Dockerize your Mulesoft API’s

Mitch Dresdner

Table of Contents

Summary

In his 24 May 2018 blog post, Aaron Landgraf shares a broad brush of capabilities in the Mule Titan release *full story here*.

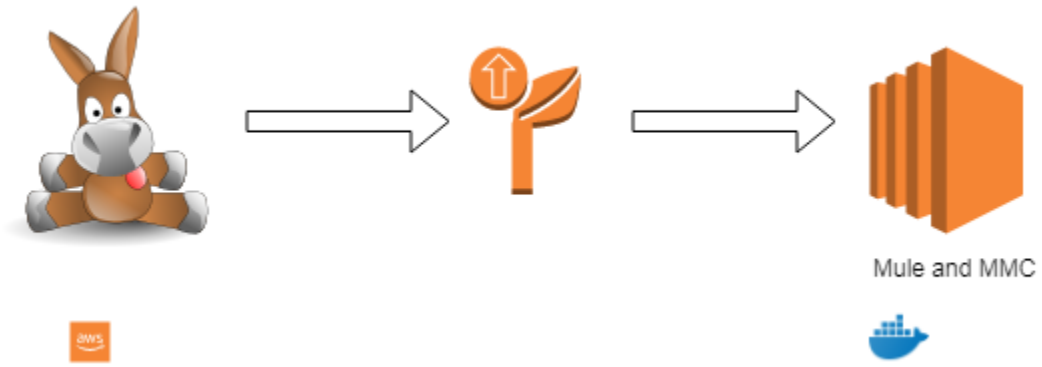
Landgraf describes the new *Anypoint Runtime Fabric* for creating, deploying and managing Mule containers in the Cloud, see excerpt below.

> *Anypoint Runtime Fabric is a container service that makes it easy to manage multi-cloud deployments of Mule runtimes and to deploy multiple runtime versions on the same Runtime Fabric. By leveraging Docker and Kubernetes, Anypoint Runtime Fabric drastically simplifies the deployment and management of Mule runtimes on Microsoft Azure, Amazon Web Services (AWS), virtual machines (VMs), and physical servers. Isolate apps, scale horizontally, and redeploy with zero downtime.*

Our project shares many similar goals articulated above, but is limited to achieving them with a more austere set of tools. This article will share how we Dockerize our Mulesoft 3.9 Enterprise edition and deploy in AWS.

Architecture

Our target Architecture will consist of deploying Mule applications from AnypointStudio into Docker containers running inside an AWS AMI image.



In the steps below you'll configure an AWS EC2 instance and configure the environment for Docker. With Docker installed you'll create images for Mule and MMC, run the containers, associate the Mule runtime with the MMC and deploy a simple *Hello Mule* application.

Docker containers are ephemeral by nature, which means that any changes to container state will be lost when a container is stopped. This isn't desirable and a quick remedy will be to use Docker volumes to persist and changes we need between container starts. The information which will be stateful is specified by the **VOLUME** tag in the Docker files below.

Prerequisites

We assume you're familiar with Mule and have some familiarity setting up EC2 instances in AWS.

If you're not confident with AWS or would like a little refresher please review the following articles:

Some refreshers before getting started

- Provision a free tier EC2 instance
- Configure Docker on your EC2 instance

Caveats

Some AWS services will incur charges, be sure to stop and/or terminate any services you aren't using. Additionaly, consider setting up billing alerts to warn you of charges exceeding a threshold that may cause you concern.

Configuration

We begin by copying the distribution files for MMC and Mule runtime to our EC2 instance, it will probably be easiest to use *wget* from the EC2 instance to download them directly. Other options include *scp* from your development environment or copy from an S3 bucket. Use the approach you feel most comfortable with.

In my EC2-User home folder I use the following hierarchy for my Dockerfile source:

Folder structure for builds

```
ls src/docker/mule

  MMC-3.8.0  MuleEE-3.9.0

ls src/docker/mule/MMC-3.8.0

  Dockerfile  mmc-3.8.x-web.tar.gz  start.sh

ls src/docker/mule/MuleEE-3.9.0

  Dockerfile  mule-enterprise-standalone-3.9.0  mule-ee-distribution-standalone-
3.9.0.tar.gz
```

Note that the tar file has been expanded and we have a folder for **mule-enterprise-standalone-3.9.0**.

The reason for this is that I install our custom EE license, make any changes to configuration files unique to our environments and repackage the tar file for creation of the Mule Docker image.

Install Mule EE license

Apply local changes to Mule configuration files

```
tar xzf mule-ee-distribution-standalone-3.9.0.tar.gz

export MULE_HOME=~/src/docker/mule/MuleEE-3.9.0/mule-enterprise-standalone-3.9.0

cd $MULE_HOME

# apply mule license
bin/mule -installLicense _path_to_your_license.lic

# re-create tar
tar czf mule-ee-distribution-standalone-3.9.0.tar.gz mule-enterprise-standalone-3.9.0
```

The Docker volumes expect to preserve stateful information under /opt, so the folder structure and permissions will need to be set up. My permissions are wide open, you may prefer to create an EC2 Mule user and group to apply stricter access control. If you do i'm confident you'll do so successfully on your own.

Folder structure for docker volumes

```
sudo mkdir /opt/mmc
sudo mkdir /opt/mmc/logs
sudo mkdir /opt/mmc/mmc-data
sudo chmod -R 777 /opt/mmc
sudo mkdir /opt/mule-enterprise-standalone-3.9.0
sudo mkdir /opt/mule-enterprise-standalone-3.9.0/apps
sudo mkdir /opt/mule-enterprise-standalone-3.9.0/conf
sudo mkdir /opt/mule-enterprise-standalone-3.9.0/domains
sudo mkdir /opt/mule-enterprise-standalone-3.9.0/logs
sudo chmod -R 777 /opt/mule-enterprise-standalone-3.9.0
sudo ln -s /opt/mule-enterprise-standalone-3.9.0 /opt/mule

ls -l /opt

  drwxrwxrwx. 4 root root     34 Feb  2 15:43 mmc
  drwxrwxrwx. 6 mule mule     57 May 24 15:05 mule-enterprise-standalone-3.9.0
```

My open AWS Mule ports look like this:

Table 1. Open ports in AWS Mule SG

Type	Protocol	Port Range	Source	Description
Custom TCP Rule	TCP	8585	0.0.0.0/0	Mule MMC Agent
Custom TCP Rule	TCP	8082	0.0.0.0/0	Mule API for HTTP

Dockerizing the Mule Management Console (MMC)

Dockerfile for MMC

```
FROM java:openjdk-8-jdk

MAINTAINER Your Name <me@myaddress.com>

USER root

WORKDIR /opt
RUN useradd --user-group --shell /bin/false mule && chown mule /opt

COPY    ./mmc-3.8.x-web.tar.gz /opt
COPY    ./start.sh /opt

# Using the most recent MMC 3.8.x version
RUN     tar xzf mmc-3.8.x-web.tar.gz \
          && ln -s mmc-3.8.x-web mmc \
          && chmod 755 mmc-3.8.x-web \
          && chmod 755 start.sh \
          && rm mmc-3.8.x-web.tar.gz

# Mule environment vars
ENV MMC_HOME /opt/mmc

# Volume mount points
VOLUME ["/opt/mmc/apache-tomcat-7.0.52/logs", "/opt/mmc/apache-tomcat-7.0.52/conf",
"/opt/mmc/apache-tomcat-7.0.52/bin", "/opt/mmc/apache-tomcat-7.0.52/mmc-data"]

# Mule work directory
# WORKDIR /opt

USER mule

# start tomcat && tail -f /var/lib/tomcat7/logs/catalina.out
CMD [ "./start.sh" ]

# Expose default MMC port
EXPOSE 8585
```

When the MMC Docker container starts, it will run the Tomcat server and tail the log contents to stdout.

Create the *start.sh* script below in you MMC folder, it will be added to the Docker image and will keep the container running after it's started in the step below.

start.sh file in MMC folder

```
#!/bin/sh

# If the apache-tomcat location is different for you, be sre to change
cd /opt/mmc/apache-tomcat-7.0.52

bin/startup.sh && tail -f logs/catalina.out
```

The initial build may take a while to complete as it needs to pull down the image layers from the Docker hub and create an image.

Build your MMC Docker container

```
# Change maxmule at end of next line to your Docker image name and optionally tag
docker build -t maxmule/mmc .
```

When the build successfully completes we can start the MMC container instance and use the browser to connect to it.

Run your MMC Docker container

```
# Change maxmule at end of next line to your Docker image name
docker run -itd --name mmc -p 8585:8585 -v /opt/mmc/mmc-data:/opt/mmc/apache-tomcat-
7.0.52/mmc-data -v /opt/mmc/logs:/opt/mmc/apache-tomcat-7.0.52/logs maxmule/mmc
```

It may take a while for the MMC to start up, you can use the Docker *logs* command to see when startup has completed. The **Ctrl-C** command will terminate the earlier logs command.

Ensure MMC is running

```
docker ps

docker logs -f mmc
^C
```

Dockerizing the Mule Runtime

Dockerfile for Mule

```
FROM java:openjdk-8-jdk

# 3.9.0 ee branch

MAINTAINER Your Name <me@myaddress.com>

USER root

WORKDIR /opt
RUN useradd --user-group --shell /bin/false mule && chown mule /opt

COPY    ./mule-ee-distribution-standalone-3.9.0.tar.gz /opt
COPY    ./start.sh /opt

RUN     tar xzf mule-ee-distribution-standalone-3.9.0.tar.gz \
          && ln -s mule-enterprise-standalone-3.9.0 mule \
          && chmod 755 mule-enterprise-standalone-3.9.0 \
          && chown -R mule:mule mule-enterprise-standalone-3.9.0 start.sh \
          && chmod 755 start.sh \
          && rm mule-ee-distribution-standalone-3.9.0.tar.gz


# Mule environment vars
ENV MULE_HOME /opt/mule
ENV PATH $MULE_HOME/bin:$PATH

# Volume mount points for persisten storage, create others for domains and conf if
necessary
VOLUME ["/opt/mule/logs", "/opt/mule/apps"]

USER mule

ENTRYPOINT ["mule"]
CMD ["console"]

# Expose port 7777 if you plan to use MMC
EXPOSE 7777

# Expose additional ports as needed for your API use
#EXPOSE 8081
EXPOSE 8082
#EXPOSE 8083
```

Similar to how we built and started the MMC above we'll follow the same steps with our Mule container.

Mule will be using port 8082 for working with our API deployment and port 7777 for

communicating with the MMC.

Build your Mule Docker container

```
# Change maxmule at end of next line to your Docker image name and optionally tag
docker build -t maxmule/mule39ee .
```

The build step is the same as we did erlier for the MMC

Build your Mule Docker container

```
# Change maxmule at end of next line to your Docker image name
docker run -itd --name mule -p 8082:8082 -v /opt/mule/apps:/opt/mule/apps -v
/opt/mule/logs:/opt/mule/logs maxmule/mule39ee
```

When build has successfully completed we run our Mule runtime instance and verify it has properly started.

Ensure Mule is running

```
docker ps

docker logs -f mule
^C
```

Deploying a model

Create a simple HTTP flow in AnypointStudio which listens on Port 8082, a port which you expose in your Docker file and the EC2 instance security group. Run the flow in AnypointStudio first to ensure it will work when deployed to your Docker container.

This concludes our brief examples with Dockerizing the Mule.

I hope you enjoyed reading this article as much as I have enjoyed writing it, i'm looking forward to your feedback!

About the Author:

Mitch Dresdner is a Senior Mule Consultant at TerraThink