TIBCO BusinessEvents®

Cloud Deployment Guide

Software Release 5.6.1
November 2019

Important Information

Contents

Figures

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

Documentation for TIBCO BusinessEvents® Enterprise Edition is available on the TIBCO BusinessEvents® Enterprise Edition Product Documentation page.

To directly access documentation for this product, double-click the following file:

`TIBCO_HOME/release_notes/TIB_businessevents-enterprise_5.6.1_docinfo.html` where *TIBCO_HOME* is the top-level directory in which TIBCO products are installed. On Windows, the default *TIBCO_HOME* is `C:\tibco`. On UNIX systems, the default *TIBCO_HOME* is `/opt/tibco`.

The following documents for this product can be found in the TIBCO Documentation site:

- *TIBCO BusinessEvents® Release Notes*
- *TIBCO BusinessEvents® Installation*
- *TIBCO BusinessEvents® Getting Started*
- *TIBCO BusinessEvents® Architect's Guide*
- *TIBCO BusinessEvents® Administration*
- *TIBCO BusinessEvents® Developer's Guide*
- *TIBCO BusinessEvents® Cloud Deployment Guide*
- *TIBCO BusinessEvents® Data Modeling Developer's Guide*
- *TIBCO BusinessEvents® Event Stream Processing Pattern Matcher Developer's Guide*
- *TIBCO BusinessEvents® Event Stream Processing Query Developer's Guide*
- *TIBCO BusinessEvents® Configuration Guide*
- *TIBCO BusinessEvents® WebStudio User's Guide*
- *TIBCO BusinessEvents® Decision Manager User's Guide*
- Online References:
  - *TIBCO BusinessEvents® Java API Reference*
  - *TIBCO BusinessEvents® Functions Reference*

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.

Before You Begin

You can containerize a TIBCO BusinessEvents application by using Docker. You can also run the dockerized TIBCO BusinessEvents application in a Kubernetes cluster on the cloud platform of your choice.

Supported Versions

Before you begin, see TIBCO BusinessEvents Readme for supported versions of Docker and cloud platforms.

Concepts

Before you begin, you must be familiar with the following concepts and services:

- Docker concepts. See Docker Documentation.
- Kubernetes concepts. See Kubernetes Documentation.
- Administration knowledge of the cloud platform and the service that you want to use:
    - Amazon Web Services (AWS)
    - Microsoft Azure and Azure Kubernetes Service (AKS)
    - Red Hat OpenShift Container Platform
    - Google Cloud Platform (GCP) and Pivotal Container Service (PKS)

Preparing for TIBCO BusinessEvents Containerization

Ensure that you have the following infrastructure in place:

- A machine with the Docker installation and initial setup based on your operating system, to generate Docker images. For complete information about Docker installation, see Docker Documentation.
- TIBCO BusinessEvents installation and a TIBCO BusinessEvents project that you want to deploy and run on the cloud. For installation instructions, see the *TIBCO BusinessEvents Installation Guide*.
- Clone or download the `cloud` folder from the TIBCO BusinessEvents GitHub repository to your system. You can download the `cloud` folder to any location on your system. The `cloud` folder contains utilities to deploy and monitor your TIBCO BusinessEvents applications on different cloud platforms by using Docker and Kubernetes.

    In this guide, it is assumed that the `cloud` folder from GitHub is downloaded at *BE_HOME* and merged with the existing *`BE_HOME`*`/cloud` folder. Thus, all file paths in this guide are mentioned based on this assumption.

- Installer ZIP files for the following software:
    - TIBCO BusinessEvents
    - TIBCO ActiveSpaces (for cache-based projects)
    - TIBCO BusinessEvents add-ons (if your project uses it)

    (macOS only) On the macOS platform, you can build only Linux containers. To build a Docker image on macOS, you must store the TIBCO BusinessEvents Linux installer ZIP file (`TIB_businessevents-enterprise_`*`<version>`*`_linux26gl25_x86_64.zip`) on your computer instead of the macOS installer ZIP file. Similarly, if your application uses cache, download Linux installers for TIBCO ActiveSpaces. Or if it uses TIBCO BusinessEvents add-ons, download Linux installers for TIBCO BusinessEvents add-ons on your computer.

- (Optional) For monitoring TIBCO BusinessEvents applications, install TIBCO Enterprise Administrator with the latest hotfix. For installation instructions, see TIBCO Enterprise Administrator 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.

Dockerize TIBCO BusinessEvents

Using the scripts provided at the TIBCO BusinessEvents GitHub repository, you can containerize and run a TIBCO BusinessEvents application by using Docker.

Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries. Your application can run in any environment as all the dependencies are already present in the image of the application. For details about Docker, see Docker Documentation.

A TIBCO BusinessEvents application comprises a common TIBCO BusinessEvents runtime and project (application) specific TIBCO BusinessEvents code running inside the TIBCO BusinessEvents runtime. Thus to containerize a TIBCO BusinessEvents application, TIBCO BusinessEvents software archive and application archive are included in the Docker image.

Docker Scripts with BusinessEvents

TIBCO BusinessEvents provides the following scripts for building images of TIBCO BusinessEvents application and its components at `BE_HOME\cloud\docker\bin`:

- `build_app_image` - Script to build the Docker image for your TIBCO BusinessEvents application.
- `build_rms_image` - Script to build the Docker image for RMS.
- `build_teagent_image` - Script to build the Docker image for TIBCO BusinessEvents Enterprise Administrator Agent.

These scripts use the platform-specific Dockerfiles bundled with TIBCO BusinessEvents at `BE_HOME\cloud\docker\bin`. For details about Dockerfiles provided with TIBCO BusinessEvents, see Dockerfile for TIBCO BusinessEvents.

Containerizing TIBCO BusinessEvents Application and Components

To deploy and run different TIBCO BusinessEvents components in Docker, see the following topics:

- Containerizing TIBCO BusinessEvents Application in Docker
- Containerizing RMS by Using Docker
- Building TIBCO BusinessEvents Enterprise Administrator Agent Docker Image

Dockerfile for TIBCO BusinessEvents

TIBCO BusinessEvents provides the Dockerfiles for creating Docker image of the TIBCO BusinessEvents application and components.

The Dockerfiles for different platforms and components provided with TIBCO BusinessEvents to build Docker image by using either software installers or by using the existing TIBCO BusinessEvents installation. The following table lists Dockerfiles provided with the TIBCO BusinessEvents installation, along with the list of the script files that are using the Dockerfile and type of the container:

Files Location	Dockerfile	Platform	Used for Creating the Docker Image of	Associated Script File
`BE_HOME/cloud/docker/bin` (*For the Docker image based on software installers*)	`Dockerfile`	Ubuntu	TIBCO BusinessEvents application	build_app_image.sh

Files Location	Dockerfile	Platform	Used for Creating the Docker Image of	Associated Script File
	`Dockerfile.rhel`	Red Hat Enterprise Linux	TIBCO BusinessEvents application	build_app_image.sh
	`Dockerfile.win`	Microsoft Windows	TIBCO BusinessEvents application	build_app_image.bat
	`Dockerfile-rms`	Ubuntu	Rule Management Server (RMS)	build_rms_image.sh
	`Dockerfile-rms.win`	Microsoft Windows	Rule Management Server (RMS)	build_rms_image.bat
	`Dockerfile-teagent`	Ubuntu **Note:** The same Dockerfile is used on both Linux and Windows platforms, and creates a Linux container.	TIBCO BusinessEvents Enterprise Administrator Agent	build_teagent_image.sh build_teagent_image.bat
BE_HOME`/cloud/docker/frominstall` *(For the Docker image based on the existing TIBCO BusinessEvents installation)*	`Dockerfile_fromtar`	Ubuntu	TIBCO BusinessEvents application	build_app_image.sh
	`Dockerfile_fromtar.win`	Microsoft Windows	TIBCO BusinessEvents application	build_app_image.bat
	`Dockerfile-rms_fromtar`	Ubuntu	Rule Management Server (RMS)	build_rms_image.sh
	`Dockerfile-rms_fromtar.win`	Microsoft Windows	Rule Management Server (RMS)	build_rms_image.bat
	`Dockerfile-teagent_fromtar`	Ubuntu	TIBCO BusinessEvents Enterprise Administrator Agent	build_teagent_image.sh

When building the Red Hat Enterprise Linux based Docker image, update the dockerfile (`Dockerfile.rhel`) with your subscribed Red Hat Docker image before running the Docker build script. In the Dockerfile, replace the `<RHEL_IMAGE>` placeholder with the Red Hat Enterprise Linux Docker image name.

To use any other platform, update the Dockerfile with the platform details. For more information about Dockerfile structure, see Docker Documentation.

The following sections identify key instructions to set up key configurations for the TIBCO BusinessEvents Docker images.

Environment Variables (ENV)

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

```
ENV key value
```

The default TIBCO BusinessEvents Dockerfiles have the following common environment variables:

- CDD_FILE: Path of the TIBCO BusinessEvents application or RMS CDD file.
- EAR_FILE: Path of the TIBCO BusinessEvents application or RMS EAR file.
- PU: The name of the processing unit to run. The value is provided at the runtime by the user. The default value is default.
- AS_DISCOVER_URL: Discovery URL of TIBCO ActiveSpaces.
- ENGINE_NAME: TIBCO BusinessEvents engine name. The default value is be-engine.
- LOG_LEVEL: Logging level for BusinessEvents. The default value is na.

```
# BusinessEvents Environment Variables
ENV CDD_FILE no-default
ENV PU default
ENV EAR_FILE no-default
ENV ENGINE_NAME be-engine
ENV LOG_LEVEL na
ENV AS_DISCOVER_URL self
```

Data Volumes (VOLUME)

The VOLUME instruction is used to enable access from your container to a directory on the host machine. The syntax for the VOLUME instruction is:

```
VOLUME /dir1, /dir2 ...
```

Using data volumes, you can persist the data across Docker runs. For example, in the default Dockerfile ActiveSpaces Shared Nothing file stores, the log file locations and the Rule Management Server directories are configured. The Docker volumes for them are created and all internal file paths are rooted to the specified directories. These volumes are predefined in Dockerfiles provided with TIBCO BusinessEvents. The following table lists the predefined (Linux) directory path for creating data volume. Similar paths are also defined in the Windows Dockerfiles.

Volumes	Dockerfiles	Description
/mnt/tibco/be/logs	Dockerfile Dockerfile-rms	Directory where log files are stored.
/mnt/tibco/be/data-store	Dockerfile Dockerfile-rms	Directory where shared nothing data is stored.
/opt/tibco/be/${BE_SHORT_VERSION}/rms/config/security	Dockerfile-rms	Directory which holds the RMS application's ACL (permission configuration) and user.pwd files.

Volumes	Dockerfiles	Description
`/opt/tibco/be/${BE_SHORT_VERSION}/examples/standard/WebStudio`	`Dockerfile-rms`	The repository directory for BusinessEvents WebStudio where all projects are stored.
`/opt/tibco/be/${BE_SHORT_VERSION}/rms/config/notify`	`Dockerfile-rms`	Directory where email notification configuration files are stored.
`/opt/tibco/be/${BE_SHORT_VERSION}/rms/shared`	`Dockerfile-rms`	Directory where RMS applications exported files are stored.
`/opt/tibco/be/${BE_SHORT_VERSION}/rms/locale`	`Dockerfile-rms`	Directory where the user locale configuration is stored.
`/mnt/tibco/be/`	`Dockerfile-teagent`	Directory where TIBCO BusinessEvents is stored.
`/opt/tibco/be/${BE_SHORT_VERSION}/teagent/logs/`	`Dockerfile-teagent`	Directory where TIBCO BusinessEvents Enterprise Administrator Agent logs are stored.

Here, *BE_SHORT_VERSION* stands for the TIBCO BusinessEvents software version in the short form. For example, for TIBCO BusinessEvents version 5.6.1, the *BE_SHORT_VERSION* is 5.6.

Ports (EXPOSE)

The `EXPOSE` instruction is used to associate a specified port to enable networking between the running process inside the container and the external nodes (that is, the host). The syntax for the `EXPOSE` instruction is:

```
EXPOSE port1 port2 ...
```

By default the following ports are exposed by the TIBCO BusinessEvents Dockerfiles:

- `50000` and `50001`: These are the ports on which TIBCO ActiveSpaces listens. These are exposed by the base image.

  The port for `AS_LISTEN_URL` and `AS_DISCOVER_URL` are set to `50000` in scripts. Also, for `AS_REMOTE_LISTEN_URL`, the port is set to `50001`. You must not change these ports.

- `5555`: This is the JMX port exposed by the base image.
- `8090` and `5000`: These are the rule management server ports exposed by the base image.

These ports can be mapped during Docker run.

Containerizing TIBCO BusinessEvents Application in Docker

You can deploy and run a TIBCO BusinessEvents application in Docker by using the Docker image of the TIBCO BusinessEvents application.

Prerequisites

See Preparing for TIBCO BusinessEvents Containerization

Procedure

1. Build the TIBCO BusinessEvents application Docker image using the script provided by TIBCO BusinessEvents.

   See Building TIBCO BusinessEvents Application Docker Image.
2. (*Linux containers only*) Create a network bridge for internal communication among Docker images by using the following command.

   ```
   docker network create <BRIDGE_NAME>
   ```

   For details about the command, see Docker Documentation.
3. Run the TIBCO BusinessEvents application image in Docker.

   See Running a TIBCO BusinessEvents Application in Docker.

Building TIBCO BusinessEvents Application Docker Image

TIBCO BusinessEvents provides script files to build a Docker image of the TIBCO BusinessEvents application by using bundled Dockerfiles.

You can build the Docker image either by using existing TIBCO BusinessEvents installation (*BE_HOME*) from your computer or by using the software installers of TIBCO BusinessEvents and other required products. The Docker image generated by using software installers is of smaller size in comparison to the Docker image generated by using your TIBCO BusinessEvents installation.

For details about Dockerfiles provided with TIBCO BusinessEvents, see Dockerfile for TIBCO BusinessEvents.

When building the Red Hat Enterprise Linux based Docker image, update the dockerfile (`Dockerfile.rhel`) with your subscribed Red Hat Docker image before running the Docker build script. In the Dockerfile, replace the `<RHEL_IMAGE>` placeholder with the Red Hat Enterprise Linux Docker image name.

Prerequisites

See Preparing for TIBCO BusinessEvents Containerization

Procedure

- **Application Docker Image by Using Software Installers**

  Go to the `BE_HOME/cloud/docker/bin` folder and run the `build_app_image` application Docker image building script.

  Syntax:

  ```
  build_app_image -l <installers-directory> -a <apps-artifact-directory> -r <app-image-name>:<app-image-version> [--gv-providers <names-of-GV-providers>] [-d <Dockerfile>] [-h]
  ```

  Example:

  ```
  build_app_image -l /home/user/tibco/installers -a /home/user/tibco/be/5.6/examples/standard/FraudDetection -r fdapp --gv-providers "consul"
  ```
- **(*Windows and Linux Only*) Application Docker Image by Using Existing TIBCO BusinessEvents Installation**

  Go to the `BE_HOME/cloud/docker/frominstall` folder and run the `build_app_image` application Docker image building script.

  Syntax:

  ```
  build_app_image [-l <BE_HOME_location>] -a <apps-artifact-directory> -r <app-image-name>:<app-image-version> [--gv-providers <names-of-GV-providers>] [-d <Dockerfile>] [-h]
  ```

Example:

```
build_app_image -a /home/user/tibco/be/5.6/examples/standard/FraudDetection -r
fdapp --gv-providers "consul"
```

For the Windows platform, enclose all arguments in double quotes (").

Application Docker Image Building Script Arguments

Argument	Required/ Optional	Description
`-l/--installers-location` (*For the Docker image based on software installers*)	Required	The location where installers for TIBCO BusinessEvents, TIBCO ActiveSpaces (optional), and TIBCO BusinessEvents add-ons (optional) are stored. This option is available for scripts that are run from *BE_HOME*`/cloud/docker/bin`.
`-l/--be-home` (*For the Docker image based on the existing TIBCO BusinessEvents installation*)	Optional	Specify TIBCO BusinessEvents installation (*BE_HOME*) location. This is optional if the script runs from its default location (*BE_HOME*`/cloud/docker/frominstall`).
`-a/-app-location`	Required	The location where the application CDD file, enterprise archive (EAR) file, and external JAR files are stored.
`-r/-repo`	Required	Name that you want to assign to application Docker image. Optionally, you can provide the version number for the Docker image. Use the following naming convention for the application Docker image: `<image-name>:[version-number]` For example, `fdc:1.0`.
`--gv-providers`	Optional	Provide names of the global variable providers as a comma-separated list. The values are: • `consul` - Use Consul as the key-value provider. This downloads the Consul CLI into the application Docker image.
`-d/--dockerfile`	Optional	The custom Dockerfile used for generating image. You can use your own Dockerfile or you can edit and use the Dockerfile provided with the TIBCO BusinessEvents installation. If you have placed the Dockerfile at a location other than the default location, provide the path of the Dockerfile. If not specified, the script uses the default bundled Dockerfile. For the list of default Dockerfiles associated with the scripts that are provided with TIBCO BusinessEvents, see Dockerfile for TIBCO BusinessEvents.

Argument	Required/ Optional	Description
`-h/--help`	Optional	Displays help for the script file.

What to do next

Running a TIBCO BusinessEvents Application in Docker

Running a TIBCO BusinessEvents Application in Docker

By using the TIBCO BusinessEvents application Docker image, you can run the TIBCO BusinessEvents application in Docker.

Prerequisites

- Build the BusinessEvents application Docker image. See Building BusinessEvents Application Docker Image.
- (Linux containers only) Ensure that a network bridge exists for internal communication between Docker images. You can use the `docker network create` command to create the network bridge. For details about the command, see Docker Documentation.

Procedure

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

  ```
  docker run --net=<BRIDGE_NETWORK> -p <CONTAINER_PORT>:<HOST_PORT> -v
  <LOCAL_DIRECTORIES>:<CONTAINER_DIRECTORIES> -e <ENVIRONMENT_VARIABLES>
  <APPLICATION_IMAGE_NAME>:<IMAGE_VERSION>
  ```

  For details about the `docker run` command options, see Docker Run Command Reference.

Example

```
docker run -p 8110:8110 -e PU=default "HOSTNAME=localhost" httpapp
```

Setting Up the Application to Use Global Variables from Consul

Your TIBCO BusinessEvents application deployed on a cloud platform can use global variables from the key-value store in Consul.

Prerequisites

- The Consul server that is to be used as key-value store for the application global variables must already be setup. For instructions on installation and setup, see the Consul documentation.
- (Optional) For a secured (HTTPS) Consul server, ensure that you have access to the CA and CLI certificates.

Procedure

1. Connect to the Consul server that you have already setup from your web browser. Set up your application global variables in the Consul server as key-value pairs.
   Syntax for keys in Consul is

   ```
   <AppName>/<ProfileName>/<GV-Key> = <GV-Value>
   ```

   Where,

- *<AppName>* is a name for the TIBCO BusinessEvents application of your choosing, for example, FraudDetection.
- *<ProfileName>* is the name for the profile in the application, for example, prod, default, and so on.
- *<GV-Key>* is the name of the global variable as defined in your TIBCO BusinessEvents application. In case of global variables within a global variable group, use the usual format of separating them with a forward slash, for example, RMS/port.
- *<GV-Value>* is the value to set for the global variable.

2. (Optional) For the secured Consul server, copy the CA and CLI certificates in the same folder as application EAR and CDD files.
3. Build the application Docker image with Consul as the key-value store for global variables of the application.

   Run the Docker image build script with the option `--gv-providers="consul"`. This downloads the Consul CLI and installs it in the application Docker image.

   For example (for Linux):

   ```
   build_app_image.sh -l /home/user/tibco/installers -a /home/user/app -r fd:latest
   --gv-providers "consul"
   ```

   For more information on `build_app_image` script, see Building TIBCO BusinessEvents Application Docker Image.
4. Run the application Docker image with Consul server details as environment variables.

   Command syntax:

   ```
   docker run -e CONSUL_SERVER_URL=<consul-server-url> -e
   APP_CONFIG_PROFILE=<profile-name> -e BE_APP_NAME=<app-name>  -e
   CONSUL_CACERT=<CA_certificate_path> -e CONSUL_CLIENT_CERT=<CLI-certificate-path>
   -e CONSUL_CLIENT_KEY=<CLI-certificate-keystore-file-
   path><APPLICATION_IMAGE_NAME>:<IMAGE_VERSION>
   ```

   The `CONSUL_CACERT`, `CONSUL_CLIENT_CERT`, and `CONSUL_CLIENT_KEY` environment variables are only required for the secured Consul server.

   Sample command (for Linux):

   ```
   docker run -e CONSUL_SERVER_URL=http://consul:8500 -e
   APP_CONFIG_PROFILE=profile1 -e BE_APP_NAME=FraudDetection -e "CONSUL_CACERT=/opt/
   tibco/be/ext/consul-agent-ca.pem" -e "CONSUL_CLIENT_CERT=/opt/tibco/be/ext/dc1-
   cli-consul-0.pem" -e "CONSUL_CLIENT_KEY=/opt/tibco/be/ext/dc1-cli-consul-0-
   key.pem"--network=mynet -p 8108:8108 --name=fd fd:latest
   ```

   For more details about the Consul server environment variables and other options available with the docker run command, see Docker Run Command Reference.

Result

The global variables from the Consul server are now available in the application with the updated value.

Containerizing RMS by Using Docker

The rule management server (RMS) is an integral part of BusinessEvents for using TIBCO BusinessEvents WebStudio and Decision Manager. To run TIBCO BusinessEvents WebStudio in a container, you must containerize RMS.

To run RMS in a container, you must build its Docker image and run it with Docker like a TIBCO BusinessEvents application.

Prerequisites

See Preparing for TIBCO BusinessEvents Containerization

Procedure

1. Build the RMS Docker image using the script provided by TIBCO BusinessEvents.

   See Building RMS Docker Image.
2. (*Linux containers only*) Create a network bridge for internal communication among Docker images by using the following command.
   ```
   docker network create <BRIDGE_NAME>
   ```
   For details about the command, see Docker Documentation.
3. Run the RMS Docker image in Docker.

   See Running RMS in Docker.

Building RMS Docker Image

TIBCO BusinessEvents provides a script file to build the RMS Docker image by using bundled Dockerfiles.

You can build the Docker image either by using existing TIBCO BusinessEvents installation (*BE_HOME*) from your computer or by using the software installer of TIBCO BusinessEvents and other required products. The Docker image generated by using software installers is of smaller size in comparison to the Docker image generated by using your TIBCO BusinessEvents installation.

For details about Dockerfiles provided with TIBCO BusinessEvents, see Dockerfile for TIBCO BusinessEvents.

Prerequisites

See Preparing for TIBCO BusinessEvents Containerization

Procedure

- **RMS Docker Image by Using Software Installers**

  Go to the `BE_HOME/cloud/docker/bin` folder and run the RMS Docker image building script `build_rms_image`.

  Syntax:
  ```
  build_rms_image -l <installers-directory> [-a <rms-artifact-directory>] [-r <rms-image-name>:<rms-image-version>]  [-d <Dockerfile>] [-h]
  ```
  Example:
  ```
  build_rms_image -l /home/user/tibco/installers
  ```
- **(*Windows and Linux Only*) RMS Docker Image by Using Existing TIBCO BusinessEvents Installation**

  Go to the `BE_HOME/cloud/docker/frominstall` folder and run the RMS Docker image building script `build_rms_image`.

  Syntax:
  ```
  build_rms_image [-l <BE_HOME_location>] [-a <rms-artifact-directory>] [-r <rms-image-name>:<rms-image-version>] [-d <Dockerfile>] [-h]
  ```
  Example:
  ```
  build_rms_image -r rms.server:1.0
  ```

For the Windows platform, enclose all arguments in double quotes (").

RMS Docker Image Script Arguments

Argument	Required/ Optional	Description
`-l/--installers-location` *(For the Docker image based on software installers)*	Required	The location where installers for TIBCO BusinessEvents, TIBCO ActiveSpaces, and TIBCO BusinessEvents add-ons (optional) are stored. This option is available for scripts that are run from *BE_HOME*`/cloud/docker/bin`.
`-l/--be-home` *(For the Docker image based on the existing TIBCO BusinessEvents installation)*	Optional	Specify TIBCO BusinessEvents installation (*BE_HOME*) location. This is optional if the script runs from its default location (*BE_HOME*`/cloud/docker/frominstall`).
`-a/--app-location`	Optional	If you have modified the RMS project, specify the location of the updated `RMS.cdd` and `RMS.ear` files. If not specified, the script file takes the `RMS.cdd` and `RMS.ear` files bundled with TIBCO BusinessEvents installers.
`-r/--repo`	Optional	Name that you want to assign to the RMS Docker image. Optionally, you can provide the version number for the Docker image. Use the following naming convention of the RMS Docker image: `<image-name>:[version-number]` For example, `rms.server:latest`. The default value is `rms:<BE_version>`. For example, for version 5.6.1 of TIBCO BusinessEvents, the default value is `rms:5.6.1`.
`-d/--dockerfile`	Optional	The custom Dockerfile used for generating image. You can use your own Dockerfile or you can edit and use the Dockerfile provided with the TIBCO BusinessEvents installation. If you have placed the Dockerfile at a location other than the default location, provide the path of the Dockerfile. If not specified, the script uses the default bundled Dockerfile. For the list of default Dockerfiles associated with the scripts that are provided with TIBCO BusinessEvents, see Dockerfile for TIBCO BusinessEvents.
`-h/--help`	Optional	Provides help for the script file.

What to do next

Running RMS in Docker

Running RMS in Docker

By using the TIBCO BusinessEvents application Docker image, you can run the TIBCO BusinessEvents application in Docker.

Prerequisites

- Build the RMS Docker image. See Building RMS Docker Image.
- (Linux containers only) Ensure that a network bridge exists for internal communication between Docker images. You can use the `docker network create` command of Docker to create the network bridge. For details about the command, see Docker Documentation .

Procedure

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

  ```
  docker run --net=<BRIDGE_NETWORK> -p <CONTAINER_PORT>:<HOST_PORT> -v
  <LOCAL_DIRECTORIES>:<CONTAINER_DIRECTORIES> -e <ENVIRONMENT_VARIABLES>
  <RMS_IMAGE_NAME>:<IMAGE_VERSION>
  ```

  For details about the Docker `run` command options, see Docker Run Command Reference.

Example

```
docker run -p 8090:8090 -e PU=default "HOSTNAME=localhost" rms:5.6.0
```

Docker Run Command Reference

The `docker run` command is used for containerizing and running a TIBCO BusinessEvents application by using its Docker image.

Syntax

```
docker run --net=<BRIDGE_NETWORK> -p <CONTAINER_PORT>:<HOST_PORT> -v
<LOCAL_DIRECTORIES> -e <ENVIRONMENT_VARIABLES>
<APPLICATION_IMAGE_NAME>:<IMAGE_VERSION>
```

Where:

- `--net=`*`<BRIDGE_NETWORK>`* - Specify the name of the network bridge that you have created. This connects the container to the specified network.
- `-p` *`<CONTAINER_PORT>:<HOST_PORT>`* - (Optional) Specify the host port and container port that you want to map.
- `-v` *`<LOCAL_DIRECTORIES>:<CONTAINER_DIRECTORIES>`* - (Optional) Specify the path of the local directory that you want to mount to the container.
- *`<APPLICATION_IMAGE_NAME>`* - Specify the name of the BusinessEvents application Docker image. If you want to use RMS, use the RMS Docker image name.
- *`<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.
  - **`AS_DISCOVER_URL`**: Specify the discover URL, which enables members to discover each other in the network. For example:

    ```
    docker run --net=simple-bridge --name=inference -e AS_DISCOVER_URL=tcp://
    cache:50000 -e PU=default -p 8109:8109 fdcache:v01
    ```

    Here the Docker name of the cache server "cacheagent" is used for the `AS_DISCOVER_URL` of the inference agent. The port number of `AS_DISCOVER_URL` must be set to `50000` as this is setup in the Docker scripts. As all agents running on the same Docker host can resolve Docker names to their IP addresses on the network, you can create clusters across instances on the same network.

- **PU**: Specify the processing unit that needs to be started. For example, running the application with "cache" as processing unit:

  ```
  docker run --net=be_network --name=cacheagent -e PU=cache fdcache:v01
  ```

- **LOG_LEVEL**: Specify the override value for the predefined log level. You can specify comma-separated values for the log patterns required. If the `LOG_LEVEL` environment variable is not specified, the `log-config` of the CDD file is used. The pattern configurations are the same as the `log-config` of the CDD file. For example:

  ```
  docker run --net=simple-bridge --name=cacheagent -e PU=cache -e
  LOG_LEVEL=*:debug fdcache:v01
  ```

- **DOCKER_HOST**: Specify the host where the `docker run` command is executed. This environment variable is required for remote JMX connections to the running container. For example:

  ```
  docker run --net=be_network --name=sample -p 5555:5555 -e PU=default -e
  DOCKER_HOST=10.97.123.56 sample:v01
  ```

  The default JMX port for engines running in Docker is `5555`. You must map this default port to the local port defined in the Dockerfile.

- **AS_PROXY_NODE**: Specifies whether the container run as a proxy node. Set the value to `true`, to start the node in proxy mode. For example:

  ```
  docker  run ... -e  AS_PROXY_NODE=true ...
  ```

  The port `50001` is set as the ActiveSpaces remote listen port which can be specified while connecting to the proxy node. For example:

  ```
  docker run ...  -e AS_REMOTE_LISTEN_URL=tcp://<container_name>:50001?
  remote=true ...
  ```

- **CONSUL_SERVER_URL**: Specify the URL of the Consul server.
- **APP_CONFIG_PROFILE**: Specify name of the global variables profile that you have used for grouping in Consul. The default value is “default”, if not specified.
- **BE_APP_NAME**: Specify the application name that you have used for grouping global variables in the key value store in Consul.
- **CONSUL_CACERT**: Specify the absolute path of the CA certificates placed in the container.
- **CONSUL_CLIENT_CERT**: Specify the absolute path of the CLI certificates placed in the container.
- **CONSUL_CLIENT_KEY**: Specify the absolute path of the client keystore file placed in the container.
- *TRA properties*: You can specify any of the BusinessEvents engine and JVM properties as an environment variable.

  To use the property, append **`tra.`** at the beginning of the property name. For example, to use `java.extended.properties`, provide **`tra.`**`java.extended.properties` and its value as environment variable. The value of the environment variable `tra.java.extended.properties` overwrites the value of the `java.extended.properties` property in the `be-engine.tra` file.

  You can also specify a few JVM properties, such as, `-Xms`, `-Xmx`, and `-Xss` as environment variable individually. These individual JVM properties, when specified as environment variable, take precedence over the JVM properties defined in the `tra.java.extended.properties` environment variable. Other JVM properties, such as, garbage collection properties still have to be defined under the `tra.java.extended.properties` environment variable. The following table lists the environment variables that you can use for these JVM property options.

Environment Variables for JVM Properties

Task	JVM Property Option	Environment Variable
Set initial Java heap size	`-Xms`	`tra.java.heap.size.initial`
Set maximum Java heap size	`-Xmx`	`tra.java.heap.size.max`
Set Java thread stack size	`-Xss`	`tra.java.stack.size`

For example:

```
docker run -e "tra.java.heap.size.initial=1024m" -e
"tra.java.heap.size.max=1024m" -e "tra.java.stack.size=2m" -
e="tra.java.extended.properties=-server -Xms512m -Xmx512m -javaagent:%BE_HOME
%/lib/cep-base.jar -XX:MaxMetaspaceSize=256m -XX:+UseParNewGC -
XX:+UseConcMarkSweepGC" com.tibco.be.fd:v016
```

In the previous example, `tra.java.heap.size.initial=1024m` and `tra.java.heap.size.max=1024m` takes precedence over the `-Xms512m` and `-Xmx512m` options of `tra.java.extended.properties`. Thus, the initial Java heap size and maximum Java heap size is set to `1024M` instead of `512M`. Also, the `tra.java.stack.size=2m` environment variable sets the `-Xss` option of `java.extended.properties` property in the `be-engine.tra` file to `2M`.

– ***Global Variable***: You can specify a global variable as an environment variable to override its value. Provide the global variable name and its value as an environment variable. For example, to specify value for the global variable `HOSTNAME` as `localhost`, run the following command:

```
docker  run ... -e  "DB_USERNAME=scott" ...
```

In order to update global variables during runtime, ensure that global variables are used in shared resources of the TIBCO BusinessEvents project. For example, to change database details at runtime without regenerating application Docker image, ensure that global variables are used in the JDBC shared resource.

For more details about the `docker run` command, see Docker Documentation.

Building TIBCO BusinessEvents Enterprise Administrator Agent Docker Image

TIBCO BusinessEvents provides script files to build TIBCO BusinessEvents Enterprise Administrator Agent Docker image by using bundled Dockerfiles.

You can build the Docker image either by using existing TIBCO BusinessEvents installation (*BE_HOME*) from your computer or by using the software installer of TIBCO BusinessEvents and other required products. The Docker image generated by using software installers is of smaller size in comparison to the Docker image generated by using your TIBCO BusinessEvents installation.

Prerequisites

See Preparing for TIBCO BusinessEvents Containerization

Procedure

- **Docker Image by Using Software Installers**

  Go to the *BE_HOME*`/cloud/docker/bin` folder and run the `build_teagent_image` TIBCO BusinessEvents Enterprise Administrator Agent Docker image building script.

Syntax:

```
build_teagent_image -l <installers-directory> [-r <teagent-image-name>:<teagent-image-version>] [-d <Dockerfile>] [-h]
```

Example:

```
build_teagent_image -l /home/user/tibco/installers
```

- **(*Linux Only*) Docker Image by Using Existing TIBCO BusinessEvents Installation**

  Go to the *BE_HOME*`/cloud/docker/bin/frominstall` folder and run the `build_teagent_image` TIBCO BusinessEvents Enterprise Administrator Agent Docker image building script.

  Syntax:

  ```
  build_teagent_image [-l <BE_HOME-location>] [-r <teagent-image-name>:<teagent-image-version>] [-d <Dockerfile>] [-h]
  ```

  Example:

  ```
  build_teagent_image -r teagent-be:1.0
  ```

For the Windows platform, enclose all arguments in double quotes (").

TIBCO BusinessEvents Enterprise Administrator Agent Docker Image Script Arguments

Argument	Required/ Optional	Description
`-l/--installers-location` (*For the Docker image based on software installers*)	Required	The location where installers for TIBCO BusinessEvents, TIBCO ActiveSpaces, and TIBCO BusinessEvents add-ons (optional) are stored. This option is available for scripts that are run from *BE_HOME*`/cloud/docker/bin`.
`-l/--be-home` (*For the Docker image based on the existing TIBCO BusinessEvents installation*)	Optional	Specify TIBCO BusinessEvents installation (*BE_HOME*) location. This is optional if the script runs from its default location (*BE_HOME*`/cloud/docker/frominstall`).
`-r/--repo`	Optional	Name that you want to assign to the TIBCO BusinessEvents Enterprise Administrator Agent Docker image. Optionally, you can provide the version number for the Docker image. Use the following naming convention for the TIBCO BusinessEvents Enterprise Administrator Agent Docker image: *`<image-name>`*`:`*`[version-number]`* For example, `teagent:1.0`. The default value is `teagent:`*`<BE_version>`*. For example, for version 5.6.1 of TIBCO BusinessEvents, the default value is `teagent:5.6.1`.

Argument	Required/ Optional	Description
`-d/--dockerfile`	Optional	The custom Dockerfile to be used for generating image. You can use your own Dockerfile or you can edit and use the Dockerfile provided with the TIBCO BusinessEvents installation. If you have placed the Dockerfile at a location other than the default location, provide the path of the Dockerfile. If not specified, the script uses the default bundled Dockerfile. For list of default Dockerfiles for the scripts provided with TIBCO BusinessEvents, see Dockerfile for TIBCO BusinessEvents.
`-h/--help`	Optional	Provides help for the script file.

What to do next

Run the TIBCO BusinessEvents Enterprise Administrator Agent Docker image in Kubernetes based on your preferred cloud platform. For details, see:

- Monitoring TIBCO BusinessEvents Applications on OpenShift Container Platform
- Monitoring TIBCO BusinessEvents Applications on Microsoft Azure
- Monitoring TIBCO BusinessEvents Applications on AWS
- Monitoring TIBCO BusinessEvents Applications on Enterprise PKS

Setting Up BusinessEvents Multihost Clustering on Amazon EC2 Instances Using Docker

You can set up BusinessEvents multihost clustering on Amazon Elastic Compute Cloud (Amazon EC2) instances using Docker and Weave Net.

Prerequisites

- An Amazon Web Services (AWS) account. Refer to the Amazon EC2 documentation at https://aws.amazon.com/documentation/ec2/ to learn Amazon EC2 concepts and how to use the Amazon EC2 console.
- TIBCO BusinessEvents application image. See Dockerize TIBCO BusinessEvents for more details on running TIBCO BusinessEvents on Docker.
- (Optional) Docker Hub registry account or any other Docker registry account. Refer to the https://docs.docker.com/ to learn more about Docker.
- Weave Net for multihost docker networking. Refer to the Weave Net documentation at https://www.weave.works/docs/net/latest/features/ to learn on how to use Weave Net and how to integrate with Docker.
- Amazon Elastic File System configuration (EFS) for shared nothing persistence. Refer to the Amazon EFS documentation at https://aws.amazon.com/documentation/efs/ to learn about Amazon EFS concepts and configurations.
- Relational Database Service configuration (RDS) for shared all persistence. Refer to the Amazon RDS documentation at https://aws.amazon.com/documentation/rds/ to learn about Amazon RDS concepts and configurations.

Setting Up Standalone Amazon EC2 Instances

For BusinessEvents multihost clustering, you must create Amazon Elastic Cloud Compute (Amazon EC2) instances and configure Docker and Weave Net on each of them. This setup is common for shared all and shared nothing persistence options.

Procedure

1. Log in to Amazon EC2 console with your credentials.

   Refer to Amazon EC2 documentation at https://aws.amazon.com/documentation/ec2/ for more details on setting Amazon EC2 account.

2. In the Amazon EC2 console, create a new security group with the following inbound rules.

   Inbound Rules

Rule No.	Type	Protocol	Port	Source
1	SSH	TCP	22	Anywhere
2	Custom TCP Rule	TCP	6783	Anywhere
3	Custom UDP Rule	UDP	6783	Anywhere
4	Custom TCP Rule	TCP	*<HTTP Port as per BusinessEvents project>*	Anywhere

   Port TCP/UDP 6783 is required for weave networking. You can configure source according to your requirement.

   Refer to Amazon EC2 documentation at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html for details on how to create security group.

3. On the Amazon EC2 console, create two or more Standalone Amazon EC2 instances of type Ubuntu or CentOS or as per your requirement. Specify the configuration parameters according to your requirement in the wizard.

   - Select the default Virtual Private Cloud (VPC) for testing purpose or you can use an customized one.
   - Select the security group created earlier in Step 2.
   - Generate a new key pair (`.pem`) per instance and save it.

   Refer to the Amazon EC2 documentation at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/LaunchingAndUsingInstances.html for more details on launching an instance.

4. In the Amazon EC2 console, on **Review Instance Launch** page, check the details of your instance, and after the verification click **Launch**.

5. Ensure that all instances are in the "running" state and status checks are marked with no error.

6. Note down the public and private IP address/DNS of all instances, which can be later used for connection.

7. Change the permission of PEM key.

   ```
   > chmod 400 mykey.pem
   ```

8. Securely log in to Amazon EC2 instances using an SSH client.

   ```
   > ssh -i /pathto/mykey.pem ec2-user@<public IP address of EC2 instance or public DNS>
   ```

User name could be `ec2-user` or `ubuntu` as per the Amazon EC2 instance type.

9. Install Docker on all Amazon EC2 instances.

   Refer to the installation instructions mentioned in the Docker Documentation at https://docs.docker.com/engine/installation/.

10. Install Weave Net all EC2 instances.

    ```
    > sudo curl -L git.io/weave -o /usr/local/bin/weave
    > sudo chmod a+x /usr/local/bin/weave
    ```

    Refer to the installation instructions in the Weave Net documentation at https://www.weave.works/docs/net/latest/installing-weave/.

11. Start weave on each instance, and provide it other peers private IP addresses.

    On Instance 1,

    ```
    > weave launch
    ```

    On Instance 2,

    ```
    > weave launch <HostName/Private IP address of Instance 1>
    ```

12. Run the following command and check status of the peers connection.

    ```
    > weave status
    ```

    If the connection is successful, the status displays the number of established connections. For example,

    ```
    Peers: 2 (with 2 established connections)
    ```

Configuring Amazon RDS for Shared All Persistence

In this approach BusinessEvents application image is built locally and the docker registry is used to push or pull images. You can also build images directly on Amazon EC2 instances. If required, you can also configure separate VPC and security group.

Prerequisites

Check Amazon Relational Database Service (RDS) prerequisites at http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SettingUp.html.

Procedure

1. Create an Amazon RDS of type Oracle.

   Refer to the Amazon RDS documentation at http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.Oracle.html for more details on how to do it.

2. Use default VPC, same used for Amazon EC2 instances. Also, in the same security group add one more inbound rule for database port.

   Inbound Rules

Type	Protocol	Port	Source
Custom TCP Rule	TCP	1501	Anywhere

   Or if required, you can create a separate security group for the database instance

3. After the database instance is running and is in "available" state, you can establish a connection to it using any SQL client.

4. Create a BusinessEvents specific user and run all BusinessEvents specific scripts that are required.

5. After the database setup is ready, use the same database setup in the JDBC shared resource. You can use the database instance endpoint as **Database URL** in the JDBC shared resource. Use the **Test Connection** feature to check if the connection is successful.
6. Create BusinessEvents application docker image locally on any machine and push it to docker registry.

   See Containerizing TIBCO BusinessEvents Application in Docker for more details on how to do it.
7. Pull this BusinessEvents application docker image on all Amazon EC2 instances.

   After the BusinessEvents application image is available on all EC2 instances, you can run BusinessEvents application containers.
8. Set the Weave environment on all Amazon EC2 instances for running BusinessEvents application containers.

   ```
   >  eval $(weave env)
   ```
9. Start containers on all Amazon EC2 instances.
   For example,

   ```
   //Start cache 1 on instance 1
   docker run -d --name=cache1SA -e PU=cache <username>/fdstore_sharedall:GA
   //Start cache 2 on instance 2
   docker run -d --name=cache2SA -e PU=cache -e AS_DISCOVER_URL=tcp://
   cache1SA:50000 <username>/fdstore_sharedall:GA
   //Start inference on instance 2
   docker run -d --name=InfSA -p 8209:8209 -e PU=default -e AS_DISCOVER_URL=tcp://
   cache1SA:50000 <username>/fdstore_sharedall:GA
   ```

   Ensure that all BusinessEvents application containers are connected to each other and inference is processing events at port 8209.
10. For sending events using `readme.html` of the example application, replace `localhost` with the public IP address of instance where the inference container is running.

    As long as the RDS database instance is in running state, data is persisted.
11. To check the data recovery, stop all Amazon EC2 instances and start them again.
12. Restart all stopped containers and check that the data is recovered in cache containers.

Configuring Amazon EFS for Shared Nothing Persistence

Procedure

1. Create an Amazon Elastic File System (EFS) with the same Virtual Private Cloud (VPC) and security group as of the Amazon EC2 instances.

   Refer to the Amazon EFS documentation at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEFS.html for detailed steps on how to create an Amazon EFS file system.
2. Note down DNS name which is required while mounting EFS on Amazon EC2 instances.
3. Open an SSH client and connect to your Amazon EC2 instance.
4. Install the NFS client on all Amazon EC2 instances.

   On an Amazon Linux, Red Hat Enterprise Linux, or SUSE Linux instance, run the following command:

   ```
   > sudo yum install -y nfs-utils
   ```

   On an Ubuntu instance, run the following command:

   ```
   > sudo apt-get install nfs-common
   ```
5. Create a new directory on all Amazon EC2 instances, such as "efs".

   ```
   >  sudo mkdir efs
   ```

6. Mount your file system by using the EFS DNS name.

   ```
   > sudo mount -t nfs4 -o
   nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 fs-
   cb8b5e62.efs.us-west-2.amazonaws.com:/ efs
   ```

   If the connection was not successful, refer to Amazon EFS troubleshooting documentation at http://docs.aws.amazon.com/efs/latest/ug/troubleshooting.html.

7. Run the following command to see the mount:

   ```
   > df -T
   ```

8. Update BusinessEvents application CDD with shared nothing datastore path as `/mnt/tibco/be/data-store`, which is declared as `VOLUME` in BusinessEvents base dockerfile.

9. Create BusinessEvents application docker image locally on any machine and push it to docker registry.

   See Containerizing TIBCO BusinessEvents Application in Docker for more details on how to do it.

10. Pull this BusinessEvents application docker image on all Amazon EC2 instances.

    Once the BusinessEvents application image is available on all EC2 instances, you can run BusinessEvents application containers.

11. Set the Weave environment on all Amazon EC2 instances for running BusinessEvents application containers.

    ```
    >  eval $(weave env)
    ```

12. Start containers on all Amazon EC2 instances.
    For example,

    ```
    //Start cache 1 on instance 1
    docker run -d --name=cache1SN -v /home/ubuntu/efs:/mnt/tibco/be/data-store -e
    PU=cache <username>/fdstore_sharednothing:GA
    //Start cache 2 on instance 2
    docker run -d --name=cache2SN -v /home/ubuntu/efs:/mnt/tibco/be/data-store -e
    PU=cache -e AS_DISCOVER_URL=tcp://cache1SN:50000 <username>/
    fdstore_sharednothing:GA
    //Start inference on instance 2
    docker run -d --name=InfSN -v /home/ubuntu/efs:/mnt/tibco/be/data-store -p
    8209:8209 -e PU=default -e AS_DISCOVER_URL=tcp://cache1SN:50000 <username>/
    fdstore_sharednothing:GA
    ```

    Ensure that all BusinessEvents application containers are connected to each other and inference is processing events at port 8209.

13. For sending events using `readme.html` of the example application, replace `localhost` with the public IP address of instance where the inference container is running.

    As long as EFS is in running state, data is persisted.

14. To check the data recovery, stop all Amazon EC2 instances and start them again. Mount the EFS target again as mentioned in Step 6.

15. Restart all stopped containers and check that the data is recovered in cache containers.

Running TIBCO BusinessEvents Applications in Kubernetes

Kubernetes is an open-source platform designed to automate deploying, scaling, and operating application containers. Kubernetes can run application containers on clusters of physical or virtual machines.

For more information about Kubernetes, see Kubernetes documentation.

In TIBCO BusinessEvents, to form a cluster, discovery nodes starts a cluster and other non-discovery nodes (cache and inference). These non-discovery nodes connect to one or more discovery nodes and become a member of the cluster. In Kubernetes, each TIBCO BusinessEvents node runs as a Kubernetes *pod*. Pods communicate with each other by using their IP addresses. However, due to the dynamic nature of the IP addresses, non-discovery nodes cannot always connect to discovery nodes. Thus, to resolve this issue, discovery nodes are modeled as Kubernetes *services*. The service is reachable by its name by using the Kubernetes DNS. Non-discovery nodes use indirection by using the Kubernetes service to connect to discovery nodes.

- TIBCO BusinessEvents on OpenShift Container Platform Based Kubernetes
- TIBCO BusinessEvents on Microsoft Azure Based Kubernetes
- TIBCO BusinessEvents on AWS Based Kubernetes
- TIBCO BusinessEvents on Amazon EKS Based Kubernetes
- TIBCO BusinessEvents on Pivotal Based Kubernetes
- TIBCO BusinessEvents on Minikube Based Kubernetes

TIBCO BusinessEvents on OpenShift Container Platform Based Kubernetes

You can run any TIBCO BusinessEvents application on OpenShift Container Platform based Kubernetes cluster and monitor them by using TIBCO BusinessEvents Enterprise Administrator Agent. You can also manage business rules through WebStudio by running RMS on OpenShift Container Platform based Kubernetes cluster.

For details, see OpenShift Container Platform documentation.

Readme for Sample Applications

TIBCO BusinessEvents provides `readme.html` files to help you in running the sample applications and components on OpenShift Container Platform. You can follow the instruction in the `readme.html` file to run the application, WebStudio, and TIBCO BusinessEvents Enterprise Administrator Agent by using the provided sample YAML files.

The following table lists location of `readme.html` and sample YAML files for running sample applications and other components:

Scenario	readme.html and Sample YAML Files Location
Running TIBCO BusinessEvents application (FraudDetection) without cache on OpenShift Container Platform	*BE_HOME*`\cloud\kubernetes\OpenShift\inmemory`
Running TIBCO BusinessEvents applications (FraudDetectionCache and FraudDetectionStore) with cache on OpenShift Container Platform	*BE_HOME*`\cloud\kubernetes\OpenShift\cache`
Running TIBCO BusinessEvents WebStudio on OpenShift Container Platform	*BE_HOME*`\cloud\kubernetes\OpenShift\rms`
Running TIBCO BusinessEvents Enterprise Administration Agent for monitoring TIBCO BusinessEvents applications on OpenShift Container Platform	*BE_HOME*`\cloud\kubernetes\OpenShift\tea`

Topics

- Running an Application on OpenShift Based Kubernetes Cluster
- Monitoring TIBCO BusinessEvents Applications on OpenShift Container Platform
- Running the RMS on OpenShift Container Platform

Running an Application on OpenShift Based Kubernetes Cluster

By using Red Hat OpenShift Container Platform, you can deploy a TIBCO BusinessEvents application in the Kubernetes cluster managed in an on-premises infrastructure.
As OpenShift Container Platform is built on top of Kubernetes cluster, you do not need to install Kubernetes separately. For details about OpenShift Container Platform, see OpenShift Container Platform documentation.

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization.
- Docker image of your TIBCO BusinessEvents application, see Building TIBCO BusinessEvents Application Docker Image.

Procedure

1. Set up OpenShift Container Platform CLI to deploy the application from a terminal.

   See Setting Up the OpenShift CLI Environment.
2. Push the application Docker image to the OpenShift Container Platform registry.

   See Pushing Application Docker Image to OpenShift Container Registry.
3. Based on your application architecture, deploy the application on the Kubernetes cluster. See the following topics based on your application persistence option:

   - Running the Application Without Backing Store on OpenShift Container Platform.
   - Running the Application with Shared Nothing Persistence on OpenShift Container Platform.
   - Running the Application with Shared All Persistence on OpenShift Container Platform.

Setting Up the OpenShift CLI Environment

To run the application in an Kubernetes cluster on OpenShift, you can use OpenShift Container Platform CLI which runs on top of the Kubernetes cluster.
You can download the `oc` client tool from the OpenShift web console and install it to execute OpenShift Container Platform commands. In OpenShift Container Platform, you can create a new project that defines the scope of resources and who can access those resources. The application images are deployed in a project.

For details about OpenShift Container Platform or any of the step in the following procedure, see the Red Hat OpenShift Container Platform documentation.

Prerequisites

- You must have valid subscription of OpenShift Container Platform on your Red Hat account, see Red Hat OpenShift Container Platform.

Procedure

1. Download and install the `oc` client tool on the master node to access the OpenShift CLI commands.
2. Log in to the OpenShift CLI by using the `oc login` command .

   For example:
   ```
   $ oc login 203.0.113.0:8443 --token=ov40AhpOCBITwHBtC_vat0SF4xJd8lQNjylccs8ZOLc
   ```
3. Create a new project by using the `oc new-project` command.

   For example:
   ```
   $ oc new-project be-project --description="For running BE applications." --
   display-name="be-project"
   ```

Result

A new project `be-project` is created and you are its project admin.

You can use the `oc status` command to see the status of your projects.

What to do next

Pushing Application Docker Image to OpenShift Container Registry.

Pushing Application Docker Image to OpenShift Container Registry

To deploy the application, you must push the application Docker image to the OpenShift Container Platform default Docker registry.

The following procedure lists sample steps to complete the task. These steps can vary based on your infrastructure setup. For details, see Red Hat OpenShift Container Platform documentation.

Prerequisites

- You must have the Docker image of your TIBCO BusinessEvents application, see Building TIBCO BusinessEvents Application Docker Image.
- You must be logged in to OpenShift Container Platform CLI, see Setting Up the OpenShift CLI Environment.

Procedure

1. Get the default Docker registry details of OpenShift Container Platform by using the `oc describe` command.

   ```
   oc describe -n default service/docker-registry
   ```

   The command returns the registry details on the terminal which you can use for pushing the application image. For example:

   ```
   $ oc describe -n default service/docker-registry
   Name:                docker-registry
   Namespace:           default
   Labels:              <none>
   Annotations:         <none>
   Selector:            docker-registry=default
   Type:                ClusterIP
   IP:                  192.0.2.0
   Port:                5000-tcp  5000/TCP
   TargetPort:          5000/TCP
   Endpoints:           198.51.100.10:5000
   Session Affinity:    ClientIP
   Events:              <none>
   ```

2. Tag the application Docker image with the OpenShift default Docker registry name and project name.

   Syntax:

   ```
   docker tag <image_name>:<version> <registry_login_server>/<project_name>/
   <image>:<version>
   ```

   For example,

   ```
   $ docker tag fdcache550:01  192.0.2.0:5000/be-project/fdcache550:01
   ```

3. Login to OpenShift Docker container registry using your OpenShift credentials.

   For example:

   ```
   $ docker login 192.0.2.0:5000 -u userid -p
   ov40AhpTXYZOwHBtC_vat0SF4xJd81QNjylccs8ZOLc
   ```

4. Push the tagged application Docker image to the OpenShift Docker container registry.

   Syntax:

   ```
   docker push <registry_login_server>/<project_name>/<image>:<version>
   ```

   For example:

   ```
   $ docker push 192.0.2.0:5000/be-project/fdcache550:01
   ```

Running the Application Without Backing Store on OpenShift Container Platform

After uploading your TIBCO BusinessEvents application Docker image with no backing store to the OpenShift Docker registry, you can deploy your application and services to the Kubernetes cluster. The cluster manages the availability and connectivity of the application and service.
In OpenShift Container Platform, you do not have to setup Kubernetes separately. For more information about Kubernetes and OpenShift Container Platform, see OpenShift Container Platform documentation.

TIBCO BusinessEvents provides a `readme.html` file at *BE_HOME*`\cloud\kubernetes\OpenShift\cache` for the Dockerized FraudDetectionCache application. You can follow the instruction in the `readme.html` file to run the application by the using the sample YAML files. These sample YAML files are available at *BE_HOME*`\cloud\kubernetes\OpenShift\cache\persistence-none` for deploying TIBCO BusinessEvents application with no backing store on OpenShift Container Platform. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications without Backing Store.

Prerequisites

Your TIBCO BusinessEvents application must be uploaded to the OpenShift Docker registry, see Pushing Application Docker Image to OpenShift Container Registry.

Procedure

1. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing Kubernetes objects in a YAML file, see the Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications without Backing Store.

2. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   Syntax:

   ```
   oc create -f <kubernetes_object.yaml>
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Step 1.

   ```
   oc create -f bediscoverynode.yaml

   oc create -f bediscovery-service.yaml

   oc create -f becacheagent.yaml

   oc create -f beinferenceagent.yaml

   oc create -f befdservice.yaml
   ```

3. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   oc logs <pod>
   ```

   For example, use the `oc get` command to get the list of pods and then use the `oc logs` command to view logs of `bediscoverynode`.

   ```
   oc get pods

   oc logs bediscoverynode-86d75d5fbc-z9gqt
   ```

4. Get the external IP of your application, which you can use to connect to the cluster.

   Syntax:

   ```
   oc get services <external_service_name>
   ```

   For example,

   ```
   oc get service befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionCache example application with no backing store, you can use the provided sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\OpenShift\cache` to test the application. Provide the external IP obtained to the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application then update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running the Application with Shared Nothing Persistence on OpenShift Container Platform

After uploading your TIBCO BusinessEvents application Docker image with shared nothing persistence to the OpenShift Docker registry, you can deploy your application and services to the Kubernetes cluster. The cluster manages the availability and connectivity of the application and service.

In OpenShift Container Platform, you do not have to setup Kubernetes separately. For more information about Kubernetes and OpenShift Container Platform, see OpenShift Container Platform documentation.

TIBCO BusinessEvents provides a `readme.html` file at *BE_HOME*`\cloud\kubernetes\OpenShift\cache` for the Dockerized FraudDetectionStore application. You can follow the instruction in the `readme.html` file to run the application by the using the provided sample YAML files. These sample YAML files are available at *BE_HOME*`\cloud\kubernetes\OpenShift\cache\shared-nothing` for deploying TIBCO BusinessEvents application with shared nothing persistence on OpenShift Container Platform. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.

Prerequisites

Your TIBCO BusinessEvents application must be uploaded to the OpenShift Docker registry, see Pushing Application Docker Image to OpenShift Container Registry.

Procedure

1. Create the persistent volume folders with NFS on the master node and make them accessible from remote server. For details about sharing the folder, refer to your operating system documentation. For example, the following steps create a new folder on the system and make it accessible from remote server.
   a) Create a new directory `pv001` and change its permission to read, write, and execute.
   ```
   mkdir -p /home/data/pv001
   chmod -R 777 /home/data/
   ```
   b) Edit the `/etc/exports` file and add the following entry for the new folder.
   ```
   /home/data/pv0001 *(rw,sync)
   ```
   c) Export the file system to the remote server which can mount the folder and use it as local file system. If you are connected to the remote server, do not mention the remote server URL in the command.
   ```
   exportfs -a
   ```
2. Create an object definition (`.yaml`) file for the persistent volume (PV) by using the folder path and URL of machine in which the folder was created.
3. Create other Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.

4. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.
   Syntax:
   ```
   oc create -f <kubernetes_object.yaml>
   ```
   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence .
   ```
   oc create -f persistentvol.yaml

   oc create -f becacheagent.yaml

   oc create -f bediscovery-service.yaml

   oc create -f beinferenceagent.yaml

   oc create -f befdservice.yaml
   ```
5. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.
   Syntax:
   ```
   oc logs <pod>
   ```
   For example, use the `oc get` command to get the list of pods and then use the `oc logs` command to view logs of `becacheagent`.
   ```
   oc get pods

   oc logs becacheagent-86d75d5fbc-z9gqt
   ```
6. Get the external IP of your application which you can use to connect to the cluster.
   Syntax:
   ```
   oc get services <external_service_name>
   ```
   For example,
   ```
   oc get service befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with shared nothing persistence, you can use the provided sample `readme.html` file at `BE_HOME\cloud\kubernetes\OpenShift\cache` to test the application. Provide the external IP obtained to the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application, then update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running the Application with Shared All Persistence on OpenShift Container Platform

After uploading your TIBCO BusinessEvents application Docker image with shared all storage to the OpenShift Docker registry, you can deploy your application and services to the Kubernetes cluster. The cluster manages the availability and connectivity of the application and service.

In OpenShift Container Platform, you do not have to setup Kubernetes separately. For more information about Kubernetes and OpenShift Container Platform, see OpenShift Container Platform documentation.

TIBCO BusinessEvents provides a `readme.html` file at `BE_HOME\cloud\kubernetes\OpenShift\cache` for the Dockerized FraudDetectionStore application. You can follow the instruction in the `readme.html` file to run the application by the using the provided sample YAML files. These sample YAML files are available at `BE_HOME\cloud\kubernetes\OpenShift\cache\shared-all` for deploying TIBCO BusinessEvents application with shared all persistence on OpenShift Container Platform. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.

As a sample use case, the procedure uses MySQL databases. The supported databases are MySQL, MariaDB, and PostgeSQL.

Prerequisites

Your TIBCO BusinessEvents application must be uploaded to the OpenShift Docker registry, see Pushing Application Docker Image to OpenShift Container Registry.

Procedure

1. Set up database with OpenShift Container Platform by using the CentOS based MySQL Docker image in Kubernetes.

   For details, see OpenShift Container Platform documentation.
2. Connect to the database by using the port forwarding.

   ```
   oc port-forward <mysql_pod_name> <port_number>:<port_number>
   ```
3. Run generated SQL scripts, such as `initialize_database_mysql.sql`, `create_tables_mysql.sql`, and the project schema specific SQL script (see *TIBCO BusinessEvents Configuration Guide*.
4. Set up the provisioner for the MySQL database by creating the persistent volume and PVC.

   For details about the sample YAML files for persistent volume and PVC, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.
5. Create a configMap resource with database details.

   You can use the Kubernetes command to enter into the pod container and then use Linux commands get the database URL.
6. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.
7. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   Syntax:

   ```
   oc create -f <kubernetes_object.yaml>
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared All Persistence.

   ```
   oc create -f mysql.yaml

   oc create -f mysql-service.yaml

   oc create -f persistent-volume-and-claim.yaml

   oc create -f db-configmap.yaml

   oc create -f bediscoverynode.yaml

   oc create -f bediscovery-service.yaml

   oc create -f becacheagent.yaml

   oc create -f beinferenceagent.yaml

   oc create -f befdservice.yaml
   ```
8. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   oc logs <pod>
   ```

For example, use the `oc get` command to get the list of pods and then use the `oc logs` command to view logs of `bediscoverynode`.

```
oc get pods

oc logs bediscoverynode-86d75d5fbc-z9gqt
```

9. Get the external IP of your application which you can use to connect to the cluster.

   Syntax:

   ```
   oc get services <external_service_name>
   ```

   For example,

   ```
   oc get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with shared all persistence, you can use the provided sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\OpenShift\cache` to test the application. Provide the external IP obtained to the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application then update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Monitoring TIBCO BusinessEvents Applications on OpenShift Container Platform

To monitor TIBCO BusinessEvents applications running on OpenShift Container Platform based Kubernetes, run TIBCO BusinessEvents Enterprise Administrator Agent container in the same Kubernetes namespace.

For TIBCO BusinessEvents Enterprise Administrator Agent, you can build only Linux containers (and not Windows containers).

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization
- Docker image of TIBCO Enterprise Administrator server. For instructions, see `readme.md` at *TEA_HOME*`/docker` in the TIBCO Enterprise Administrator installation.
- An TIBCO BusinessEvents application running on OpenShift Container Platform based Kubernetes, see Running an Application on OpenShift Based Kubernetes Cluster

Procedure

1. Build the TIBCO BusinessEvents Enterprise Administrator Agent Docker image by using the script provided by TIBCO BusinessEvents.

   See Building TIBCO BusinessEvents Enterprise Administrator Agent Docker Image.
2. Push Docker images of TIBCO BusinessEvents Enterprise Administrator Agent and TIBCO Enterprise Administrator server to OpenShift Container Registry.

   For details, see Pushing Application Docker Image to OpenShift Container Registry.
3. Run the TIBCO Enterprise Administrator server on OpenShift Container Platform based Kubernetes.
   For instructions, refer `readme.md` at *TEA_HOME*`/docker` in the TIBCO Enterprise Administrator installation.
4. Update the following Kubernetes object specification (`.yaml`) files for TIBCO BusinessEvents Enterprise Administrator Agent:

- `beteagentdeploymemt.yaml` - A deployment of TIBCO BusinessEvents Enterprise Administrator Agent Docker image with the TIBCO Enterprise Administrator server URL and login details.
- `beteagentinternalservice.yaml` - An internal service for connecting to TIBCO BusinessEvents Enterprise Administrator Agent from other nodes
- `k8s-authorization.yaml` - A ClusterRoleBinding for binding roles to the user.

These object specification files are available at `*BE_HOME*\cloud\kubernetes\*<cloud_name>*\tea`. For details about describing a Kubernetes object in a YAML file, see Kubernetes Documentation. For details about the sample YAML files, see Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent.

5. Create Kubernetes objects required for deploying and running TIBCO BusinessEvents Enterprise Administrator Agent by using the object specification (`.yaml`) files.

   Syntax:

   ```
   oc create -f <kubernetes_object.yaml>
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent.

   ```
   oc create -f k8s-authorization.yaml

   oc create -f beteagentdeploymemt.yaml

   oc create -f beteagentinternalservice.yaml
   ```

6. (Optional) If required, you can check the logs of TIBCO BusinessEvents Enterprise Administrator Agent pod.

   Syntax:

   ```
   oc logs <pod>
   ```

   For example, use the `oc get` command for a list of pods and then use the `oc logs` command to view the logs of `beteagentdeploymemt`.

   ```
   oc get pods

   oc logs beteagentdeploymemt-86d75d5fbc-z9gqt
   ```

What to do next

Launch TIBCO Enterprise Administrator in a web browser by using the external IP and port obtained from the TIBCO Enterprise Administrator external service.

For more details about the functioning of TIBCO BusinessEvents Enterprise Administrator Agent, see *TIBCO BusinessEvents Administration*..

Running the RMS on OpenShift Container Platform

To use TIBCO BusinessEvents WebStudio in OpenShift Container Platform, you must set up TIBCO BusinessEvents and Rule Management Server (RMS) in OpenShift Container Platform based Kubernetes.
In OpenShift Container Platform, you do not have to setup Kubernetes separately. For more information about Kubernetes and OpenShift Container Platform, see OpenShift Container Platform documentation.

TIBCO BusinessEvents provides a `readme.html` file at `*BE_HOME*\cloud\kubernetes\OpenShift\rms` for the Dockerized CreditCardApplication project and RMS project. You can follow the instruction in the `readme.html` file to run CreditCardApplication by the using the provided sample YAML files. These sample YAML files are available at `*BE_HOME*\cloud\kubernetes\OpenShift\rms` for deploying CreditCardApplication on OpenShift Container Platform. For details about these sample YAML files, see Sample Kubernetes YAML Files for RMS.

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization
- You must have installed and login to OpenShift Container Platform CLI, see Setting Up the OpenShift CLI Environment.

Procedure

1. To enable hot deployment for a project, add JMX connection details for each project for each environment under the **HotDeploy** section in the `RMS.cdd` file. :

   For example,

   ```
    <property name="ProjectName.ws.applicableEnvironments" type="string"
   value="QA,PROD"/>
    <property name="ProjectName.QA.ws.jmx.hotDeploy.enable" type="boolean"
   value="true"/>
    <property name="ProjectName.QA.ws.jmx.host" type="string" value="bejmx-
   service.default.svc.cluster.local"/>
    <property name="ProjectName.QA.ws.jmx.port" type="integer" value="5555"/>
    <property name="ProjectName.QA.ws.jmx.user" type="string" value=""/>
    <property name="ProjectName.QA.ws.jmx.password" type="string" value=""/>
    <property name="ProjectName.QA.ws.jmx.clusterName"
   value="CreditCardApplication"/>
    <property name="ProjectName.QA.ws.jmx.agentName" value="inference-class"/>
   ```

   For more information about hot deployment property group, see the "RMS Server Configuration Property Reference" section in *TIBCO BusinessEvents WebStudio Users Guide*.

2. Build the RMS Docker image. See Building RMS Docker Image.

3. In the RMS application CDD file, update the path for hot deployment of artifacts to the shared location in RMS. For example:

   ```
   <property name="be.engine.cluster.externalClasses.path" value="C:/
   tibco/be/5.6/rms/shared/CreditCardApplication/Decision_Tables"/>
   <property name="be.cluster.ruletemplateinstances.deploy.dir" value="C:/
   tibco/be/5.6/rms/shared/CreditCardApplication/RTI/"/>
   ```

4. Build the RMS application Docker image. See Building TIBCO BusinessEvents Application Docker Image.

5. Tag and push the RMS and application Docker images to the OpenShift Docker container registry. For details, see Pushing Application Docker Image to OpenShift Container Registry.

   For example:

   ```
   $ oc describe -n default service/docker-registry
   Name:              docker-registry
   Namespace:         default
   Labels:            <none>
   Annotations:       <none>
   Selector:          docker-registry=default
   Type:              ClusterIP
   IP:                192.0.2.0
   Port:              5000-tcp  5000/TCP
   TargetPort:        5000/TCP
   Endpoints:         198.51.100.10:5000
   Session Affinity:  ClientIP
   Events:            <none>
   $ docker tag rms:5.6.1  192.0.2.0:5000/test-project/rms:5.6.1
   $ docker tag creditcardapp:01  192.0.2.0:5000/test-project/creditcardapp:01
   $ docker login 192.0.2.0:5000 -u userid -p
   ov40AhpTXYZOwHBtC_vat0SF4xJd8lQNjylccs8ZOLc
   $ docker push 192.0.2.0:5000/test-project/rms:5.6.1
   $ docker push 192.0.2.0:5000/test-project/creditcardapp:01
   ```

6. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   You must consider the following point while creating the object specification file:

- Create separate persistent volume and PVCs for storing the project's hot deployed artifacts, project shared files, projects ACLs, and email notifications.
- Create a node with RMS container and an internal service for connecting it to the cluster.
- Create discovery node, cache agent, and inference agent by using the application Docker image.

For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about sample YAML files, see Sample Kubernetes YAML Files for RMS.

7. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   Syntax:

   ```
   oc create -f <kubernetes_object.yaml>
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Step 1.

   ```
   oc create -f bejmx-service.yaml

   oc create -f berms-persistent-volumes.yaml

   oc create -f berms-persistent-volume-claims.yaml

   oc create -f berms.yaml

   oc create -f berms-service.yaml

   oc create -f bediscoverynode.yaml

   oc create -f bediscovery-service.yaml

   oc create -f becacheagent.yaml

   oc create -f beinferenceagent.yaml

   oc create -f befdservice.yaml
   ```

8. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   oc logs <pod>
   ```

   For example, use the `oc get` command to get the list of pods and then use the `oc logs` command to view logs of `bediscoverynode`.

   ```
   oc get pods

   oc logs bediscoverynode-86d75d5fbc-z9gqt
   ```

9. Copy the masked persistent volume folders to the same path in the container. When you mount the PVC to the RMS project folder, it mask the other existing projects at the same path in the container.

   Use the following command for copying required folders in RMS pods.

   Syntax:

   ```
     oc cp <host_folder_path berms_pod_name>:<berms_pod_folder_path>
   ```

   For example:

   ```
   oc cp security berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/rms/config/

   oc cp notify berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/rms/config/

   oc cp shared berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/rms/

   oc cp WebStudio berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/examples/standard/
   ```

10. Get the external IP of the RMS service.

    Syntax:

    ```
    oc get services <external_service_name>
    ```

For example,

```
oc get service rms-service
```

What to do next

Use the IP obtained to connect to TIBCO BusinessEvents WebStudio from your browser. For example, if you have deployed the CreditCardApplication example application, you can use the provided sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\OpenShift\rms` to test the application. Provide the external IP obtained to the `readme.html` file and follow the instructions in it to run the application.

TIBCO BusinessEvents on Microsoft Azure Based Kubernetes

You can run any TIBCO BusinessEvents application on Microsoft Azure based Kubernetes cluster and monitor them by using TIBCO BusinessEvents Enterprise Administrator Agent. You can also manage business rules through WebStudio by running RMS on Microsoft Azure based Kubernetes cluster.

Readme for Sample Applications

TIBCO BusinessEvents provides `readme.html` files for running the sample applications and components on Microsoft Azure. You can follow the instruction in the `readme.html` file to run the application, WebStudio, and TIBCO BusinessEvents Enterprise Administrator Agent by using the provided sample YAML files.

The following table lists location of `readme.html` and sample YAML files for running sample applications and other components:

Scenario	readme.html and Sample YAML Files Location
Running TIBCO BusinessEvents application (FraudDetection) without cache on Microsoft Azure	*BE_HOME*`\cloud\kubernetes\Azure\inmemory`
Running TIBCO BusinessEvents application (FraudDetectionCache and FraudDetectionStore) with cache on Microsoft Azure	*BE_HOME*`\cloud\kubernetes\Azure\cache`
Running TIBCO BusinessEvents WebStudio on Microsoft Azure	*BE_HOME*`\cloud\kubernetes\Azure\rms`
Running TIBCO BusinessEvents Enterprise Administration Agent for monitoring TIBCO BusinessEvents applications on Microsoft Azure	*BE_HOME*`\cloud\kubernetes\Azure\tea`

Topics

- Running an Application on Microsoft Azure Based Kubernetes Cluster
- Monitoring TIBCO BusinessEvents Applications on Microsoft Azure
- Running RMS on Azure Based Kubernetes

Running an Application on Microsoft Azure Based Kubernetes Cluster

By using the Azure Kubernetes Service (AKS), you can easily deploy an TIBCO BusinessEvents application in the Kubernetes cluster managed by Microsoft Azure.
For more details about the AKS, see Azure Kubernetes Service documentation.

Prerequisites

- Docker image of the TIBCO BusinessEvents application, see Building TIBCO BusinessEvents Application Docker Image.
- You must have a Microsoft Azure account with an active subscription. If you don't, create a new Azure account.

Procedure

1. Set up Microsoft Azure command line environment.
2. Create an Azure Container Registry (ACR) and push the Docker image of the application to it, see Setting Up an Azure Container Registry.
3. Create a Kubernetes cluster and deploy it to the Microsoft Azure, see Setting Up a Kubernetes Cluster on AKS.
4. Based on your application architecture, deploy the application on the Kubernetes cluster. See the following topics based on your application persistence option:
   - Running the Application Without Backing Store on Azure.
   - Running an Application with Shared Nothing Persistence on Azure.
   - Running an Application with Shared All Persistence on Azure.

Setting up the Microsoft Azure CLI Environment

You can use either the Microsoft Azure Cloud Shell or Microsoft Azure command-line interface (CLI) for running the Microsoft Azure commands. In the following sections, the procedures are provided for the Azure CLI.

Prerequisites

You must have a Microsoft Azure account with an active subscription. If required, create a new Azure account.

Procedure

1. Install the Microsoft Azure command-line interface (CLI). For installation instructions, see Microsoft Azure CLI documentation
2. In Microsoft Azure CLI, sign in to Microsoft Azure by using the `login` command.
   ```
   az login
   ```
   The CLI opens a browser and loads the sign-in page.
3. Sign in with your account credentials in the browser.
   For details, see Get started with Azure CLI.

What to do next

Create an Azure Container Registry (ACR) and push the Docker image of the application to it, see Setting Up an Azure Container Registry.

Setting Up an Azure Container Registry

Microsoft Azure uses the Azure Container Registry for securely building and deploying your applications. To create an Azure Container Registry, you need to create an Azure Resource group. An Azure resource group is a logical container into which Azure resources are deployed and managed. For more information about commands used in the following procedure, see Microsoft Azure CLI documentation.

Prerequisites

- Set up the Microsoft Azure command line environment.
- Docker image of the TIBCO BusinessEvents application that you want to deploy to the Kubernetes cluster, see Building TIBCO BusinessEvents Application Docker Image.

Procedure

1. Create a resource group by using the `az group create` command.
   ```
   az group create --name <resource_group_name> --location <location>
   ```
2. Create an Azure Container Registry instance in your resource group by using the `az acr create` command.
   ```
   az acr create --resource-group <resource_group_name> --name <registry_name> --
   sku Basic --admin-enabled true
   ```
3. Login to the container registry created earlier by using the `az acr login` command.
   ```
   az acr login --name <registry_name>
   ```
   The command returns a `Login Succeeded` message once completed.
4. To use the TIBCO BusinessEvents application container image with Azure Container Registry, tag the image with the login server address of your registry.
   a) View the list of your local image by using the docker images command.
   ```
   $ docker images

   REPOSITORY                    TAG                  IMAGE ID
   CREATED                 SIZE
   fdcache                       latest               4675398c9172        13
   minutes ago       694MB
   ```
   b) Get the login server address for the Azure Container Registry by using the `az acr list` command.
   ```
   az acr list --resource-group <resource_group_name> --query "[].
   {acrLoginServer:loginServer}" --output table
   ```
   c) Tag your application image with login server address of your registry from the earlier step. This creates an alias of the application image with a fully qualifies path to your registry.
   ```
   docker tag fdcache <registry_login_server>/fdcache:01
   ```
   d) Verify the tags applied to the image by running the `docker images` command again.
   ```
   $ docker images

   REPOSITORY                                TAG                  IMAGE
   ID             CREATED              SIZE
   mycontainerregistry.azuecr.io/fdcache    01
   4675398c9172        13 minutes ago      694MB
   ```
5. Push the application image to your container registry by using the `docker push` command.
   ```
   docker push <registry_login_server>/fdcache:01
   ```
6. Validate if the image is uploaded to your registry.
   ```
   az acr repository list --name <registry_login_server> --output table
   ```

What to do next

After you have created an Azure Container Registry and pushed an image to the registry, deploy the Kubernetes cluster on Microsoft Azure, see Setting Up a Kubernetes Cluster on AKS.

Setting Up a Kubernetes Cluster on AKS

Azure Kubernetes Services (AKS) manages the Kubernetes environment and provides options to quickly deploy Kubernetes cluster.

Prerequisites

Set up the Azure Container Registry and push the application Docker image to it, see Setting Up an Azure Container Registry.

Procedure

1. To enable a Kubernetes cluster to interact with other Azure resource, an Azure Active Directory service principal is required.
   a) Create a service principal by using the `az ad sp create-for-rbac` command.
   ```
   az ad sp create-for-rbac --skip-assignment
   ```
   The output of the command provides the `appId` which is the service principal and `password` which is the `client-secret` for creating the Kubernetes cluster.
   b) Create the Kubernetes cluster with repository group and service principal created earlier.
   ```
   az aks create --orchestrator-type=kubernetes --resource-group
   <resource_group_name> --name=<cluster_name> --service-principal
   <service_principal> --client-secret <client_secret> --node-count <node_count>
   --generate-ssh-keys
   ```
   Microsoft Azure creates a storage account when a Kubernetes cluster is created.

   For more information about commands, see Microsoft Azure CLI documentation.
2. To connect to Kubernetes cluster from your local computer, use `kubectl`, the Kubernetes CLI. If you use the Azure Cloud Shell, `kubectl` is already installed. You can also install it locally by using the `az aks install-cli` command.
   ```
   az aks install-cli
   ```
3. Configure `kubectl` to connect your Kubernetes cluster by using the `az aks get-credentials` command.
   ```
   az aks kubernetes get-credentials --resource-group <resource_group_name> --
   name=<cluster_name>
   ```
4. Verify the connection to your Kubernetes cluster by using the `kubectl get nodes` command.
   ```
   kubectl get nodes
   ```

What to do next

Based on your application architecture, deploy the application on the Kubernetes cluster:

- For deployment of application for No Backing Store cluster, see Running the Application Without Backing Store on Azure.
- For deployment of application for Shared Nothing persistence, see Running an Application with Shared Nothing Persistence on Azure.
- For deployment of application for Shared All persistence, see Running an Application with Shared All Persistence on Azure.

Running the Application Without Backing Store on Azure

After uploading your TIBCO BusinessEvents application image with no backing store to the Azure Container Registry and creating the Kubernetes cluster, you can deploy your application and services to the Kubernetes cluster. The cluster manages the availability and connectivity of the application and service.
For more information about Kubernetes concepts and Microsoft Azure, see Azure Kubernetes Service documentation.

TIBCO BusinessEvents provides a `readme.html` file at *BE_HOME*`\cloud\kubernetes\Azure\cache` for the Dockerized FraudDetectionCache application. You can follow the instruction in the `readme.html` file to run the application by the using the provided sample YAML files. These sample YAML files are available at *BE_HOME*`\cloud\kubernetes\Azure\cache\persistence-none` for deploying TIBCO BusinessEvents application with no backing store on Microsoft Azure. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications without Backing Store.

Prerequisites

- Your TIBCO BusinessEvents application must be uploaded to the Azure Container Registry, see Setting Up an Azure Container Registry.
- The Kubernetes cluster must be deployed in the Microsoft Azure, see Setting Up a Kubernetes Cluster on AKS.

Procedure

1. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing Kubernetes objects in a YAML file, see the Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications without Backing Store.
2. Create Kubernetes objects required for deploying and running the application by using object specification (`.yaml`) files.

   Syntax:
   ```
   kubectl create -f <kubernetes_object.yaml>
   ```
   For example, create the following Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications without Backing Store.
   ```
   kubectl create -f bediscoverynode.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f becacheagent.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```
3. (Optional) If required, you can also check logs of TIBCO BusinessEvents pods.

   Syntax:
   ```
   kubectl logs <pod>
   ```
   For example, use the `kubectl get` command to get list of pods and then use the `kubectl logs` command to view logs of `bediscoverynode`.
   ```
   kubectl get pods

   kubectl logs bediscoverynode-86d75d5fbc-z9gqt
   ```
4. Get the external IP of your application which you can then use to connect to the cluster.

   Syntax:
   ```
   kubectl get services <external_service_name>
   ```
   For example,
   ```
   kubectl get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionCache example application without backing store, you can use the sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\Azure\cache` to test the application. Use the external IP that you have obtained in the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application, update its `readme.html` file to test that application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running an Application with Shared Nothing Persistence on Azure

After uploading your TIBCO BusinessEvents application image with shared nothing persistenceto the Azure Container Registry and creating the Kubernetes cluster, you can deploy your application to the Kubernetes cluster. The cluster manages the availability and connectivity of the application. Microsoft Azure also provide storage options to store and retrieve data.

Microsoft Azure provides two storage options for persistent volumes:

- *Azure Disks* - available for access to single node with the ReadWriteOnce privilege.
- *Azure Files* - available for access to multiple nodes and pods.

For more information about Kubernetes concepts and Microsoft Azure, see Azure Kubernetes Service documentation.

TIBCO BusinessEvents also provides a `readme.html` file at *BE_HOME*`\cloud\kubernetes\Azure\cache` for the Dockerized FraudDetectionStore application. You can follow the instruction in the `readme.html` file to run the application by the using the provided sample YAML files. These sample YAML files are available at *BE_HOME*`\cloud\kubernetes\azure\cache\shared-nothing\<`*azure_storage_type*`>` for deploying TIBCO BusinessEvents application with shared nothing persistence on Microsoft Azure. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.

Prerequisites

- Your TIBCO BusinessEvents application must be uploaded to the Azure Container Registry, see Setting Up an Azure Container Registry.
- The Kubernetes cluster must be deployed in the Microsoft Azure, see Setting Up a Kubernetes Cluster on AKS.

Procedure

1. Create Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.

2. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object_spec>.yaml
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence for Azure file storage.

   ```
   kubectl create -f manifest.yaml

   kubectl create -f becacheagent.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```

3. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   kubectl logs <pod>
   ```

For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `becacheagent`.

```
kubectl get pods

kubectl logs becacheagent-86d75d5fbc-z9gqt
```

4. Get the external IP of your application, which you can use to connect to the cluster.

   Syntax

   ```
   kubectl get services <external_service_name>
   ```

   For example,

   ```
   kubectl get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with shared nothing persistence, you can use the provided sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\Azure\cache` to test the application. Provide the external IP obtained to the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application then update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running an Application with Shared All Persistence on Azure

After uploading your TIBCO BusinessEvents application image with shared all persistence to the Azure Container Registry and creating the Kubernetes cluster, you can deploy your application to the Kubernetes cluster. The cluster manages the availability and connectivity of the application. You can use Microsoft Azure provided relational database service or you can use the Docker image of the database that you want to use.

For more information about Kubernetes concepts and Microsoft Azure, see Azure Kubernetes Service documentation.

The following procedure provides a sample implementation of Azure Database for MySQL as the database service. For more information, see Azure Database for MySQL documentation.

If you want to use any other database service, follow its documentation on how to use with Docker and Kubernetes.

TIBCO BusinessEvents also provides a `readme.html` file at *BE_HOME*`\cloud\kubernetes\Azure\cache` for the Dockerized FraudDetectionStore application. You can follow the instruction in the `readme.html` file to run the application by the using the provided sample YAML files. These sample YAML files are available at *BE_HOME*`\cloud\kubernetes\azure\cache\shared-all\`*<database_type>* for deploying TIBCO BusinessEvents application with shared nothing persistence on Microsoft Azure. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.

Prerequisites

- Your TIBCO BusinessEvents application must be uploaded to the Azure Container Registry, see Setting Up an Azure Container Registry.
- The Kubernetes cluster must be deployed in the Microsoft Azure, see Setting Up a Kubernetes Cluster on AKS.

Procedure

1. Create an Azure Database instance of MySQL server with the `az mysql server create` command.

   ```
   az mysql server create -g <resource_group_name> -n <mysql_server_name> -l
   <location> --admin-user <admin_user> --admin-password <admin_password> --sku-
   name <sku_name>
   ```

   For details about command options, see Azure Database for MySQL documentation.

2. Create a MySQL server-level firewall rule and disable the SSL connection to connect to the server from your local MySQL client.

   ```
   az mysql server firewall-rule create --resource-group <resource_group_name> --
   server <mysql_server_name> --name <firewall_rule_name> --start-ip-address
   <start_ip_address> --end-ip-address <end_ip_address>

   az mysql server update --resource-group <resource_group_name> --name
   <mysql_server_name> --ssl-enforcement Disabled
   ```

   For details about command options, see Azure Database for MySQL documentation.

3. To connect to the MySQL server, get the host information and access credentials.

   ```
   az mysql server show --resource-group <resource_group_name> --name
   <mysql_server_name>
   ```

   For details about command options, see Azure Database for MySQL documentation.

4. Use the `mysql` command-line tool to establish a connection to your Azure Database for MySQL server by using the earlier obtained host and credentials details, see mysql command-line documentation.

5. Initialize the database, create the user, create the table, and load the data in the tables by using the MySQL commands.

   For details, see MySQL documentation.

6. Create Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.

7. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object_spec>.yaml
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared All Persistence:

   ```
   kubectl create -f db-configmapmysql.yaml

   kubectl create -f bediscoverynode.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f becacheagent.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```

8. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   kubectl logs <pod>
   ```

For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `bediscovery`.

```
kubectl get pods

kubectl logs bediscovery-86d75d5fbc-z9gqt
```

9. Get the external IP of your application which you can use to connect to the cluster.
   Syntax
   ```
   kubectl get services <external_service_name>
   ```
   For example,
   ```
   kubectl get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with shared all persistence, you can use the provided sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\Azure\cache` to test the application. Provide the external IP obtained to the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application then update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Monitoring TIBCO BusinessEvents Applications on Microsoft Azure

To monitor TIBCO BusinessEvents applications running on Microsoft Azure based Kubernetes, run TIBCO BusinessEvents Enterprise Administrator Agent container in the same Kubernetes namespace.

For TIBCO BusinessEvents Enterprise Administrator Agent, you can build only Linux containers (and not Windows containers).

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization
- Docker image of TIBCO Enterprise Administrator server. For instructions, see `readme.md` at *TEA_HOME*`/docker` in the TIBCO Enterprise Administrator installation.
- A TIBCO BusinessEvents application running on Microsoft Azure based Kubernetes, see Running an Application on Microsoft Azure Based Kubernetes Cluster.

Procedure

1. Build the TIBCO BusinessEvents Enterprise Administrator Agent Docker image by using the script provided by TIBCO BusinessEvents.
   See Building TIBCO BusinessEvents Enterprise Administrator Agent Docker Image.
2. Push Docker images of TIBCO BusinessEvents Enterprise Administrator Agent and TIBCO Enterprise Administrator server to Azure Container Registry.
   For details, see Setting Up an Azure Container Registry.
3. Run the TIBCO Enterprise Administrator server on Microsoft Azure based Kubernetes.
   For instructions, refer `readme.md` at *TEA_HOME*`/docker` in the TIBCO Enterprise Administrator installation.
4. Update the following Kubernetes object specification (`.yaml`) files for TIBCO BusinessEvents Enterprise Administrator Agent:

   - `beteagentdeploymemt.yaml` - A deployment of TIBCO BusinessEvents Enterprise Administrator Agent Docker image with the TIBCO Enterprise Administrator server URL and login details.

- `beteagentinternalservice.yaml` - An internal service for connecting to TIBCO BusinessEvents Enterprise Administrator Agent from other nodes
- `k8s-authorization.yaml` - A ClusterRoleBinding for binding roles to the user.

These object specification files are available at `*BE_HOME*\cloud\kubernetes\*<cloud_name>*\tea`. For details about describing a Kubernetes object in a YAML file, see Kubernetes Documentation. For details about the sample YAML files, see Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent.

5. Create Kubernetes objects required for deploying and running TIBCO BusinessEvents Enterprise Administrator Agent by using the object specification (`.yaml`) files.
   Syntax:
   ```
   kubectl create -f <kubernetes_object.yaml>
   ```
   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent.
   ```
   kubectl create -f k8s-authorization.yaml

   kubectl create -f beteagentdeploymemt.yaml

   kubectl create -f beteagentinternalservice.yaml
   ```
6. (Optional) If required, you can check logs of TIBCO BusinessEvents Enterprise Administrator Agent pod.
   Syntax:
   ```
   kubectl logs <pod>
   ```
   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `beteagentdeploymemt`.
   ```
   kubectl get pods

   kubectl logs beteagentdeploymemt-86d75d5fbc-z9gqt
   ```

What to do next

Launch TIBCO Enterprise Administrator in a web browser by using the external IP and port obtained from the TIBCO Enterprise Administrator external service.

For more details on functioning of TIBCO BusinessEvents Enterprise Administrator Agent, see TIBCO BusinessEvents Administration guide..

Running RMS on Azure Based Kubernetes

By using the Azure Kubernetes Service (AKS), you can easily deploy the rule management server(RMS) on the Kubernetes cluster managed by Microsoft Azure.
TIBCO BusinessEvents installation provides the RMS project at `*BE_HOME*\rms\project\BRMS`.
Deploying RMS on Azure based Kubernetes is similar to deploying any other TIBCO BusinessEvents application with no backing store. However, if you want to enable the hot deployment in RMS, you must create persistent volumes claims (PVCs) (Step 4) and setup JMX environment variables (Step 1).

Prerequisites

Set up the Microsoft Azure command line environment.

Procedure

1. To enable hot deployment, in the `RMS.cdd` file, add JMX connection details for each project for each environment under the **HotDeploy** section
   For example,
   ```
    <property name="ProjectName.ws.applicableEnvironments" type="string"
   value="QA,PROD"/>
   ```

```
 <property name="ProjectName.QA.ws.jmx.hotDeploy.enable" type="boolean"
value="true"/>
 <property name="ProjectName.QA.ws.jmx.host" type="string" value="bejmx-
service.default.svc.cluster.local"/>
 <property name="ProjectName.QA.ws.jmx.port" type="integer" value="5555"/>
 <property name="ProjectName.QA.ws.jmx.user" type="string" value=""/>
 <property name="ProjectName.QA.ws.jmx.password" type="string" value=""/>
 <property name="ProjectName.QA.ws.jmx.clusterName"
value="CreditCardApplication"/>
 <property name="ProjectName.QA.ws.jmx.agentName" value="inference-class"/>
```

For more information about hot deployment property group, see the "RMS Server Configuration Property Reference" section in *TIBCO BusinessEvents WebStudio Users Guide*.

Alternatively, you can add these JMX connection details for the project from the Settings page in TIBCO BusinessEvents WebStudio. For details, see *TIBCO BusinessEvents WebStudio Users Guide*.

2. Build the Docker image of RMS, see Building RMS Docker Image.
3. Create an Azure Container Registry (ACR) and push the Docker image of RMS to it, see Setting Up an Azure Container Registry.
4. Create a Kubernetes cluster and deploy it to the Microsoft Azure, see Setting Up a Kubernetes Cluster on AKS.
5. To store hot deployment artifacts, create Azure File type storage class and PVCs by using the Kubernetes object specification (`.yaml`) files.

   The sample YAML file `berms-peristent-volume-claims.yaml` for creating storage class and PVCs is located at *`BE_HOME`*`\cloud\kubernetes\Azure\rms`. The `berms-peristent-volume-claims.yaml` file set up PVC for the following storage purpose:

   - for storing TIBCO BusinessEvents project files
   - for storing TIBCO BusinessEvents project ACL files, such as, `CreditCardApplication.ac`
   - for storing RMS artifacts after hot deployment, such as, rule template instances
   - for storing Email notification files, such as, `message.stg`

   For more information about the Kubernetes object spec files, see Kubernetes documentation.
6. Create Kubernetes objects required for deploying RMS by using the object spec (`.yaml`) files.

   These objects include deployment and services for the cluster. Thus to deploy RMS on the Kubernetes cluster, create:

   - a discovery node (pod) to start the cluster
   - a service to connect to discovery node
   - a cache agent node which connects to the discovery node service
   - an inference agent node which connects to the discovery node service
   - a service to connect to the inference agent
   - a RMS node containing the RMS Docker image and persistent volume claims to mount the respective Azure File share
   - an external service to connect to the RMS node
   - an JMX service to connect to the JMX port of the RMS pod

   For your reference, sample YAML files for deploying RMS to Kubernetes are available at *`BE_HOME`*`\cloud\kubernetes\Azure\rms`, see Sample Kubernetes YAML Files for RMS.

   In these `.yaml` files, update the `image` tag with the application Docker image tag including the registry login server name that you have used in Step 4 of Setting Up an Azure Container Registry. Also, update the `DOCKER_HOST` environment variable with `bejmx-service.default.svc.cluster.local`.

   For more information about the Kubernetes object spec files, see Kubernetes documentation.

7. Use the `kubectl create` command to create and deploy these objects to the Kubernetes cluster. This command parses the specified manifest file and creates the defined Kubernetes objects.

   ```
   kubectl create -f <kubernetes_object_spec>.yaml
   ```

   For example, enter the following command to create the Azure File type storage class.

   ```
   kubectl create -f manifest_azurefile.yaml
   ```

8. After successful creation of Kubernetes objects, use the `kubectl cp` command to upload the files (required to perform various operations in WebStudio) from your computer to the PVCs.

   ```
   kubectl cp <host_folder_path> <berms_pod_name>:<berms_pod_folder_path>
   ```

   The following table lists the files to be uploaded to PVCs.

PVC Name	Files to be uploaded
`azurefile-webstudio`	*BE_HOME*`\examples\standard\WebStudio\`
`azurefile-security`	*BE_HOME*`\rms\config\security`
`azurefile-notify`	*BE_HOME*`\rms\config\notify`
`azurefile-shared`	*BE_HOME*`\rms\shared`

9. To access WebStudio, you can get the external IP of the service of the RMS deployment by using the `kubectl get services` command.

   ```
   kubectl get services <external_service_name>
   ```

   For example,

   ```
   kubectl get services berms-service -o wide
   ```

What to do next

Use the IP obtained to connect to TIBCO BusinessEvents WebStudio from your browser.

TIBCO BusinessEvents on AWS Based Kubernetes

You can run any TIBCO BusinessEvents application on Amazon Web Services (AWS) based Kubernetes cluster by using Amazon EC2 and monitor them by using TIBCO BusinessEvents Enterprise Administrator Agent. You can also manage business rules through WebStudio by running RMS on AWS based Kubernetes cluster by using Amazon EC2.

Topics

Running TIBCO BusinessEvents® on AWS Based Kubernetes Cluster

By using Amazon EC2, you can fully manage your Kubernetes deployment. You can provision and run Kubernetes on your choice of instance types.

Prerequisites

- Docker image of TIBCO BusinessEvents application, see Building TIBCO BusinessEvents Application Docker Image.
- Download and install the following CLIs on your system:

CLI	Download and Installation Instruction Link
`kops`	https://github.com/kubernetes/kops/blob/master/docs/aws.md
`kubectl`	https://kubernetes.io/docs/tasks/tools/install-kubectl/
`aws`	https://aws.amazon.com/cli/

Procedure

1. Set up a Kubernetes cluster on Amazon Web Services (AWS). For more information, see Setting up a Kubernetes Cluster on AWS.
2. Go to the EC2 Container Services dashboard and create a repository with the same name as the Docker image of TIBCO BusinessEvents application. Upload the BusinessEvents application image to the repository. For help you can use the **View Push Commands** button.

   AWS Repository name must be the same as the Docker image of TIBCO BusinessEvents application.

   For more information on how to create a repository in Amazon AWS, refer to https://docs.aws.amazon.com/AmazonECR/latest/userguide/repository-create.html.
3. Based on your application architecture, deploy the TIBCO BusinessEvents application on the Kubernetes cluster. See the following topics based on your application persistence option:
   - Running the Application Without Backing Store on AWS
   - Running an Application with Shared Nothing Persistence on AWS.
   - Running an Application with Shared All Persistence on AWS.

Setting up a Kubernetes Cluster on AWS

Set up a Kubernetes cluster with AWS for running TIBCO BusinessEvents® application.

Procedure

Creating Cluster

1. Create an Amazon Simple Storage Service (Amazon S3) storage to store the cluster configuration and state. You can use either AWS CLI or AWS console to create the storage.

   For more information about Amazon S3, see Amazon S3 Documentation.

   For example:

   ```
   aws s3 mb s3://be-bucket
   ```

2. Create the Kubernetes cluster on AWS using the `kops` CLI.

   For more information, see the kops CLI documentation.

   For example:

   ```
   kops create cluster --zones us-west-2a --master-zones us-west-2a --master-size
   t2.large --node-size t2.large --name becluster.k8s.local --state s3://<s3-bucket-
   name> --yes
   ```

   Where,

   - `s3-bucket-name` is the name of the S3 storage created earlier.
   - `becluster.k8s.local` is the name of the cluster being created. Use `k8s.local` prefix to identify a gossip-based Kubernetes cluster and you can skip the DNS configuration.

Validating Cluster

3. Validate your cluster using the `validate` command.

   ```
   kops validate cluster
   ```

   Node and master must be in ready state. The `kops` utility stores the connection information at `~/.kops/config`, and `kubectl` uses the connection information to connect to the cluster.

Running the Application Without Backing Store on AWS

After uploading your TIBCO BusinessEvents application image with no backing store to the AWS Registry and creating the Kubernetes cluster, you can deploy your application and services to the Kubernetes cluster. The cluster manages the availability and connectivity of the application and service.

Prerequisites

For deploying BusinessEvents cluster for No Backing Store on AWS, you must first set up Kubernetes cluster on AWS and then upload your Docker image on AWS. For more information, see Running TIBCO BusinessEvents® on AWS Based Kubernetes Cluster.

Procedure

1. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing Kubernetes objects in a YAML file, see the Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications without Backing Store.

2. Create Kubernetes objects required for deploying and running the application by using object specification (`.yaml`) files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object.yaml>
   ```

For example, create the following Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications without Backing Store.

```
kubectl create -f bediscoverynode.yaml

kubectl create -f bediscovery-service.yaml

kubectl create -f becacheagent.yaml

kubectl create -f beinferenceagent.yaml

kubectl create -f befdservice.yaml
```

3. (Optional) If required, you can also check logs of TIBCO BusinessEvents pods.
   Syntax:
   ```
   kubectl logs <pod>
   ```
   For example, use the `kubectl get` command to get list of pods and then use the `kubectl logs` command to view logs of `bediscoverynode`.
   ```
   kubectl get pods

   kubectl logs bediscoverynode-86d75d5fbc-z9gqt
   ```
4. Get the external IP of your application which you can then use to connect to the cluster.
   Syntax:
   ```
   kubectl get services <external_service_name>
   ```
   For example,
   ```
   kubectl get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionCache example application with no backing store, update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running an Application with Shared Nothing Persistence on AWS

By using the Kubernetes elements such as the StatefulSets object and dynamic volume provisioning features, you can create TIBCO ActiveSpaces and Shared Nothing deployments.

StatefulSets gives deterministic names to the pods. Along with dynamic volume provisioning, StatefulSets also give deterministic names to `PersistentVolumeClaims` (PVC). This ensures that when a particular member of a StatefulSet goes down and comes up again, it attaches itself to the same PVC. For more information about the Kubernetes concepts of StatefulSets, dynamic volume provisioning, and `PersistentVolumeClaims`, see Kubernetes documentation.

Prerequisites

Ensure that CDD of your application is configured to use shared nothing persistence. For deploying BusinessEvents cluster for the shared nothing persistence on AWS, you must first set up Kubernetes cluster on AWS and then upload your docker image to an AWS docker registry. For more information, see Running TIBCO BusinessEvents® on AWS Based Kubernetes Cluster.

Procedure

1. In AWS, create an EFS file system.
   For more information on the steps to create an EFS file system, see Amazon EFS documentation. Specify the Kubernetes cluster Virtual Private Cloud (**VPC**) and **Security Group** while creating a mount target for the file system. On the File Systems page, verify that the mount target shows the **Life Cycle State** as `Available`. Under **File system access**, you see the file system **DNS name**. Make a note of this DNS name.

After successful creation of the EFS file system, note its **File System ID**, which can be used for creating EFS provisioner.

2. Create the EFS provisioner and other associated resources. Specify all the connection setup values for the EFS file system in a `manifest.yaml` file and run the `kubectl` command to create the EFS provisioner.
   a) Download the sample `manifest.yaml` file from https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/aws/efs/deploy/manifest.yaml and edit it according to your setup.
   b) In the `configmap` section, specify **File System ID** of the newly created EFS as the value of the `file.system.id:` variable and **Availability Zone** of the newly created EFS as the value of the `aws.region:` variables.
   c) In the `Deployment` section, specify DNS name of the newly created EFS as the value of the `server:` variable.
   d) Run the `kubectl` command to apply the settings in `manifest.yaml`.
   ```
   kubectl apply -f manifest.yaml
   ```
   e) Ensure that the EFS provisioner pod is in the running state using the `kubectl` command.
   ```
   kubectl get pods
   ```
3. Create other Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.
4. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.
   ```
   kubectl create -f becacheagent.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```
5. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:
   ```
   kubectl logs <pod>
   ```
   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `becacheagent`.
   ```
   kubectl get pods

   kubectl logs becacheagent-86d75d5fbc-z9gqt
   ```
6. Get the external IP of your application, which you can use to connect to the cluster.

   Syntax
   ```
   kubectl get services <external_service_name>
   ```
   For example,
   ```
   kubectl get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with shared nothing persistence, update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running an Application with Shared All Persistence on AWS

After uploading your TIBCO BusinessEvents application Docker image with shared all storage to the AWS registry and creating Kubernetes cluster, you can deploy your application and services to the Kubernetes cluster. The cluster manages the availability and connectivity of the application and service.

Ensure that your application connection properties for database use global variables.

Prerequisites

- A Kubernetes cluster, see Setting up a Kubernetes Cluster on AWS.
- An AWS service registry. For instructions, see AWS documentation.

Procedure

1. Create an Amazon RDS based instance and configure it to connect to a TIBCO BusinessEvents supported database (Oracle, MySQL, DB2, and so on).

   For configuration details, see Amazon RDS documentation.

2. Create other Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes Documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.

3. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared All Persistence.

   ```
   kubectl create -f db-configmap.yaml

   kubectl create -f bediscoverynode.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f becacheagent.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```

4. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   kubectl logs <pod>
   ```

   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `bediscoverynode`.

   ```
   kubectl get pods

   kubectl logs bediscoverynode-86d75d5fbc-z9gqt
   ```

5. Get the external IP of your application which you can use to connect to the cluster.

   Syntax:

   ```
   kubectl get services <external_service_name>
   ```

   For example,

   ```
   kubectl get service befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with shared all persistence, update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Monitoring TIBCO BusinessEvents Applications on AWS

To monitor TIBCO BusinessEvents applications running on AWS based Kubernetes, run TIBCO BusinessEvents Enterprise Administrator Agent container in the same Kubernetes namespace.

For TIBCO BusinessEvents Enterprise Administrator Agent, you can build only Linux containers (and not Windows containers).

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization
- Docker image of TIBCO Enterprise Administrator server. For instructions, see `readme.md` at *`TEA_HOME`*`/docker` in the TIBCO Enterprise Administrator installation.
- An TIBCO BusinessEvents application running on AWS based Kubernetes, see Running TIBCO BusinessEvents® on AWS Based Kubernetes Cluster.

Procedure

1. Build the TIBCO BusinessEvents Enterprise Administrator Agent Docker image by using the script provided by TIBCO BusinessEvents.

   See Building TIBCO BusinessEvents Enterprise Administrator Agent Docker Image.
2. Push Docker images of TIBCO BusinessEvents Enterprise Administrator Agent and TIBCO Enterprise Administrator server to the AWS repository.

   For details, see Running TIBCO BusinessEvents® on AWS Based Kubernetes Cluster.
3. Run the TIBCO Enterprise Administrator server on AWS based Kubernetes.
   For instructions, refer `readme.md` at *`TEA_HOME`*`/docker` in the TIBCO Enterprise Administrator installation.
4. Update the following Kubernetes object specification (`.yaml`) files for TIBCO BusinessEvents Enterprise Administrator Agent:

   - `beteagentdeploymemt.yaml` - A deployment of TIBCO BusinessEvents Enterprise Administrator Agent Docker image with the TIBCO Enterprise Administrator server URL and login details.
   - `beteagentinternalservice.yaml` - An internal service for connecting to TIBCO BusinessEvents Enterprise Administrator Agent from other nodes
   - `k8s-authorization.yaml` - A ClusterRoleBinding for binding roles to the user.

   These object specification files are available at *`BE_HOME`*`\cloud\kubernetes\`*`<cloud_name>`*`\tea`. For details about describing a Kubernetes object in a YAML file, see Kubernetes Documentation. For details about the sample YAML files, see Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent.
5. Create Kubernetes objects required for deploying and running TIBCO BusinessEvents Enterprise Administrator Agent by using the object specification (`.yaml`) files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object.yaml>
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent .

   ```
   kubectl create -f k8s-authorization.yaml

   kubectl create -f beteagentdeploymemt.yaml
   ```

```
kubectl create -f beteagentinternalservice.yaml
```

6. (Optional) If required, you can check logs of TIBCO BusinessEvents Enterprise Administrator Agent pod.

   Syntax:

   ```
   kubectl logs <pod>
   ```

   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `beteagentdeploymemt`.

   ```
   kubectl get pods

   kubectl logs beteagentdeploymemt-86d75d5fbc-z9gqt
   ```

What to do next

Launch TIBCO Enterprise Administrator in a web browser by using the external IP and port obtained from the TIBCO Enterprise Administrator external service.

For more details on functioning of TIBCO BusinessEvents Enterprise Administrator Agent, see TIBCO BusinessEvents Administration guide.

Running RMS Applications in AWS Based Kubernetes

To use TIBCO BusinessEvents® WebStudio in Kubernetes cluster, you must set up TIBCO BusinessEvents and Rule Management Server (RMS) in AWS based Kubernetes.

Prerequisites

- Docker image of TIBCO BusinessEvents application, see Building TIBCO BusinessEvents Application Docker Image.
- Download and install the following CLIs on your system:

CLI	Download and Installation Instruction Link
`kops`	https://github.com/kubernetes/kops/blob/master/docs/aws.md
`kubectl`	https://kubernetes.io/docs/tasks/tools/install-kubectl/
`aws`	https://aws.amazon.com/cli/

Procedure

1. Set up a Kubernetes cluster on Amazon Web Services (AWS). For more information, see Setting up a Kubernetes Cluster on AWS.

2. In AWS, create an EFS file system.

   For more information on the steps to create an EFS file system, see Amazon EFS documentation. Specify the Kubernetes cluster Virtual Private Cloud (**VPC**) and **Security Group** while creating a mount target for the file system. On the File Systems page, verify that the mount target shows the **Life Cycle State** as `Available`. Under **File system access**, you see the file system **DNS name**. Make a note of this DNS name.

   After successful creation of the EFS file system, note its **File System ID**, which can be used for creating EFS provisioner.

3. Create the EFS provisioner and other associated resources. Specify all the connection setup values for the EFS file system in a `manifest.yaml` file and run the `kubectl` command to create the EFS provisioner.

a) Download the sample `manifest.yaml` file from https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/aws/efs/deploy/manifest.yaml and edit it according to your setup.

b) In the `configmap` section, specify **File System ID** of the newly created EFS as the value of the `file.system.id:` variable and **Availability Zone** of the newly created EFS as the value of the `aws.region:` variables.

c) In the `Deployment` section, specify DNS name of the newly created EFS as the value of the `server:` variable.

d) Run the `kubectl` command to apply the settings in `manifest.yaml`.

```
kubectl apply -f manifest.yaml
```

e) Ensure that the EFS provisioner pod is in the running state using the `kubectl` command.

```
kubectl get pods
```

4. As RMS is running in a Docker container, separate external storage must be set up for required files and artifacts. For this, create EFS based `PersistentVolumeClaim` (PVC) using the configuration files (YAML format).

   The sample YAML file `berms-efs-persistent-volume-claims.yaml` for creating storage class and PVCs is located at *BE_HOME*`\cloud\kubernetes\AWS\rms`. The `berms-efs-persistent-volume-claims.yaml` file set up the following PVC:

   Sample PVCs for RMS

PVC Name	Description
`efs-webstudio`	Set up the PVC for storing TIBCO BusinessEvents project files.
`efs-security`	Set up the PVC for storing TIBCO BusinessEvents project ACL files, such as, `CreditCardApplication.ac`.
`efs-shared`	Set up the PVC for storing RMS artifacts after hot deployment, such as, rule template instances.
`efs-notify`	Set up the PVC for storing Email notification files, such as, `message.stg`.

   For more information about the Kubernetes object spec files, see Kubernetes documentation.

5. Run the `create` command of `kubectl` utility using the YAML files to create PVCs on the EFS file system.

   For example, create PVCs using the sample files:

```
kubectl create -f berms-efs-persistent-volume-claims.yaml
```

6. Mount the EFS file system into the Kubernetes EC2 instance nodes.

   For more information on how to mount EFS file system on EC2 instance, refer AWS Documentation at https://docs.aws.amazon.com/efs/latest/ug/mounting-fs.html.

   After successful mounting, PVCs on EFS are available for uploading files.

7. Transfer files from your system to the respective PVCs.

   - RMS project files (for example, project files at *BE_HOME*`\examples\standard\WebStudio\`) to the project storage PVC (`efs-webstudio`)
   - RMS security files and project access control (`.ac`) files at *BE_HOME*`\rms\config\security` to the security storage PVC (`efs-security`).
   - RMS notification related files (for example, `message.stg` at *BE_HOME*`\rms\config\notify`) to the notify storage PVC (`efs-notify`).

For information on how to transfer files from your system to EC2 instance, refer to the *Amazon AWS Documentation* at https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html.

8. Build the RMS Docker image for deploying to Kubernetes.

   For every project add JMX configuration in `RMS.cdd`, for example:

```
 <property name="ProjectName.ws.applicableEnvironments" type="string"
value="QA,PROD"/>
 <property name="ProjectName.QA.ws.jmx.hotDeploy.enable" type="boolean"
value="true"/>
 <property name="ProjectName.QA.ws.jmx.host" type="string" value="bejmx-
service.default.svc.cluster.local"/>
 <property name="ProjectName.QA.ws.jmx.port" type="integer" value="5555"/>
 <property name="ProjectName.QA.ws.jmx.user" type="string" value=""/>
 <property name="ProjectName.QA.ws.jmx.password" type="string" value=""/>
 <property name="ProjectName.QA.ws.jmx.clusterName"
value="CreditCardApplication"/>
 <property name="ProjectName.QA.ws.jmx.agentName" value="inference-class"/>
```

   For every project configure `jmx.host` as the fully qualified name (FQN) of the local JMX Kubernetes service defined earlier, for example, `bejmx-service.default.svc.cluster.local`. Also, for every project configure `jmx.port` as the JMX port number defined in the JMX Kubernetes service, for example, `5555`.

9. Create RMS Docker image. For more information, see Building RMS Docker Image.

10. Go to the EC2 Container Services dashboard and create a repository with the same name as the RMS Docker image. Upload the RMS image to the repository. You can use the **View Push Commands** button to view how to do that.

    AWS Repository name should be same as the RMS Docker image.

    For more information on how to create a repository in Amazon AWS, refer to https://docs.aws.amazon.com/AmazonECR/latest/userguide/repository-create.html.

11. Define an internal JMX Kubernetes service using the supplied sample YAML configuration file `bejmx-service.yaml`.

    RMS server uses this service to connect to the JMX port of a BusinessEvents pod.

12. Create other Kubernetes object specification (`.yaml`) files based on your deployment requirement.

    These resources include deployment and services for the cluster. Thus, to deploy a BusinessEvents cluster, create the following YAML files:

    - Discovery node (pod) to start the cluster.
    - Service to connect to the discovery node.
    - Cache agent node that connects to the discovery node service.
    - Inference agent node that connects to the discovery node service.
    - Service to connect to the inference agent.
    - RMS node containing RMS docker image.
    - External service to connect to the RMS node.

    For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for RMS.

13. Create Kubernetes objects required for deploying and running the application and RMS by using the object specification (`.yaml`) files.

    For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for RMS:

```
kubectl create -f bejmx-service.yaml

kubectl create -f berms.yaml
kubectl create -f berms-service.yaml

kubectl create -f bediscoverynode.yaml

kubectl create -f bediscovery-service.yaml

kubectl create -f becache.yaml

kubectl create -f beinferenceagent.yaml

kubectl create -f beinference-service.yaml
```

14. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

    Syntax:

    ```
    kubectl logs <pod>
    ```

    For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `bediscoverynode`.

    ```
    kubectl get pods

    kubectl logs bediscoverynode-86d75d5fbc-z9gqt
    ```

15. Get the external IP of your application which you can use to connect to the cluster.

    Syntax:

    ```
    kubectl get services <external_service_name>
    ```

    For example,

    ```
    kubectl get services berms-service
    ```

What to do next

Use the IP obtained to connect to TIBCO BusinessEvents WebStudio from your browser.

TIBCO BusinessEvents on Amazon EKS Based Kubernetes

You can run any TIBCO BusinessEvents application on Amazon Elastic Kubernetes Service (Amazon EKS) and monitor the application by using TIBCO BusinessEvents Enterprise Administrator Agent. You can also manage business rules in TIBCO BusinessEvents WebStudio by running RMS on the AWS based Kubernetes cluster.

For details, see Amazon EKS Documentation.

Readme for Sample Applications

The `readme.html` files that are provided with TIBCO BusinessEvents contain information about running the sample applications and components on Amazon EKS. You can follow the instruction in the `readme.html` files to run the application, WebStudio, and TIBCO BusinessEvents Enterprise Administrator Agent by using the sample YAML files.

The following table lists location of `readme.html` and sample YAML files for running sample applications and other components:

Scenario	Readme.html and Sample YAML Files Location
Running TIBCO BusinessEvents application (FraudDetection) without cache on AWS	*BE_HOME*`\cloud\kubernetes\AWS\inmemory`
Running TIBCO BusinessEvents application (FraudDetectionCache and FraudDetectionStore) with cache on AWS	*BE_HOME*`\cloud\kubernetes\AWS\cache`
Running TIBCO BusinessEvents WebStudio on AWS	*BE_HOME*`\cloud\kubernetes\AWS\rms`
Running TIBCO BusinessEvents Enterprise Administration Agent for monitoring TIBCO BusinessEvents applications on AWS	*BE_HOME*`\cloud\kubernetes\AWS\tea`

TIBCO BusinessEvents on Pivotal Based Kubernetes

You can run any TIBCO BusinessEvents application on a Pivotal based Kubernetes cluster and monitor the application by using TIBCO BusinessEvents Enterprise Administrator Agent. You can also manage your business rules in TIBCO BusinessEvents WebStudio by running RMS on the Pivotal based Kubernetes cluster.

You can use Enterprise Pivotal Container Service (PKS) on Google Cloud Platform (GCP) to deploy applications on Kubernetes. For details, see "Enterprise Pivotal Container Service" in Pivotal Docs.

Readme for Sample Applications

TIBCO BusinessEvents provides `readme.html` files to run the sample applications on a Pivotal based Kubernetes cluster by using Enterprise PKS on GCP. To run your application, RMS, and TIBCO BusinessEvents Enterprise Administrator Agent by using sample YAML files, follow the instructions given in the `readme.html` files.

The following table lists common scenarios and location of respective `readme.html` and sample YAML files for running sample applications and other components:

Scenario	readme.html and Sample YAML Files Location
Running a TIBCO BusinessEvents application (FraudDetection) without cache	*BE_HOME*`\cloud\kubernetes\PKS\inmemory`
Running a TIBCO BusinessEvents application (FraudDetectionCache and FraudDetectionStore) with cache	*BE_HOME*`\cloud\kubernetes\PKS\cache`
Running TIBCO BusinessEvents WebStudio	*BE_HOME*`\cloud\kubernetes\PKS\rms`
Running TIBCO BusinessEvents Enterprise Administration Agent for monitoring TIBCO BusinessEvents applications	*BE_HOME*`\cloud\kubernetes\PKS\tea`

Topics

- Running an Application in Enterprise PKS Installed on GCP
- Monitoring TIBCO BusinessEvents Applications on Enterprise PKS
- Running RMS on Enterprise PKS

Running an Application in Enterprise PKS Installed on GCP

By using Enterprise PKS installed on GCP, you can deploy a TIBCO BusinessEvents application in the Kubernetes cluster managed by Enterprise PKS.
For more information, see "Enterprise Pivotal Container Service" in Pivotal Docs.

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization
- Docker image of your TIBCO BusinessEvents application. See Building TIBCO BusinessEvents Application Docker Image.

- Active Pivotal and Google Cloud accounts
- Download and install the following CLIs on your system:

CLI	Download and Installation Reference
Enterprise PKS CLI (`pks`)	"Installing the PKS CLI" in Pivotal Docs
Kubernetes CLI (`kubectl`)	"Installing the Kubernetes CLI" in Pivotal Docs
Google Cloud CLI (`gcloud`)	Google Cloud SDK documentation

Procedure

1. Install Enterprise PKS on GCP.
   See Enterprise PKS documentation.
2. Set up a Kubernetes cluster on Enterprise PKS.
   See Setting Up a Kubernetes Cluster with Enterprise PKS.
3. Push the TIBCO BusinessEvents application Docker image to Google Container Registry.
   See Setting up Google Container Registry.
4. Based on your application architecture, deploy the application on the Kubernetes cluster. See the following topics based on your application persistence option:
   - Running an Application without Backing Store on Enterprise PKS
   - Running an Application with Shared Nothing Persistence on Enterprise PKS
   - Running an Application with Shared All Persistence on Enterprise PKS

Setting Up a Kubernetes Cluster with Enterprise PKS

In Pivotal, you can use Enterprise PKS to create and manage a Kubernetes cluster. Use the Enterprise PKS Command Line Interface (PKS CLI) to deploy the Kubernetes cluster and manage its lifecycle. To deploy and manage container-based workloads on the Kubernetes cluster, use the Kubernetes CLI (`kubectl`).

Prerequisites

- Enterprise PKS installed on GCP. For details, see "Enterprise Pivotal Container Service" in Pivotal Docs.
- Download and install the following CLIs on your system:

CLI	Download and Installation Reference
Enterprise PKS CLI (`pks`)	"Installing the PKS CLI" in Pivotal Docs
Kubernetes CLI (`kubectl`)	"Installing the Kubernetes CLI" in Pivotal Docs

Procedure

1. Create a Kubernetes cluster with PKS CLI.

   See "Managing Clusters" in Pivotal Docs.

   Sample `pks` command syntax:

   ```
   pks login -a <pks_api> -u <username> -p <password>

   pks create-cluster <cluster_name> --external-hostname <hostname> --plan <plan-
   name> --num-nodes <worker-nodes>

   pks cluster <cluster-name>
   ```
2. Get access to the Kubernetes cluster for managing it from the Kubernetes CLI.

   Sample `pks` command syntax:

   ```
   pks get-credentials <cluster-name>
   ```
3. Verify access to the Kubernetes cluster by using the Kubernetes CLI.

   Sample `kubectl` command syntax:

   ```
   kubectl cluster-info
   ```

Setting up Google Container Registry

For deploying the TIBCO BusinessEvents application to the Kubernetes cluster, you must push the application Docker image to Google Container Registry.
Alternatively, you can also use VMware Harbor Registry to store and manage application Docker images for your Enterprise PKS deployment. For details, see Pivotal Docs.

To access Docker images in the Google Container Registry from an environment other than GCP, set up a secret object (containing the credential information) for Kubernetes. You can use this secret object in Kubernetes object specification (YAML) files for your deployment.

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization
- Docker image of your TIBCO BusinessEvents application. See Building TIBCO BusinessEvents Application Docker Image.
- Download and install the following CLIs on your system:

CLI	Download and Installation Reference
Kubernetes CLI (`kubectl`)	"Installing the Kubernetes CLI" in Pivotal Docs
Google Cloud CLI (`gcloud`)	Google Cloud SDK documentation

Procedure

1. Retrieve the project ID of the default project of your Google Cloud account.

   Sample `gcloud` command:

   ```
   gcloud config list core/project
   ```
2. To push the TIBCO BusinessEvents application Docker image to the Google Container Registry, first tag it with the registry name and then push it. For details, see the Google Container Registry documentation.

Sample command syntax:

```
docker tag <source-image> <hostname>/<project-id>/<image-name>

docker push <hostname>/<project-id>/<image-name>
```

3. Set up the Kubernetes secret object for pulling Docker images from Google Container Registry.
   a) Create a Google cloud service account and store its key in a JSON file. See the Cloud Identity and Access Management documentation.

      Sample gcloud commands syntax:

      ```
      gcloud iam service-accounts create <service-account-name>

      gcloud iam service-accounts keys create ~/key.json --iam-account <service-
      account-name>@<project-id>.iam.gserviceaccount.com
      ```
   b) Add an IAM policy binding for the defined project and service account. See the Cloud Identity and Access Management documentation.

      Sample gcloud commands syntax:

      ```
      gcloud projects add-iam-policy-binding <project-id> --
      member=serviceAccount:<service-account-name>@<project-
      id>.iam.gserviceaccount.com --role=<role>
      ```
   c) Create the Kubernetes secret object by using the JSON file you have just created. See Kubernetes documentation.

      Sample kubectl commands syntax:

      ```
      kubectl create secret docker-registry <secret-name> --docker-server=<hostname>
      --docker-username=_json_key --docker-email=<email_id> --docker-
      password=<password>
      ```

      You can add this secret object in Kubernetes configuration (YAML) files of your deployments by using the `ImagePullSecrets` field.
   d) Add the secret object to the default service account. See Kubernetes Documentation.

      ```
      kubectl patch serviceaccount default -p "{\"imagePullSecrets\": [{\"name\":
      \"<secret-name>\"}]}"
      ```

Running an Application without Backing Store on Enterprise PKS

After uploading the Docker image of your TIBCO BusinessEvents application without backing store to Google Container Registry, you can deploy your application to the Pivotal based Kubernetes cluster. The cluster manages the availability and connectivity of the application and service.

TIBCO BusinessEvents provides a `readme.html` file at *BE_HOME*`\cloud\kubernetes\PKS\cache` for the Dockerized FraudDetectionCache application. You can follow the instructions given in the `readme.html` file to run the application by the using sample YAML files. These sample YAML files are available at *BE_HOME*`\cloud\kubernetes\PKS\cache\persistence-none` for deploying TIBCO BusinessEvents application without backing store on Enterprise PKS. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications without Backing Store.

Prerequisites

- The Kubernetes cluster must be deployed on Enterprise PKS. See Setting Up a Kubernetes Cluster with Enterprise PKS.
- Your TIBCO BusinessEvents application Docker image must be uploaded to the Google Container Registry. See Setting up Google Container Registry.

Procedure

1. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing Kubernetes objects in a YAML file, see the Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications without Backing Store.

2. Create Kubernetes objects required for deploying and running the application by using object specification (`.yaml`) files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object.yaml>
   ```

   For example, create the following Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications without Backing Store.

   ```
   kubectl create -f bediscoverynode.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f becacheagent.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```

3. (Optional) If required, you can also check logs of TIBCO BusinessEvents pods.

   Syntax:

   ```
   kubectl logs <pod>
   ```

   For example, use the `kubectl get` command to get list of pods and then use the `kubectl logs` command to view logs of `bediscoverynode`.

   ```
   kubectl get pods

   kubectl logs bediscoverynode-86d75d5fbc-z9gqt
   ```

4. Get the external IP of your application which you can then use to connect to the cluster.

   Syntax:

   ```
   kubectl get services <external_service_name>
   ```

   For example,

   ```
   kubectl get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionCache example application without backing store, you can use the sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\PKS\cache` to test the application. Use the obtained external IP in the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application, update its `readme.html` file to test that application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running an Application with Shared Nothing Persistence on Enterprise PKS

After uploading your TIBCO BusinessEvents application Docker image with shared nothing persistence to Google Container Registry and creating the Kubernetes cluster, deploy and run your application on the Kubernetes cluster. The cluster manages the availability and connectivity of the application.

For shared nothing persistence, create a Storage Class of GCP persistent disk in the Kubernetes cluster. The `StorageClass` is configured in the `manifest.yaml` file. You can use this `StorageClass` to provision persistent volumes for the cache and inference agents. For more information, see "Storage Classes" in Kubernetes Documentation.

To run the Dockerized FraudDetectionStore application with shared nothing persistence, follow the instructions in the `readme.html` file at *BE_HOME*`\cloud\kubernetes\PKS\cache`. Typically, sample YAML files are provided to run the application. For this deployment, the sample YAML files are available at *BE_HOME*`\cloud\kubernetes\PKS\cache\shared-nothing`. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.

Prerequisites

- The Kubernetes cluster must be deployed on Enterprise PKS. See Setting Up a Kubernetes Cluster with Enterprise PKS.
- Your TIBCO BusinessEvents application Docker image must be uploaded to the Google Container Registry. See Setting up Google Container Registry.

Procedure

1. Create Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.
2. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   Syntax:
   ```
   kubectl create -f <kubernetes_object_spec>.yaml
   ```
   For example, create the following Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence.
   ```
   kubectl create -f manifest.yaml

   kubectl create -f becacheagent.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```
3. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:
   ```
   kubectl logs <pod>
   ```
   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `becacheagent`.
   ```
   kubectl get pods

   kubectl logs becacheagent-86d75d5fbc-z9gqt
   ```
4. Get the external IP of your application, which you can use to connect to the cluster.

   Syntax
   ```
   kubectl get services <external_service_name>
   ```
   For example,
   ```
   kubectl get services befdservice
   ```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with the shared nothing persistence, you can use the sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\PKS\cache` to test the application. Use the obtained external IP in the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application then update its `readme.html` file to test the application. Update the server address in the application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Running an Application with Shared All Persistence on Enterprise PKS

After uploading your TIBCO BusinessEvents application image with shared all persistence to the Google Container Registry and creating the Kubernetes cluster, you can deploy your application to the Kubernetes cluster. The cluster manages the availability and connectivity of the application. You can use the Docker image of the database that you want to use.

For the shared all persistence, configure a database server and define a storage for its database. For example, you can use a MySQL server for database connections. To implement the database, you can use the `mysql` container from the Docker hub. Provide the database connection details in the cache and inference agent configuration files. Define a Storage Class object of the GCP persistent disk (`gce-pd`) provisioner to provision storage (persistent volume claim) for the MySQL database.

TIBCO BusinessEvents provides a `readme.html` file at *`BE_HOME`*`\cloud\kubernetes\PKS\cache` for the Dockerized FraudDetectionStore application. You can follow the instructions in the `readme.html` file to run the application by the using the sample YAML files. These sample YAML files for deploying TIBCO BusinessEvents application with the shared all persistence on Enterprise PKS are available at *`BE_HOME`*`\cloud\kubernetes\PKS\cache\shared-all`. For details about these sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.

Procedure

1. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   For details about describing a Kubernetes object in a YAML file, see Kubernetes documentation. For details about the sample YAML files, see Sample Kubernetes YAML Files for Applications with Shared All Persistence.

2. Create Kubernetes objects required for deploying and running the application by using the object specification (`.yaml`) files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object_spec>.yaml
   ```

   For example, create the following Kubernetes objects by using the sample YAML files mentioned in Sample Kubernetes YAML Files for Applications with Shared All Persistence.

   ```
   kubectl create -f persistent-volume-claim.yaml

   kubectl create -f mysql.yaml

   kubectl create -f mysql-service.yaml

   kubectl create -f db-configmap.yaml

   kubectl create -f bediscoverynode.yaml

   kubectl create -f bediscovery-service.yaml

   kubectl create -f becacheagent.yaml

   kubectl create -f beinferenceagent.yaml

   kubectl create -f befdservice.yaml
   ```

3. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   kubectl logs <pod>
   ```

   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `becacheagent`.

   ```
   kubectl get pods

   kubectl logs becacheagent-86d75d5fbc-z9gqt
   ```

4. Get the external IP of your application, which you can use to connect to the cluster.

Syntax

```
kubectl get services <external_service_name>
```

For example,

```
kubectl get services befdservice
```

What to do next

Test the application by using the external IP obtained. For example, if you have deployed the FraudDetectionStore example application with the shared all persistence, you can use the sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\PKS\cache` to test the application. Use the obtained external IP in the `readme.html` file and follow the instructions in it to run the application.

However, if you have deployed any other sample application then update its `readme.html` file to test the application. Update the server address in application `readme.html` file from `localhost` to the external IP obtained. Now, follow the instructions in the `readme.html` file for testing the application.

Monitoring TIBCO BusinessEvents Applications on Enterprise PKS

To monitor TIBCO BusinessEvents applications running on Enterprise PKS, run TIBCO BusinessEvents Enterprise Administrator Agent container in the same Kubernetes namespace as the application.

You can build only Linux container (and not Windows container) of TIBCO BusinessEvents Enterprise Administrator Agent.

Prerequisites

- See Preparing for TIBCO BusinessEvents Containerization.
- Docker image of TIBCO Enterprise Administrator server. For instructions, see `readme.md` at *TEA_HOME*`/docker` in the TIBCO Enterprise Administrator installation.
- A TIBCO BusinessEvents application running on Enterprise PKS. See Running an Application in Enterprise PKS Installed on GCP.

Procedure

1. Build the TIBCO BusinessEvents Enterprise Administrator Agent Docker image by using the script provided by TIBCO BusinessEvents.
   See Building TIBCO BusinessEvents Enterprise Administrator Agent Docker Image.
2. Push Docker images of TIBCO BusinessEvents Enterprise Administrator Agent and TIBCO Enterprise Administrator server to Google Container Registry.
   For details, see Setting up Google Container Registry.
3. Run the TIBCO Enterprise Administrator server on Enterprise PKS.
   For instructions, see `readme.md` at *TEA_HOME*`/docker` in the TIBCO Enterprise Administrator installation.
4. Update the following Kubernetes object specification (`.yaml`) files for TIBCO BusinessEvents Enterprise Administrator Agent:

   - `beteagentdeploymemt.yaml` - A deployment of TIBCO BusinessEvents Enterprise Administrator Agent Docker image with the TIBCO Enterprise Administrator server URL and login details.
   - `beteagentinternalservice.yaml` - An internal service for connecting to TIBCO BusinessEvents Enterprise Administrator Agent from other nodes
   - `k8s-authorization.yaml` - A ClusterRoleBinding for binding roles to the user.

   These object specification files are available at *BE_HOME*`\cloud\kubernetes\`*`<cloud_name>`*`\tea`. For details about describing a Kubernetes object in a YAML file, see Kubernetes Documentation. For details about the sample YAML files, see Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent.

5. Create Kubernetes objects required for deploying and running TIBCO BusinessEvents Enterprise Administrator Agent by using the YAML files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object.yaml>
   ```

   For example, create the Kubernetes objects by using the sample YAML files mentioned in Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent.

   ```
   kubectl create -f k8s-authorization.yaml

   kubectl create -f beteagentdeploymemt.yaml

   kubectl create -f beteagentinternalservice.yaml
   ```

6. (Optional) If required, you can check logs of TIBCO BusinessEvents Enterprise Administrator Agent pod.

   Syntax:

   ```
   kubectl logs <pod>
   ```

   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `beteagentdeploymemt`.

   ```
   kubectl get pods

   kubectl logs beteagentdeploymemt-86d75d5fbc-z9gqt
   ```

What to do next

Launch TIBCO Enterprise Administrator in a web browser by using the external IP and port obtained from the TIBCO Enterprise Administrator external service.

For more details on functioning of TIBCO BusinessEvents Enterprise Administrator Agent, see TIBCO BusinessEvents Administration guide..

Running RMS on Enterprise PKS

To use TIBCO BusinessEvents WebStudio on Pivotal, you must set up TIBCO BusinessEvents and Rule Management Server (RMS) on Pivotal by using Enterprise PKS.

To connect to WebStudio, you must set up an external service to deploy the RMS project. The RMS deployment can communicate with an instance of TIBCO BusinessEvents by using the instance JMX port. You can implement it by setting up an internal JMX service Kubernetes object.

For storing project's hot deployed artifacts, project shared files, project ACLs, and email notifications, define a Storage Class object of the GCP persistent disk (`gce-pd`) provisioner. You can use this Storage Class to provision persistence volumes claims required for storage of these artifacts. For more information about Storage Class and persistent volume claims, see Kubernetes Documentation.

TIBCO BusinessEvents provides a `readme.html` file at *BE_HOME*`\cloud\kubernetes\PKS\rms` for the Dockerized CreditCardApplication project and RMS project. You can follow the instructions in the `readme.html` file to run CreditCardApplication by using sample YAML files. These sample YAML files for deploying CreditCardApplication on Enterprise PKS are available at *BE_HOME*`\cloud\kubernetes\PKS\rms`. For details about these sample YAML files, see Sample Kubernetes YAML Files for RMS.

Prerequisites

The Kubernetes cluster must be deployed on Enterprise PKS, see Setting Up a Kubernetes Cluster with Enterprise PKS.

Procedure

1. To enable hot deployment in RMS, add JMX connection details in `RMS.cdd` for each project of each environment under the **HotDeploy** section.

   For example,

   ```
    <property name="ProjectName.ws.applicableEnvironments" type="string"
   value="QA,PROD"/>
    <property name="ProjectName.QA.ws.jmx.hotDeploy.enable" type="boolean"
   value="true"/>
    <property name="ProjectName.QA.ws.jmx.host" type="string" value="bejmx-
   service.default.svc.cluster.local"/>
    <property name="ProjectName.QA.ws.jmx.port" type="integer" value="5555"/>
    <property name="ProjectName.QA.ws.jmx.user" type="string" value=""/>
    <property name="ProjectName.QA.ws.jmx.password" type="string" value=""/>
    <property name="ProjectName.QA.ws.jmx.clusterName"
   value="CreditCardApplication"/>
    <property name="ProjectName.QA.ws.jmx.agentName" value="inference-class"/>
   ```

   For more information about hot deployment property group, see the "RMS Server Configuration Property Reference" section in *TIBCO BusinessEvents WebStudio Users Guide*.

   Alternatively, you can add these JMX connection details for the project from the **Settings** page in TIBCO BusinessEvents WebStudio. For details, see *TIBCO BusinessEvents WebStudio Users Guide*.

2. Build the RMS Docker image. See Building RMS Docker Image.

3. In the RMS application CDD file, update the path for hot deployment of artifacts to the shared location in RMS.

   For example, update the following properties for the CreditCardApplication sample:

   ```
   <property name="be.engine.cluster.externalClasses.path" value="C:/
   tibco/be/5.6/rms/shared/CreditCardApplication/Decision_Tables"/>
   <property name="be.cluster.ruletemplateinstances.deploy.dir" value="C:/
   tibco/be/5.6/rms/shared/CreditCardApplication/RTI/"/>
   ```

4. Build the RMS application Docker image. See Building TIBCO BusinessEvents Application Docker Image.

5. Tag and push the RMS and application Docker images to Google Container Registry. For details, see Setting up Google Container Registry.

6. Create the Kubernetes object specification (`.yaml`) files based on your deployment requirement.

   You must consider the following point while creating the object specification file:

   - Create separate persistent volume and persistent volume claims for storing the project hot deployed artifacts, project shared files, project ACLs, and email notifications.
   - Create a node with the RMS container, an internal JMX service for connecting it to the cluster, and an external RMS service for accessing WebStudio.
   - Create discovery node, cache agent, and inference agent by using the application Docker image.

   For details about defining Kubernetes objects in a YAML file, see Kubernetes documentation. For details about sample YAML files, see Sample Kubernetes YAML Files for RMS.

7. Create Kubernetes objects required for deploying and running the application by using the YAML files.

   Syntax:

   ```
   kubectl create -f <kubernetes_object.yaml>
   ```

   For example, create the following Kubernetes objects by using the sample YAML files in the previous step.

   ```
   kubectl create -f manifest_gcp.yaml

   kubectl create -f persistent-volume-claims.yaml
   ```

```
kubectl create -f berms.yaml

kubectl create -f berms-service.yaml

kubectl create -f bejmx-service.yaml

kubectl create -f bediscoverynode.yaml

kubectl create -f bediscovery-service.yaml

kubectl create -f becacheagent.yaml

kubectl create -f beinferenceagent.yaml

kubectl create -f befdservice.yaml
```

8. (Optional) If required, you can check logs of TIBCO BusinessEvents pod.

   Syntax:

   ```
   kubectl logs <pod>
   ```

   For example, use the `kubectl get` command to get the list of pods and then use the `kubectl logs` command to view logs of `becacheagent`.

   ```
   kubectl get pods

   kubectl logs becacheagent-86d75d5fbc-z9gqt
   ```

9. Copy the masked persistent volume folders to the same path in the container. When you mount the persistent volume claims to the RMS project folder, it masks the other existing projects that are at the same path in the container.

   Use the following command to copy the relevant folders to the RMS pods.

   Syntax:

   ```
   kubectl cp <host_folder_path berms_pod_name>:<berms_pod_folder_path>
   ```

   For example:

   ```
   kubectl cp security berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/rms/config/

   kubectl cp notify berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/rms/config/

   kubectl cp shared berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/rms/

   kubectl cp webstudio berms-65f89dff4-cwg6z:/opt/tibco/be/5.6.1/examples/standard/
   ```

10. Get the external IP of the RMS service.

    Syntax:

    ```
    kubectl get services <external_service_name>
    ```

    For example,

    ```
    kubectl get services rms-service
    ```

What to do next

Use the IP obtained from `rms-service` to connect to TIBCO BusinessEvents WebStudio from your browser. For example, if you have deployed the CreditCardApplication example application, you can use the provided sample `readme.html` file at *BE_HOME*`\cloud\kubernetes\PKS\rms` to test the application. Use the obtained external IP in the `readme.html` file and follow the instructions in it to run the application.

TIBCO BusinessEvents on Minikube Based Kubernetes

You can try out any TIBCO BusinessEvents application locally on a Kubernetes cluster by using the Minikube client and monitor them by using TIBCO BusinessEvents Enterprise Administrator Agent. You can also manage business rules through WebStudio by running RMS on Minikube based Kubernetes cluster.

For details about the Minikube client, see Kubernetes Documentation.

Readme for Sample Applications

TIBCO BusinessEvents provides `readme.html` files to help you in running the sample applications and components on Minikube. You can follow the instruction in the `readme.html` file to run the application, WebStudio, and TIBCO BusinessEvents Enterprise Administrator Agent by using the provided sample YAML files.

The following table lists location of `readme.html` and sample YAML files for running sample applications and other components:

Scenario	readme.html and Sample YAML Files Location
Running TIBCO BusinessEvents application (FraudDetection) without cache on Minikube	*BE_HOME*`\cloud\kubernetes\minikube\inmemory`
Running TIBCO BusinessEvents application (FraudDetectionCache and FraudDetectionStore) with cache on Minikube	*BE_HOME*`\cloud\kubernetes\minikube\cache`
Running TIBCO BusinessEvents WebStudio on Minikube	*BE_HOME*`\cloud\kubernetes\minikube\rms`
Running TIBCO BusinessEvents Enterprise Administration Agent for monitoring TIBCO BusinessEvents applications on Minikube	*BE_HOME*`\cloud\kubernetes\minikube\tea`

Appendix: Sample YAML Files for Kubernetes Cluster

TIBCO BusinessEvents provides sample YAML files to create Kubernetes objects for different cloud platforms and different persistence options. These YAML files are available at `BE_HOME/cloud/kubernetes/`.

Topics

Sample Kubernetes YAML Files for Applications without Backing Store

TIBCO BusinessEvents provides sample YAML files for deploying the TIBCO BusinessEvents application without a backing store at `BE_HOME\cloud\kubernetes\<cloud_name>\cache\persistence-none`.

If you are using global variable group in YAML files, instead of the slash '`/`' delimiter between global variable group name and global variable name, use "_gv_". For example, `port` is a global variable which is part of the `VariableGP` global variable group, then instead of using `VariableGP/port` in YAML files, use `VariableGP_gv_port`. Also, ensure that you do not use the "`gv`" token in any global variable group name or global variable name.

The following tables list Kubernetes object specification files provided for cloud platforms. For details on the Kubernetes objects used and YAML files, see the Kubernetes documentation.

Sample Kubernetes Resource YAML Files for No Backing Store

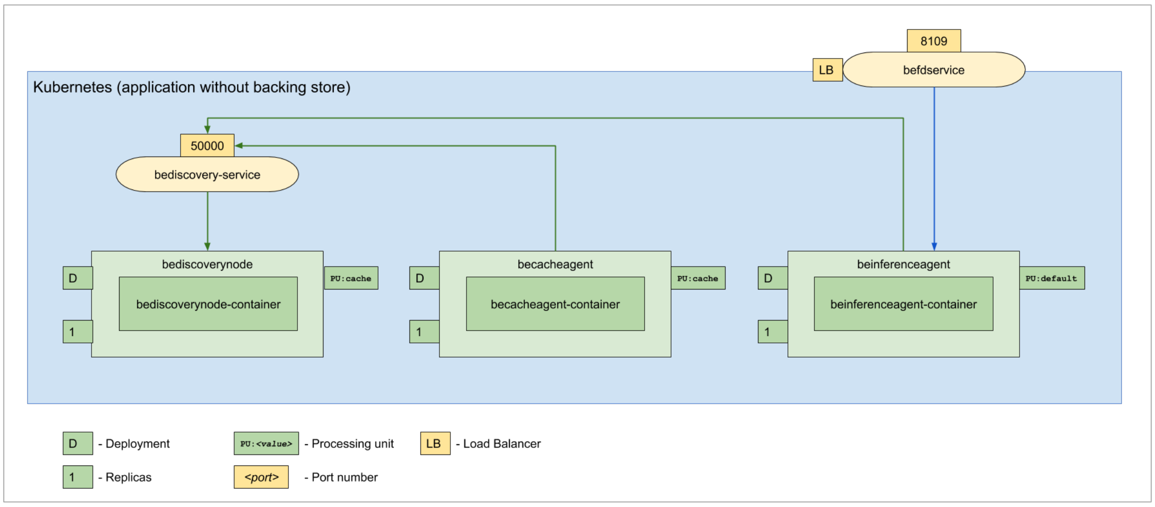
File Name	Resource	Resource Type	Description
`bediscoverynode.yaml`	Discovery node	Deployment	Set up the discovery node with the application Docker container for starting the cluster. Specify the application Docker image to create the container. This label which is used by the discovery node service as `selector`. Specify only one replica of the discovery node.
`bediscovery-service.yaml`	Discovery node service	Service (Internal)	Set up an internal service for connecting non-discovery nodes of the cluster to the discovery node. Specify the label of the discovery node as `selector`. Specify `protocol` and `port` that is used by other nodes to connect to this service.

File Name	Resource	Resource Type	Description
`becacheagent.yaml`	Cache agent node	Deployment	Set up the cache agent node with the application Docker container for the cluster. Specify the application Docker image to create the container. Specify `replicas` value based on the number of cache agent you want to start. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service.
`beinferenceagent.yaml`	Inference agent node	Deployment	Set up an inference agent with the application Docker container for connecting to external APIs. Specify the application Docker image to create the container. Provide a label to the deployment which the inference agent service can use as `selector`. Specify at least one replica of the inference agent node. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service.
`befdservice.yaml`	Inference agent service	Service (LoadBalancer/ External)	Set up an external service to connect to the inference agent. Specify label of the inference agent as `selector` for the service. Specify the `protocol` and `port` to connect to this service externally.

Kubernetes Cluster Diagram

The following diagram shows the connections between different Kubernetes objects defined by using the sample YAML files (listed in the previous table) for deploying TIBCO BusinessEvents application without backing store.

Kubernetes Cluster Diagram for an Application without Backing Store



Sample Kubernetes YAML Files for Applications with Shared Nothing Persistence

TIBCO BusinessEvents provides sample YAML files at `BE_HOME\cloud\kubernetes\<cloud_name>\cache\shared-nothing` for deploying TIBCO BusinessEvents application with shared nothing persistence.

If you are using global variable group in YAML files, instead of the slash '/' delimiter between global variable group name and global variable name, use "_gv_". For example, `port` is a global variable which is part of the `VariableGP` global variable group, then instead of using `VariableGP/port` in YAML files, use `VariableGP_gv_port`. Also, ensure that you do not use the "gv" token in any global variable group name or global variable name.

The following tables list Kubernetes object specification files provided for cloud platforms. For details on the Kubernetes objects used and YAML files, see the Kubernetes documentation.

Sample Object Definition Files for Storage

Cloud Platform	File Name	Resource Type	Description
OpenShift Container Platform	`persistentvol.yaml`	PersistentVolume	Set up the persistent volume for the agent in the Kubernetes cluster. Specify the name for the persistent volume, which the cache agent and inference agent uses to create persistent volume claims (PVCs). Specify the amount of storage to be allocated to this volume. This sample file uses the NFS plugin for the volume. Provide path of the folder that you have created, and server URL for mounting that folder.
Microsoft Azure	`manifest.yaml` (for Azure disk storage)	StorageClass	Set up the persistent volume for the Azure File provisioner type. Provide the mount options based on the Kubernetes version. For details, see Microsoft Azure documentation.
Amazon Web Services (AWS)	`manifest.yaml`	ConfigMap Deployment StorageClass	Set up the `ConfigMap` with the EFS file system details. Also, set up the deployment to create a container with the EFS provisioner and persistent volume.
Enterprise PKS	`manifest.yaml`	StorageClass	Set up a `StorageClass` of the GCP persistent disk in the Kubernetes cluster. Set the `provisioner` field as `kubernetes.io/gce-pd`. You can use this `StorageClass` to provision persistence volume claims for the cache and inference agents. For more information, see Storage Classes in Kubernetes.

Sample Kubernetes Common Resource YAML Files for Shared Nothing Persistence

File Name	Resource	Resource Type	Description
`bediscovery-service.yaml`	Discovery node service	Service (Internal)	Set up an internal service for connecting non-discovery nodes of the cluster to the discovery node. In case of the shared nothing persistence, the first cache agent node is the discovery node. Set the `selector` value as the label of the cache agent defined in `becacheagent.yaml` . Other nodes in the cluster use this service to connect to the discovery node. Specify `protocol` and `port` that are used by other nodes to connect to this service.
`becacheagent.yaml`	Cache agent node	StatefulSet	Set up the cache agent node of the application Docker container for the cluster. Specify the application Docker image to create the container. Specify `replicas` value (minimum value is `1`) and start as many cache agent as specified in the value. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service. Also, specify the details of the persistent volume claims based on the storage used by the cloud platform.
`beinferenceagent.yaml`	Inference agent node	StatefulSet	Set up an inference agent of the application Docker container for connecting to external APIs. Specify the application Docker image to create the container. Provide a label to the StatefulSet which the inference agent service can use as `selector`. Specify at least one replica of the inference agent node. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service. Also, specify the details of the persistent volume claims based on the storage used by the cloud platform.
`befdservice.yaml`	Inference agent service	Service (LoadBalancer/ External)	Set up an external service to connect to the inference agent. Set the `selector` value as the label of the inference agent defined in `beinferenceagent.yaml`. Specify `protocol` and `port` to connect to this service externally.

Sample Kubernetes YAML Files for Applications with Shared All Persistence

TIBCO BusinessEvents provides sample YAML files at `BE_HOME\cloud\kubernetes\<cloud_name>\cache\shared-all`.

If you are using global variable group in YAML files, instead of the slash '/' delimiter between global variable group name and global variable name, use "_gv_". For example, `port` is a global variable which is part of the `VariableGP` global variable group, then instead of using `VariableGP/port` in YAML files, use `VariableGP_gv_port`. Also, ensure that you do not use the "gv" token in any global variable group name or global variable name.

The following tables list the sample database and common Kubernetes object specification files provided for cloud platforms. For details on the Kubernetes objects used and YAML files, see the Kubernetes documentation.

Sample Database Object Definition Files

Cloud Platform	File Name	Resource Type	Description
OpenShift Container Platform	`mysql.yaml`	StatefulSet	Set up the MySQL node with the MySQL container. Specify the CentOS based MySQL Docker image to create the container. Specify the port that you have forwarded as the container port. Specify the persistent volume claims to be used for the MySQL database.
	`mysql-service.yaml`	Service (Internal)	Set up the service for connecting to the MySQL database.
	`persistent-volume-and-claim.yaml`	PersistentVolume PersistentVolume Claim	Set up the persistent volume and persistent volume claims to be used in MySQL. The MySQL node uses this persistent volume claims for storage. Specify the amount of storage to be allocated to this volume. This sample file uses NFS plugin for the volume. Provide path of the folder that you have created, and server URL for mounting that folder.
Microsoft Azure (MySQL)	`mysql.yaml`	StatefulSet PersistentVolume Claim	Set up the MySQL node with the MySQL container. Specify the MySQL Docker image to create the container. Specify the port that you have forwarded as the container port. Specify the PVC to be used for the MySQL database.
	`mysql-service.yaml`	Service (Internal)	Set up the service for connecting to the MySQL database.

Cloud Platform	File Name	Resource Type	Description
Enterprise PKS	`mysql.yaml`	StatefulSet	Set up the MySQL node with the `mysql` container from the Docker hub. Specify the port that you have forwarded as the container port. Define the `volume` by using the `PersistentVolumeClaim` defined in the `persistent-volume-and-claim.yaml` file for the database storage. Also provide the path of the folder to mount the volume.
	`mysql-service.yaml`	Service (Internal)	Set up the service for connecting to the MySQL database.
	`persistent-volume-claim.yaml`	StorageClass PersistentVolume Claim	Setup the `StorageClass` with `gce-pd` provisioner and its `PersistentVolumeClaim` to be used for the MySQL database. The MySQL node uses this `PersistentVolumeClaim` for storage. Specify the amount of storage to be allocated to this `PersistentVolumeClaim`.

Sample Kubernetes Resource YAML Files for Shared All Persistence

File Name	Resource	Resource Type	Description
`db-configmap.yaml`	ConfigMap	ConfigMap	Set up the environment variables for the database connection. These environment variables are used by deployment instances (`bediscoverynode.yaml`, `becacheagent.yaml`, and `beinferenceagent.yaml`) for connection to the database.
`bediscoverynode.yaml`	Discovery node	Deployment	Set up the discovery node with the application Docker container for the starting the cluster. Specify the application Docker image to create the container. Provide a label which is used by the discovery node service as `selector`. Specify only one replica of the discovery node. Use the ConfigMap environment variables to provide database connection values for the global variables that are used in the application.
`bediscovery-service.yaml`	Discovery node service	Service (Internal)	Set up an internal service for connecting non-discovery nodes of the cluster to the discovery node. Specify the label of the discovery node as `selector`. Specify the `protocol` and `port` that is used by other nodes to connect to this service.

File Name	Resource	Resource Type	Description
`becacheagent.yaml`	Cache agent node	Deployment	Set up the cache agent node with the application Docker container for the cluster. Specify the application Docker image to create the container. Specify `replicas` value based on the number of cache agent you want to start. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service. Provide database connection values for the global variables that are used in the application, using the ConfigMap environment variables.
`beinferenceagent.yaml`	Inference agent node	Deployment	Set up an inference agent with the application Docker container for connecting to external APIs. Specify the application Docker image to create the container. Provide a label to the deployment which the inference agent service can use as `selector`. Specify at least one replica of the inference agent node. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service. Provide database connection values for the global variables, that are used in the application, using the ConfigMap environment variables.
`befdservice.yaml`	Inference agent service	Service (LoadBalancer/ External)	Set up an external service to connect to the inference agent. Specify label of the inference agent as `selector` for the service. Specify the `protocol` and `port` to connect to this service externally.

Sample YAML Files for TIBCO BusinessEvents Enterprise Administrator Agent

TIBCO BusinessEvents provides sample YAML files at *BE_HOME*`\cloud\kubernetes\`*`<cloud_name>`*`\tea` for deploying TIBCO BusinessEvents Enterprise Administrator Agent for monitoring TIBCO BusinessEvents applications.

If you are using global variable group in YAML files, instead of the slash '`/`' delimiter between global variable group name and global variable name, use "_gv_". For example, `port` is a global variable which is part of the `VariableGP` global variable group, then instead of using `VariableGP/port` in YAML files, use `VariableGP_gv_port`. Also, ensure that you do not use the "`gv`" token in any global variable group name or global variable name.

The following tables list the sample database and common Kubernetes object specification files provided for cloud platforms. For details on the Kubernetes objects used and YAML files, see the Kubernetes documentation.

Sample Kubernetes Resource YAML Files for No Backing Store

File Name	Resource	Resource Type	Description
`beteagentdeploymemt.yaml`	TIBCO BusinessEvents Enterprise Administrator Agent node	Deployment	Set up the node with the TIBCO BusinessEvents Enterprise Administrator Agent Docker container. Specify the TIBCO BusinessEvents Enterprise Administrator Agent Docker image to create the container. Provide a label which is used by the `beteagentinternalservice` service as `selector`. Specify only one replica of the node. Specify the TIBCO Enterprise Administrator server URL obtained from the TIBCO Enterprise Administrator server external service and login credentials.
`beteagentinternalservice.yaml`	TIBCO BusinessEvents Enterprise Administrator Agent node service	Service (Internal)	Set up an internal service for connecting TIBCO Enterprise Administrator server to TIBCO BusinessEvents Enterprise Administrator Agent. Specify the label of the TIBCO BusinessEvents Enterprise Administrator Agent node as `selector`. Specify the `protocol` and `port` that is used by other nodes to connect to this service.
`k8s-authorization.yaml`	Authorization	ClusterRoleBinding	Assign roles to the users in TIBCO Enterprise Administrator in the same Kubernetes namespace.

Sample Kubernetes YAML Files for RMS

TIBCO BusinessEvents provides sample YAML files at `*BE_HOME*\cloud\kubernetes\rms`.

If you are using global variable group in YAML files, instead of the slash '`/`' delimiter between global variable group name and global variable name, use "_gv_". For example, `port` is a global variable which is part of the `VariableGP` global variable group, then instead of using `VariableGP/port` in YAML files, use `VariableGP_gv_port`. Also, ensure that you do not use the "gv" token in any global variable group name or global variable name.

The following tables list the sample Kubernetes object specification files provided for cloud platforms. For details on the Kubernetes objects used and YAML files, see the Kubernetes documentation.

Sample Persistent Volume Claim Object Definition Files

Cloud Platform	File Name	Resource Type	Description
OpenShift Container Platform	`persistent-volume.yaml`	PersistentVolume	Set up persistent volumes to provision storage for the following RMS artifacts: • Project hot deployed artifacts (`webstudio-pv`) • Project shared files (`shared-pv`) • Project ACLs (`security-pv`) • Email notifications (`notify-pv`) Specify the amount of storage to be allocated to these volumes. This sample file uses the NFS plugin for the volume. Provide path of the folder that you have created, and server URL for mounting that folder. Use these persistent volumes in `persistent-volume-claims.yaml` to provision persistent volume claims for storage of RMS artifacts.
	`persistent-volume-claims.yaml`	PersistentVolume Claim	Set up the persistent volume claims to store the following RMS artifacts by using the respective `PersistentVolume` defined in `persistent-volume.yaml`: • Project hot deployed artifacts (`webstudio-pvc`) • Project shared files (`shared-pvc`) • Project ACLs (`security-pvc`) • Email notifications (`notify-pvc`) Use these persistent volume claims in the inference agent `beinferenceagent.yaml` to mount volumes for RMS artifacts storage.
Microsoft Azure (MySQL)	`persistent-volume-claims.yaml`	PersistentVolume Claim	Set up the `StorageClass` object with the Azure File (`azure-file`) provisioner. This `StorageClass` is used to provision persistent volume claims for storage of the following RMS artifacts: • Project hot deployed artifacts (`azurefile-webstudio`) • Project shared files (`azurefile-shared`) • Project ACLs (`azurefile-security`) • Email notifications (`azurefile-notify`) Use these persistent volume claims in the inference agent `beinferenceagent.yaml` to mount volumes for RMS artifacts storage.

Cloud Platform	File Name	Resource Type	Description
Enterprise PKS	manifest_gcp.yaml	StorageClass	Set the StorageClass object with the GCP persistent disk (gce-pd) provisioner. This StorageClass is used to provision persistent volume claims for storage of RMS artifacts.
	persistent-volume-claims.yaml	PersistentVolumeClaim	Set up the persistent volume claims to store the following RMS artifacts by using the StorageClass defined in manifest_gcp.yaml: • Project hot deployed artifacts (webstudio-pvc) • Project shared files (shared-pvc) • Project ACLs (security-pvc) • Email notifications (notify-pvc) Use these persistent volume claims in the inference agent beinferenceagent.yaml to mount volumes for RMS artifacts storage.

Sample Kubernetes Resource YAML Files for RMS

File Name	Resource	Resource Type	Description
bediscoverynode.yaml	Discovery node	Deployment	Set up the container with the docker image of the application. Provide a label to the deployment which the discovery node service can use as selector. Specify only one replica of the discovery node. Provide the JMX Kubernetes service name created earlier (bejmx-service.default.svc.cluster.local) as the value of DOCKER_HOST. Specify the volume mounts to use the shared persistent volume claims created earlier.
bediscovery-service.yaml	Discovery node service	Service (Internal)	Set up the service to connect to the discovery node. Specify the label of the discovery node as the value of selector. Other nodes in the cluster use this service to connect to the discovery node. Specify the protocol and port to connect to this service.
becacheagent.yaml	Cache agent node	Deployment	Set up the container with the docker image of the application. Specify replicas value and start as many cache agent as specified in the value. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service. Provide the JMX Kubernetes service name created earlier (bejmx-service.default.svc.cluster.local) as the value of DOCKER_HOST. Specify the volume mounts to use the shared persistent volume claims created earlier.

File Name	Resource	Resource Type	Description
`beinferenceagent.yaml`	Inference agent node	Deployment	Set up the container with the docker image of the application. Provide a label to the deployment which the JMX service can use as `selector`. Specify at least one replica of the inference agent node. Connect to the discovery node service using the discovery protocol and port specified in the discovery node service. Provide the JMX Kubernetes service name created earlier (for example, `bejmx-service.default.svc.cluster.local`) as the value of `DOCKER_HOST`. Specify the volume mounts to use the shared persistent volume claims for storing artifacts.
`befdservice.yaml`	Inference agent service	Service (LoadBalancer/ External)	Set up an external service to connect to the inference agent. Specify label of the inference agent as value of `selector` for the service. Specify the `protocol` and `port` to connect to this service externally.
`berms.yaml`	Discovery node	Deployment	Set up the container with the RMS docker image. Provide a label to the deployment which the RMS node service can use as `selector`. Specify the volume mounts to use the shared persistent volume claims created earlier.
`berms-service.yaml`	Discovery node service	Service (LoadBalancer/ External)	Set up the service to externally connect to the RMS node. Specify the label of the RMS node as the value of `selector`. Specify the `protocol` and `port` to connect to this service.
`bejmx-service.yaml`	JMX service	Service (Internal)	Set up the service for RMS to connect to the JMX port of the inference agent. Set up the label of the inference agent as the value of the `selector` variable for connection. Specify the `protocol` and `port` to connect to this service.