**TomEE and WebSphere MQ**

**Steps to integrate TomEE with Websphere MQ**

1. Unzip rar file place jars under tomee/lib
2. Added the below to conf/tomee.xml

<tomee>

<Container id="wmq" type="MESSAGE">

ResourceAdapter=wmqRA

MessageListenerInterface=javax.jms.MessageListener

ActivationSpecClass=com.ibm.mq.connector.inbound.ActivationSpecImpl

</Container>

<Resource id="wmqRA" type="com.ibm.mq.connector.ResourceAdapterImpl" class-name="com.ibm.mq.connector.ResourceAdapterImpl">

connectionConcurrency=5

maxConnections=10

logWriterEnabled=true

reconnectionRetryCount=5

reconnectionRetryInterval=300000

traceEnabled=false

traceLevel=3

</Resource>

<Resource \*\*id="qcf"\*\* type="javax.jms.ConnectionFactory" class-name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl">

TransactionSupport=none

ResourceAdapter=wmqRA

HostName=10.a.b.c

Port=1414

QueueManager=QM\_TIERL

Channel=SYSTEM.ADMIN.SVRCONN

TransportType=Client

UserName=xyz

Password=\*\*\*\*\*

</Resource>

<Resource id="wmq-javax.jms.QueueConnectionFactory" type="javax.jms.QueueConnectionFactory" class-name="com.ibm.mq.connector.outbound.ManagedQueueConnectionFactoryImpl">

TransactionSupport=xa

ResourceAdapter=wmqRA

</Resource>

<Resource id="wmq-javax.jms.TopicConnectionFactory" type="javax.jms.TopicConnectionFactory" class-name="com.ibm.mq.connector.outbound.ManagedTopicConnectionFactoryImpl">

TransactionSupport=xa

ResourceAdapter=wmqRA

</Resource>

<Resource \*\*id="queue"\*\* type="javax.jms.Queue"

class-name="com.ibm.mq.connector.outbound.MQQueueProxy">

arbitraryProperties

baseQueueManagerName

baseQueueName

CCSID=1208

encoding=NATIVE

expiry=APP

failIfQuiesce=true

persistence=APP

priority=APP

readAheadClosePolicy=ALL

targetClient=JMS

</Resource>

<Resource id="wmq-javax.jms.Topic" type="javax.jms.Topic" class-name="com.ibm.mq.connector.outbound.MQTopicProxy">

arbitraryProperties

baseTopicName

brokerCCDurSubQueue=SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE

brokerDurSubQueue=SYSTEM.JMS.D.SUBSCRIBER.QUEUE

brokerPubQueue

brokerPubQueueManager

brokerVersion=1

CCSID=1208

encoding=NATIVE

expiry=APP

failIfQuiesce=true

persistence=APP

priority=APP

readAheadClosePolicy=ALL

targetClient=JMS

</Resource>

</tomee>

3. In web.xml add the below to access resources

<resource-ref>

<res-ref-name>myqcf< /res-ref-name>

<res-type>javax.jms.ConnectionFactory < /res-type>

<res-auth>Container</res-auth>< /br>

<res-sharing-scope>Shareable< /res-sharing-scope>

<mapped-name>qcf< /mapped-name>

</resource-ref>

<resource-env-ref>

<resource-env-ref-name>myqueue< /resource-env-ref-name>

<resource-env-ref-type>javax.jms.Queue< /resource-env-ref-type>

<mapped-name>queue< /mapped-name>

</resource-env-ref>

**Code:**

@Resource(name = "qcf")

private ConnectionFactory connectionFactory;

@Resource(name = "queue")

private Queue queue;

Connection connection = connectionFactory.createConnection();

Session session = connection.createSession(false, QueueSession.AUTO\_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(queue);

TextMessage message = session.createTextMessage();

message.setText("Test Message");

connection.start();

producer.send(message);

session.close();

connection.close();

# Install Docker on Ubuntu

## [Step 1 — Installing Docker](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04#step-1-installing-docker)

The Docker installation package available in the official Ubuntu repository may not be the latest version. To ensure we get the latest version, we’ll install Docker from the official Docker repository. To do that, we’ll add a new package source, add the GPG key from Docker to ensure the downloads are valid, and then install the package.

First, update your existing list of packages:

sudo apt update

Next, install a few prerequisite packages which let apt use packages over HTTPS:

sudo apt install apt-transport-https ca-certificates curl software-properties-common

Then add the GPG key for the official Docker repository to your system:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Add the Docker repository to APT sources:

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"

This will also update our package database with the Docker packages from the newly added repo.

Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

apt-cache policy docker-ce

You’ll see output like this, although the version number for Docker may be different:

Output of apt-cache policy docker-ce

docker-ce:

Installed: (none)

Candidate: 5:19.03.9~3-0~ubuntu-focal

Version table:

5:19.03.9~3-0~ubuntu-focal 500

500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages

Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 20.04 (focal).

Finally, install Docker:

sudo apt install docker-ce

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it’s running:

sudo systemctl status docker

The output should be similar to the following, showing that the service is active and running:

Output

● docker.service - Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)

Active: active (running) since Tue 2020-05-19 17:00:41 UTC; 17s ago

TriggeredBy: ● docker.socket

Docs: https://docs.docker.com

Main PID: 24321 (dockerd)

Tasks: 8

Memory: 46.4M

CGroup: /system.slice/docker.service

└─24321 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Installing Docker now gives you not just the Docker service (daemon) but also the docker command line utility, or the Docker client.

## [Step 2 — Executing the Docker Command Without Sudo (Optional)](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04#step-2-executing-the-docker-command-without-sudo-optional)

By default, the docker command can only be run the **root** user or by a user in the **docker** group, which is automatically created during Docker’s installation process. If you attempt to run the docker command without prefixing it with sudo or without being in the **docker** group, you’ll get an output like this:

Output

docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.

See 'docker run --help'.

If you want to avoid typing sudo whenever you run the docker command, add your username to the docker group:

sudo usermod -aG docker ${USER}

To apply the new group membership, log out of the server and back in, or type the following:

su - ${USER}

You will be prompted to enter your user’s password to continue.

Confirm that your user is now added to the **docker** group by typing:

groups

Output

sammy sudo docker

If you need to add a user to the docker group that you’re not logged in as, declare that username explicitly using:

sudo usermod -aG docker username

## [Step 3 — Using the Docker Command](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04#step-3-using-the-docker-command)

Using docker consists of passing it a chain of options and commands followed by arguments. The syntax takes this form:

docker [option] [command] [arguments]

To view all available subcommands, type:

docker

As of Docker 19, the complete list of available subcommands includes:

Output

attach Attach local standard input, output, and error streams to a running container

build Build an image from a Dockerfile

commit Create a new image from a container's changes

cp Copy files/folders between a container and the local filesystem

create Create a new container

diff Inspect changes to files or directories on a container's filesystem

events Get real time events from the server

exec Run a command in a running container

export Export a container's filesystem as a tar archive

history Show the history of an image

images List images

import Import the contents from a tarball to create a filesystem image

info Display system-wide information

inspect Return low-level information on Docker objects

kill Kill one or more running containers

load Load an image from a tar archive or STDIN

login Log in to a Docker registry

logout Log out from a Docker registry

logs Fetch the logs of a container

pause Pause all processes within one or more containers

port List port mappings or a specific mapping for the container

ps List containers

pull Pull an image or a repository from a registry

push Push an image or a repository to a registry

rename Rename a container

restart Restart one or more containers

rm Remove one or more containers

rmi Remove one or more images

run Run a command in a new container

save Save one or more images to a tar archive (streamed to STDOUT by default)

search Search the Docker Hub for images

start Start one or more stopped containers

stats Display a live stream of container(s) resource usage statistics

stop Stop one or more running containers

tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

top Display the running processes of a container

unpause Unpause all processes within one or more containers

update Update configuration of one or more containers

version Show the Docker version information

wait Block until one or more containers stop, then print their exit codes

To view the options available to a specific command, type:

docker docker-subcommand --help

To view system-wide information about Docker, use:

docker info

**JMS Configuration in Tomcat with IBM MQ**

Tomcat is Servlet container which support Servlets and JSPs. Lets connect Tomcat with Messaging provider IBM MQ9. In this we use MQ running in Docker(icr.io/ibm-messaging/mq) and ports binded to host system. Below are MQ objects.

Queue Manager : TomcatQM

Listener : SYSTEM.LISTENER.TCP.1

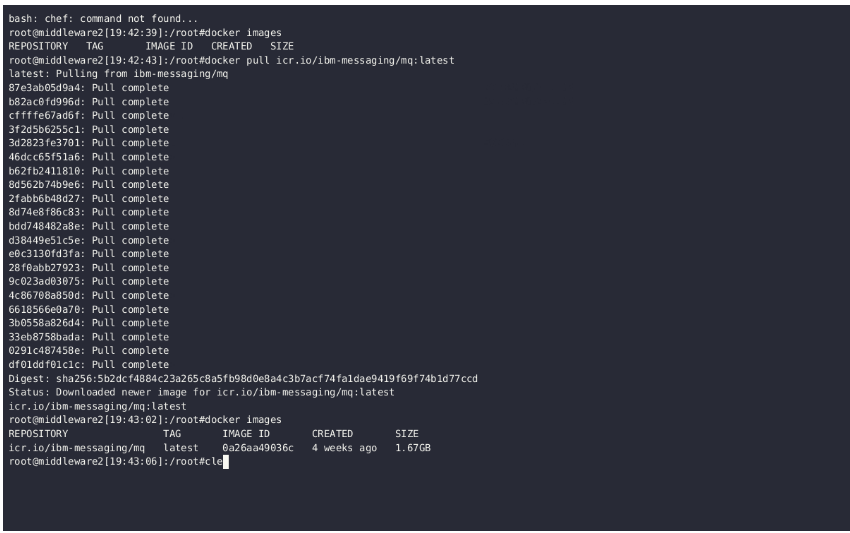
Channel : DEV.APP.SVRCONN

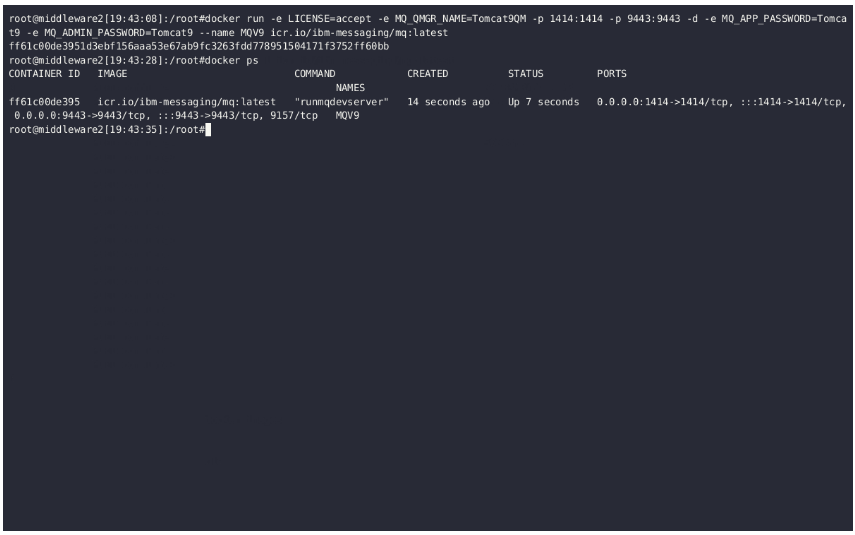
Queue: DEV.QUEUE.1

1. Get the MQ Docker image, start the container and check the logs with below commands

docker pull icr.io/ibm-messaging/mq:latest

docker run -e LICENSE=accept -e MQ\_QMGR\_NAME=TomcatQM -p 1414:1414 -p 9443:9443 -d -e MQ\_APP\_PASSWORD=Tomcat -e MQ\_ADMIN\_PASSWORD=Tomcat --name MQV9 icr.io/ibm-messaging/mq:latest

docker ps



2) Log into container and check MQ Details

doker exec ff61c00de395 -it /bin/bash

dspmpver -: Display MQ version

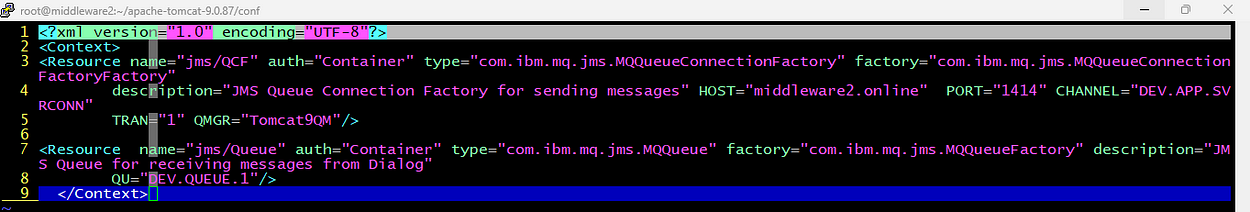
dspmq -: Display avaliable QueueManager and status

runmqc Commands are used to check MQ objects running inside QueueManager

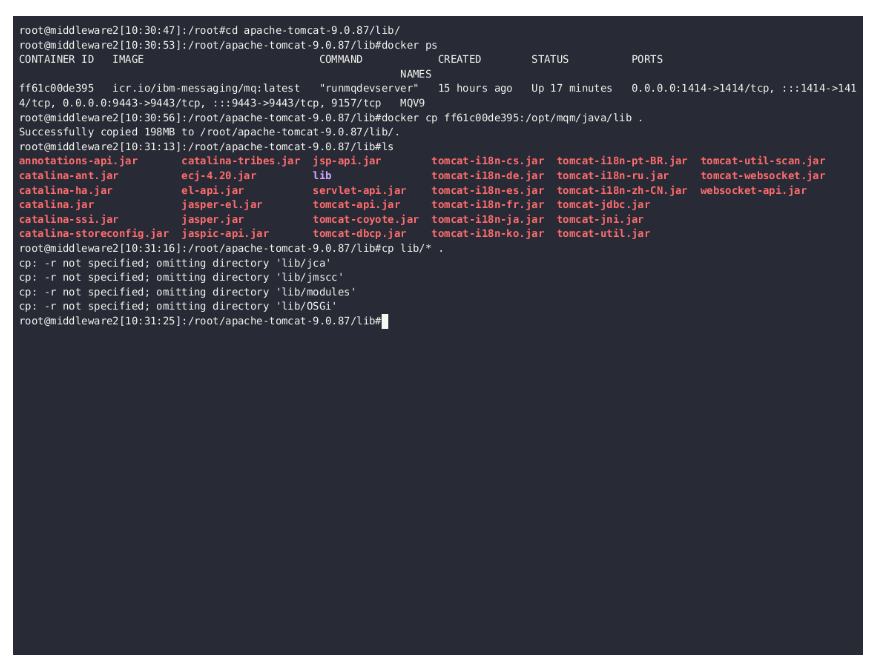
3) Download Tomcat and check the start-up

4) Create a file context.xml in /root/apache-tomcat/conf and add below stanza for resource configuration. Queue connection Factory and Queue

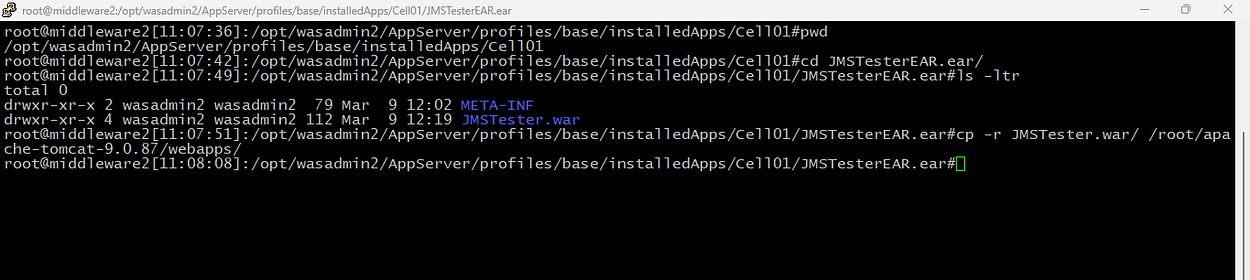
<?xml version="1.0" encoding="UTF-8"?>  
<Context>  
<Resource name="jms/QCF" auth="Container" type="com.ibm.mq.jms.MQQueueConnectionFactory" factory="com.ibm.mq.jms.MQQueueConnectionFactoryFactory"  
 description="JMS Queue Connection Factory for sending messages" HOST="middleware2.online" PORT="1414" CHANNEL="DEV.APP.SVRCONN"  
 TRAN="1" QMGR="TomcatQM"/>  
   
<Resource name="jms/Queue" auth="Container" type="com.ibm.mq.jms.MQQueue" factory="com.ibm.mq.jms.MQQueueFactory" description="JMS Queue for receiving messages from Dialog"  
 QU="DEV.QUEUE.1"/>  
 </Context>

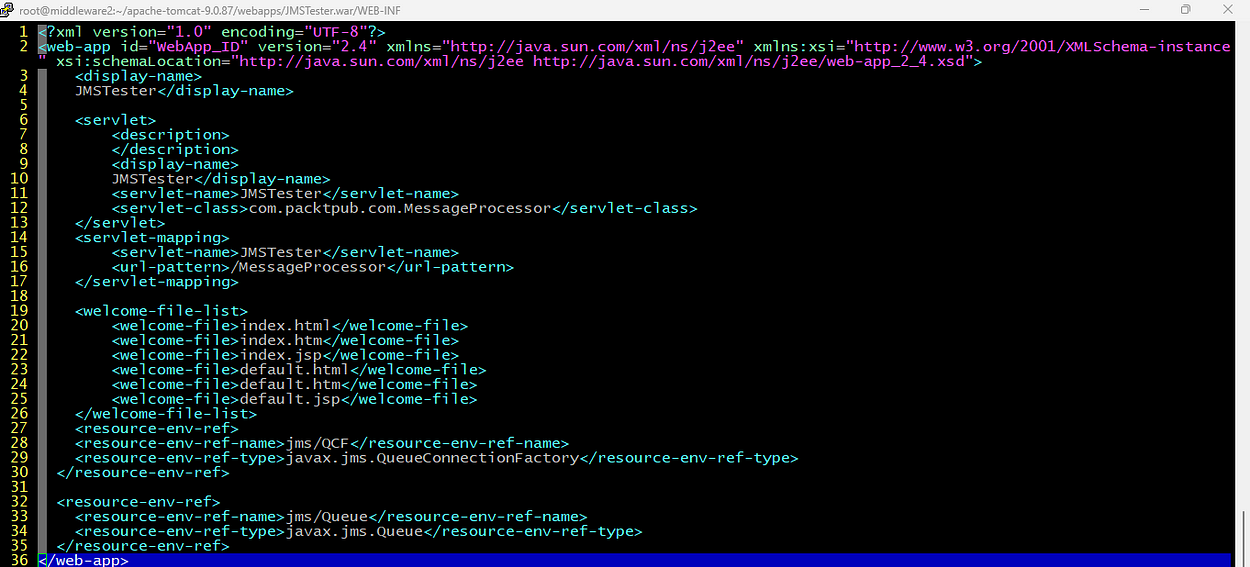


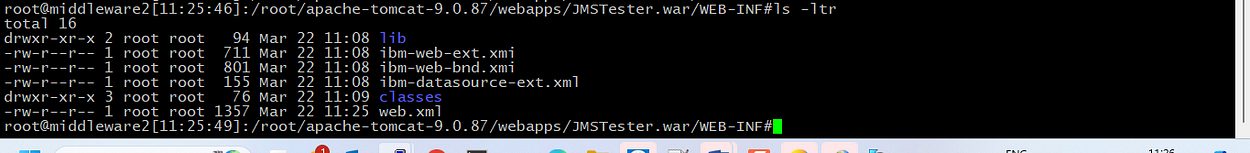
5) Below Jar files are required to connect to MQ successfully. Copy these jar files from running container to Tomcat Lib directory. com.ibm.mq.allclient.jar  
com.ibm.mq.commonservices.jar  
com.ibm.mq.headers.jar  
com.ibm.mq.jakarta.client.jar  
com.ibm.mq.jar  
com.ibm.mq.jmqi.jar  
com.ibm.mqjms.jar  
com.ibm.mq.pcf.jar  
com.ibm.mq.tools.ras.jar  
fscontext.jar  
com.ibm.mq.traceControl.jar  
jackson-annotations.jar  
jackson-core.jar  
jackson-databind.jar  
jakarta.jms-api.jar  
jms.jar  
org.json.jar  
providerutil.jar



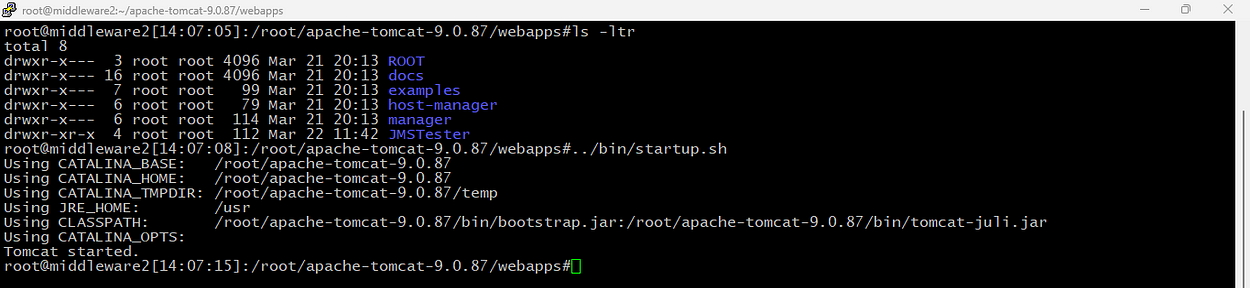
6) Copy Test Application deployed in WebSphere Application server 9 to /root/apache-tomcat/webapps and update web.xml

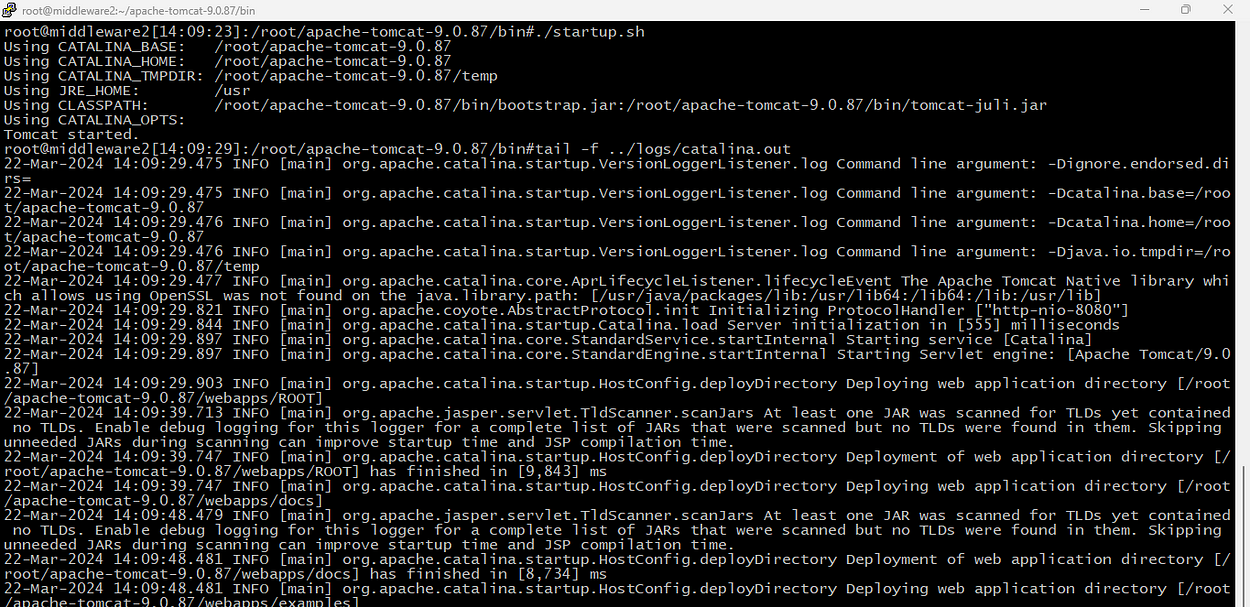


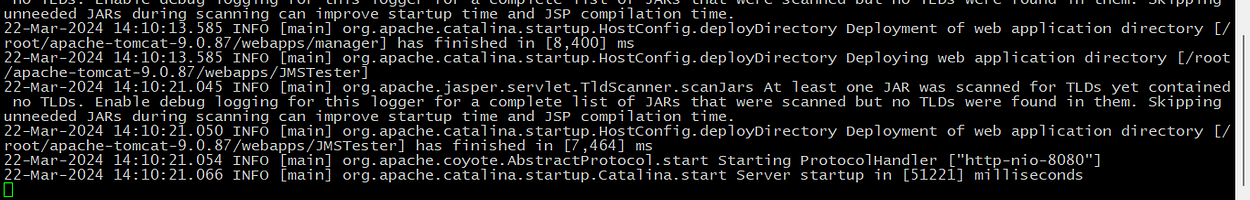




7)Rename the application name to JMSTester , Start Tomcat and check log file and access application







Connect to MQ container and execute the below mqsc commands to disable Password check while connecting to MQ

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(NONE) CHCKLOCL(NONE)

REFRESH SECURITY

8) Access application, put and get messages

