



TIBCO ActiveMatrix® Service Grid - Container Edition

Cloud Deployment Guide

Version 1.0.0

December 2020

Document Updated: May 2021



Contents

Contents	2
ActiveMatrix Service Grid - Container Edition Overview	4
ActiveMatrix Service Grid - Container Edition Components	5
Preparing for Containerization	7
Dockerize ActiveMatrix Service Grid	9
Dockerfiles for ActiveMatrix Service Grid - Container Edition	9
Image Building Process	18
Building Docker Images for ActiveMatrix Service Grid - Container Edition	
Components and Applications	20
Building Application Configurator Docker Image	20
Running Application Configurator in Docker	21
Running Application Configurator with SSL Enabled	22
Configuring Session Timeout	23
Updating Default JVM Arguments and Logging Configuration for Application Configurator	23
Containerizing ActiveMatrix Service Grid Applications	24
Generating Application Configuration File (YAML)	24
Validating Application Configuration	27
Default JVM Arguments and Logging Configuration for Application Image and Builder Image	27
Building the Application Docker Image by Using Docker Build	28
Adding Custom Features to Builder Image	34
Creating ActiveMatrix Service Grid - Container Edition Builder Image Using S2I Tool	35
Creating Application Docker Image from Builder Image by Using S2I Tool	38
Setting JVM Arguments and Logging Configuration for Application Monitor	40
Building Application Monitor Agent Docker Image	40

Building TIBCO Enterprise Administrator Server Docker Image	41
Running ActiveMatrix Service Grid - Container Edition Application in Docker	42
Viewing Container Logs	42
Docker Run Command Reference	43
Inspecting Docker Images and Containers	47
Running ActiveMatrix Service Grid - Container Edition in Kubernetes Cluster	49
Running ActiveMatrix Service Grid - Container Edition on Kubernetes	50
Deploying Application Configurator to Kubernetes	51
Deploying ActiveMatrix Service Grid - Container Edition Application in Kubernetes ..	53
Deploying Application Monitor to Kubernetes	54
Deploying TIBCO Hawk Container Edition to Kubernetes	56
HTTP Endpoints for Health Checks	58
Configuring Probes in Kubernetes	58
Using ConfigMaps with ActiveMatrix Service Grid - Container Edition Applications ..	59
Using Kubernetes Secrets	60
Sample YAML Files for Kubernetes Cluster	61
Sample Kubernetes YAML File for Deploying an Application	61
Sample Kubernetes YAML File for Deploying Application Monitor	64
Sample Kubernetes YAML File for Deploying Hawk Container Edition	66
Troubleshooting	68
Appendix	71
Generating JDBC Driver DAA	71
TIBCO Documentation and Support Services	83
Legal and Third-Party Notices	85

ActiveMatrix Service Grid - Container Edition Overview

With TIBCO ActiveMatrix® Service Grid - Container Edition, you can run and monitor TIBCO ActiveMatrix Service Grid applications on container-based platforms such as Docker and Kubernetes.

To run an application, you need the application, all its dependencies, and configuration files. A container provides an OS environment to hold together all supporting components and tools that are required to run the application. This provides a consistent environment without the overhead of OS dependencies and other infrastructural requirements.

Key Highlights of ActiveMatrix Service Grid - Container Edition

- ActiveMatrix® Service Grid - Container Edition reduces complexity of setup and deployment in ActiveMatrix Service Grid Enterprise.
- There is no need of TIBCO EMS, Database, TCT configurations to run ActiveMatrix Service Grid - Container Edition.
- An application running on TIBCO ActiveMatrix Service Grid 3.x can be migrated to ActiveMatrix Service Grid - Container Edition.
- You can deploy ActiveMatrix Service Grid - Container Edition on Kubernetes cluster. Kubernetes provides many advantages such as scaling, high-availability, and recovery.

ActiveMatrix Service Grid - Container Edition Components

ActiveMatrix Service Grid - Container Edition consists of the Application Configurator (a web application for configurations), a DAA2Config command-line Tool, an Application Extractor to extract the application configuration, Application Monitor to monitor applications, and a Hawk microagent for monitoring.

Component	Description
Application Configurator	<p>The Application Configurator is a web application to perform configuration tasks:</p> <ul style="list-style-type: none">• Upload and Configure TIBCO ActiveMatrix application• Generate the YAML configuration file for an application• Generate other optional configurations such as Log4j configuration and JVM configuration.• Configure a range of entities such as substitution variables, bindings, resource templates, and JVM arguments <p>To perform any of these tasks, you must first upload the Distributed Application Archive (DAA) in the Application Configurator. For more information about the Application Configurator, see <i>TIBCO ActiveMatrix® Service Grid – Container Edition Administration</i>.</p>
DAA2Config Command-Line Tool	<p>The DAA2Config is a command-line tool to generate application configuration from DAA. This is a CLI alternative to the Application Configurator (which is GUI-based). All functionalities are similar in both.</p> <p>Script for this CLI tool:</p> <pre>amsgce-runtime-<version>/dautil/bin/DAA2Config.sh</pre> <p>For more information, see <i>TIBCO ActiveMatrix® Service Grid – Container Edition Administration</i>.</p>

Component	Description
Application Monitor	<p>You can monitor ActiveMatrix Service Grid - Container Edition applications deployed on Docker and Kubernetes platforms by using Application Monitor.</p> <p>ActiveMatrix Service Grid - Container Edition provides an Application Monitor agent, which allows you to monitor ActiveMatrix Service Grid - Container Edition applications. For more information, see <i>ActiveMatrix Service Grid - Container Edition Monitoring</i>.</p>
Application Extractor	<p>You can use the Application Extractor to extract application DAA and the application deployment configuration from ActiveMatrix Service Grid 3.x setup and use the extracted files to containerize the application. For more information, see <i>TIBCO ActiveMatrix® Service Grid - Container Edition Quick Start</i>.</p>
Hawk Monitoring	<p>ActiveMatrix Service Grid - Container Edition provides a Hawk microagent to monitor the applications by using Hawk® Container Edition.</p> <p>Each ActiveMatrix Service Grid - Container Edition node contains one Hawk microagent. Each ActiveMatrix Service Grid - Container Edition node contains one user application and system applications such as platform app, monitoring app, and mediation app. The Hawk agent runs on each Kubernetes node. For more information about Hawk concepts and procedures, see the Hawk® Container Edition documentation. For more information about Hawk microagent methods, see <i>ActiveMatrix Service Grid - Container Edition Monitoring</i>.</p>



Note: ActiveMatrix Service Grid - Container Edition version is 1.0.0. However, the TIBCO ActiveMatrix Service Grid - Container Edition runtime node version is 5.0.0. The node version is displayed as 5.0.0 in logs and in the Application Monitor UI.

Preparing for Containerization

Supported Versions

Before you begin, see the Readme for information about the supported versions of Docker and cloud platforms.

Concepts

You must be familiar with the following concepts and services:

- Docker concepts. See [Docker Documentation](#).
- Kubernetes concepts. See [Kubernetes Documentation](#).

You must have administration knowledge of the cloud platform and the service that you want to use.

Infrastructure Requirements

Ensure that you have the following infrastructure in place:

- A machine with Docker installation (see the Readme for the Docker version to use) and initial setup based on your operating system, to generate Docker images. For complete information about Docker installation, see [Docker Documentation](#).
- The machine on which you are building the Docker image must have Java 11 installed. <JAVA_HOME> must be added in the *Path* environment variable. Java 11 is the default and supported Java Runtime Environment(JRE) for ActiveMatrix Service Grid - Container Edition.
- Download TIBCO ActiveMatrix® Service Grid - Container Edition software package from the [TIBCO eDelivery](#). Extract the `amsgce-runtime-<version>.zip` file to a temporary folder on the machine. The `amsgce-runtime-<version>` folder contains all the Dockerfiles and scripts to build ActiveMatrix Service Grid - Container Edition components Docker images and deploy the components on cloud platform.
- You must have file read, write, and execute permissions on the machine where you are building Docker images. Ensure that you have write permission to the `amsgce-runtime-<version>` folder.
- For monitoring ActiveMatrix Service Grid - Container Edition applications, install

TIBCO Enterprise Administrator 2.4.0. For installation instructions, see [TIBCO Enterprise Administrator documentation](#).

- (Optional) For monitoring ActiveMatrix Service Grid - Container Edition using Hawk - Container Edition (see the Readme for the version to use), download Hawk - Container Edition software package from the [TIBCO eDelivery](#). Extract the Hawk - Container Edition archive file to a temporary folder on the machine. For Hawk - Container Edition concepts and procedures, see [TIBCO Hawk Container Edition documentation](#).
- If you are running the application in a Kubernetes cluster on a cloud platform, ensure that you have an active account on that cloud platform.

i Note: ActiveMatrix Service Grid - Container Edition does not require TIBCO Enterprise Message Service for messaging bus and message notification.

Dockerize ActiveMatrix Service Grid

To run ActiveMatrix Service Grid - Container Edition in Docker, you must build images of the components.

ActiveMatrix Service Grid - Container Edition does not have a universal installer. Deployment package contains only the Dockerfiles with preloaded settings for creating a Docker image of each component. After the Docker image of each component is created, you can run it in the Docker container.

To deploy and run different ActiveMatrix Service Grid - Container Edition components on Docker and Kubernetes, see the following topics:

- [Dockerfiles for ActiveMatrix Service Grid - Container Edition](#)
- [Building Docker Images for ActiveMatrix Service Grid - Container Edition Components and Applications](#)
- [Running ActiveMatrix Service Grid - Container Edition Application in Docker](#)
- [Running ActiveMatrix Service Grid - Container Edition in Kubernetes Cluster](#)

Dockerfiles for ActiveMatrix Service Grid - Container Edition

The deployment package contains the following Dockerfiles and scripts for building images of ActiveMatrix Service Grid - Container Edition components:

Script	Purpose of the Script
<code>amsgce-runtime- <version>/runtime/build/build_ amxce.sh</code>	To build the Docker image of ActiveMatrix Service Grid - Container Edition application.
<code>amsgce-runtime- <version>/runtime/build/s2i/build_ amxce_s2i.sh</code>	To create the ActiveMatrix Service Grid - Container Edition builder Docker image by using Source-To-Image(S2I) tool. By using the

Script	Purpose of the Script
	S2I tool, you can create a builder image, which contains an operating system, all required libraries, and <TIBCO_HOME> with all the required scripts. You can use the builder image as a base image to build a Docker image of applications. For more information about Source-To-Image(S2I) tool, see Source-to-Image (S2I) documentation.
amsgce-runtime- <version> /applicationConfigurator/build/build_amsce.sh	To build the Docker image of the Application Configurator.
amsgce-runtime- <version>/teaagent/build/build_amsceteaagent.sh	To build the Docker image for ActiveMatrix Service Grid - Container Edition TEA agent.

The following table lists Dockerfiles provided with ActiveMatrix Service Grid - Container Edition for different platforms, with the list of the script files that are using the Dockerfile, and the type of the container. The default base image used is Alpine Linux with Java 11. You can specify the other Dockerfile than the default one by using the following command:

Example:

```
dockerfile=Dockerfile_ubi8_java11 ./build_amsce.sh --image_tag
"bookstore:1.0" -app /Users/amsce/demo/sample_app
```

Dockerfile	Details
Location: amsgce-runtime-<version>/runtime/docker	
Dockerfile_alpine_java11	Platform: Alpine Linux Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition

Dockerfile	Details
	<p>Application</p> <p>Associated script file: build_amxce.sh</p>
Dockerfile_ubi7_java11	<p>Platform: Red Hat Universal Base Image</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition Application</p> <p>Associated script file: build_amxce.sh</p>
Dockerfile_ubi8_java11	<p>Platform: Red Hat Universal Base Image</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition Application</p> <p>Associated script file: build_amxce.sh</p>
Dockerfile_ubuntu_java11	<p>Platform: Ubuntu</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition Application</p> <p>Associated script file: build_amxce.sh</p>
Dockerfile_alpine_glibc_java11	<p>Platform: Alpine Linux</p> <p>If you want create image with Alpine Linux and TIBCO JRE or Oracle JRE, you need to install glibc.</p> <p>Use this Docker file which has steps to install glibc.</p> <p>Associated script file: build_amxce.sh</p>
Location: amsgce-runtime- <i><version></i> /runtime/docker/s2i	

Dockerfile	Details
Dockerfile_alpine_java11	<p>Platform: Alpine Linux</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition Builder Image</p> <p>Associated script file: build_amxce_s2i.sh</p>
Dockerfile_ubi7_java11	<p>Platform: Red Hat Universal Base Image</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition Builder Image</p> <p>Associated script file: build_amxce_s2i.sh</p>
Dockerfile_ubi8_java11	<p>Platform: Red Hat Universal Base Image</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition Builder Image</p> <p>Associated script file: build_amxce_s2i.sh</p>
Dockerfile_ubuntu_java11	<p>Platform: Ubuntu</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition Builder Image</p> <p>Associated script file: build_amxce_s2i.sh</p>
Dockerfile_alpine_glibc_java11	<p>Platform: Alpine Linux</p> <p>If you want create image with Alpine Linux and TIBCO JRE or Oracle JRE, you need to install glibc.</p>

Dockerfile	Details
	<p>Use this Docker file which has steps to install glibc.</p> <p>Associated script file: build_amxce_s2i.sh</p>
Location: amsgce-runtime-<version>/teaagent/docker	
Dockerfile_alpine_java11	<p>Platform: Alpine Linux</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition TEA Agent</p> <p>Associated script file: build_amxceteaagent.sh</p>
Dockerfile_ubuntu_java11	<p>Platform: Ubuntu</p> <p>Used for Creating Docker Image of: ActiveMatrix Service Grid - Container Edition TEA Agent</p> <p>Associated script file: build_amxceteaagent.sh</p>
Dockerfile_alpine_glibc_java11	<p>Platform: Alpine Linux</p> <p>If you want create image with Alpine Linux and TIBCO JRE or Oracle JRE, you need to install glibc.</p> <p>Use this Docker file which has steps to install glibc.</p> <p>Associated script file: build_amxceteaagent.sh</p>

For more information about Dockerfile structure, see [Docker Documentation](#).

Only the following base images and versions are tested in ActiveMatrix Service Grid - Container Edition 1.0.0:

Base Image	Version
Alpine Linux	3.12
	3.9.6
Ubuntu	18.04.5
Red Hat UBI	8.2
	7.8

The following Platforms are supported to build the Docker images:

- Windows with terminal emulator
- MacOS
- Linux

The following sections describe Environment Variables and Labels for ActiveMatrix Service Grid - Container Edition Docker images.

Environment Variables (ENV)

The ENV instruction is used to set the environment variables in the Dockerfile. These variables consist of key-value pairs that can be accessed from within the container by scripts and applications alike. The syntax for the ENV instruction is:

```
ENV key value
```

You can view environment variables and labels for a Docker image by using Docker inspect command:

```
docker image inspect <Docker Image>
```

The default ActiveMatrix Service Grid - Container Edition Dockerfiles have the following common environment variables:

Environment variables	Description
<i>TIBCO_AMXCE_VERSION</i>	ActiveMatrix Service Grid - Container Edition version

Environment variables	Description
<i>TIBCO_AMXCE_APPTEMPLATE</i>	<p>Name of the application template from which the application is created.</p> <p>Each application is associated with a single application template.</p>
<i>TIBCO_AMXCE_APPTEMPLATE_VERSION</i>	Version of the application template.
<i>TIBCO_AMXCE_NODE_HTTP_PORTS</i>	<p>Exposed ports of the container. Port 9998 is added by default in the exposed ports, which is used for monitoring application.</p> <p>Default: 9998</p>
<i>TIBCO_AMXCE_UID</i>	<p>Build date timestamp.</p> <p>This is used to create the default keystore password.</p>
<i>TIBCO_AMXCE_APPLICATION_LOCATION</i>	<p>Location where applications are stored in the Docker image.</p> <p>Default: /opt/tibco/tibco.home/data/applications/daas</p>

Labels used in ActiveMatrix Service Grid - Container Edition Docker Files

The LABEL instruction adds metadata to an image. A LABEL is a key-value pair.

Label	Description
<i>maintainer</i>	<p>Describes author of the Docker image.</p> <p>Default: TIBCO Software Inc.</p>
<i>tibco.amxce.app_description</i>	Description of the application from configuration YAML file.

Label	Description
<i>tibco.amxce.app_template_name</i>	Name of the application template from which the application is created.
<i>tibco.amxce.app_template_version</i>	Version of the application template from which the application is created. This label is set from application DAA. If configuration YAML file and DAA have a different application template version, then the label is set from the DAA application template version.
<i>tibco.amxce.build_date</i>	Date and time stamp when the Docker image is built.
<i>tibco.amxce.env_name</i>	Environment name in which the application is running in the container.
<i>tibco.amxce.node_name</i>	Node on which the application container is running.
<i>tibco.amxce.version</i>	Version of ActiveMatrix Service Grid - Container Edition

Labels and Environment Variables for S2I Builder Image

Environment Variables

Environment Variable	Description
<i>TIBCO_AMXCE_VERSION</i>	ActiveMatrix Service Grid - Container Edition version

Environment Variable	Description
<i>TIBCO_AMXCE_UID</i>	Build date timestamp.
<i>TIBCO_AMXCE_APPLICATION_LOCATION</i>	Location where applications are stored in the Docker image. Default: /opt/tibco/tibco.home/data/applications/daas
<i>TIBCO_AMXCE_DATA_FOLDER</i>	Root location where all ActiveMatrix Service Grid - Container Edition files are stored in the Docker image. Default: /opt/tibco/tibco.home/data

Builder Image Labels

Label	Description
org.opencontainers.image.revision	Source control revision identifier for the packaged software.
org.opencontainers.image.created	Date and time on which the image was built.
io.openshift.s2i.scripts-url	Location of S2I build script in the Docker image. Default: image:///usr/libexec/s2i

Labels for Application Image Created from Builder Image

Label	Description
io.openshift.expose-services	Exposed ports of the container

Label	Description
org.opencontainers.image.created	Image build date and time stamp. Format: date -u +'%Y-%m-%dT%H:%M:%SZ'
org.opencontainers.image.vendor	Name of the distributing organization that is TIBCO.
org.opencontainers.image.title	Name of the application template from which the application is created.
org.opencontainers.image.description	Application description

Image Building Process

The following section lists the main stages of building images from the Dockerfiles. For detailed steps and comments, refer to the template Dockerfiles at the following locations:

amsgce-runtime-*<version>*/runtime/docker

amsgce-runtime-*<version>*/runtime/docker/s2i

amsgce-runtime-*<version>*/teaagent/docker

1. Includes base Operating System:

The Dockerfiles begin with FROM command which specifies the image that starts the build process. The default base image on the provided Dockerfile is Alpine Linux with Java 11. For example,

```
FROM alpine:3.9
```

2. Copy and unzip tibco.home folder:

The build script copies tibco.home folder from amsgce-runtime-*<version>* folder.

```
COPY tibco.home.zip /opt/tibco/
```

3. Adds Java (supports external JRE):

After specifying the base image, image build script installs Java. It can be externally downloaded JRE or installed by using respective Operating System web repository.

```
RUN ./tmp/java/install_jdk.sh
```

4. Uses intermediate build stages to reduce image size:

Deletes unused files. The runtime zip file is common for three different types of images that is Runtime, Application Configurator, Application Monitor (TEA agent), so based on the current image being built, this script deletes unused plugins and features.

5. Creates an user "amxceur":

```
RUN adduser -S $USER -G $GROUP
```

6. Offline Deployment:

Scripts folder contains ant script to support <CONFIG_HOME> creation, integration node and offline deployment.

```
RUN $DATA_FOLDER/scripts/config.sh
```

7. Sets LABEL and ENV variables.

8. Set EXPOSE port:

Sets all exposed ports by scanning DAA and configuration file for all HTTP connectors port.

```
EXPOSE ${AMXCE_NODE_HTTP_PORTS}
```

9. Sets ENTRYPOINT:

On container start it calls start.sh file.

```
ENTRYPOINT ["/opt/tibco/tibco.home/data/scripts/start.sh"]
```

Building Docker Images for ActiveMatrix Service Grid - Container Edition Components and Applications

To run the ActiveMatrix Service Grid - Container Edition components on Docker or Kubernetes platform, you must first build the Docker images for the components and applications.

See the following topics for building Docker images for ActiveMatrix Service Grid - Container Edition components and applications:

- [Building Application Configurator Docker Image](#)
- [Containerizing ActiveMatrix Service Grid Applications](#)
- [Building Application Monitor Agent Docker Image](#)
- [Building TIBCO Enterprise Administrator Server Docker Image](#)

Building Application Configurator Docker Image

The Application Configurator is a web application, which you can use to configure the TIBCO ActiveMatrix application and generate a YAML configuration file for the application. The YAML file is required for creating the Docker image of the application.

Before you begin

- See [Preparing for Containerization](#).
- (Optional) To update default JVM arguments and Log4j configuration for the Application Configurator, see [Updating Default JVM Arguments and Logging Configuration for Application Configurator](#).
- You can run the Application Configurator with SSL enabled. For more information, see [Running Application Configurator with SSL Enabled](#).

Procedure

1. Navigate to `amsgce-runtime-<version>/applicationConfigurator/build`.
2. Build the Docker image of TIBCO ActiveMatrix Application Configurator by using the

following script:

```
./build_amxce.sh
```

i Note: The Application Configurator uses the following nomenclature, which cannot be changed:

- Node name: ACNode
- Environment name: ACEEnvironment
- Image name: amxce_ac:1.0

The default port exposed by the Application Configurator container is 8087. You can change the port exposed by the Application Configurator container by updating the port value in the `com.tibco.amxce.appconfig.config.yaml` file. This YAML file is provided at `amsgce-runtime-<version>/applicationConfigurator/application`.

Running Application Configurator in Docker

You can run the Application Configurator in a Docker by using the `docker run` command.

Before you begin

- See [Preparing for Containerization](#).
- Ensure that you have created a Docker image of the Application Configurator.

Procedure

1. Run the Docker image of the Application Configurator by using the following command:

Syntax:

```
docker run --rm -p <CONTAINER_PORT>:<HOST_PORT> --name <CONTAINER_NAME> <APPLICATION_IMAGE_NAME>:<IMAGE_VERSION>
```

Example:

```
docker run --rm -p 8087:8087 -p 9998:9998 --name amxceac amxce_ac:1.0
```

Here port 8087 is the application port and port 9998 is used for monitoring.

2. You can access the Application Configurator by opening the following URL in a browser: `http://<hostname>:<hostport>/appconfig` or `http://<IP>:<hostport>/appconfig`.
For example: `http://localhost:8087/appconfig/`.

What to do next

In the Application Configurator, you can configure an application and generate a configuration YAML file. For more information about generating application configuration YAML file, see *ActiveMatrix Service Grid - Container Edition Administration*.

Running Application Configurator with SSL Enabled

In the YAML file, you can enable SSL on the HTTP port that is used by the Application Configurator. Here are some resources required during the procedure:

- Sample YAML file:
`amsgce-runtime-
<version>
/samples/applicationConfigurator/ssl/com.tibco.amxce.appconfig.config.yaml`
- Certificates for default SSL configuration stored in the keystore:
`appconfig.ssl.keystore.jceks`
You can find this keystore at `amsgce-runtime-
<version>/applicationConfigurator/application/certs`.

Procedure

1. Copy `com.tibco.amxce.appconfig.config.yaml` from `amsgce-runtime-
<version>/samples/applicationConfigurator/ssl` to `amsgce-runtime-
<version>/applicationConfigurator/application/`.
2. To use custom certificate, copy the keystore file to `amsgce-runtime-
<version>/applicationConfigurator/application/certs`.
3. In the `com.tibco.amxce.appconfig.config.yaml` file, update the following parameters:
 - `keyStorePassword`
 - `keyAlias`

- keyPassword
4. Build the Application Configurator Docker image. For instructions, see [Building Application Configurator Docker Image](#).
 5. Run the Application Configurator container. For instructions, see [Running Application Configurator in Docker](#).

Configuring Session Timeout

Application Configurator UI has default session timeout as 30 minutes. If session remains idle for 25 minutes, you will get a warning to reset the timeout. If the session times out, you can not complete the configuration further or download the configuration already completed. The configuration is not saved, you must download the configuration before session timeout. You can update the default session timeout value by specifying `sys.sessionTimeout` environment variable, when running the Application Configurator container. The timeout value must be in milliseconds.

Example:

```
docker run --rm -p 8087:8087 -p 9998:9998 -e sys.sessionTimeout=3600000  
--name amxce_ac amxce_ac:1.0
```

In this example command timeout of 60 minutes is set.

Updating Default JVM Arguments and Logging Configuration for Application Configurator

To update default JVM Arguments and Logging Configuration for Application Configurator, update the following files before creating the Application Configurator Docker image:

Log4j Configuration: `amsgce-runtime-<version>/runtime/scripts/node-log4j.xml_template`

JVM Arguments: `amsgce-runtime-<version>/runtime/scripts/node.properties`

This updated configuration will be applied to all applications configured by using Application Configurator.

Containerizing ActiveMatrix Service Grid Applications

ActiveMatrix Service Grid application comprises of a common TIBCO ActiveMatrix runtime and an application-specific code. Thus, to containerize a TIBCO ActiveMatrix application, TIBCO ActiveMatrix runtime and an application DAA are included in the Docker image.

You can build the TIBCO ActiveMatrix application Docker image in the following ways:

- [Building the Application Docker Image by Using Docker Build](#)

You can build an application Docker image by using the script provided with ActiveMatrix Service Grid - Container Edition and `docker build` command. In this way, each time, you need to build all dependencies (such as operating system, `<TIBCO_HOME>`, libraries, and tools) required to run an application.

- [Creating Application Docker Image from Builder Image by Using S2I Tool](#)

With OpenShift S2I tool, you can build reproducible Docker images. First, you must create the builder image, which contains the operating system, `<TIBCO_HOME>`, all required libraries, and tools. You can build an application Docker image based on this builder image. You need not build all required dependencies each time you build an application Docker image.

Generating Application Configuration File (YAML)

In TIBCO ActiveMatrix Service Grid 3.x, we have Administrator UI to configure an application before deployment. All configurations are stored in the Administrator database.

ActiveMatrix Service Grid - Container Edition, does not have an Administrator database. All configurations are stored in the YAML configuration file. When building the Docker image of an application, the YAML file is copied to the Docker image. The YAML file has a higher precedence as compared with DAA, which means that the YAML file overrides the configuration given in the DAA. The Application Configuration (YAML) file with DAA and Log4j (Optional) and JVM configuration (Optional) is required to create a Docker image of an application. You can generate this configuration by using one of the following ways:

- [Application Configurator](#)
- [DAA2Config Command-Line Tool](#)
- [Application Extractor](#)

Application Configurator

The Application Configurator is a web application that you can use to configure an application and generate an application configuration (YAML) file. You must upload application DAA as an input to the Application Configurator. From the Application Configurator UI, you can configure different entities such as Substitution variables, Bindings, Resource Template, JVM Arguments, and Log4j configuration. You can download the generated application configuration (YAML) file with DAA as a .zip file. You can provide this .zip file as an argument when building an application Docker image. For more information about generating the Application Configuration (YAML) file by using the Application Configurator, see *ActiveMatrix Service Grid - Container Edition Administration*.

DAA2Config Command-Line Tool

DAA2Config is a command line tool, which you can use to generate the application configuration YAML file from DAA. This tool generates the application configuration YAML file with default values. You can edit the file in a text editor or upload the file in the Application Configurator to configure an application. You can use the script provided in the `amsgce-runtime-<version>/dautil/bin/DAA2Config.sh` file. You must pass the DAA file name with the path when running this script. You can specify the location where to generate the Application Configuration (YAML) file. If you do not specify, it is generated in the same folder where the DAA is located. For more information about generating Application Configuration (YAML) file by using DAA2Config CLI, see *ActiveMatrix Service Grid - Container Edition Administration*.

Application Extractor

To generate application configuration (YAML) files for applications deployed in your TIBCO ActiveMatrix Service Grid 3.x setup, you can use the Application Extractor. The Application Extractor tool is a TIBCO ActiveMatrix Service Grid 3.x-based SOA application that you can use to extract DAA and deployment configuration of an application in YAML files, so that you can containerize an application. You can download the DAA and configuration YAML file as a .zip file. For single or multiple applications, the format for the downloaded zip file is `[Enterprise_Name]_[Environment_Name]_[Timestamp].zip`. For more information about how to deploy Application Extractor, see [Deploying the Application Extractor](#).

Deploying the Application Extractor

Use the Application Extractor to extract your application DAA and deployment configuration from ActiveMatrix Service Grid 3.x, so that you can containerize the application.

The Application Extractor is provided at `amsgce-runtime-
<version>/applicationExtractor`. You must deploy the Application Extractor to your existing ActiveMatrix Service Grid 3.x enterprise to extract applications for migration to ActiveMatrix Service Grid - Container Edition.

i Note: You can use the Application Extractor with TIBCO ActiveMatrix Service Grid 3.3.0 Hotfix 23 and later.

Deploying the Application Extractor from the Administrator UI

You must deploy the Application Extractor to the SystemNode or SystemNodeReplica. It uses ActiveMatrix Administrator default HTTP connector. Do not deploy it to any runtime node.

1. In Administrator UI, click **SystemEnvironment**.
2. Create a new application by uploading the DAA that is provided at `amsgce-runtime-
<version>/applicationExtractor/com.tibco.amxce.amx.extractor.app.daa`
3. Deploy the DAA to SystemNode. No additional configuration is needed.

Deploying the Application Extractor from the Administrator CLI

Sample build files to deploy the Application Extractor from the Administrator CLI are provided at

`amsgce-runtime-<version>/applicationExtractor/amx-3.x-cliScripts`.

1. Change the proper settings in `remote_props.properties`.
2. Run the CLI script.

```
ant -f com.tibco.amxce.amx.extractor.app.deployment-build.xml
```

Accessing the Application Extractor UI

You can access the Application Extractor with the following URL:

`http://<admin-machine>:8120/appextractor`.

- `<admin-machine>`: URL of the machine on which the Administrator instance is running.
- `<port-number>`: Port number to access ActiveMatrix Administrator (default port 8120).

The Application Extractor uses the default HTTP connector of ActiveMatrix Administrator, that is, `amxAdminDefaultHttpConnector`. You cannot change this. The Application Extractor runs on HTTP port defined in `amxAdminDefaultHttpConnector`. If SSL is enabled in this HTTP connector, you can access the Application Extractor using `https://`.

What to do next

You can extract an existing application or multiple applications from TIBCO ActiveMatrix Service Grid 3.x setup as a .zip file and containerize these applications. For more information, see *TIBCO ActiveMatrix® Service Grid - Container Edition Quick Start*.

Validating Application Configuration

The application configuration artifacts (YAML file, JVM arguments, Log4j configuration) generated directly by Application Configurator, DAA2Config, or Application Extractor are error free. You can use a text editor to further update these artifacts. This manual step may introduce inconsistencies to the configurations. In such a case, you can use `checkConfig` command to validate the configuration. You must fix errors reported by the `checkConfig` command, otherwise Image building will fail.

By using the `amsgce-runtime-<version>\dautil\bin\checkConfig` command, you can validate the application configuration including configuration YAML file, JVM arguments file, and Log4j configuration file. You can use the `checkConfig` command to validate the configuration before containerizing an application. When running the `checkConfig` command, you need to specify the location of the folder that contains DAA, configuration YAML file, JVM arguments file, and Log4j configuration file. For more information, see "Validating Application Configuration" in *TIBCO ActiveMatrix® Service Grid - Container Edition Administration*.

Default JVM Arguments and Logging Configuration for Application Image and Builder Image

To update default JVM Arguments and Logging Configuration for an application, update the following files before creating the application Docker image. If you are using OpenShift S2I tool to build the Builder image, the default configuration is applied to the Builder image also.

Log4j Configuration: `amsgce-runtime-<version>/runtime/scripts/node-log4j.xml_template`

JVM Arguments: `amsgce-runtime-<version>/runtime/scripts/node.properties`

This updated configuration will be applied to all applications images built.

i Note: If you include `node_jvm_parameters.config.yaml` and `node-log4j.xml` file in the build pack then these files will override the default JVM arguments and Log4j configuration.

Building the Application Docker Image by Using Docker Build

Scripts and Dockerfiles are provided in the deployment package to build a Docker image for the ActiveMatrix Service Grid - Container Edition application.

Before you begin

- See [Preparing for Containerization](#).
- For more information about Dockerfiles, see [Dockerfiles for ActiveMatrix Service Grid - Container Edition](#).
- Ensure that you have generated and configured Application Configuration (YAML) file and optionally the Log4j and JVM configuration files. For more information, see [Generating Application Configuration File \(YAML\)](#).
- If you do not provide `node_jvm_parameters.config.yaml` and `node-log4j.xml` files, Image creation takes `amsgce-runtime-<version>/runtime/scripts/node.properties` and `amsgce-runtime-<version>/runtime/scripts/node-log4j.xml_template` files. You can update the settings of these files to update the default JVM Arguments and Logging Configuration for the application.
- (Optional) To copy custom features to application Docker image, see [Adding Custom Features to Application Docker Image](#).

Procedure

1. Navigate to `<temp_directory>/amsgce-runtime-<version>/runtime/build`.
2. To build TIBCO ActiveMatrix Application Docker image, run the following command. Specify the location of the folder or the `.zip` file that contains configuration YAML file, DAA, and optionally the Log4j and JVM configuration files by using `--app_location` argument.

Syntax:

```
./build_amxce.sh --app_location <Application Configuration folder
location> -i <Image name> -n <Node name> -e <Environment name> -l
<Image Build log level>
```

Example:

```
./build_amxce.sh --image_tag "bookstore:1.0" --app_location
/home/tibco/bookstore
```

The following table lists the arguments that you can pass when building an application Docker image.

Argument	Optional / Required	Description
-i / --image_tag <Image name>	Required	<p>Image tag for the Docker image. <Image name> must be in the form of "Image name (Repository):tag or version" such as bookstore:1.0.</p> <p>Image name must not contain space.</p>
-n / --amxce_node <Node name>	Optional	<p>ActiveMatrix Service Grid - Container Edition node name in the container.</p> <p>Default: AMXCENode</p> <p>Node name must start with an alphabet character. It can have only alphanumeric, underscore (_), and hyphen (-) characters.</p> <p>If the node name is specified when running the command, it overrides the node name specified in the YAML file.</p> <p>If the node name is not specified as a command argument, node name specified in the YAML file is used.</p> <p>If node name is not specified as a command argument or in YAML file, default node name is used.</p>

Argument	Optional / Required	Description
<code>-e / --amxce_env</code> <i><Environment name></i>	Optional	<p>ActiveMatrix Service Grid - Container Edition Environment name in a container.</p> <p>Default: AMXCEEnvironment</p> <p>If the environment name is specified when running the command, it overrides the environment name specified in the YAML file.</p> <p>If the environment name is not specified as a command argument, environment name specified in the YAML file is used.</p> <p>If environment name is not specified as a command argument or in YAML file, default environment name is used.</p>
<code>-l / --log_level</code> <i><Image Build Log level></i>	Optional	<p>Set <code>log_level</code> to 1 for detailed output.</p> <p>Helpful in debugging build failures.</p> <p>Default: 0</p>
<code>-java</code> <i><Java package file location></i>	Optional	<p>Specify the location of the Java Package to be used to create the image. This option can be used to provide pre-downloaded JDK or JRE to be used while creating the image. Image creation script skips downloading Java from a remote site when this option is provided. Specified Java package is automatically copied to the <code>amsgce-runtime-<i><version></i>/runtime/java</code> folder. Previously specified Package is removed from <code>amsgce-runtime-<i><version></i>/runtime/java</code> before copying the new package.</p> <div> <p>Note: Do not manually copy the package directly to <code>amsgce-runtime-<i><version></i>/runtime/java</code>.</p> </div>

Argument	Optional / Required	Description
		<p>Note: Do not specify the folder location. This must be the compressed Java package file location.</p> <p>Example:</p> <pre>dockerfile=Dockerfile_ubuntu_java11 ./build_amxce.sh -i test:1.0 -java /Users/Downloads/openjdk-11+28_linux-x64_bin.tar12.gz -app /Users/amxce/demo/sample_app</pre>
-app --app_location <Application folder location>	Required	<p>Location of the application folder or the .zip file that contains configuration YAML file, DAA, and optionally the Log4j and JVM configuration files.</p> <p>Note: Ensure that the application file or folder location is not in the amsgce-runtime- <version>/runtime folder. Image creation fails if the application location is in the runtime folder.</p> <p>Supported folder structure of the Application .zip file or the Application Folder:</p> <ul style="list-style-type: none"> • Application zip downloaded from Application Configurator can be used which follows the following structure inside the zip. It contains one or more custom features DAA, one application DAA, and one application configuration YAML directly under the root and the certs folder which contains the Keystores used by the application. • Application .zip extracted by Application Extractor can not be used for the -app argument. You will have to find the target application folder from extracted .zip file and

Argument	Optional / Required	Description
		<p>zip the files with the same file structure that Application Configurator creates in the downloaded .zip file.</p> <ul style="list-style-type: none"> Application folder or .zip must contain one DAA (optionally the Custom Feature DAA for JDBC Drivers), one yaml configuration file, the "certs" folder which contains the Keystore files. If you have multiple yaml configuration files, image creation will fail. <p>Note: Application configuration file with the .yml extension is not supported when creating an application Docker image. You must use application configuration file with the .yaml extension.</p> <p>Example:</p> <pre>./build_amxce.sh -i bookstore:1.0 -app /Users/amxce/demo/sample_app.zip</pre> <pre>./build_amxce.sh -i bookstore:1.0 -app /Users/amxce/demo/sample_app</pre>
dockerfile <Dockerfile Name>	Optional	<p>Specify Dockerfile name to use a Dockerfile other than the default one. Default Dockerfile is Dockerfile_alpine_java11.</p> <p>Example:</p> <pre>dockerfile=Dockerfile_ubi8_java11 ./build_amxce.sh --image_tag "bookstore:1.0" -app /Users/amxce/demo/sample_app</pre>
-h / --help	Optional	Displays help for the script file.

Checking Image Build Logs

You can view the logs at `amsgce-runtime-<version>/runtime/build/logs`. The name of the log file is `amxcebuild-<timestamp>.log`.

Adding Custom Features to Application Docker Image

Before you begin

Package a custom feature as DAA. For more information about how to generate JDBC driver DAA, see [Generating JDBC Driver DAA](#).

Procedure

1. To add custom features such as database drivers to your application Docker image, you must copy one or more custom feature DAA to the same folder which contains application DAA and application configuration YAML file.
2. Specify the location of the folder or the .zip file that contains custom feature DAA, configuration YAML file, application DAA, and optionally the Log4j and JVM configuration files by using `--app_location` argument. See [Building the Application Docker Image by Using Docker Build](#).

Third-Party Driver Details

The following table lists supported third-party drivers.

Supported Driver	Description
JDBC	<ul style="list-style-type: none">• TIBCO enabled JDBC driver for IBM DB2 4.12.55• TIBCO enabled JDBC driver for IBM DB2 4.19.66• TIBCO enabled JDBC driver for IBM DB2 4.24.92• TIBCO enabled JDBC driver for Oracle 11.1.0• TIBCO enabled JDBC driver for Oracle 12.1.100• TIBCO enabled JDBC driver for Oracle 12.2.0• TIBCO enabled JDBC driver for Oracle 18.3.0• TIBCO enabled JDBC driver for Oracle 19.3.0

Supported Driver	Description
	<ul style="list-style-type: none">• TIBCO enabled JDBC driver for Microsoft SQL Server 4.0.0• TIBCO enabled JDBC driver for Microsoft SQL Server 4.2.0• TIBCO enabled JDBC driver for Microsoft SQL Server 6.0.0• TIBCO enabled JDBC driver for Microsoft SQL Server 7.0.0• TIBCO enabled JDBC driver for PostgreSQL 10.7.0 (postgresql-42.2.8.jar is required)• TIBCO enabled JDBC driver for PostgreSQL 11.5.0 (postgresql-42.2.8.jar is required)

Adding Custom Features to Builder Image

Before you begin

Package a custom feature as DAA. For more information about how to generate JDBC driver DAA, see [Generating JDBC Driver DAA](#).

Procedure

1. To add custom features such as database drivers to all your application Docker images, you must copy one or more custom feature DAA to a folder. Ensure that you have read write permissions on the folder which contains custom feature DAA.
2. Specify the location of folder containing custom feature when building base image by using the option:

-cf | --custom_feature

Example:

```
./build_amxce_s2i.sh --image_tag "amxce_builder:1.0" --custom_feature  
"/path/to/Custom Feature folder"
```

Note: File or folder location specified in `-cf` option must not be inside `amsgce-runtime-<version>/runtime` folder.

Note: You must not copy anything in the `amsgce-runtime-<version>/runtime/application` folder. The `runtime/application` folder is used as a temporary folder and deleted each time you create a new image.

Result

When the builder image is created, you can find custom features later in the running container at `/opt/tibco/tibco.home/data/applications/custom_feature`.

Creating ActiveMatrix Service Grid - Container Edition Builder Image Using S2I Tool

OpenShift S2I is a tool for building reproducible Docker images. By using the S2I tool, you can create a builder image, which contains an operating system, all required libraries, and `<TIBCO_HOME>` with all the required scripts. You can reuse this builder image to create a new application Docker image. S2I supports reusing of previously downloaded dependencies and previously built artifacts. You need not build all dependencies each time you build an application Docker image.

Before you begin

- See [Preparing for Containerization](#).
- Download and install latest binary of S2I from <https://github.com/openshift/source-to-image/releases/>. After installation, ensure to add the location of the S2I binary to your PATH environment variable.
- For default JVM arguments and Log4j configuration, see [Default JVM Arguments and Logging Configuration for Application Image and Builder Image](#).
- To deploy custom features such as database drivers to all application Docker images, see [Adding Custom Features to Builder Image](#).

Procedure

1. Navigate to `amsgce-runtime-<version>/runtime/build/s2i`.
2. Create Builder image by using the following script. You must create the builder image only once for the ActiveMatrix Service Grid - Container Edition release.

```
./build_amxce_s2i.sh -i <Builder Image Name>
```

Example:

```
./build_amxce_s2i.sh -i amxce_builderimage
```

The following table lists the arguments to pass when building Builder Docker image.

Argument	Optional / Required	Description
<code>-i / --image_tag</code> <code><Image name></code>	Required	Image tag for the Docker image. Image name must not contain space.
<code>-java <Java package file location></code>	Optional	Location of Java Package to be used to create the image. This option can be used to provide pre-downloaded JDK or JRE to be used while creating the image. Image creation script skips downloading Java from a remote site when this option is provided. Specified Java package is copied automatically to <code>amsgce-runtime-<i><version></i>/runtime/java</code> . Previously specified package is removed from <code>amsgce-runtime-<i><version></i>/runtime/java</code> before copying the new package. Note: Do not manually copy the package directly to <code>amsgce-runtime-<i><version></i>/runtime/java</code> .

Argument	Optional / Required	Description
		<p>Note: Do not specify the folder location. This must be the compressed Java package file location.</p> <p>Example:</p> <pre>dockerfile=Dockerfile_ubuntu_java11 ./build_amxce_s2i.sh -i bookstore:1.0 -java /Users/Downloads/openjdk-11+28_linux-x64_bin.tar12.gz</pre>
-cf --custom_feature	Optional	<p>To add a custom feature such as database drivers to all application images of S2I build, add custom feature to the builder image. Then this custom feature is installed on all application images created by using the builder image.</p> <p>Specify the folder containing custom feature or custom feature DAA file. If folder location is passed as an argument then all custom features in that folder are copied.</p> <p>Example:</p> <pre>./build_amxce_s2i.sh --image_tag "bookstore:1.0" --custom_feature "/path/to/Custom Feature folder"</pre>
dockerfile <Dockerfile Name>	Optional	<p>Specify Dockerfile name to use a Dockerfile other than the default one.</p> <p>Example:</p> <pre>dockerfile=Dockerfile_ubi8_java11 ./build_amxce_s2i.sh --image_tag "bookstore:1.0"</pre>
-h / --help	Optional	Displays help for the script file.

What to do next

You can create an application Docker image based on builder image. For more information, see [Creating Application Docker Image from Builder Image by Using S2I Tool](#).

Creating Application Docker Image from Builder Image by Using S2I Tool

You can build an application Docker image based on a builder image by using OpenShift S2I tool. This application image contains `<TIBCO_HOME>`, operating system, and all required libraries required to run an application.

Before you begin

- See [Preparing for Containerization](#).
- Ensure that you have generated and configured application Configuration (YAML) file. For more information about generating application configuration YAML file, see [Generating Application Configuration File \(YAML\)](#).
- Ensure that you have created ActiveMatrix Service Grid - Container Edition builder Docker image. For more information, see [Creating ActiveMatrix Service Grid - Container Edition Builder Image Using S2I Tool](#).

Procedure

To build application Docker image run the following command:

Syntax:

```
s2i build <Path to DAA and Application Configuration YAML file> <Builder Image Name > <Application Image name>
```

Example:

```
s2i build /home/amxce/com.tibco.restbt.sample.bookstore amxce_builder:1.0 bookstore:v1
```

You can pass the following arguments when building an application Docker image. To get a list of arguments, run the following command:

```
s2i usage <Builder Image Name>
```

Argument	Required	Description
<code>-e amxce_node</code> <code>"<Node name>"</code>	No	<p>ActiveMatrix Service Grid - Container Edition Node name.</p> <p>Default: AMXCENode</p> <p>Node name must start with a letter of the alphabet character. It can have only alphanumeric, underscore (_) and hyphen (-) characters.</p> <ul style="list-style-type: none"> • If the node name argument is used when running the command, it overrides the node name specified in the YAML file. • If the node name argument is not used, node name specified in the YAML file is used. • If node name is not specified as a command argument or in YAML file, default node name is used. <p>Example:</p> <pre>s2i build -e amxce_node="BookStoreNode" /home/amxce/com.tibco.restbt.sample.bookstore amxce_builder:1.0 bookstore:v1</pre>
<code>-e amxce_env</code> <code>"<Environment name>"</code>	No	<p>ActiveMatrix Service Grid - Container Edition Environment name.</p> <p>Default: AMXCEEnvironment</p> <ul style="list-style-type: none"> • If the environment name argument is used when running the command, it overrides the environment name specified in the YAML file. • If the environment name argument is not used, environment name specified in the YAML file is used.

Argument	Required	Description
		<ul style="list-style-type: none"> If environment name is not specified as a command argument or in the YAML file, default environment name is used. <p>Example:</p> <pre>s2i build -e amxce_env="TestEnv" /home/amxce/com.tibco.restbt.sample.bookstore amxce_builder:1.0 bookstore:v1</pre>

Checking Image Build Logs

You can view the logs at `amsgce-runtime-<version>/runtime/s2i/build/logs`. The name of the log file is `amxcebuild-<timestamp>.log`.

Setting JVM Arguments and Logging Configuration for Application Monitor

To update default JVM Arguments and Logging Configuration for Application Monitor, update the following files before creating the TEA agent image:

Log4j configuration

The Application Monitor uses logback for logger configuration. It has a default `logback.xml` file, located at `amsgce-runtime-<version>/teaagent/config/logback.xml`. To update the logging configuration, change this file before creating the Image.

JVM Arguments

To update JVM arguments, edit the `amsgce-runtime-<version>/teaagent/config/application_monitor.properties` file before creating the Docker image.

Building Application Monitor Agent Docker Image

ActiveMatrix Service Grid - Container Edition provides an Application Monitor agent. After you register the Application Monitor agent with TIBCO Enterprise Administrator, you can

monitor ActiveMatrix Service Grid - Container Edition applications by using the Application Monitor.

Before you begin

- See [Preparing for Containerization](#).
- To update JVM arguments and Log4j configuration for the Application Monitor, see [Setting JVM Arguments and Logging Configuration for Application Monitor](#).

Procedure

1. Navigate to `<temp_directory>/amsgce-runtime-<version>/teaagent/build`.
2. Build the Application Monitor agent Docker image by using the following script:

```
./build_amxceteaagent.sh
```

You can list the most recently created images by using the following command:

```
docker images
```



Note: `./build_amxceteaagent.sh` script creates the image with the default name `amxceteaagent:1.0`, though you specify different image name by using `-- image tag`.

Building TIBCO Enterprise Administrator Server Docker Image

Before you begin

Install TIBCO Enterprise Administrator (see ActiveMatrix Service Grid - Container Edition Readme for supported version). For installation instructions, see [TIBCO Enterprise Administrator documentation](#).

Procedure

For creating a Docker image of TIBCO Enterprise Administrator Server, see `readme.md` at `TEA_HOME/docker` in *TIBCO Enterprise Administrator Installation*.

Running ActiveMatrix Service Grid - Container Edition Application in Docker

After creating the application Docker image, the next step is to run the application in Docker.

Before you begin

- Build ActiveMatrix Service Grid - Container Edition application Docker image. For more information, see [Containerizing ActiveMatrix Service Grid Applications](#).

Procedure

Execute the `docker run` command on the machine where you have created the application Docker image.

```
docker run -d -p <Host_port>:<Container_port> -e <Environment_variables> --name <Container_name> <Application_Image_Name>:<Image_Version>
```

Example:

```
docker run -d -p 7777:7777 --name bookstore bookstore:1.0
```

For more information about the `docker run` command options, see [Docker Run Command Reference](#).

Viewing Container Logs

All component containers of ActiveMatrix Service Grid - Container Edition publish their logs on stdout. Container logs have some extra log lines than AMXCE node logs which gets printed before JVM starts and logger initializes.

To view logs of a particular container, run the following command:

```
docker logs <container_id>
```

Docker Run Command Reference

The docker run command is used for containerizing and running a ActiveMatrix Service Grid - Container Edition application by using a Docker image.

Syntax

```
docker run -p <HOST_PORT>:<CONTAINER_PORT> -e <ENVIRONMENT_VARIABLES>
<APPLICATION_IMAGE_NAME>:<IMAGE_VERSION>
```

Where:

- `-p <HOST_PORT>:<CONTAINER_PORT>` - Specify the host port and container port that you want to map.
- `<APPLICATION_IMAGE_NAME>` - Specify the name of the ActiveMatrix Service Grid - Container Edition application Docker image.
- `<IMAGE_VERSION>` - (Optional) Specify the version of the specified Docker image.
- `-e <ENVIRONMENT_VARIABLES>` - Use the `-e` option to set environment variables, as required, with syntax `VAR=Value`. You can use the following environmental variables at the run time.

Following are the environment variables you can specify when running the container:

Substitution Variables

You can override values for Substitution Variables by specifying the values as environment variables when running the container.

The format of environment variables must be as follows:

For Shared resource Substitution Variables: `svar.sr.<svar_name>=<svar_value>`

For Application Substitution Variables: `svar.app.<svar_name>=<svar_value>`

Example:

```
docker run -it -p 7777:7777 -p 2222:2222 -p 8010:8010 -e svar.sr.port=8010
-e svar.app.httpconn=httpConnector_new bookstore:1.0
```

i Note: If substitution variable type is password, do not specify plain text for password as SVAR value by using container environment variable, because password value is printed in container logs. But you can specify an encrypted value like '#!fVZYfnXhUJQmJgOVhkPzAwXNpL/xm53B'. For more information about creating an obfuscated password from ActiveMatrix Administrator CLI, see "Creating an Obfuscated Password" in *TIBCO ActiveMatrix Service Grid Administration Guide*.

Shared Resource User name and Password

You can specify the user name or password for the shared resource as the environment variable when you run the container. The format of the environment variables must be as follows:

To update user name: `sr.<SharedResourceName>.username=<NewUserName>`

To update Password: `sr.<SharedResourceName>.password=<NewPassword>`

i Note: This applies to the shared resources that use inline credentials (that is user name and password) provided in the Shared Resource Configuration. Example: JDBC Shared Resource

Example:

```
docker run -it -p 7777:7777 -p 2222:2222 -p 8010:8010 -e svar.sr.port=8010
-e sr.NewJDBCResource.username=USERNAME -e
sr.NewJDBCResource.password=password bookstore:1.0
```

JVM Arguments

You can override JVM arguments set in the image when running the container by passing `JAVA_OPTION` as environment variable. This overrides JVM arguments set when creating the image.

Example:

```
docker run --rm -e JAVA_OPTION="-Xmx512m -Xms128m -
XX:+HeapDumpOnOutOfMemoryError" bookstore
```

JAVA_EXTENDED_OPTION

To add JVM arguments (TRA properties from ActiveMatrix Service Grid 3.x) use this environment variable.

This environment variable adds property to the existing JVM arguments.

For example, to add `com.tibco.amf.node.disableSendServerVersion=true`, use the `JAVA_EXTENDED_OPTION` environment variable.

```
docker run --rm -e JAVA_EXTENDED_OPTION="-
Dcom.tibco.amf.node.disableSendServerVersion=true" bookstore
```

If you are adding more than one properties, add value to the same environment variable:

```
docker run --rm -e JAVA_EXTENDED_OPTION="-
Dcom.tibco.amf.node.disableSendServerVersion=true -Dproperty2=value2"
bookstore
```

For OpenID Single Sign-On with ADFS and SAML Single Sign-On with ADFS, you must pass JVM parameter `allowed.referrers` when running the container:

```
docker run --rm -p 9895:9895 --name saml_adfs_cont1 --add-host
example.com:192.0.2.24 -e JAVA_OPTION="-Xmx1024m -Xms128m -
XX:+HeapDumpOnOutOfMemoryError -
Dcom.tibco.amf.hpa.tibcohost.jetty.httpconnector.allowed.referrers=account
s.google.com,example.com" saml_adfs:v1
```

where "example.com" is ADFS machine name.

If you do not pass this JVM configuration then container will fail with the "Invalid Referrer Header" error.

OSGI Console Port

By default all ActiveMatrix Service Grid - Container Edition node enables `osgi.console` on 2222 port.

This port is configurable when running the container. Set environment variable `osgi.console` to change it.

Example:

```
docker run -e osgi.console=<port> <docker image>
```

valid port range is (0-65535).

You can view the following log lines when container runs:

```
-init:

SetOsgiConsolePort:
[echo]
[echo]
[echo]
#####
#####
[echo]
#####
#####
[echo]
[echo] 16 October 2020 06:08:32
[echo]
[echo] Checking for osgi.console in the environment properties and setting
osgi.console if found.
[echo]
[echo]
#####
#####
[echo]
#####
#####
[echo]
[echo]
[echo] Setting osgi.console = 2278

BUILD SUCCESSFUL
Total time: 1 second
```

Use the telnet quit command to close the connection after the debugging session is complete. Do not use the telnet exit command as this will shut down the AMSGCE Node.

Custom Logging Element

1. Use layout class as `com.tibco.tpcl.org.apache.log4j.PatternLayoutEx`.
2. In the Log4j configuration, use logging pattern as `%R{_cl.amxce.correlationId}`.
3. Set `_cl.amxce.correlationId` as environment variable.

Example:

```
docker run -p 7777:7777 -e _cl.amxce.correlationId=test123 bookstore.
```

Sample Node Log File:

```
<layout class="com.tibco.tpcl.org.apache.log4j.PatternLayoutEx">
  <param name="ConversionPattern" value="[%R{_cl.amxce.correlationId}] %R
{_cl.physicalCompId.matrix.env} %R{_cl.physicalCompId.matrix.node} [%t]
[%-5p] %c - %m%n"/>
</layout>
```

Sample Output:

As shown in the following sample, each line contains [test123] which is set as environment variable when running the container and the same is mentioned in Log4j configuration file of AMXCE node.

```
16 Dec 2019 20:41:51,584 [test123] AMXCENode [ComponentFrameworkTask]
[INFO ] org.eclipse.jetty.server.Server - jetty-9.2.25.v20180606
16 Dec 2019 20:41:51,614 [test123] AMXCENode [ComponentFrameworkTask]
[INFO ] org.eclipse.jetty.util.log.Logger - Opened
C:/tibco/scripts/workflows/alacarte/amxce/config.home/tibcohost/AMXCEInst
ance/host/logs/jetty.2019_12_16.request.log
16 Dec 2019 20:41:51,685 [test123] AMXCENode [ComponentFrameworkTask]
[INFO ] org.eclipse.jetty.server.NetworkTrafficServerConnector - Started
hello2Connector@6b92a741{HTTP/1.1}{0.0.0.0:7788}
16 Dec 2019 20:41:51,686 [test123] AMXCENode [ComponentFrameworkTask]
[INFO ] org.eclipse.jetty.server.Server - Started @47892ms
16 Dec 2019 20:41:51,686 [test123] AMXCENode [ComponentFrameworkTask]
[INFO ] com.tibco.amx.hpa.web.jetty.hello2Connector - TIBCO-AMX-HPA-
014364: Started Jetty server hello2Connector
16 Dec 2019 20:41:51,759 [test123] AMXCENode [ComponentFrameworkTask]
[INFO ] org.eclipse.jetty.server.handler.ContextHandler - Started
o.e.j.s.ServletContextHandler@4778fae8{/helloWorldPT,null,AVAILABLE}
```

Inspecting Docker Images and Containers

Docker Container Inspect

You can get information about a running container by using the `docker inspect` command. The `inspect` command lists the complete information of the container. You must pass the container ID with the `docker inspect` command.

To get the container ID, run the following command:

```
docker ps
```

Use the container ID in the first column of the output to inspect the container details.

```
docker inspect <Container ID>
```

You can get the following information about files and folders in the container. All folders are located at /opt/tibco.

- `tibco.home`: ActiveMatrix Service Grid - Container Edition installation folder; contains all product-specific files.
- `config.home`: Configuration folder location
- `/opt/tibco/tibco.home/data/application`: Contains application DAA and configuration files.

Docker Image Inspect

You can run Docker image inspect command to view detailed information about one or more images:

```
docker image inspect <Docker Image>
```

Labels and Environment Variables

You can view labels and environment variables configured for Docker image by using the `docker inspect` command.

For more information about labels and environment variables for the Docker image, see [Dockerfiles for ActiveMatrix Service Grid - Container Edition](#).

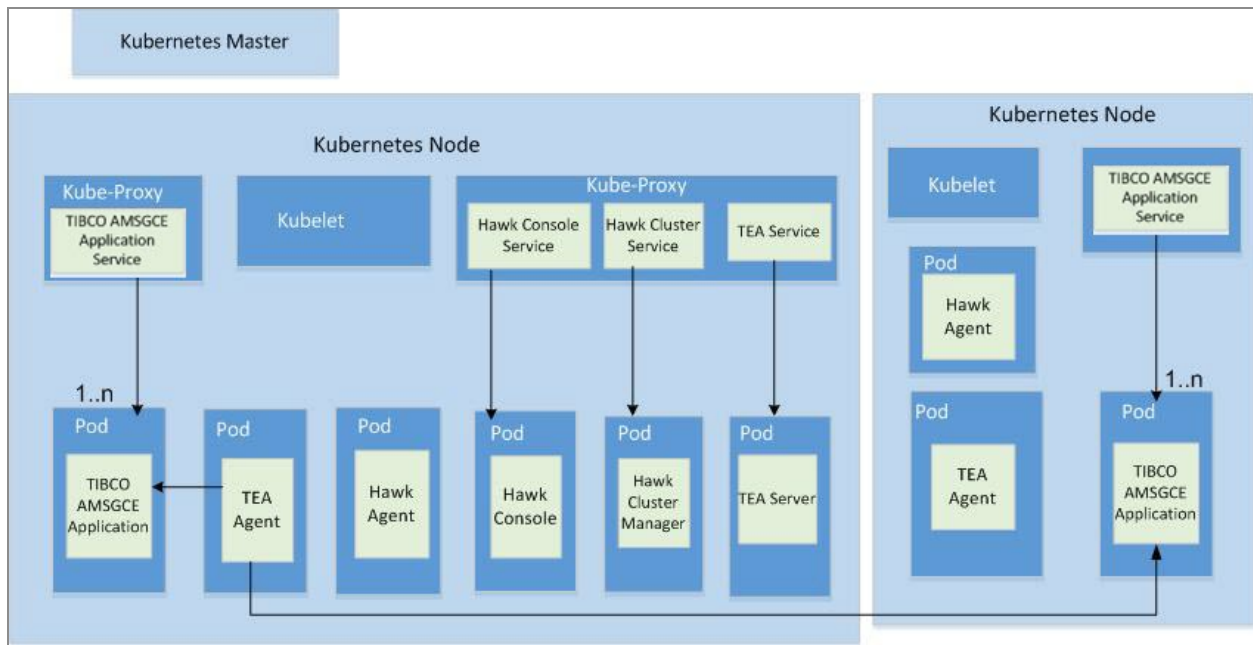
Running ActiveMatrix Service Grid - Container Edition in Kubernetes Cluster

Kubernetes is required to run ActiveMatrix Service Grid - Container Edition. Kubernetes is an orchestration engine for managing containerized applications across multiple hosts providing basic mechanisms for deployment, maintenance, and scaling of applications. ActiveMatrix Service Grid - Container Edition cluster consists of the following containers: ActiveMatrix Service Grid - Container Edition application, Application Monitor TEA agent, TEA Server, and optionally Hawk agent, Hawk Cluster Manager, Hawk Console.

In a Kubernetes cluster, pods communicate with each other by using their IP addresses. Each pod contains a single ActiveMatrix Service Grid - Container Edition application container. Each container runs a single ActiveMatrix Service Grid - Container Edition node. You can expose ActiveMatrix Service Grid - Container Edition applications by using Kubernetes services to communicate with other applications. Inter-component communication is done by using Kubernetes Services.

Application Monitor TEA agent monitors one or more ActiveMatrix Service Grid - Container Edition applications. There is a TEA service that exposes the TIBCO Enterprise Administrator server. The agent communicates with the TIBCO Enterprise Administrator server for UI interactions and communicates with ActiveMatrix Service Grid - Container Edition applications using JMX. To monitor applications using Hawk® Container Edition, deploy Hawk Cluster Manager, Hawk Console, and Hawk agent pods in the same cluster.

The following diagram illustrates how ActiveMatrix Service Grid - Container Edition is deployed in a Kubernetes cluster:



See the following topics for deploying ActiveMatrix Service Grid - Container Edition in a Kubernetes cluster:

- [Running ActiveMatrix Service Grid - Container Edition on Kubernetes](#)

Running ActiveMatrix Service Grid - Container Edition on Kubernetes

There are many ways to set up Kubernetes locally by using different solutions available. For more information about setting up Kubernetes, see [Kubernetes Documentation](#).

You can deploy ActiveMatrix Service Grid - Container Edition components on Kubernetes by using the Readme and sample YAML files at `amsgce-runtime-<version>/samples/`.

The following table lists location of Readme and sample YAML files for running applications and components:

Component	Readme and Sample YAML File Location
Bookstore sample application	<code>amsgce-runtime-<i><version></i>/samples/bookstore</code>

Component	Readme and Sample YAML File Location
Sample ActiveMatrix Service Grid - Container Edition application template file for deployment	amsgce-runtime- <version>/samples/kubernetes/amxce_k8s_ template.yaml
Application Configurator	amsgce-runtime- <version>/samples/kubernetes/Application_ Configurator
Application Monitor	amsgce-runtime- <version> /samples/kubernetes/Application_Monitor
Grafana	amsgce-runtime- <version>/samples/kubernetes/grafana
Hawk® Container Edition Monitoring	amsgce-runtime- <version>/samples/kubernetes/Hawk_ Monitoring
Prometheus	amsgce-runtime- <version>/samples/kubernetes/prometheus

See the following topics for deploying ActiveMatrix Service Grid - Container Edition components to Kubernetes cluster:

- [Deploying Application Configurator to Kubernetes](#)
- [Deploying ActiveMatrix Service Grid - Container Edition Application in Kubernetes](#)
- [Deploying Application Monitor to Kubernetes](#)
- [Deploying TIBCO Hawk Container Edition to Kubernetes](#)

Deploying Application Configurator to Kubernetes

You can deploy the Application Configurator to Kubernetes by using the sample Kubernetes deployment YAML file provided.

Before you begin

- See [Preparing for Containerization](#).
- Ensure that you have created an Application Configurator Docker image. Upload the Docker image to a suitable public or private registry that is accessible from your Kubernetes cluster. For more information, see [Building Application Configurator Docker Image](#).

Procedure

1. Navigate to `amsgce-runtime-<version>/samples/kubernetes/Application_Configurator`. This folder contains sample Kubernetes deployment YAML file that is required to deploy the Application Configurator.
2. Run the `apply` command of the `kubectl` utility to deploy the Application Configurator to Kubernetes.

```
kubectl apply -f <deployment_file>
```

Example:

```
kubectl apply -f amxce_ac_k8s.yaml
```

3. Check Pod status by using the following command:

```
kubectl get pod -n <namespace>
```

Example:

```
kubectl get pod -n app-config-ns
```

4. You can check the logs of container pods by using the following command:

```
kubectl logs <pod> -n <namespace>
```

Example:

```
kubectl logs <pod> -n app-config-ns
```

5. Enter the following URL in browser to access Application Configurator:

`http://<hostname>:31087/appconfig`. Here, 31087 is the default nodePort defined in `amsgce-runtime-<version>/samples/kubernetes/Application_Configurator/amxce_ac_k8s.yaml`.

What to do next

By using the Application Configurator, you can configure an application and generate an application configuration YAML file. The application configuration YAML file is required to build an application Docker image. For more information about generating an application configuration YAML file, see *ActiveMatrix Service Grid - Container Edition Administration*.

Deploying ActiveMatrix Service Grid - Container Edition Application in Kubernetes

You can deploy ActiveMatrix Service Grid - Container Edition application to Kubernetes by using the Kubernetes deployment YAML file.

Before you begin

- See [Preparing for Containerization](#).
- Ensure that you have created a Docker image of an application. Upload the Docker image to a suitable public or private registry that is accessible from your Kubernetes cluster. For more information, see [Containerizing ActiveMatrix Service Grid Applications](#).

Procedure

1. Create a Kubernetes deployment YAML file that is required to deploy an application. Sample YAML file is at `amsgce-runtime-<version>/samples/kubernetes/amxce_k8s_template.yaml`. For more information about YAML file configurations, see [Sample Kubernetes YAML File for Deploying an Application](#).
2. Run the `apply` command of `kubectl` utility to deploy an application to Kubernetes.

```
kubectl apply -f <deployment_file>
```

Example:

```
kubectl apply -f bookstore-deployment.yaml
```

3. Get the external IP of your application, which you can then use to connect to the cluster. Use the external service name defined in the application deployment file.

Syntax:

```
kubectl get services <external_service_name> -n <namespace>
```

Example:

```
kubectl get services bookstore-service -n bookstore-ns
```

4. You can check the logs of container pods by using the following command:

```
kubectl logs <pod> -n <namespace>
```

What to do next

Test the application by using the external IP obtained.

Deploying Application Monitor to Kubernetes

You can deploy the Application Monitor to Kubernetes by using Kubernetes deployment YAML files. In a Kubernetes cluster, Application Monitor agent auto discovers all containers running ActiveMatrix Service Grid - Container Edition application and displays their status on the Application Monitor dashboard.

The Application Monitor needs cluster-wide view permission. So you must create a service account with cluster view permission.

i Note: When deploying in the Kubernetes cluster, Application Monitor agent and TIBCO Enterprise Administrator server must be deployed in the same cluster. For multiple cluster setup, deploy Application Monitor agent and TIBCO Enterprise Administrator server in each cluster separately.

Before you begin

- See [Preparing for Containerization](#).
- Ensure that you have created a Docker image of TEA Server. Upload the Docker

image to a suitable public or private registry that is accessible from your Kubernetes cluster. For more information, see [Building TIBCO Enterprise Administrator Server Docker Image](#)

- Ensure that you have created a Docker image of Application Monitor agent. Upload the Docker image to a suitable public or private registry that is accessible from your Kubernetes cluster. For more information, see [Building Application Monitor Agent Docker Image](#).

Procedure

1. Navigate to `amsgce-runtime-<version>/samples/kubernetes/Application_Monitor`. For more information about deployment YAML file, see [Sample Kubernetes YAML File for Deploying Application Monitor](#).

2. To create a service account with a cluster view role, run the following command:

```
kubectl apply -f amxce_authorization.yaml
```

3. In the `tea-server.yaml` file, update `<tea-server-docker-image>` with a TEA Server Docker image. Run the `apply` command of `kubectl` utility to deploy the TEA Server to Kubernetes.

```
kubectl apply -f tea-server.yaml
```

4. In the `tea-agent.yaml` file, update `<amxceteaagent-image-name>` with Application Monitor agent Docker image. Run the `apply` command of `kubectl` utility to deploy the Application Monitor agent to Kubernetes.

```
kubectl apply -f tea-agent.yaml
```

5. Verify that the deployment has succeeded and that the pod is running. For example:

```
~$ kubectl get pod -n monitoring
```

NAME	READY	STATUS	RESTARTS
AGE			
tea-agent-deployment-6c7f784fb8-pjr6g	2/2	Running	0
11s			
tea-server-deployment-7699c48774-s46vf	1/1	Running	0
2m57s			

6. You can check the logs of container pods by using the following command:

```
kubectl logs <pod> -n monitoring
```

7. You can access the Application Monitor dashboard by using the `http://<hostname>:31877/tea` link.

Configuring Multiple TEA Agents

You can configure more than one TEA agent to monitor different namespaces and deploy them to Kubernetes cluster.

To configure multiple TEA agents, there is sample deployment file available `<amsgce-runtime-1.0.0>/samples/kubernetes/Application_Monitor/multiagent/application_monitor_2.yaml`.

In this file, you need to update the namespace that agent must monitor by using the following environment variable:

```
name: amxce_namespace_to_monitor
value: namespace2
```

If you do not provide any namespace to monitor, it will connect to all ActiveMatrix Service Grid pods running in the cluster.

The namespaces in each tea-agent deployment file must be different.

i Note: You must not create different service accounts specific to each namespace. If you are using multiple agents then use the same service account in all agents.

Deploying TIBCO Hawk Container Edition to Kubernetes

Hawk Container Edition can be deployed on Kubernetes by using the deployment configuration files (YAML format), which contain the configuration details for deployment including environment variables.

Before you begin

- See [Preparing for Containerization](#).

- Ensure that you have created docker images of Hawk agent, Hawk Cluster Manager, and Hawk Console. Upload the Docker images to a suitable public or private registry that is accessible from your Kubernetes cluster. For more information, see [TIBCO Hawk Container Edition Documentation](#).

Procedure

1. Navigate to `amsgce-runtime-<version>/samples/kubernetes/Hawk_Monitoring`.
2. In the `hawk.yaml` file, update Docker images of Hawk agent, Hawk Cluster Manager, and Hawk Console. For more information about the Kubernetes deployment YAML files configurations of Hawk Container Edition components, see [Sample Kubernetes YAML File for Deploying Hawk Container Edition](#).
3. Run the `apply` command of `kubectl` utility to deploy Hawk Container Edition to Kubernetes.

```
kubectl apply -f hawk.yaml
```

4. You can check the logs of container pods by using the following command:

```
kubectl logs <pod> -n <namespace>
```

5. You can also get the external IP to the external service of the cluster by using the `get services` command. You can then use that IP to connect to the cluster.

```
kubectl get services hkce-console-service -n <namespace>
```

What to do next

If you have a Hawk console container running, you can access it at `http://<Console_host_IP>:<Host_port>/HawkConsole`.

For example: `http://<host>:31083/HawkConsole`. Where 31083 is default NodePort specified in the sample `hawk.yaml` file. You can change it to any valid NodePort.

For more information about Monitoring ActiveMatrix Service Grid - Container Edition, see *ActiveMatrix Service Grid - Container Edition Monitoring*.

HTTP Endpoints for Health Checks

When the ActiveMatrix Service Grid - Container Edition application is started, it exposes REST API to show component status and node health on container port 9998. This endpoint can be used to configure health checks on Docker or Kubernetes-based platforms.



Note: Port 9998 is not configurable.

Checking Application Health in Docker

While running Docker container, you must bind the user defined host port with container's port 9998. After the host port is bound, start the container. You can access the following endpoint URLs to check health of an application:

- `http://<host>:9998/status/healthz:`
 - Returns "RUNNING" with HTTP 200 status code, if all the components are in running state.
 - If all the components are not in running state, then it returns HTTP 503 with the list of components that are not in the running state.

This can be used as Kubernetes liveness probe.
- `http://<host>:9998/status/errors:` Returns list of components that are not in the running state.
- `http://<host>:9998/status/swagger.json:` Returns swagger JSON specification.
- `http://<host>:9998/status/nodehealth:` Returns status of all components running on the node.
- `http://<host>:9998/status/apphealth:` Returns status of all user application components.

Configuring Probes in Kubernetes

In Kubernetes, health checks are performed by configuring liveness probes or readiness probes. The liveness probes indicate when to restart a container and the readiness probes are used to know when a container can start accepting traffic. The HTTP endpoint in ActiveMatrix Service Grid - Container Edition application is used to configure readiness probes in Kubernetes. The endpoint must be accessed only after the application is started.

Setting Up Liveness Probe

To view the liveness of an application, you must add the liveness probe in the manifest file. For more information, see "Configure Liveness and Readiness Probes" in Kubernetes documentation.

```
livenessProbe:
  httpGet:
    path: /status/healthz
    port: 9998
```

Setting Up Readiness Probe

To view the readiness of an application, you must add the readiness probe in the manifest file. For more information, see "Configure Liveness and Readiness Probes" in the Kubernetes documentation.

```
readinessProbe:
  httpGet:
    path: /status/healthz
    port: liveness-port
```

Using ConfigMaps with ActiveMatrix Service Grid - Container Edition Applications

ConfigMap stores configuration data for containers. ConfigMap separates out configurations from your Pods and components. It is easier to change and manage ConfigMaps, without hardcoding configuration data to Pod specifications. You can use ConfigMaps to pass substitution variables to applications.

Procedure

1. Create a ConfigMap in Kubernetes.

Sample manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```

name: airport-config
namespace: airport
data:
  svar.app.TEMP_IN_UNIT: C

```

2. Run the following command:

```
kubectl create configmap <name-of-configmap-file-to-be-created>
```

It creates a ConfigMap with the name "airport-config" and has one application level substitution variable: svar.app.TEMP_IN_UNIT: C

3. Use this ConfigMap in the application deployment file as environment variable:

```

envFrom:
  - configMapRef:
      name: <name-of-the-configmap>
      optional: true

```

Using Kubernetes Secrets

Kubernetes secret object lets you store and manage sensitive information like passwords or keys. This section explains how a secret can be used with ActiveMatrix Service Grid - Container Edition application. For more information on Kubernetes secrets, refer to the Kubernetes documentation.

Procedure

1. Create a secret using Kubernetes manifest file, here is the sample manifest file:

```

apiVersion: v1
data:
  username: dXNlcjEK
  password: cGFzcjEK
kind: Secret
metadata:
  name: mysecret
  namespace: airport
type: Opaque

```

Here fields username and password have values base64 encoded.

2. Create Secret by using the following command:

```
kubectl apply -f <manifest_file>.yaml
```

3. Use this value in the ActiveMatrix Service Grid - Container Edition application deployment file as environment variable.

```
- name: sr.airPortCodeDB.password
  valueFrom:
    secretKeyRef:
      name: mysecret
      key: password
- name: sr.airPortCodeDB.username
  valueFrom:
    secretKeyRef:
      name: mysecret
      key: username
```

Here airPortCodeDB is the shared resource name.

Sample YAML Files for Kubernetes Cluster

All YAML files are located at `amsgce-runtime-<version>/samples/kubernetes/`. The YAML files define the Kubernetes objects that are required for deployment. You can update YAML files and deploy objects to the cluster to change configuration. Use YAML files to configure Kubernetes resources such as pods, services, and deployments. For more information about resources configured in the YAML files, see [Kubernetes documentation](#).

- [Sample Kubernetes YAML File for Deploying an Application](#)
- [Sample Kubernetes YAML File for Deploying Application Monitor](#)
- [Sample Kubernetes YAML File for Deploying Hawk Container Edition](#)

Sample Kubernetes YAML File for Deploying an Application

Sample Kubernetes YAML file for deploying an application is at `amsgce-runtime-<version>/samples/kubernetes/amxce_k8s_template.yaml`.

In the YAML file, update the application name (*<app-name>*), application Docker image name (*<image-name>*), and other placeholders as per your application.

In this file,

- `containerPort: 55560` is used by Application Monitor to connect to ActiveMatrix Service Grid - Container Edition nodes running in the Kubernetes cluster. Port 55560 is the default JMX port and it is not configurable.
- You must add `tibco.amxce: application` label for all ActiveMatrix Service Grid - Container Edition applications. Application Monitor checks for label "tibco.amxce: application" to discover ActiveMatrix Service Grid - Container Edition pods.

i Note: All applications which are required to be linked with the Application Monitor must specify this label.

- Port 9998 is used by Monitoring app. This port is not configurable.
- Port 2555 is used by Hawk Microagent. This port is not configurable.

To enable monitoring by using Hawk Container Edition, uncomment the following environment variables from sample YAML file :

- `tcp_self_url`
- `agent_ami_session_url`
- `microagent_name_suffix` (optional)

i Note: In ActiveMatrix Service Grid - Container Edition, instead of running one TIBCO AMSGCE node with more hardware (CPU, memory) capacity, TIBCO recommends to use horizontal scaling and use load balancer with it. For example, instead of running one node with 16 CPU cores and 32 GB RAM, run 4 TIBCO AMSGCE containers, each having 4 CPUs and use load balancer with it.

Kubernetes Rolling Update

To achieve high availability of applications, you can perform a rolling update. Rolling update is a feature of Kubernetes in which you can update deployments with zero downtime. Pods instances are incrementally replaced with new ones. New pods instances are then scheduled on nodes with available resources. For more information about rolling updates, see Kubernetes documentation.

Using Kubernetes Services to Access an Application in a Cluster

In Kubernetes, TIBCO ActiveMatrix applications run as pods. Pods communicate with each other by using services. Therefore, you must use Kubernetes services to connect two dependent applications.

For example, consider Hello World2 application provided at `<TIBCO_HOME>/amx/<version>/samples/java/helloworld2.zip`. For more information about this sample Hello World2 application, see *TIBCO ActiveMatrix Service Grid documentation*.

In Kubernetes to run this application, you must set up a service for connecting to Date Manager application. This service must be used as host in the HTTP client resource of the helloworld2 application configuration YAML file. helloworld2 application uses this service to connect to datemanager application. Both helloworld2 and datemanager applications must run in the same namespace.

Sample helloworld2 application configuration YAML file:

```
resources:
  httpClients:
    - name: HttpClient_DateManagerSOAP
      description: This is updated using Application Configurator (1.0.0)
      host: datemanager-service
      port: '9097'
```

Sample Kubernetes deployment file for Date Manager application:

```
apiVersion: v1
kind: Service
metadata:
  name: datemanager-service
  namespace: enhanced-helloworld
spec:
  type: LoadBalancer
  externalIPs:
    - 10.107.174.117
  ports:
    - port: 9097
      targetPort: 9097
      name: appport
    - port: 9998
      targetPort: 9998
      name: monitoringapplicationport
  selector:
    app: datemanager
```

Sample Kubernetes YAML File for Deploying Application Monitor

Sample Kubernetes YAML files for deploying Application Monitor are at `amsgce-runtime-<version>/samples/kubernetes/Application_Monitor`.

Update the following Kubernetes object specification (.yaml) files for Application Monitor:

- `amxce_authorization.yaml` - A ClusterRoleBinding for binding roles to the user.
- `tea-agent.yaml` - Kubernetes deployment YAML file to deploy Application Monitor agent.
- `tea-server.yaml` - Kubernetes deployment YAML file to deploy TEA Server.

In the sample YAML files, specify the image tag of TEA Server Docker image (`<tea-server-docker-image>`) and Application Monitor agent (`<amxceteaagent-image-name>`).

Environment Variables

You can use the following environment variables to configure Application Monitor:

Environment Variable	Description
<code>tea_server_url</code> (Required)	<p>URL of a TIBCO Enterprise Administrator Server running in the cluster.</p> <p>This is required to connect and autoregister Application Monitor agent with TEA Server.</p> <p>Example: <code>http://tea-service:8777/tea</code></p>
<code>amxce_tea_agent_port</code> (Optional)	<p>Application Monitor agent port.</p> <p>Default: 7073</p>
<code>amxce_tea_agent_name</code>	<p>Application Monitor agent display name. The name is displayed in the Application Monitor UI.</p>
<code>amxce_swaggerui</code> (Optional)	<p>The externally hosted Swagger UI URL.</p>

Environment Variable	Description
	If specified, then in the Application Monitor, you can open REST service in the Swagger UI by clicking the link provided in the Endpoints tab.
amxce_tea_agent_shutdown_port (Optional)	This is used to set shutdown port of TIBCO Enterprise Administrator server.
tibco.amxce.reload.interval.ms	<p>Time interval in which Application Monitor synchronizes with Kubernetes API server and checks ActiveMatrix Service Grid - Container Edition nodes for active connection. It runs in scheduled thread and thread name starts with "amxce-reload-"</p> <p>Default: 300000 ms (5 min)</p> <p>Note: For small scale setup, set this parameter to lower value.</p>
tibco.amxce.agent.node.connect.max.thread	<p>This property specifies the number of running threads when Application Monitor connects to AMXCE Nodes. The thread name starts with "amxce-hpa-".</p> <p>Default: 5</p>
tibco.amxce.agent.node.connect.timeout.ms	<p>This is the maximum time interval for which Application Monitor waits for creating connection to AMXCE Node. After connection timeout, the Node is displayed as unreachable. Application Monitor retries to connect to unreachable Node after time interval value specified in the tibco.amxce.reload.interval.ms.</p> <p>Default: 5000 ms (5 seconds)</p>

i Note: Communication between Application Monitor agent and ActiveMatrix Service Grid - Container Edition application container is done by using JMX. Communication between Application Monitor agent and application pods with SSL enabled is not supported in ActiveMatrix Service Grid - Container Edition 1.0.0.

If you are using ClusterRole resource, you must provide "replicasets" access to the ClusterRole. This is required to get the deployment name and desired state of the pod. For more information about using ClusterRole for authorization, see Kubernetes documentation.

Example:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: amxcecluster-role
rules:
- apiGroups: [ "", "extensions", "apps" ]
  resources: ["pods*", "replicasets*"]
  verbs: ["get", "list", "watch"]
```

Sample Kubernetes YAML File for Deploying Hawk Container Edition

Sample Kubernetes YAML file for deploying Hawk Container Edition is provided in the folder `amsgce-runtime-<version>/samples/kubernetes/Hawk_Monitoring/hawk.yaml`.

The `hawk.yaml` file contains deployment configuration for Hawk Cluster Manager, Hawk agent and Hawk Console.

In the `hawk.yaml` file, set up the container (the `containers` tag) with the Docker image (`image`) of the Hawk Cluster Manager, Hawk agent, and Hawk Console.

Environment Variables

You can use the following environment variables to configure the Kubernetes YAML file with the Hawk agent container:

Environment Variable	Required?	Description
<i>tcp_self_url</i>	Yes	<p>The <i>tcp_self_url</i> environment variable.</p> <p>It specifies the self URL for the TCP Transport for Hawk® Container Edition. The URL is in the form:</p> <p><SELF_IP_ADDRESS>:<PORT>.</p>
<i>agent_ami_session_url</i>	Yes	<p>Specifies the URL used by the microagent to connect to Hawk agent container by using Hawk TCP AMI session.</p>
<i>microagent_name_suffix</i>	No	<p>Specifies the suffix for the microagent name. The default value is used if no value is specified for the variable.</p> <p>Default: Default microagent name is com.tibco.hawk.amxce.<AMXCENode name></p> <p>With suffix it is: com.tibco.hawk.amxce.<suffix ></p> <p>This suffix gets appended to microagent's display name as well.</p>

Troubleshooting

If you encounter problems with installation, make sure that your system meets all the prerequisites. Next, check the logs for potential problems.

Problem

Docker image build fails while downloading base image.

Troubleshooting tips

Ensure that you can pull the Docker image.

Pull the image by using `docker pull` command and then try building the image.

Example:

For UBI7 Docker image, you can manually pull the Docker image from the Docker registry by using following command:

```
docker pull registry.access.redhat.com/ubi7/ubi:latest
```

Problem

When building the application Docker image the following error occurs:

```
failed to solve with frontend dockerfile.v0: failed to build LLB: executor failed running [/bin/sh -c ./config.sh]: exit code: 1
```

Troubleshooting tips

This error comes from Docker by default when any build process returns non-zero code. Ensure that application configuration YAML file is valid. Use the `checkConfig` script in the `amsgce-runtime-<version>/dautil/bin` folder to validate the configuration. For more information about validating configuration YAML file, see *TIBCO ActiveMatrix® Service Grid - Container Edition Administration*.

Problem

Pod console logs do not print timestamp at start of each line.

Troubleshooting tips

To print the timestamp in logs, add the following lines in the console appender logging configuration:

```
%d{dd MMM yyyy HH:mm:ss,SSS}
```

Example:

```
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
  <param name="Target" value="System.out"/>
  <param name="Threshold" value="TRACE"/>
  <layout class="com.tibco.tpcl.org.apache.log4j.PatternLayoutEx">
    <param name="ConversionPattern" value="%d{dd MMM yyyy HH:mm:ss,SSS}%t]
    [%-5p] %c - %m%n"/>
  </layout>
</appender>
```

Alternatively, for kubectl, use timestamps flag to print timestamps in logs:

```
kubectl logs <POD> -n <Namespace_Name> --timestamps
```

Problem

The following error occurs in the Application Configurator container logs when running Application Configurator in Docker. You can also view this error when running a TIBCO ActiveMatrix application in Docker:

```
AMXCENode [Start Level Event Dispatcher] [ERROR] com.tibco.hkce.konex -
Failed to initialize HAWK agent, tcp_self_url and/or ami_agent_url not
specified
```

Troubleshooting tips

This error is because the Hawk microagent needs Hawk agent details to connect. When running the Application Configurator, you do not need to run TIBCO Hawk - Container Edition. Therefore you can ignore this error. But if you are running TIBCO ActiveMatrix application in Docker and want to run TIBCO Hawk - Container Edition with it, you must pass the agent_ami_session_url and tcp_self_url environment variables. For more information about these variables, refer *TIBCO Hawk - Container Edition documentation*.

For example:

```
docker run --rm -d -p 7777:7777 --name bookstore -e agent_ami_session_
url=hawkagent:2571 -e tcp_self_url=<hostname>:2551 --network amxce-net
bookstore
```

Viewing Hostname Information for the Running Java Processes

You can view the hostname for the running java processes in the JVM arguments. Using the hostname, you can identify a container, where the process is running. To view the hostname for the running java processes, run the following command:

```
ps -eaf | grep java
```

JVM arguments contain the hostname, as highlighted in the following sample command output:

```
1001280+ 30842 30836 2 Oct13 ? 00:22:29 java -DTIBCO_
HOME=/opt/tibco/tibco.home -DSCRIPT_VERSION=3 -DHOSTNAME=amxce-
bookstore-deployment-547748b7b5-ltjzt -Dcom.tibco.amf.hpa.tibcohost
```

Appendix

The appendix lists steps to generate JDBC driver DAA in TIBCO ActiveMatrix Business Studio.

Generating JDBC Driver DAA

This topic explains the steps to generate JDBC driver DAA that can be used in ActiveMatrix Service Grid 3.x or in ActiveMatrix Service Grid - Container Edition as JDBC driver custom feature.

Need of JDBC Driver Custom Feature

If your SOA Application needs to connect to database, you need to use JDBC Driver. It is best practice to not package JDBC Driver in your application instead keep it outside of your application. In ActiveMatrix Service Grid 3.x, this JDBC driver is provisioned to each Node that needs this JDBC driver and multiple applications can share that common JDBC driver. In ActiveMatrix® Service Grid - Container Edition, each application is packaged in its own container image. If your application needs JDBC driver then you need to provide JDBC driver in the form of DAA (custom feature DAA) . In this topic we will see how you can create JDBC driver custom feature .

In ActiveMatrix Service Grid - Container Edition, you can also add JDBC Driver Custom feature (DAA) to base image. So in future when developer uses this base image to package actual application, they don't need to provide JDBC driver each time. This way administrator can implement better governance.

Before you begin

You will need:

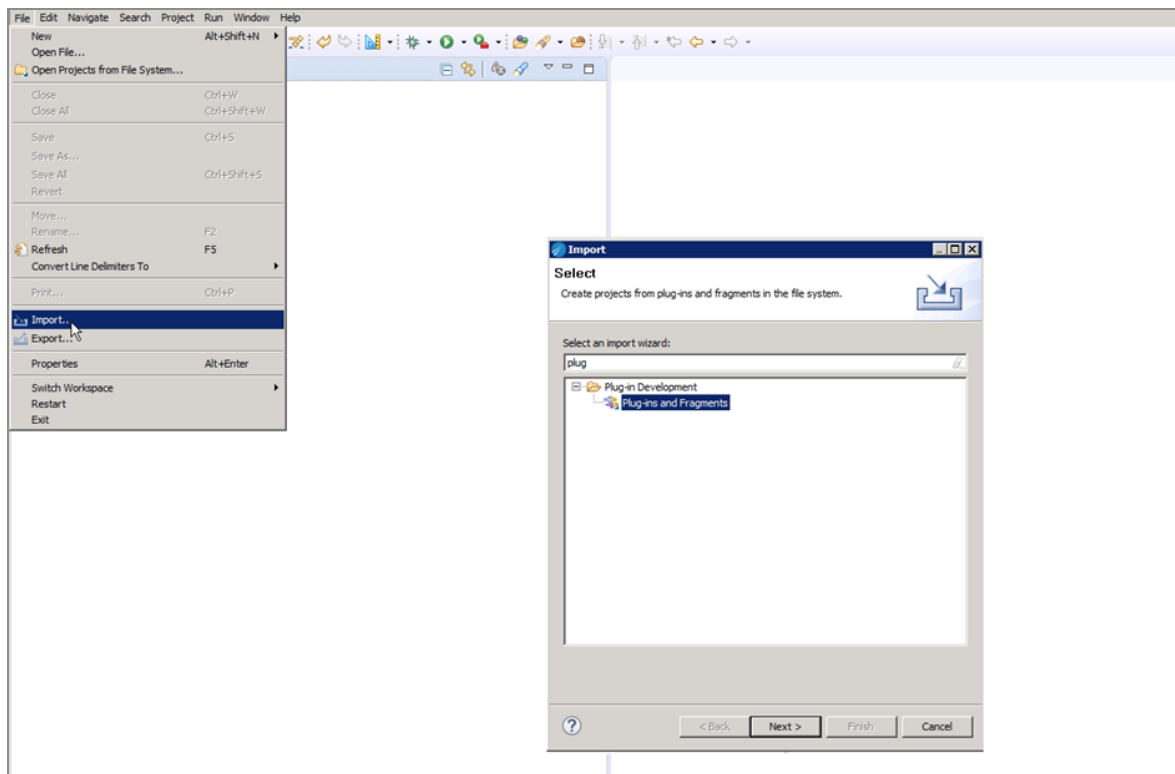
- TIBCO Business Studio bundled with TIBCO ActiveMatrix Service Grid 3.3 or 3.4.
- JDBC Driver jar file(s) downloaded from vendor

In these steps, you are going to wrap JDBC OSGI plug-in (that is generated by using TCT) as custom feature. Once you have custom feature, you can easily export that as DAA.

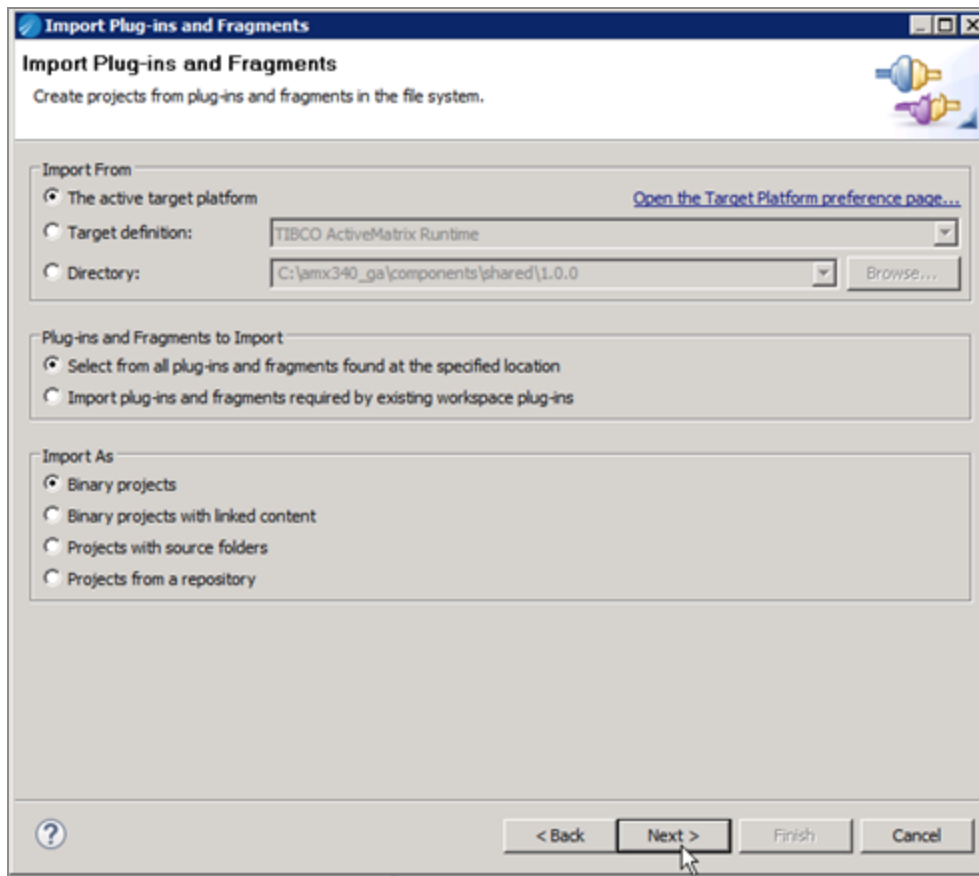
In this example, you will see how you can generate Oracle JDBC driver (v12.2.0.001) custom feature DAA.

Procedure

1. By using TCT wizard enable JDBC driver to TIBCO_HOME. Ensure that your TIBCO ActiveMatrix Business Studio is also using the same TIBCO_HOME. For more information about how to enable JDBC Driver using TCT, see ActiveMatrix Service Grid documentation.
2. Start (or restart) TIBCO ActiveMatrix Business Studio. If possible use new or empty workspace.
3. From **File** menu, select **Import** and then select **Plug-ins and Fragments**.

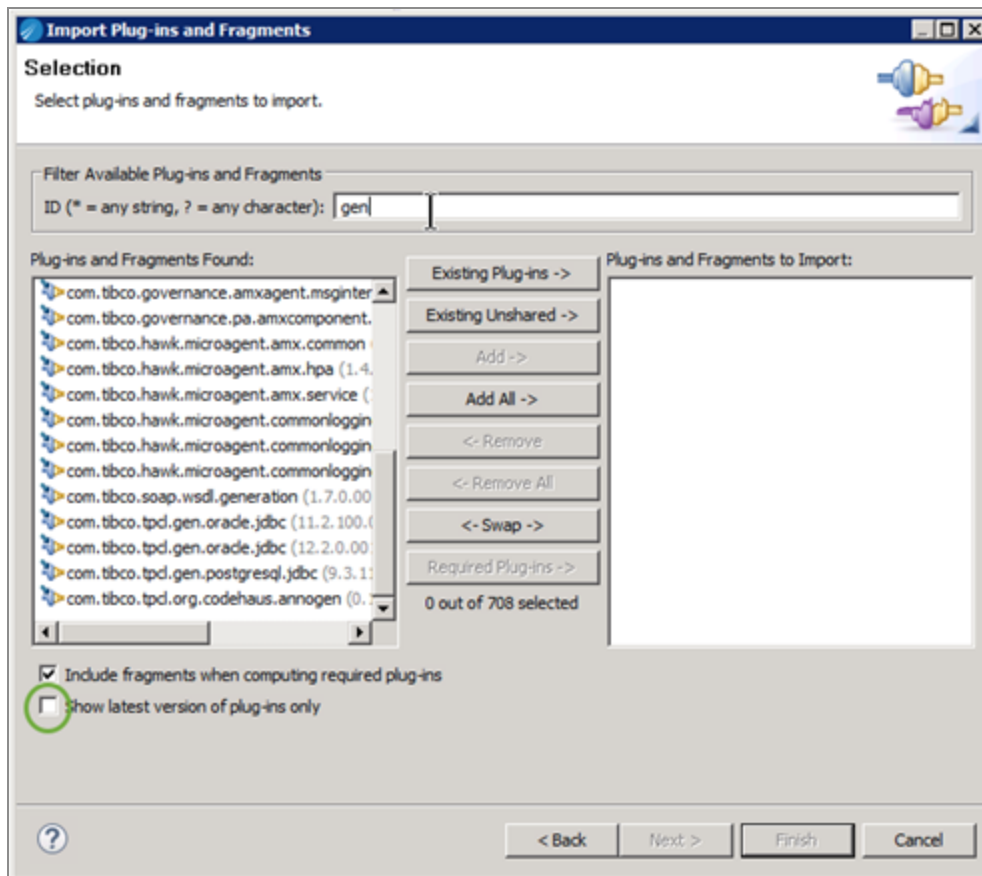


4. Select default options and click **Next**.

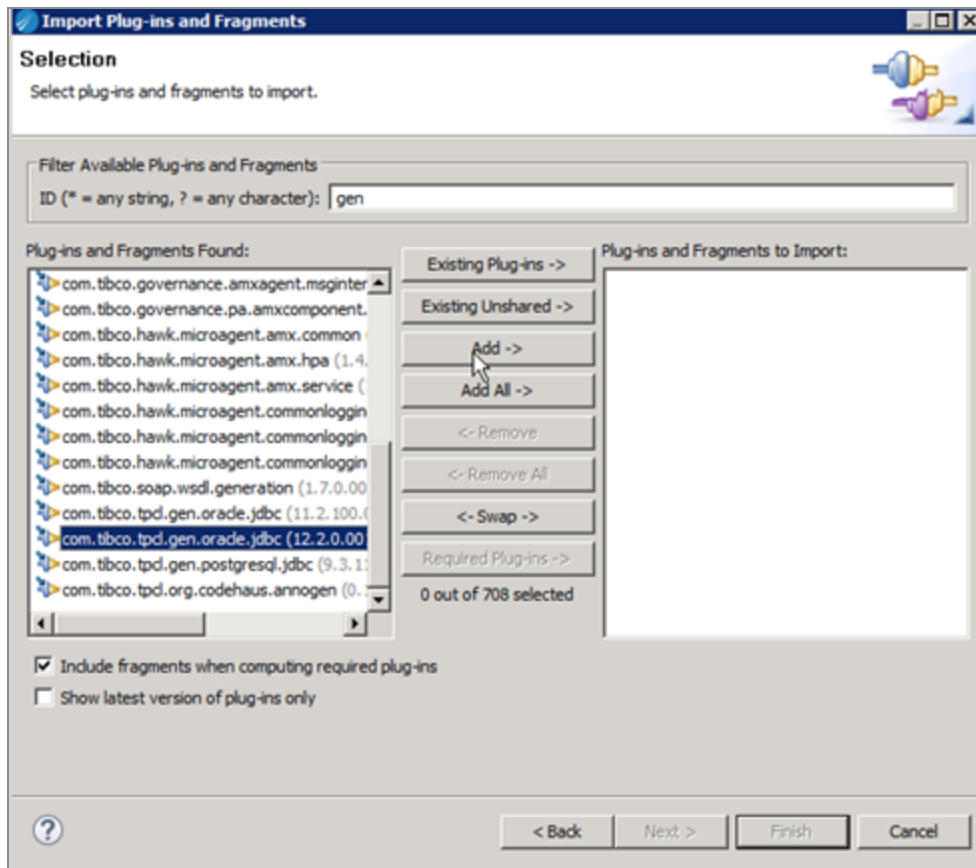


5. In the search box for **ID**, type "gen" and press **Enter**.

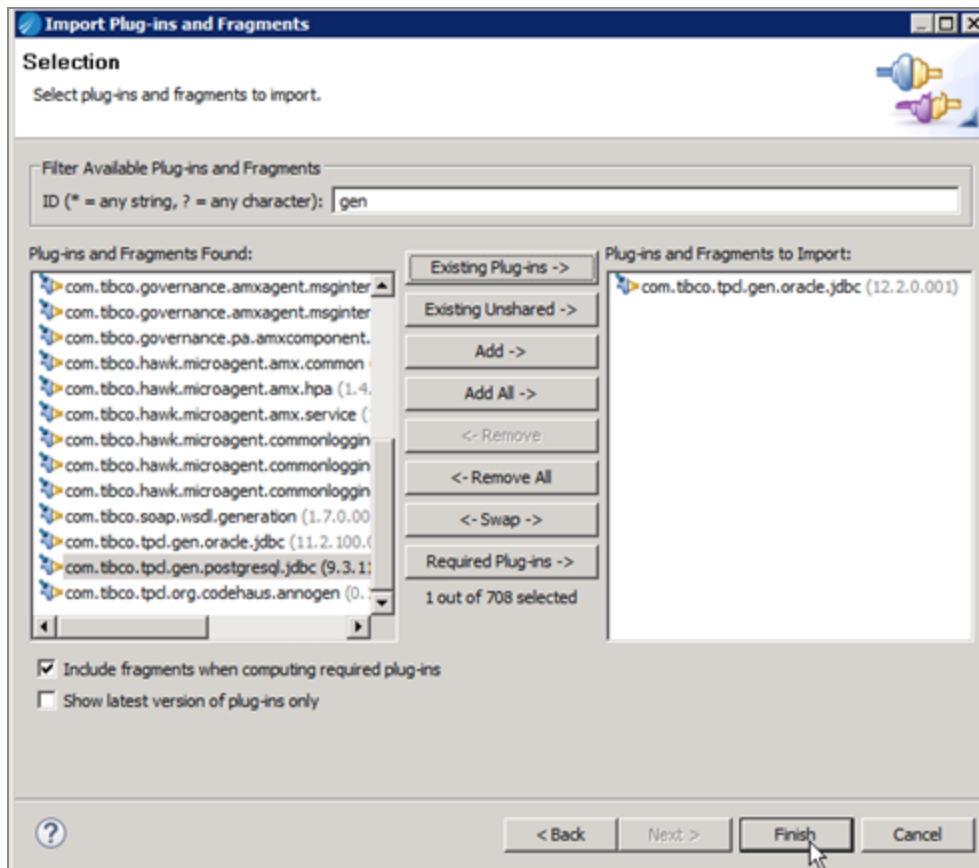
If you have multiple JDBC Drivers versions enabled then ensure that you clear the check box **Show latest version of plug-in only**.



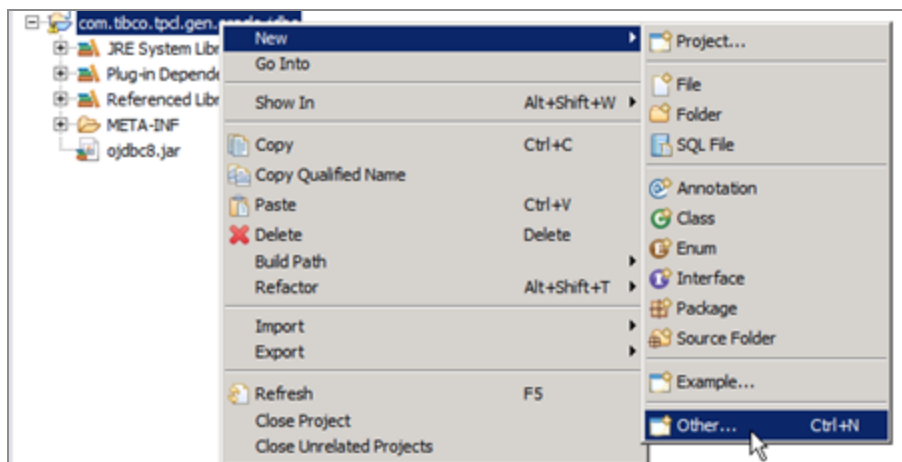
6. Select driver that you need and click **Add** to move that to right-hand side.



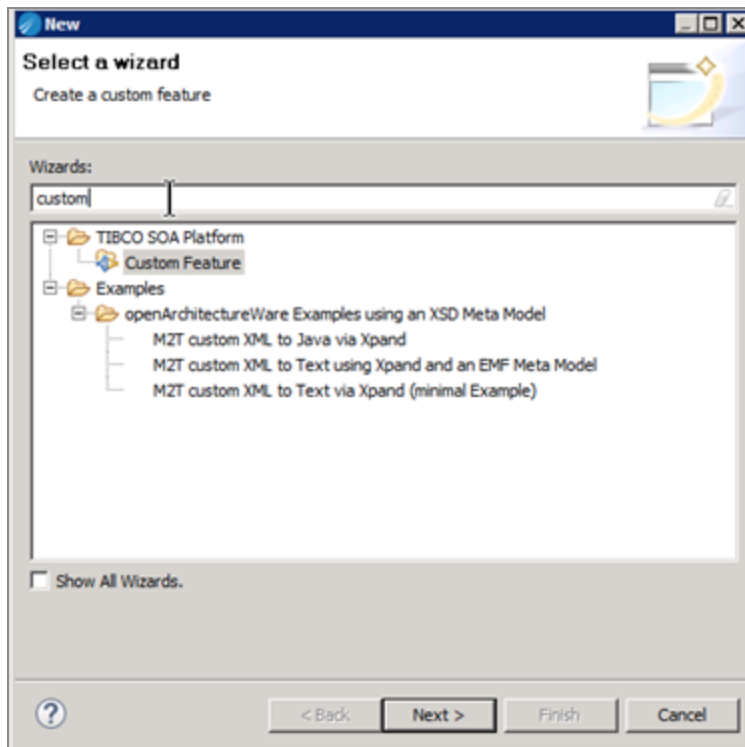
7. Click **Finish**.



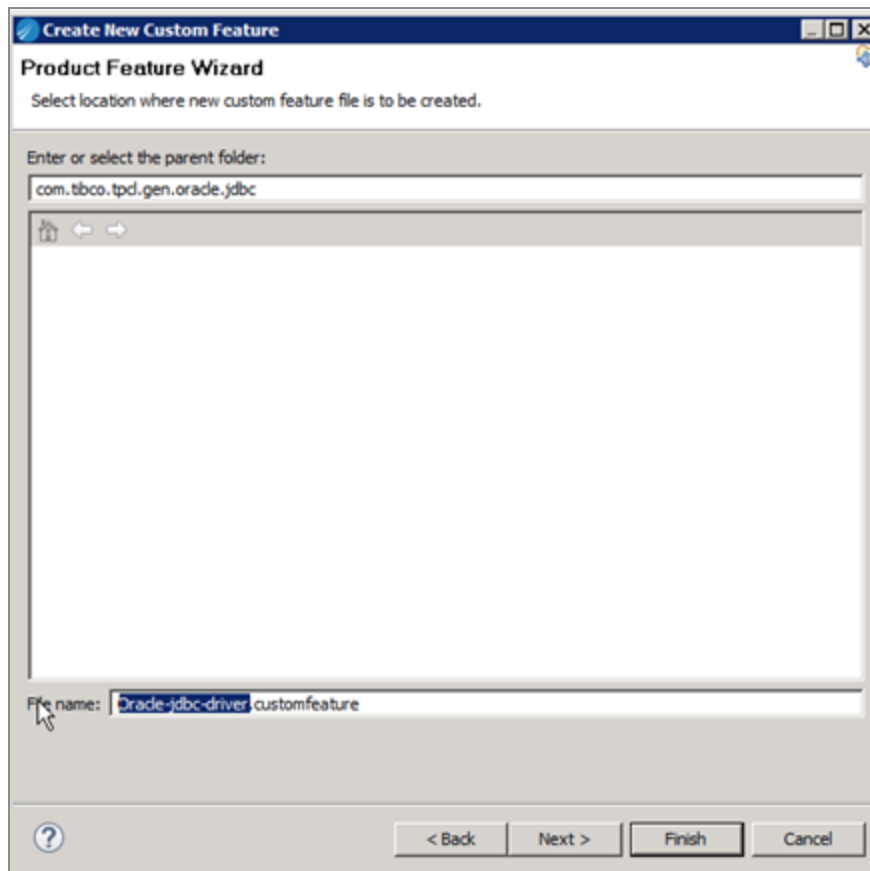
8. Now again from the **File** menu select **New > Other**.



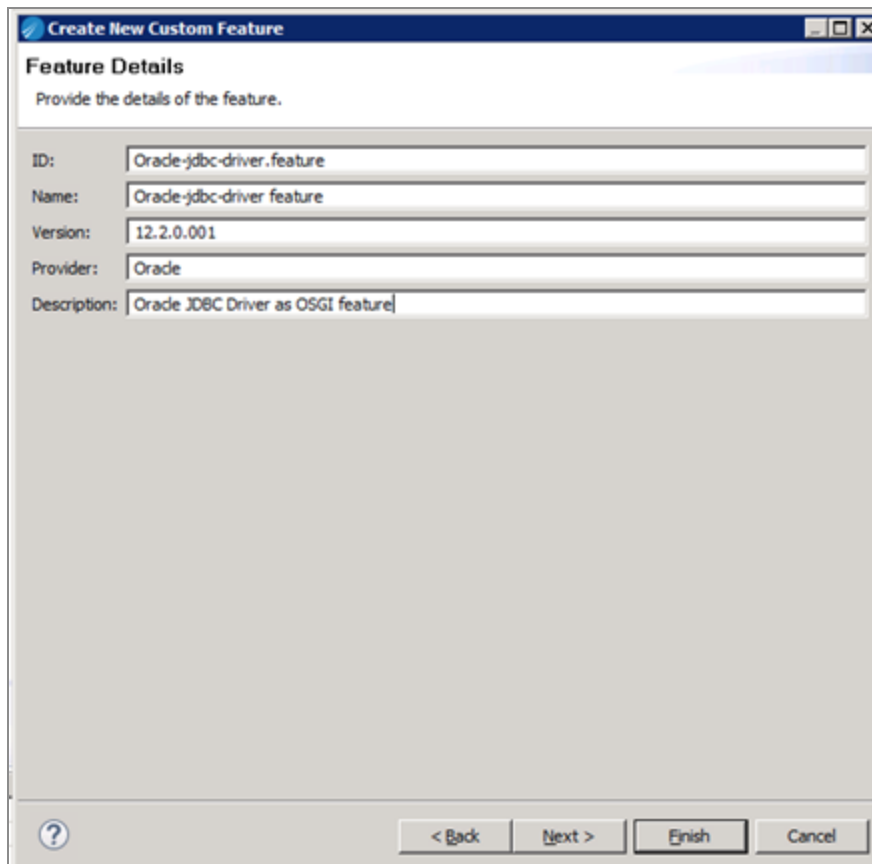
9. Search for "custom".



10. Specify the name for the Custom feature.



11. Specify "Feature Details" and click **Next**.

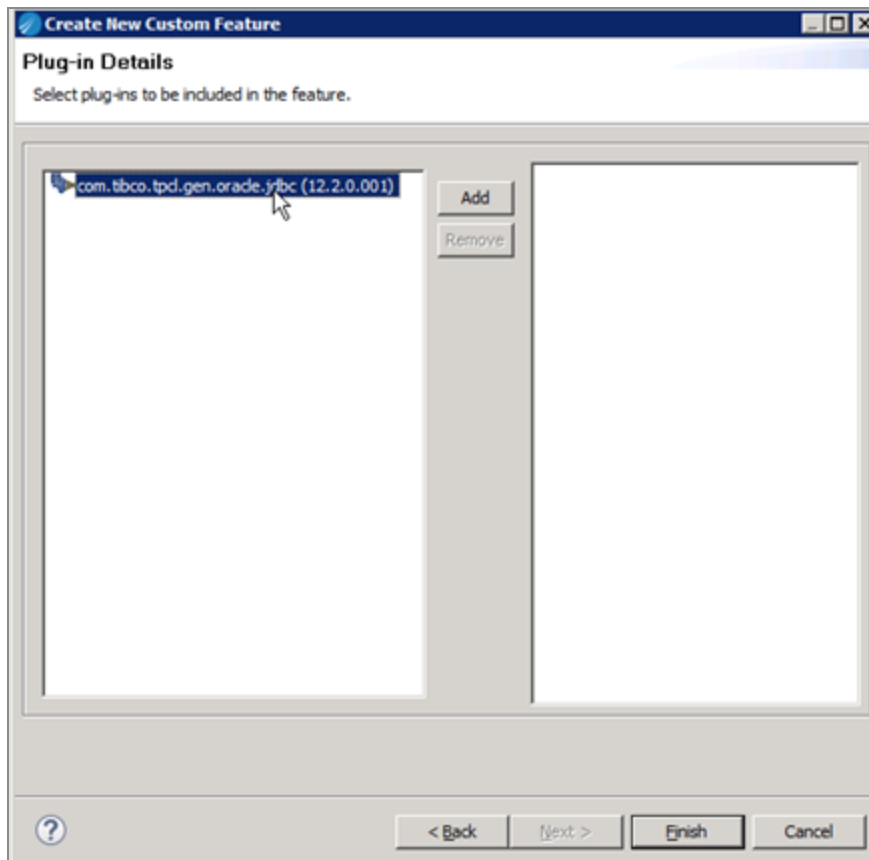


The screenshot shows a Windows-style dialog box titled "Create New Custom Feature". Below the title bar is a section labeled "Feature Details" with the instruction "Provide the details of the feature." Below this are five text input fields:

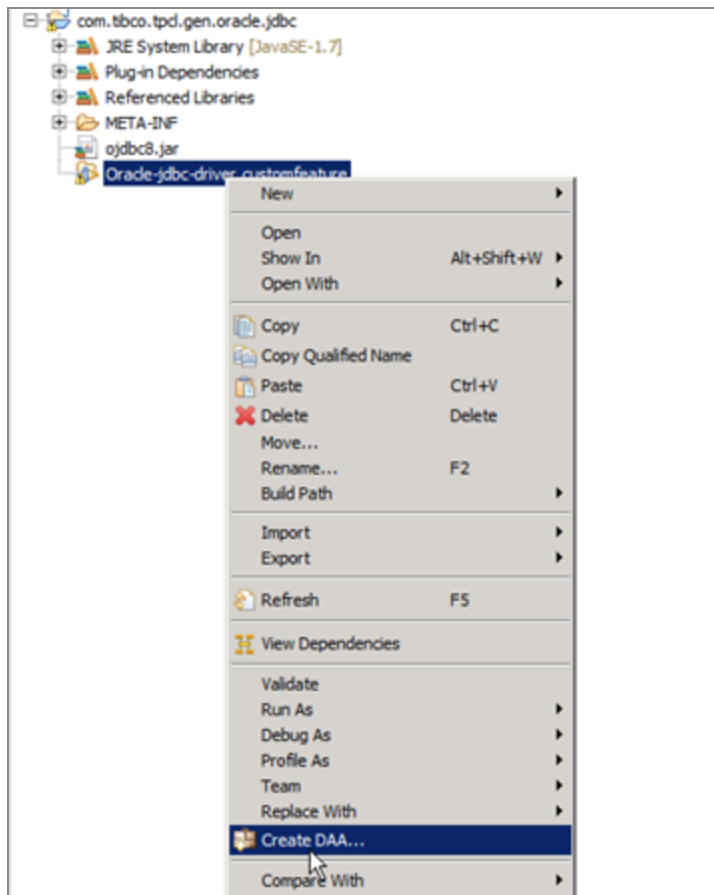
- ID: Orade-jdbc-driver.feature
- Name: Orade-jdbc-driver feature
- Version: 12.2.0.001
- Provider: Oracle
- Description: Oracle JDBC Driver as OSGI feature

At the bottom of the dialog, there is a help icon (question mark in a circle) on the left and four buttons on the right: "< Back", "Next >", "Finish", and "Cancel". The "Finish" button is highlighted with a darker border.

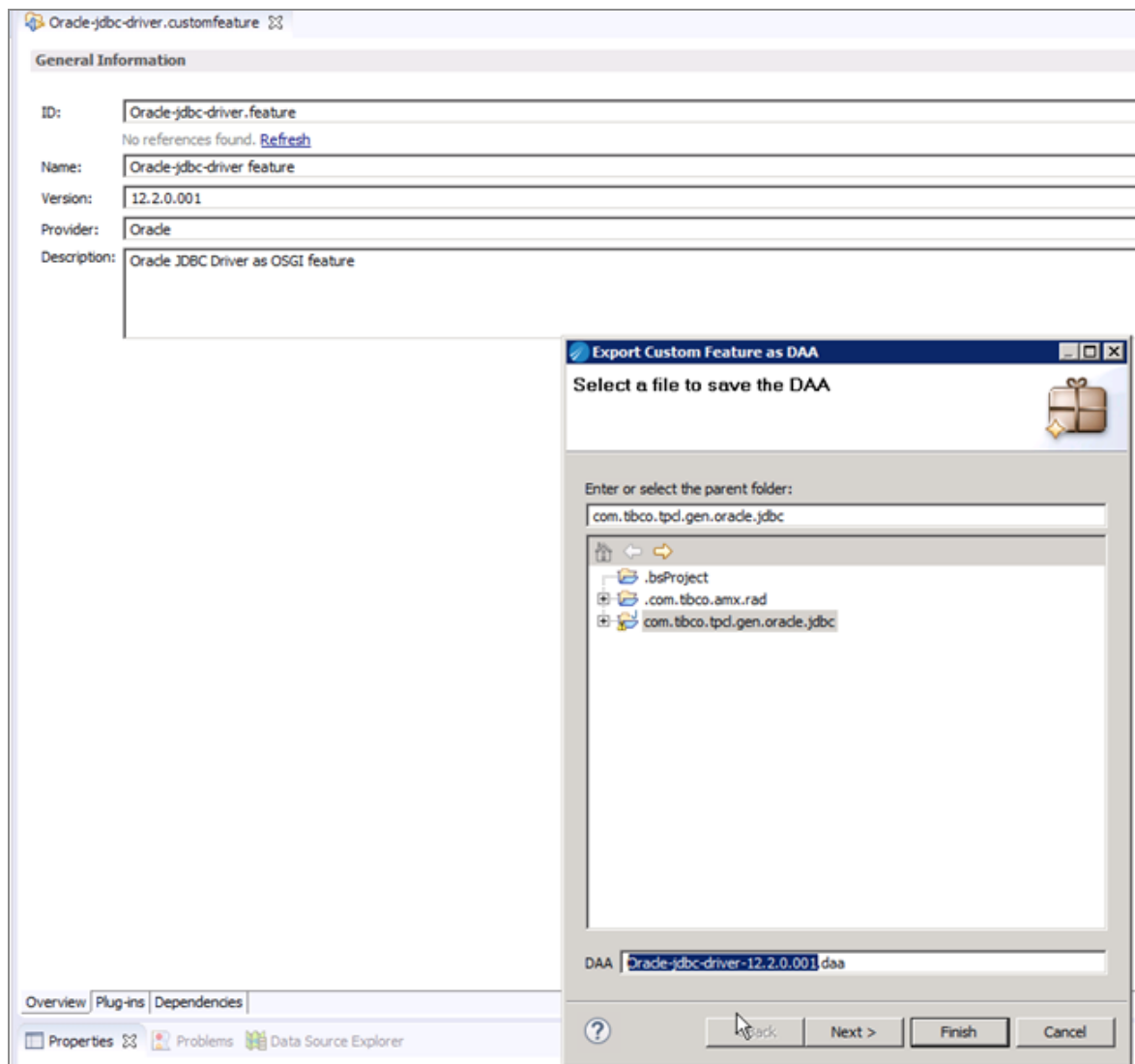
12. Select JDBC Driver that you imported before and click **Finish**.



13. Right-click the custom feature that you just created and select the option **Create DAA**.



14. Follow the standard DAA creation wizard. After completion of the wizard, you will have the DAA that contains your JDBC Driver as Custom feature that you can use during ActiveMatrix Service Grid - Container Edition container Image building.



What to do next

You can use the generated DAA when building application Docker image or when building base image.

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

Product-Specific Documentation

The following documentation for TIBCO ActiveMatrix® Service Grid - Container Edition is available on the [TIBCO ActiveMatrix® Service Grid - Container Edition Product Documentation](#) page:

- *TIBCO ActiveMatrix® Service Grid - Container Edition Release Notes*
- *TIBCO ActiveMatrix® Service Grid - Container Edition Cloud Deployment*
- *TIBCO ActiveMatrix® Service Grid - Container Edition Quick Start*
- *TIBCO ActiveMatrix® Service Grid - Container Edition Administration*
- *TIBCO ActiveMatrix® Service Grid - Container Edition Monitoring*

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to <https://support.tibco.com>. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to <https://community.tibco.com>.

Legal and Third-Party Notices

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix, Business Studio, TIBCO Business Studio, Enterprise Message Service, and Hawk are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2020-2021. TIBCO Software Inc. All Rights Reserved.