TIBCO Software Inc.
Global Headquarters
3307 Hillview Avenue
Palo Alto, CA 94304
Tel: +1 650-846-1000
Toll Free: 1 800-420-8450
Fax: +1 650-846-1005
www.tibco.com

TIBCO Enterprise Message Service™ on Kubernetes

This document describes how to run TIBCO Enterprise Message Service servers on a generic Kubernetes Container Platform.

Version 1.0 October 2018 Initial Document

Trademarks
TIBCO, the TIBCO logo, TIBCO Enterprise Message Service, TIBCO FTL, Rendezvous, and SmartSockets are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or 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.

Content Warranty
The information in this document is subject to change without notice. **THIS DOCUMENT IS PROVIDED "AS IS" AND TIBCO MAKES NO WARRANTY, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO ALL WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.** TIBCO Software Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

For more information, please contact:

TIBCO Software Inc.
3303 Hillview Avenue
Palo Alto, CA 94304
USA

TIBCO®

Table of Contents

1 Overview

Running TIBCO Enterprise Message Service on the Kubernetes Platform involves:

- Installing and configuring the Kubernetes container platform (Not covered as part of this document)
- 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 Kubernetes
- Configuring and creating EMS containers based on the EMS Docker image

1.1 Supported Versions

The steps described in this document are supported for the following versions of the products and components involved:

- TIBCO EMS 8.4.0 with hotfixes 5 and 6
- TIBCO FTL 5.2.1 and later (static TCP transports only)
- Docker 1.13.1
- Red Hat Enterprise/CentOS Linux 7.4 or newer
- Kubernetes 1.11.3
- NFSv4

1.2 Excluded TIBCO EMS Features

As of May 2018, TIBCO EMS on Kubernetes supports all EMS features, with the following exceptions:

- Excludes transports for TIBCO Rendezvous®
- Excludes transports for TIBCO SmartSockets®
- Excludes stores of type dbstore
- Excludes stores of type mstores

TIBCO®

1.3 Prerequisites

The reader of this document should be familiar with:

- Docker concepts
- Kubernetes installation and administration
- TIBCO EMS configuration
- NFSv4

The following infrastructure should already be in place:

- A machine equipped for building Docker images
- A Docker registry
- A Kubernetes cluster
- A shared folder on an NFSv4 server
- JRE installation package (.tar.gz)

1.4 Kubernetes Installation Considerations

The following should be considered when installing and configuring the Kubernetes container platform on a Linux platform:

- Ensure all nodes of the Kubernetes cluster have disabled swapping. If not, *kubelet* agent will not configure or run correctly without modifications (not covered in this document)
- The Docker version matters. Non-supported versions of Docker, will prevent the *kubelet* agent from configuring or running correctly without modifications (not covered in this document)
- If nodes are virtual, ensure their is sufficient RAM available - minimum of 2GB is required, but 8GB is recommended

2 Fault Tolerance and Shared Folder Setup

2.1 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 device such as a SAN or a NAS using NFS.

The fault tolerance model used by EMS on Kubernetes is different in that it relies on Kubernetes 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: the liveness and readiness probes. At this point, the implementation of these is provided as a separate package to be included in the EMS Docker image. Health check probes are detailed in Appendix A.

In any case, the server still needs its state to be shared. The shared storage required by EMS on Kubernetes is NFSv4.

2.2 Controlling User ID and Group ID Used for Accessing NFS Shared Folders

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

While version 1.11 of Kubernetes allows for controlling the **`uid`** of a container through a field called **`runAsUser`**, controlling its **`gid`** isn't possible yet. A **`runAsGroup`** field should be provided in a future release of Kubernetes but, in the meantime, we need to use a workaround to enforce this, if need be.

That workaround consists of creating a specific user and group in the EMS Docker image (see section 3.1 below) and setting its **`uid`** and **`gid`** to the desired values upfront. 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`** dictated by the NFS server.

2.3 Shared Folder Setup

- Log on to a machine that can access the NFS shared folder with the user account meant to be used by the EMS server.
- Create the shared folder, for example **`/shared/kubernetes`**.
- Adjust its permissions to your requirements, for example **`750`** (**`rwxr-x---`**).
- Ensure the there is a corresponding folder on the NFS server, and that the once the share is mounted, the shared folder on the NFS server is accessible. In the example below, the machine that can access the NFS server is mounting the shared folder "/shared/kubernetes" on "/" of the NFS server.

For example:

```
> sudo mkdir -p /shared/kubernetes
> sudo chmod -R 750 /shared/kubernetes
> mount -t nfs -o nfsvers=4,soft,proto=tcp <nfs server>:/
/shared/kubernetes
```

3 The EMS Docker image

3.1 Creating the Base Docker Image

The content of the container that will run on Kubernetes derives from a Docker image that first needs to be created and then hosted in a Docker registry.

To create an EMS Docker image, use the **`tibemscreateimage`** script on a machine equipped for building Docker images.

The script needs to be pointed to the software packages to be installed: the EMS installation package, optional EMS hotfixes, the health check probes package and an optional Java package. It lets you choose which EMS installation features to embed (**`server`**, **`client`**, **`emsca`**, **`dev`**, **`source`**, **`hibernate`**) and whether to save the image as an archive. It also creates a user and group set to the required **`uid`** and **`gid`**.

For example:

```
> tibemscreateimage TIB_ems_8.4.0_linux_x86_64.zip \
                    -h TIB_ems_8.4.0_HF-005.zip \
                    -h TIB_ems_8.4.0_HF-006_linux_x86_64.zip \
                    -p TIB_ems_8.4.0_probes.zip \
                    -j <JRE installation package>.tar.gz \
                    -f server \
                    -f emsca \
                    -u 1000 \
                    -g 1000
```

creates a Docker image with the **`server`** and **`emsca`** EMS installation features based on the EMS 8.4.0 Linux installation package, adding the EMS 8.4.0_HF-005 Java client hotfix, the EMS 8.4.0_HF-006 Linux server hotfix, the probe package, a JVM, the **`1000 uid`** and the **`1000 gid`**. It is up to you to first download the **`TIB_ems_8.4.0_linux_x86_64.zip`**, **`TIB_ems_8.4.0_HF-005.zip`**, **`TIB_ems_8.4.0_HF-006_linux_x86_64.zip`**, **`TIB_ems_8.4.0_probes.zip`** and Java packages.

Note: Depending on what version of Docker is used, the **`tibemscreateimage`** script may fail with the following error:

```
Step 19/26 : COPY --chown=tibuser:tibgroup . /install
Unknown flag: chown
docker build failed.
```

If this occurs, modify the **`tibemscreateimage`** script, find and replace:

```
COPY --chown=tibuser:tibgroup . /install with:

COPY . /install
CMD chown=tibuser:tibgroup /install
```

If you are curious to run this image stand-alone:

```
> docker run -p 7222:7222 -v `pwd`:/shared ems:8.4.0_HF-006 tibemsd
```

creates a sample EMS server folder hierarchy and configuration in the current directory and starts the corresponding server.

```
> docker run -p 8080:8080 -v `pwd`:/shared ems:8.4.0_HF-006 tibemsca
```

creates a sample Central Administration server folder hierarchy and configuration in the current directory and starts the corresponding server.

You can override the creation and use of the sample configuration with your own setup:

```
> docker run -p 7222:7222 -v <path to shared location>:/shared \
  ems:8.4.0_HF-006 tibemsd -config /shared/<your server config file>
```

starts an EMS server using the <path to shared location>/<your server config file> configuration.

Feel free to modify the **`tibemscreateimage`** script to suit your needs.

3.2 Extending the Base Docker Image

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

3.2.1 Provisioning FTL Client Libraries to Use the Corresponding Transports

1. Copy the FTL client library files to a temporary folder.
2. From the temporary folder, use a **`Dockerfile`** based on the example given below to copy these files into the base Docker image:

```
FROM ems:8.4.0_HF-006
COPY --chown=tibuser:tibgroup . /opt/tibco/ems/docker/ftl
```

Note: If build should fail, refer to the ***Note*** in Section 3.1.

```
> docker build -t ems:8.4.0_HF-006_ftl .
```

3. Upon customizing your EMS configuration, make sure to include **`/opt/tibco/ems/docker/ftl`** in the *Module Path* property.

3.2.2 Provisioning Custom JAAS Authentication or JACI authorization Modules

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

```
FROM ems:8.4.0_HF-006
COPY --chown=tibuser:tibgroup . /opt/tibco/ems/docker/security
```

Note: If build should fail, refer to the ***Note*** in Section 3.1.

```
> docker build -t ems:8.4.0_HF-006_security .
```

3. Upon customizing your EMS configuration, make sure to include the relevant paths to those files in the *Security Class path*, *JAAS Classpath* and *JACI Classpath* properties.
4. Note that 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/8.4/bin/tibemsd_jaas.jar:/opt/tibco/ems/8.4/lib/tibjmsadmin.jar`, etc.

3.3 Hosting the Image

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

For example:

```
> docker tag ems:8.4.0_HF-006 docker.company.com/path/ems
```

TIBCO®

```
> docker push docker.company.com/path/ems
```

4 Kubernetes Setup

4.1 Creating a Project

In Kubernetes, the command line tool, *kubectl,* can be used to configure the Kubernetes cluster for EMS. Another option is to install and use a Kubernetes Web Console. This document will only cover deploying EMS using the *kubectl* command line tool.

The first step of the setup consists of creating a new *namespace* in Kubernetes, if desired. This is optional. The *default* namespace can be used.

Note: If the namespace *ems-project* is not used, the provided yaml files and the examples shown below must be modified to the namespace used.

For example:

```
> kubectl create namespace ems-project
```

4.2 Provisioning the NFS Shared Folder

A storage resource is provisioned in Kubernetes by the cluster administrator through a *Persistent Volume* (PV), which we need 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 will create one PV and one PVC at the same time since these are meant to be bound together.

Adjust provided file **`nfs-pv-pvc.yaml`** to your particular 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,proto=tcp,nfsvers=4 (1)
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteMany
  nfs:
    path: / (2)
    server: 192.168.0.197 (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. The provided values are recommended.
(2) : The path that is exported by the NFS server. In this example, we want it to match the folder on the NFS server discussed in section 2.3.
(3) : The host name or IP address of the NFS server.
(4) : This needs to match the name of the project created previously.

Create the PV and PVC:

```
> kubectl create -n ems-project -f nfs-pv-pvc.yaml
```

Check the results with the following:

```
> kubectl get -n ems-project pv,pvc
```

Note that 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.

4.3 Adjusting the Services NodePort Range (Optional)

Services of type *NodePort* are used to expose EMS server listen ports outside the cluster (see section 4.4.2). The range of allowed values defaults to **`30000-32767`**. If you intend to use port numbers outside this range for the EMS server or Central Administration server, you can alter the range by using a *load balancer* in Kubernetes. See the Kubernetes documentation for details.

4.4 EMS Server Template

EMS server containers are created in a Kubernetes cluster through the provided **`tibemsd-template.yaml`** sample *template*. This template includes sections that define a limited set of *parameters*, a *deployment*, and a *service*.

4.4.1 Parameters Objects

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

`EMS_SERVICE_NAME:` The name of the service through which the EMS server is accessible inside the cluster. The default is *emstest01*.

`EMS_PUBLIC_PORT:` The port number through which the EMS server is accessible (both inside and outside the cluster). The default is *30722*.

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

However, unlike commercial versions of *Kubernetes,* the open version of Kubernetes cannot accept parameters for all values. This following should be updated in **`tibemsd-template.yaml`**

- Under *containers* in the *Deployment* section, modify *image* and *imagePullPolicy* for the location of the EMS Docker image in the Docker registry. Also, the pull policy if the image should *Never* or *Always* be pulled.
- Under the *Service* modify the *port* and *nodePort* to match the external port to be used. By default, *30722* is used.
- Throughout the template, *7222* is used for the internal EMS port. If *7222,* is not used, change the value accordingly.
- Change *runAsUser* to the **uid** the EMS server container must run as with respect to accessing NFS. The default is *1000.*
- For the value of *claimName,* us the name of the PVC previously configured to access the NFS shared folder. The default is *npvc-nfs-ems-project*

Note: The **`uid`** provided here must match that used when creating the EMS Docker image. This constraint should be removed in a future version of Kubernetes.

TIBCO®

4.4.2 Service Object 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, and if you have exposed your EMS server through a service using port number **`30722`**, EMS clients running outside the cluster will be able to access it either through the **`tcp://node1:30722`**, **`tcp://node2:30722`** or **`tcp://node3:30722`** URL, regardless of the node where the container is actually running. This works by virtue of each node proxying port **`30722`** 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 **`emstest01`** and the port still is **`30722`**, that amounts to using the **`tcp://emstest01:30722`** URL.

To ensure EMS client automated fault-tolerance failover, these must connect with FT double URLs. Using the example above: **`tcp://node1:30722,tcp://node1:30722`** from outside the cluster or **`tcp://emstest01:30722,tcp://emstest01:30722`** from inside the cluster. For the first form, since all nodes will proxy port **`30722`** 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.

As types of service other than NodePort may fit your requirements, feel free to explore those.

TIBCO®

4.4.3 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.

Key items:

```
kind: Deployment
…
spec:
  replicas: 1 (1)
  …
  strategy:
    type: Recreate (2)
    …
  template:
    …
    spec:
      containers:
      - name: tibemsd-container
        image: <EMS_IMAGE_LOCATION>
        imagePullPolicy: Always/Never (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: 7222
          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. Use *Always* to download the Docker image every time, or *Never* to use the existing image.
(4): Environment variables that will passed to the container.
(5): Arguments to be passed to the Docker **`ENTRYPOINT`** (see section 4.4.6).
(6): For details on the liveness and readiness probes, see Appendix A.
(7): The **`uid`** the container will run as. The default is *1000*.
(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 so that the *kubelet* will always try to restart an exited 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. The default is *npvc-nfs-ems-project*.

4.4.4 Creating a Deployment and Service

Create a deployment and service with an EMS server using the corresponding template. You will need to override a number of default parameter values to tailor the configuration for this particular instance.

For example:

```
> kubectl -n ems-project create -f tibemsd-template.yaml
```

The **`kubectl`** 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 **`create`** process for creation. In this particular case, it results in the creation of a deployment, a ReplicaSet, a pod and a service. All four objects can be selected through the **`emstest01`** label.

The service exposes itself as **`emstest01`** inside the cluster and maps internal port **`7222`** to port **`30722`** both inside and outside the cluster. PVC **`claim-nfs-ems-project`** is used to mount the NFS shared folders. Defaults values of other parameters are used to pull the EMS Docker image from the Docker registry and to set the **`uid`** of the container.

Check the results using the following:

```
> kubectl -n ems-project get --selector name=emstest01 all
> kubectl -n ems-project describe deploy/emstest01
> kubectl -n ems-project describe svc/emstest01
```

or in the Kubernetes Web Console (not documented).

TIBCO®

4.4.5 Stopping or Deleting an EMS Server

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

For example:

```
> kubectl -n ems-project scale --replicas=0 deploy emstest01
```

To start it again, set its number of replicas back to 1:

```
> kubectl -n ems-project scale --replicas=1 deploy emstest01
```

To delete an EMS server deployment and service entirely, use the **`kubectl delete`** operation. For example:

```
> kubectl -n ems-project delete --selector name=emstest01 deploy,svc
```

The corresponding pod and ReplicaSet will also be deleted.

4.4.6 EMS Server Configuration

As mentioned in section 3.1, running a container off of the EMS Docker image creates a default EMS server folder hierarchy and configuration. In a Kubernetes 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 **`tibemscreateimage`** 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. Feel free to alter **`tibems.sh`** or to directly provision your own configuration files to suit your needs.

TIBCO®

4.5 Central Administration Server Template

A Central Administration server container is created in an Kubernetes cluster through the **`tibemsca-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 the previous section also apply to the Central Administration server.

Note: Ensure to update the Docker image location from *"docker.company.com/path/ems",* and the external port number. The default is *30080.*

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

```
> kubectl -n ems-project create -f tibemsca-template.yaml
```

You can then use a Web browser to connect to **`http://node1:30080`** and add EMS servers to Central Administration.

Appendix A: Health Checks for the EMS Server

This appendix documents how EMS server container health checks are implemented using the sample **`TIB_ems_<version>_probes.zip`**. At this point, these rely on executing commands in the container.

A.1 Liveness and Readiness Probes

The Kubernetes and Kubernetes documentation describe health checks here:

https://docs.Kubernetes.com/container-platform/3.9/dev_guide/application_health.html
https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes
https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes

For an EMS server container, a liveness health check helps detect when an EMS server stops servicing client heartbeats but somehow does not exit. When this health check fails a number of times in a row, the EMS server container will be restarted.

A readiness health check helps detect when an EMS server stops servicing new client requests but is still up and running. 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. This could be useful for deliberately hiding an EMS server while it handles a long internal operation (e.g. performing a compact) that prevents it from servicing client requests. 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 to you, feel free to remove it from the template.

A.2 Implementation

Sample health checks are provided in **`TIB_ems_<version>_probes.zip`**, which includes a number of internal tools.

These rely on *server heartbeats* flowing from the EMS server to its clients as well as a *client timeout* of missing server heartbeats. Those are set through the following two EMS server properties:

- **`server_heartbeat_client`** (set to a value > 0)
- **`client_timeout_server_connection`** (set to a value > 0)

The implementation also relies on establishing an *admin client* connection to the EMS server. As a result, credentials for an *admin user* with at least the **`view-server`** *administrator permission* need to be configured. The admin user name and password will be used to configure the liveness and readiness probes. Additionally, if the EMS server is TLS only, TLS credentials will also be required when configuring the probes.

A.2.1 Liveness Health Check

The sample liveness probe is provided through the **`live.sh`** script configured in the deployment object (see section 4.4.3):

```
...
livenessProbe:
  exec:
    command:
    - live.sh
    - '-spawnTimeout'
    - '4'
    - '--'
    - '-server'
    - tcp://localhost:<EMS_INTERNAL_PORT>
    - '-user'
    - 'probeuser'
    - '-password'
    - 'probepassword'
  initialDelaySeconds: 1
  timeoutSeconds: 5
  periodSeconds: 6
...
```

Here the cluster will perform a periodic liveness check based on the **`live.sh`** script and the corresponding parameters.

The **`-spawnTimeout`** parameter is an internal timeout used by the probe that should be set relative to the probe's **`periodSeconds`** setting (see recommended settings in section A.2.4).

The above example matches the EMS server sample configuration. It should be tailored to your target configuration using the following additional probe parameters, when relevant:

`-server <server-url>`	Connect to specified server (default is tcp://localhost:7222).
`-timeout <timeout>`	Timeout of server request (in seconds) (default is 10).
`-delay <delay>`	Delay between server requests (in seconds) (default is 1).
`-user <user-name>`	Use this user name to connect to server (default is admin).
`-password <password>`	Use this password to connect to server (default is NULL).
`-pwdfile <passwd file>`	Use the password in the specified file (to hide it).
`-module_path <path>`	Path to find dynamic libraries such as SSL.
`-ssl_trusted <filename>`	File containing trusted certificate(s). This parameter may be entered more than once if required.
`-ssl_identity <filename>`	File containing client certificate and optionally extra issuer certificate(s) and private key.
`-ssl_issuer <filename>`	File containing extra issuer certificate(s) for client-side identity.
`-ssl_password <password>`	Private key or PKCS12 password.

-ssl_pwdfile <pwd file>	Use the private key or the PKCS12 password from this file.
-ssl_key <filename>	File containing private key.
-ssl_noverifyhostname	Do not verify host name against the name in the certificate.
-ssl_hostname <name>	Name expected in the certificate sent by host.
-ssl_trace	Show loaded certificates and certificates sent by the host.
-ssl_debug_trace	Show additional tracing.

A.2.2 Readiness Health Check

The sample readiness probe is provided through the **ready.sh** script configured in the deployment object (see section 4.4.3):

```
...
readinessProbe:
  exec:
    command:
    - ready.sh
    - '-spawnTimeout'
    - '4'
    - '-responseTimeout'
    - '4'
    - '--'
    - '-server'
    - tcp://localhost:<EMS_INTERNAL_PORT>
    - '-user'
    - 'probeuser'
    - '-password'
    - 'probepassword'
  initialDelaySeconds: 1
  timeoutSeconds: 5
  periodSeconds: 6
...
```

Here the cluster will perform a periodic readiness check based on the **ready.sh** script and the corresponding parameters.

The **-spawnTimeout** and **-responseTimeout** parameters are internal timeouts used by the probe that should be set relative to the probe's **periodSeconds** setting (see recommended settings in section A.2.4).

The above example matches the EMS server sample configuration. Just as with the liveness health check, it should be tailored to your target configuration using the same parameters.

A.2.3 Configuring Liveness and Readiness Probes

In addition to the above settings, Kubernetes provides settings to adjust the behavior of the probes:

`initialDelaySeconds`: Number of seconds after the container has started before the liveness or readiness probes are initiated.

`timeoutSeconds`: Number of seconds after which the probe times out. Defaults to 1 second. Minimum value is 1.

`periodSeconds`: How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.

These settings are individually set on each probe and affect the timing of health check events.

A.2.4 Recommended settings

`livenessProbe`	`initialDelaySeconds`:	1
	`periodSeconds`:	6 = <PERIOD> used below
	`timeoutSeconds`:	<PERIOD> -1
`readinessProbe`	`initialDelaySeconds`:	1
	`periodSeconds`:	<PERIOD>
	`timeoutSeconds`:	<PERIOD> - 1
`live.sh`	`-spawnTimeout`:	<PERIOD> - 2
`ready.sh`	`-spawnTimeout`:	<PERIOD> - 2
	`-responseTimeout`:	<PERIOD> - 2
EMS server configuration	`server_heartbeat_client`:	5
	`client_timeout_server_connection`:	20

The **`tibemsd-template.yaml`** sample template is already populated with recommended values.

Appendix B: TLS Configuration

This appendix takes you through the steps of modifying the EMS server template and 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 Kubernetes 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, we will assume we want the EMS server to 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`

B.1 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
> kubectl 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
```

You can check the result this way:

```
> kubectl describe secret tibemsd-secret
> kubectl export secret tibemsd-secret
```

B.2 Adjusting the Template

The **tibemsd-template.yaml** template needs to be adjusted to mount the secret as a volume. This involves adding one new entry to the **volumes** section and another one to the **volumeMounts** sections. We also need to alter the **livenessProbe** and the **readinessProbe** to connect to the server through **ssl**.

```
kind: Deployment
…
spec:
  …
  template:
    …
    spec:
      containers:
      - name: tibemsd-container
        …
        livenessProbe:
            …
            - ssl://localhost:7222
            …
        readinessProbe:
            …
            - ssl://localhost:7222
            …
        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
```

Note: You should eventually turn the **secretName** entry into a parameter.

B.3 Adjusting the `tibemscreateimage` EMS Docker image build script

In the **`tibemsd-configbase.json`** section:

Modify the **`primary_listen`** to use **`ssl`**:

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

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"
    },
```

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_passwor
d \`cat /etc/secret/ssl_password\`"
    fi
    …
```

B.4 Applying the Adjustments

- Regenerate the EMS Docker image, tag it and push it to the Registry (see section 3.1).
- Create a new deployment and service (see section 4.4.4).

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

```
> java tibjmsSSL -server ssl://node1:70222 \
                 -ssl_trusted ../certs/server_root.cert.pem \
                 -ssl_hostname server
```