To see how IBM MQ works, you will be guided through creating and configuring a queue manager (server). Then, you will connect an application (client) to it in the next set of activities.

After completing the tutorial, you will be able to send messages to and retrieve messages from a queue.
You can download, install, and run IBM MQ queue manager (server) in a variety of ways:

- In a container (this tutorial)
- In the IBM Cloud, AWS, Microsoft Azure, or Google Cloud.
- On various operating systems: Linux/Ubuntu or Windows.

Learning objectives

After completing this tutorial, you will understand these concepts:

- IBM MQ queue managers
- IBM MQ queues

Step 1. Install Docker

Docker is already installed on the VDI environment. To verify, open the terminal on the VDI machine and run the following command to check the Docker version:

```
docker --version
```

If Docker is installed, it will display the Docker version information. If it's not installed, you'll likely see an error message.

Step 2. Get the MQ in Docker image

Containers are run from images and images are built from a specification listed in a Dockerfile. We will use a pre-built IBM MQ server image so that we can just run our container without having to build an image. We will end up with a working MQ installation and a queue manager that is preconfigured with objects ready for you to work with.

1. Pull the image from the IBM Container Registry that contains the latest version of the MQ server.
   ```
   docker pull icr.io/ibm-messaging/mq:latest
   ```

2. When it's done, check which images you have
   ```
   docker images
   ```

   You should see output like this:

```
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
icr.io/ibm-messaging/mq                     latest              8a2073f3babf        4 weeks ago         1.61 GB
my-connect-cluster-image                    latest              d1824baa7d5c        6 months ago        752 MB
cp.icr.io/cp/ibm-eventstreams-kafka         11.1.0              f64063c09980        10 months ago       724 MB
docker.io/edenhill/kafkacat                 1.6.0               18c4197f86da        3 years ago         23.7 MB
liberty-app-cache                           latest              516a5a50bb4f        3 years ago         874 MB
spring-app-cache                            latest              29b6d436c825        3 years ago         414 MB
docker.io/eclipse/codewind-performance-amd64  latest              f2c36fa539c5        3 years ago         334 MB
docker.io/eclipse/codewind-pfe-amd64        latest              debb84fb93a5        3 years ago         846 MB
docker.io/ibmjava                           8-sfj               e00902e30a47        3 years ago         167 MB
docker.io/eclipse/codewind-performance-amd64  0.11.0              711db7987738        3 years ago         338 MB
docker.io/eclipse/codewind-pfe-amd64        0.11.0              29df5ae49825        3 years ago         833 MB
docker.io/eclipse/codewind-performance-amd64  0.9.0               5b6569c30d13        3 years ago         334 MB
docker.io/eclipse/codewind-pfe-amd64        0.9.0               32f02842317e        3 years ago         784 MB
docker.io/websphere-liberty                 19.0.0.3-webProfile7  c6892fcebf3c        4 years ago         460 MB
```

Step 3. Run the container from the image

Now that the MQ server image is in your local image repository, you can run the container to stand up MQ in RHEL in a container.

When you stand up a container, an in-memory file system is used that is deleted when the container is deleted. Queue manager and queue data is saved in this file system. To avoid losing the queue manager and queue data, we can use Volumes.

Volumes are attached to containers when they are run and persist after the container is deleted. When you run a new container you can attach an existing volume and later reuse your queue manager and queue data.

1. Use Docker to create a volume
   `docker volume create qm1data`

2. Run the MQ server container. **Edit the command to set your own password for connecting applications**. You will need this password later, for when you run your own client applications. In this example, we're setting the password to "passw0rd" but you can also choose your own.

   ```
   docker run --env LICENSE=accept --env MQ_QMGR_NAME=QM1 --volume
   qm1data:/mnt/mqm --publish 1414:1414 --publish 9443:9443 --detach --env
   MQ_APP_PASSWORD=passw0rd --name QM1 icr.io/ibm-messaging/mq:latest
   ```

   Your queue manager has been set up with some simple default configuration to help you connect your application(s).

   We've added parameters to the docker run command, for example to accept the license for IBM MQ Advanced for developers and name the queue manager "QM1" where our queue(s) will live.

   Because MQ is running inside a container, it would be isolated from the rest of the world, so we've opened a couple of ports that MQ uses. The queue manager's listener listens on port 1414 for incoming connections and port 9443 is used by MQ console. The MQ application(s) will use the listener port and MQ Console dashboard can be seen in your browser on port 9443.

   Give the container a moment to start, and then check that it's running.
   `docker ps`

   You should see output like this:

   ```
   CONTAINER ID   IMAGE                          COMMAND             CREATED          STATUS          PORTS                                                      NAMES
   7128b7fed074   icr.io/ibm-messaging/mq:latest   "runmqdevserver"   9 seconds ago   Up 8 seconds   0.0.0.0:1414->1414/tcp, 0.0.0.0:9443->9443/tcp, 9157/tcp   QM1
   ```

   Congratulations! You've just created your first simple queue manager. It's called QM1, and it's running inside the container.

   To access this queue manager, you'll be connecting over TCP/IP, which is why you needed to expose the port 1414.

What you've done so far

You **downloaded the pre-built Docker image** and **ran the container** to get MQ running. The IBM MQ **objects and permissions** that the applications need to connect to a queue manager and to be able to put and get messages to and from the queue are **created automatically**. Docker and MQ are using your host computer resources and connectivity.

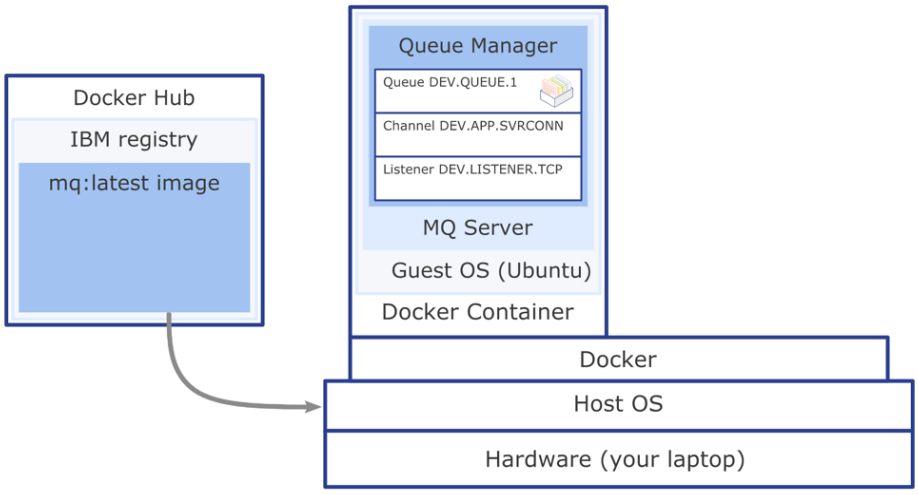
Inside the container, the MQ installation has the following objects:

- Queue manager QM1
- Queue DEV.QUEUE.1
- Channel: DEV.APP.SVRCONN
- Listener: SYSTEM.LISTENER.TCP.1 on port 1414

One of the queues that you will be using, **DEV.QUEUE.1**, "lives" on the **queue manager QM1**.
The **queue manager also has a listener** that listens for incoming connections, for example, **on port 1414**. **Client applications can connect** to the queue manager and **can open, put, and get messages, and close the queue**.

Applications use an **MQ channel** to connect to the queue manager. Access to these three objects is restricted in different ways. For example, **user "app", who is a member of the group "mqclient" is permitted to use the channel DEV.APP.SVRCONN to connect to the queue manager QM1 and is authorized to put and get messages to and from the queue DEV.QUEUE.1.**

All the MQ objects and permissions that the client application needs are created and configured when you run the MQ server container.



Step 4. Connect to Queue Manager on IBM MQ Explorer / MQ Console

Before you can create the connection on **IBM MQ Explorer**, you must know the following information about the remote queue manager:

- The name of the queue manager: **QM1**
- The name of the computer that hosts the queue manager: **localhost**
- The port number of the queue manager's listener: **1414**
- The name of the server-connection channel on the queue manager that IBM MQ Explorer uses to connect to the queue manager. **DEV.ADMIN.SVRCONN**

Lab 1 – Set up IBM MQ in a container

To manually create a connection from **IBM MQ Explorer** to a remote queue manager, complete the following steps.

1. On your VDI Machine menu bar, click Applications > Programming > IBM MQ Explorer
2. Right-click on Queue Managers in the Navigator view, then click **Add Remote Queue Manager**
   The Add Queue Manager wizard opens enabling you to create a connection.
3. In the **Queue manager name** field, type the name of the queue manager to which you want to connect. (see details above)
4. Ensure that **Connect directly** is selected, then click Next.
5. Ensure that **Specify connection details** is selected, then type the following details:
   - In the Host name or IP address field, type the name of the computer that hosts the remote queue manager (see details above)
   - In the **Port number** field, type the port number; (see details above)
   - In the **Server-connection channel** field, type the name of the channel to use (see details above)
6. Click Next twice
7. Tick the **Enable user identification** tick-box
   - In the **UserId** field type the relevant value (admin)
8. Once you click Finish, you will be prompted to provide a password. Enter the password created in Step 3 when creating the queue manager.
9. To view the Content click Window > Show View > MQ Explorer - Content. You should now be able to see 4 pre-configured queues

Congratulations! You've just connected to the Queue Manager you created from container.

To view the **IBM MQ Console**, complete the following steps:

1. Open Firefox and navigate to https://localhost:9443
2. You will be prompted with a Warning. Click Advanced > Accept the Risk and Continue
3. Log in to IBM MQ using the credentials provided in previous step (admin, passw0rd)
4. You should now be in the Home page, click Manage QM1 to view you local queues and other QM configuration

Congratulations! You've just connected to MQ Console dashboard for the Queue Manager you created from container.