1. Go to <https://github.com/apache/activemq-artemis> and download.
2. Copy the folder artemis-docker to your preferred devops folder for starting.

The contents of your folder should be like this.

|  |  |
| --- | --- |
| [**. .**](https://github.com/apache/activemq-artemis) | |
| [Dockerfile-adoptopenjdk-11](https://github.com/apache/activemq-artemis/blob/master/artemis-docker/Dockerfile-adoptopenjdk-11) |  |
| [Dockerfile-centos](https://github.com/apache/activemq-artemis/blob/master/artemis-docker/Dockerfile-centos) |  |
| [Dockerfile-debian](https://github.com/apache/activemq-artemis/blob/master/artemis-docker/Dockerfile-debian) |  |
| [docker-run.sh](https://github.com/apache/activemq-artemis/blob/master/artemis-docker/docker-run.sh) |  |
| [prepare-docker.sh](https://github.com/apache/activemq-artemis/blob/master/artemis-docker/prepare-docker.sh) |  |
| [readme.md](https://github.com/apache/activemq-artemis/blob/master/artemis-docker/readme.md) |  |

# **Docker Image Example**

This is an example on how you could create your own Docker Image for Apache ActiveMQ Artemis based on CentOS or Debian (JDK 8) or AdoptOpen JDK 11 (Ubuntu).

# **Preparing**

Use the script ./prepare-docker.sh as it will copy the docker files under the binary distribution.

You have two ways to prepare the build.

## Locally (I don’t prefer).

## Remotely.

The command to prepare the build of the Docker Image starting from the official release of ActiveMQ Artemis is shown below

# Prepare for build the Docker Image from the release version. Replace the

# {release-version} with the version that you want

#$ ./prepare-docker.sh --from-release --artemis-version {release-version}

$ ./prepare-docker.sh --from-release --artemis-version 2.17.0

The output of the previous command is shown below.

Downloading apache-artemis-2.17.0-bin.tar.gz from https://downloads.apache.org/activemq/activemq-artemis/2.16.0/...

######################################################################## 100,0%

Expanding \_TMP\_/artemis/2.17.0/apache-artemis-2.17.0-bin.tar.gz...

Removing \_TMP\_/artemis/2.17.0/apache-artemis-2.17.0-bin.tar.gz...

Using Artemis dist: \_TMP\_/artemis/2.17.0

Docker file support files at : \_TMP\_/artemis/2.17.0/docker

\_TMP\_/artemis/2.16.0/docker

├── Dockerfile-adoptopenjdk-11

├── Dockerfile-centos

├── Dockerfile-debian

└── docker-run.sh

0 directories, 4 files

Well done! Now you can continue with the Docker image build.

Building the Docker Image:

Go to \_TMP\_/artemis/2.17.0 where you prepared the binary with Docker files.

# Go to \_TMP\_/artemis/2.17.0

$ cd \_TMP\_/artemis/2.17.0

# For Debian

$ docker build -f ./docker/Dockerfile-debian -t artemis-debian .

# For CentOS

$ docker build -f ./docker/Dockerfile-centos -t artemis-centos .

# For AdoptOpen JDK 11

$ docker build -f ./docker/Dockerfile-adoptopenjdk-11 -t artemis-adoptopenjdk-11 .

Note: -t artemis-debian, -t artemis-centos and artemis-adoptopenjdk-11 are just

tag names for the purpose of this guide

# **Environment Variables**

Environment variables determine the options sent to artemis create on first execution of the Docker container. The available options are:

**ARTEMIS\_USER**

The administrator username. The default is artemis.

**ARTEMIS\_PASSWORD**

The administrator password. The default is artemis.

**ANONYMOUS\_LOGIN**

Set to true to allow anonymous logins. The default is false.

**EXTRA\_ARGS**

Additional arguments sent to the artemis create command. The default is --http-host 0.0.0.0 --relax-jolokia. Setting this value will override the default. See the documentation on artemis create for available options.

**Final broker creation command:**

The combination of the above environment variables results in the docker-run.sh script calling the following command to create the broker instance the first time the Docker container runs:

${ARTEMIS\_HOME}/bin/artemis create --user ${ARTEMIS\_USER} --password ${ARTEMIS\_PASSWORD} --silent ${LOGIN\_OPTION} ${EXTRA\_ARGS}

Note: LOGIN\_OPTION is either --allow-anonymous or --require-login depending on the value of ANONYMOUS\_LOGIN.

# **Mapping point**

* /var/lib/artemis-instance

It's possible to map a folder as the instance broker. This will hold the configuration and the data of the running broker. This is useful for when you want the data persisted outside of a container.

🡺Note: Here on windows 10, all images are incapsulated in a single file found in %AppData%\Local\Docker\wsl

# **Lifecycle of the execution**

A broker instance will be created during the execution of the instance. If you pass a mapped folder for /var/lib/artemis-instance an image will be created or reused depending on the contents of the folder.

# **Running a CentOS image**

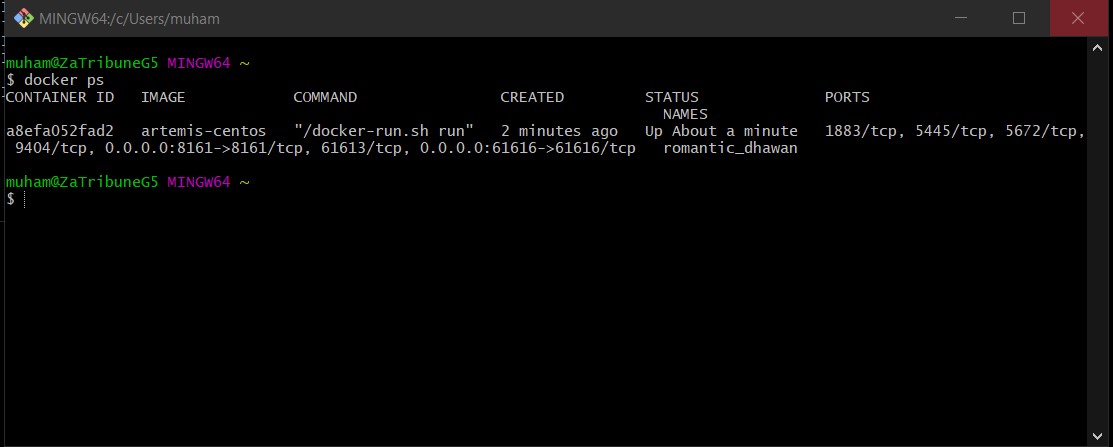
The image just created in the previous step allows both stateless or stateful runs. The stateless run is achieved by:

$ docker run --rm -it -p 61616:61616 -p 8161:8161 artemis-centos

The image will also support mapped folders and mapped ports. To run the image with the instance persisted on the host:

$docker run -it -p 61616:61616 -p 8161:8161 -v <broker folder on host>:/var/lib/artemis-instance artemis-centos

where <broker folder on host> is a folder where the broker instance is supposed to be saved and reused on each run.



We’re listening on **8161** which is the web console.

**61613 or 61616** -> the port we’ll be sending messages over.

Configuration on Spring Boot:

@Configuration

public class JmsConfig {

String BROKER\_URL = "tcp://localhost:61616";  
String BROKER\_USERNAME = "artemis";  
String BROKER\_PASSWORD = "artemis";  
  
//configure the broker's connection  
@Bean  
public ActiveMQConnectionFactory activeMQConnectionFactory() throws JMSException {  
 ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory();  
 connectionFactory.setBrokerURL(BROKER\_URL);  
 connectionFactory.setUser(BROKER\_USERNAME);  
 connectionFactory.setPassword(BROKER\_PASSWORD);  
 *log*.info("Broker Connection Factory created.");  
 return connectionFactory;  
}

}