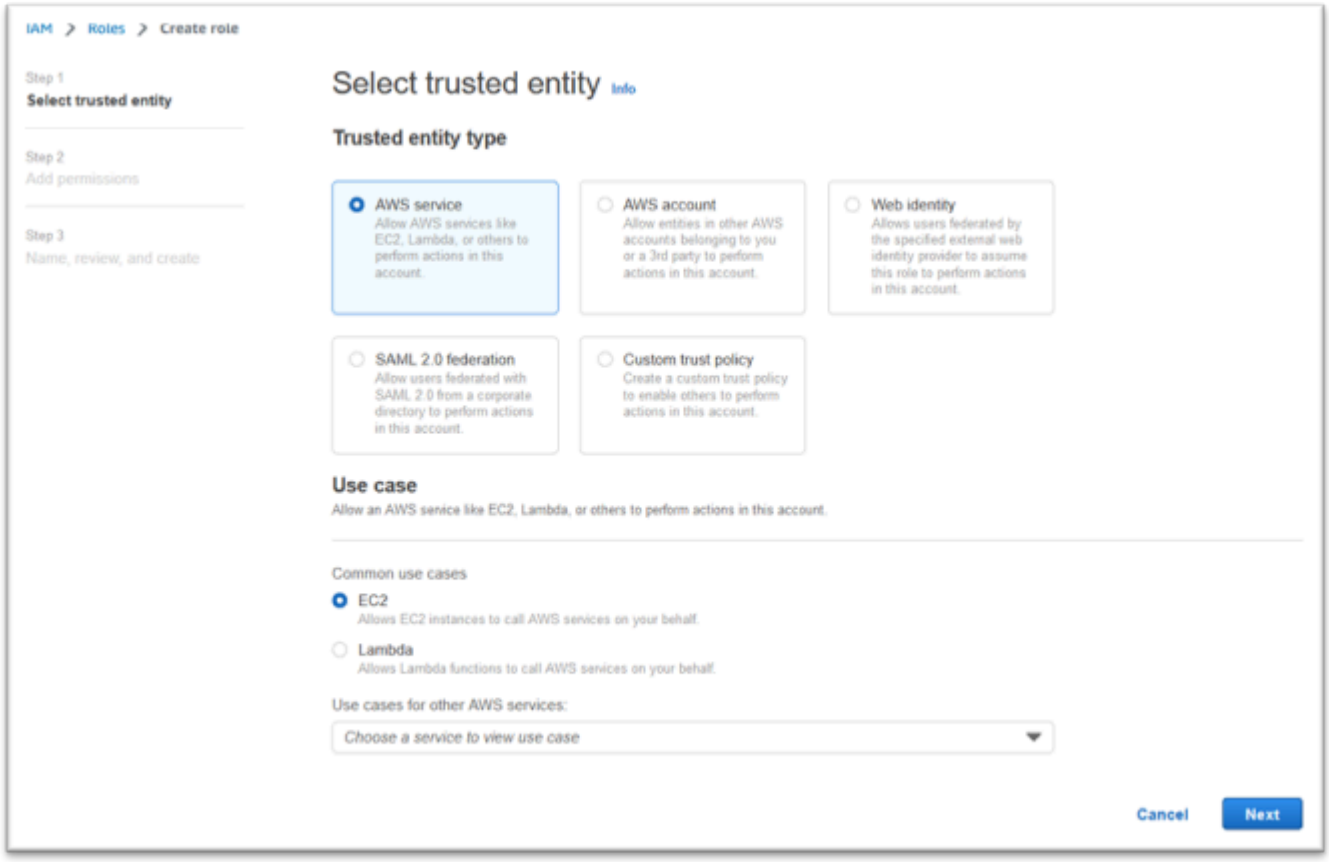
Create the IAM role

After creating an IAM policy, you create an IAM role and attach it to the policy.

1. Navigate to IAM in the AWS Management Console.
2. Choose **Roles** and then **Create Role**.

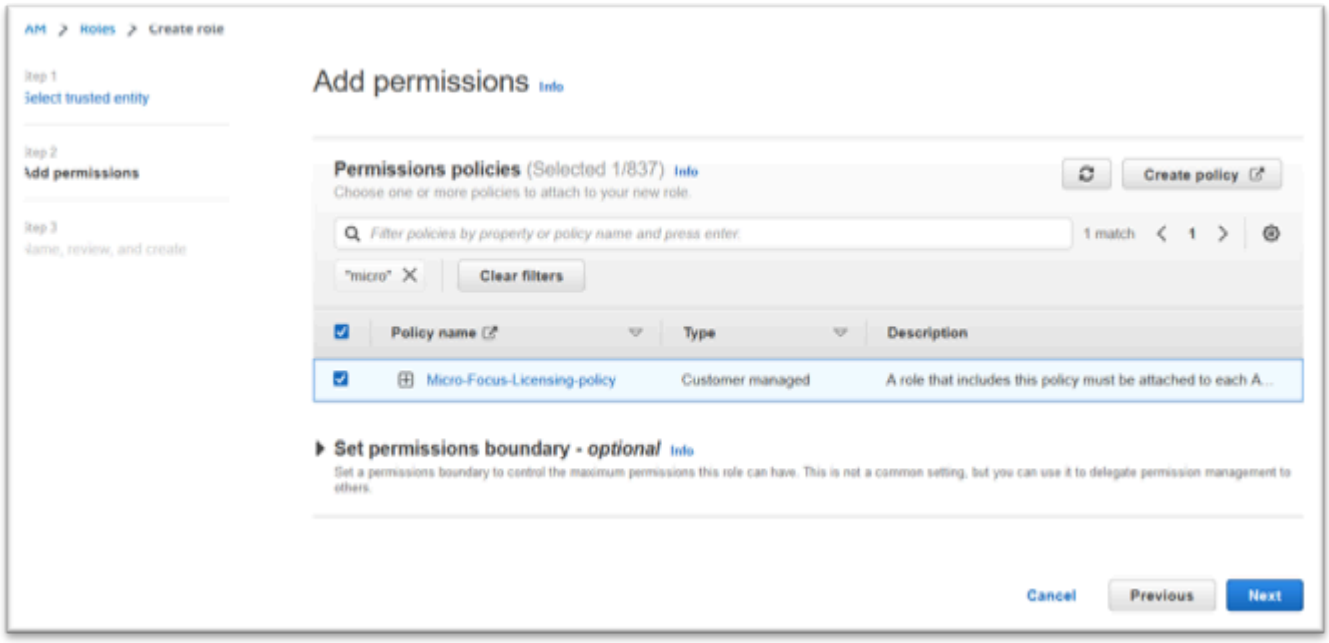


3. Leave **Trusted entity type** as **AWS service** and choose the **EC2** common use case.

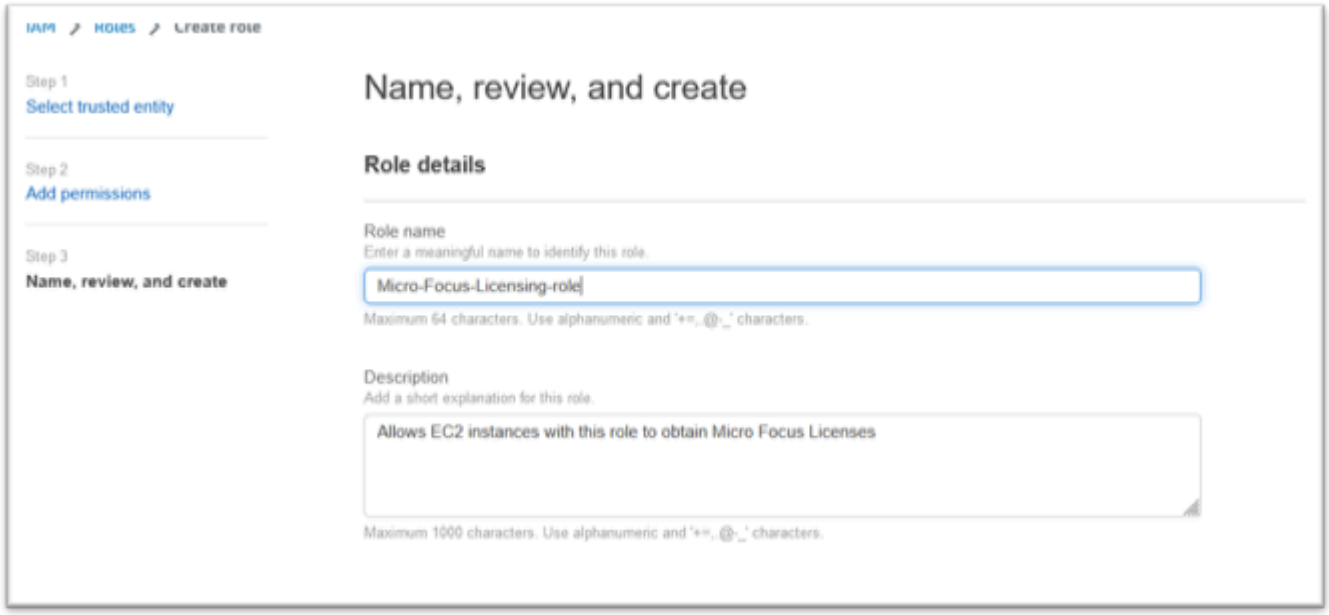


4. Choose **Next**.

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”.



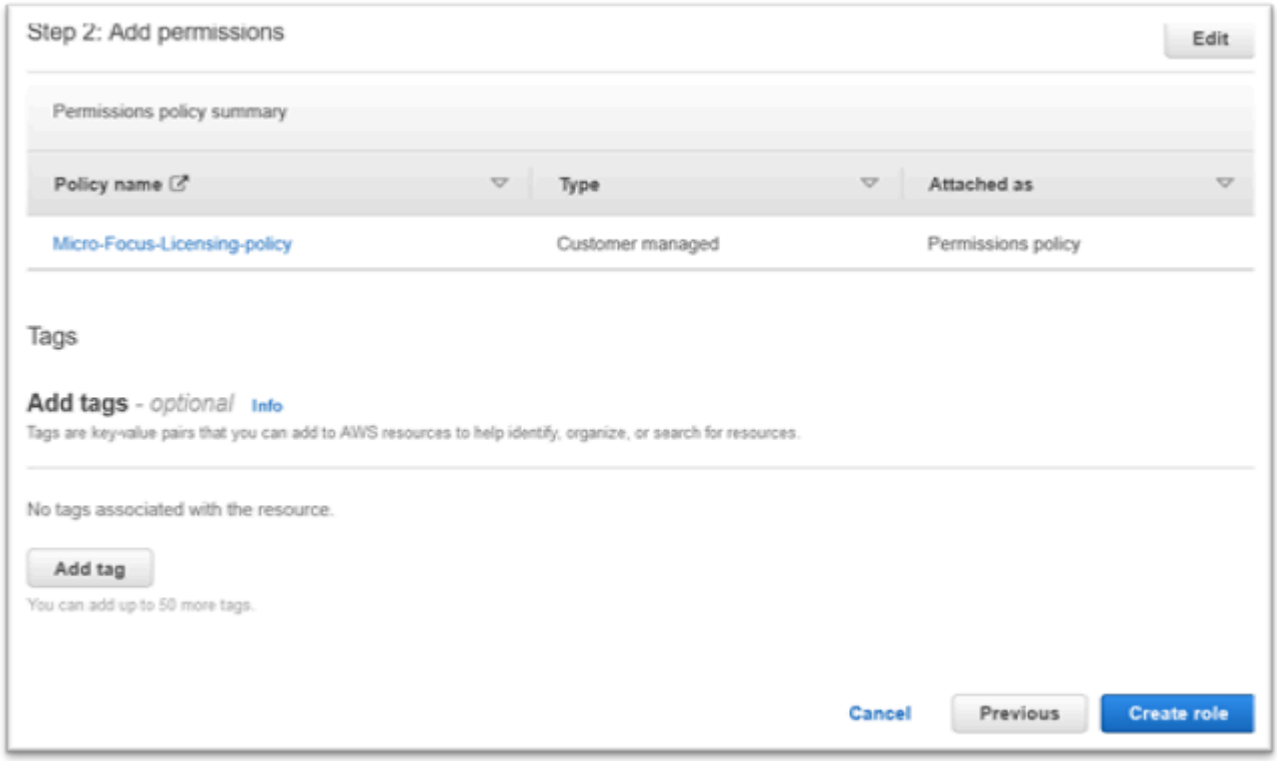
10. Under **Step 1: Select trusted entities** review the JSON and confirm it has the following values:

```
{
    "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.



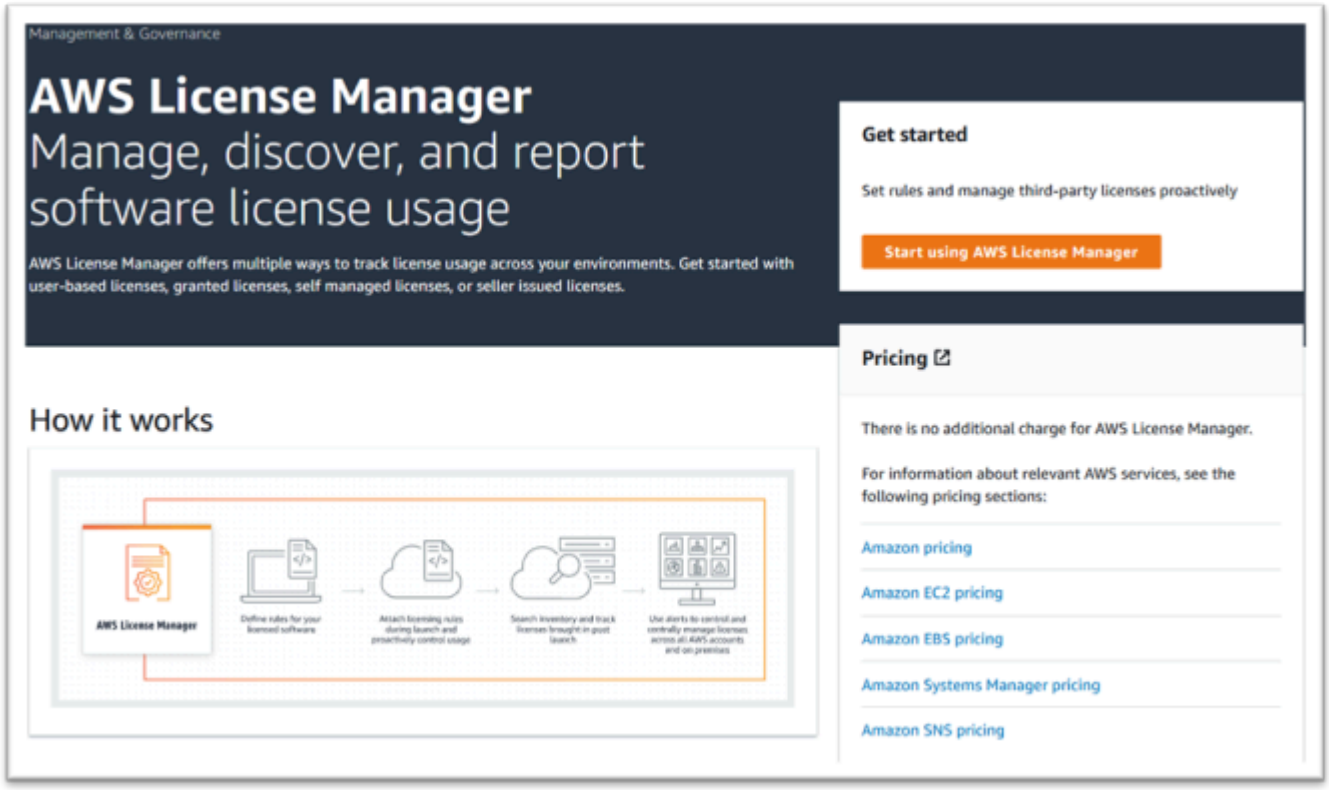
12. Choose **Create role**.

After the allowlist request is complete, continue with the following steps.

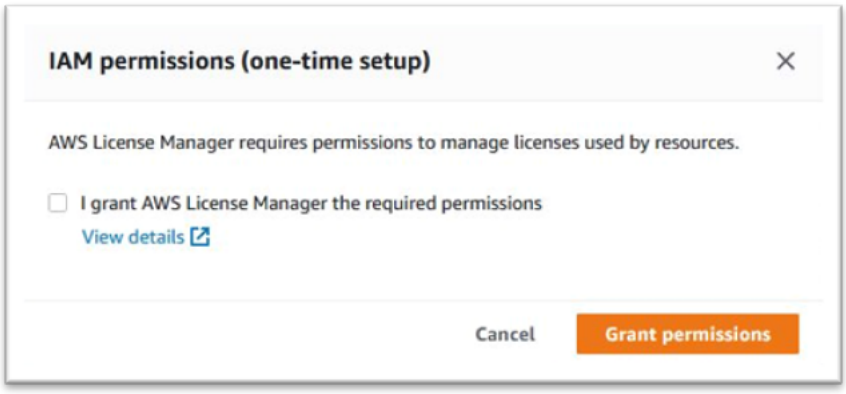
Grant License Manager the required permissions

You need to grant permissions to your AWS License Manager to set up Rocket Software runtime engine (on Amazon EC2).

1. Navigate to AWS License Manager in the AWS Management Console.



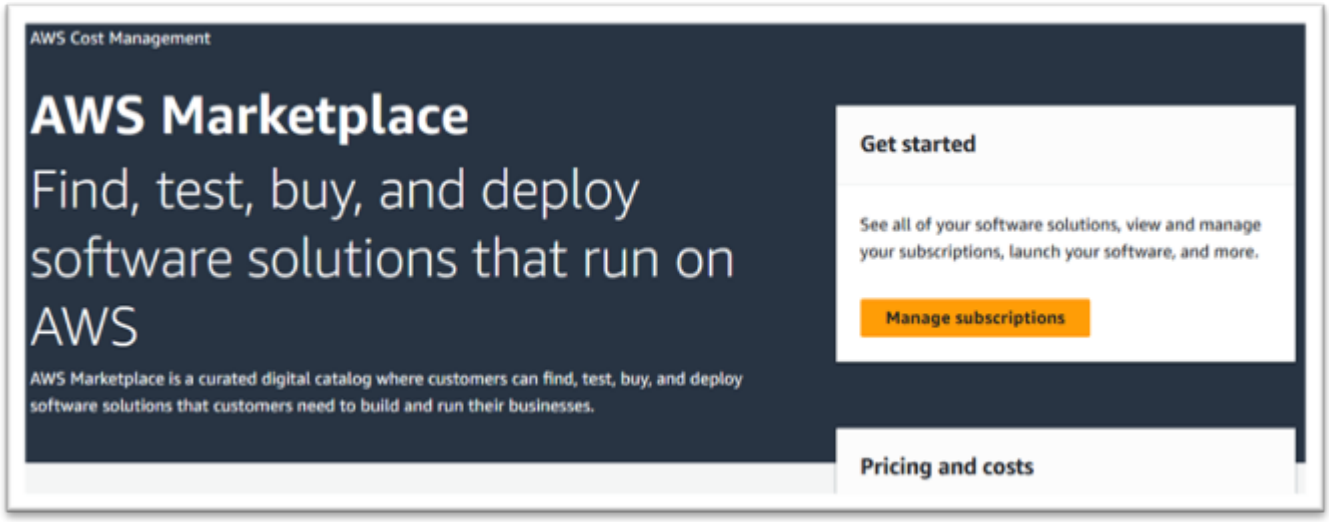
2. Choose **Start using AWS License Manager**.
3. If you see the following pop-up, view the details, then choose the check-box and press **Grant Permissions**.



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. You can also manage your subscribed AMIs when setting up Rocket Software (formerly Micro Focus) runtime engine (on Amazon EC2).

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**
>
> 1. Only choose a link for one of the products you have been authorized to use.
> 2. Make sure your account is allowlisted by following the Request the allowlist update for the account page to use these links.

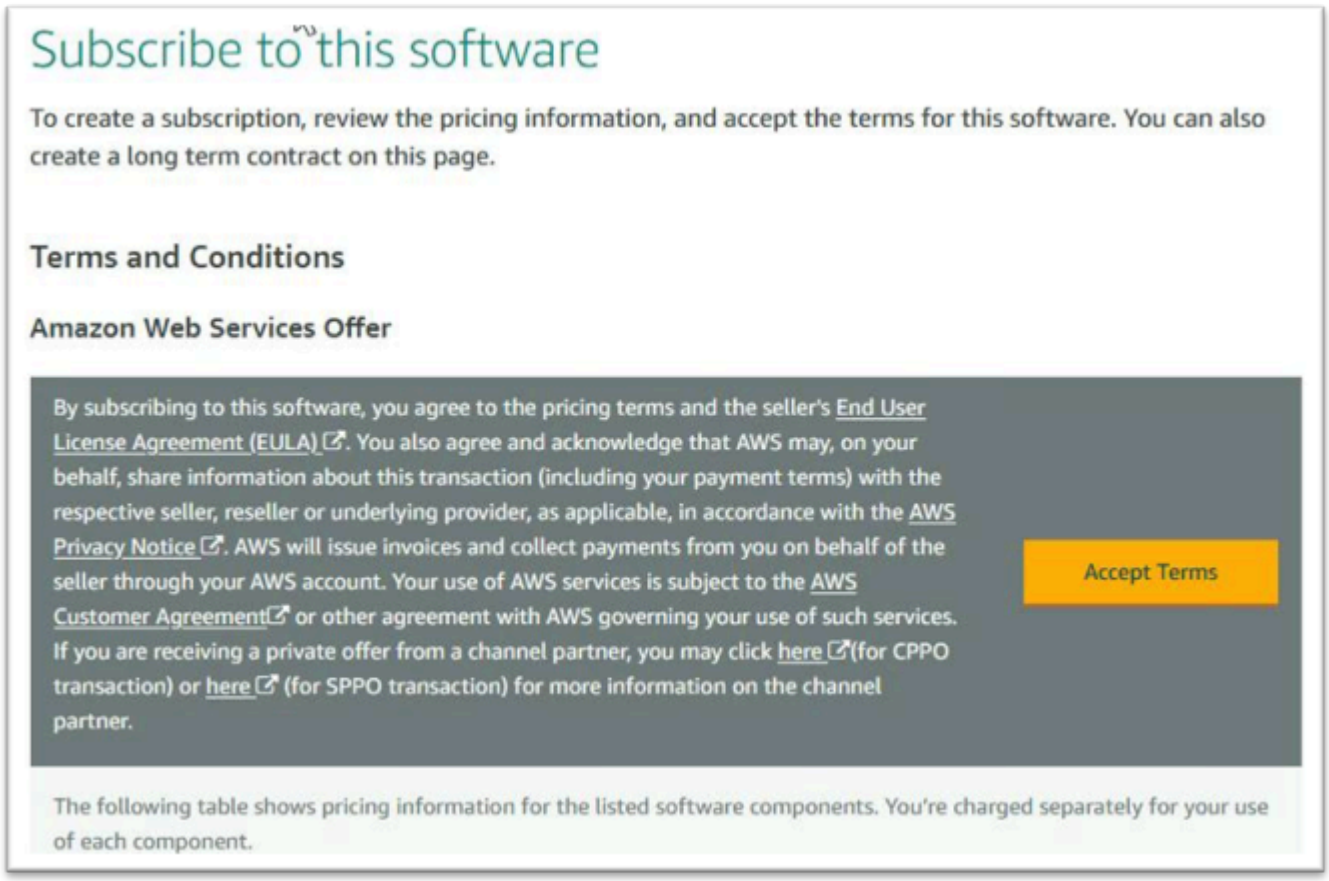
- Enterprise Server: https://aws.amazon.com/marketplace/pp/prodview-g5emev63l7blc
- Enterprise Server for Windows: https://aws.amazon.com/marketplace/pp/prodview-lwybsiyikbhc2
- Enterprise Developer: https://aws.amazon.com/marketplace/pp/prodview-77qmpr42yzxwk
- Enterprise Developer with Visual Studio 2022: https://aws.amazon.com/marketplace/pp/prodview-m4l3lqiszo6cm
- Enterprise Analyzer: https://aws.amazon.com/marketplace/pp/prodview-tttheylcmcihm
- Enterprise Build Tools for Windows: https://aws.amazon.com/marketplace/pp/prodview-2rw35bbt6uozi
- Enterprise Stored Procedures: https://aws.amazon.com/marketplace/pp/prodview-zoeyqnsdsj6ha

- Enterprise Stored Procedures with SQL Server 2019: https://aws.amazon.com/marketplace/pp/prodview-ynfklquwubnz4

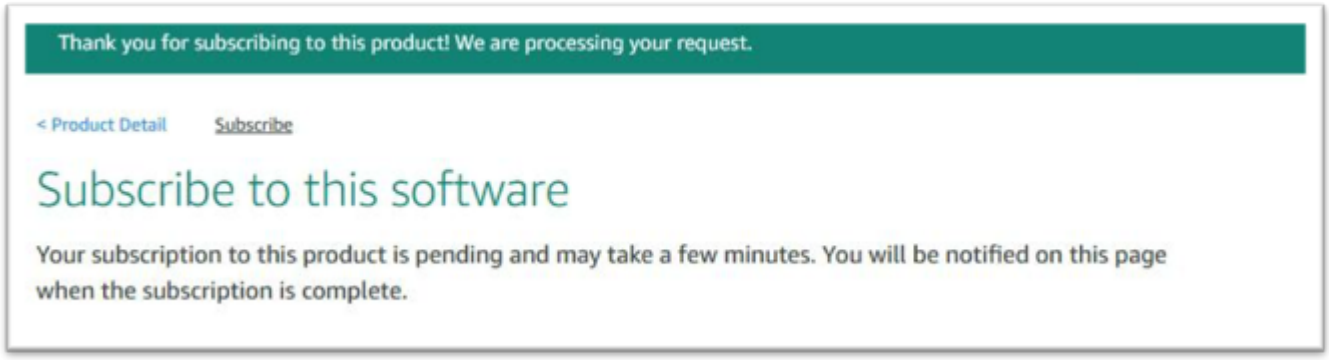
4. Choose **Continue to Subscribe**.



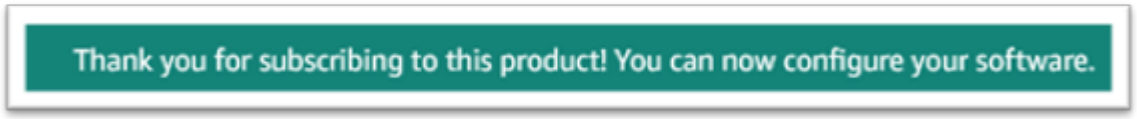
5. If the Terms and Conditions are acceptable, choose **Accept Terms**.



6. The subscription might take a few minutes to process.

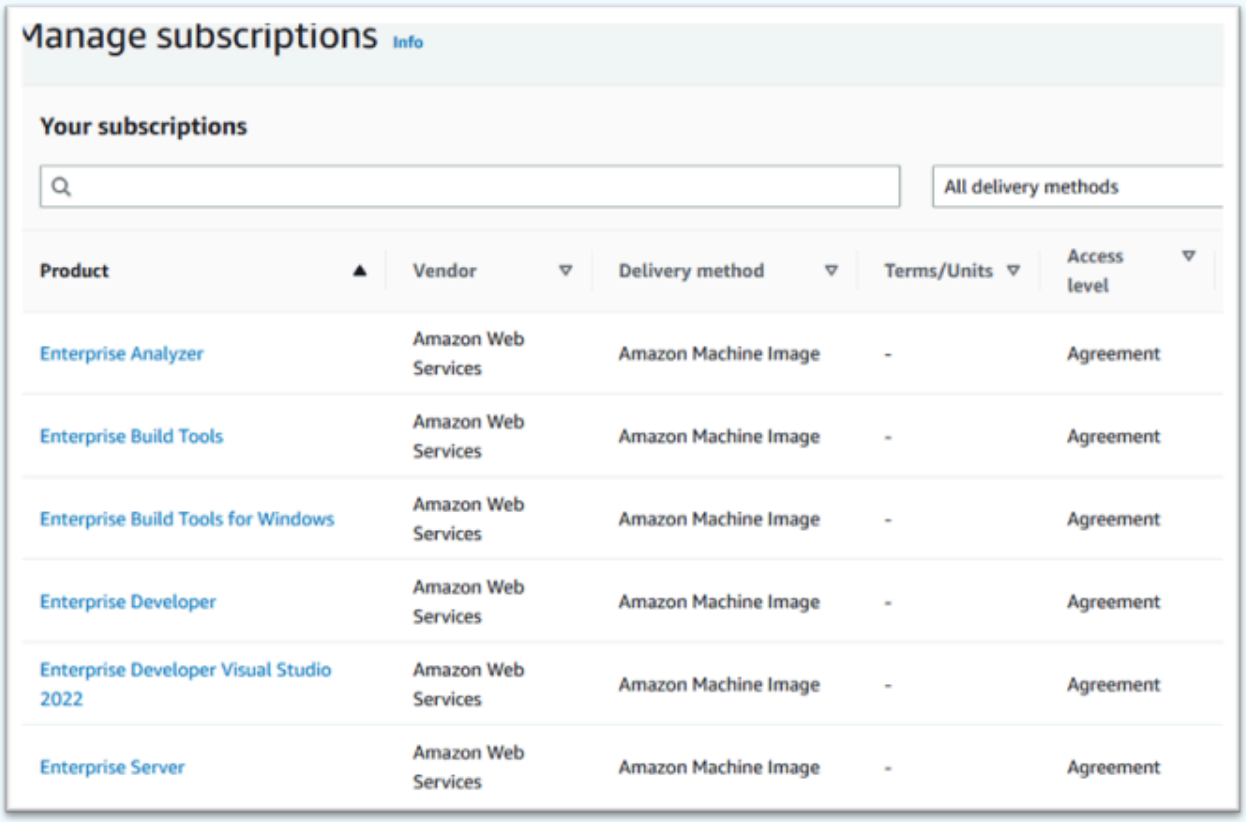


7. After the Thank you message shows, copy and paste the next link from step 3 to continue adding subscriptions.



8. Stop when **Manage subscriptions** shows all your subscribed AMIs.

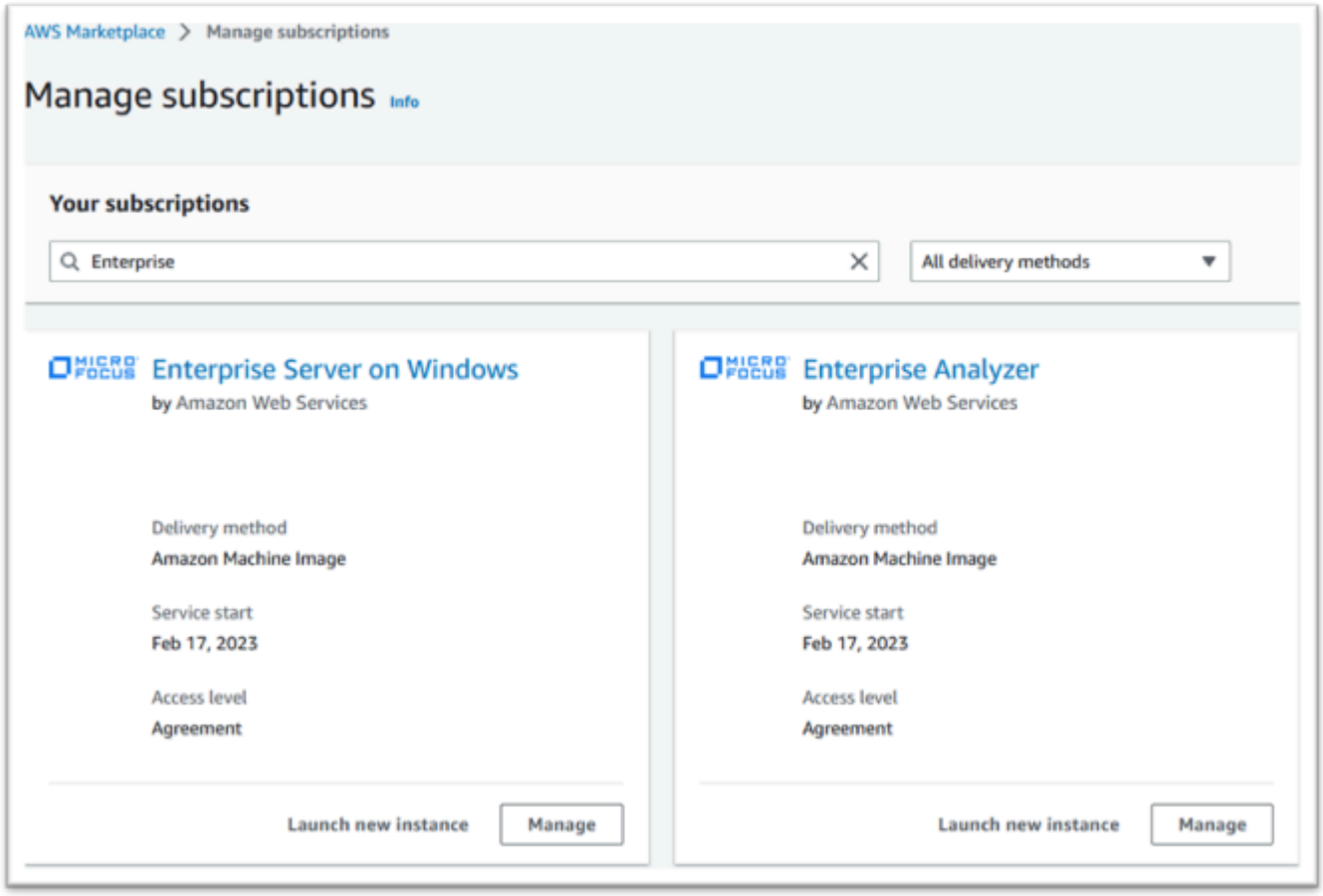
> **Note**
>
> The panel preferences (gear icon) are set to show the View as a Table.



Launch an AWS Mainframe Modernization Rocket Software (formerly Micro Focus) instance

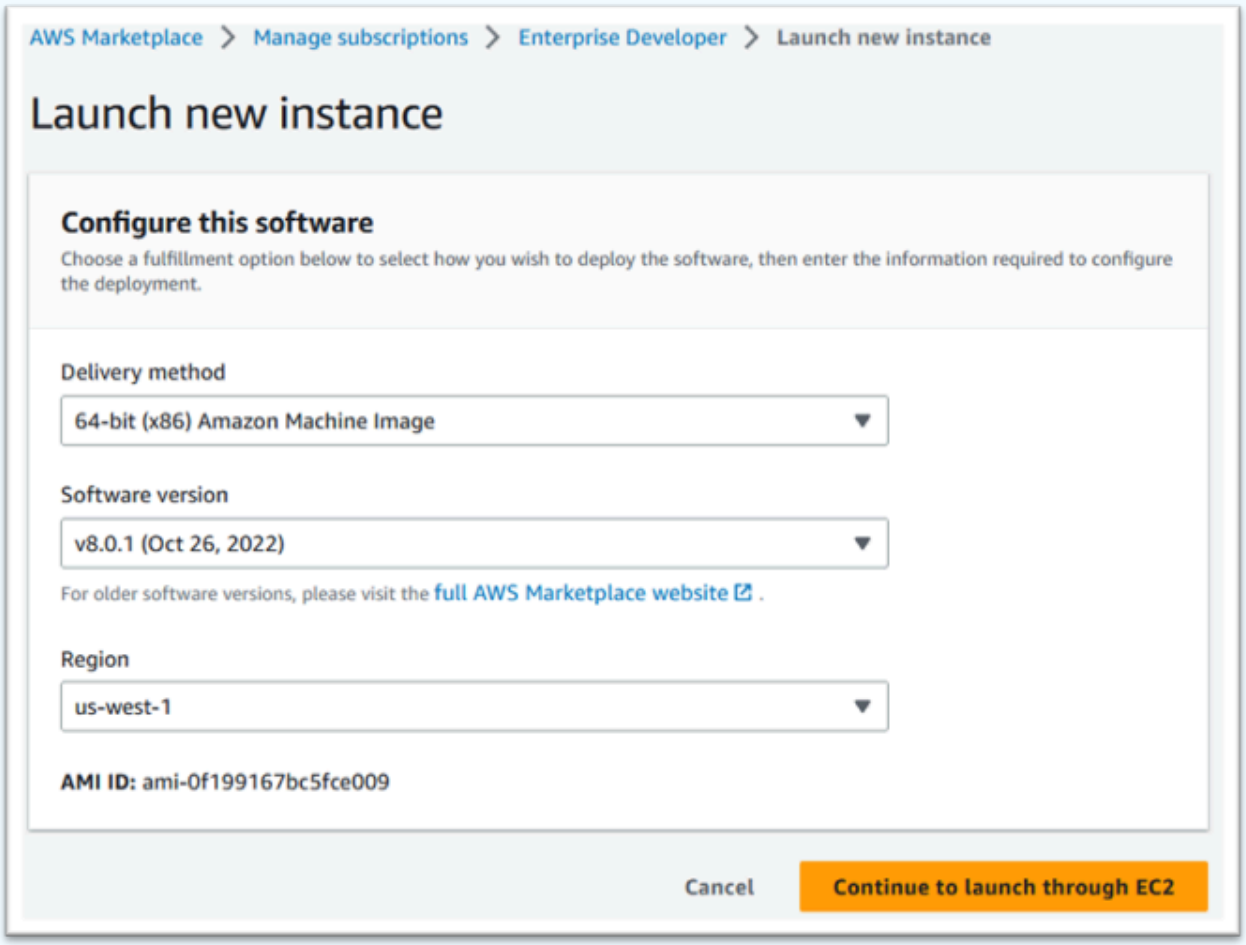
After creating endpoints, IAM policy, IAM role, and subscribing to AMIs, you are ready to launch an AWS Mainframe Modernization Rocket Software (Micro Focus) instance in the AWS Management Console.

1. Navigate to AWS Marketplace Subscriptions in the AWS Management Console.
2. Locate the AMI to be launched and choose **Launch New Instance**.



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.

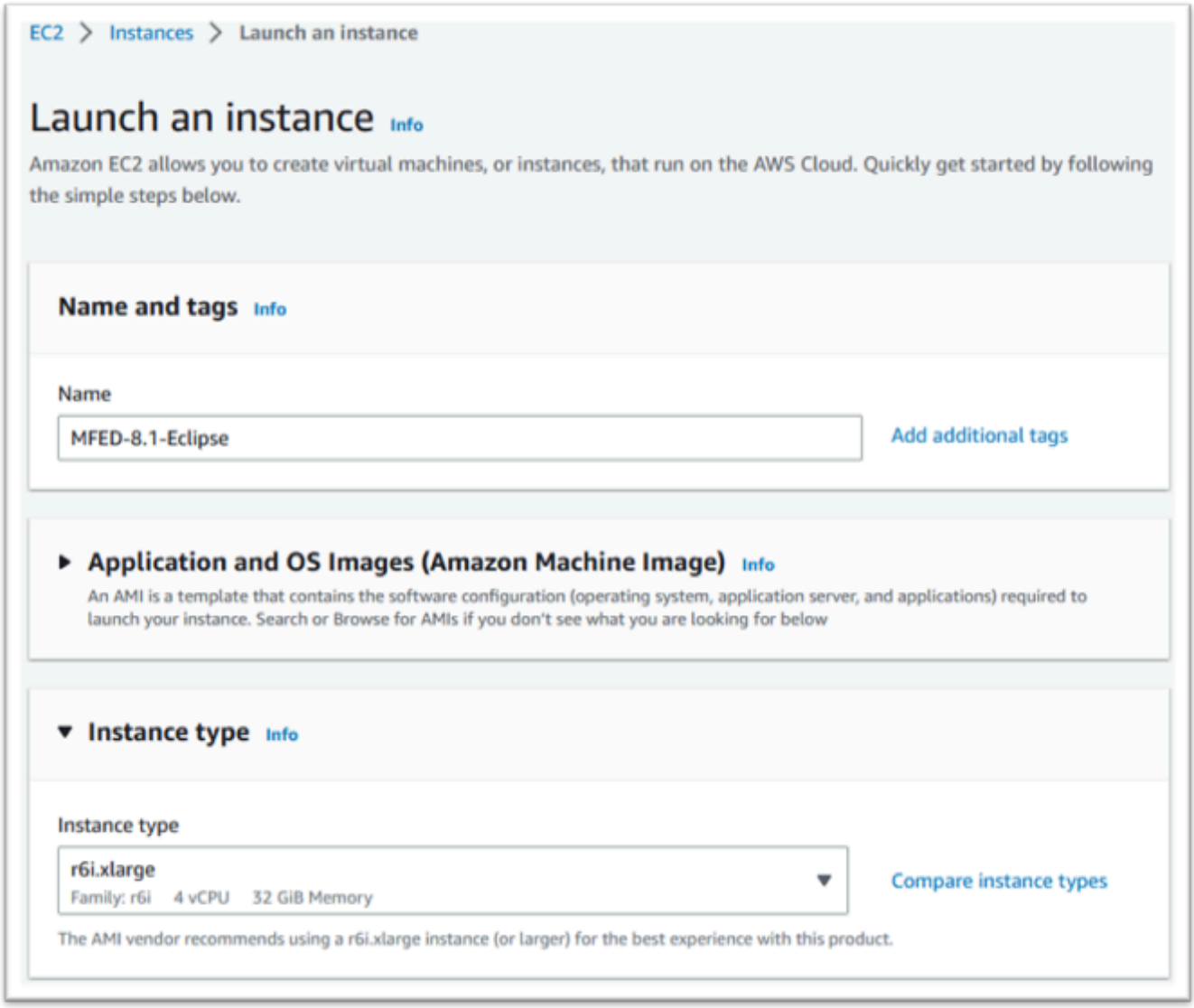


5. Enter a name for the server.
6. Choose an instance type.

   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 Rocket Software 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).

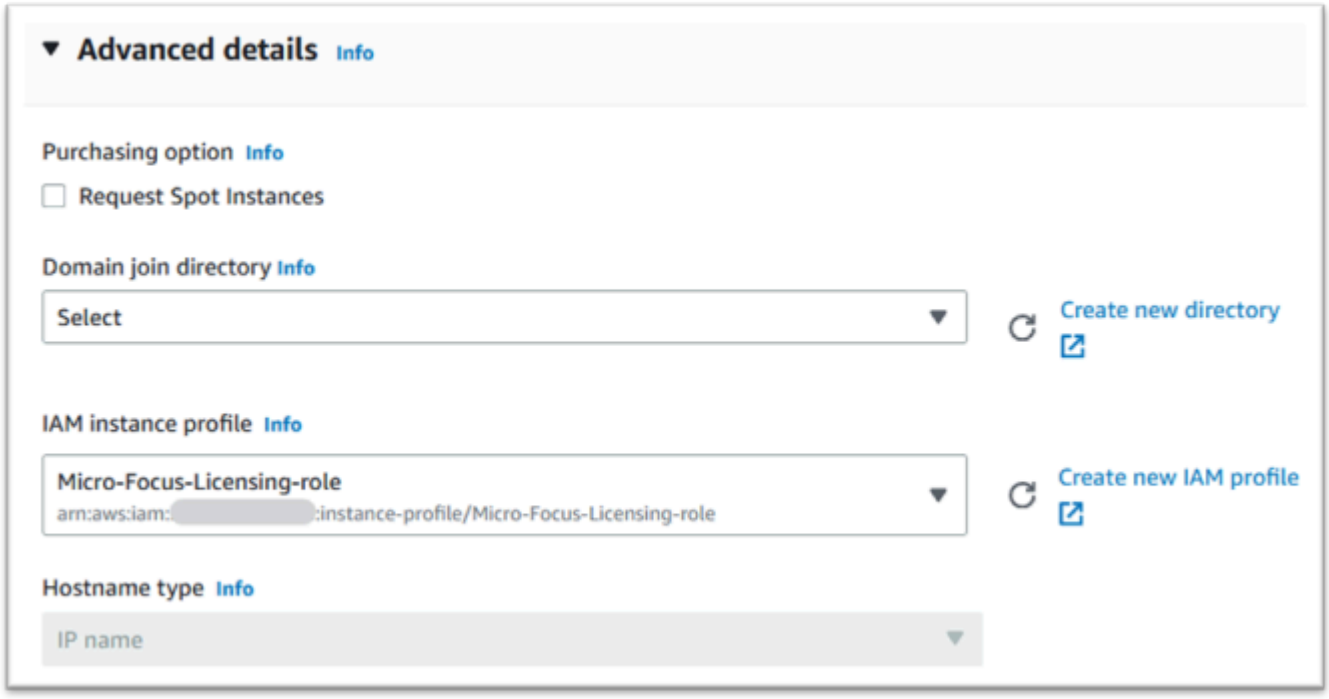
   For more information on key pairs for Linux instances, see Amazon EC2 key pairs and Linux instances.

   For more information on key pairs for Windows instances, see Amazon EC2 key pairs and Windows instances.

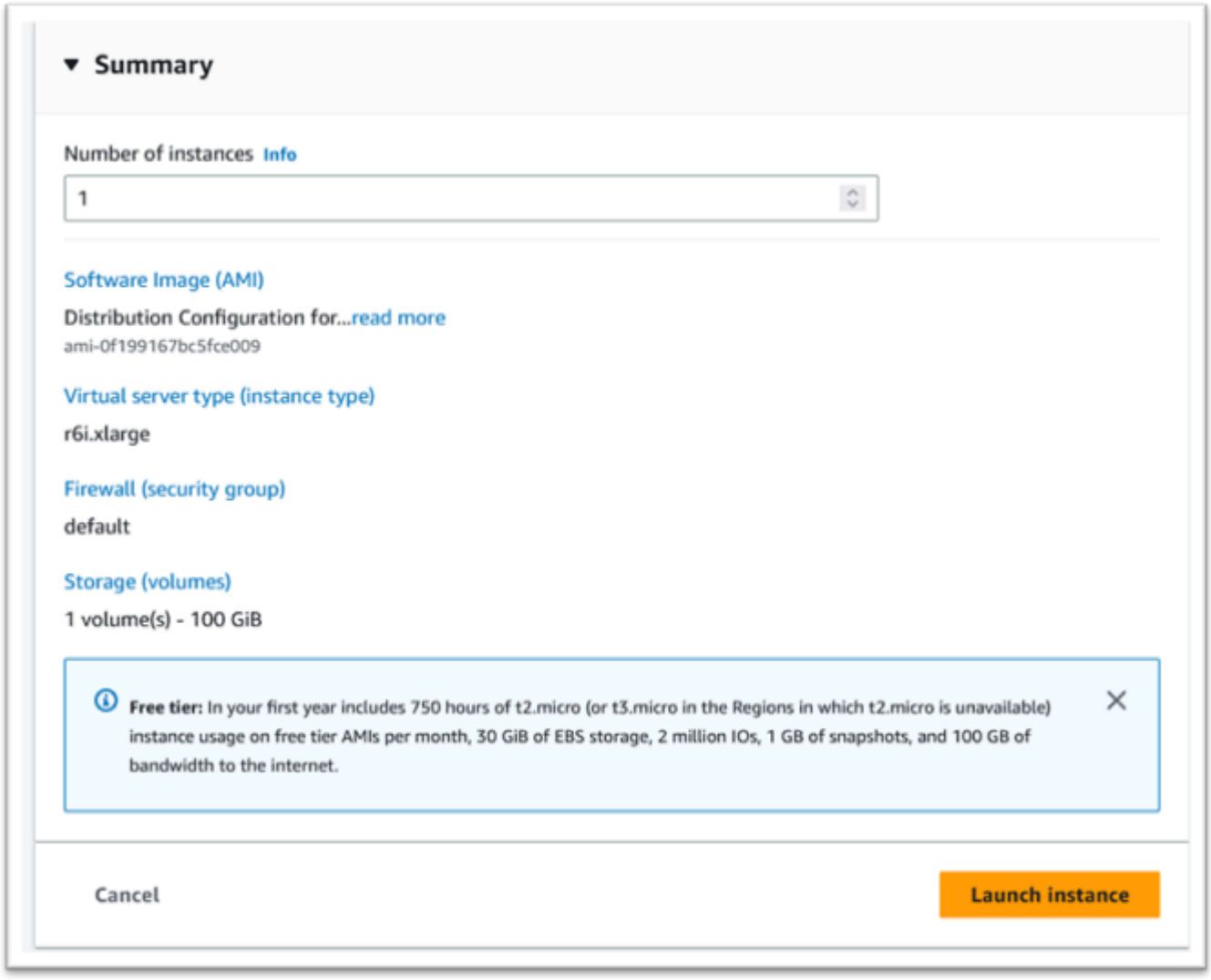
8. Edit the Network settings and **choose the allowlisted VPC** and appropriate Subnet.
9. **Choose or create a Security Group**. If this is an Enterprise Server EC2 instance it is typical to allow TCP traffic to ports 86 and 10086 to administer the Rocket Software configuration.
10. Optionally configure 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.

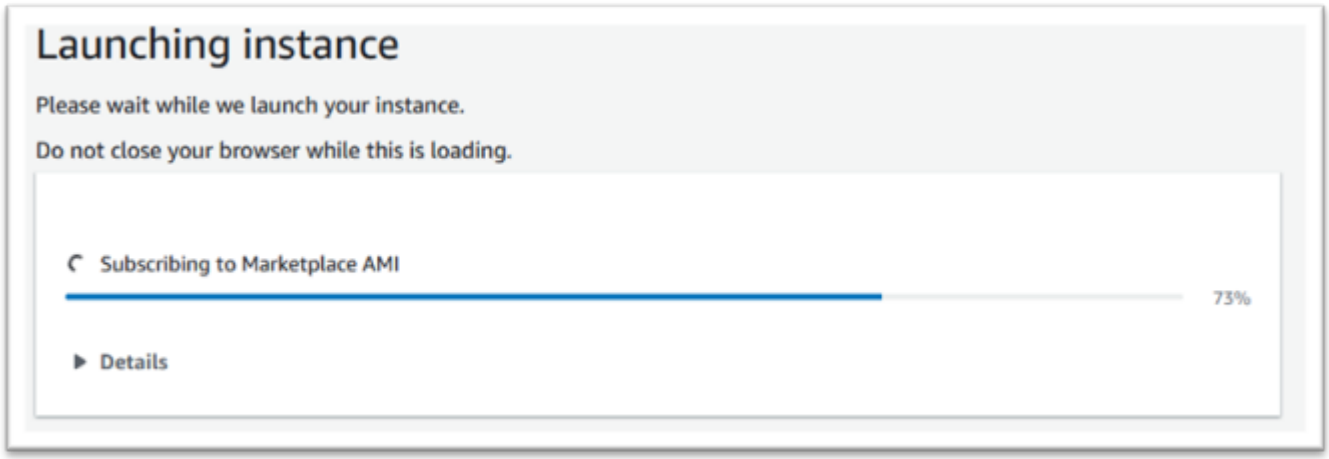


12. Review the Summary and push **Launch Instance**.

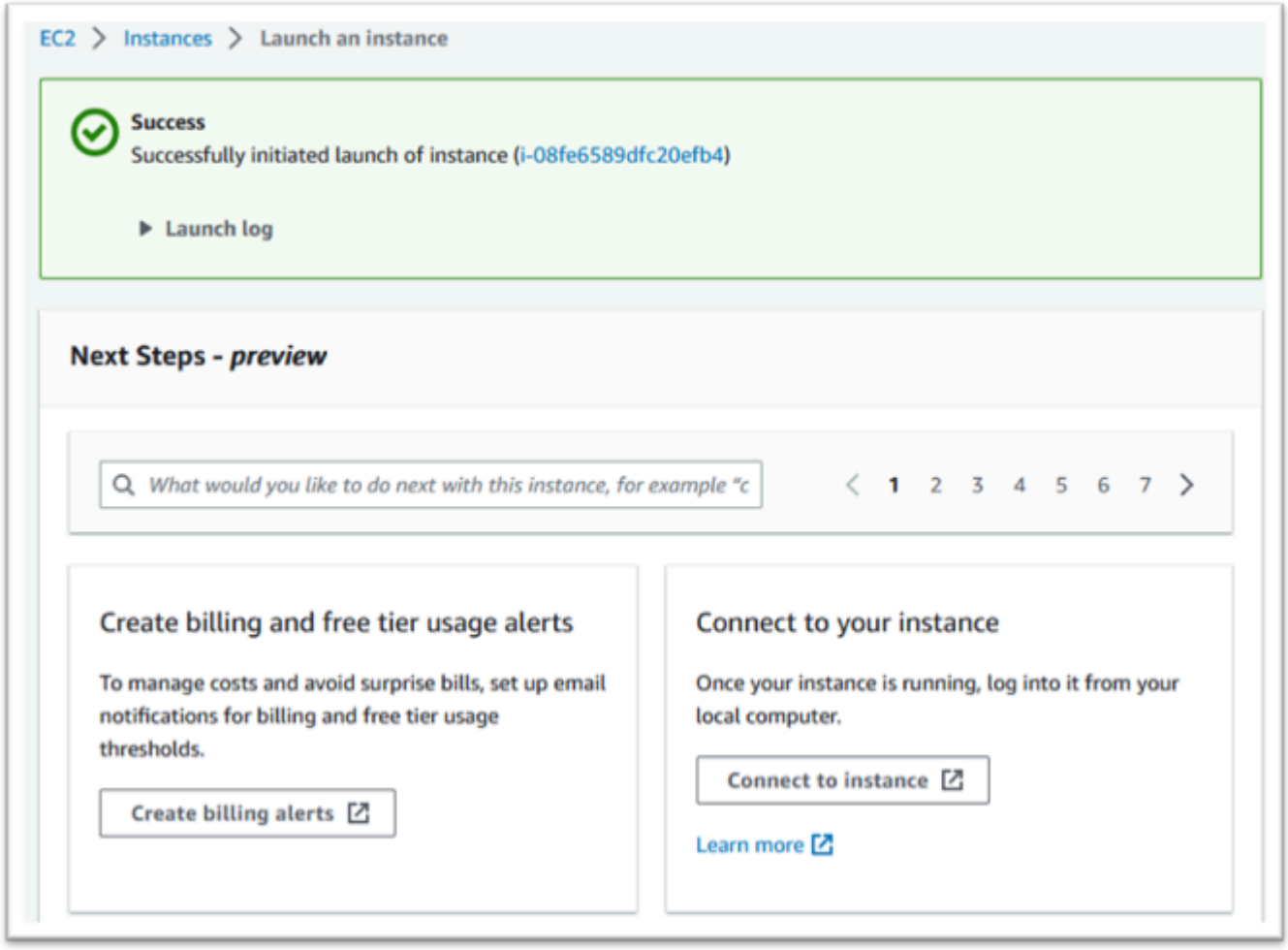


13. The instance launch will fail if an invalid virtual server type is chosen.

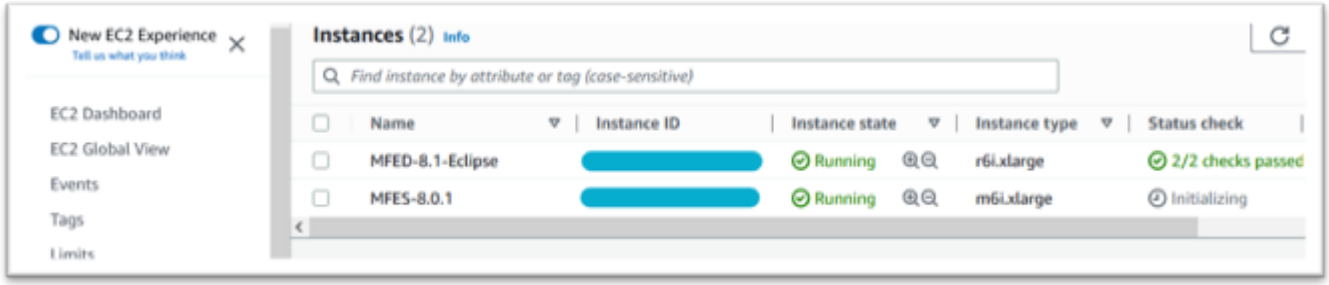
    If this happens, choose **Edit instance config** and change the instance type.



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.

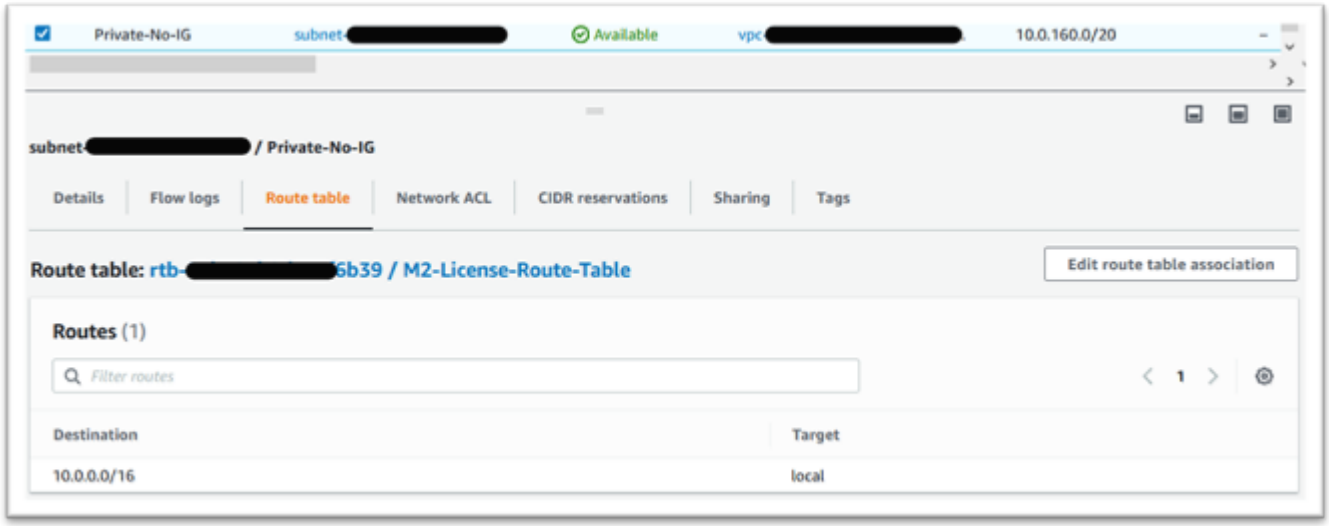
The additional three services will be defined as interface endpoints.

Topics

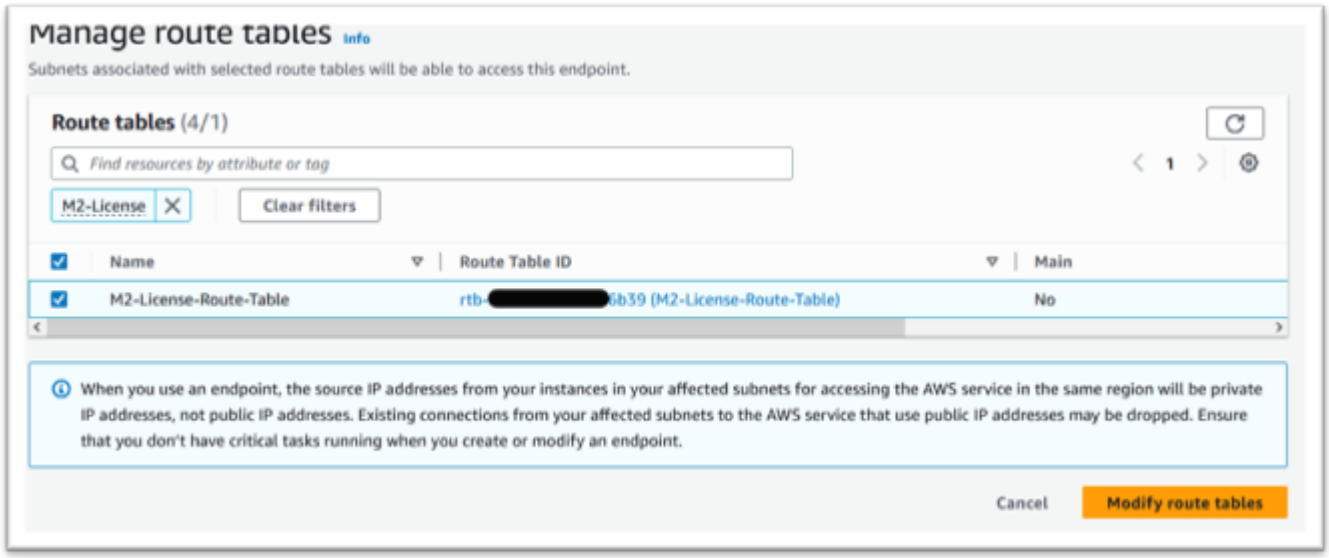
- Add the Route table entry for the Amazon S3 endpoint
- Define the required security group
- Create the service endpoints

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.



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.

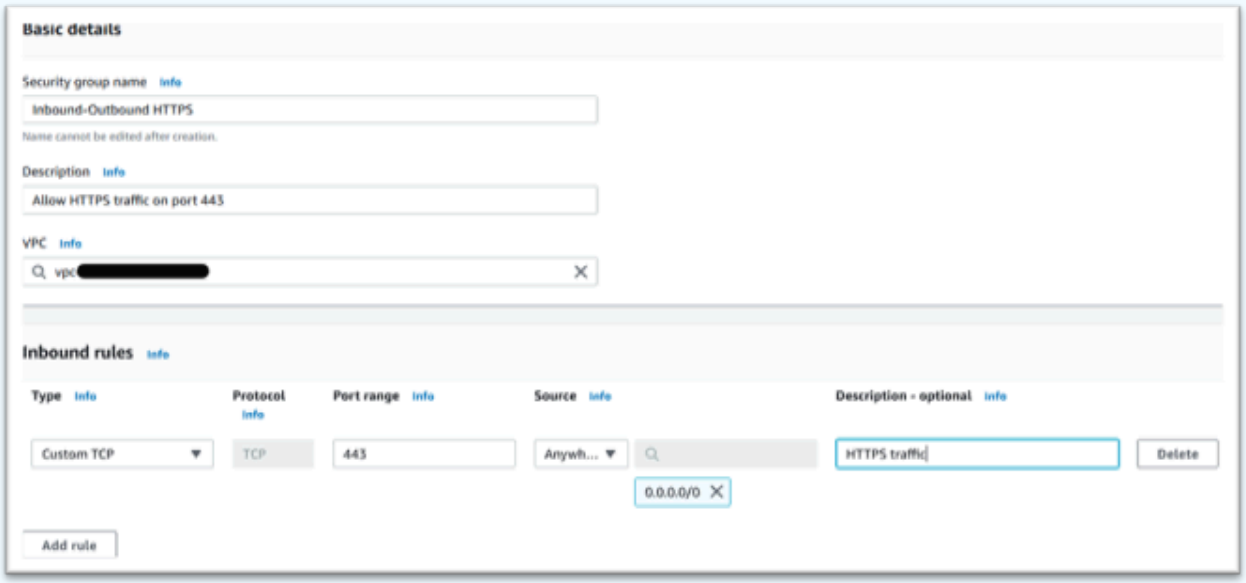


Define the required security group

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 in the *Amazon VPC User Guide*.

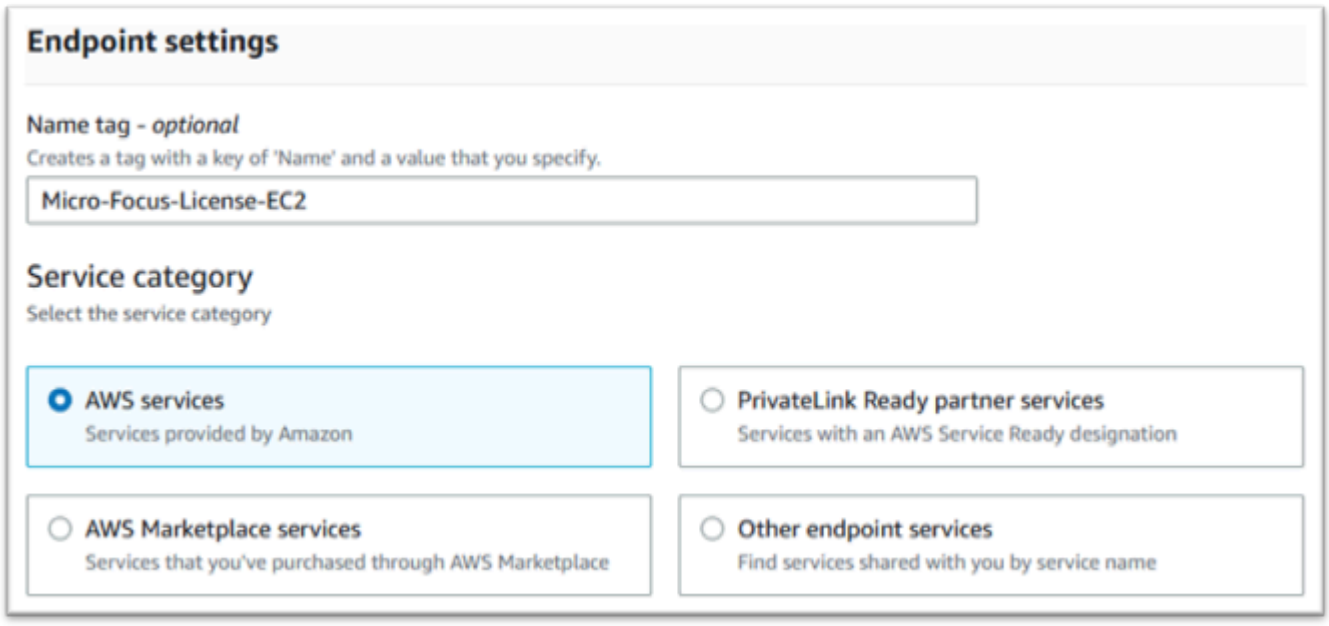


6. Press **Create security group**.

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:

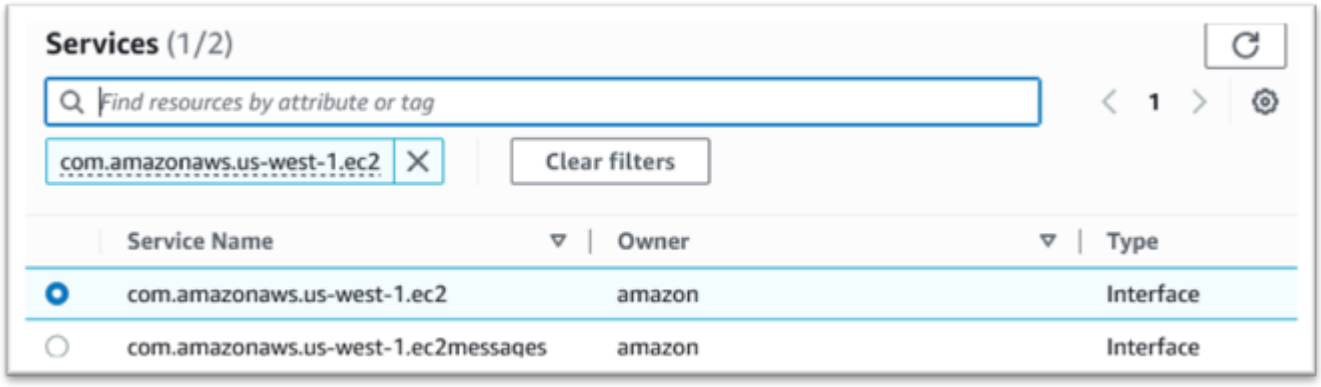
   - “com.amazonaws.*region*.ec2”
   - “com.amazonaws.*region*.sts”
   - “com.amazonaws.*region*.license-manager”

   For example:

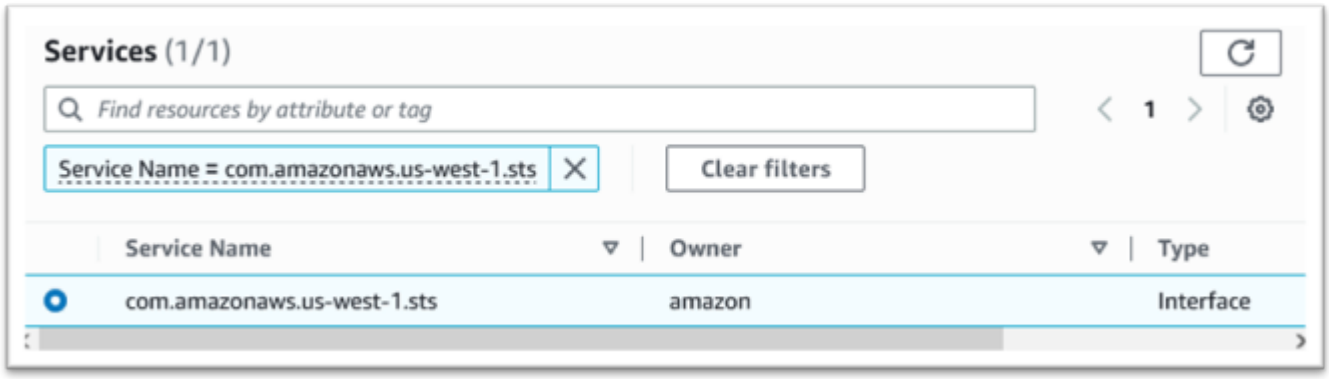
   - “com.amazonaws.us-west-1.ec2”
   - “com.amazonaws.us-west-1.sts”
   - “com.amazonaws.us-west-1.license-manager”

6. Choose the matching Interface service.

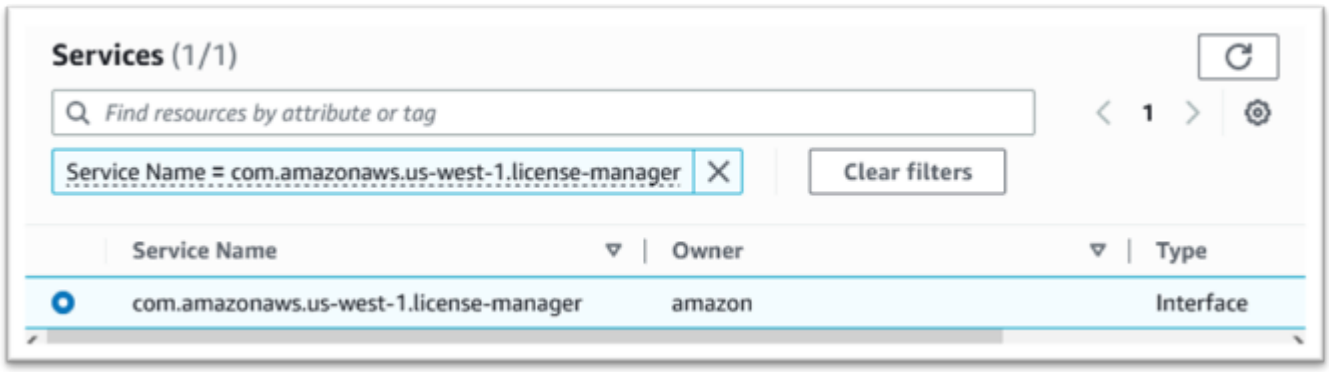
   com.amazonaws.*region*.ec2:



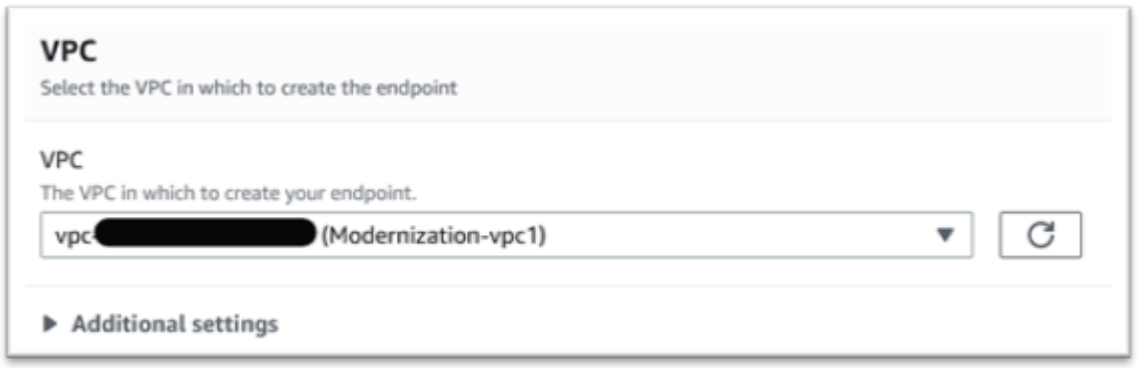
com.amazonaws.*region*.sts:



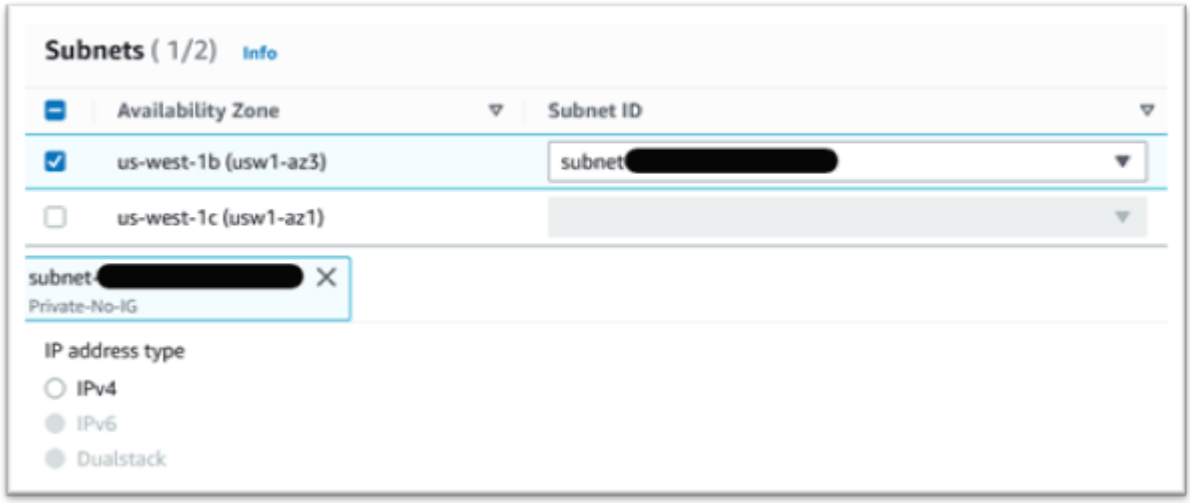
com.amazonaws.*region*.license-manager:



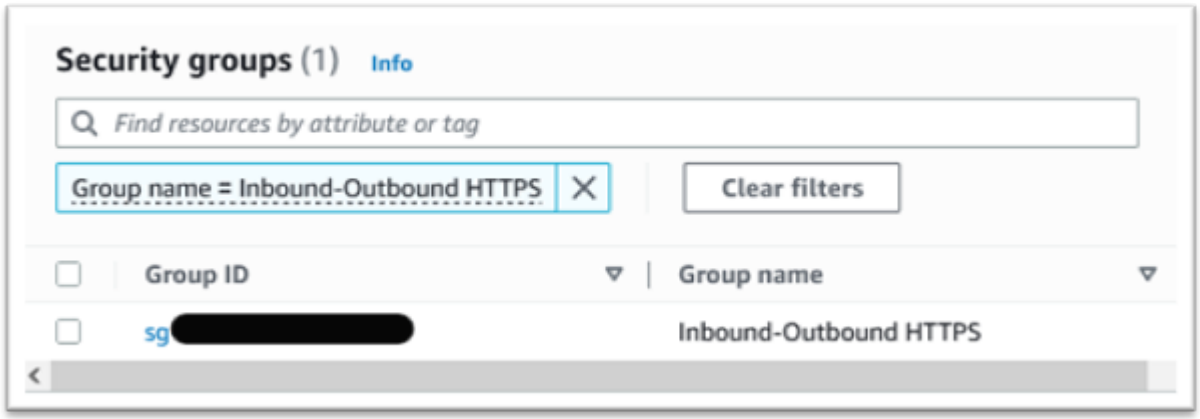
7. For VPC choose the VPC for the instance.



8. Choose the **Availability Zone** and the **Subnets** for the VPC.



9. Choose the Security Group created earlier.



10. Under Policy choose **Full Access**.



11. Choose **Create Endpoint**.
12. Repeat this process for the remaining interfaces.

Set up Automation for Rocket Enterprise Analyzer (formerly Micro Focus) and Rocket 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:

- `m2-enterprise-analyzer-v8.0.4.R1`
- `m2-enterprise-developer-v8.0.4.R1`

Topics

- Set up automation at session start
- Set up automation at session end

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 Rocket Enterprise Developer (formerly Micro Focus Enterprise Developer)

You can access mainframe datasets that are deployed in AWS Mainframe Modernization using the Rocket Software (formerly Micro Focus) runtime. You can view the migrated data sets as tables and columns from an Rocket 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
- Step 1: Set up ODBC Connection to Rocket Software datastore (Amazon RDS database)
- Step 2: Create the MFDBFH.cfg file
- Step 3: Create a structure (STR) file for your copybook layout
- Step 4: Create a database view using the structure (STR) file
- Step 5: View Rocket Software (formerly Micro Focus) data sets as tables and columns

Prerequisites

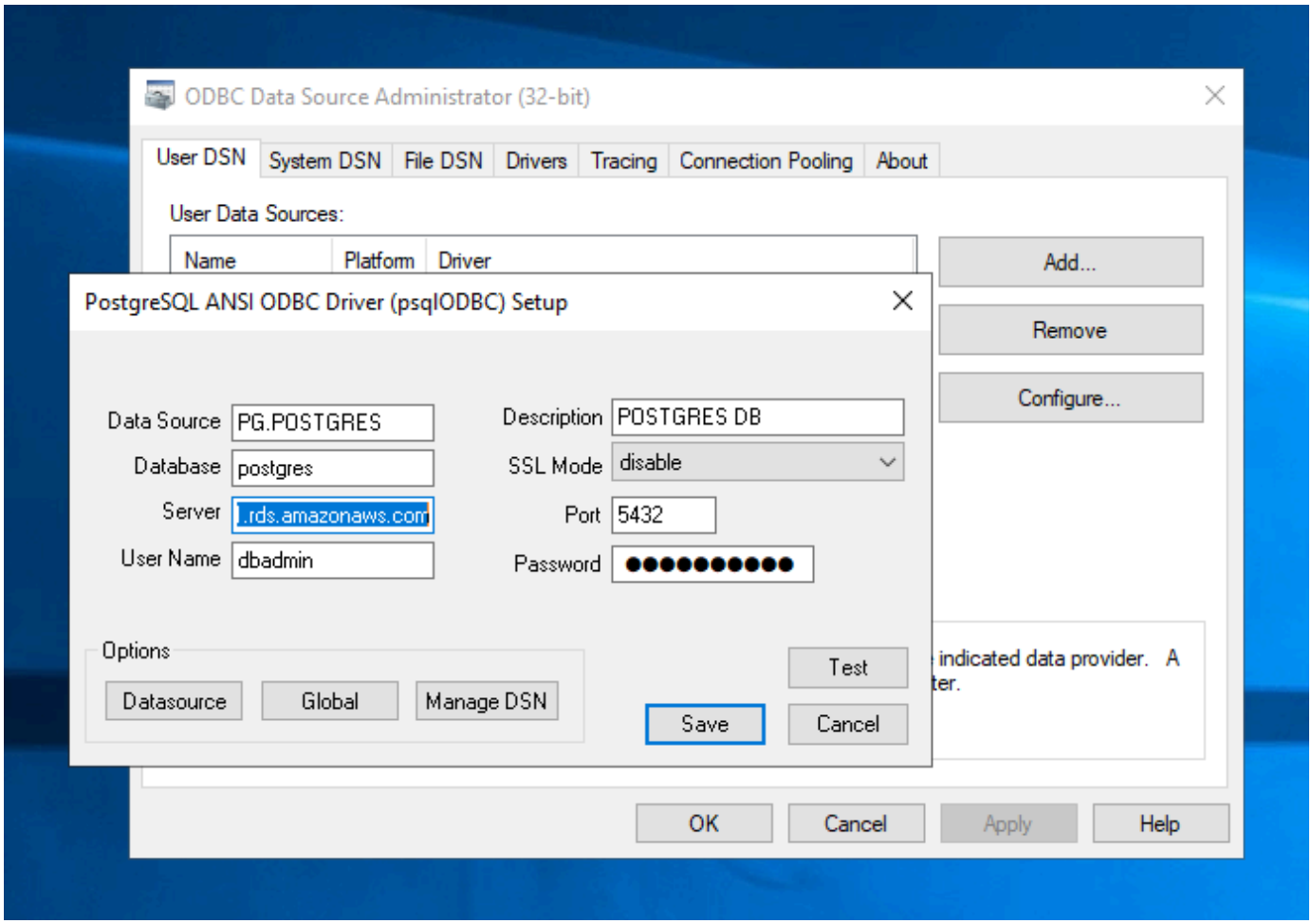
- You must have access to Rocket Enterprise Developer Desktop via AppStream 2.0.
- You must have an application deployed and running under AWS Mainframe Modernization using the Rocket Software runtime engine.
- You are storing your application data in Aurora PostgreSQL-Compatible Edition.

Step 1: Set up ODBC Connection to Rocket Software 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 Rocket 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:

```
Data Source : PG.POSTGRES
Database    : postgres
Server      : rds_endpoint.rds.amazonaws.com
Port        : 5432
User Name   : user_name
Password    : user_password
```



5. Choose **Test** to make sure the connection works. You should see the message `Connection successful` if the test succeeds.

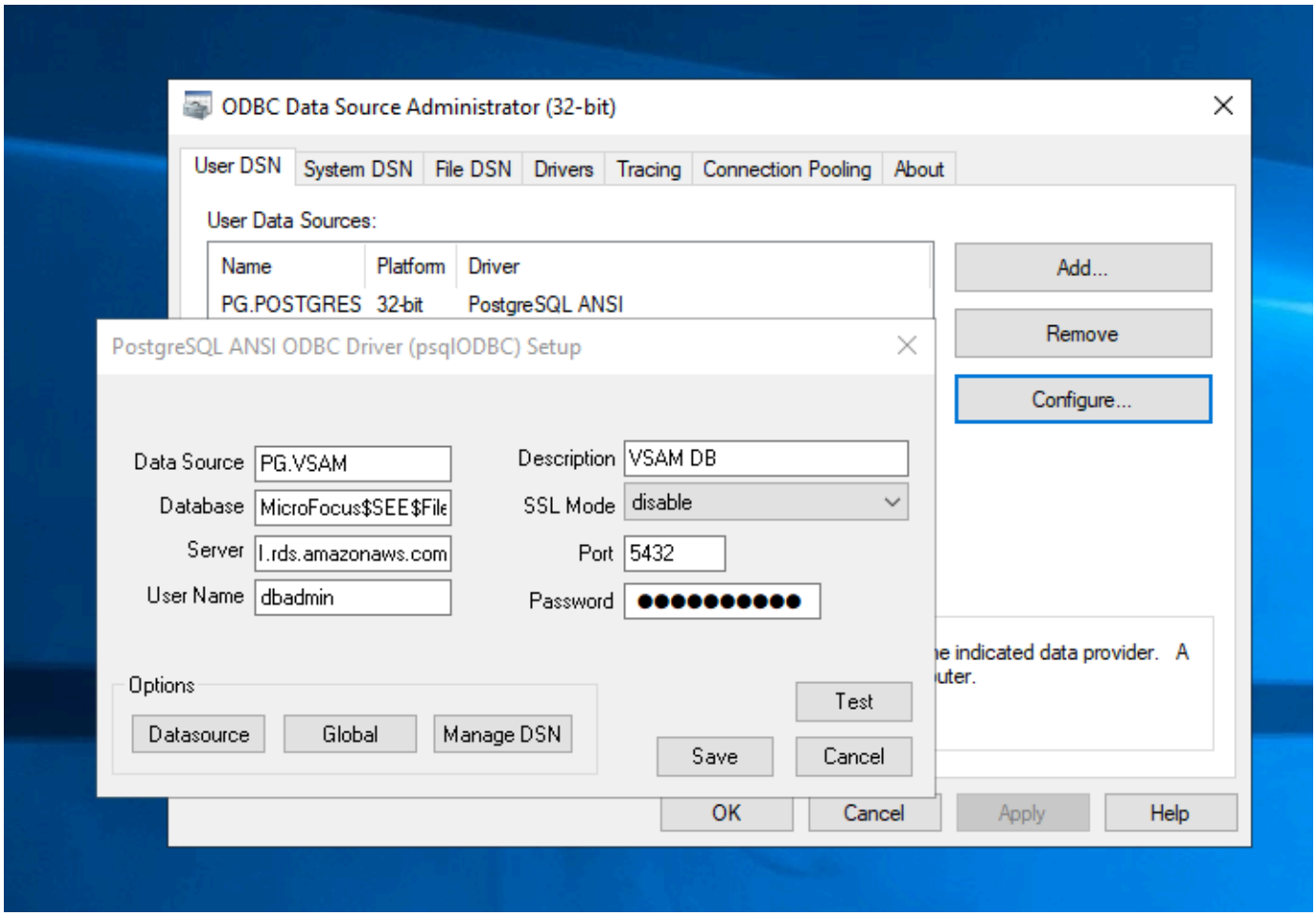
   If the test doesn't succeed, review the following information.

   - 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:

```
Data Source : PG.VSAM
Database    : MicroFocus$SEE$Files$VSAM
Server      : rds_endpoint.rds.amazonaws.com
Port        : 5432
User Name   : user_name
Password    : user_password
```



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.

```
<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:

```
*##*
*## 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.

```
IDENTIFICATION DIVISION.
      PROGRAM-ID. TESTPGM1.

      ENVIRONMENT DIVISION.
      CONFIGURATION SECTION.

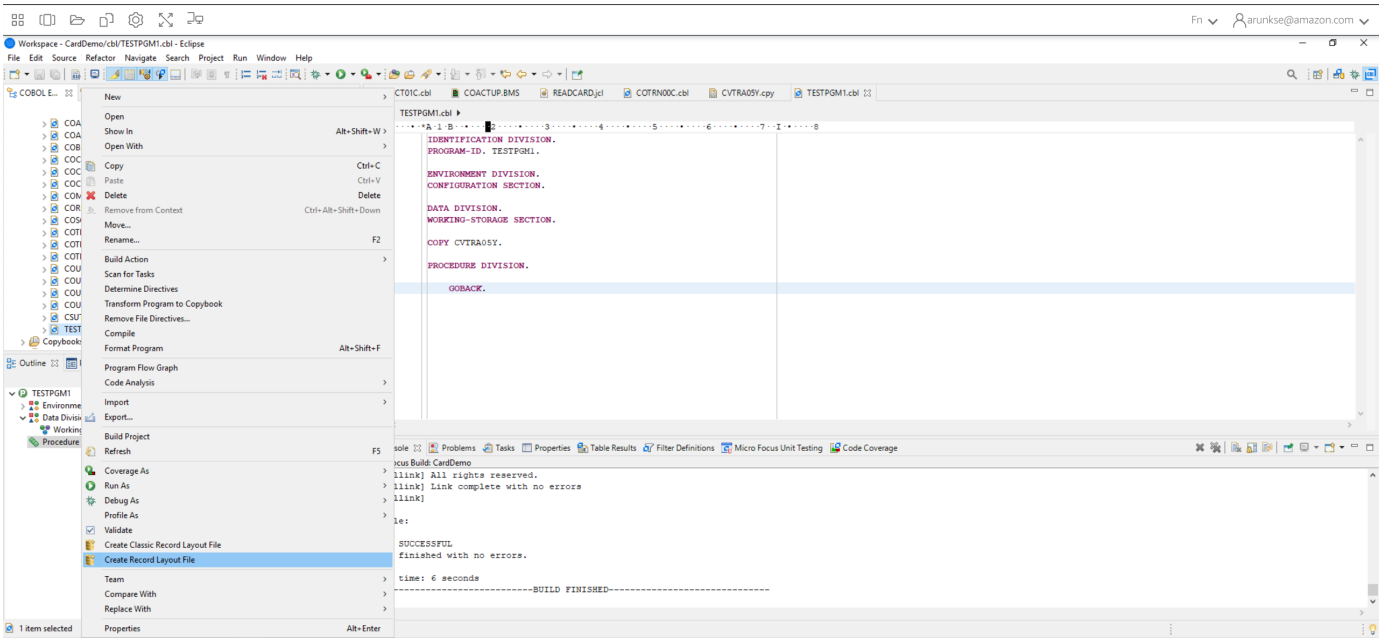
      DATA DIVISION.
      WORKING-STORAGE SECTION.

      COPY CVTRA05Y.
```

```
       PROCEDURE DIVISION.

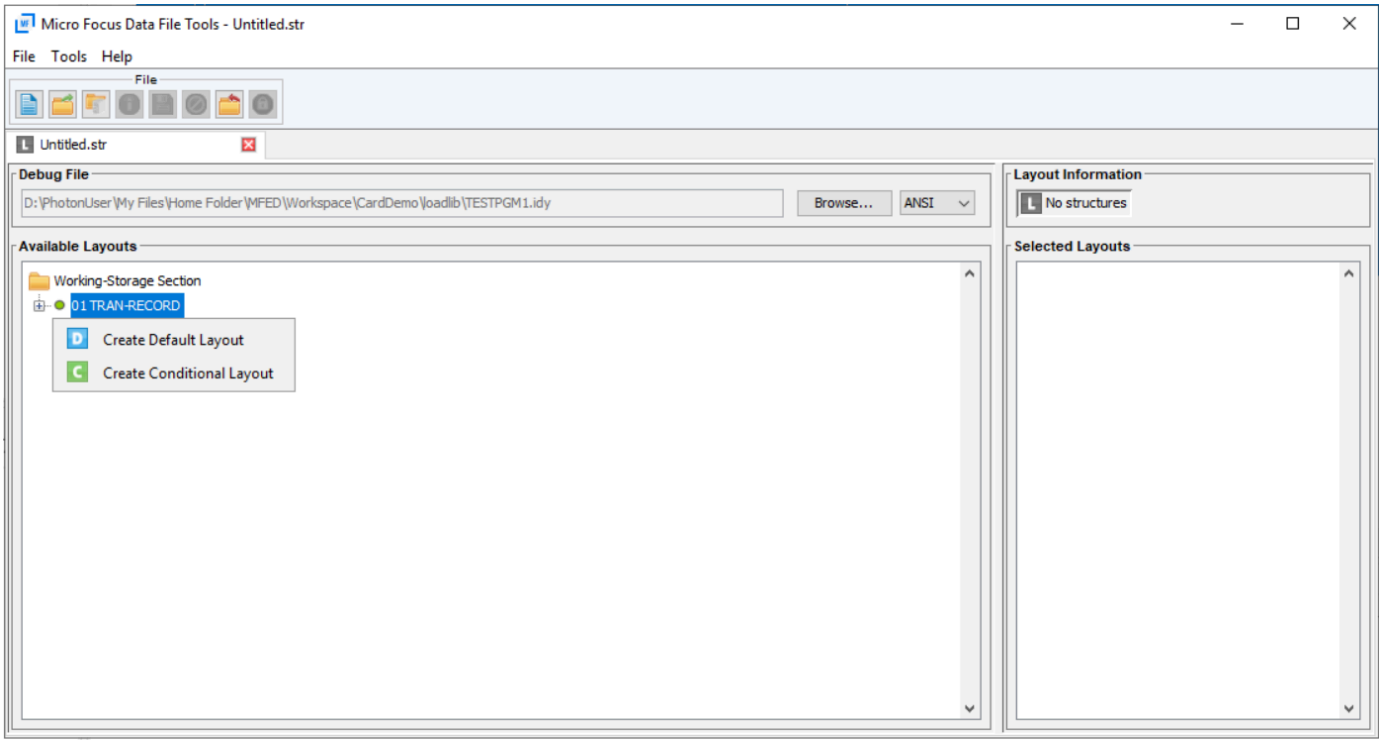
       GOBACK.
```

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.

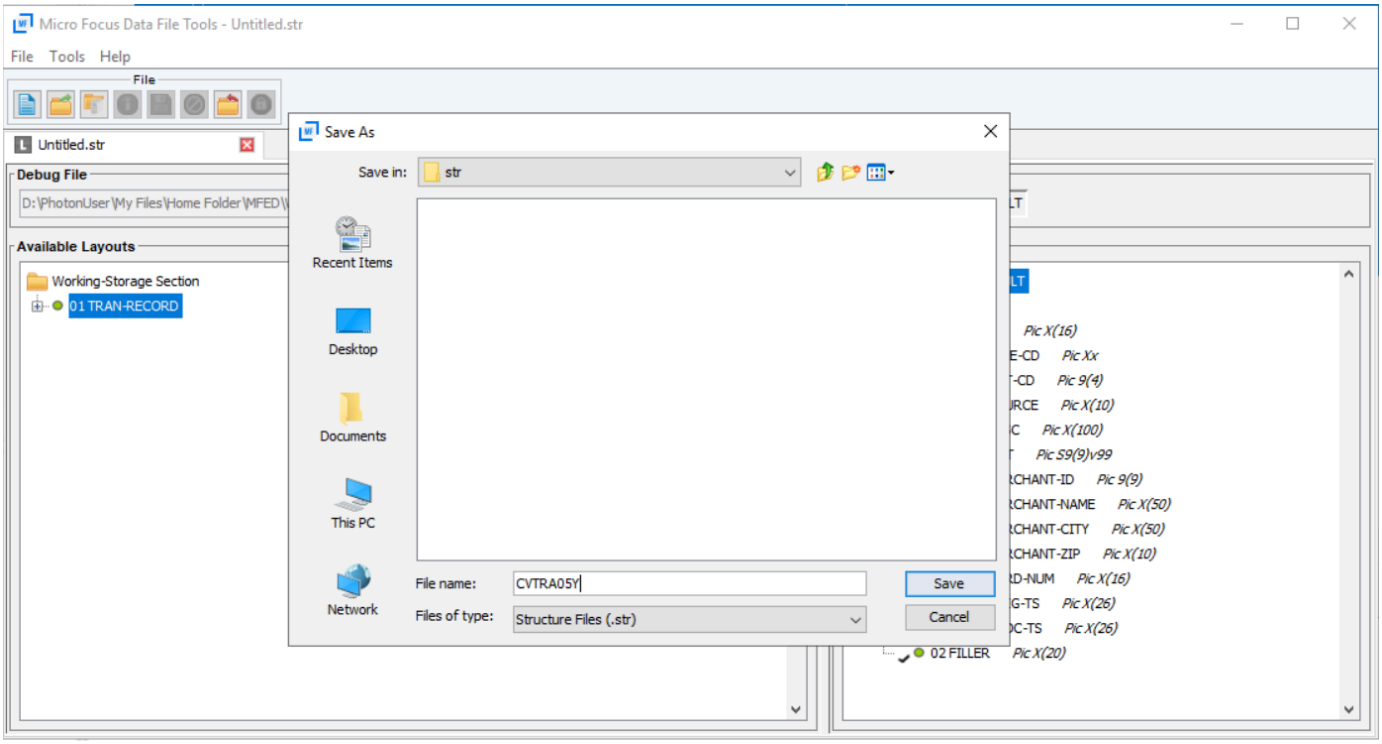


3. Right click on the Record structure and choose **Create Default Layout** (single structure) or **Create Conditional Layout** (multi structure) depending on the layout.

   For more information, see Creating Structure Files and Layouts in the Micro Focus documentation.



4. After creating the layout, choose **File** from the menu and then choose **Save As**. Browse and save the file under your Home Folder with same file name as your copybook. You can choose to create a folder called `str` and save all your structure files there.



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://
ESPACDatabase/VSAM/AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT?folder=/DATA

      ##
      ## Output:
      ##

      Micro Focus Database File Handler - View Generation Tool Version 8.0.00
      Copyright (C) 1984-2022 Micro Focus. All rights reserved.

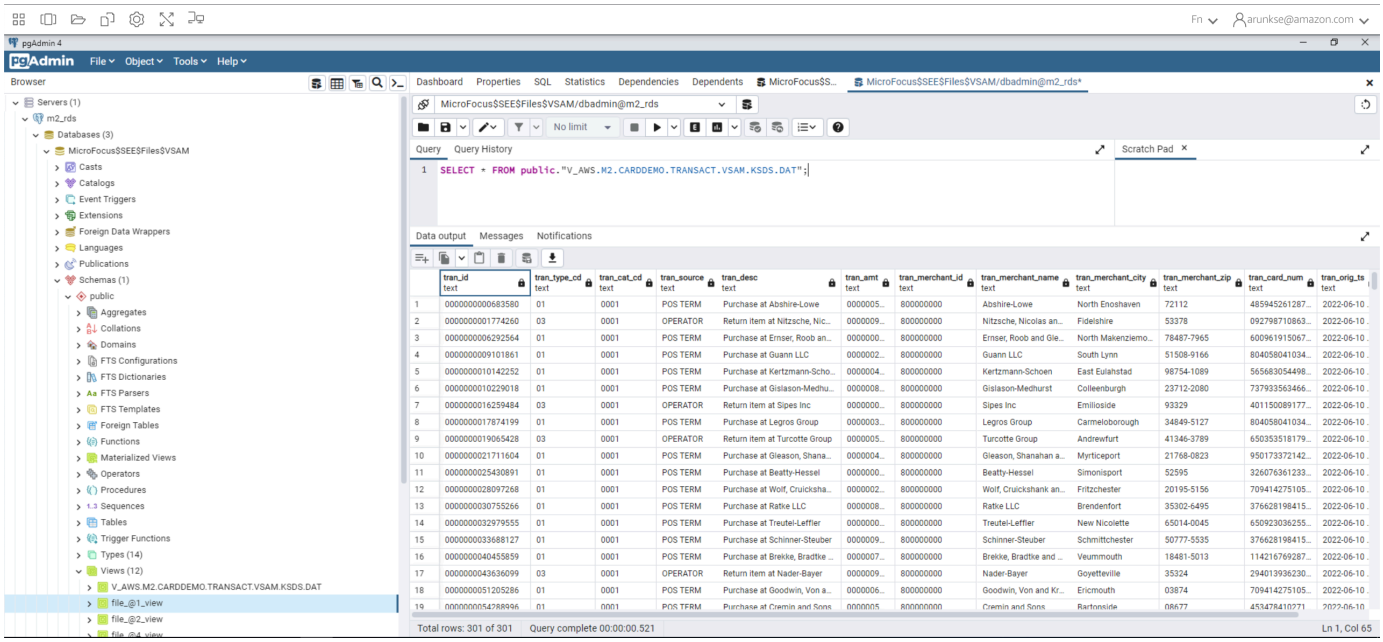
      VGN0017I Using structure definition 'TRAN-RECORD-DEFAULT'
      VGN0022I View 'V_AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS.DAT' installed in
 datastore 'sql://espacdatabase/VSAM'
      VGN0002I The operation completed successfully
```

Step 5: View Rocket Software (formerly 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";
```



Edit data sets using Rocket Software (formerly Micro Focus) Data File Tools in Enterprise Developer

You can view and edit data sets in AWS Mainframe Modernization using the Rocket Software runtime for any migrated data sets.
The steps in this document will guide you through the process of accessing data sets using Data File Tools.
This allows you to view and edit the migrated data sets as needed.

Topics

- Prerequisites
- Launch Rocket Software(formerly Micro Focus) Data File Tools
- Edit VSAM data sets stored in the MFDBFH database
- Edit non-VSAM data sets stored in the MFDBFH database
- Edit VSAM and non-VSAM data sets stored in the File System (EFS/FSx)

Prerequisites

Before you start, you must have an application deployed with the data sets

imported under the AWS Mainframe Modernization service using the Rocket Software engine.

To continue with editing the data sets, you must complete the Step 1, Step 2,
and (optionally) Step 3 from the the section called "View data sets as tables in Enterprise Developer" page
to configure the ODBC connection, and Micro Focus datastore (that is, MFDBFH).

> **Important**
>
> This guide assumes that you are using Amazon Aurora Postgres as Micro Focus datastore (MFDBFH)
> to store your application data.

Launch Rocket Software(formerly Micro Focus) Data File Tools

After completing the prerequisites, you launch the Micro Focus Data File Tools by setting up the MFDBFH_CONFIG environment variable to access the data sets stored in the database (MFDBFH).

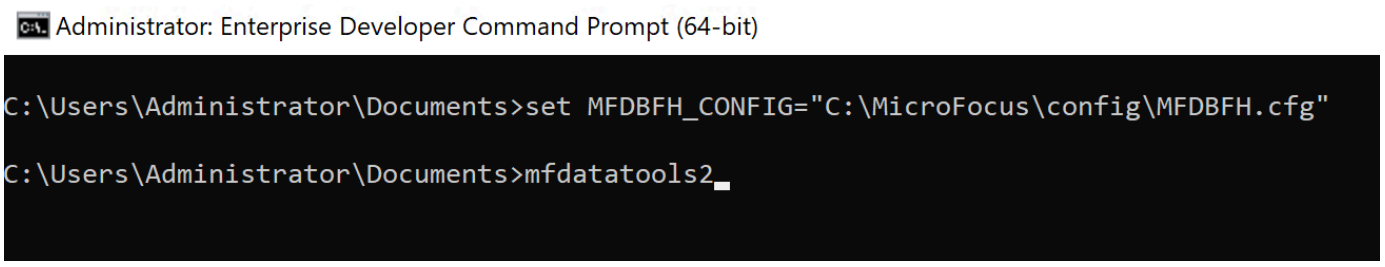
To do this,

1. Log in to the Micro Focus Enterprise Developer desktop, and launch the **Enterprise Developer command prompt (64-bit)** from the **Start Menu**.
2. Set the MFDBFH_CONFIG environment variable with the full path to your MFDBCH.cfg file.

```
set MFDBFH_CONFIG="C:\MicroFocus\config\MFDBFH.cfg"
```

3. Launch Micro Focus Data File Tools from the Enterprise Developer command line using the following command.

```
mfdatatools2
```



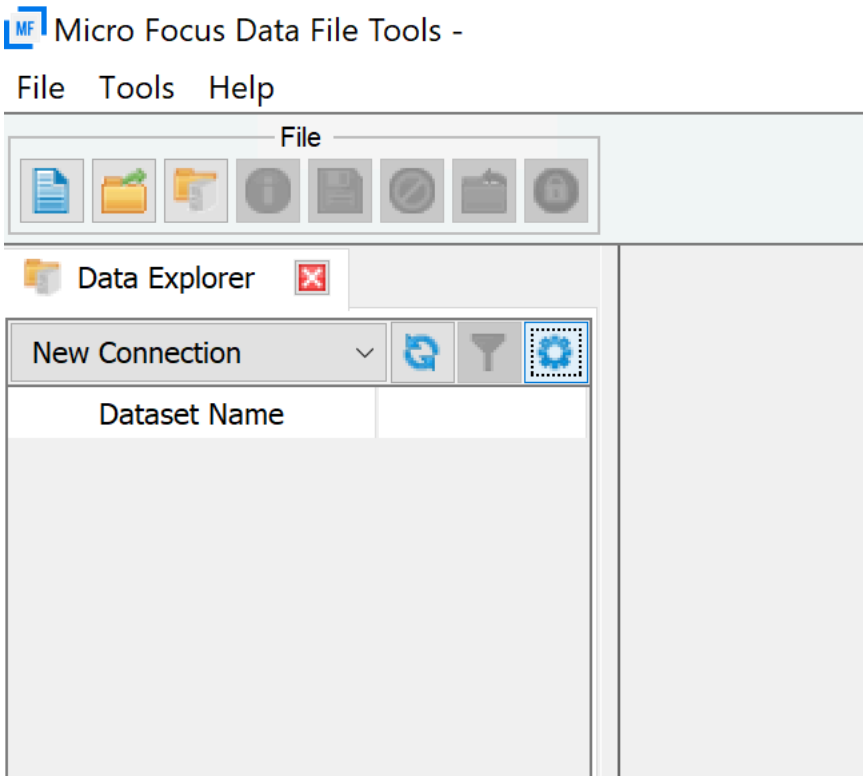
This opens the Micro Focus Data File Tools in a separate window.

Edit VSAM data sets stored in the MFDBFH database

Once you launch the Micro Focus Data File Tools, you open a VSAM data set that is stored in the Micro Focus datastore.

To do this,

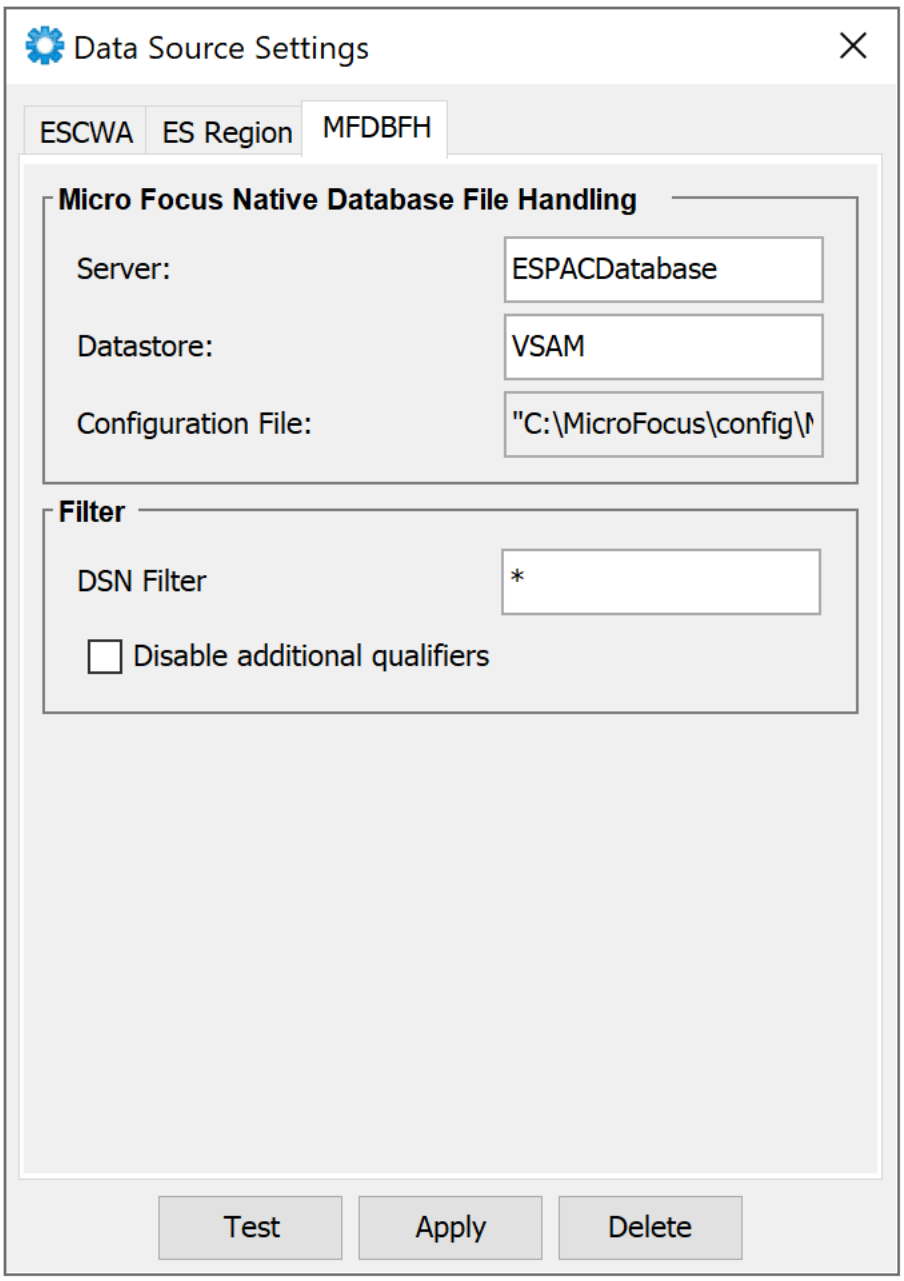
1. From the **File menu** in the Micro Focus Data File Tools window, choose **Data Explorer**.
2. In the Data Explorer section, choose **Settings** (gear icon) to configure a new connection. This opens a **Data Source Settings** window.



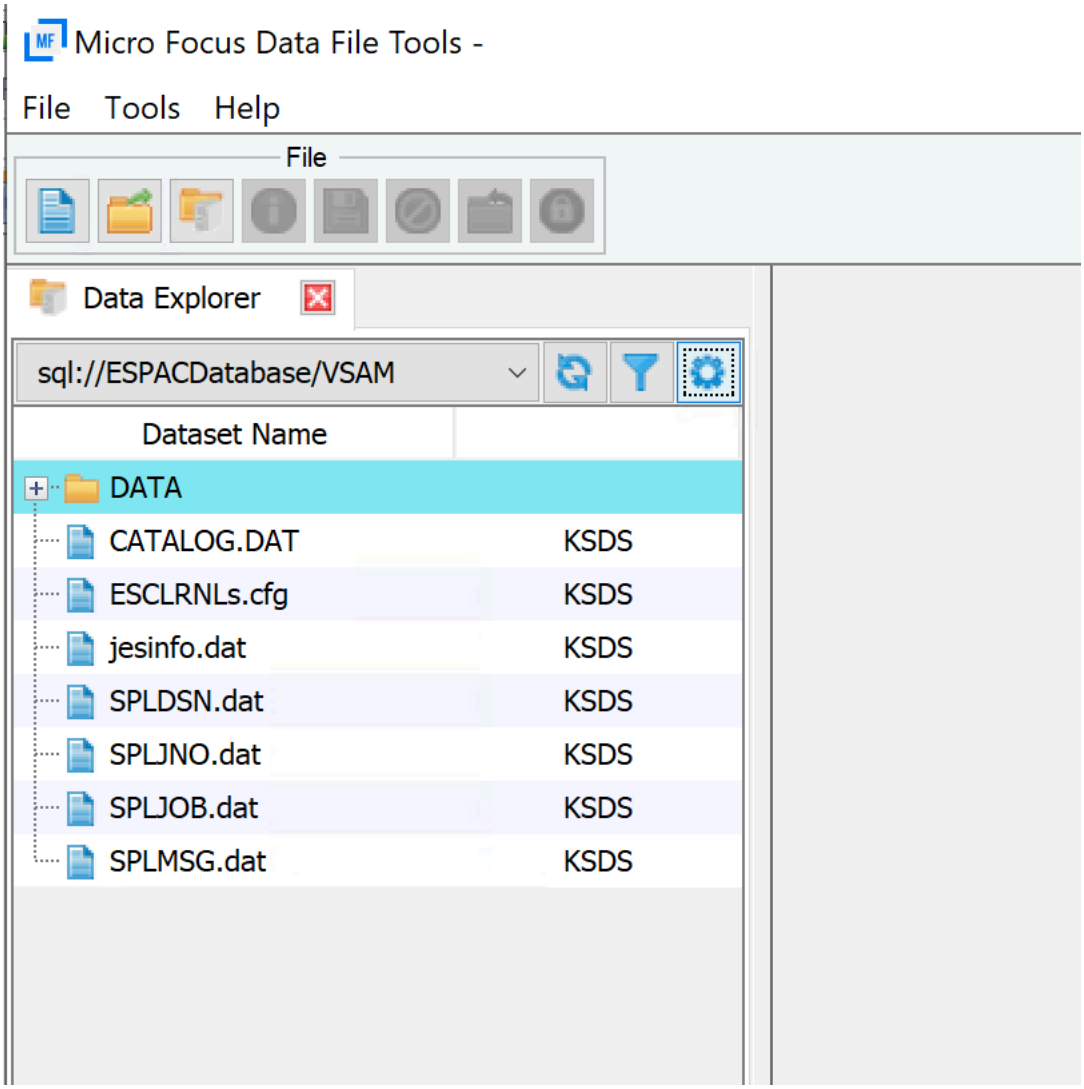
3. In the Data Source Settings window, choose the **MFDBFH** tab, and enter the following values:

   - Server: `ESPACDatabase`
   - Datastore: `VSAM`

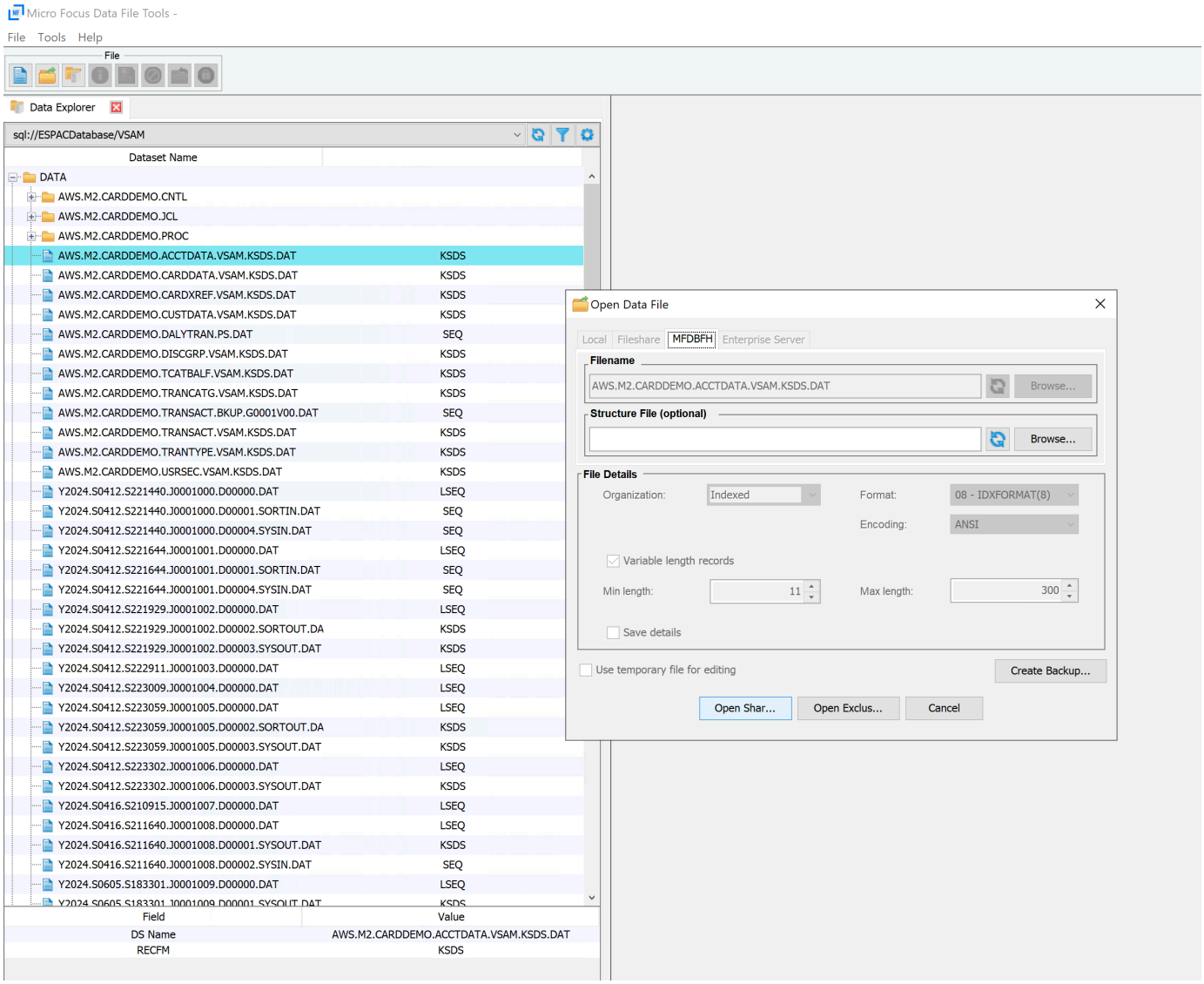
   Choose **Apply** to save the configuration.



The Data Explorer now shows all data sets that are stored in MFDBFH.



4. Expand the relative path DATA and double click on the VSAM data set you want to open.
5. In the **Open Data File** window, choose **Open Shared** or **Open Exclusive** to open the data set.



You can now view or edit the open data set.

Edit non-VSAM data sets stored in the MFDBFH database

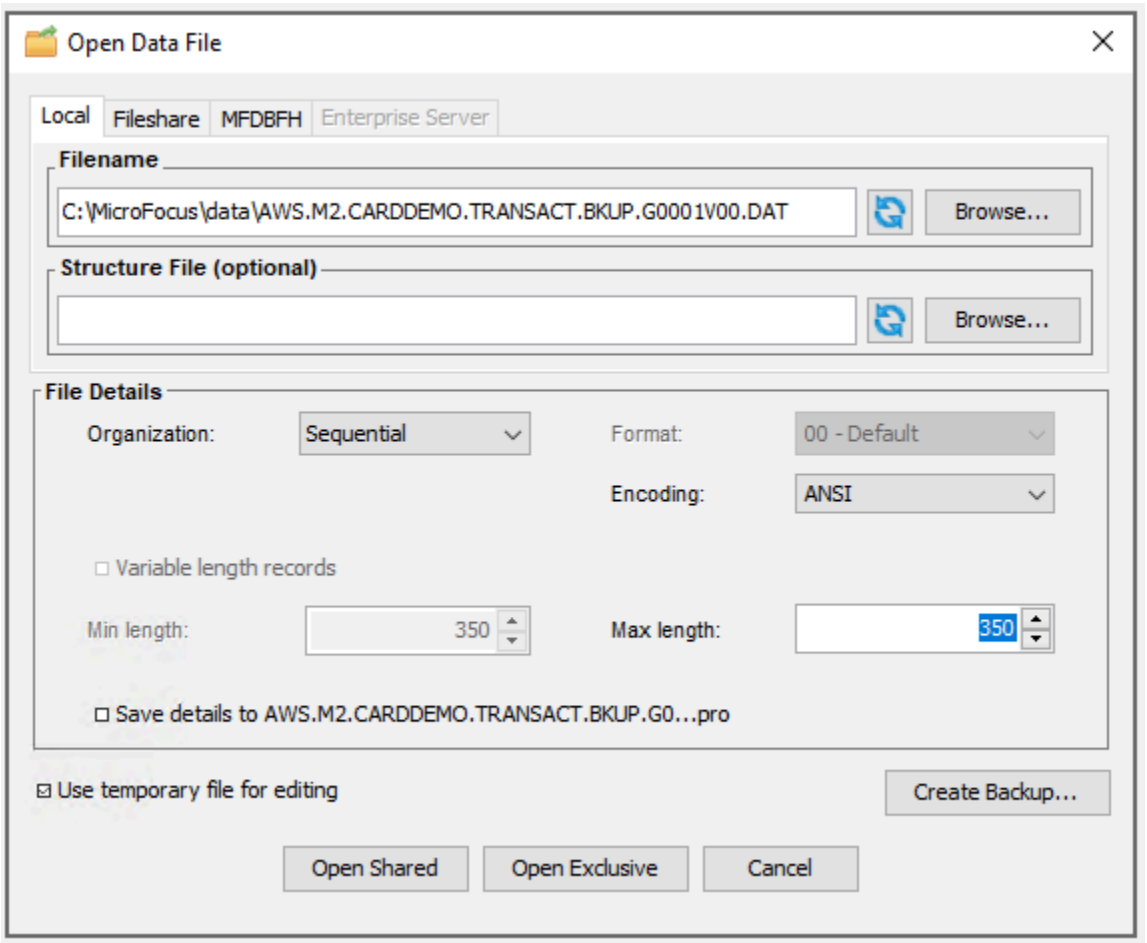
If you want to edit non-VSAM data sets, you open a non-VSAM data set that is stored in the Micro Focus datastore.

To do this,

1. From the Enterprise Developer command prompt (64-bit) run the `dbfhdeploy data extract` command to download the non-VSAM data set to your local file system.

> **Note**
>
> Before running this command, make sure you have set the `MFDBFH_CONFIG` environment variable with the full path to your `MFDBFH.cfg` file.
>
> ```
> dbfhdeploy data extract sql://ESPACDatabase/VSAM/
> AWS.M2.CARDDEMO.TRANSACT.BKUP.G0001V00.DAT?folder=/DATA C:\MicroFocus\data
> \AWS.M2.CARDDEMO.TRANSACT.BKUP.G0001V00.DAT
> ```

2. Launch Micro Focus Data File Tools from the **Start Menu**.
3. From the File Menu of Micro Focus Data File Tools, choose **Open**, and then choose **Data File**.
4. In the Open Data File window, browse the downloaded data set in your local file system. Edit the **File Details** as required. Then choose **Open Shared** or **Open Exclusive** to open the data set.



You can now view or edit the open data set.

The edited or updated data sets can be imported back to the Micro Focus datastore using steps in the section called “Import data sets for applications” or by using The dbfhdeploy Command Line Utility.

Edit VSAM and non-VSAM data sets stored in the File System (EFS/FSx)

You can also open a data set stored in a file system.

To do this,

1. Mount the EFS/FSx file system in on the Enterprise Developer EC2 instance.
2. Use the Micro Focus Data File Tools to browse, and open the data sets from the file system.

Tutorials for Rocket Software (formerly Micro Focus)

The tutorials in this section help you to get started with setting up various tasks in the Rocket Software runtime engine for the AWS Mainframe Modernization service. These tutorials are for setting up sample application, CI/CD pipelines, using templates with Rocket Enterprise Developer, and setting up Enterprise Analyzer.

Topics

- Tutorial: Setting up the Rocket Software (formerly Micro Focus) build for the BankDemo sample application
- Tutorial: Setting up a CI/CD pipeline for use with Rocket Enterprise Developer (formerly Micro Focus Enterprise Developer)
- Tutorial: Set up AppStream 2.0 for use with Rocket Enterprise Analyzer and Rocket Enterprise Developer
- Tutorial: Use templates with Rocket Enterprise Developer (formerly Micro Focus Enterprise Developer)
- Tutorial: Set up Enterprise Analyzer on AppStream 2.0
- Tutorial: Set up Rocket Enterprise Developer on AppStream 2.0

Tutorial: Setting up the Rocket Software (formerly 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 reponsitories 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
- Step 1: Share the build assets with AWS account
- Step 2: Create Amazon S3 buckets
- Step 3: Create the build spec file
- Step 4: Upload the source files
- Step 5: Create IAM policies

- Step 6: Create an IAM role
- Step 7: Attach the IAM policies to the IAM role
- Step 8: Create the CodeBuild project
- Step 9: Start the build
- Step 10: Download output artifacts
- Clean up resources

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 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: Share the build assets with AWS account

In this step, you ensure that you share the build assets with your AWS account, especially in the Region where assets are being used.

1. Open the AWS Mainframe Modernization console at 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**.

> **Important**
>
> You need to do this step once in every AWS Region where you intend to do builds.

Step 2: 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 3: 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.

```
version: 0.2
env:
  exported-variables:
    - CODEBUILD_BUILD_ID
```

```
    - CODEBUILD_BUILD_ARN
phases:
  install:
    runtime-versions:
      python: 3.7
  pre_build:
    commands:
      - echo Installing source dependencies...
      - ls -lR $CODEBUILD_SRC_DIR/source
  build:
    commands:
      - echo Build started on `date`
      - /start-build.sh -Dbasedir=$CODEBUILD_SRC_DIR/source -Dloaddir=
$CODEBUILD_SRC_DIR/target
  post_build:
    commands:
      - ls -lR $CODEBUILD_SRC_DIR/target
      - echo Build completed on `date`
artifacts:
  files:
    - $CODEBUILD_SRC_DIR/target/**
```

Here `CODEBUILD_BUILD_ID`, `CODEBUILD_BUILD_ARN`, `$CODEBUILD_SRC_DIR/source`, and `$CODEBUILD_SRC_DIR/target` are environment variables available within CodeBuild. For more information, see Environment variables in build environments.

At this point, your directory should look like this.

```
(root directory name)
    |-- build.xml
    |-- buildspec.yml
    |-- LICENSE.txt
    |-- source
         |... etc.
```

3. Zip the contents of the folder to a file named `BankDemo.zip`.. For this tutorial, you can't zip the folder. Instead, zip the contents of the folder to the file `BankDemo.zip`.

Step 4: 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**.
4. Navigate to and choose your `BankDemo.zip` file.
5. Choose **Upload**.

Step 5: 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 Rocket Software 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.

```
{
    "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-*-<region>-prod"
        }
    ]
}
```

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.

```
{
      "Version": "2012-10-17",
      "Statement": [
          {
              "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
                  "logs:PutLogEvents"
              ],
              "Resource": [
                  "arn:aws:logs:regionId:accountId:log-group:/aws/codebuild/
codebuild-bankdemo-project",
                  "arn:aws:logs:regionId:accountId:log-group:/aws/codebuild/
codebuild-bankdemo-project:*"
              ],
              "Effect": "Allow"
          },
          {
              "Action": [
```

```
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::codebuild-regionId-accountId-input-bucket",
                "arn:aws:s3:::codebuild-regionId-accountId-input-bucket/*",
                "arn:aws:s3:::codebuild-regionId-accountId-output-bucket",
                "arn:aws:s3:::codebuild-regionId-accountId-output-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

2. Provide a name for the policy, for example, `BankdemoCodeBuildRolePolicy`.

Step 6: 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 7: 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 8: 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,

   - For Rocket Software v9: Enter `673918848628.dkr.ecr.us-west-1.amazonaws.com/m2-enterprise-build-tools:9.0.7.R1`. If you're using a different AWS Region with Rocket Software v9, you can also specify `673918848628.dkr.ecr.<m2-region>.amazonaws.com/m2-enterprise-build-tools:9.0.7.R1`, where <m2-region> is an AWS Region in which AWS Mainframe Modernization service is available (for example, `eu-west-3`).
   - For Rocket Software v8: Enter `673918848628.dkr.ecr.us-west-2.amazonaws.com/m2-enterprise-build-tools:8.0.9.R1`
   - For Rocket Software v7: Enter `673918848628.dkr.ecr.us-west-2.amazonaws.com/m2-enterprise-build-tools:7.0.R10`
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`.
13. Under **Artifacts packaging**, choose **Zip**.
14. Choose **Create build project**.

Step 9: Start the build

In this step, you start the build.

1. Log in to the CodeBuild console.
2. In the left navigation pane, choose **Build projects**.
3. Choose the build project that you created previously, for example, `codebuild-bankdemo-project`.
4. Choose **Start build**.

This command starts the build. The build runs asynchronously. The output of the command is a JSON that includes the attribute id. This attribute idis a reference to the CodeBuild build id of the build that you just started. You can view the status of the build in the CodeBuild console. You can also see detailed logs about the build execution in the console. For more information, see View detailed build information in the *AWS CodeBuild User Guide*.

When the current phase is COMPLETED, it means that your build finished successfully, and your compiled artifacts are ready on Amazon S3.

Step 10: Download output artifacts

In this step, you download the output artifacts from Amazon S3. The Rocket Software 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** role="bold"> section, choose the name of your output bucket, for example, `codebuild-regionId-accountId-output-bucket`.
3. Choose **Download** role="bold">.
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 Rocket Enterprise Developer (formerly Micro Focus Enterprise Developer)

This tutorial shows you how to import, edit, compile, and run the BankDemo sample application in Rocket Enterprise Developer, and then to commit your changes to trigger a CI/CD pipeline.

Contents

- Prerequisites
- Create CI/CD pipeline basic infrastructure
- Create AWS CodeCommit repository and CI/CD pipeline
  - Sample YAML Trigger File config_git.yml
- Enterprise Developer AppStream 2.0 Creation
- Enterprise Developer Setup and Test
  - Clone the BankDemo CodeCommit repository in Enterprise Developer
  - Create BankDemo mainframe COBOL project and build application
  - Create local BankDemo CICS and batch environment for testing
  - Start the BANKDEMO server from Enterprise Developer
  - Start the Rumba 3270 terminal
  - Run a BankDemo transaction
  - Stop the BANKDEMO server from Enterprise Developer
- Exercise 1: Enhance Loan Calculation in BANKDEMO Application
  - Add loan analysis rule to Enterprise Developer Code Analysis
  - Step 1: Perform code analysis for loan calculation
  - Step 2: Modify CICS BMS map and COBOL program and test
  - Step 3: Add total amount calculation in COBOL program
  - Step 4: Commit changes and run CI/CD pipeline
- Exercise 2: Extract loan calculation in BankDemo application

- Step 1: Refactor loan calculation routine into a COBOL section
- Step 2: Extract loan calculation routine to a standalone COBOL program
- Step 3: Commit changes and run the CI/CD pipeline
- Clean up resources

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`
  - Download from Europe (Frankfurt) Region.
  - Download from US East (N. Virginia) Region.

The purpose of each file is as follows:

`basic-infra.yaml`

This AWS CloudFormation template creates the basic infrastructure needed for the CI/CD pipeline: VPC, Amazon S3 buckets, and so on.

`pipeline.yaml`

This AWS CloudFormation template is used by an Lambda function to launch the pipeline stack. Make sure this template is located in a publicly accessible Amazon S3 bucket. Add the link to this bucket as the default value for the `PipelineTemplateURL`parameter in the `basic-infra.yaml` template.

`m2-code-sync-function.zip`

This Lambda function creates the CodeCommit repository, the directory structure based on the `config_git.yaml`, and launches the pipeline stack using `pipeline.yaml`. Make sure this zip file is available in a publicly accessible Amazon S3 bucket in all the AWS Regions where AWS Mainframe Modernization is supported. We recommend that you store the file in a bucket in one AWS Region and replicate it to buckets across all AWS Regions. Use a naming convention for the bucket with a suffix that identifies the specific AWS Region (for example, `m2-cicd-deployment-source-eu-west-1`) and add the prefix `m2-cicd-deployment-source` as default value for parameter `DeploymentSourceBucket` and form the full bucket by using the AWS CloudFormation substitution function `!Sub {DeploymentSourceBucket}-${AWS::Region}` while referring to that bucket in the `basic-infra.yaml` template for resource `SourceSyncLambdaFunction`.

`config_git.yml`

CodeCommit directory structure definition. For more information, see Sample YAML Trigger File config_git.yml.

`BANKDEMO-source.zip`.

BankDemo source code and configuration file created from the CodeCommit repository.

`BANKDEMO-exercise.zip`.

BankDemo source for tutorial exercises created from the CodeCommit repository.

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.

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
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?

If you select 'Yes', then you must provide the VPC ID and the Subnet IDs.

No (Create one)

Which VPC ID should be used?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Which private subnet ID should be used?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Which private subnet ID in a different AZ should be used for HA?

If you selected 'Yes' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

Enter the CIDR block that should be used for the new VPC

If you selected 'No (Create one)' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

11.11.0.0/16

CIDR bits for creating subnets. Choose 5 for /27, 6 for /26, 7 for /25, 8 for /24 range

If you selected 'No (Create one)' for UseExistingVPC, this parameter is required. Otherwise, this value will be ignored.

5

Deployment Configuration

Name of the S3 bucket which contains the source files for this stack deployment

Don't change unless you know what you are doing.

m2

Name of the source package file for the infrastructure Lambda function

Don't change unless you know what you are doing.

cicd/mf/m2-code-sync-function.zip

Full URL of the pipeline CloudFormation template file

Don't change unless you know what you are doing.

https://m2-us-east-1.s3.us-east-1.amazonaws.com/cicd/mf/pipeline.yaml

What name prefix to use for the new S3 buckets?

A name prefix for the S3 buckets that will be created by this stack.

mf

Cancel Previous Next

Make the following changes:

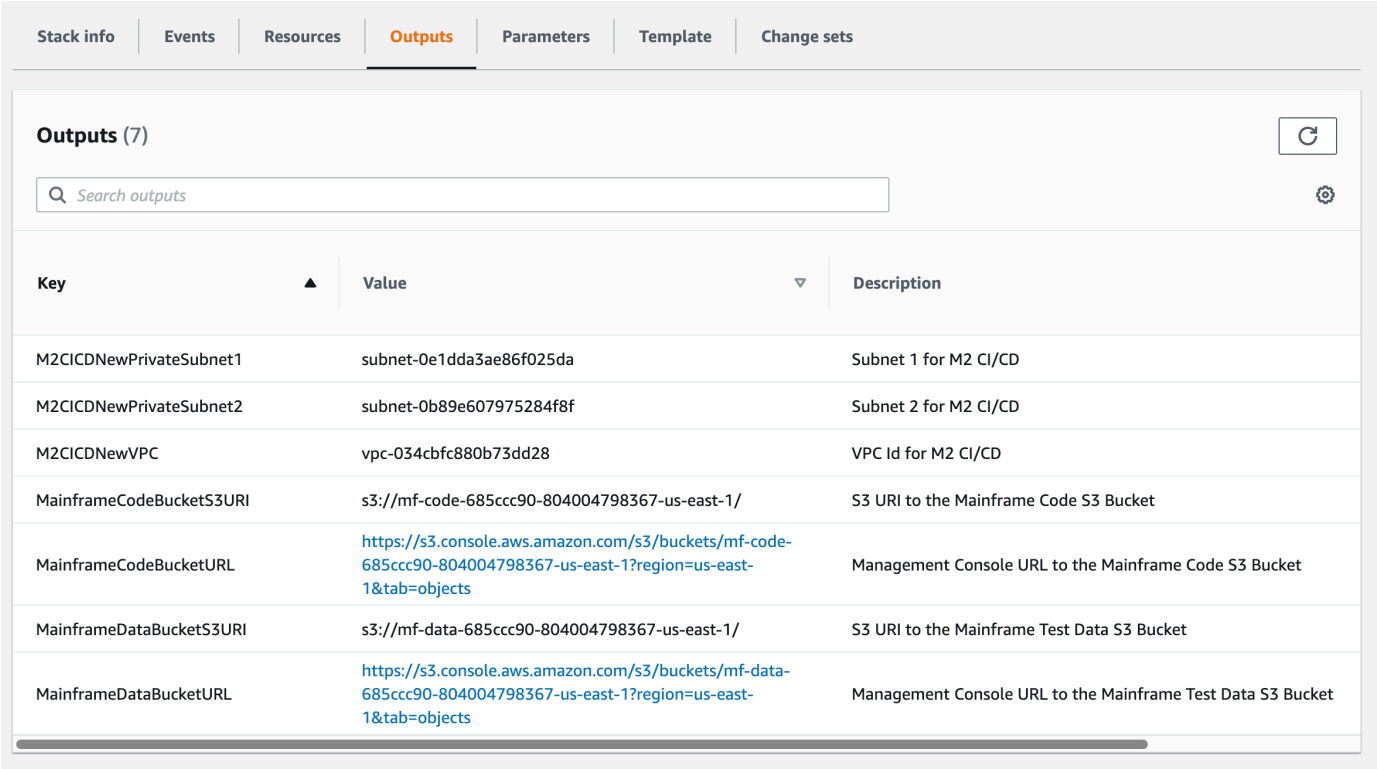
- Provide appropriate values for **Stack name** and parameters for **Networking Configuration**.
- Most parameters in **Deployment Configurations** are pre-populated appropriately so you don't need to modify them. Depending on your AWS Region, change the pipeline AWS CloudFormation template to one of the following Amazon S3 URLs.
  - `https://m2-us-east-1.s3.amazonaws.com/cicd/mf/pipeline.yaml`
  - `https://m2-eu-central-1.s3.eu-central-1.amazonaws.com/cicd/mf/pipeline.yaml`
- Choose **Next**.

> **Note**
>
> Don't change the default parameter values unless you have modified the AWS CloudFormation template yourself.

4. In **Configure stack options**, choose **Next**.
5. In **Capabilities**, choose **I acknowledge that AWS CloudFormation might create IAM resources** to allow permission for AWS CloudFormation to create IAM Role on your behalf. Choose **Create stack**.

> **Note**
>
> It can take 3 to 5 minutes for this stack to be provisioned.

6. After the stack has been created successfully, navigate to the **Outputs** section of the newly provisioned stack. There you'll find the Amazon S3 bucket where you need to upload your mainframe code and dependent files.



Key	Value	Description
M2CICDNewPrivateSubnet1	subnet-0e1dda3ae86f025da	Subnet 1 for M2 CI/CD
M2CICDNewPrivateSubnet2	subnet-0b89e607975284f8f	Subnet 2 for M2 CI/CD
M2CICDNewVPC	vpc-034cbfc880b73dd28	VPC Id for M2 CI/CD
MainframeCodeBucketS3URI	s3://mf-code-685ccc90-804004798367-us-east-1/	S3 URI to the Mainframe Code S3 Bucket
MainframeCodeBucketURL	https://s3.console.aws.amazon.com/s3/buckets/mf-code-685ccc90-804004798367-us-east-1?region=us-east-1&tab=objects	Management Console URL to the Mainframe Code S3 Bucket
MainframeDataBucketS3URI	s3://mf-data-685ccc90-804004798367-us-east-1/	S3 URI to the Mainframe Test Data S3 Bucket
MainframeDataBucketURL	https://s3.console.aws.amazon.com/s3/buckets/mf-data-685ccc90-804004798367-us-east-1?region=us-east-1&tab=objects	Management Console URL to the Mainframe Test Data S3 Bucket

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.
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.

   ```
   repository-config:
     target-repository: bankdemo-repo
     target-branch: main
     commit-message: Initial commit for bankdemo-repo main branch
   ```

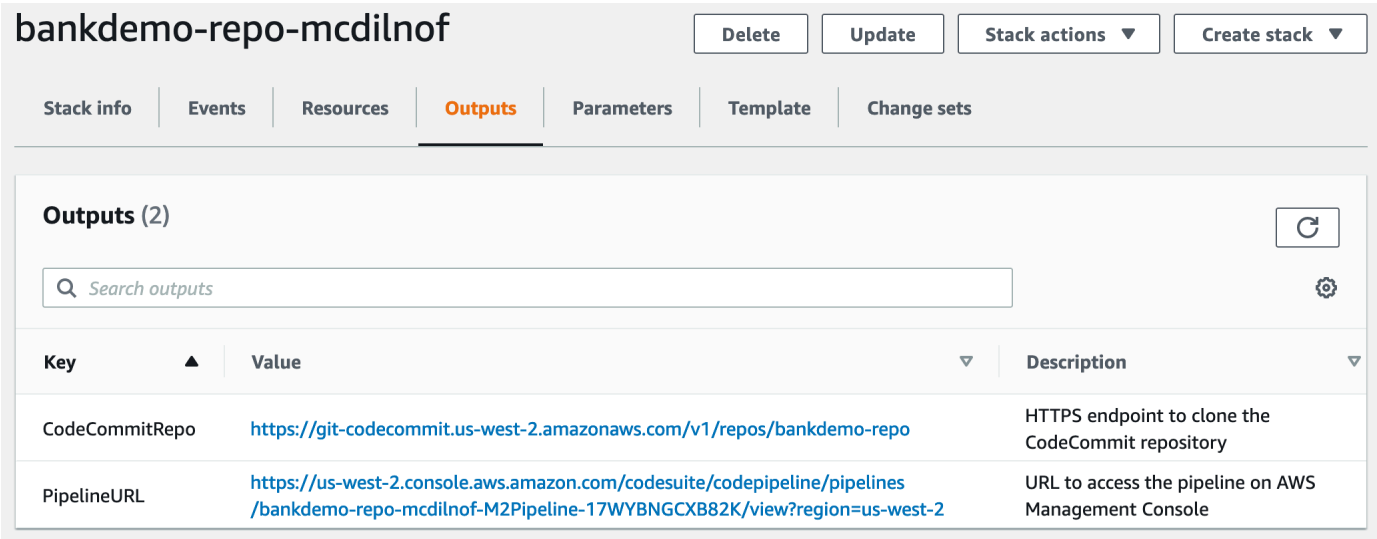
   - Add the email address you want to receive notifications.

```
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. 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'
        - 'ESPGSQLXA.dll'
        - 'ESPGSQLXA64.so'
        - 'ESPGSQLXA64_S.so'
        - 'EXTFH.cfg'
        - 'm2-2021-04-28.normal.json'
        - 'MFDBFH.cfg'
        - 'application-definition-template-config.json'
        readme: |
          # Config Folder
```

```
          This folder contains the application configuration files.
          - 'BANKDEMO.csd'     : CICS Resource definitions export file
          - 'BANKDEMO.json'    : Enterprise Server configuration
          - 'BANKDEMO_ED.json' : Enterprise Server configuration for ED
          - 'dfhdrdat'         : CICS resource definition file
          - 'ESPGSQLXA.dll'    : XA switch module Windows
          - 'ESPGSQLXA64.so'   : XA switch module Linux
          - 'ESPGSQLXA64_S.so' : XA switch module Linux
          - 'EXTFH.cfg'        : Micro Focus File Handler configuration
          - 'm2-2021-04-28.normal.json' : M2 request document
          - 'MFDBFH.cfg'       : Micro Focus Database File Handler
          - 'application-definition-template-config.json' : Application definition for
 M2
    - source:
        subdirs:
        - .settings:
            files:
              - '.bms.mfdirset'
              - '.cbl.mfdirset'
        - copybook:
            files:
              - '*.cpy'
              - '*.inc'
            readme: |
              # Copy folder
              This folder contains the source for COBOL copy books, PLI includes, ...
              - .cpy COBOL copybooks
              - .inc PLI includes
#        - ctlcards:
#            files:
#              - '*.ctl'
#              - 'KBNKSRT1.txt'
#            readme: |
#              # Control Card folder
#              This folder contains the source for Batch Control Cards
#              - .ctl Control Cards
        - ims:
            files:
              - '*.dbd'
              - '*.psb'
            readme: |
              # ims folder
              This folder contains the IMS DB source files with the extensions
              - .dbd for IMS DBD source
```

```
              - .psb for IMS PSB source
        - jcl:
            files:
              - '*.jcl'
              - '*.ctl'
              - 'KBNKSRT1.txt'
              - '*.prc'
            readme: |
              # jcl folder
              This folder contains the JCL source files with the extensions
              - .jcl
#        - proclib:
#            files:
#              - '*.prc'
#            readme: |
#              # proclib folder
#              This folder contains the JCL procedures referenced via PROCLIB
 statements in the JCL with extensions
#              - .prc
        - rdbms:
            files:
              - '*.sql'
            readme: |
              # rdbms folder
              This folder contains any DB2 related source files with extensions
              - .sql for any kind of SQL source
        - screens:
            files:
            - '*.bms'
            - '*.mfs'
            readme: |
              # screens folder
              This folder contains the screens source files with the extensions
              - .bms for CICS BMS screens
              - .mfs for IMS MFS screens
            subdirs:
            - .settings:
                files:
                    - '*.bms.mfdirset'
        - cobol:
            files:
              - '*.cbl'
              - '*.pli'
            readme: |
```

```
              # source folder
              This folder contains the program source files with the extensions
              - .cbl for COBOL source
              - .pli for PLI source
            subdirs:
            - .settings:
                files:
                  - '*.cbl.mfdirset'
    - tests:
        files:
        - 'test_script.py'
        readme: |
          # tests Folder
          This folder contains the application test scripts
pipeline-config:
  alert-notifications:
    - myname@mycompany.com
  approval-notifications:
    - myname@mycompany.com
```

Enterprise Developer AppStream 2.0 Creation

To set up Rocket Enterprise Developer on AppStream 2.0, see Tutorial: Set up Rocket Enterprise Developer on AppStream 2.0.

To connect the CodeCommit repository to Enterprise Developer, use the name specified in `target-repository` in Sample YAML Trigger File config_git.yml.

Enterprise Developer Setup and Test

Topics

- Clone the BankDemo CodeCommit repository in Enterprise Developer
- Create BankDemo mainframe COBOL project and build application
- Create local BankDemo CICS and batch environment for testing
- Start the BANKDEMO server from Enterprise Developer
- Start the Rumba 3270 terminal
- Run a BankDemo transaction
- Stop the BANKDEMO server from Enterprise Developer

Connect to the Enterprise Developer AppStream 2.0 instance you created in Enterprise Developer AppStream 2.0 Creation.

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

1. From Windows Start, launch Micro Focus Rumba+ Desktop / Rumba+ Desktop.
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`.
9. Press CTRL+Shift+Z to clear screen.

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
- Step 1: Perform code analysis for loan calculation

- Step 2: Modify CICS BMS map and COBOL program and test
- Step 3: Add total amount calculation in COBOL program
- Step 4: Commit changes and run CI/CD pipeline

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 Rocket 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` and `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 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
2. Start the Rumba 3270 terminal
3. Run a BankDemo transaction

   In addition you should now also see the text `Total Loan Amount.....:`.
4. Stop the BANKDEMO server from Enterprise Developer

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
2. Start the Rumba 3270 terminal
3. Run a BankDemo transaction

   In addition you should now also see the text `Total Loan Amount.....................: $10230.60`.
4. Stop the BANKDEMO server from Enterprise Developer

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
- Step 2: Extract loan calculation routine to a standalone COBOL program
- Step 3: Commit changes and run the CI/CD pipeline

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.

```
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
2. Start the Rumba 3270 terminal
3. Run a BankDemo transaction

   In addition you should now also see the text `Total Loan Amount.....................: $10230.60`.
4. Stop the BANKDEMO server from Enterprise Developer

> **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\Exercis02\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
2. Start the Rumba 3270 terminal
3. Run a BankDemo transaction

   In addition you should now also see the text `Total Loan Amount.....................: $10230.60`.
4. Stop the BANKDEMO server from Enterprise Developer

> **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\Exercis02\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.mfdirset**.
   - 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:

- Delete the CodePipeline pipeline. For more information, see Delete a pipeline in CodePipeline in the *AWS CodePipeline User Guide*.
- Delete the CodeCommit repository. For more information, see Delete an CodeCommit repository in the *AWS CodeCommit User Guide*.
- Delete the S3; bucket. For more information, see Deleting a bucket in the *Amazon Simple Storage Service User Guide*.
- Delete the AWS CloudFormation stack. For more information, see Deleting a stack on the AWS CloudFormation console in the *AWS CloudFormation User Guide*.

Tutorial: Set up AppStream 2.0 for use with Rocket Enterprise Analyzer and Rocket 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 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 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 Rocket Enterprise Analyzer and Rocket Enterprise Developer tools used with AWS Mainframe Modernization. Rocket Enterprise Analyzer is usually used during the Assess phase and Rocket 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.

Topics

- Prerequisites
- Step 1: Get the AppStream 2.0 images
- Step 2: Create the stack using the AWS CloudFormation template
- Step 3: Create a user in AppStream 2.0
- Step 4: Log in to AppStream 2.0

- Step 5: Verify buckets in Amazon S3 (optional)
- Next steps
- Clean up resources

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*.
- 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.

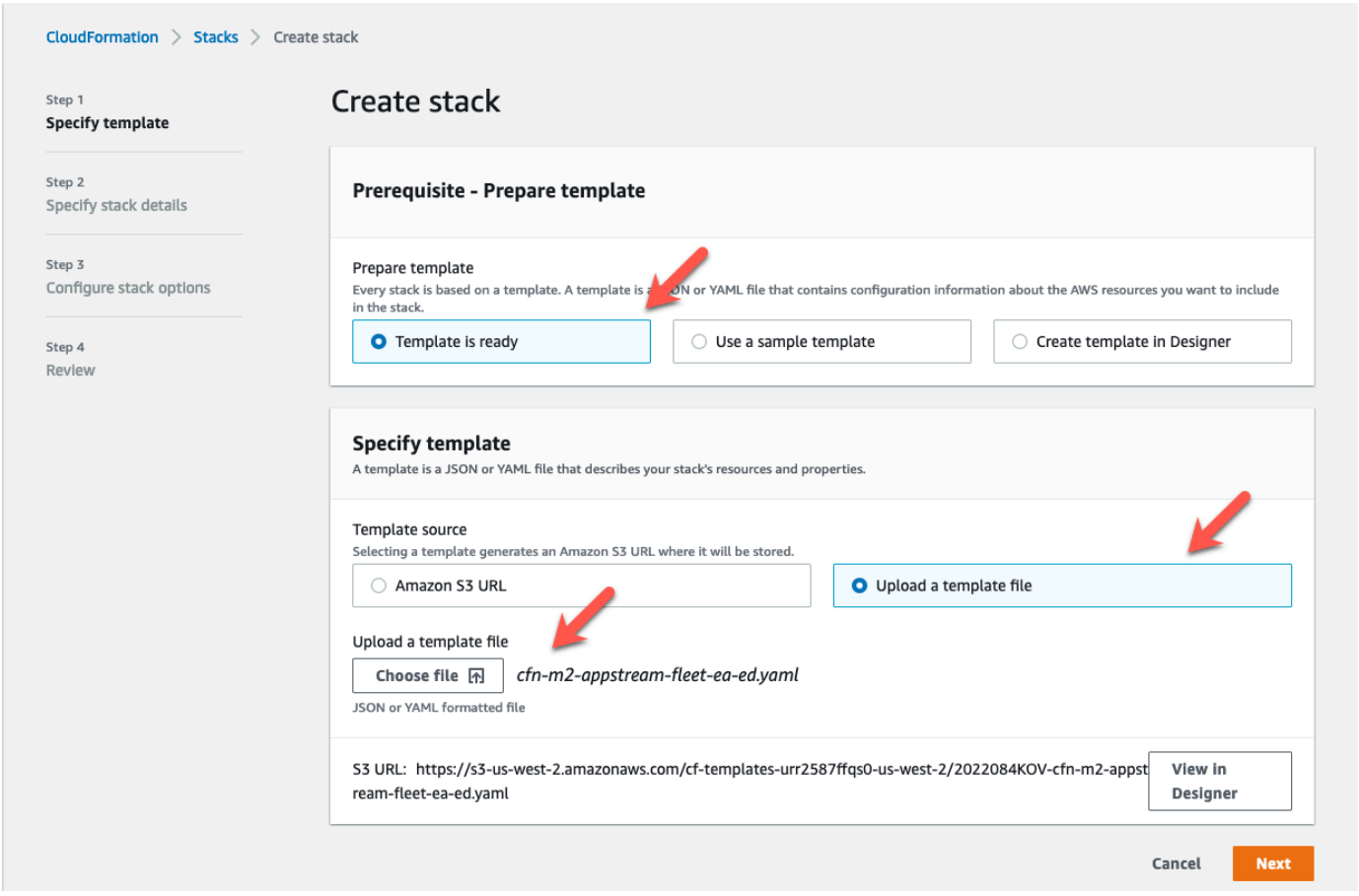
1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the left navigation, choose **Tools**.
3. In **Analysis, development, and build assets**, choose **Share assets with my AWS account**.

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 Rocket Enterprise Analyzer. You can repeat this step later to create another AppStream 2.0 stack and fleet for running Rocket 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*.

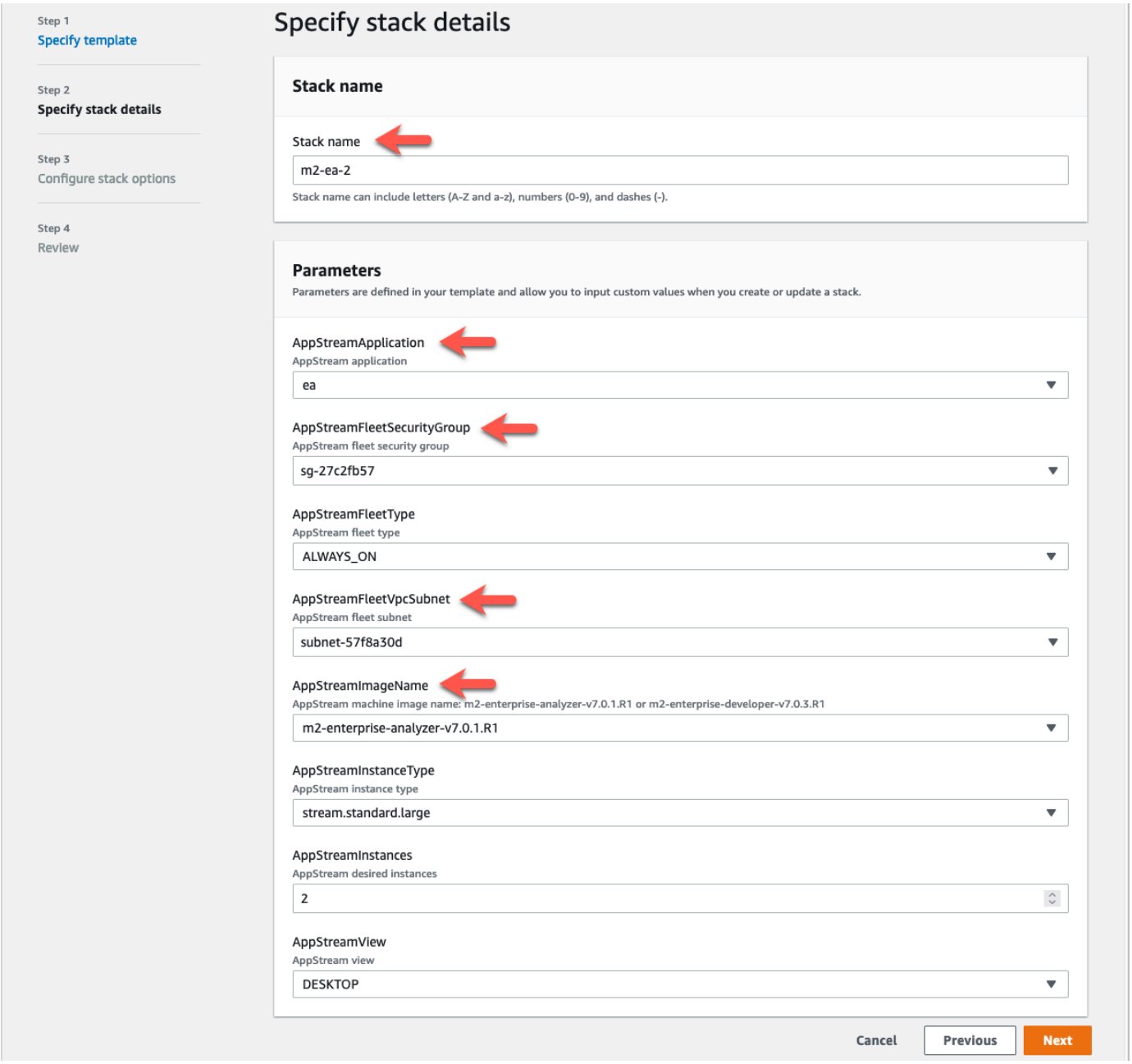
> **Note**
>
> AWS Mainframe Modernization adds an additional fee to the standard AppStream 2.0 pricing for the use of Enterprise Analyzer and Enterprise Developer. For more information, see AWS Mainframe Modernization Pricing.

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 Rocket 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

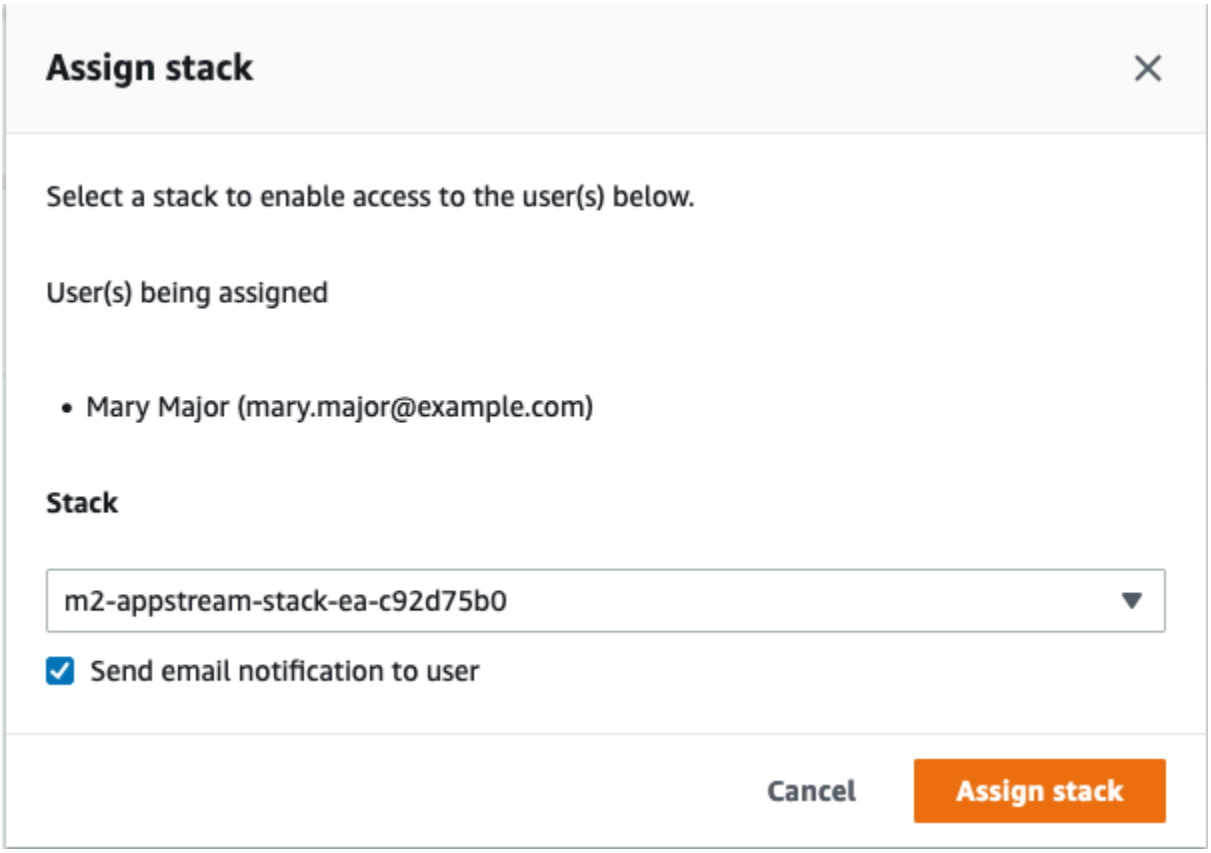
While you are waiting for AWS CloudFormation to finish creating the stack, you can create one or more users in AppStream 2.0. These users are those who will be using Enterprise Analyzer in AppStream 2.0. You will need to specify an email address for each user, and ensure that each user has sufficient 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.

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.

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. You can also set up Enterprise Developer. For more information, see Tutorial: Set up Rocket Enterprise Developer on AppStream 2.0.

Clean up resources

The procedure to clean up the created stack and fleets is described in Create an AppStream 2.0 Fleet and Stack.

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 in the *Amazon AppStream 2.0 Administration Guide*.

Tutorial: Use templates with Rocket Enterprise Developer (formerly Micro Focus Enterprise Developer)

This tutorial describes how to use templates and predefined projects with Rocket 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.

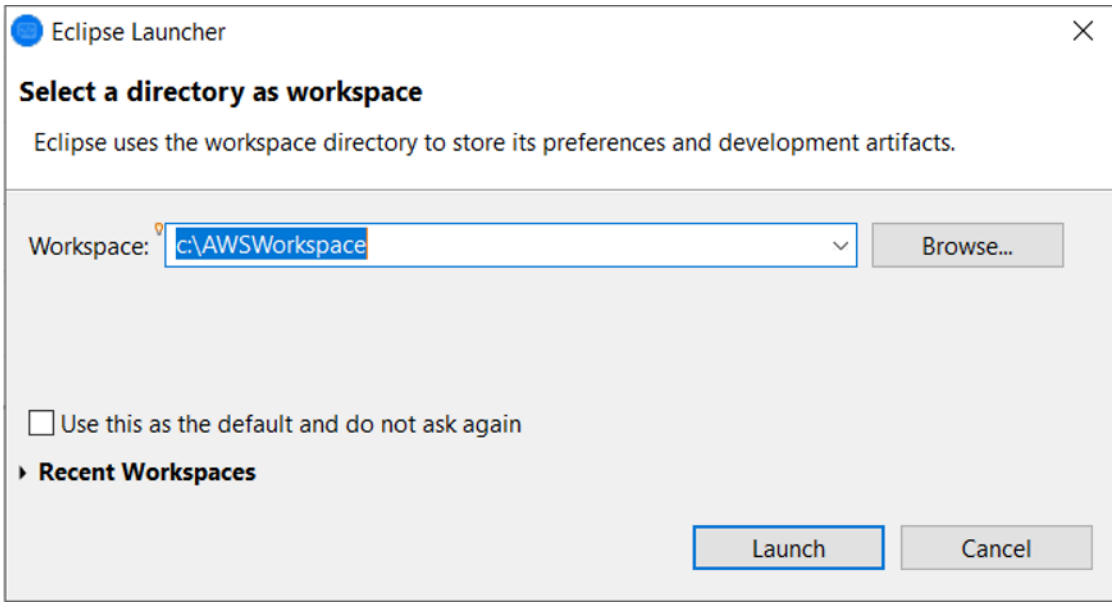
Topics

- Use Case 1 - Using the COBOL Project Template containing source components
- Use Case 2 - Using the COBOL Project Template without source components
- Use Case 3 - Using the pre-defined COBOL project linking to the source folders
- Using the Region Definition JSON Template

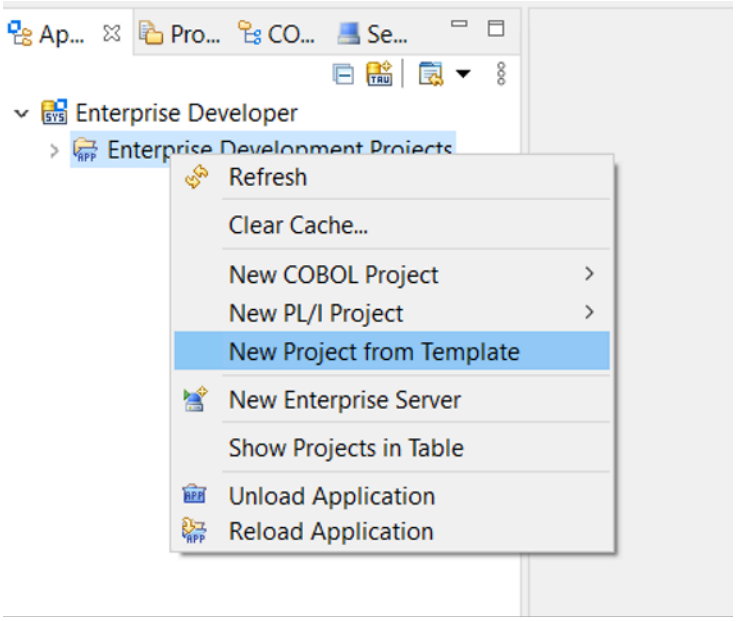
Use Case 1 - Using the COBOL Project Template containing source components

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.

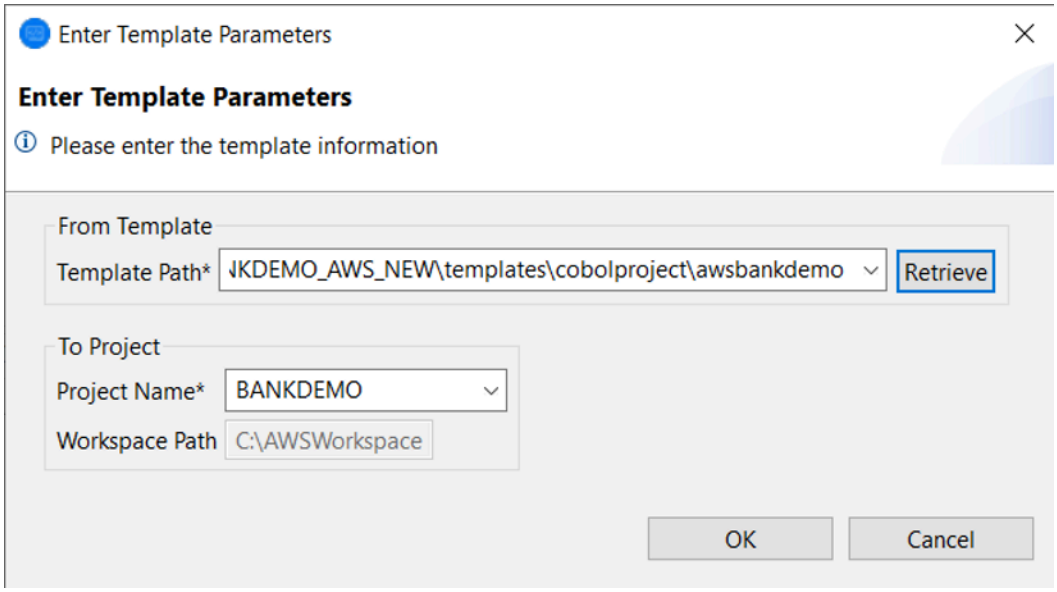


2. Within the **Application Explorer** view, from the **Enterprise Development Project** tree view item, choose **New Project from Template** from the context menu.

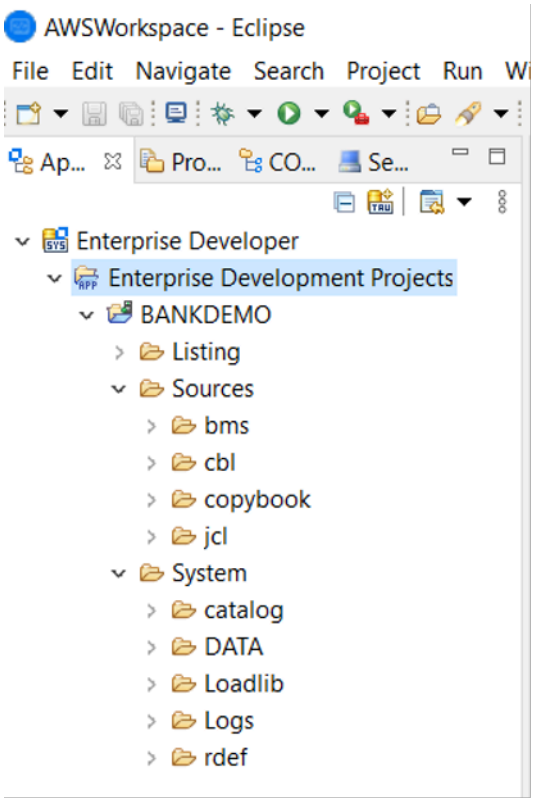


3. Enter the template parameters as shown.

> **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.

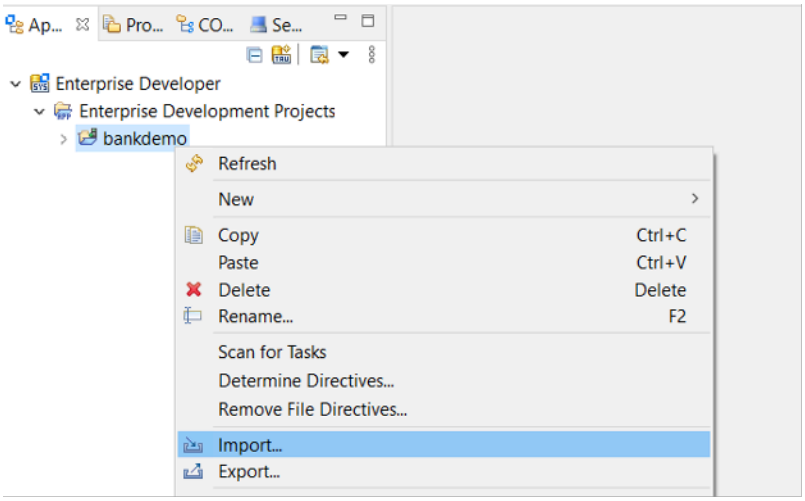
Use Case 2 - Using the COBOL Project Template without source components

Steps 1 to 3 are identical to Use Case 1 - Using the COBOL Project Template containing source components.

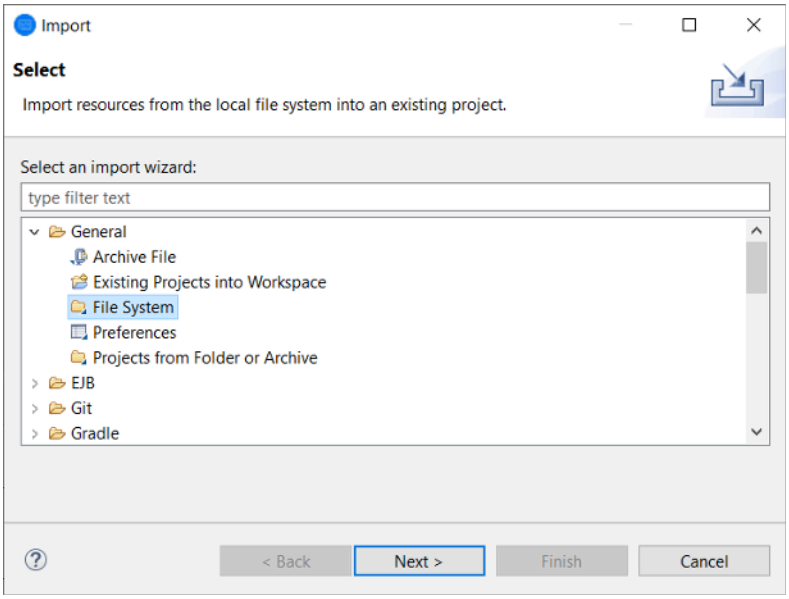
The `System` structure in this use case also contains a complete resource definition file with the required entries for BankDemo, the required catalog with entries added, and the corresponding ASCII data files.

However, the template source structure does not contain any components. You must import these into the project from whatever source repository you are using.

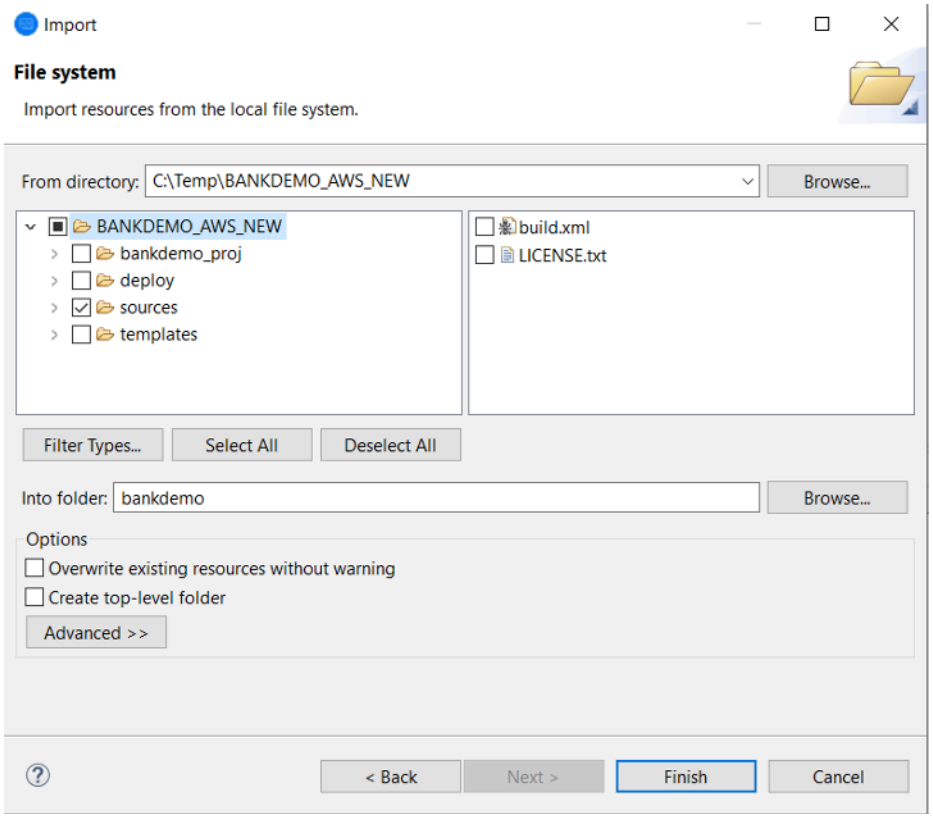
1. Choose the project name. From the related context menu, choose **Import**.



2. From the resulting dialog, under the **General** section, choose **File System** and then choose Next.



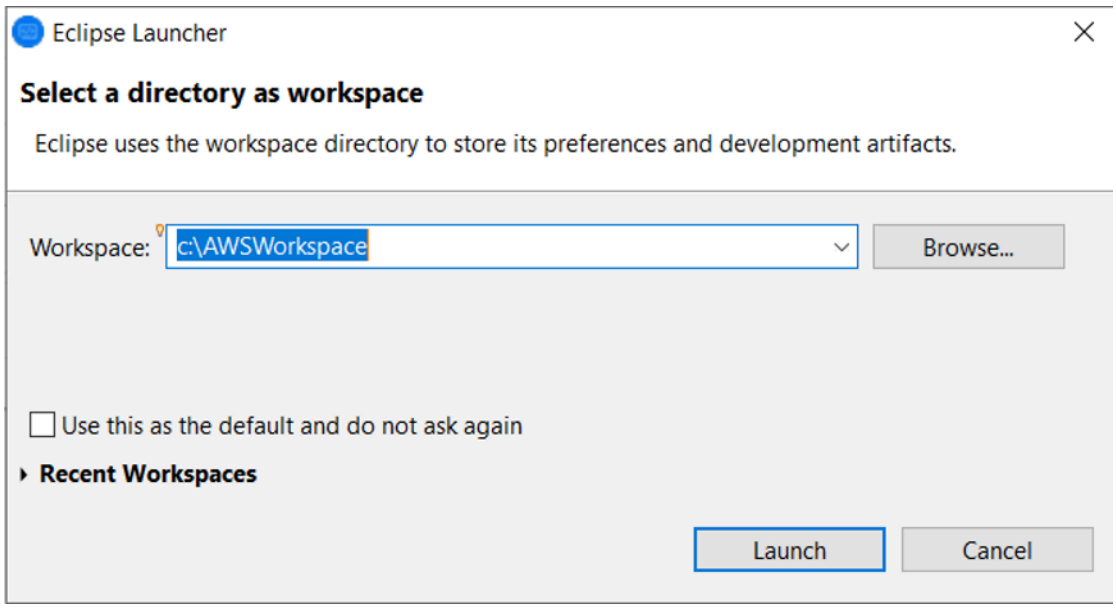
3. Populate the **From directory** field by browsing the file system to point to the repository folder. Choose all the folders you wish to import, such as `sources`. The `Into folder` field will be pre-populated. Choose **Finish**.



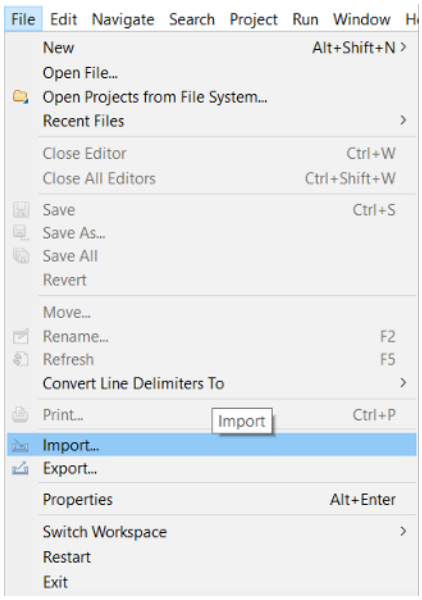
After the source template structure contains all the source items, they are built automatically in Enterprise Developer.

Use Case 3 - Using the pre-defined COBOL project linking to the source folders

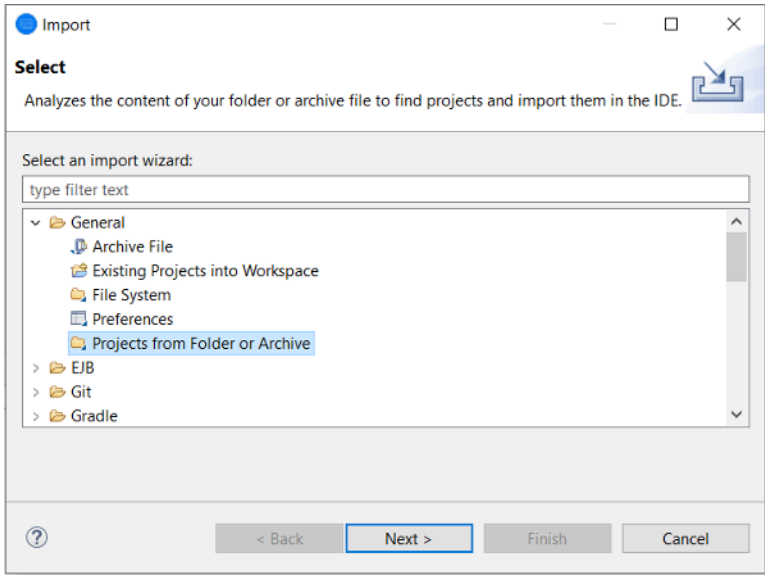
1. Start Enterprise Developer and specify the chosen workspace.



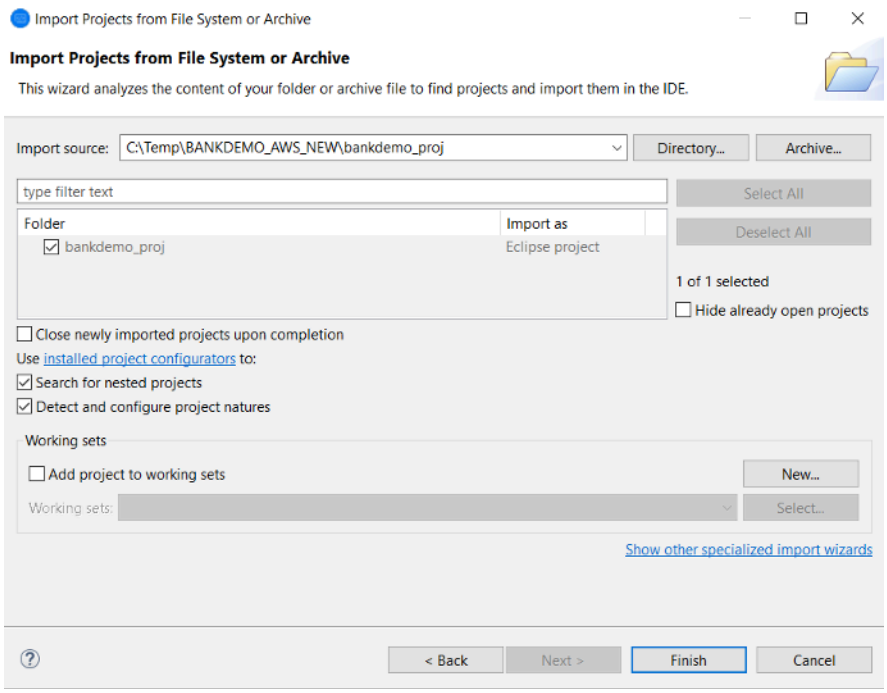
2. From the **File** menu, choose **Import**.



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.

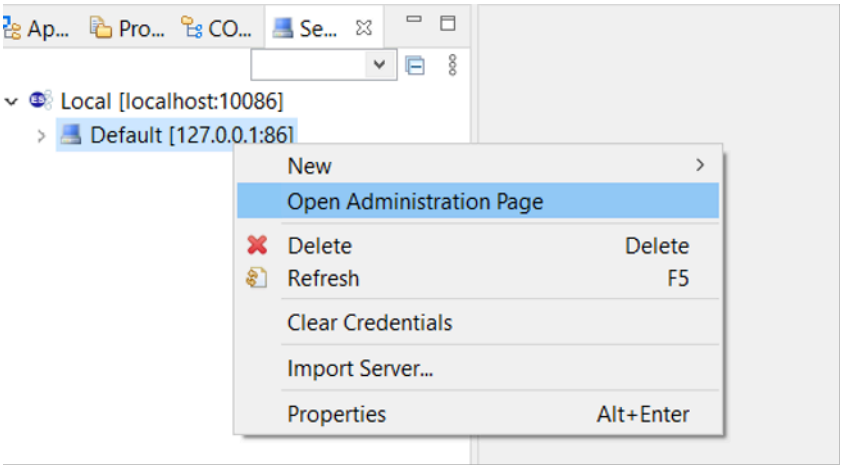


Choose **Finish**.

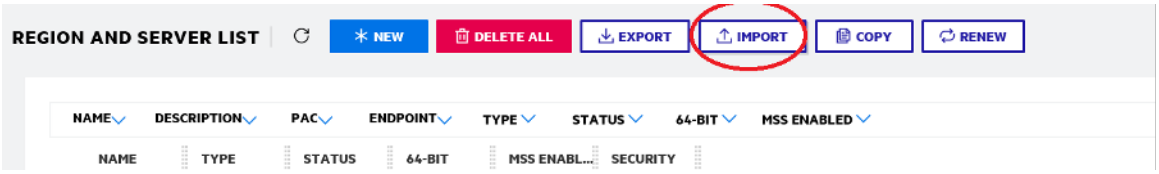
Because the project is populated by the links to the source folder, the code is automatically built.

Using the Region Definition JSON Template

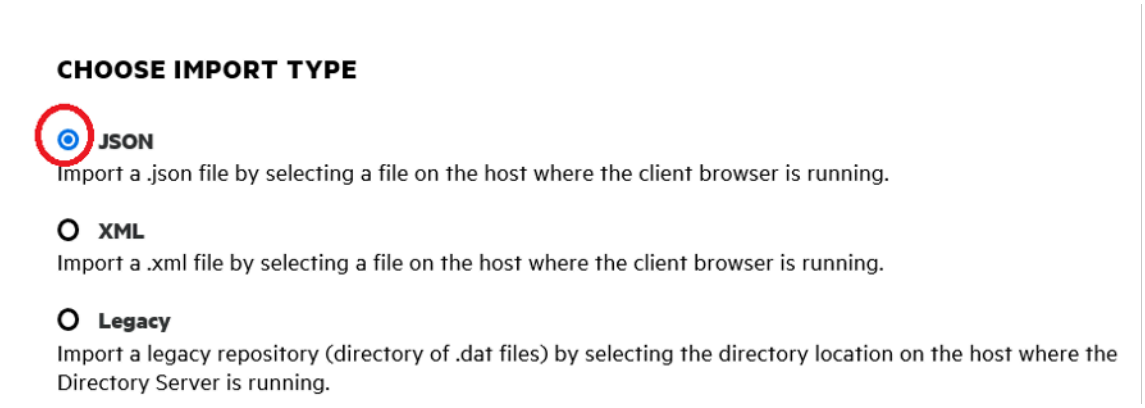
1. Switch to the Server Explorer view. From the related context menu, choose **Open Administration Page**, which starts the default browser.



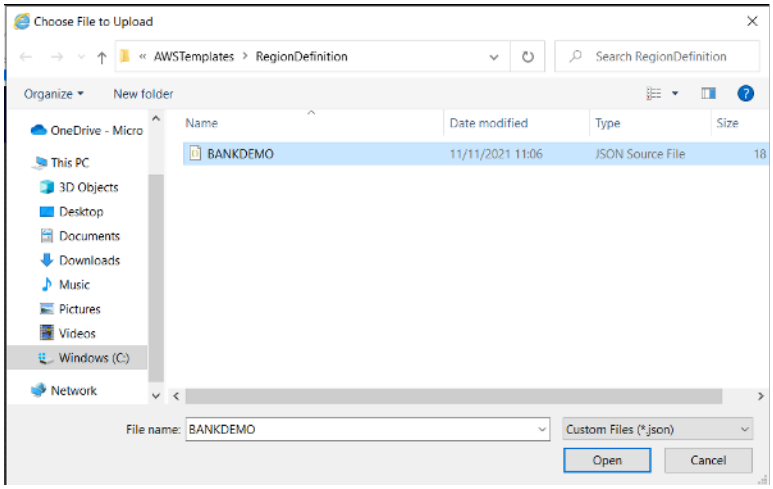
2. From the resulting Enterprise Server Common Web Administration (ESCWA) screen, choose **Import** .



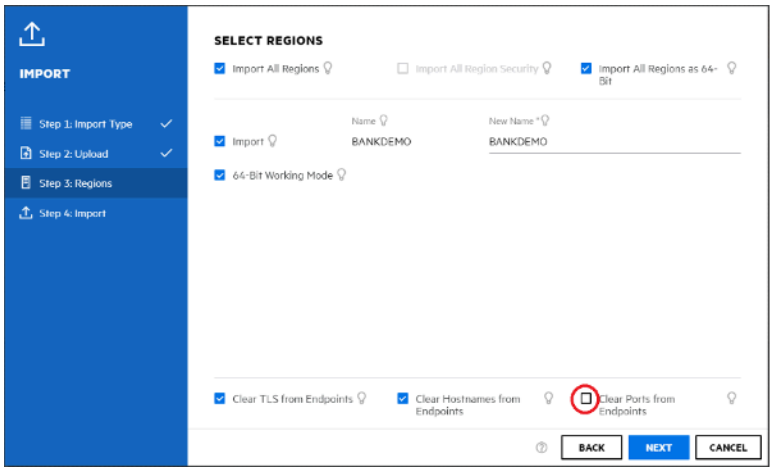
3. Choose the **JSON** import type and choose **Next**.



4. Upload the supplied `BANKDEMO.JSON` file.

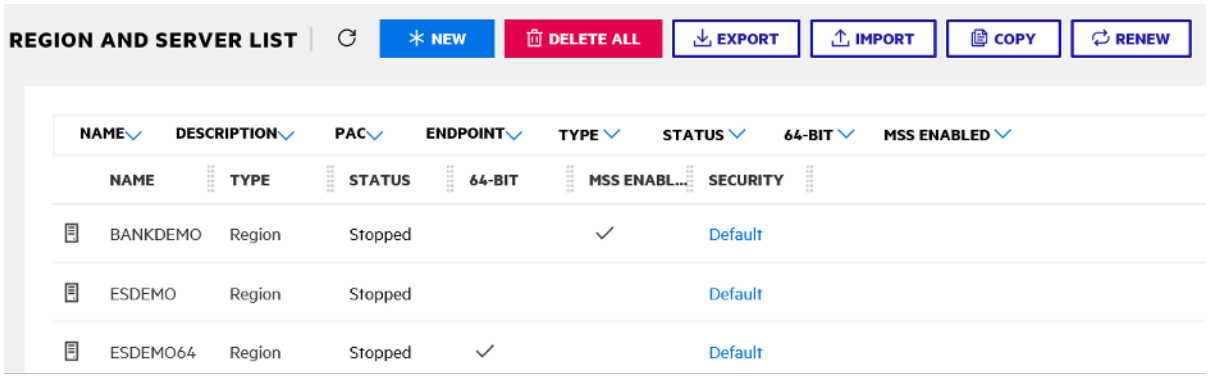


Once selected, choose **Next**.

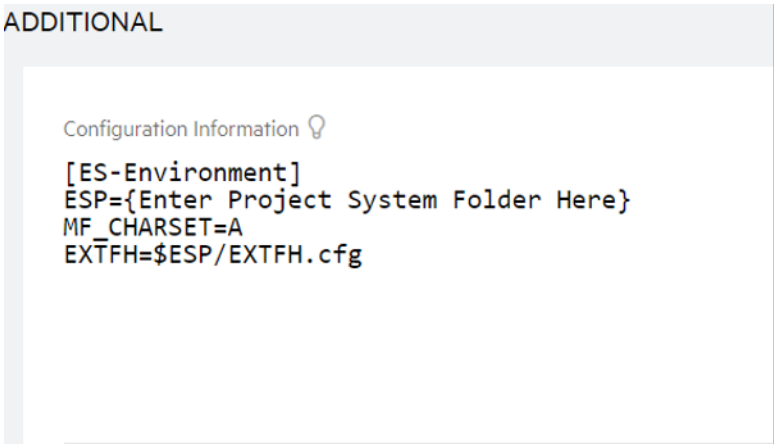


On the **Select Regions** panel, ensure that the **Clear Ports from Endpoints** option is not selected, and then continue to choose **Next** through the panels until the **Perform Import** panel is shown. Then choose **Import** from the left navigation pane.

Finally click **Finish**. The BANKDEMO region will then be added to the server list.

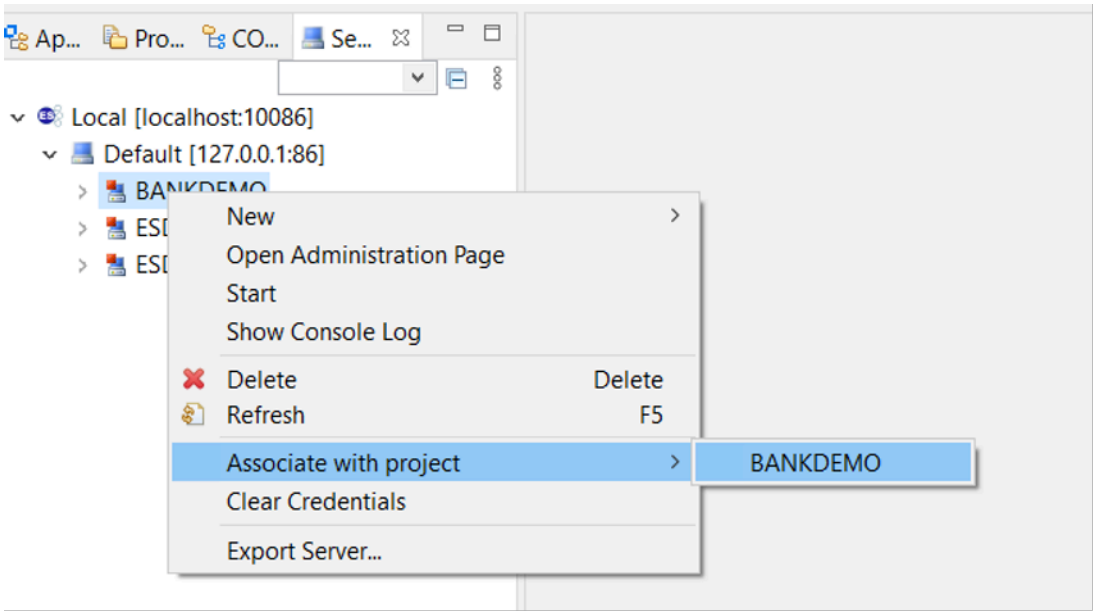


5. Navigate to the **General Properties** for the BANKDEMO region.
6. Scroll to the **Configuration** section.
7. The ESP environment variable needs to be set to the `System` folder relevant to the Eclipse Project created in the previous steps. This should be `workspacefolder/projectname/System`.



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: Set up Enterprise Analyzer on AppStream 2.0

This tutorial describes how to set up Rocket Enterprise Analyzer (formerly 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 Rocket Enterprise Analyzer and Rocket Enterprise Developer.

> **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 Rocket Enterprise Analyzer and Rocket Enterprise Developer.
> 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 Rocket Software (formerly 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
- Step 1: Setup
- Step 2: Create the Amazon S3 based virtual folder on Windows
- Step 3: Create an ODBC source for the Amazon RDS instance
- Subsequent sessions
- Troubleshooting workspace connection
- Clean up resources

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
     |--> RDEF
     |--> transaction
     |--> xa
```

- 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.

```
[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 `amzn-s3-demo-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 bucket 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.

```
:loop
timeout /T 10
"C:\Program Files\rclone\rclone.exe" sync m2-s3:amzn-s3-demo-bucket/your-s3-
folder-key "D:\PhotonUser\My Files\Home Folder\your-local-folder-path" --config "D:
\PhotonUser\My Files\Home Folder\m2-rclone.conf"
```

```
goto :loop
```

4. Open a Windows command prompt, cd to `C:\Users\PhotonUser\My Files\Home Folder` if needed and run `m2-rclone.cmd`. This command script runs a continuous loop, syncing your Amazon S3 bucket and key to the local folder every 10 seconds. You can adjust the time out as needed. You should see the source code of the application located in the Amazon S3 bucket in Windows File Explorer.

To add new files to the set that you are working on or to update existing ones, upload the files to the Amazon S3 bucket and they will be synced to your directory at the next iteration defined in `m2-rclone.cmd`. Similarly, if you want to delete some files, delete them from the Amazon S3 bucket. The next sync operation will delete them from your local directory.

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 tasks.

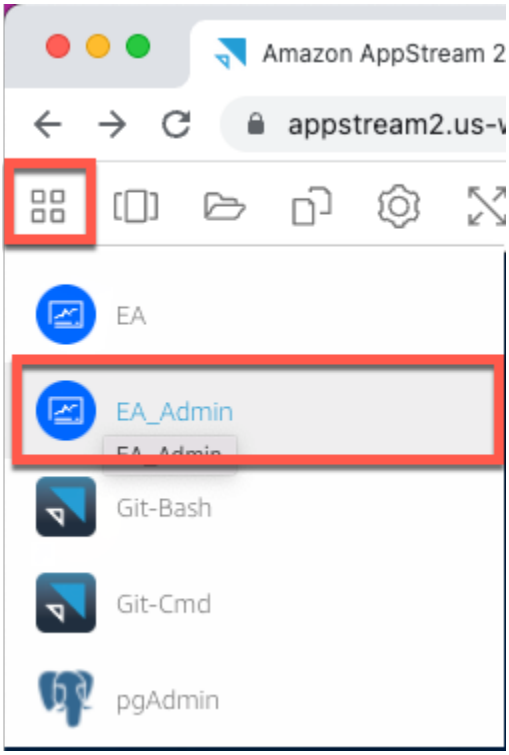
Troubleshooting workspace connection

When you try to reconnect to your Enterprise Analyzer workspace, you might see an error like this:

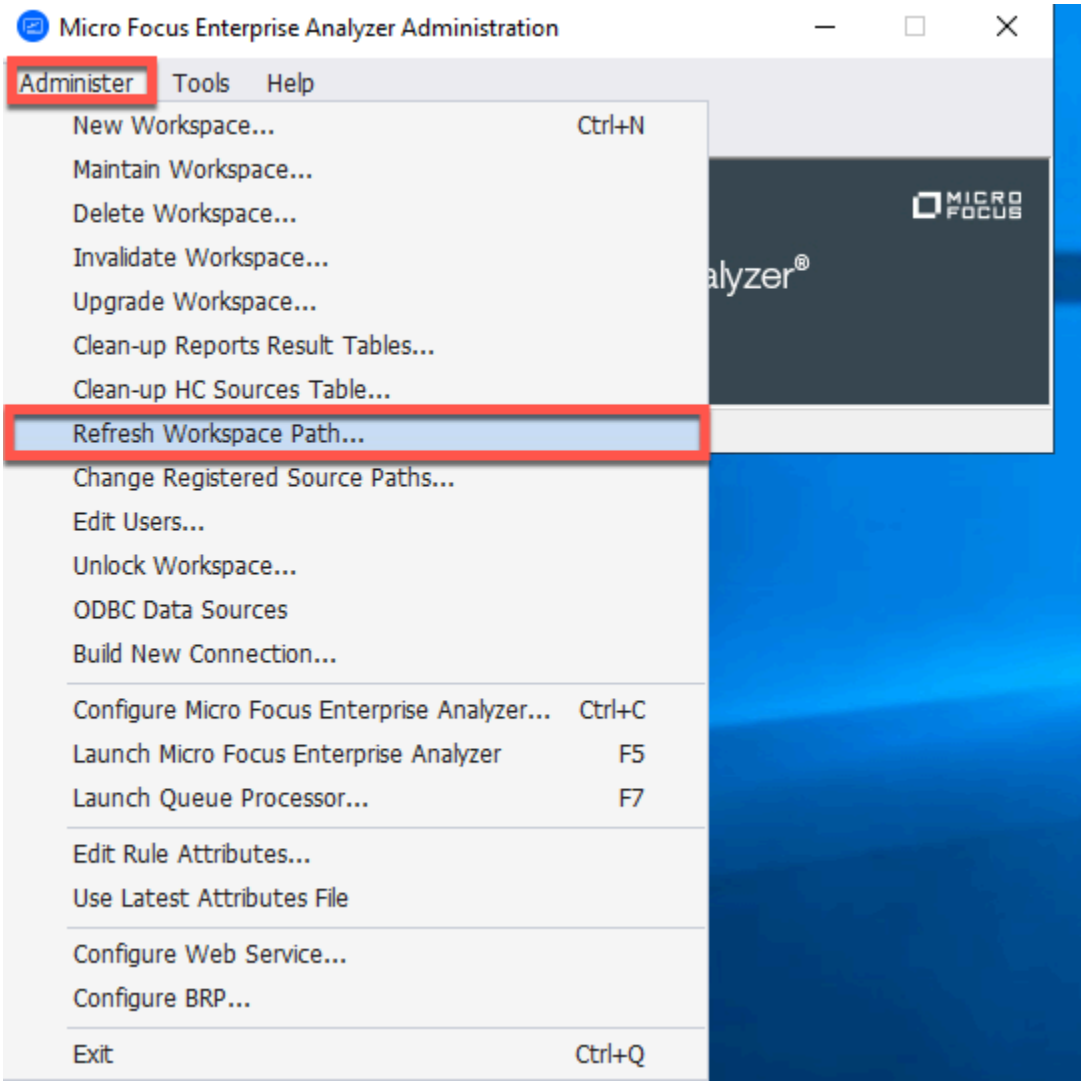
```
Cannot access the workspace directory D:\PhotonUser\My Files\Home Folder\EA_BankDemo.
 The workspace has been created on a non-shared disk of the EC2AMAZ-E6LC33H computer.
 Would you like to correct the workspace directory location?
```

To resolve this issue, choose **OK** to clear the message, and then complete the following steps.

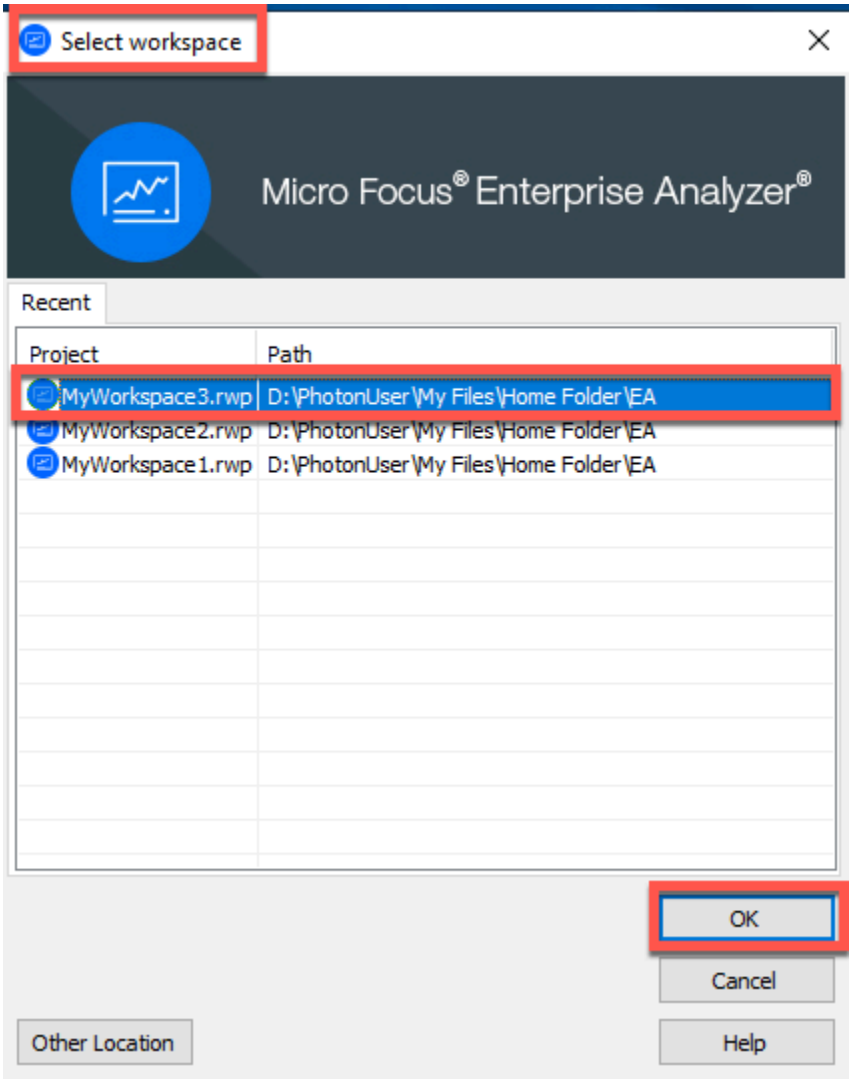
1. In AppStream 2.0, choose the **Launch Application** icon on the toolbar, and then choose **EA_Admin** to start the Enterprise Analyzer Administration tool.



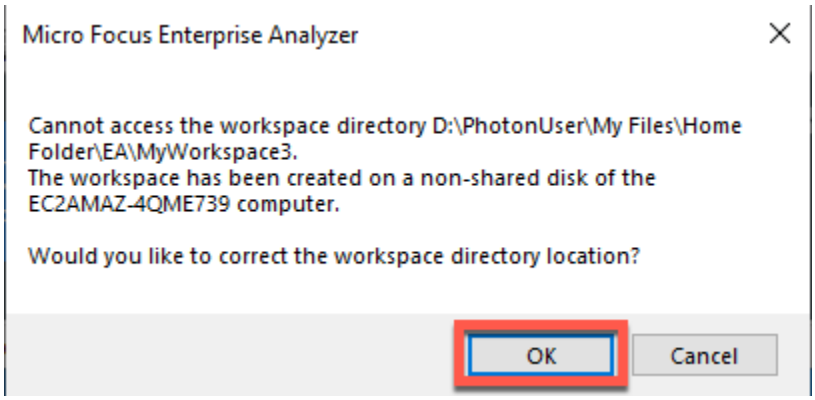
2. From the **Administer** menu, choose **Refresh Workspace Path...**.



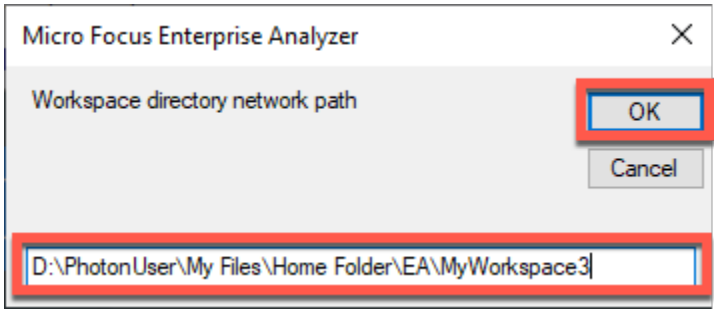
3. Under **Select workspace**, choose the workspace that you want, and then choose **OK**.



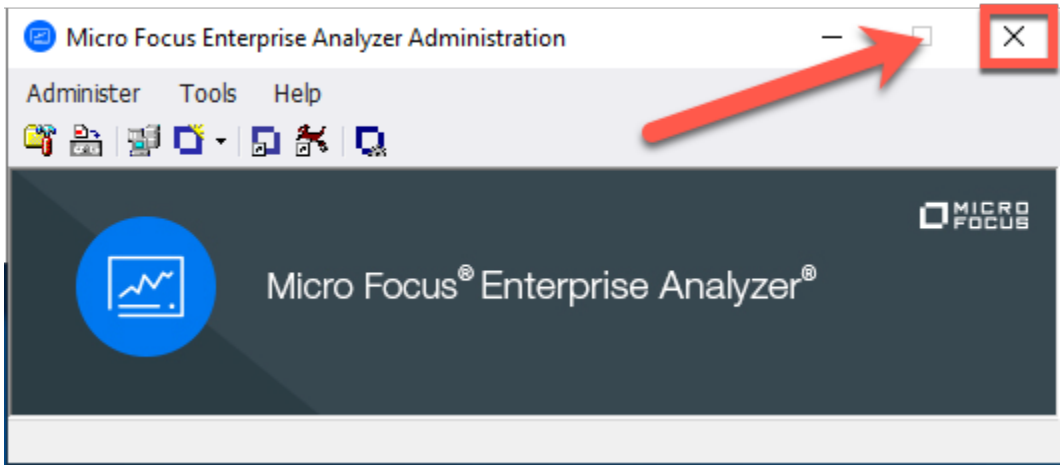
4. Choose **OK** to confirm the error message.



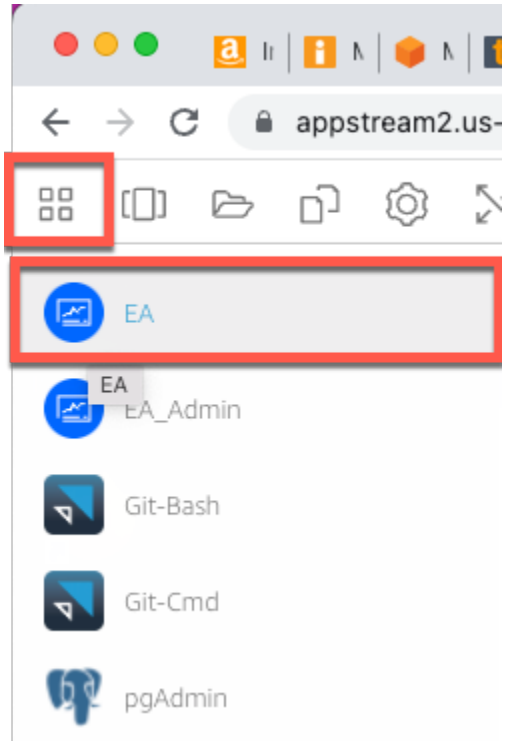
5. Under **Workspace directory network path**, enter the correct path to your workspace, for example, `D:\PhotonUser\My Files\Home Folder\EA\MyWorkspace3`.



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.
- Delete the S3 buckets that you created for this tutorial. For more information, see Deleting a bucket in the *Amazon S3 User Guide*.
- Delete the database that you created for this tutorial. For more information, see Deleting a DB instance.

Tutorial: Set up Rocket Enterprise Developer on AppStream 2.0

This tutorial describes how to set up Rocket Enterprise Developer (formerly 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 Rocket Enterprise Analyzer and Rocket Enterprise Developer.

> **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 Rocket Enterprise Analyzer and Rocket Enterprise Developer.
>
> 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 Rocket Software (formerly Micro Focus) site.

Image contents

In addition to Enterprise Developer itself, the image contains 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.

Topics

- Prerequisites
- Step 1: Setup by individual Enterprise Developer users
- Step 2: Create the Amazon S3-based virtual folder on Windows (optional)
- Step 3: Clone the repository
- Subsequent sessions
- Clean up resources

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 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:
    1. Connect to the AWS console at https://console.aws.amazon.com/iam/.
    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).

> **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.

```
[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 `amzn-s3-demo-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 bucket 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.

```
:loop
timeout /T 10
"C:\Program Files\rclone\rclone.exe" sync m2-s3:amzn-s3-demo-bucket/your-s3-
folder-key "D:\PhotonUser\My Files\Home Folder\your-local-folder-path" --config "D:
\PhotonUser\My Files\Home Folder\m2-rclone.conf"
goto :loop
```

4. Open a Windows command prompt, cd to `C:\Users\PhotonUser\My Files\Home Folder` if needed and run `m2-rclone.cmd`. This command script runs a continuous loop, syncing your Amazon S3 bucket and key to the local folder every 10 seconds. You can adjust the time out as needed. You should see the source code of the application located in the Amazon S3 bucket in Windows File Explorer.

To add new files to the set that you are working on or to update existing ones, upload the files to the Amazon S3 bucket and they will be synced to your directory at the next iteration defined in `m2-rclone.cmd`. Similarly, if you want to delete some files, delete them from the Amazon S3 bucket. The next sync operation will delete them from your local directory.

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 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 `Rclone` 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.

Available 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.
- M2DFUTIL - provides backup, restore, delete, and copy functions on data sets that is similar to the support provided by the mainframe ADRDSSU utility.
- M2RUNCMD - lets you run Rocket Software (formerly Micro Focus) commands, scripts, and system calls directly from JCL.

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
- M2SFTP batch utility
- M2WAIT batch utility
- TXT2PDF batch utility
- M2DFUTIL batch utility
- M2RUNCMD batch utility

Binary Location

These utilities are preinstalled on the Rocket Enterprise Developer (ED) and Rocket Software (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
- Installing dependencies
- Configure M2SFTP for AWS Mainframe Modernization Managed
- Configure M2SFTP for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)
- Sample JCLs
- Putty SFTP (PSFTP) client command reference
- Next steps

Supported platforms

You can use M2SFTP on any of the following platforms:

- AWS Mainframe Modernization Rocket Software (formerly Micro Focus) Managed
- Rocket Software Runtime (on Amazon EC2)
- All variants of Rocket Software Enterprise Developer (ED) and Rocket Software 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 Rocket 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 Rocket 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 `ENVVAR DD` statement.

```
//M2SFTP1 JOB 'M2SFTP1',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*-------------------------------------------------------------------**
//* Sample SFTP JCL step to send a file to SFTP server*
//*-------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2SFTP,
//            PARM='127.0.0.1 (EXIT=99 TIMEOUT 300'
//*
//SYSFTPD  DD  *
RECFM FB
LRECL 80
SBSENDEOL CRLF
MBSENDEOL CRLF
```

```
TRAILINGBLANKS FALSE
/*
//NETRC    DD  *
machine 127.0.0.1 login sftpuser password sftppass
/*
//SYSPRINT DD  SYSOUT=*
//OUTPUT   DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//INPUT    DD  *
type a
locsite notrailingblanks
cd files
put 'AWS.M2.TXT2PDF1.PDF' AWS.M2.TXT2PDF1.pdf
put 'AWS.M2.CARDDEMO.CARDDATA.PS' AWS.M2.CARDDEMO.CARDDATA.PS1.txt
quit
/*
//ENVVAR   DD *
MFFTP_VERBOSE_OUTPUT=ON
MFFTP_KEEP=N
/*
//*
//
```

M2SFTP2.jcl

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.

```
//M2SFTP2 JOB 'M2SFTP2',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*-------------------------------------------------------------------**
//* Sample SFTP JCL step to receive a file from SFTP server*
//*-------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=M2SFTP
//*
//SYSPRINT DD  SYSOUT=*
//OUTPUT   DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//INPUT    DD  *
```

```
open 127.0.0.1
sftpuser
sftppass
cd files
locsite recfm=fb lrecl=150
get AWS.M2.CARDDEMO.CARDDATA.PS.txt +
'AWS.M2.CARDDEMO.CARDDATA.PS2' (replace
quit
/*
//ENVVAR   DD *
MFFTP_VERBOSE_OUTPUT=ON
MFFTP_KEEP=N
/*
//*
//
```

Note

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.

Command	Description
!	Run a local command
bye	Finish your SFTP session
cd	Change your remote working directory
chmod	Change file permissions and modes
close	Finish your SFTP session but do not quit PSFTP
del	Delete files on the remote server

Command	Description
dir	List remote files
exit	Finish your SFTP session
get	Download a file from the server to your local machine
help	Give help
lcd	Change local working directory
lpwd	Print local working directory
ls	List remote files
mget	Download multiple files at once
mkdir	Create directories on the remote server
mput	Upload multiple files at once
mv	Move or rename file(s) on the remote server
open	Connect to a host
put	Upload a file from your local machine to the server
pwd	Print your remote working directory
quit	Finish your SFTP session
reget	Continue downloading files
ren	Move or rename file(s) on the remote server
reput	Continue uploading files
rm	Delete files on the remote server

Command	Description
rmdir	Remove directories on the remote server

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 Rocket Software (formerly 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
- Configure M2WAIT for AWS Mainframe Modernization Managed
- Configure M2WAIT for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)
- Sample JCL

Supported platforms

You can use M2WAIT on any of the following platforms:

- AWS Mainframe Modernization Rocket Software (formerly Micro Focus) Managed
- Rocket Software Runtime (on Amazon EC2)
- All variants of Rocket Software Enterprise Developer (ED) and Rocket Software 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.

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.

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 a text file 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 Rocket Software (formerly Micro Focus) runtime environment.

Topics

- Supported platforms
- Configure TXT2PDF for AWS Mainframe Modernization Managed
- Configure TXT2PDF for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)
- Sample JCL
- Modifications
- References

Supported platforms

You can use TXT2PDF on any of the following platforms:

- AWS Mainframe Modernization Rocket Software Managed
- Rocket Software Runtime (on Amazon EC2)
- All variants of Rocket Enterprise Developer (ED) and Rocket 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)
  - `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.

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 Rocket Software environment variable `MFREXX_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 `MFREXX_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`
   - Windows (32bit): `C:\AWS\M2\MicroFocus\Utilities\32bit`
   - Windows (64bit): `C:\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.

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.

```
//TXT2PDF1 JOB 'TXT2PDF1',CLASS=A,MSGCLASS=X,TIME=1440
//*
//* Copyright Amazon.com, Inc. or its affiliates.*
//* All Rights Reserved.*
//*
//*-------------------------------------------------------------------**
//* PRE DELETE*
//*-------------------------------------------------------------------**
//*
//PREDEL  EXEC PGM=IEFBR14
//*
//DD01     DD DSN=AWS.M2.TXT2PDF1.PDF.VB,
//            DISP=(MOD,DELETE,DELETE)
//*
//DD02     DD DSN=AWS.M2.TXT2PDF1.PDF,
```

```
//            DISP=(MOD,DELETE,DELETE)
//*
//*--------------------------------------------------------------------**
//* CALL TXT2PDF TO CONVERT FROM TEXT TO PDF (VB)*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=IKJEFT1B
//*
//SYSEXEC  DD DISP=SHR,DSN=AWS.M2.REXX.EXEC
//*
//INDD     DD *
1THIS IS THE FIRST LINE ON THE PAGE 1
0THIS IS THE THIRD LINE ON THE PAGE 1
-THIS IS THE   6TH LINE ON THE PAGE 1
THIS IS THE   7TH LINE ON THE PAGE 1
+_____________________________________ - OVERSTRIKE 7TH LINE
1THIS IS THE FIRST LINE ON THE PAGE 2
0THIS IS THE THIRD LINE ON THE PAGE 2
-THIS IS THE   6TH LINE ON THE PAGE 2
THIS IS THE   7TH LINE ON THE PAGE 2
+_____________________________________ - OVERSTRIKE 7TH LINE
/*
//*
//OUTDD    DD DSN=AWS.M2.TXT2PDF1.PDF.VB,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=VB,BLKSIZE=0)
//*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=SYSIN
//*
//SYSIN    DD *
%TXT2PDF BROWSE Y IN DD:INDD +
OUT DD:OUTDD +
CC YES
/*
//*
//*--------------------------------------------------------------------**
//* CONVERT PDF (VB) TO PDF (LSEQ - BYTE STREAM)*
//*--------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=VB2LSEQ
//*
//INFILE   DD DSN=AWS.M2.TXT2PDF1.PDF.VB,DISP=SHR
//*
```

```
//OUTFILE  DD DSN=AWS.M2.TXT2PDF1.PDF,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=LSEQ,BLKSIZE=0)
//*
//SYSOUT   DD SYSOUT=*
//*
//
```

TXT2PDF2.jcl

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=(MOD,DELETE,DELETE)
//*
//DD02     DD DSN=AWS.M2.TXT2PDF2.PDF,
//            DISP=(MOD,DELETE,DELETE)
//*
//*--------------------------------------------------------------------**
//* CALL TXT2PDF TO CONVERT FROM TEXT TO PDF (VB)*
//*--------------------------------------------------------------------**
//*
//STEP01 EXEC PGM=IKJEFT1B
//*
//SYSEXEC  DD DISP=SHR,DSN=AWS.M2.REXX.EXEC
//*
//INDD     DD *
1THIS IS THE FIRST LINE ON THE PAGE 1
0THIS IS THE THIRD LINE ON THE PAGE 1
-THIS IS THE   6TH LINE ON THE PAGE 1
THIS IS THE   7TH LINE ON THE PAGE 1
+_____________________________________ - OVERSTRIKE 7TH LINE
1THIS IS THE FIRST LINE ON THE PAGE 2
```

```
0THIS IS THE THIRD LINE ON THE PAGE 2
-THIS IS THE   6TH LINE ON THE PAGE 2
THIS IS THE   7TH LINE ON THE PAGE 2
+_____________________________________ - OVERSTRIKE 7TH LINE
/*
//*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DDNAME=SYSIN
//*
//SYSIN    DD *
%TXT2PDF BROWSE Y IN DD:INDD +
OUT 'AWS.M2.TXT2PDF2.PDF.VB' +
CC YES
/*
//*
//*--------------------------------------------------------------------**
//* CONVERT PDF (VB) TO PDF (LSEQ - BYTE STREAM)*
//*--------------------------------------------------------------------**
//*
//STEP02 EXEC PGM=VB2LSEQ
//*
//INFILE   DD DSN=AWS.M2.TXT2PDF2.PDF.VB,DISP=SHR
//*
//OUTFILE  DD DSN=AWS.M2.TXT2PDF2.PDF,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(LRECL=256,DSORG=PS,RECFM=LSEQ,BLKSIZE=0)
//*
//SYSOUT   DD SYSOUT=*
//*
//
```

Modifications

To make the TXT2PDF program run on the AWS Mainframe Modernization Rocket Software runtime environment, we made the following changes:

- Changes to the source code to ensure compatibility with the Rocket Software 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

M2DFUTIL batch utility

M2DFUTIL is a JCL utility program that provides backup, restore, delete, and copy functions on datasets, similar to the support provided by the mainframe ADRDSSU utility. This program retains many of the SYSIN parameters from ADRDSSU, which streamlines the process to migrate to this new utility.

Topics

- Supported platforms
- Platform requirements
- Planned future support
- Asset locations
- Configure M2DFUTIL or AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)
- General syntax
- Sample JCLs

Supported platforms

You can use M2DFUTIL on any of the following platforms:

- Rocket Software (formerly Micro Focus) ES on Windows (64 bit and 32 bit)
- Rocket Software ES on Linux (64 bit)

Platform requirements

M2DFUTIL depends on calling a script to perform a regular expression test. On Windows, you must install Windows Services for Linux (WSL) for this script to run.

Planned future support

Features that are not currently available from the mainframe ADRDSSU utility, but are in the future scope include:

- M2 Managed
- VSAM
- COPY support for file name renaming
- RENAME support for RESTORE
- Multiple INCLUDE and EXCLUDE
- BY clause for subselecting by DSORG, CREDT, EXPDT
- MWAIT clause to retry enqueue failures
- S3 storage support for DUMP/RESTORE

Asset locations

The load module for this utility is called `M2DFUTIL.so` on Linux and `M2DFUTIL.dll` on Windows. This load module can be found in the following locations:

- 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`

The script used for regular expression testing is called `compare.sh`. This script can be found in the following locations:

- Linux: `/opt/aws/m2/microfocus/utilities/scripts`
- Windows (32 bit): `C:\AWS\M2\MicroFocus\Utilities\scripts`

Configure M2DFUTIL or AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)

Configure your Enterprise Server region with the following:

- Add the following variables in **[ES-Environment]**

- M2DFUTILS_BASE_LOC - The default location for DUMP output
- M2DFUTILS_SCRIPTPATH - The location of the compare.sh script documented in **Asset Locations**
- M2DFUTILS_VERBOSE - [VERBOSE or NORMAL]. This controls the level of detail in the SYSPRINT output

- Verify that the load module path is added to the JES\Configuration\JES Program Path setting
- Verify that the scripts in the utilities directory have run permissions. You can add a run permission using the chmod + x <script name> command, in the Linux environment

General syntax

DUMP

Provides the ability to copy files from the present cataloged location to a backup location. This location must currently be a file system.

Process

DUMP will perform the following:

1. Create the target location directory.
2. Catalog the target location directory as a PDS member.
3. Determine the files to be included by processing the INCLUDE parameter.
4. Deselect included files by processing the EXCLUDE parameter.
5. Determine if the files being dumped are to be DELETED.
6. Enqueue the files to be processed.
7. Copy the files.
8. Export the copied files cataloged DCB information to a side file in the target location to assist with future RESTORE operations.

Syntax

```
DUMP
TARGET ( TARGET LOCATION  )    -
INCLUDE ( DSN. )
```

```
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ DELETE ]
```

Required parameters

Following are the required parameters for DUMP:

- `SYSPRINT DD NAME` - To contain additional logging information
- `TARGET` - Target location. It can be either:
  - Full path of the dump location
  - Subdirectory name created in the location defined in the **M2DFUTILS_BASE_LOC** variable
- `INCLUDE` - Either a single named DSNAME or a valid mainframe DSN search string
- `EXCLUDE` - Either a single named DSNAME or a valid mainframe DSN search string

Optional parameters

- CANCEL - Cancel if any error occurs. Files that were processed will be retained
- (Default) IGNORE - Ignore any error and process until end
- DELETE - If no ENQ error occurs, then the file is deleted and is uncataloged

DELETE

Provides the ability to mass delete and uncatalog files. Files are not backed up.

Process

DELETE will perform the following:

1. Determine the files to be included by processing the INCLUDE parameter.
2. Deselect included files by processing the EXCLUDE parameter.
3. Enqueue the files to be processed. Setting the disposition to OLD, DELETE, KEEP.

Syntax

```
DELETE
INCLUDE ( DSN )
```

```
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ DELETE ]
```

Required parameters

Following are the required parameters for DELETE:

- `SYSPRINT DD NAME` - To contain additional logging information
- `INCLUDE` - Either a single named DSNAME or a valid mainframe DSN search string
- `EXCLUDE` - Either a single named DSNAME or a valid mainframe DSN search string

Optional parameters

- CANCEL - Cancel if any error occurs. Files that are processed will be retained
- (Default) IGNORE - Ignore any error and process until end

RESTORE

Provides the ability to restore files previously backed up using DUMP. Files are restored to the original cataloged location unless RENAME is used to alter the restored DSNAME.

Process

RESTORE will perform the following:

1. Validate the source location directory.
2. Determine the files to be included by processing the catalog export file.
3. Deselect included files by processing the EXCLUDE parameter.
4. Enqueue the files to be processed.
5. Catalog files that aren't cataloged based on their export information.
6. If a file is already cataloged and the export catalog information is the same, RESTORE will replace the cataloged dataset if the REPLACE option is set.

Syntax

```
RESTORE
SOURCE ( TARGET LOCATION )
```

```
INCLUDE ( DSN )
[ EXCLUDE ( DSN ) ]
[ CANCEL | IGNORE ]
[ REPLACE]
```

Required parameters

Following are the required parameters for RESTORE:

- SYSPRINT DD NAME - To contain additional logging information
- SOURCE - Source location. It can be either:
  - Full path of the dump location
  - Subdirectory name created in the location defined in the **M2DFUTILS_BASE_LOC** variable
- INCLUDE - Either a single named DSNAME or a valid mainframe DSN search string
- EXCLUDE - Either a single named DSNAME or a valid mainframe DSN search string

Optional parameters

- CANCEL - Cancel if any error. Files processed retained
- (Default) IGNORE - Ignore any error and process until end
- REPLACE - If the file being restored is already cataloged and the catalog records are the same, then replace the cataloged file

Sample JCLs

DUMP job

This job will create a subdirectory called TESTDUMP. This is the default backup location specified by the **M2DFUTILS_BASE_LOC** variable. It will create a PDS library for this backup called M2DFUTILS.TESTDUMP. The exported catalog data is stored in a line sequential file in the backup directory called CATDUMP.DAT. All files selected will be copied to this backup directory.

```
//M2DFDMP JOB 'M2DFDMP',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
//SYSPRINT DD DSN=TESTDUMP.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSIN    DD *
```

```
DUMP TARGET(TESTDUMP)               -
     INCLUDE(TEST.FB.FILE*.ABC)     -
 CANCEL
/*
//
```

DELETE job

This job will delete all files from the catalog that match the INCLUDE parameter.

```
/M2DFDEL JOB 'M2DFDEL',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
//SYSPRINT DD DSN=TESTDEL.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
  DELETE                            -
     INCLUDE(TEST.FB.FILE*.ABC)     -
 CANCEL
/*
 //
```

RESTORE job

This job will restore the files that match the INCLUDE parameter from the TESTDUMP backup location. Files that are cataloged will be replaced if the cataloged file is the same as the one in the CATDUMP export and the REPLACE option is specified.

```
//M2DFREST JOB 'M2DFREST',CLASS=A,MSGCLASS=X
//STEP001  EXEC PGM=M2DFUTIL
////SYSPRINT DD DSN=TESTREST.SYSPRINT,
//        DISP=(NEW,CATLG,DELETE),
//        DCB=(RECFM=LSEQ,LRECL=256)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
RESTORE SOURCE(TESTDUMP)            -
     INCLUDE(TEST.FB.FILE*.ABC)     -
 IGNORE
 REPLACE
/*
//
```

M2RUNCMD batch utility

You can use M2RUNCMD, a batch utility program, to run Rocket Software (formerly Micro Focus) commands, scripts, and system calls directly from JCL instead of running them from a terminal or command prompt. The output from the commands is logged to the batch job's spool log.

Topics

- Supported platforms
- Configure M2RUNCMD for AWS Mainframe Modernization runtime on Amazon EC2 (including AppStream 2.0)
- Sample JCLs

Supported platforms

You can use M2RUNCMD on the following platforms:

- Rocket Software Runtime (on Amazon EC2)
- All variants of Rocket Software Enterprise Developer (ED) and Rocket Software Enterprise Server (ES) products.

Configure M2RUNCMD 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 M2RUNCMD as follows.

- Change the Micro Focus JES Program Path to include the binary location for batch utilities. If you must 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`

Sample JCLs

To test the installation, you can use either of the following sample JCLs.

RUNSCRL1.jcl

This sample JCL creates a script and runs it. The first step creates a script called `/tmp/TEST_SCRIPT.sh` and with content from SYSUT1 in-stream data. The second step sets the run permission and runs the script created in the first step. You can also choose to perform only the second step to run already existing Rocket Software and system commands.

```
//RUNSCRL1 JOB 'RUN SCRIPT',CLASS=A,MSGCLASS=X,TIME=1440
//*
//*
//*--------------------------------------------------------------------*
//*  CREATE SCRIPT (LINUX)
//*--------------------------------------------------------------------*
//*
//STEP0010 EXEC PGM=IEBGENER
//*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//*
//SYSUT1   DD *
#!/bin/bash

set -x

## ECHO PATH ENVIRONMNET VARIABLE
echo $PATH

## CLOSE/DISABLE VSAM FILE
casfile -r$ES_SERVER -oc  -ed -dACCTFIL

## OPEN/ENABLE VSAM FILE
casfile -r$ES_SERVER -ooi -ee -dACCTFIL

exit $?
/*
//SYSUT2   DD DSN=&&TEMP,
//            DISP=(NEW,CATLG,DELETE),
//            DCB=(RECFM=LSEQ,LRECL=300,DSORG=PS,BLKSIZE=0)
//*MFE: %PCDSN='/tmp/TEST_SCRIPT.sh'
//*
//*--------------------------------------------------------------------*
//*   RUN SCRIPT (LINUX)                                               *
//*--------------------------------------------------------------------*
```

```
//*
//STEP0020 EXEC PGM=RUNCMD
//*
//SYSOUT  DD  SYSOUT=*
//*
//SYSIN   DD *
*RUN SCRIPT
 sh /tmp/TEST_SCRIPT.sh
/*
//
```

SYSOUT

The output from the command or script that is run, is written into the SYSOUT log. For each carried out command, it displays the command, output, and return code.

```
************ CMD Start ************

CMD_STR: sh /tmp/TEST_SCRIPT.sh

CMD_OUT:


+ echo /opt/microfocus/EnterpriseServer/bin:/sbin:/bin:/usr/sbin:/usr/bin
/opt/microfocus/EnterpriseServer/bin:/sbin:/bin:/usr/sbin:/usr/bin
+ casfile -rMYDEV -oc -ed -dACCTFIL

-Return Code:   0

Highest return code:    0

+ casfile -rMYDEV -ooi -ee -dACCTFIL

-Return Code:   8

Highest return code:    8

+ exit 8


CMD_RC=8
```

```
************  CMD End  ************
```

RUNCMDL1.jcl

This sample JCL uses RUNCMD to run multiple commands.

```
//RUNCMDL1 JOB 'RUN CMD',CLASS=A,MSGCLASS=X,TIME=1440
//*
//*
//*--------------------------------------------------------------------*
//*   RUN SYSTEM COMMANDS                                              *
//*--------------------------------------------------------------------*
//*
//STEP0001 EXEC PGM=RUNCMD
//*
//SYSOUT  DD  SYSOUT=*
//*
//SYSIN   DD *
*LIST DIRECTORY
 ls
*ECHO PATH ENVIRONMNET VARIABLE
 echo $PATH
/*
//
```

File Transfer in AWS Mainframe Modernization

AWS Mainframe Modernization File Transfer lets you transfer and convert mainframe data sets to Amazon S3 for mainframe modernization, migration, and augmentation use cases. It simplifies the process of transferring data sets from your mainframe to the AWS Cloud. Key features include: discovery of source mainframe data sets and artifacts, and scalability and efficiency for faster data transfers to Amazon S3. File Transfer supports various mainframe data set types like sequential, PDS, GDS, GDG, and VSAM KSDS. The service transfers the data sets to an intermediate Amazon S3 bucket, converts them to the specified target code page, and then moves them to your desired target S3 bucket.

Topics

- What is AWS Mainframe Modernization File Transfer?
- Install a File Transfer agent
- Configure a File Transfer agent
- Create data transfer endpoints for File Transfer
- Create transfer tasks in File Transfer
- Tutorial: Getting started with AWS Mainframe Modernization File Transfer
- Supported source and target encodings in AWS Mainframe Modernization File Transfer

What is AWS Mainframe Modernization File Transfer?

With AWS Mainframe Modernization File Transfer, you can transfer and convert datasets and files with a fully managed service to accelerate and simplify modernization, migration, and augmentation use cases to the AWS Mainframe Modernization service and Amazon S3.

Topics

- Benefits of AWS Mainframe Modernization File Transfer
- How AWS Mainframe Modernization File Transfer works

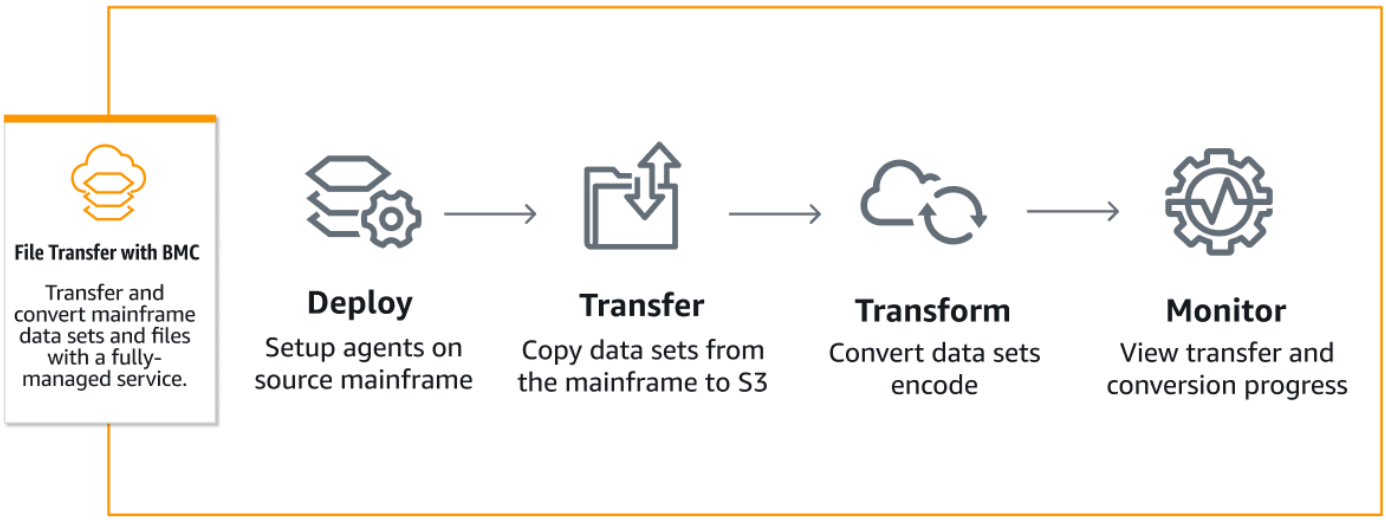
Benefits of AWS Mainframe Modernization File Transfer

AWS Mainframe Modernization File Transfer helps you transfer datasets from mainframe to Amazon S3. Some benefits include:

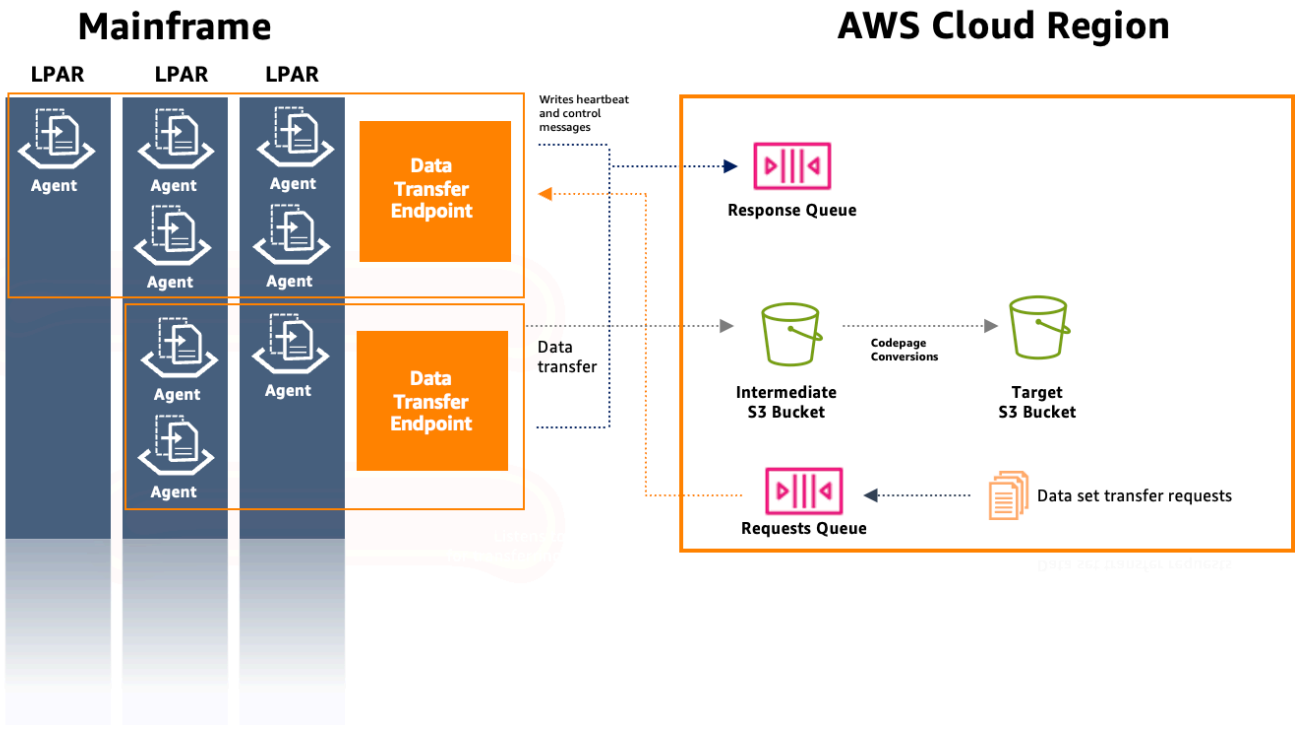
- Discovery of source mainframe datasets and artifacts
- Automated transfers and datasets conversion
- Scalability, efficiency, and speed to achieve faster dataset transfers to AWS

How AWS Mainframe Modernization File Transfer works

The following figure is an overview of how AWS Mainframe Modernization File Transfer works on a conceptual level.



The following figure is an architectural overview of AWS Mainframe Modernization File Transfer feature.



Install a File Transfer agent

You can use this document as a step-by-step guide to install an agent on the source mainframe.

> **Note**
>
> This guide is only for the mainframe system programmers.

Topics

- Step 1: Create a zFS data set for the M2-agent
- Step 2: Format the data set as zFS
- Step 3: Mount the filesystem
- Step 4: Verify the mount
- Step 5: Enter OMVS
- Step 6: Set the agent installation directory environment variable
- Step 7: Set the work directory environment variable

- Step 8: Create the work directory
- Step 9: Copy the agent tar file and copy the work directory
- Step 10: Finish the agent installation

Step 1: Create a zFS data set for the M2-agent

Create a zFS for the M2-agent installation using the JCL (Job Control Language) below:

```
DEFINE   EXEC   PGM=IDCAMS
SYSPRINT DD     SYSOUT=A
SYSIN    DD     *
 DEFINE CLUSTER (NAME(yourhlq.M2AGENT.ZFS) -
 VOLUMES(*) -
 LINEAR CYL(1000 200))
```

Step 2: Format the data set as zFS

After creating the data set, format it as a zFS filesystem.

One way to do that is using the following JCL:

```
FORMAT  EXEC PGM=IOEAGFMT,
PARM=('-aggregate yourhlq.M2AGENT.ZFS -size 1200') ,
SYSPRINT DD  SYSOUT=*
```

Submit this job and check if it completed successfully.

Step 3: Mount the filesystem

To mount the filesystem, use the `MOUNT` command. You can mount the filesystem in command line in ISPF or in batch.

For example:

```
MOUNT FILESYSTEM('yourhlq.M2AGENT.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/usr/lpp/aws/
m2-agent')
```

You will use this mount point in step 6.

> **Note**
>
> Defining the mount path is optional and you should use an existing directory for this.

Step 4: Verify the mount

Verify that the filesystem is correctly mounted using `D OMVS,F` command or by checking within Unix System Service (USS).

Step 5: Enter OMVS

Use the following command to enter OMVS:

```
TSO OMVS
```

Step 6: Set the agent installation directory environment variable

Use the following command to set the agent installation directory environment:

```
export AGENT_DIR=/usr/lpp/aws/m2-agent
```

> **Note**
>
> Mount point is defined in step 3.

Step 7: Set the work directory environment variable

Use the following command to set the work directory environment variable:

```
export WORK_DIR=$AGENT_DIR/tmp
```

Step 8: Create the work directory

Use the following command to set the work directory environment:

```
mkdir -p $WORK_DIR
```

Step 9: Copy the agent tar file and copy the work directory

Download the agent tar file from AWS using the M2 agent link.

The transfer mechanism will depend on your environment, but make sure that the tar file is transferred in binary mode.

Step 10: Finish the agent installation

Follow these steps to finish the agent installation.

1. Set the m2-agent version environment variable to the version currently being installed using the following command:

   ```
   export M2_AGENT_VERSION=1.0.0
   ```

2. Extract the agent tar package using the following command:

   ```
   tar -xpf m2-agent-$M2_AGENT_VERSION.tar -C $AGENT_DIR
   ```

3. Create a `current-version` symbolic link to the current agent installation directory with the following command:

   ```
   ln -s $AGENT_DIR/m2-agent-v$M2_AGENT_VERSION $AGENT_DIR/current-version
   ```

4. Update and submit CPY#PDS to create the File Transfer agent data sets.

   > **Note**
   >
   > JCL uses the `SYS2.AWS.M2` HLQ.

   To create the File Transfer agent, update the three symbolic variables HLQ (High level qualifier), `VOLSER`, and `AGNTPATH` to be used later in the JCL:

   ```
   oedit $AGENT_DIR/current-version/installation/CPY#PDS
   ```

> **Note**
>
> This JCL is tailored for setting up certain aspects of the agent installation on the mainframe. It allocates necessary data sets and then copies specific files from the Unix filesystem to these data sets.

Configure a File Transfer agent

Once you have installed a file transfer agent, follow these steps to configure the agent. If you need to install a new agent, follow instructions on the the section called “Install a File Transfer agent” page.

Topics

- Step 1: Configure permissions and Started Task Control (STC)
- Step 2: Create Amazon S3 buckets
- Step 3: Create an AWS KMS customer managed key for encryption
- Step 4: Create an AWS Secrets Manager secret for the mainframe credentials
- Step 5: Create an IAM policy
- Step 6: Create an IAM user with long-term access credentials
- Step 7: Create an IAM role for the agent to assume
- Step 8: Agent configuration

Step 1: Configure permissions and Started Task Control (STC)

1. Update and submit one of `SYS2.AWS.M2.SAMPLIB(SEC#RACF)` (for setting up RACF permissions) or `SYS2.AWS.M2.SAMPLIB(SEC#TSS)` (for setting up TSS permissions) in accordance with their instructions. These members were created by the previous `CPY#PDS` step.

   > **Note**
   >
   > `SYS2.AWS.M2` should be replaced with the high-level qualifier (HLQ) chosen during installation.

2. Update the PWD export in the `SYS2.AWS.M2.SAMPLIB(M2AGENT)` STC JCL, if the default File Transfer agent directory path(`/usr/lpp/aws/m2-agent`) was changed.
3. Update the PROC according to your site standards:
    a. Update the PROC card per your installation requirements.
    b. Update the STEPLIB with the `M2 LOADLIB PDSE ALIAS`.
    c. Edit PWD to point the agent installation path (only this is included).
    d. Update `JAVA_HOME` if required.
4. Update and copy the `SYS2.AWS.M2.SAMPLIB(M2AGENT)` JCL to `SYS1.PROCLIB` or a one of the PROCLIBs in your `PROCLIB` concatenation.
5. Add `SYS2.AWS.M2.LOADLIB` to the APF list using the following command:

```
SETPROG APF ADD DSNAME(SYS2.AWS.M2.LOADLIB) SMS
```

6. Set the agent's group and owner to the agent user/group (M2USER/M2GROUP). Use the following command in the OMVS:

```
chown -R M2USER:M2GROUP $AGENT_DIR/current-version
```

> **Note**
>
> Edit the M2USER and M2GROUP with the names you used in the security definitions job.

Step 2: Create Amazon S3 buckets

AWS Mainframe Modernization File Transfer requires an intermediate Amazon S3 bucket as a work area. We recommend creating a bucket specifically for this.

Optionally, create a new target Amazon S3 bucket for the transferred data sets. Otherwise you can also use your existing Amazon S3 bucket. For more information on creating Amazon S3 buckets, see Creating a bucket.

Step 3: Create an AWS KMS customer managed key for encryption

To create a customer managed key in AWS KMS

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. Choose **Customer managed keys** in left navigation pane.
3. Choose **Create key**.
4. Under **Configure key**, choose **Key type** as **Symmetric**, and **Key usage** as **Encrypt and decrypt**. Use other default configurations.
5. Choose **Next**.
6. In **Add labels**, add Alias and description for your key.
7. Choose **Next**.
8. Under **Define key administrative permissions**, choose at least one IAM user and role who administers this key.
9. Choose **Next**.
10. Optionally, under **Define key administrative permissions**, choose at least one IAM user and role who can use this key.
11. Choose **Next**.
12. In **Edit key policy** section, choose **Edit**, and add the following syntax to the **Key policy**. This allows the AWS Mainframe Modernization service to read and use these keys for encryption/decryption.

> **Important**
>
> Add the statement to the existing statements. Don't replace what's already in the policy.

```
{
    "Sid" : "Enable AWS M2 File Transfer Permissions",
    "Effect" : "Allow",
    "Principal" : {
        "Service" : "m2.amazonaws.com"
    },
    "Action" : [
        "kms:Encrypt",
```

```
        "kms:Decrypt"
    ],
   "Resource" : "*"
},
```

13. Choose **Next**.
14. On the **Review** page, check all the details, and choose **Finish**.

Copy and save the ARN for the customer managed key by opening the newly created KMS key. It will be used in the policy later.

Step 4: Create an AWS Secrets Manager secret for the mainframe credentials

Mainframe credentials are required to access the data sets to be transferred and these must be stored as an AWS Secrets Manager secret.

To create an AWS Secrets Manager secret

1. Open Secrets manager console at https://console.aws.amazon.com/secretsmanager.
2. Choose **Store a new secret**.
3. In **Choose Secret type**, choose **Other type of secret**.
4. Use the key value `userId` for the mainframe userId that has access to the data sets.. Use the key value `password` for the password field.
5. For **Encryption Key**, choose the AWS customer managed key created earlier.
6. Choose **Next**.
7. On the **Configure secret** page, provide a name and description.
8. On the same page, edit the **Resource permissions**, and use the following resource policy so the AWS Mainframe Modernization service can access it.

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

```
                "secretsmanager:DescribeSecret" ],
    "Resource" : "*"
  } ]
}
```

9. Choose **Save** to save the updated permissions.
10. Choose **Next**.
11. Skip through **Configure rotations** page, and choose **Next**.
12. On the **Review** page, check all configurations and choose **Store** to save the secret.

> **Important**
>
> The `userId` and `password` secret keys are case-sensitive and must be entered as shown.

Step 5: Create an IAM policy

To create a new policy with the permissions required for the agent

1. Open the IAM console at https://console.aws.amazon.com/iam.
2. Choose **Policies** under **Access management**.
3. Choose **Create policy**.
4. On **Specify permissions** page, under **Policy editor**, switch from the Visual editor to the JSON editor and replace the contents with the following template:
5.

```
{
"Version": "2012-10-17",
    "Statement": [
        {
         "Sid": "FileTransferAgentSQSReceive",
            "Effect": "Allow",
            "Action": [
                "sqs:DeleteMessage",
                "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:*:111122223333:m2-*-request-queue.fifo"
        },
        {
         "Sid": "FileTransferAgentSQSSend",
```

```
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource": "arn:aws:sqs:*:111122223333:m2-*-response-queue.fifo"
        },
        {
         "Sid": "FileTransferWorkingS3",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "<file-transfer-endpoint-intermediate-bucket-arn>/*"
        },
        {
         "Sid": "FileTransferAgentKMSDecrypt",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "<kms-key-arn>"
        }
    ]
}
```

6. Replace the 111122223333 in the request-queue and response-queue ARN's with your account.

> **Note**
>
> These are wildcard ARN's that match the two Amazon SQS queues created during the data transfer endpoint initialization. After creating a File Transfer endpoint, optionally replace these ARN's with the actual values from Amazon SQS.

7. Replace `file-transfer-endpoint-intermediate-bucket-arn` with the ARN of the transfer bucket created earlier. Leave the "/*" wildcard at the end.
8. Replace `kms-key-arn` with the ARN of the AWS KMS key created earlier.
9. Choose **Next**.
10. On the **Review and create** page, add the Policy name and description.
11. Choose **Create policy**.

Step 6: Create an IAM user with long-term access credentials

Create an IAM user that allows the mainframe agent to connect to your AWS account. The agent will connect with this user and then assume a role you define with permissions to use Amazon SQS response and request queues and to save datasets to Amazon S3 buckets.

To create this IAM user

1. Navigate to the IAM console at https://console.aws.amazon.com/iam.
2. Choose **Users** under **Access management**.
3. Choose **Create user**.
4. Add a meaningful **User name** under **User details**. For example, `Configure-ft-agent`.
5. Choose **Next**.
6. In the **Permissions options**, choose the **Attach policies directly** option but do not attach any permissions policies. These permissions will be managed by a role that will be attached.
7. Choose **Next**.
8. Review the details, and choose **Create user**.
9. Once the user is created, choose the user and open **Security credentials** tab.
10. Under **Access keys**, choose **Create access key**.
11. Then, choose **Other** when prompted for Use case.
12. Choose **Next**.
13. Optionally, you can set description tag such as, `Access key for configuring file transfer agent`.
14. Choose **Create access key**.
15. Copy, and securely save the generated **Access key**, and **Secret access key**. These will be used later.

For more information on creating IAM access key, see Managing access keys for IAM users.

> **Important**
>
> Save the **Access key** and **Secret access key** displayed on the last page of the access key creation wizard, before choosing **Done**. These keys are used to configure the mainframe agent, and cannot be retrieved later.

> **Note**
>
> Save the IAM user ARN used to set up a trust relationship with an IAM role.

Step 7: Create an IAM role for the agent to assume

To create a new IAM role for the agent

1. Choose **Roles** in the IAM console at https://console.aws.amazon.com/iam.
2. Choose **Create role**.
3. On the **Select trusted entity** page, choose **Custom trust policy** for the **Trusted entity type**.
4. Replace the Custom trust policy with the following and replace `<iam-user-arn>` with the ARN of the user created earlier.

```
{
    "Version": "2012-10-17",
    "Statement": [ {
         "Sid": "FileTransferAgent",
         "Effect": "Allow",
         "Principal": {
            "AWS": "<IAM-User-arn>"
         },
         "Action": "sts:AssumeRole"
    } ]
}
```

5. Choose **Next**.
6. In **Add Permissions**, filter for the **Policy name** you created earlier and choose it.
7. Choose **Next**.
8. Name the role, and choose **Create Role**.

> **Note**
>
> Save the *role name*, which you will use later to configure the mainframe agent.

Step 8: Agent configuration

To configure the File Transfer agent

1. Navigate to `$AGENT_DIR/current-version/config`.
2. Edit the agent's configuration file `appication.properties` to add an environments configuration using the following command:

```
oedit $AGENT_DIR/current-version/config/application.properties
```

For example:

```
agent.environments[0].account-id=<AWS_ACCOUNT_ID>
agent.environments[0].agent-role-name=<AWS_IAM_ROLE_NAME>
agent.environments[0].access-key-id=<AWS_IAM_ROLE_ACCESS_KEY>
agent.environments[0].secret-access-id=<AWS_IAM_ROLE_SECRET_KEY>
agent.environments[0].bucket-name=<AWS_S3_BUCKET_NAME>
agent.environments[0].environment-name=<AWS_REGION>
agent.environments[0].region=<AWS_REGION>
zos.complex-name=<File_Transfer_Endpoint_Name>
```

Where:

- `AWS_ACCOUNT_ID` is the ID of the AWS account.
- `AWS_IAM_ROLE_NAME` is the name of the IAM role created in the the section called "Step 7: Create an IAM role for the agent to assume".
- `AWS_IAM_ROLE_ACCESS_KEY` is the access key of the IAM user created in the section called "Step 6: Create an IAM user with long-term access credentials".
- `AWS_IAM_ROLE_SECRET_KEY` is the access secret key for the IAM user created in the section called "Step 6: Create an IAM user with long-term access credentials".
- `AWS_S3_BUCKET_NAME` is the name of the transfer bucket created with the data transfer endpoint.
- `AWS_REGION` is the region in which you configure the File Transfer agent.

> **Note**
>
> You can have the File Transfer agent transfer to multiple regions and accounts in AWS by defining multiple environments.

- (Optional). `zos.complex-name` is the complex name you created when creating a File Transfer endpoint.

> **Note**
>
> This field is necessary only if you want to customize the complex name (which defaults to your sysplex name) that is the same as you defined when creating your File Transfer endpoint. For more information, see the section called “Create data transfer endpoints”.

> **Important**
>
> There can be several such sections, as long as the index in brackets — `[0]`— is incremented for each.

You must restart the agent for changes to take effect.

Requirements

1. When a parameter is added or removed, the agent has to be stopped and started. Start the File transfer agent using the following command in the CLI:

   ```
   /S M2AGENT
   ```

   To stop the M2 agent, use the following command in CLI:

   ```
   /P M2AGENT
   ```

2. You can have the File Transfer agent configured to transfer data to multiple regions and accounts in AWS by defining environment entries.

> **Note**
>
> Replace the values with the parameter values you created and configured previously.

```
#Region 1
agent.environments[0].account-id=AWS_ACCOUNT_ID
agent.environments[0].agent-role-name=AWS_IAM_ROLE_NAME
agent.environments[0].access-key-id=AWS_IAM_ROLE_ACCESS_KEY
agent.environments[0].secret-access-id=AWS_IAM_ROLE_SECRET_KEY
agent.environments[0].bucket-name=AWS_S3_BUCKET_NAME
agent.environments[0].environment-name=AWS_REGION
agent.environments[0].region=AWS_REGION

#Region 2
agent.environments[1].account-id=AWS_ACCOUNT_ID
agent.environments[1].agent-role-name=AWS_IAM_ROLE_NAME
agent.environments[1].access-key-id=AWS_IAM_ROLE_ACCESS_KEY
agent.environments[1].secret-access-id=AWS_IAM_ROLE_SECRET_KEY
agent.environments[1].bucket-name=AWS_S3_BUCKET_NAME
agent.environments[1].environment-name=AWS_REGION
agent.environments[1].region=AWS_REGION
```

Create data transfer endpoints for File Transfer

Data transfer endpoints enable connectivity with the source mainframe, and support high availability, scalability, and streamlined management of agents. Individual agents are installed on mainframe LPARs and can be grouped together into a data transfer endpoint. When a request is made to transfer a dataset, one agent in the data transfer endpoint will handle that specific transfer. To initiate data transfers, at least one agent on the data transfer endpoint must be online.

This procedure assumes that you have completed the steps in Set up for AWS Mainframe Modernization and Configure File Transfer agent on the source mainframe.

Create data transfer endpoints

To create data transfer endpoints for File Transfer, follow these steps in the AWS Mainframe Modernization console.

To create a data transfer endpoint

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 transfer files from your mainframe to an Amazon S3 bucket.
3. On the **Data transfer endpoints** page, under **File Transfer**, choose **Create data transfer endpoint**.
4. On the **Data transfer endpoint prerequisites** page, read all the instructions to make sure you have completed these steps on the source mainframe. Once confirmed, choose **Next**.
5. On the **Configure data transfer endpoint** page, add basic information for your data transfer endpoint.

    1. In the basic information section, enter your data transfer endpoint name.

        Note

        The data transfer endpoint name must match the Sysplex name, unless you specify a complex name in the agent configuration.

    2. An optional description.
    3. The KMS key used to encrypt the secret.

        Note

        You must add the following resource-based policy for KMS so the AWS Mainframe Modernization service can read and use these keys for encryption/decryption:

```
{
  "Sid" : "Enable AWS M2 Permissions",
  "Effect" : "Allow",
  "Principal" : {
      "Service" : "m2.amazonaws.com"
            },
   "Action" : [
       "kms:Encrypt",
       "kms:Decrypt"
              ],
   "Resource" : "*"
```

```
}
```

4. Specify the **S3 location for intermediate data**, which is the intermediate S3 location where transferred datasets from the mainframe are stored before they are converted and transferred to the target Amazon S3 bucket.

   .

   > **Note**
   >
   > It's recommended that you create a new Amazon S3 bucket for your transfer tasks. For additional information, see Creating a bucket. You can also browse your existing Amazon S3 buckets by choosing **Browse S3** option.

5. After entering required fields, choose **Next**.
6. On the **Review and create data transfer endpoint** page, check if you have completed prerequisites, and review basic information. Once confirmed, choose **Create data transfer endpoint**.

You will be redirected to the **Data transfer endpoints overview** page where you can see the list of all data transfer endpoints. You will also be able to see the data transfer endpoints that are available or have failed.

You can also search data transfer endpoints by name and access additional information for each available agent.

Create transfer tasks in File Transfer

Transfer tasks are used to specify the data sets to be transferred from the mainframe to Amazon S3 and allow you to choose the code page conversion options.

These instructions assume that you have completed the steps in Set up for AWS Mainframe Modernization and have created the section called “Create data transfer endpoints”.

Topics

- Create transfer tasks
- View transfer tasks

Create transfer tasks

To create transfer tasks in File Transfer, follow these steps in the AWS Mainframe Modernization console.

To create a transfer task

> **Important**
>
> You must have at least one data transfer endpoint to create new transfer tasks.

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 transfer files from your mainframe to an Amazon S3 bucket.
3. On the **Transfer tasks** page, you can choose any data transfer endpoint to create transfer tasks.
4. On the **Create transfer task** page, set up properties for your transfer task. If you have not created any transfer tasks previously, you can create your first one by choosing the **Create Transfer task** option.

   - On this page, enter the basic information of your transfer task, including the transfer task name, description, and secret key.

   > **Note**
   >
   > - Encrypt the secret using the KMS key defined with the data transfer endpoint. The secret should contain the mainframe credentials needed to access data sets on the mainframe using the `userId` and `password` keys. For more information, see the AWS Secrets Manager secret.
   > - You must configure the secret key with the following resource-based policy so that AWS Mainframe Modernization service can access it to perform data transfer task.

   ```
   {
     "Version" : "2012-10-17",
     "Statement" : [ {
       "Effect" : "Allow",
       "Principal" : {
   ```

```
        "Service" : "m2.amazonaws.com"
    },
    "Action" : [ "secretsmanager:GetSecretValue",
                 "secretsmanager:DescribeSecret" ],
    "Resource" : "*"
  } ]
}
```

> **Note**
>
> The current maximum supported dataset size for transfer is 90 GB.

- Next, select the target Amazon S3 bucket location where the target data sets from the mainframe will be transferred.
- The previously chosen data transfer endpoint will be selected. You can also select another endpoint from the available endpoints.

5. Choose **Next**.
6. On the **Add data sets** page, in the **Transfer task configuration** section, you can choose to either configure your transfer task in binary mode or to convert and transfer your data sets.

   - **Transfer in binary mode** option allows you to transfer data sets by skipping code page conversions and retaining their Record Descriptor Word (RDW) bytes.
   - **Transfer and convert data sets** option allows you to transfer data sets by setting the source and target code pages for your data sets. You can see the available code pages for File Transfer on the the section called “Supported source and target code pages” page.

7. Enter your query in the **Search mainframe for data sets** to search the mainframe for data sets to be included in your transfer task. Choose **View data sets**.

   The following wildcard symbols can be used as part of the data set search criteria for mainframe:

   - A single asterisk (*) as a qualifier (between periods or after the final period) matches a single qualifier in that position.
   - A single asterisk (*) within a qualifier matches zero or more characters in that position.
   - A double asterisk (**) as a qualifier (between periods or after the final period) matches zero or more qualifiers in that position.

- A double asterisk (**) within a qualifier is not a valid query.
- A single percent sign (%) matches any single alphanumeric or national character in that position. You can use up to eight percent signs in each qualifier.

> **Note**
>
> We suggest always ending your search criteria with a period followed by a double asterisk (.**) and then refine the search further, if needed.

For more information on wildcard rules, see the Filtering data set names in the *IBM documentation*.

8. These data sets will load under **Mainframe data sets** section, where you can search or choose one or more data sets you want to configure code page conversions for. These chosen data sets will be displayed in the **Added data sets** section. If no data sets are loaded, you need to revisit step 7.

> **Note**
>
> You can select data sets from multiple search queries and add them to your transfer task.

9. In the **Added data sets** section, you will see name, type, and volume name of your data sets.

> **Important**
>
> For **Transfer and convert data sets** option, you need to manually enter the source code page and target code page for each of your chosen data set. *Source code page* is the source data set format, and *target code page* is the target data set format used to convert the data sets and store them in the target Amazon S3 bucket.

10. After confirming the data sets in the **Added data sets** section, (and source and target code pages for Transfer and convert data sets option), choose **Next**.
11. On the **Review and create** page, you can review or edit information for your transfer task.
12. Then, choose **Create transfer task**.

> **Important**
>
> Choosing the **Create transfer task** button will start the data transfer, which is billable per the AWS Mainframe Modernization pricing page. This billing is based on the amount of data (GB) transferred as measured by the data set size.

View transfer tasks

To view transfer tasks in File Transfer, you must follow these steps in the AWS Mainframe Modernization console.

To view transfer tasks

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 transfer files from your mainframe to an Amazon S3 bucket.
3. On the **Transfer tasks** page, select the data transfer endpoint to view your transfer tasks.
4. For endpoints that have pre-existing transfer tasks, these will be displayed under the **Transfer tasks** section. You can choose to view details of any transfer task from this list.

Tutorial: Getting started with AWS Mainframe Modernization File Transfer

AWS Mainframe Modernization File Transfer lets you transfer and convert mainframe data sets for mainframe modernization, migration, and augmentation use cases.

Follow the steps in this tutorial to understand how AWS Mainframe Modernization File Transfer works.

Overview

File Transfer consists of the following:

1. An agent to be installed on the source mainframe.
2. Access to dataset discovery, transfer, and conversion capabilities directly from the AWS Mainframe Modernization management service console.

As a user, you can transfer datasets from the mainframe to your Amazon S3 bucket.

Topics

- Step 1: Transfer the agent binaries tar package from AWS to the mainframe logical partition
- Step 2: Configure the File Transfer agent on the source mainframe
- Step 3: Create a data transfer endpoint
- Step 4: Create a transfer task
- Step 5: View transfer task progress

Step 1: Transfer the agent binaries tar package from AWS to the mainframe logical partition

Download tar files from the M2-agent tar link.

Step 2: Configure the File Transfer agent on the source mainframe

In this step, you configure and start the AWS Mainframe Modernization File Transfer agent on the source mainframe. The agent is required to facilitate communications between the File Transfer service feature and the source mainframe. At least one agent is required per mainframe. More than one agent can be started for high availability and enhanced scalability.

Follow the instructions in the section called “Configure a File Transfer agent” guide to complete File Transfer agent installation on the mainframe.

Step 3: Create a data transfer endpoint

Follow steps on the section called “Create data transfer endpoints” page to create a new data transfer endpoint.

Step 4: Create a transfer task

Follow steps on the section called “Create transfer tasks” page to create and manage your transfer tasks.

Step 5: View transfer task progress

You can view your transfer task's progress in the AWS Mainframe Modernization console. For more details, refer the section called “View transfer tasks” section.

Supported source and target encodings in AWS Mainframe Modernization File Transfer

AWS Mainframe Modernization File Transfer supports various data set types and code page conversion options.

Mainframe data set types

AWS Mainframe Modernization File Transfer supports the following mainframe data set types:

- Non-VSAM: Sequential (PS), PDS, GDS, GDG
- VSAM types: KSDS

Supported code pages

AWS Mainframe Modernization File Transfer supports the following code pages for data set conversion (from/to):

"BIG5" , "BIG5_HKSCS" , "CESU_8" , "EUC_JP" , "EUC_KR" , "GB18030" , "GB2312" , "GBK" , "IBM00858" , "IBM01140" , "IBM01141" , "IBM01142", "IBM01143" , "IBM01144" , "IBM01145" , "IBM01146" , "IBM01147" , "IBM01148" , "IBM01149" , "IBM037" , "IBM1026" , "IBM1047" , "IBM273" , "IBM277" , "IBM278" , "IBM280" , "IBM284" , "IBM285" , "IBM290" , "IBM297" , "IBM420" , "IBM424" , "IBM437" , "IBM500" , "IBM775" , "IBM850" , "IBM852" , "IBM855" , "IBM857" , "IBM860" , "IBM861" , "IBM862" , "IBM863" , "IBM864" , "IBM865" , "IBM866" , "IBM868" , "IBM869" , "IBM870" , "IBM871" , "IBM918" , "IBM_THAI" , "ISO_2022_CN" , "ISO_2022_JP" , "ISO_2022_JP_2" , "ISO_2022_KR" , "ISO_8859_1" , "ISO_8859_13" , "ISO_8859_15" , "ISO_8859_16" , "ISO_8859_2" , "ISO_8859_3" , "ISO_8859_4" , "ISO_8859_5" , "ISO_8859_6" , "ISO_8859_7" , "ISO_8859_8" , "ISO_8859_9" ,"JIS_X0201" , "JIS_X0212_1990" , "KOI8_R" , "KOI8_U" , "SHIFT_JIS" , "TIS_620" , "US_ASCII" , "UTF_16" , "UTF_16BE" , "UTF_16LE" , "UTF_32" , "UTF_32BE" , "UTF_32LE" , "UTF_8" , "WINDOWS_1250" , "WINDOWS_1251" , "WINDOWS_1252" , "WINDOWS_1253" ,"WINDOWS_1254" , "WINDOWS_1255" , "WINDOWS_1256" , "WINDOWS_1257" , "WINDOWS_1258" , "WINDOWS_31J" , "X_BIG5_HKSCS_2001" , "X_BIG5_SOLARIS" , "X_EUCJP_OPEN" , "X_EUC_JP_LINUX" , "X_EUC_TW" , "X_IBM1006" , "X_IBM1025" , "X_IBM1046" , "X_IBM1097" , "X_IBM1098" , "X_IBM1112" , "X_IBM1122" , "X_IBM1123" , "X_IBM1124" , "X_IBM1129" , "X_IBM1166" , "X_IBM1364" , "X_IBM1381" , "X_IBM1383" , "X_IBM29626C" , "X_IBM300" , "X_IBM33722" , "X_IBM737" , "X_IBM833" , "X_IBM834" , "X_IBM856" , "X_IBM874" , "X_IBM875" , "X_IBM921" , "X_IBM922" , "X_IBM930" ,

"X_IBM933" , "X_IBM935" , "X_IBM937" , "X_IBM939" , "X_IBM942" , "X_IBM942C" , "X_IBM943" , "X_IBM943C" , "X_IBM948" , "X_IBM949" , "X_IBM949C" , "X_IBM950" , "X_IBM964" , "X_IBM970" , "X_ISCII91" , "X_ISO_2022_CN_CNS" , "X_ISO_2022_CN_GB" , "X_ISO_8859_11" , "X_JIS0208" , "X_JISAUTODETECT" , "X_JOHAB", "X_MACARABIC" , "X_MACCENTRALEUROPE" , "X_MACCROATIAN" , "X_MACCYRILLIC" , "X_MACDINGBAT", "X_MACGREEK" , "X_MACHEBREW" , "X_MACICELAND" , "X_MACROMAN" , "X_MACROMANIA" , "X_MACSYMBOL" , "X_MACTHAI" , "X_MACTURKISH" , "X_MACUKRAINE" , "X_MS932_0213" , "X_MS950_HKSCS" , "X_MS950_HKSCS_XP" , "X_MSWIN_936" , "X_PCK" , "X_SJIS_0213" , "X_UTF_16LE_BOM" , "X_UTF_32BE_BOM" , "X_UTF_32LE_BOM" , "X_WINDOWS_50220" , "X_WINDOWS_50221" , "X_WINDOWS_874" , "X_WINDOWS_949" , "X_WINDOWS_950" , "X_WINDOWS_ISO2022j"

AWS Transform for modernizing mainframe applications

AWS Transform for mainframe empowers you to modernize legacy COBOL mainframe applications to Java applications faster by automating the complex, time-consuming tasks of analyzing codebases, planning the transformation, generate technical documentation, extracting business logic, decomposition of code, wave planning, and executing the refactoring and reforging. It reduces the cost and complexity of mainframe modernization by leveraging generative AI and automation, while preserving your mission-critical business logic. AWS Transform's natural language interface and objective-driven approach keeps you in control of the transformation—enabling you to focus on strategic priorities, while the automation handles the heavy lifting of the modernization journey.

For more information on capabilities and key features, high-level walkthrough, and human involvement in inputs and processing, see Migration and modernization of mainframe applications in the *AWS Transform User Guide*.

Key benefits

AWS Transform's capabilities for modernizing mainframe applications has multiple benefits. Some of which include:

- **Accelerate mainframe modernization journey with generative AI**: AWS Transform helps you to transform your COBOL code into modern Java code within a few months, instead of the traditional timeline in years.
- **Bridge knowledge gaps**: AWS Transform generates comprehensive documentation for your mainframe applications, bridging the knowledge gap, and enabling better-informed decisions.
- **Preserve the critical business logic**: AWS Transform preserves the critical business logic of your legacy systems while refactoring them to modern, cloud-optimized Java applications.
- **Business and technical logical domain decomposition**: AWS Transform automatically decomposes the mainframe codebase into distinct business domains to reduce manual effort and time required for codebase analysis and decomposition.
- **Human in the loop (HITL) capabilities**: AWS Transform provides an autonomous, objective-driven approach that keeps you in control of the mainframe modernization journey.

Transformation of mainframe application console walkthrough

In the AWS Transform web experience, you can perform transformation of your mainframe applications from COBOL to Java. To understand how to use this function, follow all steps in the Transformation of mainframe applications page in the *AWS Transform User Guide*.

AWS Mainframe Modernization data replication with Precisely

AWS Mainframe Modernization offers a variety of Amazon Machine Images (AMIs). These AMIs facilitate rapid provisioning of Amazon EC2 instances, creating a tailored environment for data replication from Mainframe systems to AWS using Precisely. This guide provides the steps required to access and use these AMIs.

Prerequisites

- Ensure that you have administrator access to an AWS account where you can create Amazon EC2 instances.
- Verify that the AWS Mainframe Modernization service is available in the Region where you plan to create the Amazon EC2 instances. See List of AWS Services Available by Region.
- Identify the Amazon Virtual Private Cloud (Amazon VPC) where the Amazon EC2 instances will be created.
- When creating Amazon EC2 instances in an Amazon VPC, ensure that the associated route table has an internet gateway or a NAT gateway.

> **Note**
>
> Successful data replication requires the AWS EC2 instance has communication access to the AWS Marketplace. If there's a connectivity issue with the AWS Marketplace, the replication process will fail.

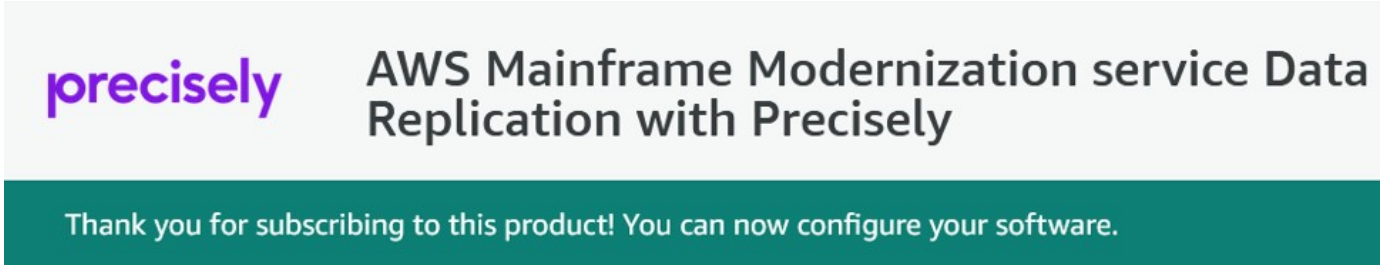
Subscribe to the Amazon Machine Image

When you subscribe to an AWS Marketplace product, you can launch an instance from the product's AMI.

1. Sign in to the AWS Management Console and open the AWS Marketplace console at https://console.aws.amazon.com/marketplace.
2. Choose **Manage subscriptions**.

3. Navigate to any of the following links based on your use case:

   - **Data Replication for IBM z/OS**: https://aws.amazon.com/marketplace/pp/prodview-doe2lroefogia
   - **Data Replication for IBM i**: https://aws.amazon.com/marketplace/pp/prodview-iqrkflccxf7ko
4. Choose **Continue to Subscribe**.
5. If the terms and conditions are acceptable, choose **Accept Terms**. The subscription might take a few minutes to process.
6. Wait for the thank you message to appear, as shown below. This message confirms that you have successfully subscribed to the product.



7. In the left navigate pane, choose **Manage subscriptions**. This view shows you all the subscriptions that you've subscribed to.

Launch AWS Mainframe Modernization data replication with Precisely

1. Open the AWS Marketplace console at https://console.aws.amazon.com/marketplace.
2. In the left navigation pane, choose **Manage subscriptions**.
3. Find the AMI that you want to launch, and choose **Launch new instance**.
4. Under **Region**, select the allow-listed Region.
5. Choose **Continue to launch through EC2**. This action takes you to the Amazon EC2 console.
6. Enter a name for the server.
7. Select an instance type that matches your project performance and cost requirements. The suggested starting point for instance size is `c5.2xLarge`.
8. Choose an existing key pair or create and save a new one. For information about key pairs, see Amazon EC2 key pairs and Linux instances in the Amazon EC2 User Guide.

9. Edit the network settings and choose the allow-listed VPC and appropriate subnet.
10. Choose an existing security group or create a new one. In addition to allowing SSH access (by default on port 22), for data replication with a Precisely server EC2 instance, it is typical to allow TCP traffic to its default port 2626.
11. Configure the storage for the Amazon EC2 instance.
12. Review the summary and choose **Launch instance**. For the launch to success, the instance type must be valid. If the launch fails, choose **Edit instance configuration** and choose a different instance type.
13. After you see the success message, choose **Connect to instance**.
14. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
15. In the left navigation pane, under the **Instances** menu, choose **Instances**.
16. In the main pane, check the status of your instance.

Create an IAM policy

To successfully operate AWS Mainframe Modernization EC2 instances deployed via our AWS Marketplace listing, you must configure an IAM role and policy. This specifically-tailored IAM setup is not optional; it authorizes your Amazon EC2 instances to interact with the AWS Marketplace service. The IAM role and policy allow AWS Mainframe Modernization to accurately record usage data, which is essential for precise billing. Failing to implement this configuration may lead to unsuccessful data replication attempts and operational disruptions.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.
3. If this is your first-time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
4. At the top of the page, choose **Create policy**.
5. In the **Policy** editor section, choose the **JSON** option.
6. Enter the following JSON policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": ["aws-marketplace:MeterUsage"],
```

```
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Create an IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then choose **Create role**.
3. In the **Trusted entity type** section, choose **AWS service**.
4. In the **Use case** section, under **Service or use case**, choose **Amazon EC2**.
5. Choose **Next**.
6. In the list of policies, select **Customer managed** from the **Filter by Type** drop-down and enter the name of the policy that you created. Select the check box next to the name of the policy.
7. Choose **Next**.
8. Enter a name and, optionally, a description for the role.
9. Review the trust policy and permissions, and then choose **Create role**.

Attach the IAM role to the Amazon EC2 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select your Amazon EC2 instance.
4. From the **Actions** menu, choose **Security**, and then choose **Modify IAM role**.
5. Select the role to attach to your instance, and then choose **Update IAM role**.

For more information regarding getting started with AWS Data replication for IBM i, see Data replication overview.

AWS Mainframe Modernization Code Conversion with mLogica

AWS Mainframe Modernization Code Conversion with mLogica (Code conversion), is an AWS Mainframe Modernization feature that automatically converts z/OS mainframe Assembler code to COBOL. You can use Code conversion to pull an assembler image using the AWS CodeBuild service for your intended code conversion with your AWS account.

Topics

- What is Assembler Conversion with mLogica?
- Understand Code conversion billing for Assembler conversion
- Code conversion concepts
- Understand components and processes for Code conversion
- Tutorial: Convert code from Assembler to COBOL in AWS Mainframe Modernization

What is Assembler Conversion with mLogica?

AWS Mainframe Modernization Code Conversion with mLogica (Code conversion) automatically converts z/OS mainframe Assembler code to COBOL. The service runs within your AWS account and doesn't transmit or store Assembler or COBOL source code outside the AWS account. Code conversion allows your authorized account to pull an assembler image using the AWS CodeBuild service for your intended code conversion.

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 and Amazon S3 to provide this feature. AWS CodeBuild is a fully managed build service that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. Amazon S3 is an object storage service that offers industry- leading scalability, data availability, security, and performance.

Topics

- Code conversion compilers
- Code conversion architecture
- Automation approach

- Security
- Additional resources

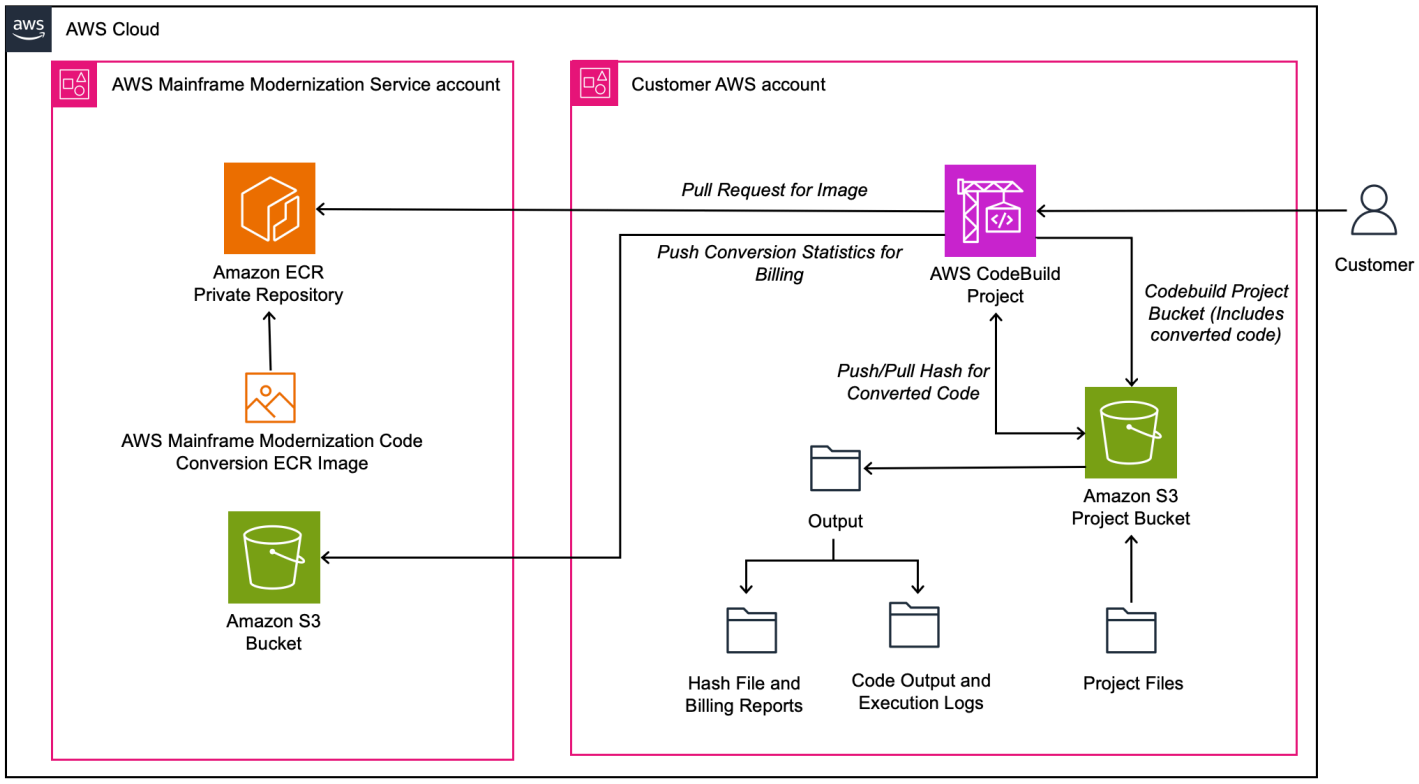
Code conversion compilers

Code conversion can be configured to emit COBOL suitable for compilation and running in several target environments with different compilers. Some of these include:

- M2 Re-platforming with Rocket Software (formerly Micro Focus) and other Rocket Enterprise Server environments
- M2 Re-platforming with NTT DATA Enterprise COBOL (UniKix)
- mLogica LIBER*COBOL
- z/OS Mainframe using IBM Enterprise COBOL
- Veryant isCOBOL

Code conversion architecture

The following is an architectural diagram for the Code conversion process:



Automation approach

To use Code conversion with CodeBuild, the Assembler code needs to be uploaded to an Amazon S3 bucket, to later configure conversion parameters and invoke a CodeBuild project to perform each step in the conversion process. The target COBOL code is automatically stored in a specified path in the Amazon S3 bucket.

Security

AWS Mainframe Modernization Code conversion enables conversion while keeping all source and target code in your AWS account. Source Assembler code, target COBOL code, and configuration files are stored in your Amazon S3 bucket. The automated conversion tool runs as a container in the CodeBuild environment in your AWS account. The code stays in your account at all times.

To enable the Conversion tool to access your Amazon S3 bucket, you grant permissions to the bucket to an AWS service role. When you configure CodeBuild, you will set this service role so that CodeBuild can access the container image and access your Amazon S3 bucket.

Additional resources

Along with the the section called “Tutorial: Convert code from Assembler to COBOL”, here are some additional resources where you can learn about creating the AWS CloudFormation templates and other information about converting Assembler to COBOL.

- Workshop link for Automated Code conversion from Assembler to COBOL: https://catalog.workshops.aws/awsm2ccm-assembler-cobol/en-US.
- Blog post: https://aws.amazon.com/blogs/migration-and-modernization/unlocking-new-potential-transform-your-assembler-programs-to-cobol-with-aws-mainframe-modernization/.

Understand Code conversion billing for Assembler conversion

You will refer this page to understand the Code conversion billing scope and process before doing the actual conversion. The billing calculation section mentions the process though which conversion from Assembler to COBOL is charged per each line of code.

Code conversion billing and scope

Assembler code conversion generates charges (billing reports) on your AWS account only after completing the *conversion* step. The charge is based on the number of lines of code converted. If

you perform multiple conversion steps, for instance after adding new Assembler code, changing the conversion configuration, or applying a new version of the container, only changed lines and/or newly added lines are used to calculate the charge. We won't charge you twice for conversion of the same line of code in the same program.

> **Note**
>
> The modules with changed lines of code and all lines of code in new or renamed programs will be charged.

To avoid multiple charges, Code conversion stores an encoded binary file for each Assembler or Macro module in the project bucket in `<Project_bucket>/awsm2ccm-do-not-delete/<AWS_account_number>/Hash`. These encoded files do not contain any customer code.

> **Important**
>
> Don't manually edit or delete these files. Changes may result in multiple billings for converting the same components.

The AWS Mainframe Modernization Code conversion analysis report (“Analysis Report”) provides customers with details about the anticipated conversion scope, outcome, and billing to ensure accurate expectations of the actual conversion. Conversion may result in some lines of code not getting converted, some lines of code partially getting converted, and some lines of code converting completely. The Analysis Report shows the number of lines of code for each category. **You must run and read the Analysis Report prior to processing any conversion of programs, macros, and copybooks.** Once a customer reviews the Analysis Report and agrees with the reported scope, expected outcome, and expected billing, the customer can move forward with executing the conversion.

> **Note**
>
> By executing the AWS Mainframe Modernization Code Conversion **`Convert`** command, you acknowledge that you have run and read the Analysis Report, and agree with the expected outcome and the billable number of lines of code.

Scope of Conversion

AWS Mainframe Modernization Code conversion processes all lines of code of all assembler, macro and copybook components available in the *scrlib* and *macrolib* directories in the configured S3 source location. Assembler programs, and any macros, and copybooks referenced in an assembler program are *in scope*. Macro and copybook components that are not referenced by an assembler program are considered as *out of scope* and not converted. During processing, the converter executes advanced algorithms that consider each in-scope component holistically. All lines of code of these components participate in the processing regardless of whether they are totally converted, partially converted, or not converted. AWS Mainframe Modernization Code conversion ignores blank lines and doesn't count them as lines of code. Comment lines and lines containing any other text (e.g., JCL statements for assembler embedded in JCL) are counted as lines of code for billing.

Billing calculation

AWS Mainframe Modernization Code conversion charges for the in-scope components in their entirety. This means that it charges for every line of code within each in-scope component, including lines that could not be converted, were partially converted, and were totally converted. AWS Mainframe Modernization Code conversion adds up all lines of code of components supplied for processing (including assembler programs, referenced copybooks, and referenced macros), and uses the total number of lines of code for billing.

> **Note**
>
> Copybooks and macros not referenced by an Assembler program are not considered in-scope.

For example, assume a program has 1,000 lines of code:

- 700 lines are fully converted
- 200 lines are partially converted
- 100 lines are not converted

1,000 lines of code would be processed and will be billable.

Improving the conversion

If you as a customer seek a higher conversion rate for the lines of code or have other specific requirements, you can reach out to the AWS representatives for additional engagement options such as a calibration effort, or professional services assistance.

Code conversion concepts

To learn how code conversion happens, understanding some key concepts such as Macro handling, Code pages, and CodeBuild is important.

Topics

- Macro Handling
- Code pages (EBCDIC vs ASCII)
- CodeBuild

Macro Handling

Mainframe Assembler code frequently uses Macros to encapsulate functionality for reuse. Macro behavior is typically determined at application runtime based on parameters passed from an Assembler program. Code conversion provides several mechanisms for expanding Assembler Macros prior to conversion to COBOL.

Code pages (EBCDIC vs ASCII)

Mainframe Assembler often contain character literals expressed as hexadecimal values corresponding to EBCDIC characters. Code conversion provides a configurable capability to automatically manage character literals in ASCII when emitting COBOL for ASCII environments.

CodeBuild

Code conversion is available through the AWS CodeBuild service. AWS CodeBuild is a build automation tool originally designed as a part of a CI/CD pipeline. In AWS Mainframe Modernization, AWS CodeBuild is used to automate the MCCAC Conversion tool and other tools such as the Rocket Software (formerly Micro Focus) COBOL compiler.

Understand components and processes for Code conversion

AWS Mainframe Modernization Code conversion process includes various components such as AWS Mainframe Modernization container, S3 project bucket, and Log file locations.

Topics

- AWS Mainframe Modernization container
- S3 project bucket
- Log file locations
- Process overview

AWS Mainframe Modernization container

AWS Mainframe Modernization Code conversion container runs in the AWS CodeBuild project, and provides commands to set up the project directories and configuration files, assess Assembler code, expand Assembler macros, and convert Assembler code to COBOL.

You will have access to the following AWS ECR Repository:`381492161314.dkr.ecr.us-east-1.amazonaws.com/aws-mlogica-codebuild-prod`.

To use the images, you can follow either of the two options:

- Use the latest tag when consuming the image via AWS CodeBuild. When using the image, you will use this path: `381492161314.dkr.ecr.us-east-1.amazonaws.com/aws-mlogica-codebuild-prod`. This means that AWS CodeBuild will pick up whichever was the last pushed image into the repository.
- Listing the version and selecting from that. To do this use the following command via CLI to list the different versions in the repository:

```
aws ecr describe-images \
  --registry-id 381492161314 \
  --repository-name aws-mlogica-codebuild-prod \
  --query 'imageDetails[*].{ImagePushedAt: imagePushedAt, ImageTags: imageTags}' \
  --output json | jq '[.[] | {ImageURI: (.ImageTags[] |
 "381492161314.dkr.ecr.us-east-1.amazonaws.com/aws-mlogica-codebuild-prod:" + .),
 ImagePushedAt: .ImagePushedAt}] | sort_by(.ImagePushedAt) | reverse'
```

This will list all the images with the associated tag on each image, and the time when a particular image was released to the repository. Based on the above code, you will get a list of images where the tag on the image represents the version of the code conversion utility. You can select the appropriate image based on your requirements.

S3 project bucket

The input and output code, code updated with expanded Macros, and the reports generated by AWS Mainframe Modernization Code conversion are stored in the project bucket you create in your AWS Account Management. You provide AWS Mainframe Modernization Code conversion with access to the bucket by granting permissions to an AWS service role.

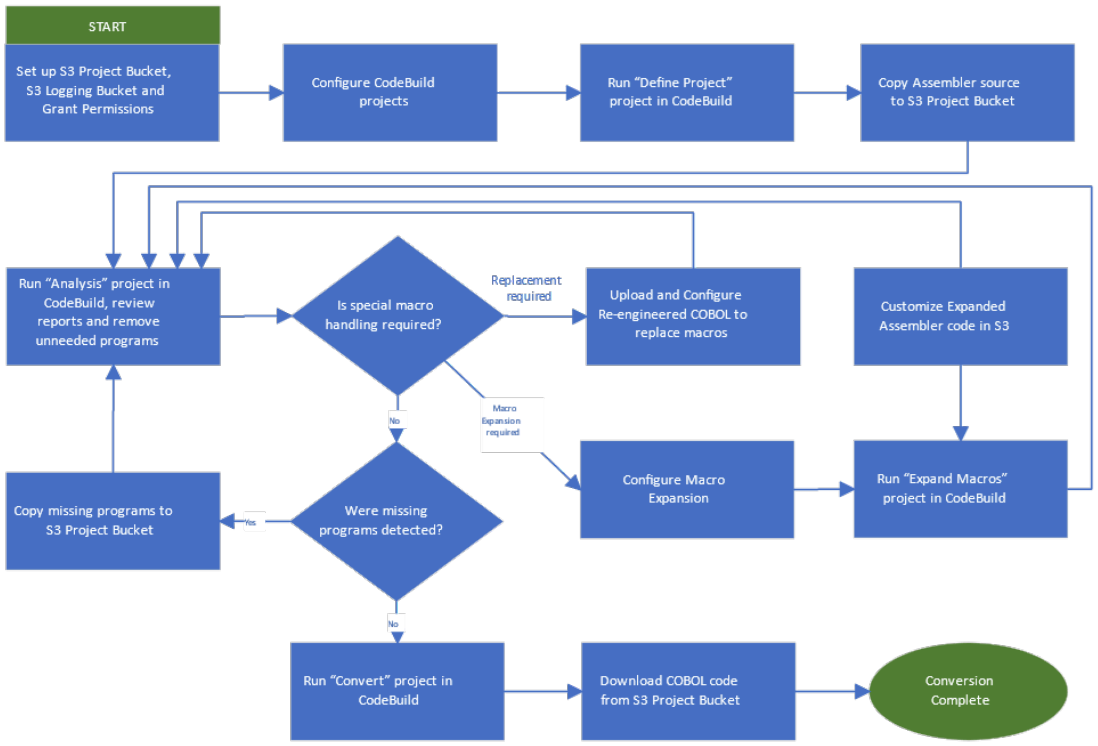
Log file locations

Log files are written in two locations during each CodeBuild project execution:

- Log files with high-level results of each CodeBuild step are written to log files in the Logging bucket configured in the CodeBuild. These files appear as gzip archives with a GUID- type file name generated by the CodeBuild framework (e.g., `0c03e183-ab40-4fe0-ba77-bc1d87e73b14.gz`). Each archive contains the log generated by the execution of a CodeBuild project. If a CodeBuild project execution fails, this log file will contain important troubleshooting information.
- Log files with detailed execution results at a component level are written to the log files in the main Project bucket path with the filename pattern `<Project_Bucket_name>_.log` (e.g. `project- bucket_202406131200.log`). These logs provide:
  - A configuration summary noting input and output locations.
  - A log of each Assembler or Macro component processed with the target filename.
  - A list of reports generated with file locations.
  - For conversion executions, a list of the run-time copybooks supplied.

Process overview

The following diagram illustrates the process of converting Assembler to COBOL:



Tutorial: Convert code from Assembler to COBOL in AWS Mainframe Modernization

You can use this document as a step-by-step guide to understand how to convert the mainframe modernization Assembler code to COBOL. In addition to this, you can also refer the Automated code conversion from Assembler to COBOL workshop to learn more about the conversion process.

Topics

- Prerequisites
- Step 1: Share the build assets with AWS account
- Step 2: Create Amazon S3 buckets
- Step 3: Create IAM policy
- Step 4: Create an IAM role
- Step 5: Attach the IAM policy to the IAM role
- Step 6: Create the CodeBuild project
  - Step 6.1: Create the Define project
  - Step 6.2: Create the Code Analysis project

  - Step 6.3: Create the Code Conversion project
- Step 7: Define the project and upload the source code
- Step 8: Run the analysis and understand the reports
- Step 9: Run the Code conversion
- Step 10: Verify the Code conversion
- Step 11: Download converted code
- Clean up resources

Prerequisites

Read the Understand Code conversion billing for Assembler conversion section to understand how Assembler code conversion generates charges (billing reports) on your AWS Account Management, and the way billing works.

Step 1: Share the build assets with AWS account

In this step, ensure that you share the build assets with your AWS account, especially in the Region where assets are being used.

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the left navigation, choose **Tools**.
3. In **AWS Mainframe Modernization Code Conversion with mLogica**, choose **Share assets with my AWS account**.

> **Important**
>
> You need to do this step once in every AWS Region where you intend to do builds.

Step 2: Create Amazon S3 buckets

In this step, you create Amazon S3 buckets. The first bucket is the project bucket for AWS CodeBuild to hold the source code and then push the output bucket to hold the AWS CodeBuild output (converted code). For more information, see Creating, configuring, and working with Amazon S3 buckets in the *Amazon S3 User Guide*.

1. To create the project 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-bucket`*, where:

   - `regionId` is the AWS Region of the bucket.
   - `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**.

Whatever names you choose for these buckets, be sure to use them throughout this tutorial.

Step 3: Create IAM policy

In this step, you create an IAM policy. The provided IAM policy grants specific permissions AWS CodeBuild for interacting with Amazon S3, Amazon Elastic Container Registry, Amazon CloudWatch logs that CodeBuild generates, and Amazon Elastic Compute Cloud resources for Code conversion. This policy is not customized for customers. The policy grants permissions for AWS Mainframe Modernization to interact, and fetch the Code conversion statistics to bill the customer appropriately.

To learn about creating an IAM policy, see Creating IAM policies in the *IAM user guide*.

To create a policy

1. Log in to the IAM console, and choose **Policies** in the left navigation pane.
2. Choose **Create policy**.
3. Copy and paste the following JSON policy into the policy editor.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetBucketLocation",
                "s3:ListBucket",
                "s3:PutObjectAcl",
                "s3:GetBucketAcl"
            ],
            "Resource": [
                "arn:aws:s3:::codebuild-regionId-accountId-bucket",
                "arn:aws:s3:::codebuild-regionId-accountId-bucket/*",
                "arn:aws:s3:::aws-m2-repo-*" ],
                    "Effect": "Allow"
         },
         {
             "Action": [
               "ecr:GetAuthorizationToken",
               "ecr:BatchCheckLayerAvailability",
               "ecr:BatchGetImage",
               "ecr:GetDownloadUrlForLayer",
               "logs:*",
               "ec2:DescribeSecurityGroups",
               "ec2:DescribeSubnets",
               "ec2:DescribeNetworkInterfaces",
               "ec2:DeleteNetworkInterface",
               "ec2:CreateNetworkInterface",
               "ec2:DescribeDhcpOptions",
               "ec2:DescribeVpcs",
               "ec2:CreateNetworkInterfacePermission"
            ],
            "Resource": "*",
            "Effect": "Allow"
         }
    ]
}
```

4. You can optionally add tags to the policy. Tags are key-value pairs that can help you organize, track, or control access for the policy.
5. Choose **Next:Review**.
6. Provide a name for the policy, for example, *CodeBuildAWSM2CCMPolicy*.

7. You can optionally enter a description for the policy, and review the policy summary to ensure it's correct.
8. Choose **Create policy**.

Step 4: 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, `IAMRoleTaskExecutionRoleForCodeBuild`.
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 5: Attach the IAM policy to the IAM role

In this step, you attach the IAM policy you previously created to the `IAMRoleTaskExecutionRoleForCodeBuild` 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, `IAMRoleTaskExecutionRoleForCodeBuild`.
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, *`CodeBuildAWSM2CCMPolicy`*.
5. Choose **Attach policies.**

Step 6: Create the CodeBuild project

In this step, you create three different CodeBuild projects based on the `buildspec.yml` file mentioned above.

Step 6.1: Create the Define project

To create the Define 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, `1-awsm2ccm-define-project`.
3. In the **Source** section, for **Source provider**, leave the default selection.
4. In the **Environment** section, choose **Custom image**.
5. In the **Environment type** field, choose **Linux**.
6. Under **Image registry**, choose **Other registry**.
7. In the **External registry URL** field, follow the the section called “AWS Mainframe Modernization container” section.
8. Under **Service role**, choose **Existing service role**, and in the **Role ARN** field, choose the service role you created previously (e.g., `IAMRoleTaskExecutionRoleForCodeBuild`).
9. Expand the **Additional configuration** section, do the following:

    a. VPC: Configure if needed based on your setup.

b. Timeout: Set to **60 minutes**.

c. Queued timeout: Set to **480 minutes**.

d. Encryption: Choose the appropriate encryption settings (default is fine).

e. In the **Environment variables** section, add the following one by one:

- Name : **PROJECT_BUCKET**. Value : **`codebuild-regionId-accountId- bucket`**. Type : **Plaintext**
- Name : **PROJECT_DIR**. Value : **`prj_codebuild_01`**. Type : **Plaintext**
- Name : **AWSM2CCM_ACTION**. Value : **`define_project`**. Type : **Plaintext**
- Name : **AWSM2CCM_LOGGING_BUCKET**. Value : **`s3:// codebuild-regionId-accountId-bucket`**. Type : **Plaintext**

10. In the **Buildspec** section, choose **Insert build commands**, and then **Switch to editor**.

11. Replace the current values with this:

```
version: 0.2
phases:
    build:
        commands:
            - . /app/awsm2ccm_prod/bin/setup_env.sh
            - run_awsm2ccm.sh $PROJECT_DIR
artifacts:
    files:
        - '**/*'
    discard-paths: no
    base-directory: $PROJECT_DIR
```

where, PROJECT_DIR are environment variables available within CodeBuild. For more information, see Environment variables in build environments.

12. In the **Artifacts** section, do this:

- under **Type**, choose **Amazon S3**, and then choose your output bucket, for example, `codebuild-regionId-accountId-bucket`.
- for **Path**, leave this field empty.
- for **Name**, enter **`prj_codebuild_01`**.
- for **Artifact packaging**, select **None**.
- for **Override artifact name**, uncheck this option.

- for **Encryption**, leave it to default settings.

13. For the **Logs** section, do the following:

    - CloudWatch logs: **Disabled**
    - S3 Logs: **Enabled**

    - Bucket: **`codebuild-regionId-account-bucket`**
    - Log path: **`CODEBUILD-LOGS`**

14. Choose **Create build project**.

Step 6.2: Create the Code Analysis project

To create the Code Analysis 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, `2-awsm2ccm-analysis`.
3. In the **Source** section, for **Source provider**, choose **Amazon S3**, and then choose the input bucket you created previously (e.g., `codebuild-regionId-accountId-bucket`).
4. In the **S3 object key** or **S3 folder** field, enter **`prj_codebuild_01`**.
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, follow the the section called "AWS Mainframe Modernization container" section.
9. Under **Service role**, choose **Existing service role**, and in the **Role ARN** field, choose the service role you created previously (e.g.,`IAMRoleTaskExecutionRoleForCodeBuild`).
10. Expand the **Additional configuration** section, do the following:

    a. VPC: Configure if needed based on your setup.

    b. Timeout: Set to **60 minutes**.

    c. Queued timeout: Set to **480 minutes**.

    d. Encryption: Choose the appropriate encryption settings (default is fine).

e. In the **Environment variables** section, add the following one by one:

- Name: **PROJECT_BUCKET**. Value : **`codebuild-regionId-accountId-bucket`**. Type : **Plaintext**
- Name : **PROJECT_DIR**. Value : **`prj_codebuild_01`**. Type : **Plaintext**
- Name : **AWSM2CCM_ACTION**. Value : **`analysis`**. Type : **Plaintext**
- Name : **AWSM2CCM_LOGGING_BUCKET**. Value : **`s3:// codebuild-regionId-accountId-bucket`**. Type : **Plaintext**

11. In the **Buildspec** section, choose **Insert build commands**, and then **Switch to editor**.
12. Replace the current values with this:

```
version: 0.2
phases:
    build:
        commands:
            - ln -s $CODEBUILD_SRC_DIR $PROJECT_DIR
            - . /app/awsm2ccm_prod/bin/setup_env.sh
            - run_awsm2ccm.sh $PROJECT_DIR
artifacts:
    files:
        - '*.log'
        - '_Converted/*/*'
        - '_Reports/*'
    secondary-artifacts:
        reports:
            files:
                - '_Reports/AWSM2CCM*'
    discard-paths: no
    base-directory: $PROJECT_DIR
```

where, PROJECT_DIR are environment variables available within CodeBuild. For more information, see Environment variables in build environments.

13. In the **Artifacts** section, do this:

- under **Type**, choose **Amazon S3**, and then choose your output bucket (e.g., `codebuild-regionId-accountId-bucket`).
- for **Path**, enter **ARTIFACTS**.
- for **Name**, enter **`prj_codebuild_01`**.

- for **Artifact packaging**, select **None**.
- for **Override artifact name**, uncheck this option.
- for **Encryption**, leave it to default settings.

14. For the **Logs** section, do the following:

    - CloudWatch logs: **Disabled**
    - S3 Logs: **Enabled**
    - Bucket: **`codebuild-regionId-account-bucket`**
    - Log path: **`CODEBUILD-LOGS`**

15. Choose **Create build project**.

Step 6.3: Create the Code Conversion project

To create the Code Conversion 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 (e.g.,`3-awsm2ccm-convert`).
3. In the **Source** section, for **Source provider**, choose **Amazon S3**, and then choose the input bucket you created previously (e.g.,`codebuild-regionId-accountId-bucket`).
4. In the **S3 object key** or **S3 folder** field, enter **`prj_codebuild_01`**.
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, follow the the section called "AWS Mainframe Modernization container" section.
9. Under **Service role**, choose **Existing service role**, and in the **Role ARN** field, choose the service role you created previously; for example, `IAMRoleTaskExecutionRoleForCodeBuild`.
10. Expand the **Additional configuration** section, do the following:

    a. VPC: Configure if needed based on your setup.

    b. Timeout: Set to **60 minutes**.

c. Queued timeout: Set to **480 minutes**.

d. Encryption: Choose the appropriate encryption settings (default is fine).

e. In the **Environment variables** section, add the following one by one:

- Name: **PROJECT_BUCKET**. Value : **`codebuild-regionId-accountId-bucket`**. Type : **Plaintext**
- Name : **PROJECT_DIR**. Value : **`prj_codebuild_01`**. Type : **Plaintext**
- Name : **AWSM2CCM_ACTION**. Value : **`conversion`**. Type : **Plaintext**
- Name : **AWSM2CCM_LOGGING_BUCKET**. Value : **`s3:// codebuild-regionId-accountId-bucket`**. Type : **Plaintext**

11. In the **Buildspec** section, choose **Insert build commands**, and then **Switch to editor**.
12. Replace the current values with this:

```
version: 0.2
phases:
    build:
        commands:
            - export AWSM2CCM_PUSH_RUNTIME_COPYBOOKS=y
            - ln -s $CODEBUILD_SRC_DIR $PROJECT_DIR
            - . /app/awsm2ccm_prod/bin/setup_env.sh
            - run_awsm2ccm.sh $PROJECT_DIR
artifacts:
    files:
        - '*.log'
        - '_Converted/*/*'
        - '_Reports/*'
    discard-paths: no
    base-directory: $PROJECT_DIR
```

where, PROJECT_DIR are environment variables available within CodeBuild. For more information, see Environment variables in build environments.

13. In the **Artifacts** section, do this:

- under **Type**, choose **Amazon S3**, and then choose your output bucket (e.g., `codebuild-regionId-accountId-bucket`).
- for **Path**, enter **ARTIFACTS**.
- for **Name**, enter **`prj_codebuild_01`**.

- for **Artifact packaging**, select **None**.
- for **Override artifact name**, uncheck this option.
- for **Encryption**, leave it to default settings.

14. For the **Logs** section, do the following:

    - CloudWatch logs: **Disabled**
    - S3 Logs: **Enabled**

    - Bucket: **`codebuild-regionId-account-bucket`**
    - Log path: **`CODEBUILD-LOGS`**

15. Choose **Create build project**.

Step 7: Define the project and upload the source code

The Define Project sets up the project folder and configuration files, initialized with default configurations. In this step, you start the build. To do this:

1. Log in to the AWS CodeBuild console.
2. In the left navigation pane choose **Build projects**.
3. Select the previously created project (`1-awsm2ccm-define-project`) to build
4. Choose **Start build**, and then **Start now** to define the project. Once the build starts, the status will change to *in progress*.
5. Choose **Phase details** to see the progress of each step which is orchestrated by the AWS CodeBuild project.
6. Wait until the status has changed to **succeeded** for all the steps.
7. Go to the Amazon S3 console.
8. Locate and click on Amazon S3 bucket named `codebuild-regionId-accountId-bucket`

    - **`CODEBUILD-LOGS/`** folder contains the AWS CodeBuild logs for the running AWS CodeBuild projects.
    - **`prj_codebuild_01/`** folder that contains the project structure. It's used during analysis, expand_macros, and convert steps. You can select `prj_codebuild_01/` to explore details
    - **`cobol_reserved.rsw`** configuration file (list of COBOL words) reserved for the converter. It's used during the convert step.

- **`Macro_Expansion/`** folder contains macros to expand into Assembler programs. It's used during the expand_macros step.
- **`macro_settings.json`** configuration file contains customized macro replacement. It's used during the expand_macros step.
- **`macrolib/`** folder contains the Assembler macros to be converted. It's used during the analysis and convert step.
  1. Select `macrolib/`.
  2. By default one Assembler macro named `MACRO1.mac` is provided as a sample file. Delete this file since it's not needed for the analysis.
  3. Upload your Macros in this directory.
- **`project_settings_aux.json`** configuration file contains settings related to code page. It's used during the convert step.
- **`project_settings.json`** configuration file contains settings for the converter. It's used during the convert step.
- **`srclib/`** folder contains the Assembler programs to be converted. It's used during the analysis and convert step.
  1. Choose `srclib/`.
  2. By default, two Assembler programs named `SQtest01.asm` and `SQtest02.asm` are provided as samples. Delete these files as they aren't needed for your analysis and conversion.
  3. Upload your Assembler programs in this directory.

9. Verify the status for `1-awsm2ccm-define-project` step. It should have succeeded under the **Latest build status** tab.

You are ready for the next step: **Code analysis**.

Step 8: Run the analysis and understand the reports

> **Note**
>
> AWS Mainframe Modernization Code conversion *analysis* step is free of charge.

In this step, you kickoff another build:

1. In the left navigation pane, choose **Build projects**.
2. Choose the project you created in step 6.2 to build: `2-awsm2ccm-analysis`.
3. Choose **Start build**, and then **Start now** to generate analysis reports. This will start the build and change status to *in progress*.
4. Choose **Phase details** where you will see the progress of each step orchestrated by the AWS CodeBuild project. Wait until the status changes to *succeeded* for all steps.
5. From the AWS Management Console, go to the Amazon S3 service console.
6. Locate and click on the Amazon S3 bucket: `codebuild-regionId-accountId-bucket`

    a. **`ARTIFACTS/`** folder contains the outputs of *analysis* and *convert* steps.

    b. Choose `ARTIFACTS/prj_codebuild_01/_Reports/`.

    c. The following reports will be available:

    - `AWSM2CCM-Analysis-Report-<timestamp>.pdf` is an executive report that provides the AWS Mainframe Modernization Code conversion billing and scope, improving the conversion, conversion summary, and the detailed conversion statistics. It also summarizes the code counts and billable code counts at a project level and provides metrics and lists of referenced members for each component. It's critical to run and examine this report prior to running the actual conversion.
    - `Conversion_Detailed_Statistics.txt` provides the frequency and expected conversion result (shown as “Conversion status”) for each instruction found in each component. This provides a quick way to identify whether instructions are clear that the converter does not support. Possible *Conversion status* results are:
        - **Totally converted**: the instruction will be accurately converted to COBOL.
        - **Partially converted**: the instruction is supported but uses an unsupported parameter or expression. Manual adjustments are likely required after conversion.
        - **Not converted**: the instruction is not supported by the converter.
        - **Pre-compile instructions to verify**: these are normally included inside the Macros, and refer to what is probably known also as *Conditional Assembly Language* (e.g., AIF, AGO) instructions on mainframe. These are handled by the pre-compiler which is driven by such instructions or directives selects and produces *clean/static* ASM code. These instructions depend on the actual values of the Macro parameters which get compiled. So, the same Macro can generate different pieces of ASM code, depending on the values of the passed parameters. This is because of the presence of such *pre-compile instructions*. In that case, consider expanding or re-engineering the Macro.

- `Conversion_Global_Statistics.txt` provides a summary of the *Conversion status* at a component level.
- `CrossReference_PgmToCpyMacro.txt` reports on Assembler program dependencies on Macros. It provides a quick way to determine if any Macros are missing from the uploaded code.
- `CrossReference_PgmToPgm.txt` reports on Assembler program dependencies on other Assembler programs. It provides a quick way to determine if any Assembler programs are missing from the uploaded code.

7. Return to the AWS CodeBuild service console.
8. Verify the status for **2-awsm2ccm-analysis** step. It should have **succeeded** under the **Latest build status** tab.

You are ready for the next step: **Code conversion**.

Step 9: Run the Code conversion

> **Important**
>
> AWS Mainframe Modernization Code conversion *conversion* step will be billed per your usage. For more information on billing, see the section called “Understand Code conversion billing”.

In this step, you will configure the conversion process, and then start the build.

1. From the AWS Management Console, go to the Amazon S3 service.
2. Locate and click on the Amazon S3 bucket: `codebuild-regionId-accountId-bucket`.
   a. Go to `prj_codebuild_01/`.
   b. Select `project_settings.json`, and choose **Download**.
   c. Open the `project_settings.json` file to see the following JSON structure:

```
{
"Source programs directory":"srclib",
"Source copybooks/macros directory":"macrolib",
"Copybook/Macros Conversion":"Called_only",
"Do not regenerate the Copy/Macro if already exists":"false",
```

```
"Target Compiler":"IBM",
"Endianess":"Big",
"Converted programs extension":"",
"Converted CICS programs extension":"",
"Converted copies/macros extension":"",
"Trace Level":"STANDARD",
"Trace file open mode":"append",
"Data definition level":5,
"Start picture column":40,
"Generate Sync FILLER with name":"FILL-SYNC",
"Use SYNC clause":"yes",
"Decimal Point Comma":"true",
"Original Source Placement":"RIGHT"
}
```

where,

- **Source program directory**: contains the Assembler programs that are needed for the conversion.
- **Source copybooks/Macros directory**: contains the Assembler Macros and copybooks that are needed for the conversion.
- **Copybooks/Macros conversion** can be either:
  - **All**: This radio button denotes that the *full conversion* will convert all copybook/Macros available in the directory irrespective of whether that is being used by the programs or not.
  - **Called_only**: This radio button denotes that the *full conversion* will only convert the copybook/Macros that actually used by the programs.
- > **Important**
  >
  > You don't need to regenerate the Copy/Macro if it already exists.

  When this is true, the tool won't convert the copybook/Macro again, if it's already converted (exists in the output folder).
- **Target**: The conversion of the programs (generated code) depends on the target COBOL compiler. The following options are supported:
  - "IBM" for IBM mainframe

  - "MF" for Micro Focus COBOL
  - “VERYANT” for Veryant isCOBOL
  - “NTT” for NTT DATA Enterprise COBOL (Unikix)
- **Endianess and Bitness**: The conversion of the programs (generated code) depends on the target platform (bit/endianess). This combo allows the selection of the following supported options:
  - Endianess: *Big* (for Big-Endian)/ Little (Little-Endian). For example, IBM z/OS mainframe is Big-Endian, Windows is Little-Endian, Linux varies by distribution (e.g. Amazon Linux 2 on EC2 is Little-Endian).
  - Bitness: 32/64 (if not given, default will be 32). The recommended setting is 32 bits.
- **Converted program extension**: This is to set the file extension for the generated COBOL programs. Empty (“”): no extension. For Rocket Software (formerly Micro Focus) COBOL targets, *CBL* is recommended to enable Rocket Enterprise Developer to correctly recognize the files.
- **Converted CICS program extension**: This is to set the file extension for the generated CICS COBOL programs. Empty (“”):: no extension. For Rocket Software COBOL targets, *CBL* is recommended to enable Rocket Enterprise Developer to correctly recognize the files.
- **Converted copybooks/Macros extension**: This is to set the file extension for the generated COBOL copybooks. Empty (“”):: no extension. For Rocket Software COBOL targets, *CPY* is recommended to enable Rocket Enterprise Developer to correctly recognize the files.
- **Trace level**: Trace is the information that is logged using CodeBuild during the conversion. The user can select the level of detail by selecting any one of the provided options.
  - **ERROR** = TRACE ERROR: only conversion errors are displayed.
  - **STANDARD** = TRACE STANDARD: conversion errors and standard information are displayed. This is the recommended setting.
  - **ALL** = TRACE ALL: maximum level of tracing
- **Trace file open mode**: Not used. Default setting of*append* is recommended.
- **Data definition level**: This indicates the initial level of the sub-fields (after level “01”) defined in working-storage and linkage section. Must be a number.

- **Start picture column**: This is about the format of the generated COBOL code and indicates the column where the *PIC* clause is placed (after the field names). Must be a number.
- **Original source placement:** This indicates the position where the comments are placed in the program. It has two options:
  - **RIGHT**: This option will place the comment or additional information at the right position after seventy-third (73) column. In COBOL the code is written in the first seventy-two (1-72) columns and anything from the seventy-third (>= 73) column will be treated as a comment.
  - **ABOVE**: This option will place the comment above the translated content.
- **Generate Sync FILLER with name**: This option is related to the alignment in memory of binary fields (Assembler "H", "F", "D" data-types, which are converted to COBOL "COMP" data-type). In order to guarantee the proper *alignment boundary*, explicit *filler* fields will be added during the conversion. This is a text based option, the value must be a string (like FILL-SYNC).
- **Use SYNC clause**: This option refers to the alignment in memory of binary fields. Yes = all the fields converted to COBOL. "COMP" will be defined with the clause "SYNC" (e.g.,, 05 WRKFLD PIC S9(09) COMP SYNC).
- **Decimal point comma**: When this is true, the *DECIMAL-POINT IS COMMA* clause will be added to the "SPECIAL-NAMES" COBOL paragraph.

d. Based on your requirements, change appropriate parameters, and then save the `project_settings.json`.

e. Remove the existing `project_settings.json` file from `prj_codebuild_01/` in Amazon S3 bucket, and then upload the new version.

3. Go back to the AWS CodeBuild service.
4. Select the project to build you created previously : `3-awsm2ccm-convert`

   a. Choose **Start build**, and then **Start now** to convert Assembler programs and Macros to COBOL programs and copybooks.

   b. Wait for the build status to change to **Succeeded** for this project. It will be under the **Latest build status** tab.

Step 10: Verify the Code conversion

1. From the AWS Management Console, go to Amazon S3 service.
2. Locate and click on the Amazon S3 bucket: `codebuild-regionId-accountId-bucket`.
3. Navigate to **`awsm2ccm-do-not-delete`** . AWS Mainframe Modernization Code conversion creates encoded binary files for each Assembler or Macro module during the conversion process. These files are essential for preventing duplicate billing to the customers and also to track how much of the provided Assembler code was analyzed and converted. Files are stored in the following location: `codebuild-regionId-accountId- bucket/awsm2ccm-do-not-delete/<your_AWS_account_id>/Hash`. The encoded files do not contain any Assembler code and It is also not possible to extract customer code from these files.

   > **Important**
   >
   > Neither manually edit these files or delete these files. Editing or deleting these files may result in multiple billings for the same components.

   Treat the **`awsm2ccm-do-not-delete/`** folder as a system-managed directory. Consult Support before making any changes to this directory or its contents.
4. Click `codebuild-regionId-accountId-bucket` to go back to the bucket.
5. Choose **`ARTIFACTS/prj_codebuild_01/`**.**`_Converted/`** folder contains the generated COBOL outputs as a result of Code conversion step. It will have the following subdirectories:

   - **copybooks/** folder contains the generated COBOL copybooks.
   - **programs/** folder contains the generated COBOL programs.
   - **runtime_lib/** folder contains additional COBOL programs and copybooks provided by the solution.
6. If the *Analysis Reports* and other reports indicate that the conversion was successful, and the AWS CodeBuild project `3-awsm2ccm-convert` is marked **Succeeded**, download the COBOL code and copybooks from the **_Converted/** directory.

Step 11: Download converted code

In this step, download the COBOL code and copybooks from the **_Converted/** directory, and compile them in the target COBOL environment.

1. From the AWS Management Console, go to Amazon S3 service.
2. Locate and click on the Amazon S3 bucket: `codebuild-regionId-accountId-bucket`.
3. Navigate to the location: `ARTIFACTS/prj_codebuild_01/_Converted/`.
4. Download the converted COBOL code from all the subdirectories under **_Converted/**. You can also use the following CLI command to download them at once:

```
aws s3 cp s3://codebuild-regionId-accountId-
bucket/ARTIFACTS/prj_codebuild_01/_Converted/ . --recursive
```

5. Analyze and compile the converted COBOL in the target COBOL environment.

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*.

Charon integration

Introduction to Charon-SSP

In 1987, Sun Microsystems released the SPARC V7 processor, a 32-bit RISC processor. The SPARC V8 followed in 1990 - a revision of the original SPARC V7, with the most notable inclusion of hardware divide and multiply instructions. The SPARC V8 processors formed the basis for a number of servers and workstations such as the SPARCstation 5, 10 and 20. In 1993, the SPARC V8 was followed by the 64-bit SPARC V9 processor. This too became the basis for a number of servers and workstations, such as the Enterprise 250 and 450.

Due to hardware obsolescence and lack of spare or refurbished parts, software and systems developed for these older SPARC-based workstations and servers have become harder to maintain. To fill the continuous need for certain, end-of-life SPARC-based systems, Stromasys S.A. developed the Charon-SSP line of SPARC emulator products. The following products are software-based, virtual machine replacements for the specified native- hardware SPARC systems. The following is a general overview of the emulated hardware families.

Charon-SSP/4M emulates the following SPARC hardware:

- Sun-4m family (represented by the Sun SPARCstation 20): originally, a multiprocessor Sun-4 variant, based on the MBus processor module bus introduced in the SPARCServer 600MP series. The Sun-4m architecture later also encompassed non-MBus uniprocessor systems such as the SPAR Cstation 5, utilizing SPARC V8-architecture processors. Supported starting with SunOS 4.1.2 and by Solaris 2.1 to Solaris 9. SPARCServer 600MP support was dropped after Solaris 2.5.1.

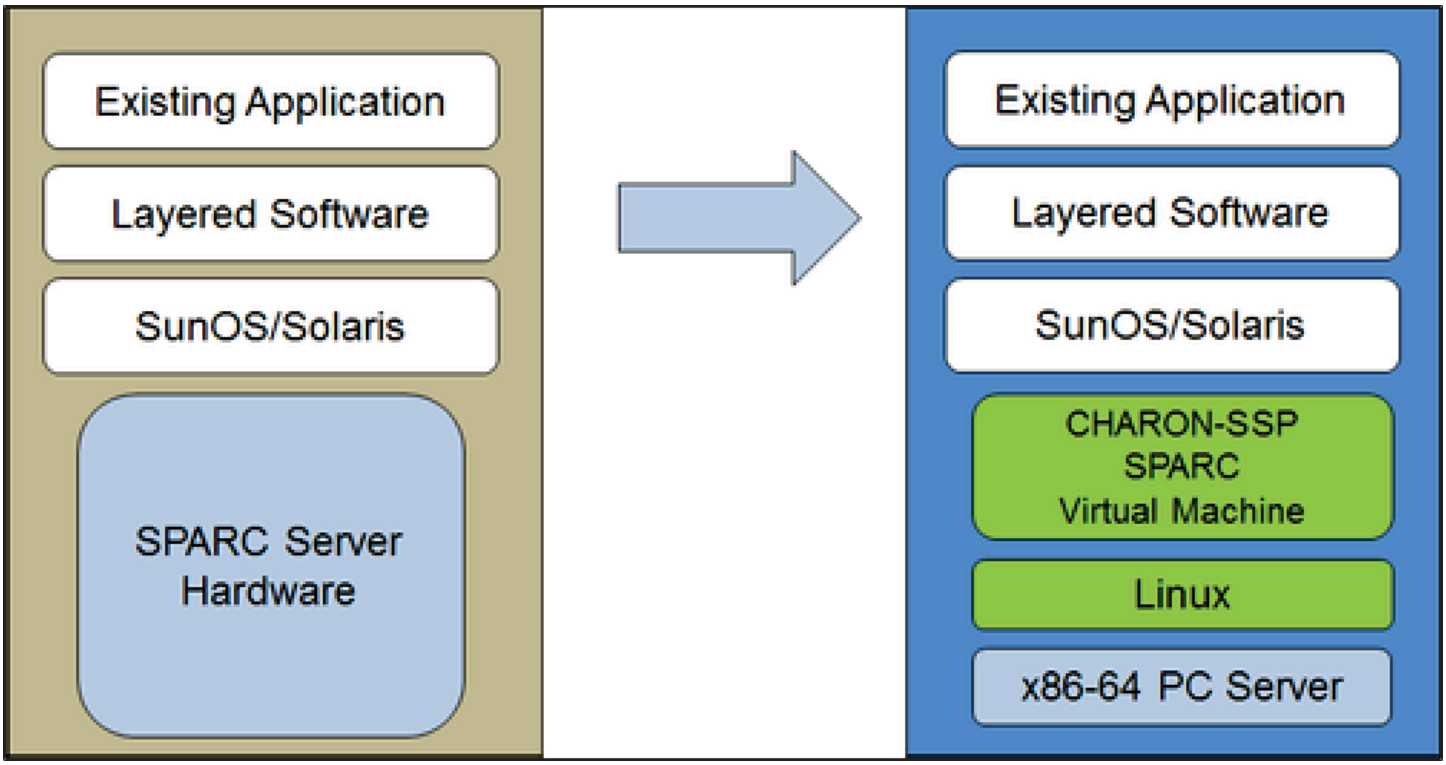
Charon-SSP/4U(+) emulates the following SPARC hardware:

- Sun-4u family (represented by the Sun Enterprise 450): (U for UltraSPARC) - this variant introduced the 64-bit SPARC V9 processor architecture and UPA processor interconnect first used in the Sun Ultra series. Supported by 32-bit versions of Solaris starting from version 2.5.1. The first 64-bit Solaris release for Sun-4u was Solaris 7. UltraSPARC I support was dropped after Solaris 9. Solaris 10 supports Sun-4u implementations from UltraSP ARC II to UltraSPARC IV.

Charon-SSP/4V(+) emulates the following SPARC hardware:

- Sun-4v family (represented by the SPARC T2 and T4): this variation added hypervisor processor virtualization to the Sun-4u; introduced in the Ultra SPARC T1 multicore processor. Selected hardware was supported by Solaris version 10 starting from release 3/05 HW2 (most models - including the hardware emulated by Charon-SSP - require newer versions of Solaris 10). Several Solaris 11 versions are also supported.

The following image shows the basic concept of migrating physical hardware to an emulator.



The Charon-SSP virtual machines allow users of Sun and Oracle SPARC-based computers to replace their native hardware in a way that requires little or no change to the original system configuration. This means you can continue to run your applications and data without the need to switch or port to another platform. The Charon-SSP software runs on commodity, Intel 64-bit systems ensuring the continued protection of your investment.

Charon-SSP/4U+ supports the same virtual SPARC platforms as Charon-SSP/4U, and Charon-SSP/4V+ the same as Charon-SSP/4V. However, the 4U+ and 4V+ versions take advantage of Intel's VTx/EPT and AMD's AMD-v/NPT hardware assisted virtualization technology in modern CPUs to offer better virtual CPU performance. Charon-SSP/4U+ and Charon-SSP/4V+ require CPUs with VT-x/EPT or AMD-v/NPT support and must be installed on a dedicated host system. Running these product variants in a VM (e.g., on VMware) is not supported.

> **Note**
>
> If you plan to run Charon-SSP/4U+ or 4V+ in a cloud environment, contact Stromasys or a Stromasys VAR to discuss your requirements.

Supported guest operating systems

The Charon-SSP/4M virtual machines support the following guest operating system releases:

- SunOS 4.1.3 - 4.1.4
- Solaris 2.3 to Solaris 9

The Charon-SSP/4U(+) virtual machines support the following guest operating system releases:

- Solaris 2.5.1 to Solaris 10

The Charon-SSP/4V(+) virtual machines support the following guest operating system releases:

- Solaris 10 (starting with update 4, 08/07) and Solaris 11.1 to Solaris 11.4

For Charon-SSP/4V(+), note the following:

- For the emulated SPARC T4, supported Solaris 10 versions are: Oracle Solaris 10 1/13, Oracle Solaris 10 8/11, and Solaris 10 9/10, or Solaris 10 10/09 with the Oracle Solaris 10 8/11 patch set.
- The emulated SPARC T4 model is a prerequisite for running Solaris 11.4 in the emulator.
- Solaris kernel zones are not supported.

Charon-SSP cloud instance prerequisites

By selecting an instance type or shape, you select the virtual hardware that will be used for the Charon-SSP host instance in the cloud. Therefore, the selection of an instance type or shape determines the hardware characteristics of the Charon-SSP virtual host hardware (e.g., how many CPU cores and how much memory your virtual Charon host system will have).

Note

If you use a Charon-SSP marketplace image to launch your instance, all Linux host operating system requirements are fulfilled.

The minimum hardware requirements are described below.

Important points regarding the sizing guidelines:

- The sizing guidelines below-in particular regarding number of host CPU cores and host memory-show the minimum requirements. Every deployment situation must be reviewed and the actual host sizing has to be adapted as necessary. For example, the number of CPU cores available for I/O must be increased if the guest applications produce a high I/O load. Also, a system with many emulated CPUs is typically able to create a higher I/O load and thus the number of CPU cores available for I/O may have to be increased. In a hyper-threading environment, for best performance, the number of CPU cores (that is, real/physical CPUs) must be sufficient to fulfill CPU requirements of the active emulators, thus avoiding high-workload threads sharing one physical CPU core.
- The CPU core allocation for emulated CPUs and CPU cores for I/O processing is determined by the configuration. See CPU Configuration in the general Charon-SSP User's Guide for more information about this and the default allocation of CPU cores for I/O processing.

Important general information

- To facilitate a fast transfer of emulator data from one cloud instance to another, it is strongly recommended to store all relevant emulator data on a separate disk volume that can easily be detached from the old instance and attached to a new instance.
- Make sure to dimension your instance correctly from the beginning (check the minimum requirements below). The Charon-SSP license for Charon-SSP AL is created when the instance is first launched. Changing later to another instance size/type and thereby changing the number of CPU cores will invalidate the license and thus prevent Charon instances from starting (new instance required). If planning to use the Charon-SSP AL instance in AutoVE mode, be sure to include the AutoVE server information before first launch, otherwise the public license servers will be used. The license for Charon-SSP VE is created based on the fingerprint taken on the license server. If the license server is run

directly on the emulator host and the emulator host later requires, for example, a change in the number of CPU cores, the license will be invalidated (new license and possibly new instance required).

Instance prerequisites

General CPU requirements: Charon-SSP supports modern x86-64 architecture processors based Amazon EC2 instances.

Minimum requirements for Charon-SSP:

- Minimum number of host system CPU cores:
  - At least one CPU core for the host operating system, plus:
  - For each emulated SPARC system:
    - One CPU core for each emulated CPU of the instance, plus:
    - At least one additional CPU core for I/O processing (at least two, if server JIT optimization is used). See the CPU Configuration section mentioned above for configuration options. By default, Charon will assign 1/3 (min. 1; rounded down) of the number of CPUs visible to the Charon host to I/O processing.
- Minimum memory requirements:
  - 4GB or more of RAM for the Linux host operating system. The actual requirements may be higher and will depend on the requirements of the non- emulator services running on the Linux host. The previous recommendation of at least 2GB of RAM for the Linux host will still be valid for many systems, but the increasing requirements of the Linux operating system and applications have led to the updated recommendation for new installations. Plus:
  - For each emulated SPARC system:
    - The configured memory of the emulated instance, plus:
    - 2GB of RAM (6GB of RAM if server JIT is used) to allow for DIT optimization, emulator requirements, run-time buffers, SMP and graphics emulation.
- If hyper-threading is enabled on modern x86-64 CPUs, two threads can run on one physical CPU core providing two logical CPUs to the host operating system. If possible, disable hyper-threading on the Charon-SSP host. However, this is frequently not possible in VMware and cloud environments, or it is unclear whether hyper-threading is used or not. The Charon-SSP hyper-threading option enables Charon-SSP to adapt to such environments. See the CPU

Configuration section in your general Charon-SSP User's Guide mentioned above for detailed configuration information. P lease note: for best performance, Charon-SSP threads should not share a physical CPU core - enough physical cores should be available on the host system to satisfy the requirements of the configured emulator(s).

- One or more network interfaces, depending on customer requirements.
- Charon-SSP/4U+ and Charon-SSP/4V+ must run on physical hardware supporting Intel VT-x/EPT or AMD-v/NPT (baremetal instances) and therefore cannot run in all cloud environments. Please check your cloud provider's documentation for the availability of such hardware. In addition, note the following points:
  - Charon-SSP/4U+ and Charon-SSP/4V+ are only available when using a Linux kernel supported by Stromasys.
  - If you need this type of emulated SPARC hardware, contact Stromasys or your Stromasys VAR to discuss your requirements in detail.

Creating and configuring an AWS cloud instance for Charon (New GUI)

This section reflects the AWS Management Console in spring 2022. If you still use the older console, refer to the Appendix of the Charon-SSP AWS Getting Started guide.

General prerequisites

This description shows the basic setup of a Linux instance in AWS. It does not list specific prerequisites. However, depending on your use case, consider the following prerequisites:

- Amazon account and AWS Marketplace subscriptions
  - To set up a Linux instance in AWS, you need an AWS account with administrator access.
  - Identify the AWS Region in which you plan to launch your instance. Ensure that AWS services that you plan to use are available in that Region. See AWS Services by Region.
  - Identify the VPC and subnet in which you plan to launch your instance.
  - If your instance requires internet access, ensure that the route table associated with your VPC has an internet gateway. If your instance requires VPN access to your on-premises network, ensure that a VPN gateway is available. The exact configuration of your VPC and its subnets will depend on your network design and application requirements.

- To subscribe to a specific AWS Marketplace service, choose **AWS Marketplace Subscriptions** in the AWS Management Console and then choose **Manage subscriptions**.
- Search for the service that you plan to use and subscribe to it. After a successful subscription, you will find the subscription in the **Manage subscriptions** section. From there you can directly launch a new instance.

- The instance hardware and software prerequisites will be different depending on the planned use of the instance:
  - Option 1: the instance is to be used as a Charon emulator host system:
    - Refer to the hardware and software prerequisite sections of the User's Guide and/or Getting Started guide of your Charon product to determine the exact hardware and software prerequisites that must be fulfilled by the Linux instance. The image you use to launch your instance and the instance type you chose determine the software and hardware of your cloud instance.
    - A Charon product license is required to run emulated legacy systems. Refer to the licensing information in the documentation of your Charon product, or contact your Stromasys representative or Stromasys VAR for additional information.
  - Option 2: the instance is to be used as a dedicated VE license server:
    - See the VE License Server Guide for detailed prerequisites.
- Certain legacy operating systems that can run in the emulated systems provided by Charon emulator products require a license of the original vendor of the operating system. The user is responsible for any licensing obligations related to the legacy operating system and has to provide the appropriate licenses.

Using the AWS Management Console to launch a new instance

To create a new instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Launch instance**.
3. Enter a name for the instance.
4. Select an AMI. An AMI is a prepackaged image used to launch cloud instances. It includes the operating system and applicable application software. The choice of AMI depends on how you plan to use the instance:

- If the instance is to be used as a Charon emulator host system several AMI choices are possible:
  - Installing the Charon host system from a prepackaged Charon marketplace image: they contain the underlying operating system and the preinstalled Charon software.
    - Check with your Stromasys representative which options are currently available in your cloud providers marketplace.
    - Depending on the cloud provider and the Stromasys product release plans, there can be two variants:
      - Automatic licensing (AL) for use with a public, Stromasys-operated license server, or with a private, customer-operated AutoVE license server
      - Virtual environment (VE) for use with a private, customer-operated VE license server
  - Installing the Charon host system using a conventional Charon emulator installation with the Charon emulator installation RPM packages for Linux:
    - Choose a Linux AMI of a distribution supported by your selected Charon product and version. See the user guide for your product on the Stromasys documentation site.
- If the instance is to be used as a dedicated VE license server, see the VE License Server Guide in Licensing Documentation for the requirements of the Linux instance.

After you decide which AMI is required, select a matching Linux or Charon product AMI. If you don't see the AMI that you need, choose **Browse more AMIs**. Choose the Linux AMI that matches how you plan to use the instance. It can be one of the following:

- A prepackaged Charon VE marketplace image. The name of the AMI will include the string "ve".
- A prepackaged Charon AL marketplace image for Automatic Licensing or AutoVE.
- A Linux version supported for an RPM product installation.
- A Linux version supported for the VE license server.

5. Select an instance type. Amazon EC2 offers instance types with varying combinations of CPU, memory, storage, and networking capacity. Select an instance type that matches the requirements of the Charon product that you want to use. Some marketplace images have a restricted selection of instance types.

6. Select an existing key pair or create and save a new one. If you select an existing key pair, make sure you have the matching private key. Otherwise, you will not be able to connect to your instance.

   > **Note**
   >
   > If your management system supports it, for RHEL 9.x, Rocky Linux 9.x, and Oracle Linux 9.x, use SSH key type ECDSA or ED25519. These types allow you to connect to these Charon host Linux systems by using an SSH tunnel without needing to change the the default crypto-policy settings on the Charon host to less secure settings. For example, this is important for the Charon-SSP Manager. See Using system-wide cryptographic policies in the *Red Hat* documentation.

7. In the **Network settings** section, choose **Edit**. Choose the settings that correspond to your environment.

   - Specify a VPC.
   - Specify an existing subnet or create a new one.
   - Enable or disable the automatic assignment of a public IP address to the primary interface. Automatic assignment is only possible if the instance has a single network interface.
   - Assign an existing or new custom security group. The security group must allow at least SSH to access the instance. Any ports required by applications that you plan to run on the instance must also be allowed. You can modify the security group at any time after you create the instance.

8. In the **Storage** section, for the root volume (the system disk), choose a size that is appropriate for your environment. The recommended minimum system disk size for the Linux system is 30 GiB. To provide space for virtual disk containers and other storage requirements, you can add more storage now or after you launch the instance. But the system disk size must cover the Linux system requirements, including any applications and utilities that you plan to install.

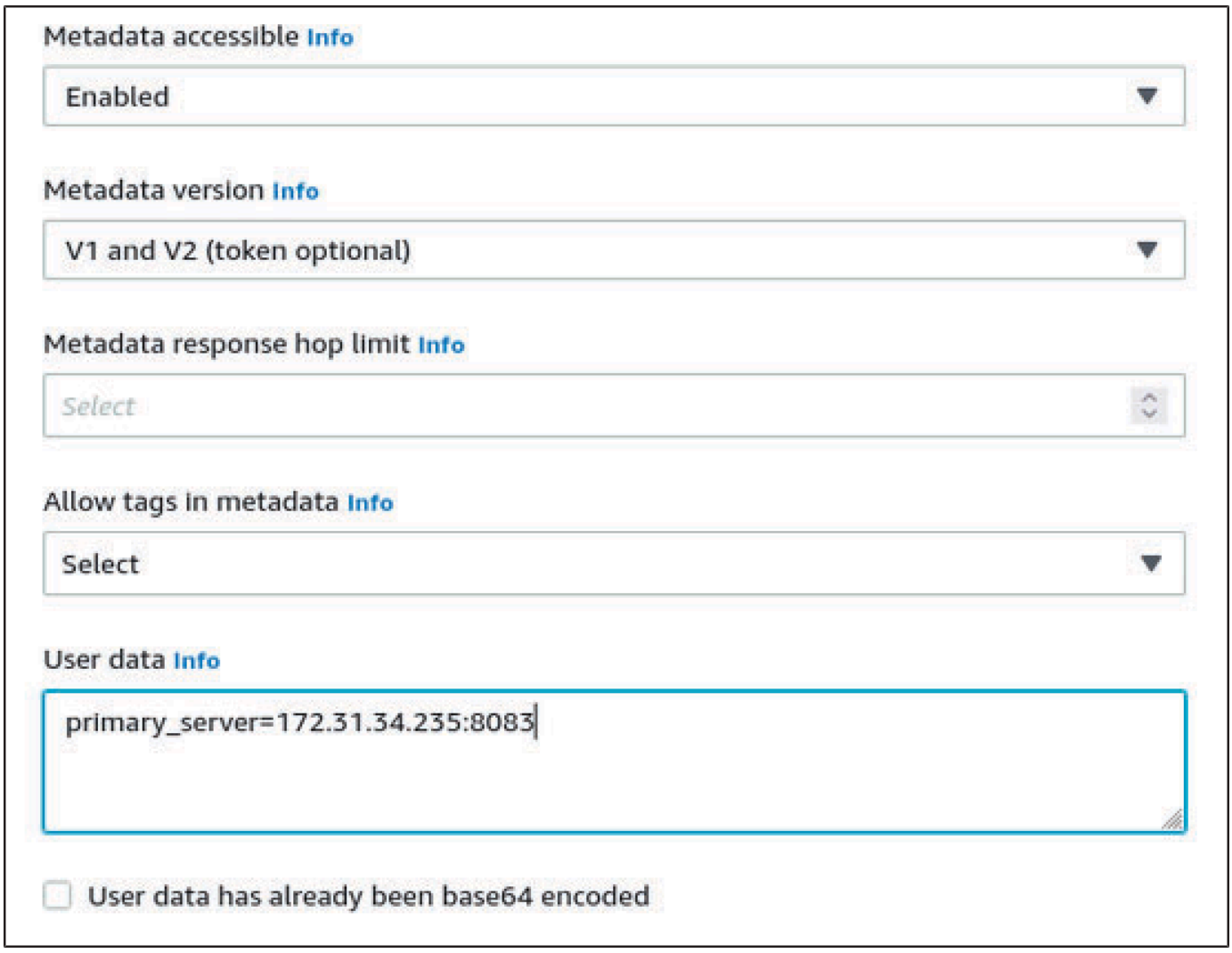
   > **Note**
   >
   > We recommend that you create separate storage volumes for Charon application data (e.g., disk images). If necessaryou, you can later migrate such volumes to another instance.

9. Expand the **Advanced details** section, scroll down, and select **Specify CPU options**. Three that are more likely to be useful to a Charon emulator environment are shown in the following image as examples.



10. For a VE license server system with a version earlier than 1.1.23, you must assign the required IAM role to the instance. It must be a role that allows the `ListUsers` action. To assign a role, in the expanded **Advanced details** section either select a role under **IAM instance profile**, or choose **Create a new IAM profile**. For more information, see IAM roles for Amazon EC2.
11. If your instance is based on a Charon AL AWS Marketplace image and you plan to use the Stromasys-operated public license servers, you must add the corresponding information to the instance configuration before you launch the instance.

    Enter the information for the AutoVE license server as shown in the following image.



The following are valid user data configuration options:

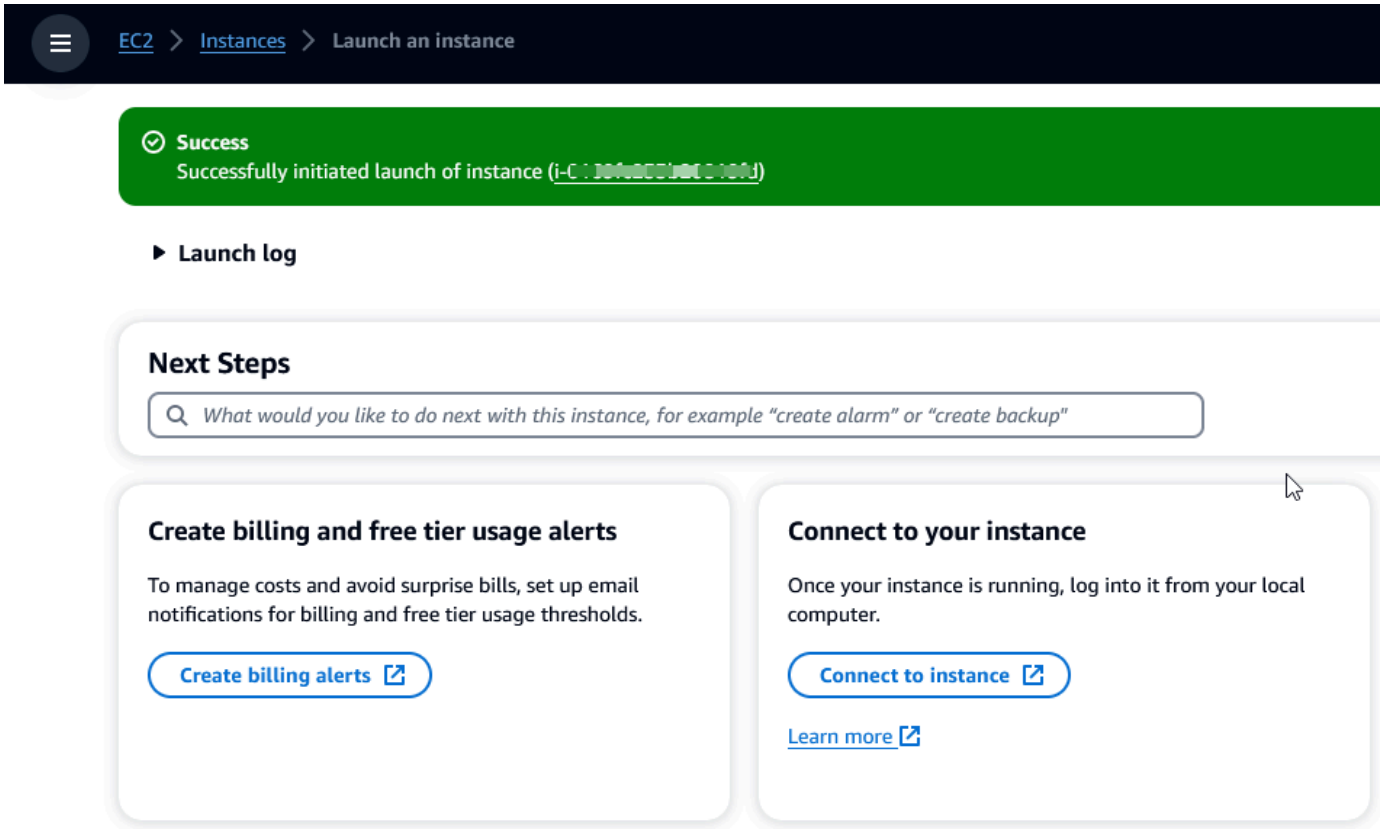
- **`primary_server=`***`<ip-address>`*`[:`*`<port>`*`]`
- **`backup_server=`***`<ip-address>`*`[:`*`<port>`*`]`

Where

- *<ip-address>* stands for the IP address of the primary and the backup server as applicable.
- *<port>* stands for a non-default TCP port used to communicate with the license server (default: TCP/8083).

> **Note**
>
> At least one license server must be configured at initial launch to enable AutoVE mode. 0therwise, the instance will bind to one of the public license servers operated by Stromasys.

12. In the **Summary** section, choose **Launch instance**. After a while, you will see the following success message:



13. At the bottom-right corner of the screen, choose **View all instances**.
14. To see the details of your instance, select the check box to the left of the row that represents the instance in the **Instances** table. Your instance details will appear in the bottom half of the screen. For information on how to connect to your instance, see Connect in the Amazon EC2 User Guide.

AWS Mainframe Modernization replatforming with NTT DATA

AWS Mainframe Modernization offers a variety of Amazon Machine Images (AMIs). These AMIs facilitate the rapid provisioning of Amazon EC2 instances, creating a tailored environment for rehosting and replatforming mainframe applications in AWS by using NTT Data. This guide provides the steps required to access and use these AMIs.

Prerequisites

- Ensure that you have administrator access to an AWS account where you can create Amazon EC2 instances.
- Verify that the AWS Mainframe Modernization service is available in the Region where you plan to create the Amazon EC2 instances. See List of AWS Services Available by Region.
- Identify the Amazon VPC where you want to create the Amazon EC2 instances.

Subscribe to the Amazon Machine Image

When you subscribe to an AWS Marketplace product, you can launch an instance from the product's AMI.

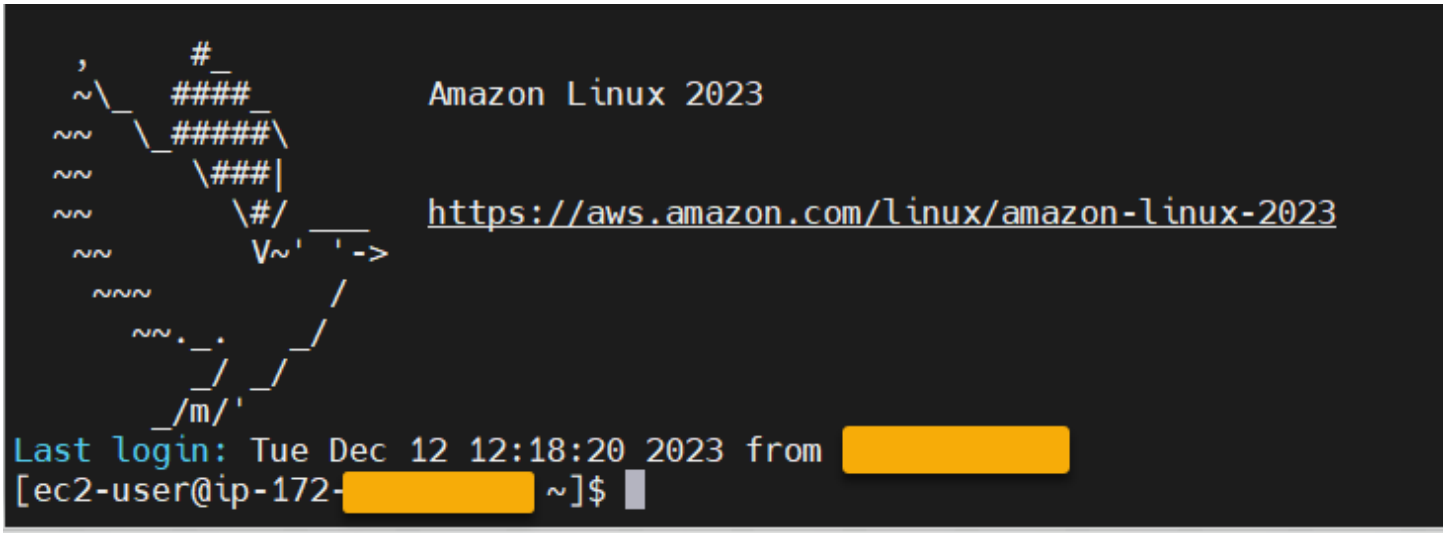
1. Sign in to the AWS Management Console and open the AWS Marketplace console at https://console.aws.amazon.com/marketplace.
2. Choose **Manage subscriptions**.
3. Copy and paste the following link into the browser address bar: https://aws.amazon.com/marketplace/pp/prodview-eg227ymldsnx2
4. Choose **Continue to Subscribe**.
5. If the terms and conditions are acceptable, choose **Accept Terms**. The subscription might take a few minutes to process.
6. Wait for a thank-you message to appear. This message confirms that you have successfully subscribed to the product.
7. In the left navigate pane, choose **Manage subscriptions**. This view shows you all of your subscriptions.

Launch AWS Mainframe Modernization replatform with NTT DATA instance

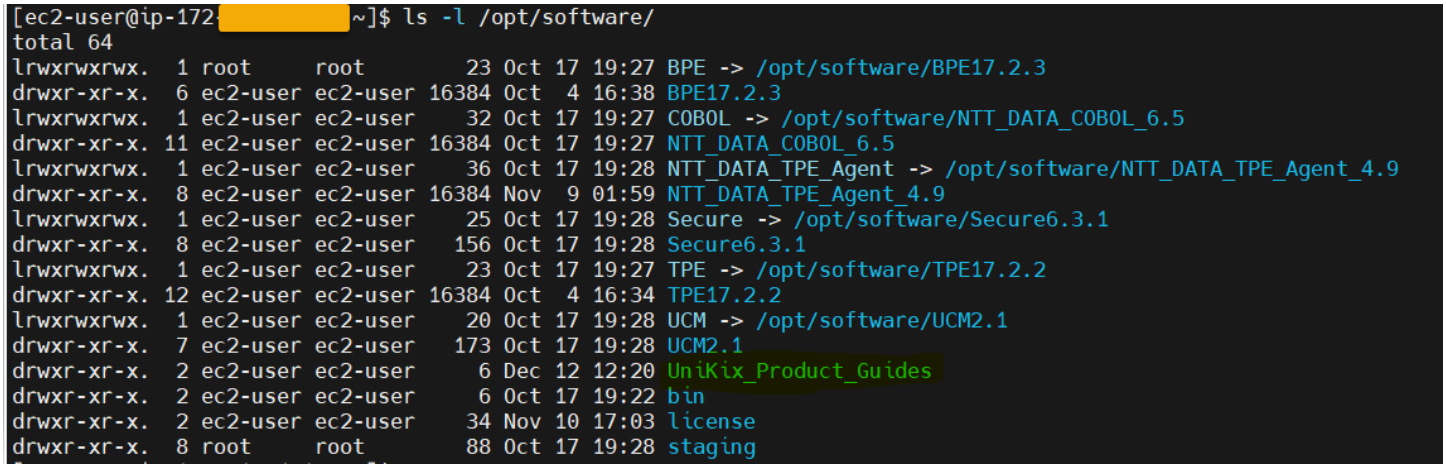
1. Open the AWS Marketplace console at https://console.aws.amazon.com/marketplace.
2. In the left navigation pane, choose **Manage subscriptions**.
3. Find the AMI that you want to launch, and choose **Launch new instance**.
4. Under **Region**, select the allow-listed Region.
5. Choose **Continue to launch through EC2**. This action takes you to the Amazon EC2 console.
6. Enter a name for the server.
7. Select an instance type that matches your project performance and cost requirements. The suggested starting point for instance size is `c5.2xLarge`.
8. Choose an existing key pair or create and save a new one. For information about key pairs, see Amazon EC2 key pairs and Linux instances in the Amazon EC2 User Guide.
9. Edit the network settings and choose the allow-listed VPC and appropriate subnet.
10. Choose an existing security group or create a new one. If this is an Enterprise Server Amazon EC2 instance it is typical to allow TCP traffic to ports 86 and 10086 to administer the Rocket Software (formerly Micro Focus) configuration.
11. Configure the storage for the Amazon EC2 instance.
12. Review the summary and choose **Launch instance**. For the launch to success, the instance type must be valid. If the launch fails, choose **Edit instance configuration** and choose a different instance type.
13. After you see the success message, choose **Connect to instance**.
14. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
15. In the left navigation pane, under the **Instances** menu, choose **Instances**.
16. In the main pane, check the status of your instance.

Getting started with NTT Data

After you provision the Amazon EC2 instance, SSH into it with the user name `ec2-user`. The screen will look like the following image.



Under the `/opt/software/` folder, there is a folder named `UniKix_Product_Guides`, as shown in the following image.



The `UniKix_Product_Guides` folder includes the documentation for the following components that are installed on this Amazon EC2 instance:

- NTT DATA TPE
- NTT DATA BPE
- NTT DATA Enterprise COBOL
- NTT DATA UniKix Secure
- NTT DATA UniKix Central Manager

The `software` folder that appears in the previous image has the binaries for the components that are listed above.

After you successfully validate the Amazon EC2 instance, get started using AWS Mainframe Modernization Replatform with NTT DATA by following the NTT Data documentation.

Tutorial: Deploy CardDemo application on NTT DATA

This page guides you through the step-by-step process for deploying the CardDemo sample application on the AWS Mainframe Modernization replatform with NTT DATA Unikix runtime.

The CardDemo sample application is a simplified mainframe application designed and developed to test and showcase AWS and partner technology for mainframe migration and modernization use-cases.

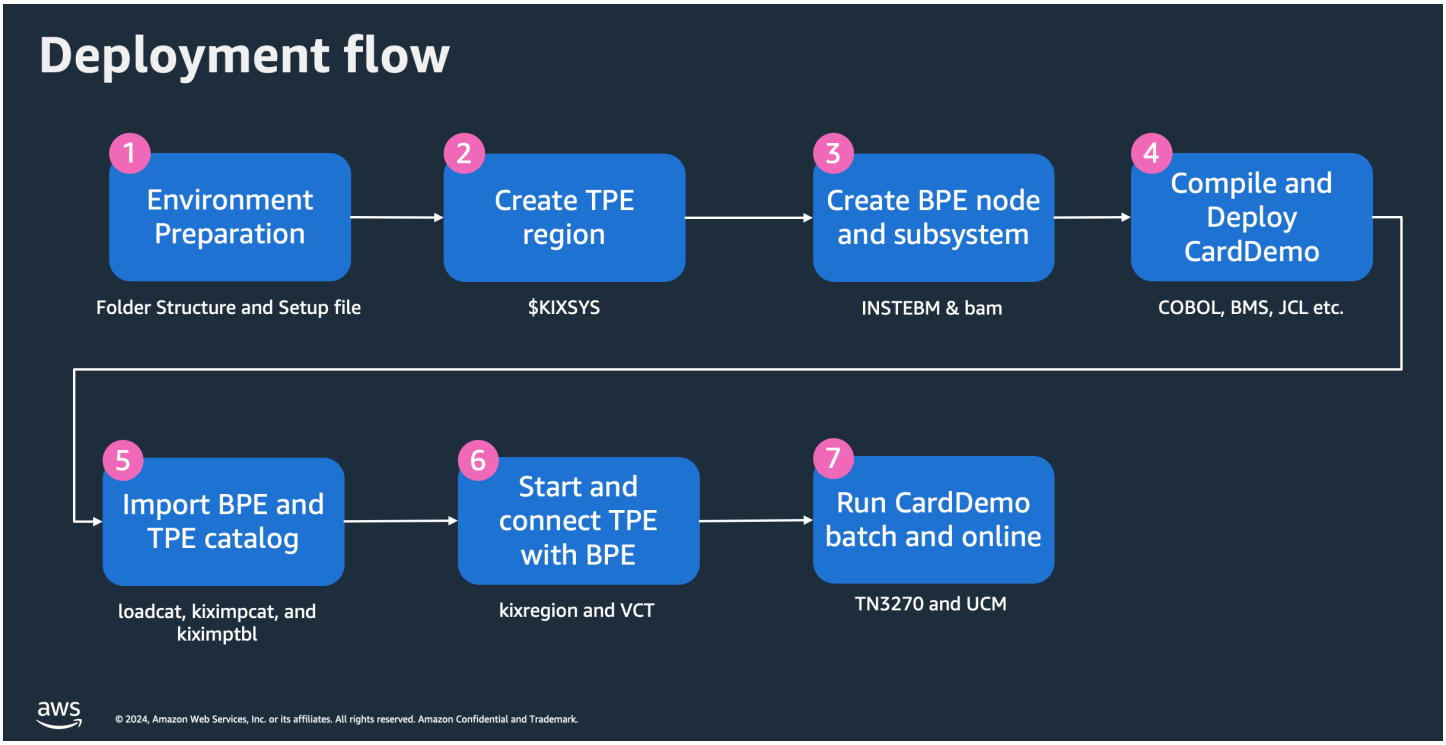
For more information about this application, see, GitHub repository for CardDemo.

Topics

- Deployment flow diagram
- Prerequisites
- Step 1: Prepare the environment
- Step 2: Create a TPE region
- Step 3: Create the BPE node and subsystem
- Step 4: Compile and deploy CardDemo application
- Step 5: Import BPE and TPE catalog
- Step 6: Start and connect TPE with BPE
- Step 7: Run the CardDemo application
- Troubleshooting

Deployment flow diagram

The following diagram shows each step in the workflow for deploying an application on the NTT DATA Unikix runtime.



Prerequisites

- Follow the instructions provided in the *Replatforming with NTT DATA* using the NTT DATA UniKix Marketplace AMI.
- Modify the instance metadata option **IMDSv2** to **Optional** as mentioned in Restore the use of IMDSv1 in the *Amazon EC2 user guide*.
- Download CardDemo runtime components for NTT DATA UniKix from the GitHub repository.
- Log in to UniKix runtime EC2 instance as `ec2-user`.
- Extract the downloaded CardDemo runtime components using this link: UniKix_CardDemo_runtime_v1.zip.
  - The extracted directory should contain `bin` and `migrated_app` directories.
  - Move both `bin` and `migrated_app` directories under your `$HOME` directory. The path would look like `/home/ec2-user`.
  - You should have the following directories in your `$HOME`:
    - `/home/ec2-user/bin`
    - `/home/ec2-user/migrated_app`

- Move all the files inside the $HOME/bin directory the following command:
- • `chmod +x $HOME/bin/*`

Step 1: Prepare the environment

After completing the prerequisites, the first step is to prepare the environment where you want to deploy the CardDemo application.

1. Log in to UniKix runtime EC2 instance as `ec2-user`.
2. Observe the list of UniKix software that are prepackaged within the AMI, such as TPE, BPE, and COBOL, along with others from the NTT DATA UniKix product location by using the following command in your EC2 instance:

   ```
   ls -l /opt/software/
   ```

3. Examine the migrated CardDemo application. You will see all the source code, including BMS maps, COBOL programs, COBOL Copybooks, and JCLs. You will also find the export of BPE and TPE catalogs, CICS resource definitions, and the migrated data such as sequential files and VSAM files by doing this:

   ```
   ls  $HOME/migrated_app/*/*
   ```

4. Create folder structure by running the `create_project` script with the following command:

   ```
   sh $HOME/bin/create_project
   ```

5. Activate the CardDemo environment by sourcing the `carddemo.env` setup file using:

   ```
   source $HOME/bin/carddemo.env
   ```

Step 2: Create a TPE region

Once you have activated the environment where you want to deploy the application, you need to create a TPE region.

1. Create a TPE region using the `kixregion createRegion` command which requires inputs such as `$KIXSYS`, `$JAVA_HOME`, and `$KIXLICDIR`. These environment variables are already setup in the `carddemo.env` setup file.

```
kixregion createRegion $KIXSYS $JAVA_HOME $KIXLICDIR
```

2. Configure the TPE region using the `kixregion setAttr` command.

```
kixregion setAttr $KIXSYS server.tx.languages.cobol.enabled true
kixregion setAttr $KIXSYS server.tx.languages.cobol.flavor vcobol
kixregion setAttr $KIXSYS server.tx.languages.cobol.home $VCOBOL
kixregion setAttr $KIXSYS maps.location $PROJECT_ROOT/maps
kixregion setAttr $KIXSYS programs.location $PROJECT_ROOT/loadlib
kixregion setAttr $KIXSYS environment.KIXDATA $KIXDATA
kixregion setAttr $KIXSYS td.jobq.submission.node $EBMHOME
kixregion setAttr $KIXSYS td.jobq.submission.subsys $EBMSYS
```

3. Generate the user environment file specific to this TPE region by executing the `kixregion createScript` command. This command creates or updates `$KIXSYS/bin/userenv` based on the TPE region configuration.

```
kixregion createScript $KIXSYS
```

4. Activate TPE region by sourcing the user environment file (`$KIXSYS/bin/userenv`).

```
source $KIXSYS/bin/userenv
```

5. Build the TPE region by running the `kixinstall2` command.

```
kixinstall2
```

Step 3: Create the BPE node and subsystem

After creating a TPE region, you need to create the BPE node and subsystem by following these steps.

1. Change the ownership and permissions of `INSTEBM`.

```
sudo chown root $INSTEBM
```

```
sudo chmod 4755 $INSTEBM
```

2. Create a BPE node using the `INSTEBM` command. The BPE node directory is provided as the input parameter.

```
$INSTEBM $EBMHOME
```

3. Activate the batch environment by sourcing the `batchenv` file from the newly created BPE node.

```
source $EBMHOME/batchenv
```

4. Create the BPE subsystem within this node using the Batch Administration Manager (bam). The `bam` command will open the Batch Administration Manager interface.

```
bam
```

   a. Start the BPE node using the BAM interface. Choose option 2, **System Environments** from the main menu.



   b. Choose option 2, **Start/(Stop) Batch Node** to start the BPE node.

```
                System Environments                          2024/09/17 21:27:03




             1   Report System Status
             2   Start/(Stop) Batch Node
             3   Assign the Console
             4   Change the Date
             5   Redirect Job Output
             6   Inter-Node Communications
             7   Job Accounting
             8   Assign the Initial Job Number
             9   Enable/Disable Duplicate Name Execution Delay : Enabled

             H - Help
             R - Return to Main Menu

--------------------------------------------------------------------------------
Type an option and press Return: 2
```

c. Once started, press the **Return key** twice to return to the BAM main menu.

```
                System Environments: Start Batch Node        2024/09/17 21:28:28









              Batch Node Startup Completed.









--------------------------------------------------------------------------------
Press Return to Continue
```

d. To create the BPE subsystem, choose option 3, **Applications & Subsystems**.

```
                Batch Administration Manager                2024/09/17 21:29:03




             1   Software License Management
             2   System Environments
             3   Applications & Subsystems
             4   Security & Users
             5   Classes & Activities
             6   Problem Determination



             H - Help
             Q - Quit

--------------------------------------------------------------------------------
Type an option and press Return: 3
```

e. Then choose option 3, **Create a Subsystem**.

```
                Applications & Subsystems                   2024/09/17 21:29:57




             1   List All Subsystems
             2   Query a Subsystem
             3   Create a Subsystem
             4   Update a Subsystem
             5   Delete a Subsystem
             6   Change/Show Default Subsystem
             7   Import a Subsystem (from another batch node)
             8   Create a BPESUB Project
             9   Export Subsystem bldsub Config File

             H - Help
             R - Return to Main Menu

--------------------------------------------------------------------------------
Type an option and press Return: 3
```

f. Enter the subsystem name as `sys1`.

```
                Applications & Subsystems: Create          2024/09/17 21:30:22









         No Subsystems are currently defined.









             R - Return to Previous Screen

--------------------------------------------------------------------------------
Enter the Subsystem's name you want to create: sys1
```

g. Choose option 3, **Data Management**.

```
                Applications & Subsystems: Create          2024/09/17 21:30:53



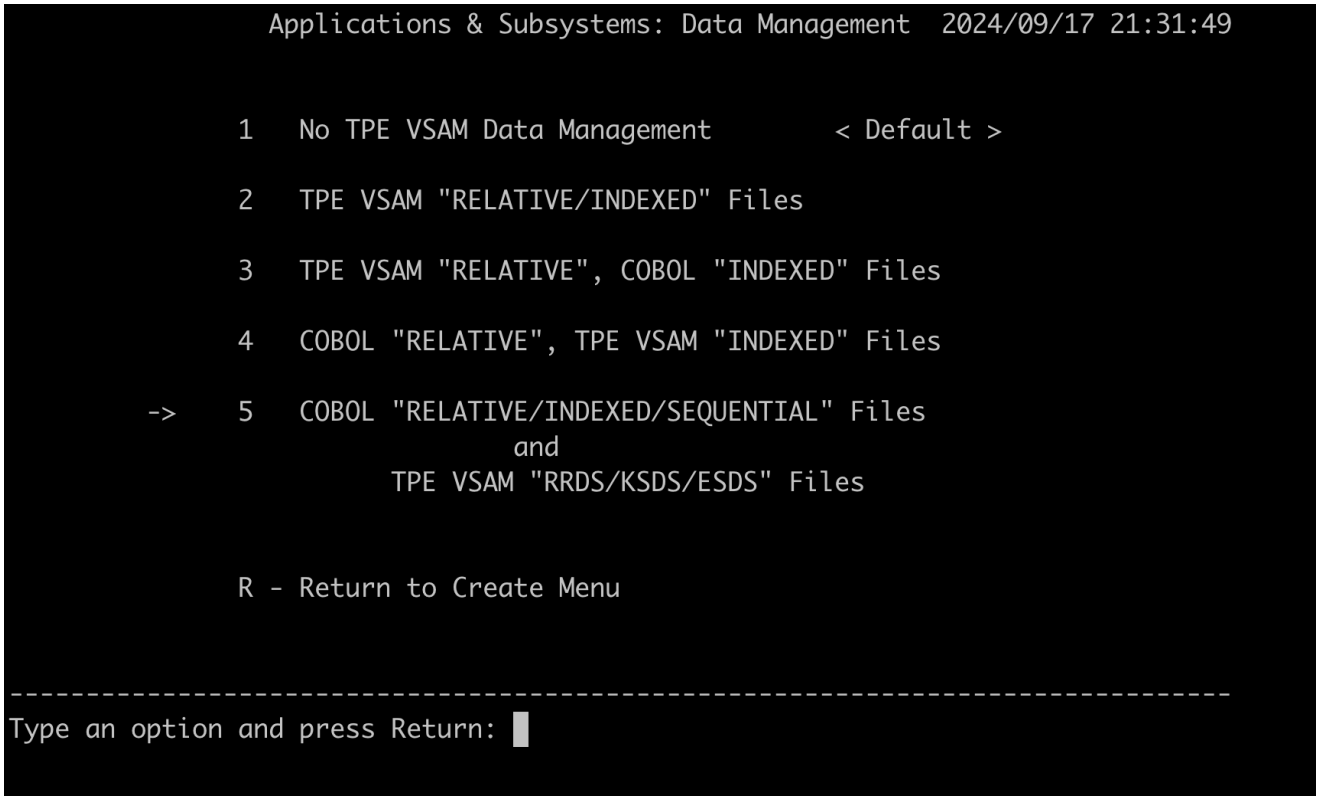
             D   Display Current sys1 Subsystem's Configuration
             S   Set to Default Subsystem Configuration

             1   Application Languages
             2   Database Management System (DBMS)
             3   Data Management
             4   Optional Packages
             5   Date/Time Management
             6   User-Specific Objects
             7   Configuration Options

             C - Create sys1 Subsystem
             H - Help
             R - Return to Main Menu

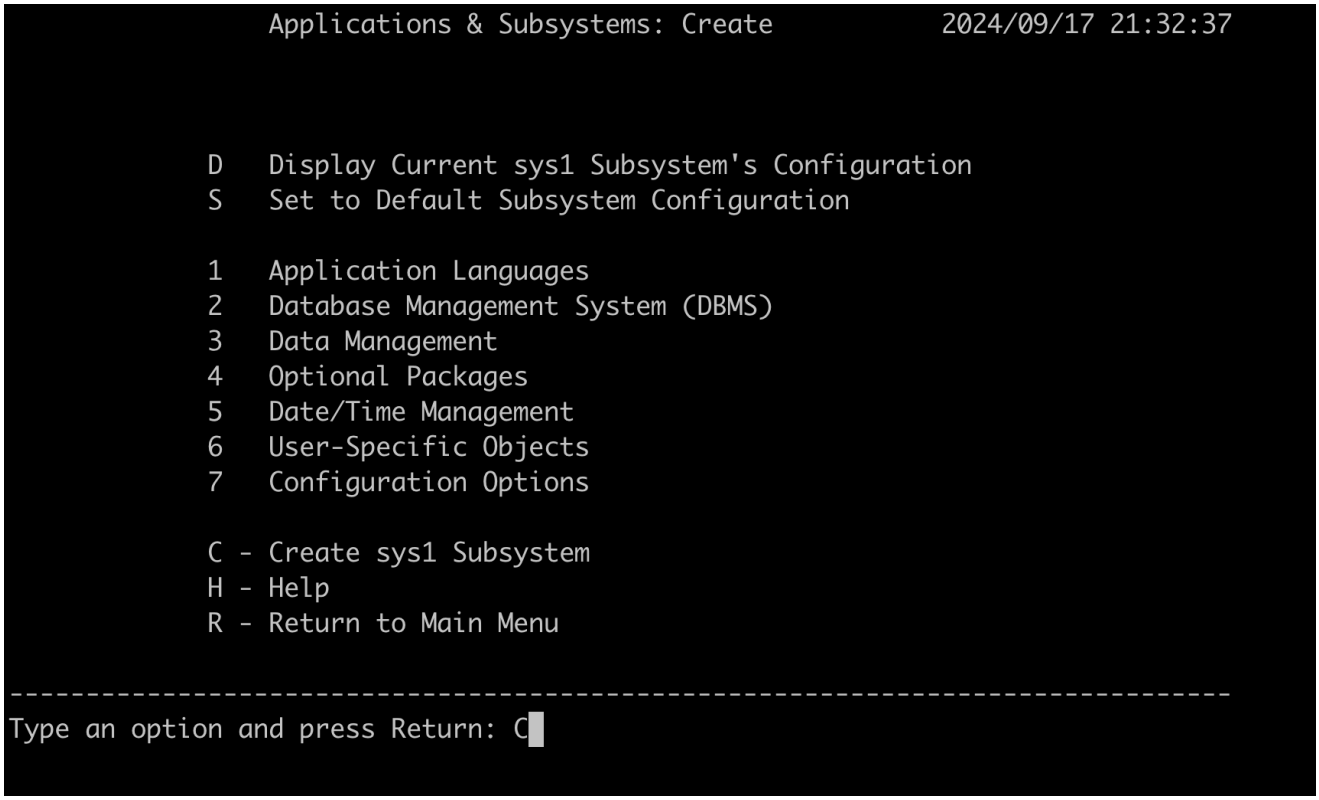
--------------------------------------------------------------------------------
Type an option and press Return: 3
```

h. Choose option 5, as the CardDemo application involves both sequential and VSAM files.



i. (Optional). Press "R" to return to the **Create Menu** page, review the different configuration options available.

j. On the **Create** page, enter "C" to create the subsystem `sys1`.



k. Review the settings, and enter "C" to continue for the rest of the environment settings. These environment settings are pre populated due to the necessary environment variables defined in the `carddemo.env` setup file and the recommended folder structure being in place.

l. Enter "y" to confirm and save the current environment settings.

```
                Applications & Subsystems: Create          2024/09/17 21:33:47









                sys1 Subsystem's environment setting completed









--------------------------------------------------------------------------------
Do you want to save current sys1's environment <y/n> ? : y
```

m. Enter "y" to display the log when building the subsystem.

```
                Applications & Subsystems: Create          2024/09/17 21:34:17









                Building sys1's NTT DATA COBOL runtime system



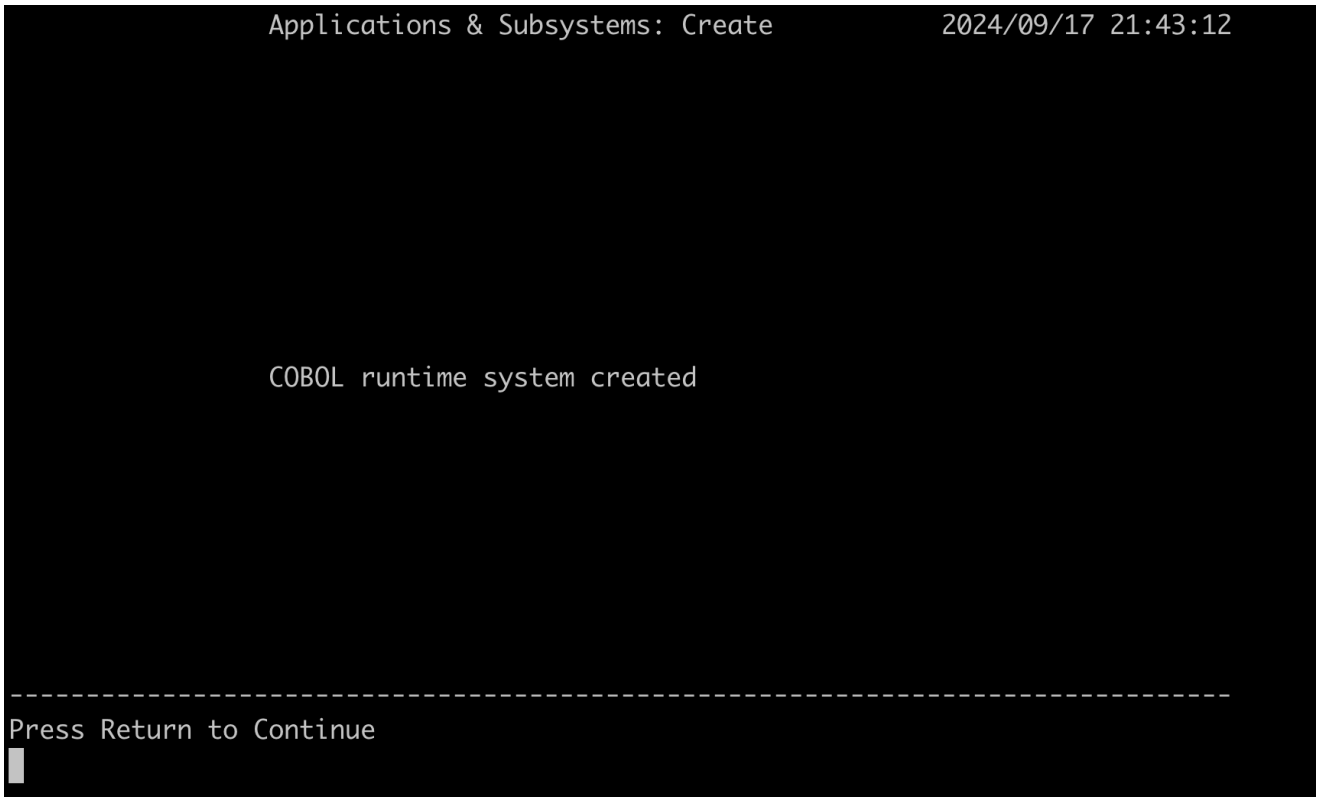


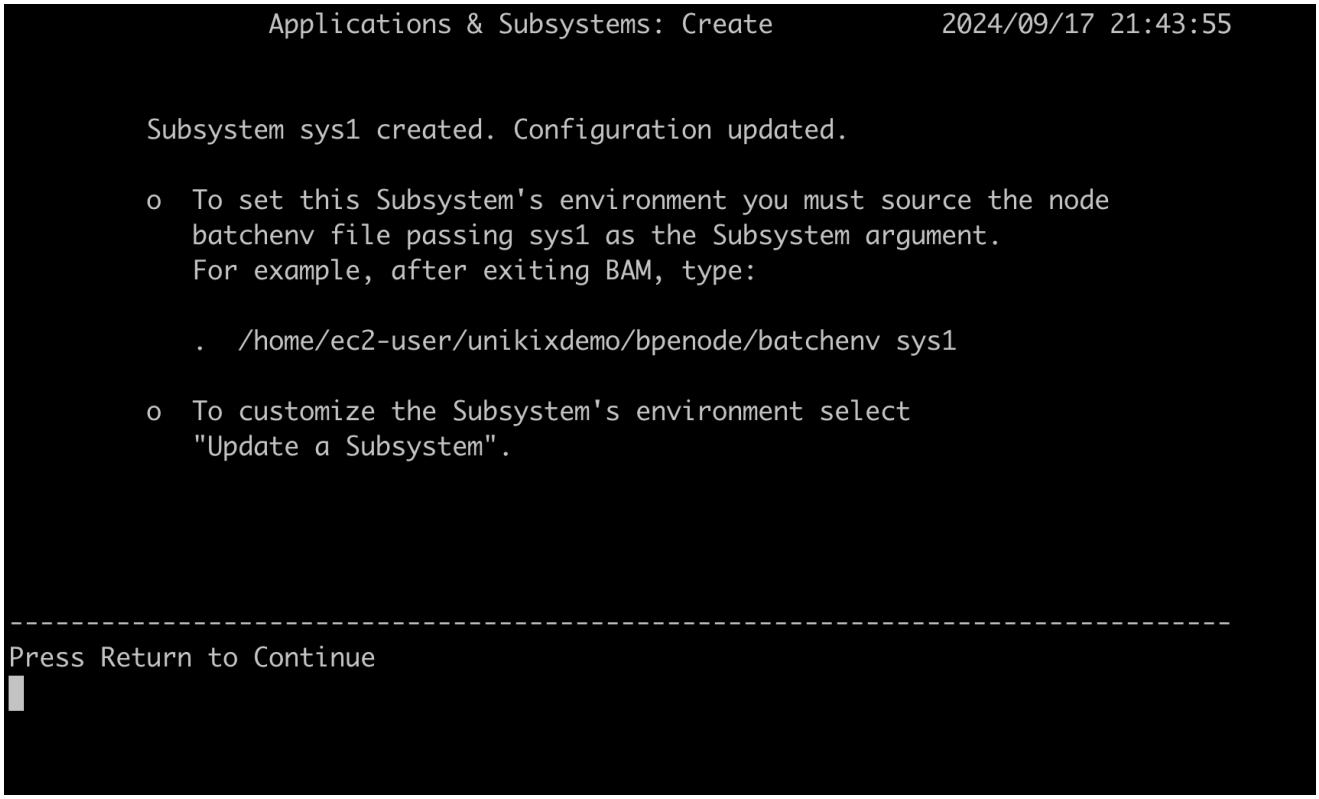


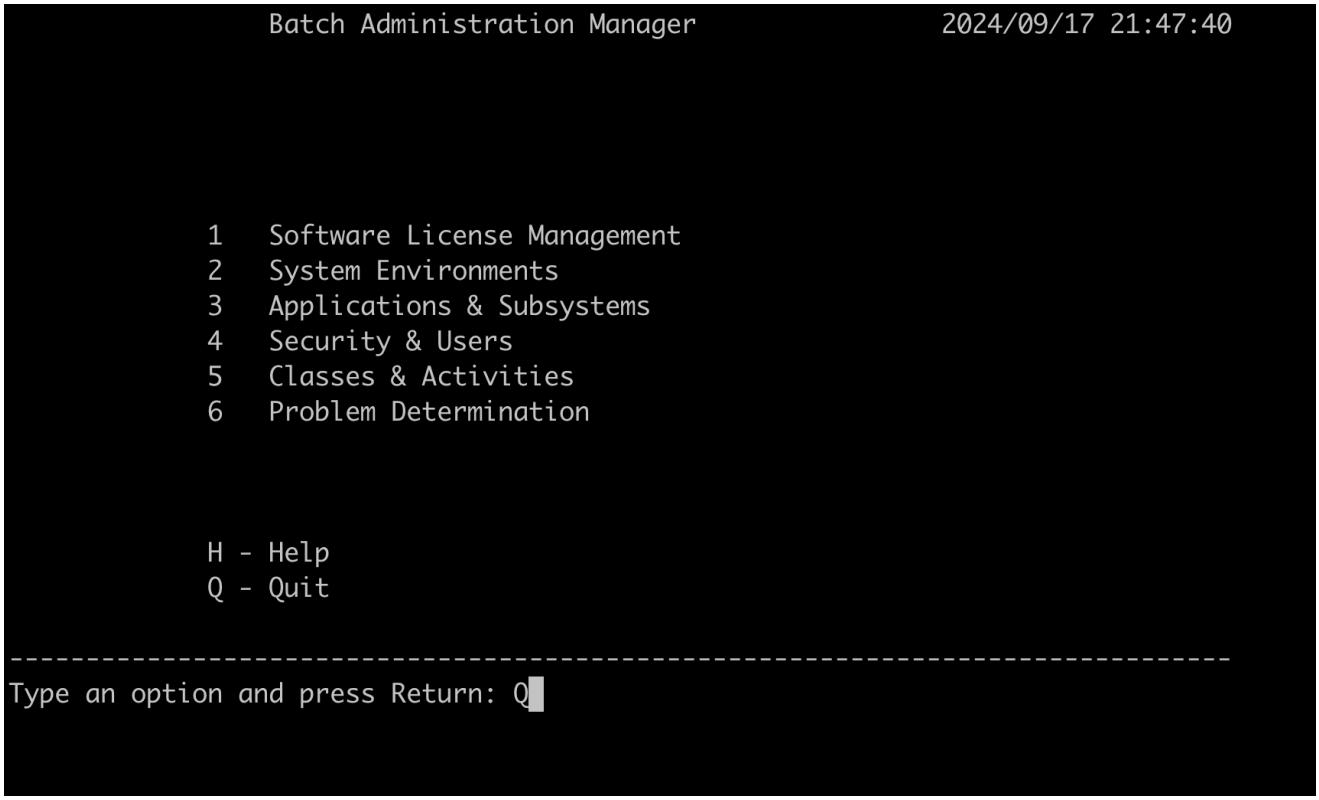


--------------------------------------------------------------------------------
Show log information while building the runtime system ?  <y/n> y
```

n. Press the **Return key** until you go back to the Main Menu and exit the BAM interface by selecting the **Quit** option.







5. Activate the BPE subsystem by sourcing the `batchenv` with the subsystem name `sys1`.

```
source $EBMHOME/batchenv sys1
```

Step 4: Compile and deploy CardDemo application

In this step, you compile the COBOL programs and deploy application artifacts such as JCL, procedures, data files, and CICS resource definitions.

1. Activate the CardDemo environment again by sourcing the `carddemo.env` setup file.

```
source $HOME/bin/carddemo.env
```

2. Navigate to the COBOL source directory.

```
cd $MIGAPP_DIR/cbl
```

3. Compile Cobol program `CBACT01C.cbl` using `compile` script.

```
compile CBACT01C.cbl
```

4. Compile all Cobol programs using `compile.all` script.

```
compile.all
```

5. Navigate to BMS maps source directory.

```
cd $MIGAPP_DIR/bms
```

6. Compile BMS map `COACTUP.bms` using `compbms` script.

```
compbms COACTUP.bms
```

7. Compile all BMS maps using `compbms.all` script.

```
compbms.all
```

8. Verify compiled binaries for COBOL and BMS maps.

```
ls $PROJECT_ROOT/loadlib
ls $PROJECT_ROOT/maps
```

9. Deploy other application artifacts such as JCL, procedures, data files, and CICS resource definitions using the `deploy_app` script.

```
deploy_app
```

10. Navigate to the project JCL directory.

```
cd $PROJECT_ROOT/jcl
```

11. Translate JCL ACCTFILE to BPE JCL Macro. Use the `mvstrans` command, utilizing the "-v" option for JCL verification, and the "-f" option to create the macro.

```
mvstrans ACCTFILE -v
mvstrans ACCTFILE -f
```

12. Translate JCL procedure REPROC to BPE JCL procedure Macro. Use the `mvstrans` command with the "-p" option along with the "-v" option for verification, and the "-f" option to create the macro.

```
mvstrans REPROC -v -p
```

```
mvstrans REPROC -f -p
```

13. Translate all JCLs and JCL procedures.

```
for file in "./jmvs/*"; do mvstrans $file -f; done > jmvs.out
for file in "./mvsp/*"; do mvstrans $file -p -f; done > mvsp.out
```

Step 5: Import BPE and TPE catalog

In this step, you import the BPE and TPE catalog using different commands.

1. Import BPE catalog using `loadcat` command.

```
loadcat $MIGAPP_DIR/catlg/bpe/BPECAT*
```

2. Navigate to $KIXSYS directory.

```
cd $KIXSYS
```

3. Import TPE catalog using `kiximpcat` command.

```
kiximpcat -c CATALOG -l CATALOG.lst
```

4. Import CICS resource definitions using kiximptbl command.

```
kiximptbl
```

Step 6: Start and connect TPE with BPE

In this step, you need to start the previously created TPE region along with the BPE manager and connect these to be able to run the sample CardDemo application.

1. Run the `kixverify` command against all VSAM files to ensure they are reset and any previously opened files are closed.

```
kixverify -r ALL
```

2. Start the TPE region.

```
kixregion start $KIXSYS
```

3. Ensure that both BPE and TPE are connected. This is crucial as the VSAM files are owned by TPE, and any batch operation accessing the VSAM will require a connection to TPE.

```
ebmsys -t
```



Step 7: Run the CardDemo application

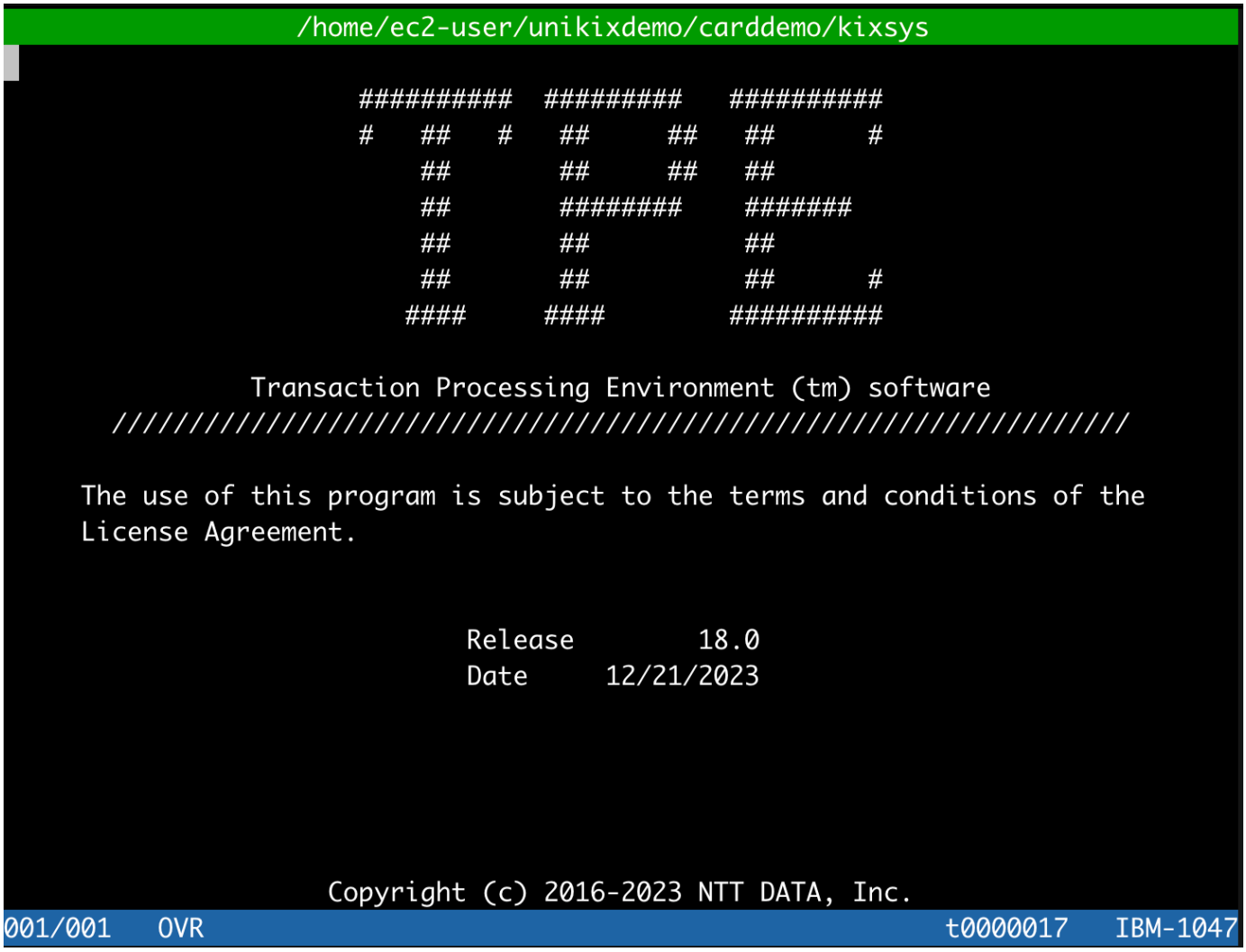
In this step, you run the CardDemo application in the TN3270 terminal emulator.

The UniKix runtime AMI comes with TN3270 terminal emulator that you can launch directly from the UniKix EC2 instance.

Connect to TPE using TN3270 terminal emulator

- Launch TN3270 terminal using the `kixterm` command.

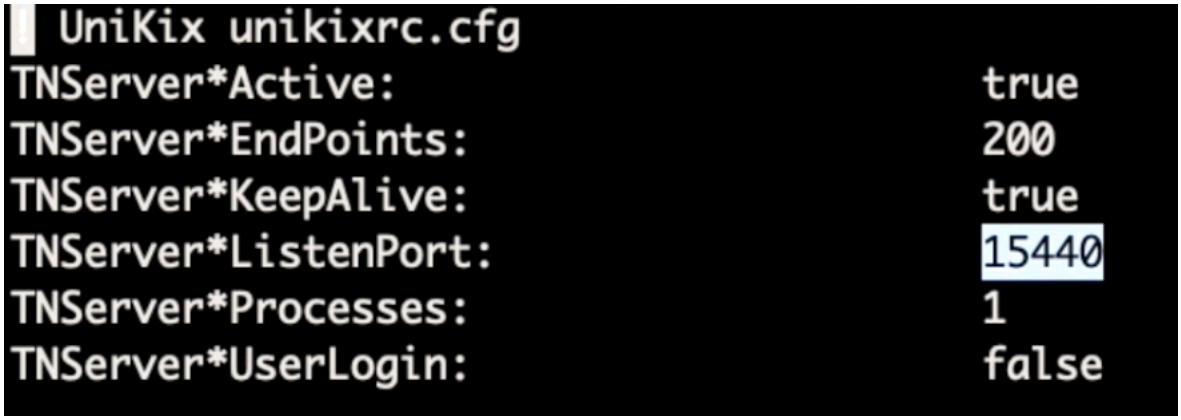
```
kixterm
```



(Optional). If you want to use your own terminal emulator:

1. Get IP address of UniKix runtime instance from the Amazon EC2 console.
2. Get the port number for connecting to the TPE region using the TN3270 terminal emulator. You can find this at TNServer ListenPort from unikixrc.cfg file.

```
cat $KIXSYS/unikixrc.cfg
```

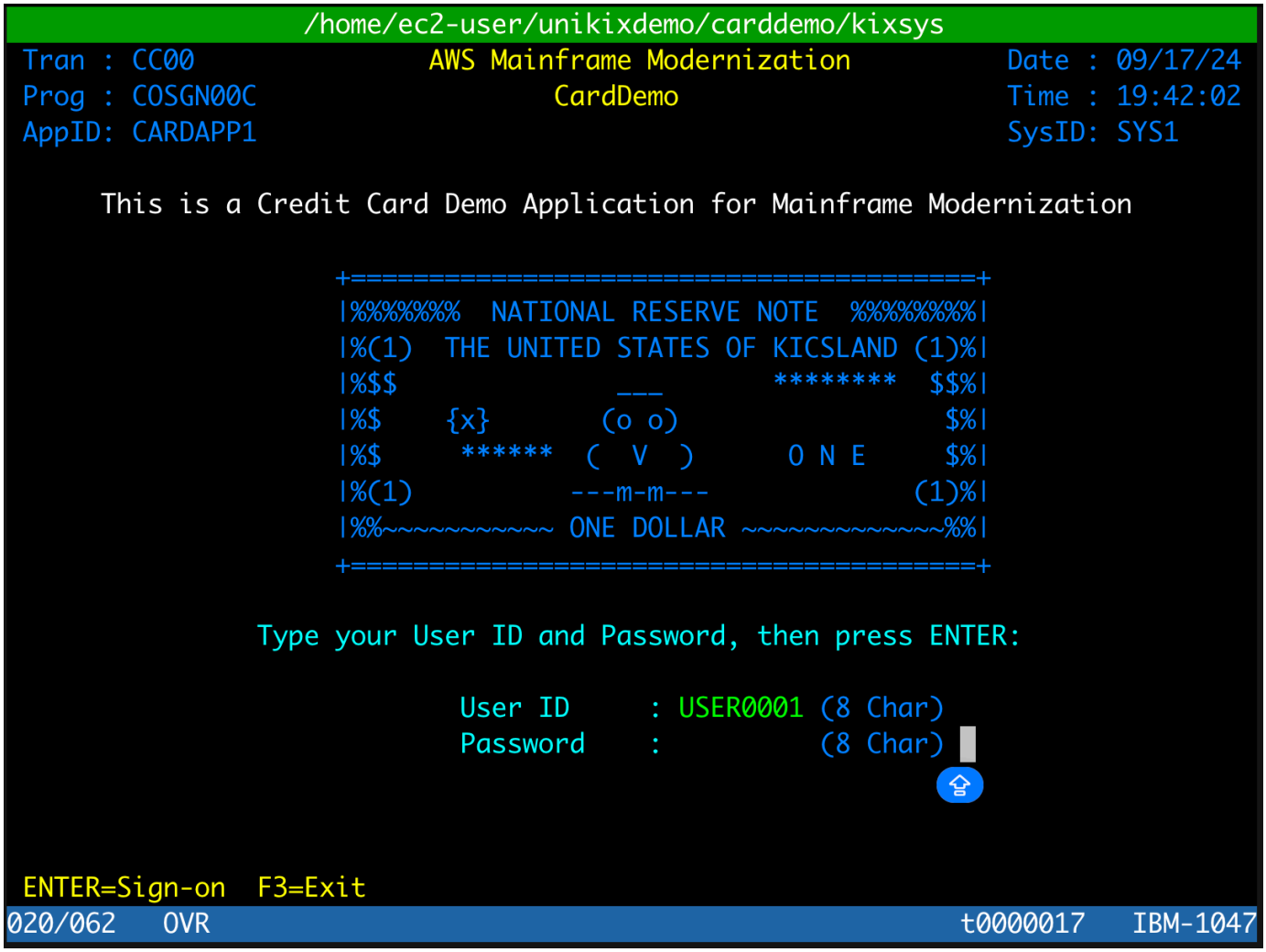


3. Configure your TN3270 terminal emulator to use IP address of UniKix runtime instance and port number 15440.

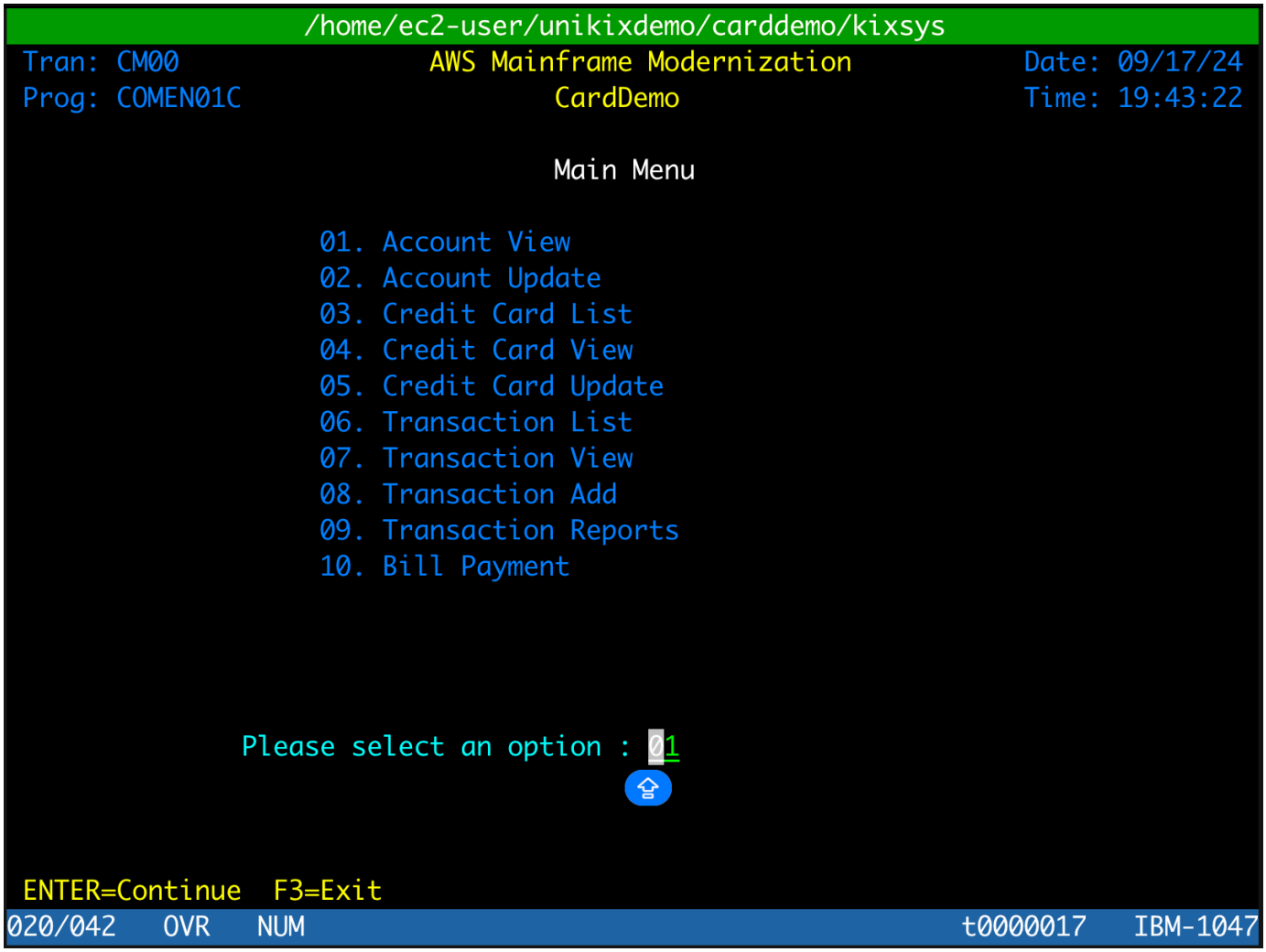
Online Transactions

This section assumes that you have connected to the TN3270 terminal emulator using the `kixterm` command.

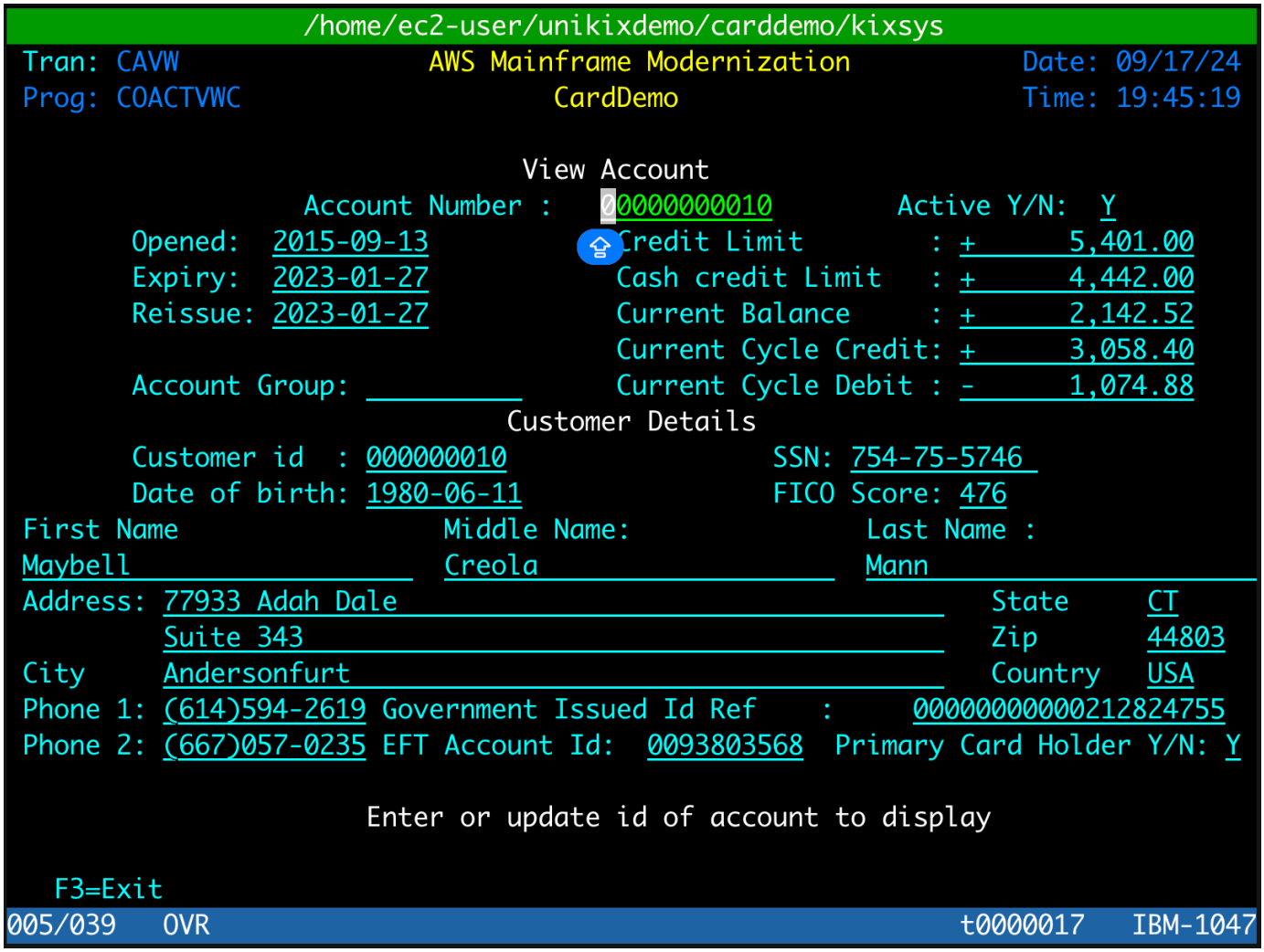
1. After connecting from TN3270 terminal emulator, press "Enter" key to clear the TPE screen and enter the initial transaction.
2. On the initial transaction CC00 (logon screen) enter `USER001` for username and `PASSWORD` for the password.



3. Choose option “01” from the **Main Menu** to view accounts.



4. In the **View Account** screen, enter an account number (e.g., 00000000010). You should see the account information populated from the migrated data.



5. Press “PF03” key twice to go back to Log in screen, and exit the TN3270 terminal by pressing "Ctrl+C" (Windows) or "Cmd+C" (Macbook).

Batch jobs

1. Navigate to the JCL directory.

```
cd $MBMSUB
```

2. Submit the job `MFCATGL1` and observe the job log output.

```
BPESUB READCARD
```

3. Optionally, you could view the job logs from the `$SUBSYS_OUTDIR` directory.

```
ls -lrt $SUBSYS_OUTDIR/*
```

You have now successfully deployed the CardDemo application onto NTT DATA UniKix runtime and verified the running application by navigating through a few CICS online screens and batch jobs.

Troubleshooting

Following are some common errors you may find when setting up the CardDemo application.

Error: Licensing error

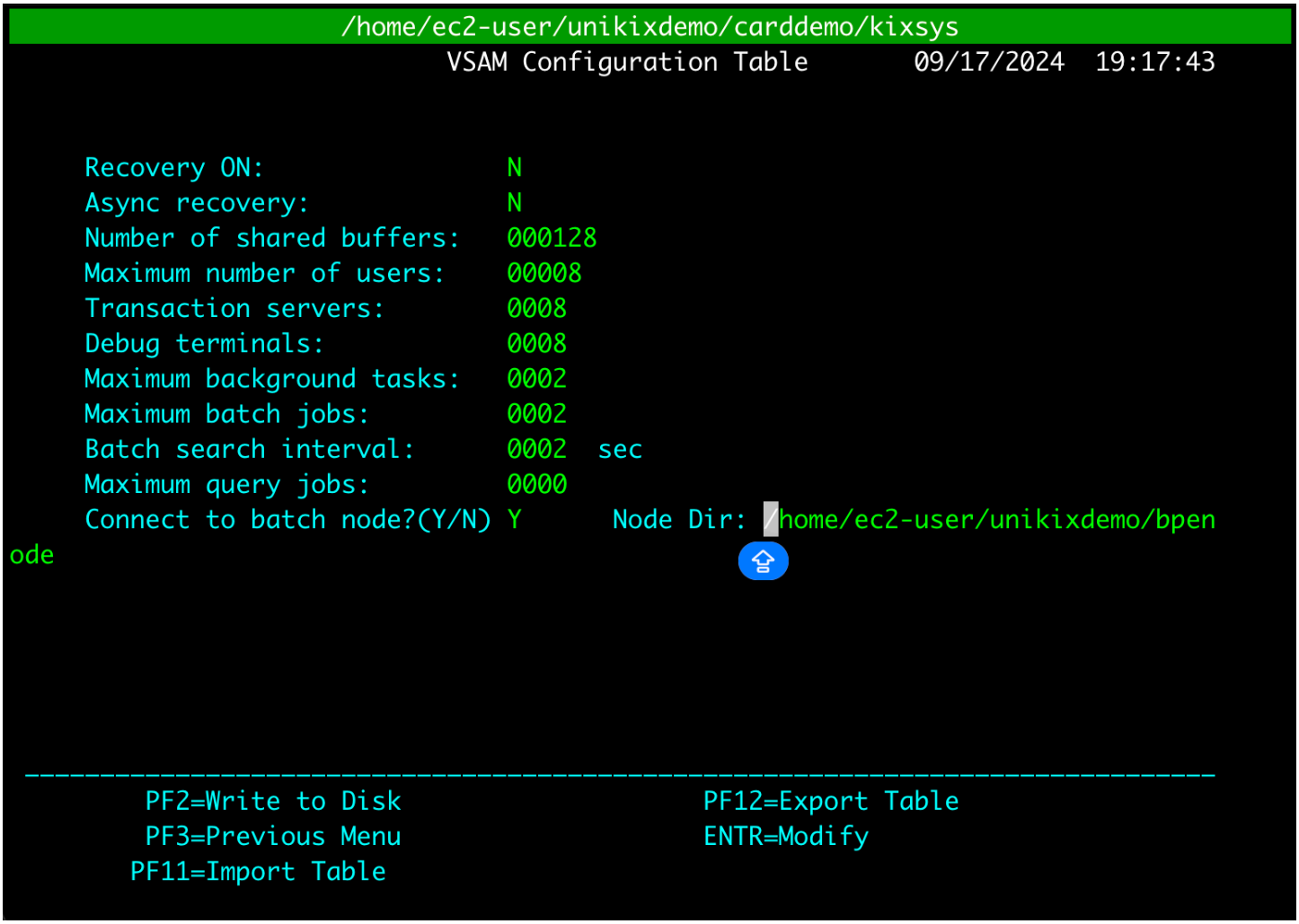
If you receive a license failure error during when following this tutorial, it could be that the **IMDSv2** is enabled in your EC2. You can resolve this issue by modifying the instance metadata option **IMDSv2** to **Optional** as mentioned in Restore the use of IMDSv1 in the *Amazon EC2 user guide*.

Error: TPE is not connected to BPE

If the TPE is not connected to BPE, make sure the **VSAM Configuration Table** is configured correctly with BPE Node directory. To access the VSAM Configuration Table, launch TN3270 terminal emulator using the following command:

```
kixterm
```

1. Enter transaction name as CTBL.
2. In the **Table Manager** menu, choose the option **Standard Tables**.
3. On the Standard tables screen, choose the option **VSAM Configuration Table**.
4. Check if **Connect to batch node?** is set to "**Y** and the Node Directory is correct.


/home/ec2-user/unikixdemo/carddemo/kixsys
VSAM Configuration Table 09/17/2024 19:17:43
Recovery ON: N
Async recovery: N
Number of shared buffers: 000128
Maximum number of users: 00008
Transaction servers: 0008
Debug terminals: 0008
Maximum background tasks: 0002
Maximum batch jobs: 0002
Batch search interval: 0002 sec
Maximum query jobs: 0000
Connect to batch node?(Y/N) Y Node Dir: /home/ec2-user/unikixdemo/bpen
ode
PF2=Write to Disk
PF3=Previous Menu
PF11=Import Table
PF12=Export Table
ENTR=Modify

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
- Identity and Access Management for AWS Mainframe Modernization

- Compliance validation for AWS Mainframe Modernization
- Resilience in AWS Mainframe Modernization
- Infrastructure security in AWS Mainframe Modernization
- Access AWS Mainframe Modernization using an AWS PrivateLink interface endpoint

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. You are also responsible for 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. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- 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 140-3 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) 140-3.

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.

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.

You can't disable this layer of encryption or select an alternate encryption type when you create runtime environments and applications.

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 .

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.

> **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.

Data type	AWS managed key encryption	Customer managed key encryption
`Definition` Contains the definition for a particular application.	Enabled	Enabled
`EnvironmentSummary` Contains information about the runtime environment.	Enabled	Enabled
`ApplicationSummary`	Enabled	Enabled

Data type	AWS managed key encryption	Customer managed key encryption
Contains information about the AWS Mainframe Modernization application.		
`DeploymentSummary` Contains information about a deployment of an AWS Mainframe Modernization application.	Enabled	Enabled

> **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 when 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 Rocket Software (formerly 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 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 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, 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.

```
{
    "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, see the *AWS Key Management Service Developer Guide*.

For more information about troubleshooting key access, see the *AWS Key Management Service Developer Guide*.

Specifying a customer managed key for AWS Mainframe Modernization

You can specify a customer managed key for the following resources:

- Application
- 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 add your customer managed key when you create a runtime environment with the AWS CLI, specify the `kms-key-id` parameter, as follows:

```
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:

```
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 related to an application (create application and create deployment), where the key is `aws:m2:app` and the value is the unique identifier of the application.

Example

```
"encryptionContextSubset": {
        "aws:m2:app": "a1bc2defabc3defabc4defabcd"
}
```

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 is a grant that AWS Mainframe Modernization leverages to encrypt application artifact when creating an application.

```
//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:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
```

```
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T19:40:26Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-12-06T20:23:43Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.182",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "keyId": "00dd0db0-0000-0000-ac00-b0c000SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
        "clientProvidedHostHeader": "kms.us-west-2.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
```

```
}
```

CreateGrant

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.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:11:45Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
```

```
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "operations": [
            "Encrypt",
            "Decrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "GenerateDataKey",
            "GenerateDataKey",
            "DescribeKey",
            "CreateGrant"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


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.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65MZFUPM4JO:EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfxDj5zz86tr",
        "arn": "arn:aws:sts::111122223333:assumed-role/EnvironmentWorkflow-
alpha-CreateEnvironmentLambdaS-1AU4A8VNQEEKN/EnvironmentWorkflow-alpha-
CreateEnvironmentLambda7-HfxDj5zz86tr",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4A8VNQEEKN",
                "accountId": "111122223333",
                "userName": "EnvironmentWorkflow-alpha-
CreateEnvironmentLambdaS-1AU4A8VNQEEKN"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:22:28Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-12-06T20:23:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "54.148.236.160",
    "userAgent": "aws-sdk-java/2.18.21 Linux/4.14.255-276-224.499.amzn2.x86_64
 OpenJDK_64-Bit_Server_VM/11.0.14.1+10-LTS Java/11.0.14.1 vendor/Amazon.com_Inc. md/
internal exec-env/AWS_Lambda_java11 io/sync http/Apache cfg/retry-mode/legacy",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "operations": [
            "Encrypt",
            "Decrypt",
            "ReEncryptFrom",
```

```
            "ReEncryptTo",
            "GenerateDataKey",
            "GenerateDataKey",
            "DescribeKey",
            "CreateGrant"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
        "clientProvidedHostHeader": "kms.us-west-2.amazonaws.com"
    }
}
}
```


GenerateDataKey

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:


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65EEXVIEH7D:AutoScaling",
        "arn": "arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForAutoScaling/
AutoScaling",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAutoScaling"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T20:23:16Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "autoscaling.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:18Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "autoscaling.amazonaws.com",
    "userAgent": "autoscaling.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:ebs:id": "vol-080f7a32d290807f3"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "numberOfBytes": 64
```

```
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

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:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ebs.amazonaws.com"
    },
    "eventTime": "2022-12-06T20:23:22Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ebs.amazonaws.com",
    "userAgent": "ebs.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "aws:ebs:id": "vol-080f7a32d290807f3"
        }
    },
```

```
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventCategory": "Management"
}
```

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:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
```

```
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T21:51:45Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T22:47:04Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "constraints": {
            "encryptionContextSubset": {
                "aws:m2:app": "a1bc2defabc3defabc4defabcd"
            }
        },
        "retiringPrincipal": "m2.us-west-2.amazonaws.com",
        "operations": [
            "GenerateDataKey"
        ],
        "granteePrincipal": "m2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
```

```
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateDataKey

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:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65CLCEKKC7Z:ApplicationWorkflow-alpha-
CreateApplicationVersion-CstWZUn5R4u6",
        "arn": "arn:aws:sts::111122223333:assumed-role/ApplicationWorkflow-
alpha-CreateApplicationVersion-1IZRBZYDG20B/ApplicationWorkflow-alpha-
CreateApplicationVersion-CstWZUn5R4u6",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/ApplicationWorkflow-alpha-
CreateApplicationVersion-1IZRBZYDG20B",
```

```
                "accountId": "111122223333",
                "userName": "ApplicationWorkflow-alpha-
CreateApplicationVersion-1IZRBZYDG20B"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T23:28:32Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:29:08Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:m2:app": "a1bc2defabc3defabc4defabcd",
            "aws:s3:arn": "arn:aws:s3:::supernova-processedtemplate-111122223333-us-
west-2/111122223333/a1bc2defabc3defabc4defabcd/1/cics-transaction/ZBNKE35.so"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
```

```
}
```

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:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T21:51:45Z",
                "mfaAuthenticated": "false"
            }
        },
```

```
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:40:07Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:m2:app": "a1bc2defabc3defabc4defabcd"
            }
        },
        "granteePrincipal": "m2.us-west-2.amazonaws.com",
        "retiringPrincipal": "m2.us-west-2.amazonaws.com",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
```

```
}
```

Decrypt

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:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3YPCLM65BSPZ37E6G:m2-hm-bqe367dxtfcpdbzmnhfzranisu",
        "arn": "arn:aws:sts::111122223333:assumed-role/
SupernovaEnvironmentInstanceScopeDownRole/m2-hm-bqe367dxtfcpdbzmnhfzranisu",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:iam::111122223333:role/
SupernovaEnvironmentInstanceScopeDownRole",
                "accountId": "111122223333",
                "userName": "SupernovaEnvironmentInstanceScopeDownRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-12-06T23:19:29Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "m2.us-west-2.amazonaws.com"
    },
    "eventTime": "2022-12-06T23:40:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "m2.us-west-2.amazonaws.com",
    "userAgent": "m2.us-west-2.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
```

```
            "aws:m2:app": "a1bc2defabc3defabc4defabcdm",
            "aws:s3:arn": "arn:aws:s3:::supernova-processedtemplate-111122223333-us-
west-2/111122223333/a1bc2defabc3defabc4defabcdm/1/cics-transaction/BBANK40P.so"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


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
- Authenticating with identities
- Managing access using policies
- How AWS Mainframe Modernization works with IAM
- Identity-based policy examples for AWS Mainframe Modernization
- Troubleshooting AWS Mainframe Modernization identity and access
- Using service-linked roles for AWS Mainframe Modernization

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.

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.

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.

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 AWS Signature Version 4 for 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 AWS Multi-factor authentication in IAM 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 Use cases for IAM users 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. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). 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 Methods to assume a role 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 Create a role for a third-party identity provider (federation) 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 Cross account resource access in IAM 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.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
- **Service role** – 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 Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – 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.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

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 Define custom IAM permissions with customer managed 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 Choose 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 Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) 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

IAM feature	AWS Mainframe Modernization support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Forward access sessions (FAS)	Yes
Service roles	Yes
Service-linked roles	Yes

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 Define custom IAM permissions with customer managed 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.

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 Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS Mainframe Modernization

Supports policy actions: Yes

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.

Policy resources for AWS Mainframe Modernization

Supports policy resources: Yes

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). 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) in the *AWS General Reference*.

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 in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, 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.

AWS Mainframe Modernization API permissions: Actions, resources, and conditions reference

When you are writing permissions policies that you can attach to an IAM identity (identity-based policies), you can use the following table as a reference. The table includes the following:

- 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

AWS Mainframe Modernization API Operations	Required Permissions (API Actions)	Resources
CancelBatchJobExecution		Application
CreateApplication	iam:PassRole kms:DescribeKey kms:CreateGrant s3:GetObject s3:ListBucket	Application
CreateDataSetImportTask	s3:GetObject	Application
CreateDataSetExportTask	kms:DescribeKey s3:PutObject	Application
CreateDeployment	elasticloadbalancing:AddTags elasticloadbalancing:CreateListener elasticloadbalancing:CreateTargetGroup elasticloadbalancing:RegisterTargets elasticloadbalancing:DeleteListener	Application

AWS Mainframe Modernization API Operations	Required Permissions (API Actions)	Resources
	elasticloadbalancing:DeleteTargetGroup elasticloadbalancing:DeregisterTargets elasticloadbalancing:DeleteLoadBalancer logs:CreateLogDelivery logs:GetLogDelivery logs:UpdateLogDelivery logs:DeleteLogDelivery logs:ListLogDeliveries logs:PutResourcePolicy logs:DescribeResourcePolicies logs:DescribeLogGroups	

AWS Mainframe Modernization API Operations	Required Permissions (API Actions)	Resources
CreateEnvironment	ec2:CreateNetworkInterface ec2:CreateNetworkInterfacePermission ec2:DescribeNetworkInterfaces ec2:DescribeSecurityGroups ec2:DescribeSubnets ec2:DescribeVpcAttribute ec2:DescribeVpcs ec2:ModifyNetworkInterfaceAttribute elasticfilesystem:DescribeMountTargets elasticloadbalancing:AddTags elasticloadbalancing:CreateLoadBalancer elasticloadbalancing:DeleteLoadBalancer kms:DescribeKey kms:CreateGrant fsx:DescribeFileSystems	Environment

AWS Mainframe Modernization API Operations	Required Permissions (API Actions)	Resources
	`iam:CreateServiceLinkedRole`	

DeleteApplication	`elasticloadbalancing:DeleteListener`
	`elasticloadbalancing:DeleteTargetGroup`
	`logs:DeleteLogDelivery`
		Application

DeleteApplicationFromEnvironment	`elasticloadbalancing:DeleteListener` `elasticloadbalancing:DeleteTargetGroup`	Application Environment
DeleteEnvironment	`elasticloadbalancing:DeleteLoadBalancer`	Environment
GetApplication		Application
GetApplicationVersion		Application
GetBatchJobExecution		Application
GetDataSetDetails		Application
GetDataSetImportTask		Application
GetDataSetExportTask		Application
GetDeployment		Application
GetEnvironment		Environment

AWS Mainframe Modernization API Operations	Required Permissions (API Actions)	Resources
ListApplications		*
ListApplicationVersions		*
ListBatchJobDefinitions		*
ListBatchJobExecutions		*
ListDataSetImportHistory		*
ListDataSetExportHistory		*
ListDataSets		*
ListDeployments		*
ListEngineVersions		*
ListEnvironments		*
ListTagsForResource		*
StartApplication		Application
StartBatchJob		Application
StopApplication		Application
TagResource		*
UntagResource		*

AWS Mainframe Modernization API Operations	Required Permissions (API Actions)	Resources
UpdateApplication	s3:GetObject s3:ListBucket	Application
UpdateEnvironment	kms:DescribeKey	Environment

Policy condition keys for AWS Mainframe Modernization

Supports service-specific policy condition keys: Yes

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.

Access control lists (ACLs) in AWS Mainframe Modernization

Supports ACLs: No

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 Define permissions with ABAC authorization 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 Switch from a user to an IAM 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.

Forward access sessions for AWS Mainframe Modernization

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

> **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

Supports service roles: Yes

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 Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

AWS Mainframe Modernization supports service roles for activity hooks (transaction / jobs abend 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 in the *Amazon EC2 Auto Scaling User Guide*.

Service-linked roles for AWS Mainframe Modernization

Supports service-linked roles: Yes

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 AWS Mainframe Modernization.

For details about creating or managing service-linked roles, see AWS services that work with IAM. 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 Create IAM policies (console) 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
- Using the AWS Mainframe Modernization console
- Allow users to view their own permissions

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 Validate policies with IAM Access Analyzer 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 Secure API access with MFA 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.

```
{
    "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.

Topics

- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS Mainframe Modernization resources

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.
- 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 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 in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Using service-linked roles for AWS 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 AWS Mainframe Modernization. Service-linked roles are predefined by AWS 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 AWS Mainframe Modernization easier because you don't have to manually add the necessary permissions. AWS Mainframe Modernization defines the permissions of its service-linked roles, and unless defined otherwise, only AWS 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 AWS 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 AWS Mainframe Modernization

AWS 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 AWS Mainframe Modernization to complete the following actions on the specified resources:

- Create, delete, describe, and attach permissions to Amazon EC2 network interfaces for the AWS 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 AWS 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.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeSubnets",
    "ec2:CreateNetworkInterface",
    "ec2:DeleteNetworkInterface",
    "ec2:DescribeNetworkInterfaces",
    "ec2:CreateNetworkInterfacePermission",
    "ec2:ModifyNetworkInterfaceAttribute"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "elasticfilesystem:DescribeMountTargets"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "elasticloadbalancing:RegisterTargets",
```

```
     "elasticloadbalancing:DeregisterTargets"
    ],
    "Resource": "*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "fsx:DescribeFileSystems"
    ],
    "Resource": "*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "cloudwatch:PutMetricData"
    ],
    "Resource": "*",
    "Condition": {
     "StringEquals": {
      "cloudwatch:namespace": [
       "AWS/M2"
      ]
     }
    }
   }
  ]
}
```

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 AWS 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, AWS 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, AWS Mainframe Modernization creates the service-linked role for you again.

Editing a service-linked role for AWS Mainframe Modernization

AWS 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 using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for AWS 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 AWS 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 AWS Mainframe Modernization resources used by the AWSServiceRoleForAWSM2

- Delete the runtime environments in AWS 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 AWS Mainframe Modernization service-linked roles

AWS 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 AWS 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.

Access AWS Mainframe Modernization using an AWS PrivateLink interface endpoint

You can use AWS PrivateLink to create a private connection between your VPC and AWS Mainframe Modernization. You can access AWS 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 AWS 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 AWS Mainframe Modernization.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for AWS Mainframe Modernization

Before you set up an interface endpoint for AWS Mainframe Modernization, review Considerations in the *AWS PrivateLink Guide*.

AWS Mainframe Modernization supports making calls to all of its API actions through the interface endpoint.

Create an interface endpoint for AWS Mainframe Modernization

You can create an interface endpoint for AWS 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 AWS 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 AWS 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 AWS Mainframe Modernization through the interface endpoint. To control the access allowed to AWS 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 AWS 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 AWS Mainframe Modernization actions for all principals on all resources.

```
//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.
- *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.
- *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.

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.

The following tables list the metrics and dimensions for AWS Mainframe Modernization. The namespace for these metrics is `AWS/M2`.

Runtime Environment Metrics

Metric	Description
CPUUtilization	The CPU utilization of instances in the environment. Dimension: environmentId Units: Percent Valid statistics: Average, Minimum, Maximum
InboundNetworkThroughput	Inbound network throughput of instances in the environment. Dimension: environmentId Units: Bytes per second Valid statistics: Average, Minimum, Maximum
MemoryUtilization	The memory utilization of instances in the environment. Dimension: environmentId Units: Percent Valid statistics: Average, Minimum, Maximum
OutboundNetworkThroughput	Outbound network throughput of the instances in the environment. Dimension: environmentId Units: Bytes per second Valid statistics: Average, Minimum, Maximum

Application Metrics

Metric	Description
BatchJobCompletedCount	The number of completed jobs during the time interval. This metric is available for Rocket Software (formerly Micro Focus) and for AWS Blu Age 3.7.0 and later releases. Dimension: applicationId Units: Count Valid statistics: Sum
BatchJobFailedCount	The number of failed jobs during the time interval. This metric is available for Rocket Software and for AWS Blu Age 3.7.0 and later releases. Dimension: applicationId Units: Count Valid statistics: Sum
JvmMemoryFree	The amount of available memory that is not currently in use by the Java Virtual Machine. 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. Dimension: applicationId Units: Bytes Valid statistics: Average, Minimum, Maximum

Metric	Description
JvmMemoryMax	The maximum amount of memory allowed for the Java Virtual Machine. 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. Dimension: applicationId Units: Bytes Valid statistics: Average, Minimum, Maximum
JvmMemoryUsed	The amount of memory actively used by the Java Virtual Machine. 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. Dimension: applicationId Units: Bytes Valid statistics: Average, Minimum, Maximum
ProcessesActiveCount	The active number of concurrent service execution processes that are processing requests. This metric is only available for the Rocket Software runtime engine. Dimension: applicationId Units: Count Valid statistics: Sum

Metric	Description
SessionCount	The number of HTTP sessions for the application. 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. Dimension: applicationId Units: Count Valid statistics: Average, Minimum, Maximum
SharedMemoryFree	The memory that is available for the enterprise server to store all the information it needs to run transactions and jobs. This metric is only available for the Rocket Software runtime engine. Dimension: applicationId Units: Kilobytes Valid statistics: Average, Minimum, Maximum
SharedMemoryTotal	The total amount of shared memory allocated for the enterprise server to store all the information it needs to run transactions and jobs. This metric is only available for the Rocket Software runtime engine. Dimension: applicationId Units: Kilobytes Valid statistics: Average, Minimum, Maximum

Metric	Description
ThreadActiveCount	The number of engine threads that are processing requests. 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. Dimension: applicationId Units: Count Valid statistics: Average, Minimum, Maximum
TransactionCompletedCount	The number of committed transactions during the time interval. This metric is available for Rocket Software and for AWS Blu Age 3.7.0 and later releases. Dimension: applicationId Units: Count Valid statistics: Sum
TransactionFailedCount	The number of failed transactions during the time interval. This metric is available for Rocket Software and for AWS Blu Age 3.7.0 and later releases. Dimension: applicationId Units: Count Valid statistics: Sum

Metric	Description
TransactionResponseTime	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. This metric is available for Rocket Software and for AWS Blu Age 3.7.0 and later releases. Dimension: applicationId Units: Milliseconds Valid statistics: Average, Minimum, Maximum

Dimensions

Dimension	Description
applicationId	This dimension filters the metric to the identified application by ID.
environmentId	This dimension filters the metric to the identified environment by ID.

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.

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 aren't 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.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAII6WZTHGYAEXAMPLE",
        "arn": "arn:aws:sts::444455556666:assumed-role/Admin/Mary_Major",
        "accountId": "444455556666",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAII6WZTHGYAEXAMPLE",
                "arn": "arn:aws:iam::444455556666:role/Admin",
                "accountId": "444455556666",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-06-01T20:38:22Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-06-01T20:40:39Z",
    "eventSource": "m2.amazonaws.com",
    "eventName": "CreateApplication",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.196.65",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:91.0) Gecko/20100101
 Firefox/91.0",
```

```
    "requestParameters": {
        "clientToken": "1abc23de-f45g-6789-h01i-jkl2m3456789",
        "name": "MyApp",
        "description": "",
        "engineType": "microfocus",
        "definition": {
            "content": "{}"
        },
        "tags": {}
    },
    "responseElements": {
        "applicationVersion": 1,
        "Access-Control-Expose-Headers": "x-amzn-RequestId,x-amzn-ErrorType,x-amzn-
ErrorMessage,Date",
        "applicationArn": "arn:aws:m2:us-east-1:444455556666:app/
lsfhmwhw7fffrosff2lncwqcua",
        "applicationId": "lsfhmwhw7fffrosff2lncwqcua"
    },
    "requestID": "36982d38-fcde-4bfe-a89a-7bd78d43c926",
    "eventID": "d7f0fc36-46ae-4157-9a79-c79f385fda98",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "444455556666",
    "eventCategory": "Management"
}
```

Troubleshooting in AWS Mainframe Modernization

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 Rocket Software engines.

Topics

- Troubleshooting error: Time out while waiting for data set name to be unlocked
- Troubleshooting error: Cannot access an application URL
- Troubleshooting: AWS Blu Insights does not open from the console
- Troubleshooting error: Environment unhealthy
- Troubleshooting license issues for Rocket Software (formerly Micro Focus)

Troubleshooting error: Time out while waiting for data set name to be unlocked

This page describes how you can resolve your error when you see another application in an environment is holding a lock on a shared data set.

- 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 data set. Typically, this situation occurs if the other application crashes or otherwise fails and does not release the lock.

Look for a failed application and check whether it uses the same data set 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.

Common cause

Application `example-app-1` tries to lock a record `example-record-1` for a write operation. This operation creates both a lock on data set `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 data set 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`.

Resolution

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 (Redis OSS) 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:

Command	Purpose
KEYS *	Get all existing keys.
KEYS *YOUR_DATASET_NAME	Get a dataset lock key.

Command	Purpose
DEL *THE_RETURNED_KEY*	Delete a dataset lock.
FLUSHDB	Clean the entire Redis. **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 data sets 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.

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 data set 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.

Troubleshooting error: Cannot access an application URL

This page describes how you can resolve your error when you can't access URL for a running AWS Mainframe Modernization application.

- Engine: AWS Blu Age and Rocket Software (formerly Micro Focus)

- Component: applications

If you can't 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.

Common cause

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.

Resolution

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:

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. In the left navigation, choose **Environments**.
3. Choose the runtime environment that hosts the application you want to connect to.
4. Choose **Configurations**.
5. In **Security & Network**, choose the security group. The link opens the details of the security group in the Amazon VPC console.
6. If necessary, choose **Edit inbound rules** and add the following rule if not already present:

   Type

   Custom TCP

   Port

   8196 or the port that matches the listener properties specified in the application definition. For more information, see Step 2: Create the application definition.

   Source

   The IP address from where you are calling the application. You can choose **myIP** from the dropdown. If you still have timeout issues, try choosing **Anywhere IPV4** or **Anywhere IPV6**.

Make sure to stop the application and start it again after you add the inbound rule on the security group.

For more information, see Work with security group rules in *Amazon VPC User Guide*.

Troubleshooting: AWS Blu Insights does not open from the console

This page describes how you can resolve Blu Insights page not opening from the AWS Mainframe Modernization console.

- Engine: AWS Blu Age
- Component: Blu Insights

When you try to access Blu Insights from the AWS Mainframe Modernization console, it doesn't open and the new tab is closed immediately.

Common cause

The role you are using to access Blu Insights does not have sufficient permissions.

Resolution

Attach an IAM policy to the role to allow it to access Blu Insights. Make sure the policy includes at least the following permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "m2:GetSignedBluinsightsUrl"
            ],
            "Resource": "*"
            }
    ]
}
```

Make sure to replace `region` and `account` with the correct AWS Region and AWS account.

Troubleshooting error: Environment unhealthy

This page describes how you can resolve your error when you receive a notification that one of your AWS Mainframe Modernization environments are unhealthy.

- Engine: AWS Blu Age and Rocket Software (formerly Micro Focus)
- Component: environments

If you receive a notification that says one of your AWS Mainframe Modernization environments has become unhealthy, this applies to you. You are notified through one of these sources:

- The unhealthy environment status is shown in your AWS Mainframe Modernization console.
- Email notification about the unhealthy environment status from AWS Health.
- You see a related event from AWS Mainframe Modernization in your AWS Health dashboard, under **Your account health**.

Common cause

The error occurs when the resources in your AWS account associated with the AWS Mainframe Modernization environment is inaccessible. A common reason for this issue is that the resources related to the environment are being modified or deleted.

Resolution

For specific guidance, use the error code provided in the email from **AWS Health**, or through your **AWS Mainframe Modernization console**.

Error code:

- Storage unreachable

This error indicates that the attached storage (Amazon Elastic File System or Amazon FSx file systems) for the environment has failed to mount correctly. To check details about unhealthy environment, complete the following steps:

1. Open the AWS Mainframe Modernization console at https://console.aws.amazon.com/m2/.
2. Select the unhealthy environment, and choose **Configuration**.
3. Choose **Attached Storage** to view the storage resources associated with this environment.
4. Check the network-related configurations, such as the security group, subnet, and Amazon VPC associated with the storage. If these configurations are incorrect, try to restore them to solve this issue.

> **Note**
>
> If the storage has been deleted, the environment can't be recovered. In this case, you should consider deleting the unhealthy environment.

Troubleshooting license issues for Rocket Software (formerly Micro Focus)

This page describes how you can resolve license issues with the Rocket Software Runtime engine

- Engine: Rocket Software
- Component: Amazon EC2

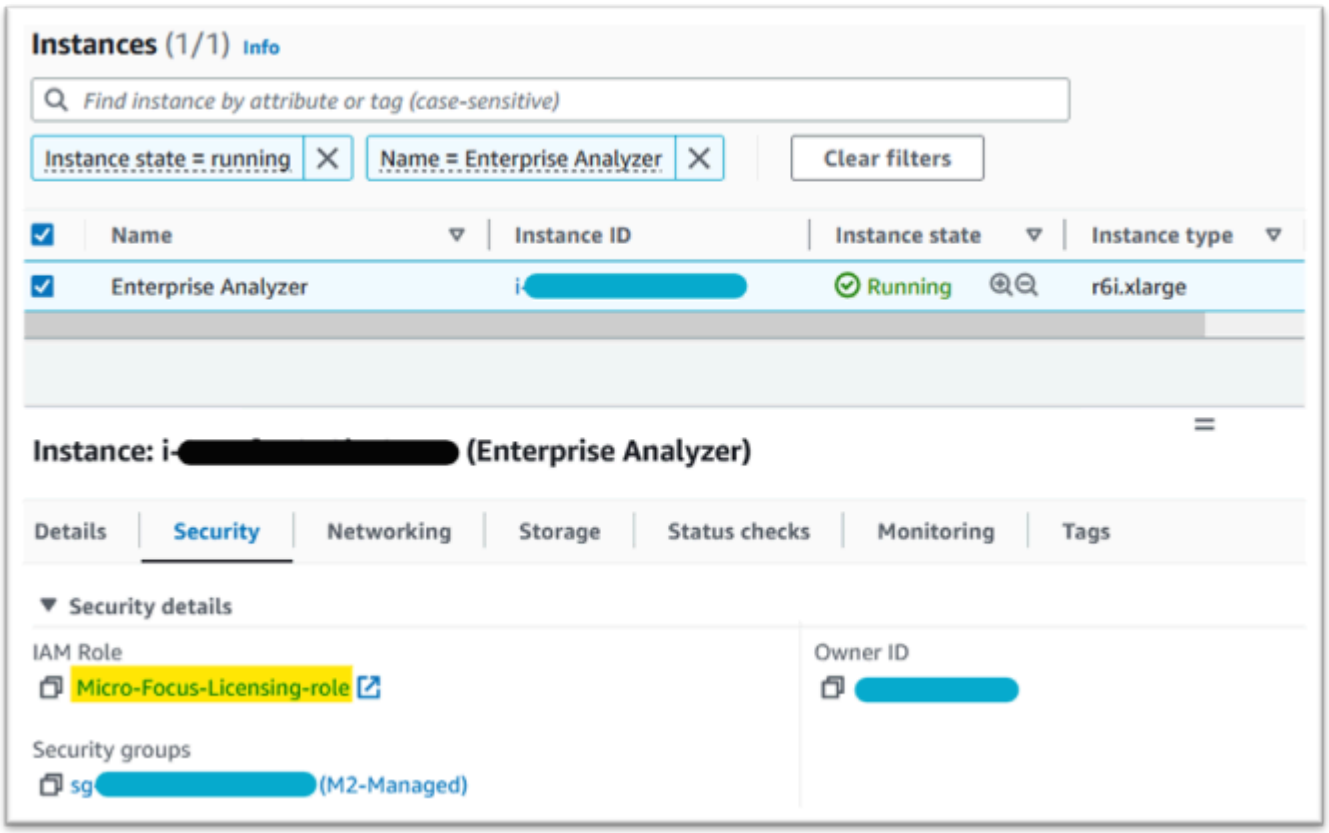
If you have trouble accessing or using the AMIs, the following information might help you.

Topics

- Verify the Amazon EC2 instance has the IAM licensing role
- Use the reachability analyzer
- Run the license-daemon
- License issues with Enterprise Server or Enterprise Build Tools on Linux after OS patching

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.

License issues with Enterprise Server or Enterprise Build Tools on Linux after OS patching

If you're having license issues with Enterprise Server or Enterprise Build Tools on Linux after OS patching, update the license daemon by downloading and running a patch script. To do that, use the following commands on the Command Prompt:

```
sudo curl https://d148y999krizvm.cloudfront.net/patch/v8/linux/patch.sh -o /var/
microfocuslicensing/bin/patch.sh
sudo chmod +x /var/microfocuslicensing/bin/patch.sh
sudo /var/microfocuslicensing/bin/patch.sh
sudo ./startmfcesd.sh
```

Note

This patch script will also work with version 9 even if the download path is for version 8.

Document history for the AWS Mainframe Modernization User Guide

The following table describes the documentation releases for AWS Mainframe Modernization.

Change	Description	Date
AWS Blu Age release notes 4.7.0 and 4.8.0	The 4.7.0 release of AWS Blu Age Runtime and AWS Blu Age Transformation Engines includes core dependency upgrades to Spring Boot 3.4.2 and Angular 19, enhancing security and performance while modernizing the frontend application framework. The 4.8.0 release includes key features like IBM IMS database transformation support and a new in-memory cache feature that improves performance for data-intensive programs for AS400 modernization..	May 18, 2025
AWS Blu Age release notes 4.6.0	This release of AWS Blu Age Runtime and AWS Blu Age transformation engine focuses on new features and improvements for zOS and AS400.	January 24, 2025
AWS Blu Age release notes 4.5.0	This release of AWS Blu Age Runtime and AWS Blu Age transformation engine focuses on key features of JCL	December 20, 2024

	support, binding directories and activation groups support for AS/400 modernized applications, and updated dependencies.	
AWS Blu Age FAQ	Learn about the AWS Blu Age refactoring capability with this comprehensive list of FAQs.	December 20, 2024
Amazon Q Developer transform for mainframe	You can learn about Amazon Q Developer's transform for mainframe feature that empowers you to modernize your legacy COBOL mainframe applicati ons to Java applications.	December 2, 2024
AWS Blu Age release notes 4.4.0	This release of the AWS Blu Age Runtime and transform ation engines is focused on upgrading critical dependenc ies and supported technolog ies while boosting performan ce in multiple functionalities.	November 13, 2024
Deploy AWS Blu Age Runtime on containers	You can learn how to set up AWS Blu Age Runtime on containers to deploy it on Amazon ECS(managed by Amazon EC2 or AWS Fargate), and Amazon EKS managed by Amazon EC2.	October 28, 2024

Updated LDAP user credentials	You can now create and manage LDAP user credentials for authentication and authorization using either AWS console or AWS CLI (or SDK).	October 21, 2024
Configure the Rocket Software managed application	You can now configure your applications with Rocket Software runtime engine to customize additional properties including integrations.	October 21, 2024
Replatforming with NTT DATA Unikix runtime	Learn about how to use Amazon Machine Images (AMIs) to create a tailored environment for rehosting and replatforming mainframe applications in AWS by using NTT DATA.	September 25, 2024
AWS Mainframe Modernization Application Testing IAM	You can learn about managing access for AWS Mainframe Modernization Application Testing with the available IAM features and policies.	September 20, 2024

AWS Blu Age releases	This section of the AWS Mainframe Modernization user guide captures all the AWS Blu Age versionin g details, AWS Blu Age release notes, AWS Blu Age upgrading instructions, and the overall AWS Blu Age lifecycle.	September 16, 2024
AWS Mainframe Moderniza tion components lifecycle	This page captures the lifecycle of each AWS Mainframe Modernization components, including their version upgrades, overall release plan, and end of support and retirement plans.	September 5, 2024
Assembler Conversion with mLogica	AWS Mainframe Moderniza tion Code conversion with mLogica is an AWS Mainframe Modernization feature that automatically converts z/OS mainframe Assembler code to COBOL.	July 22, 2024

Application Testing GA release	General availability docs for Application Testing. AWS Mainframe Moderniza tion Application Testing provides automated functiona l equivalence testing for your migration projects. This release includes data protection page, console workflows, and updates to other doc pages since preview.	June 12, 2024
Updated Managed Runtime for Rocket Softwaretutorial	This tutorial shows how to deploy and run the CardDemo sample application in an AWS Mainframe Modernization managed runtime environme nt with the Rocket Software runtime engine.	February 5, 2024
Release notes for AWS Blu Age Runtime and Moderniza tion Tools version 3.9.0.	This release of AWS Blu Age Runtime and Modernization Tools is focused on multiple transversal enhancements across the product striving to increase performance in high-availability architectures, along with new capabilities to raise jobs executions to the next level.	December 18, 2023
Transfer files between mainframe and AWS	New feature released to transfer files from the source mainframe to AWS.	November 27, 2023

Manage transactions for applications	New feature released to display and edit transactions for applications for AWS Mainframe Modernization.	October 16, 2023
Release notes for AWS Blu Age Runtime and Modernization Tools version 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.	August 4, 2023
You can now deploy a new version of an application when the application is stopped.	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.	July 26, 2023
AWS Blu Age runtime packaged for easier Amazon EC2 deployment	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.	July 6, 2023

Single sign-on to AWS Blu Age Blu Insights.	AWS Blu Age Blu Insights available from the AWS Management Console through single sign-on.	March 31, 2023
GA release	GA release of the AWS Mainframe Modernization User Guide.	June 8, 2022
Initial release	Initial release (public preview) of the AWS Mainframe Modernization User Guide.	November 30, 2021