



TIBCO Enterprise Message Service™ Installation on Red Hat OpenShift Container Platform

Software Release 8.5

May 2019

Document Updated: August 2020

Important Information

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

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

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

TIBCO, the TIBCO logo, the TIBCO O logo, Two-Second Advantage, TIBCO Cloud Integration, TIBCO Flogo Apps, TIBCO Flogo, TIB, Information Bus, TIBCO Enterprise Message Service, Rendezvous, and TIBCO Rendezvous are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

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

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

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

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

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

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

Copyright © 1997-2020. TIBCO Software Inc. All Rights Reserved.

Contents

About this Product	4
TIBCO Documentation and Support Services	5
Overview	7
Fault Tolerance and Shared Folder Setup	8
Shared Storage	8
Control Access to NFS Shared Folders	8
Setting Up the Shared Folder	8
EMS Docker Image	9
Creating the Base Docker Image	9
Extending the Base Docker Image	9
Provisioning FTL Client Libraries to Use the Corresponding Transports	9
Provisioning Custom JAAS Authentication or JACI Authorization Modules	10
Hosting the Image	10
OpenShift Setup	11
Creating a Project	11
Provisioning the NFS Shared Folder	11
Enabling Images to Run with the USER Specified in the Dockerfile	12
Modifying the Services NodePort range (Optional)	12
EMS Server Template	13
Parameters Objects	13
Service Objects and EMS Client URLs	14
Deployment Object	14
Health Checks: Liveness and Readiness Probes	15
Creating a Deployment and Service	16
Stopping or Deleting an EMS Server	16
EMS Server Configuration	17
Central Administration Server Template	17
Creating a Deployment and Service through the Web Console	17
TLS Configuration	20
Creating a Secret	20
Modifying the Template	20
Modifying the tibemscreeimage EMS Docker Image Build Script	21
Applying the Modifications	21

About this Product

TIBCO is proud to announce the latest release of TIBCO Enterprise Message Service™ software.

This release is the latest in a long history of TIBCO products that leverage the power of the Information Bus® technology to enable truly event-driven IT environments. To find out more about how TIBCO Enterprise Message Service software and other TIBCO products are powered by TIB® technology, please visit us at www.tibco.com.

TIBCO Enterprise Message Service software lets application programs send and receive messages according to the Java Message Service (JMS) protocol. It also integrates with TIBCO FTL, TIBCO Rendezvous, and TIBCO SmartSockets messaging products.

TIBCO EMS software is part of TIBCO® Messaging.

Product Editions

TIBCO Messaging is available in a community edition and an enterprise edition.

TIBCO Messaging - Community Edition is ideal for getting started with TIBCO Messaging, for implementing application projects (including proof of concept efforts), for testing, and for deploying applications in a production environment. Although the community license limits the number of production clients, you can easily upgrade to the enterprise edition as your use of TIBCO Messaging expands.

The community edition is available free of charge. It is a full installation of the TIBCO Messaging software, with the following limitations and exclusions:

- Users may run up to 100 application instances or 1000 web/mobile instances in a production environment.
- Users do not have access to TIBCO Support, but you can use TIBCO Community as a resource (<https://community.tibco.com>).
- Available on Red Hat Enterprise Linux Server, Microsoft Windows & Windows Server and Apple macOS.

TIBCO Messaging - Community Edition has the following additional limitations and exclusions:

- Excludes Fault Tolerance of the server.
- Excludes Unshared State Failover.
- Excludes Routing of messages between servers.
- Excludes Central Administration.
- Excludes JSON configuration files.

TIBCO Messaging - Enterprise Edition is ideal for all application development projects, and for deploying and managing applications in an enterprise production environment. It includes all features presented in this documentation set, as well as access to TIBCO Support.

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>.

TIBCO Enterprise Message Service Documentation

The following documents for this product can be found on the [TIBCO Enterprise Message Service™](#) product documentation page:

- *TIBCO Enterprise Message Service User's Guide* Read this manual to gain an overall understanding of the product, its features, and configuration.
- *TIBCO Enterprise Message Service Central Administration* Read this manual for information on the central administration interface.
- *TIBCO Enterprise Message Service Installation* Read the relevant sections of this manual before installing this product.
- *TIBCO Enterprise Message Service C & COBOL Reference* The C API reference is available in HTML and PDF formats.
- *TIBCO Enterprise Message Service Java API Reference* The Java API reference can be accessed only through the HTML documentation interface.
- *TIBCO Enterprise Message Service .NET API Reference* The .NET API reference can be accessed only through the HTML documentation interface.
- *TIBCO Enterprise Message Service Installation on Red Hat OpenShift Container Platform* This manual describes how to run TIBCO Enterprise Message Service servers on the Red Hat® OpenShift Container Platform.
- *TIBCO Enterprise Message Service Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release. This document is available only in PDF format.

Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO FTL®
- TIBCO Rendezvous®
- TIBCO SmartSockets®
- TIBCO EMS® Client for z/OS (CICS)
- TIBCO EMS® Client for z/OS (MVS)
- TIBCO EMS® Client for IBM i

Third-Party Documentation

- Java™ Message Service specification, available through <http://www.oracle.com/technetwork/java/jms/index.html>.

- *Java™ Message Service* by Richard Monson-Haefel and David A. Chappell, O'Reilly and Associates, Sebastopol, California, 2001.
- Java™ Authentication and Authorization Service (JAAS) LoginModule Developer's Guide and Reference Guide, available through <http://www.oracle.com/technetwork/java/javase/jaas/index.html>.

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>.

Overview

Running TIBCO Enterprise Message Service on Red Hat OpenShift Container Platform involves:

- Preparing a shared folder on NFS.
- Creating a Docker® image embedding EMS and hosting it in a Docker registry.
- Provisioning the NFS shared folder in OpenShift.
- Configuring and creating EMS containers based on the EMS Docker image.

Supported Versions

Refer to the Release Notes for:

- The supported versions of the software involved.
- The EMS features supported on the OpenShift Container Platform.

Prerequisites

Before you continue, familiarize yourself with the following:

- Docker concepts.
- Red Hat OpenShift Container Platform administration.
- TIBCO EMS configuration.
- NFSv4.

Before you continue, make sure you have the following infrastructure in place:

- A machine equipped for building Docker images.
- A Docker registry.
- An OpenShift Container Platform cluster.
- A shared folder on an NFSv4 server.

Fault Tolerance and Shared Folder Setup

Shared Storage

A traditional EMS server configured for fault tolerance relies on its state being shared by a primary and a secondary instance, one being in the active state while the other is in standby, ready to take over. The shared state relies on the server store and configuration files to be located on a shared storage such as a SAN or a NAS using NFS.

By contrast, the fault tolerance model used by EMS on OpenShift relies on OpenShift restart mechanisms. Only one EMS server instance is running and, in case of a server failure, will be restarted inside its container. In case of a failure of the container or of the corresponding cluster node, the cluster will recreate the container, possibly on a different node, and restart the EMS server there.

Within the container, the health of the EMS server is monitored by two health check probes:

- liveness
- readiness

For more information on the probes, see [Health Checks: Liveness and Readiness Probes](#).

The server requires its state to be shared. The shared storage required by EMS on OpenShift is NFSv4.

Control Access to NFS Shared Folders

You can control access to the NFS shared folders using User and Group IDs.

Depending on how your NFS server is configured, programs accessing shared folders may have to run with a specific user ID (`uid`) and group ID (`gid`).

While you can control the `uid` of a container through a field called `runAsUser`, controlling its `gid` is not possible in version 3.11 of OpenShift. If your NFS setup requires controlling the `gid` used by the EMS server, a workaround consists of creating a specific user and group in the EMS Docker image (see [EMS Docker Image](#)) and setting its `uid` and `gid` to the desired values.

As a result, an EMS server running in a container started from that image will access its store, log, configuration, and other files using the `uid` and `gid` specified by the NFS server.

Setting Up the Shared Folder

Procedure

1. Log on to a machine that can access the NFS shared folder with the user account meant to be used by the EMS server.
2. Create the shared folder.
For example, `~/OpenShift/shared`.
3. Modify the permissions to your requirements.
For example, `750 (rwxr-x---`)

Example

```
> mkdir -p ~/OpenShift/shared
> chmod -R 750 ~/OpenShift/shared
```

EMS Docker Image

Creating the Base Docker Image

The content of the container that will run on OpenShift derives from a Docker image that is first created and then hosted in a Docker registry.

To create an EMS Docker image, use the `samples/docker/tibemscreeimage` script on a machine equipped for building Docker images.

This script needs to be pointed to the software packages to be installed:

- EMS installation package
- Optional EMS hotfixes
- The Java package (optional)

The script also lets you choose whether to save the image as an archive and creates a user and group set to the required `uid` and `gid` values to bypass the lack of a `runAsGroup` OpenShift feature mentioned in section [Control Access to NFS Shared Folders](#).

The following command creates a Docker image based on the EMS 8.5 Linux installation package, adding a JVM, the 12500 `uid` and the 9500 `gid`.

```
> tibemscreeimage TIB_ems_8.5.0_linux_x86_64.zip \
    -j <JRE installation package>.tar.gz \
    -u 12500 \
    -g 9500
```

The following examples illustrate how you can experiment with that Docker image:

- This following command creates a sample EMS server folder hierarchy and configuration in the current directory and starts the corresponding server:


```
> docker run -p 7222:7222 -v `pwd`::/shared ems:8.5.0 tibemsd
```
- To create a sample Central Administration server folder hierarchy and configuration in the current directory and starts the corresponding server:


```
> docker run -p 8080:8080 -v `pwd`::/shared ems:8.5.0 tibemsc
```
- You can override the creation and use of the sample configuration with your own setup. The following example starts an EMS server using the *<path to shared location>/<your server config file>* configuration.


```
> docker run -p 7222:7222 -v <path to shared location>:/shared \
    ems:8.5.0 tibemsd -config /shared/<your server config file>
```

You can modify the `tibemscreeimage` script to suit your environment.

Extending the Base Docker Image

The base Docker image can be extended to include FTL client libraries, custom JAAS authentication, and JACI authorization modules.

Provisioning FTL Client Libraries to Use the Corresponding Transports

Procedure

1. Copy the FTL client library files to a temporary folder.

- From the temporary folder, use a Dockerfile based on the example given below to copy these files into the base Docker image:

```
FROM ems:8.5.0
COPY --chown=tibuser:tibgroup . /opt/tibco/ems/docker/ftl
```

Execute the command

```
> docker build -t ems:8.5.0_ftl .
```

- After customizing your EMS configuration, include `/opt/tibco/ems/docker/ftl` in the Module Path property.

Provisioning Custom JAAS Authentication or JACI Authorization Modules

Procedure

- Copy your custom JAAS or JACI plugin files, including the static configuration files they may rely on, to a temporary folder.
- From the temporary folder, use a Dockerfile based on the example given below to copy these files into the base Docker image:

```
FROM ems:8.5.0
COPY --chown=tibuser:tibgroup . /opt/tibco/ems/docker/security
```

Execute the command

```
> docker build -t ems:8.5.0_security .
```

- After customizing your EMS configuration, make sure you include the relevant paths to those files in the Security Classpath property.



The other required files are in their usual location `/opt/tibco/ems/version/bin` and `/opt/tibco/ems/version/lib`.

For example, `/opt/tibco/ems/docker/security/user_jaas_plugin.jar:/opt/tibco/ems/8.5/bin/tibemsd_jaas.jar:/opt/tibco/ems/8.5/lib/tibjmsadmin.jar`, and so on.

Hosting the Image

Tag the image to match your Docker registry location and push it there.

For example:

```
> docker tag ems:8.5.0 docker.company.com/path/ems
> docker push docker.company.com/path/ems
```

OpenShift Setup

Creating a Project

Create a project with an OpenShift user that has the required permission.

For example:

```
> oc login -u <user>
> oc new-project ems-project
```

Provisioning the NFS Shared Folder

A storage resource is provisioned in OpenShift by the cluster administrator through a Persistent Volume (PV), which has to be of type NFS. A project can then claim that resource through a Persistent Volume Claim (PVC). That claim will eventually be mounted as a volume inside containers.

In the case of an EMS project, we create one PV and one PVC at the same time since these are meant to be bound together.

Procedure

1. Modify the `samples/openshift/nfs-pv-pvc.yaml` file for your setup.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs-ems-project
  annotations:
    # Should be replaced by spec.mountOptions in the future
    volume.beta.kubernetes.io/mount-options: soft (1)
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /vol/home/user/OpenShift/shared (2)
    server: 10.98.128.50 (3)
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim-nfs-ems-project
    namespace: ems-project (4)
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claim-nfs-ems-project
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  volumeName: pv-nfs-ems-project
```

(1): Optional comma-separated list of NFS mount options used when the PV is mounted on a cluster node.

(2): The path that is exported by the NFS server. In this example, we want the patch to match the `~/OpenShift/shared` folder created in [Setting Up the Shared Folder](#).

(3): The host name or IP address of the NFS server.

(4): This needs to match the name of the project created previously.

2. Switch to the `system:admin` cluster administrator and create the PV and PVC:

```
> oc login -u system:admin
> oc create -n ems-project -f nfs-pv-pvc.yaml
```

3. Check the result using the following command:

```
> oc get pv,pvc
```



The same PV/PVC can be used by multiple pods within the same project.
Creating the PV/PVC is done once for the lifetime of the project.

Enabling Images to Run with the USER Specified in the Dockerfile

By default, programs in OpenShift containers run with a user the `uid` of which is automatically pre-allocated by the cluster. As we need the EMS server to run with the specific `uid` required by NFS, we must create a new Security Context Constraint (SCC) to address this. We will export the existing restricted SCC and use it as a basis for our new SCC.

Procedure

1. As `system:admin`, export the restricted SCC into a file:


```
> oc get -o yaml scc/restricted > nfs-scc.yaml
```
2. Edit this file and give this SCC a new name by changing `restricted` into `nfs-scc` and give it a higher priority.
3. Add a `uidRangeMax` and a `uidRangeMin` field to the `runAsUser` entry. These define a range of allowed `uid` values and should match the values you expect your pods to use for accessing NFS. For example:

```
...
metadata:
...
  name: nfs-scc (1)
priority: 9 (2)
...
runAsUser:
  type: MustRunAsRange
  uidRangeMax: 13000 (3)
  uidRangeMin: 12000 (3)
...
```

(1): Name of this new SCC.

(2): Larger values mean greater priority.

(3): The range of allowed `uid` values will be 12000-13000. This works with the 12500 `uid` set in [Creating the Base Docker Image](#).

4. Create the new SCC using the following command

```
> oc create -f nfs-scc.yaml
```

5. Check the result

```
> oc get -o yaml scc/nfs-scc
```



Edit the new SCC using the command:

```
> oc edit scc/nfs-scc
```



You create the SSC once for the lifetime of the cluster.

Modifying the Services NodePort range (Optional)

Services of type `NodePort` are used to expose EMS server listen ports outside the cluster. See [Service Object and EMS Client URLs](#) for more information.

The range of allowed values defaults to 30000–32767.

To use port numbers outside this range for the EMS server or Central Administration server, you can alter the range in the *Kubernetes Master Configuration*:

Procedure

1. Locate the file `master/master-config.yaml` on each OpenShift Master and edit it to set the value of `servicesNodePortRange` to the range of your choice.

For example:

```
serviceNodePortRange: 7000-8000
```

2. Restart each OpenShift Master service.

EMS Server Template

EMS server containers are created in an OpenShift cluster through the `samples/openshift/tibemsd-template.yaml` sample template. This template includes sections that define a set of parameters, a deployment, and a service.

Parameters Objects

The parameters let you configure the aspects of the container and service that can be adjusted at creation time. These include:

Option	Description
EMS_SERVICE_NAME	The name of the service through which the EMS server is accessible inside the cluster.
EMS_PUBLIC_PORT	The port number through which the EMS server is accessible (both inside and outside the cluster).
EMS_INTERNAL_PORT	The port number used by the EMS server inside its container.
EMS_PROBE_PORT	The internal port number on which the EMS server responds to health check requests.
EMS_UID	The uid the EMS server container must run as with respect to accessing NFS.
EMS_IMAGE_LOCATION	The location of the EMS Docker image in your Docker registry.
EMS_PVC	The name of the PVC previously configured to access the NFS shared folder.

All parameters have a default value that can be overridden upon creation.



The uid provided here must match the one used when creating the EMS Docker image.

Service Objects and EMS Client URLs

The service object exposes the EMS server listen port (both inside and outside the cluster).

The service defined in `tibemsd-template.yaml` is of type `NodePort`, which means that the corresponding port number will be accessible through all nodes of the cluster.

For example, if your cluster runs on three nodes called `node1`, `node2` and `node3` that can be addressed by those host names. If you have exposed your EMS server through a service using port number 8222, EMS clients running outside the cluster will be able to access it either through the `tcp://node1:8222`, `tcp://node2:8222` or `tcp://node3:8222` URL, regardless of the node where the container is actually running. This works by virtue of each node proxying port 8222 into the service.

EMS clients running inside the cluster will be able to access the EMS server either in the fashion described above or through its service name. Assuming the service name is `emsdev01` and the port still is 8222, that amounts to using the `tcp://emsdev01:8222` URL.

To ensure EMS client automated fault-tolerance failover, these must connect with FT double URLs. Using the example above: `tcp://node1:8222, tcp://node1:8222` from outside the cluster or `tcp://emsdev01:8222, tcp://emsdev01:8222` from inside the cluster. For the first form, since all nodes will proxy port 8222 into the service, repeating the same node name twice fits our purpose. The connection factories in the sample EMS server configuration generated by default upon creating a container illustrate that pattern. Should the EMS server or its container fail, clients will automatically reconnect to the same URL once the server has been restarted.

You can use types of service other than `NodePort` if they fit your requirements.

Deployment Object

A deployment includes the definition of a set of containers and the desired behavior in terms of number of replicas (underlying `ReplicaSet`) and deployment strategy.

```
kind: Deployment
...
spec:
  replicas: 1 (1)
  ...
  strategy:
    type: Recreate (2)
  ...
  template:
    ...
    spec:
      containers:
      - name: tibemsd-container
        image: ${EMS_IMAGE_LOCATION}
        imagePullPolicy: Always (3)
        env: (4)
        - name: EMS_NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: EMS_PUBLIC_PORT
          value: ${EMS_PUBLIC_PORT}
      ...
      args: (5)
      - tibemsd
      livenessProbe: (6)
      ...
      readinessProbe: (6)
      ...
      ports:
      - containerPort: ${EMS_INTERNAL_PORT}
        name: tibemsd-tcp
        protocol: TCP
      ...
      securityContext:
```

```

    runAsUser: ${EMS_UID} (7)
    ...
    volumeMounts:
    - mountPath: /shared (8)
      name: tibemsd-volume (9)
    ...
    restartPolicy: Always (10)
    ...
    volumes:
    - name: tibemsd-volume (9)
      persistentVolumeClaim:
        claimName: ${EMS_PVC} (11)

```

- (1): The number of replicated pods: 1, since we want a single instance of the EMS server.
- (2): The deployment strategy: `Recreate` means that an existing pod must be killed before a new one is created.
- (3): Determines if the EMS Docker image should be pulled from the Docker registry prior to starting the container.
- (4): Environment variables that will be passed to the container.
- (5): Arguments to be passed to the Docker `ENTRYPOINT`. For more information, see [EMS Server Configuration](#).
- (6): For details on the liveness and readiness probes, see [Liveness and Readiness Probes](#).
- (7): The `uid` the container will run as.
- (8): The path where our NFS shared folder will be mounted inside of the container.
- (9): The internal reference to the volume defined here.
- (10): The pod restart policy: Set such that the kubelet will always try to restart container. If the EMS server stops or fails, its container will exit and be restarted.
- (11): The name of the PVC created by the cluster administrator.

Health Checks: Liveness and Readiness Probes

Refer to the OpenShift and Kubernetes documentation for a description of health checks.

For an EMS server container, a liveness health check helps detect when an EMS server is not running. When this health check fails a number of times in a row, the EMS server container is restarted.

A readiness health check helps detect when an EMS server that is up and running is not in the active state. When this health check fails a number of times in a row, the EMS server endpoints are removed from its container, such that the server is made unreachable. As it may or may not fit your operations, it is up to you to decide whether you need the readiness health check. If not relevant, you may remove the health check from the template.

The sample probes are configured in the [deployment object](#):

```

...
livenessProbe:
  httpGet:
    path: /isLive
    port: probe-tcp
  initialDelaySeconds: 1 (1)
  timeoutSeconds: 5 (2)
  periodSeconds: 6 (3)
readinessProbe:
  httpGet:
    path: /isReady
    port: probe-tcp
  initialDelaySeconds: 1 (1)
  timeoutSeconds: 5 (2)

```

```
    periodSeconds: 6 (3)
    ...
```

(1): Number of seconds after the container has started before the probe is initiated.

(2): Number of seconds after which the probe times out. Defaults to 1 second. Minimum value is 1.

(3): How often (in seconds) to perform the probe. Defaults to 10 seconds. Minimum value is 1.

Creating a Deployment and Service

Prerequisites

If you are logged in as a cluster administrator, log out and log back in as a regular user.

Procedure

1. Edit the `tibemsd-template.yaml` template and override the default parameters as needed.
2. Create a deployment and service with an EMS server using the modified template.

For example,

```
> oc login -u <user>
> oc process -f tibemsd-template.yaml -p EMS_SERVICE_NAME=emsdev01 \
  -p EMS_PUBLIC_PORT="7779" -p EMS_PVC="claim-nfs-ems-project" \
  | oc create -f -
```

The `oc process` operation transforms the `tibemsd-template.yaml` template into a list of resources using the default and overridden parameter values. That list is then passed on to `oc create` for creation. In the above example, a deployment, a ReplicaSet, a pod, and a service are created. All four objects can be selected using the `emsdev01` label.

The service exposes itself as `emsdev01` inside the cluster and maps the internal port 7222 to port 7779 inside and outside the cluster. The PVC `claim-nfs-ems-project` is used to mount the NFS shared folders. Default values of other parameters are used to pull the EMS Docker image from the Docker registry and to set the `uid` of the container.

You can verify the results using the following commands:

```
> oc get --selector name=emsdev01 all
> oc describe deploy/emsdev01
> oc describe svc/emsdev01
```

or in the OpenShift Web Console.

To deploy additional EMS servers

```
> oc process -f tibemsd-template.yaml -p EMS_SERVICE_NAME=emsdev02 \
  -p EMS_PUBLIC_PORT="7780" -p EMS_PVC="claim-nfs-ems-project" \
  | oc create -f -
> oc process -f tibemsd-template.yaml -p EMS_SERVICE_NAME=emsdev03 \
  -p EMS_PUBLIC_PORT="7781" -p EMS_PVC="claim-nfs-ems-project" \
  | oc create -f -
```

Stopping or Deleting an EMS Server

To stop an EMS server without deleting it, use the `oc scale` operation to set its number of replicas to 0.

For example,

```
> oc scale --replicas=0 deploy emsdev01
```

To restart this EMS server, set the number of replicas to 1.

```
> oc scale --replicas=1 deploy emsdev01
```

To delete an EMS server deployment and service, use the `oc delete` operation.

For example,

```
> oc delete --selector name=emsdev01 deploy,svc
```

The corresponding pod and ReplicaSet will also be deleted.

EMS Server Configuration

As mentioned in [Creating the Base Docker Image](#), running a container off of the EMS Docker image creates a default EMS server folder hierarchy and configuration. In an OpenShift cluster, the configuration will be created under `ems/config/${EMS_SERVICE_NAME}.json` in the NFS shared folder if absent. The Central Administration server works in a similar way.

This is handled by the `tibems.sh` script embedded in `tibemscreeimage` and is invoked through the Docker image `ENTRYPOINT`. It can be overridden by altering the `args` entry in the template and is provided only for illustration purposes. You can either alter `tibems.sh` or directly provision your own configuration files to suit your needs.

Central Administration Server Template

A Central Administration server container is created in an OpenShift cluster through the `samples/openshift/tibemscsca-template.yaml` sample template provided.

The structure of this template is almost identical to that of the EMS server template. Most of the concepts described in previous sections also apply to the Central Administration server.

Example of deployment and service creation with a Central Administration server:

```
> oc process -f tibemscsca-template.yaml \
  -p EMSCA_SERVICE_NAME=emscadev01 -p EMSCA_PUBLIC_PORT="7080"
  -p EMSCA_PVC="claim-nfs-ems-project" | oc create -f -
```

You can then use a Web browser to connect to `http://node1:7080` and add EMS servers to Central Administration.

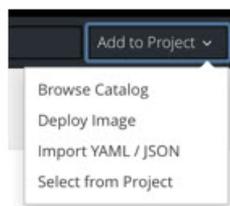
Creating a Deployment and Service through the Web Console

You can create deployments and services using the OpenShift Web Console.

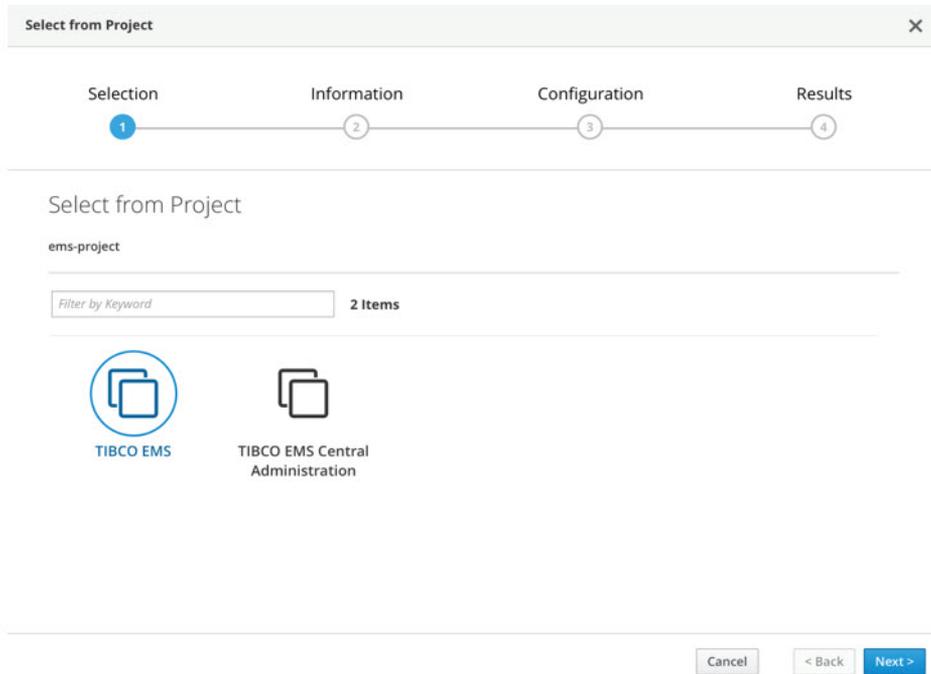
Procedure

1. Upload the templates to your project using the OpenShift Command Line Interface (CLI)


```
> oc create -f tibemsd-template.yaml
> oc create -f tibemscsca-template.yaml
```
2. Log in to the Web Console, select project `ems-project`, and then Add to Project > Select from Project:



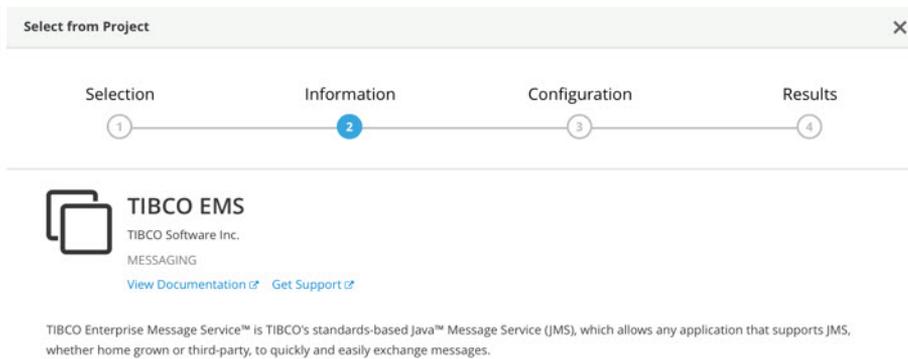
3. The two templates that you just uploaded are available for you to choose from.



available

4. Select **TIBCO EMS** and click **Next**.

The **Information** screen displays links to reach the TIBCO EMS documentation and TIBCO Support.



5. Click **Next**.

The Configuration screen is displayed. This is where you can modify the template parameters with values for this particular instance.

Select from Project
✕

Selection Information 3 Results

*** EMS Service Name**

The name of the service through which the EMS server is accessible inside the cluster.

*** Public Listen Port**

The port number through which the EMS server is accessible, both inside and outside the cluster.

*** Internal Listen Port**

The port number used by the EMS server inside its container.

*** Health Check Probe Port**

The internal port number on which the EMS server responds to health check requests.

*** User ID**

The user ID the EMS server container must run as with respect to accessing NFS.

Cancel < Back Create

6. Modify the values and click **Create**.

Select from Project
✕

Selection Information Configuration 4

✔

TIBCO EMS has been created.

[Continue to the project overview.](#)

Applied Parameter Values

These parameters often include things like passwords. If you will need to reference these values later, copy them to a safe location.

[Show parameter values](#)

Cancel < Back Close

TLS Configuration

The following topics describe how to modify the EMS server template and the Docker image build script so that EMS clients can connect to the server through TLS (formerly SSL).

Whether an EMS listen port is configured for TCP or TLS makes no difference in terms of exposing it through a service. However, you need to decide how to provision the corresponding certificate files.

While these could be placed in the NFS shared folder or embedded in the EMS Docker image, the standard practice in the OpenShift world consists of using secret objects. These are meant to decouple sensitive information from the pods and can be mounted into containers as volumes populated with files to be accessed by programs.

In this example, the EMS server will be authenticated by EMS clients. This involves providing the server with its certificate, private key and the corresponding password, which we will store inside a secret. We will mount that secret into the container, point the EMS server configuration to the certificate and private key files and pass the corresponding password to the server through its `-ssl_password` command-line option.

Based on the sample certificates that ship with EMS, the files will eventually be made available inside the container as follows:

```
/etc/secret/server.cert.pem
/etc/secret/server.key.pem
/etc/secret/ssl_password
```

Creating a Secret

To store the server certificate, private key, and the corresponding password in a secret, based on the sample certificates available in the EMS package under `ems/version/samples/certs`:

```
> cd ../ems/<version>/samples
> oc create secret generic tibemsd-secret \
  --from-file=server.cert.pem=certs/server.cert.pem \
  --from-file=server.key.pem=certs/server.key.pem \
  --from-literal=ssl_password=password
```

Check the result using these commands:

```
> oc describe secret tibemsd-secret
> oc get -o yaml secret/tibemsd-secret
```

Modifying the Template

The `tibemsd-template.yaml` template has to be modified to mount the secret as a volume. This involves adding one entry to the `volumes` section and another to the `volumeMounts` section.

```
kind: Deployment
...
spec:
...
  template:
    ...
    spec:
      containers:
        - name: tibemsd-container
          ...
          volumeMounts:
            - mountPath: /shared
              name: tibemsd-volume
            - mountPath: /etc/secret
              name: tibemsd-secret-volume
              readOnly: true
          ...
      volumes:
```

```
- name: tibemsd-volume
  persistentVolumeClaim:
    claimName: ${EMS_PVC}
- name: tibemsd-secret-volume
  secret:
    secretName: tibemsd-secret
```



You should eventually convert the `secretName` entry into a parameter.

Modifying the tibemscreeimage EMS Docker Image Build Script

Procedure

1. In the `tibemsd-configbase.json` section

a) Modify the `primary_listen` to use `ssl`:

```
"primary_listens": [
  {
    "url": "ssl://7222"
  }
],
```

b) Add an `ssl` section pointing to the certificate files:

```
"tibemsd": {
  ...
  "ssl": {
    "ssl_server_identity": "/etc/secret/server.cert.pem",
    "ssl_server_key": "/etc/secret/server.key.pem"
  },
}
```

2. In the `tibems.sh` section

The `tibemsd_run()` function needs to be modified to launch the EMS server with the proper value for its `-ssl_password` command-line option:

```
...
if [[ \ $# -ge 1 ]]; then
  PARAMS=\ $*
else
  tibemsd_seed
  PARAMS="-config /shared/ems/config/\ $EMS_SERVICE_NAME.json -ssl_pass
word \ `cat /etc/secret/ssl_password\ `\"
fi
...
```

Applying the Modifications

Procedure

1. Regenerate the EMS Docker image, tag it, and push it to the Registry.
See [Creating the Base Docker Image](#)
2. Create a new deployment and service. See [Creating a Deployment and Service](#).

You can check the result by connecting to the server with one of the EMS TLS sample clients:

```
> java tibjmsSSL -server ssl://node1:7779 \
  -ssl_trusted ../certs/server_root.cert.pem \
  -ssl_hostname server
```