Creating a Scalable Containerized IoT Network Topology

Nathaniel Holeksa

August 2018

1 Introduction

MAJOR NOTE: If at any point you run into permissions issues with a bash command, I recommend that you rerun the command starting with 'sudo' and it may resolve the issue. I will be referencing files from my github, so please follow along there as well.

2 Virtual Machines

2.1 What is a Virtual Machine?

A virtual machine is essentially an isolated computer within a computer. You can run a different operating system on a partition of your hard drive, and run this in parallel with your computer and other virtual machines.

In our case, it will allow you to have different services isolated from one another, that are still able to interact. The operating system that we are going to be installing is ubuntu-16.04.1-server-amd64.iso which you can download from Ubuntu. Ensure that the version you download is the exact one listed above to avoid any compatibility issue. You also need to install VirtualBox to create your virtual machine.

2.2 VM Setup

2.2.1 VirtualBox

I am not going to delve into the setup as this resource is exactly the same setup we will be using.

2.2.2 Savi Testbed

3 Kafka/Zookeeper

3.1 What is Kafka?

Kafka is a distributed streaming platform that allows you to publish messages to different topics and have consumers pull from specific topics. In our case, Kafka is the middleman between the gateway and the spark streaming jobs. All of the gateways will publish to one topic and the spark job will pull the data from that topic.

3.2 What is Zookeeper?

Zookeeper is a effectively a coordinator for distributed systems that is not extremely relevant to the architure we aim to produce, but is necessary to run kafka. The reason it is not as relevant is that we are not interested in the scaling of kafka in this topology and so the service will not be distributed, it will be local to one virtual machine.

3.3 Setup

Kafka makes the setup very simple for you as it bundles zookeeper in with its download, so there is no need to download Zookeeper separately. To install kafka, run the following commands on the virtual machine on which you wish to install it.

```
wget http://apache.mirror.rafal.ca/kafka/1.1.0/kafka_2.11-1.1.0.tgz
tar -xzf kafka_2.11-1.1.0.tgz
cd kafka_2.11-1.1.0
```

Now that kafka and zookeeper are installed, we must setup the configurations

```
echo "listeners=PLAINTEXT://<your.host.ip>:9092" >> config/properties
```

NOTE: The host ip will be the inet addr listed under enp0s8 when you run the command ifconfig
NOTE: This allows us to listen to kafka from a spark job

Now we start the services and test that it is working:

Note that you should have several terminal or PuTTY windows connected to the virtual machine to run the following commands as the servers give continuous output and so you can only run one command from those windows. The producer and consumer also each require their own window. Also ensure that the zookeeper server is started before the kafka server. Likewise, when you shut them down, shut down kafka first.

Finally, it is very important that you use the host ip and NOT just put localhost:2181 as this will conflict with the configuration that we added to server properties.

Window 1

```
./bin/zookeeper-server-start.sh config/zookeeper.properties
```

Window 2

```
./bin/kafka-server-start.sh config/server.properties
```

Window 3

```
./bin/kafka-topics.sh --create --zookeeper <your.host.ip>:2181 --partitions 1
    --replication-factor 1 --topic <topic.name>
./bin/kafka-console-consumer.sh --bootstrap-server <your.host.ip>:9092 --topic <topic.name>
```

Window 4

```
./bin/kafka-console-producer.sh --broker-list <your.host.ip>:9092 --topic <topic.name>
```

Now you should be able to type messages into the producer and receive them in the consumer window.

4 Cassandra

4.1 What is Cassandra?

Cassandra is a NoSQL database that can be distributed across multiple servers. Again this functionality is not as important to the task as we do not wish to scale the Cassandra server.

4.2 Setup

Run the following commands on the desired virtual machine, we will be installing version 2.2.12

```
echo "deb http://www.apache.org/dist/cassandra/debian 22x main" | sudo tee -a
    /etc/apt/sources.list.d/cassandra.sources.list
curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
sudo apt-get update
sudo apt-get install cassandra
```

To check the status/start/stop:

```
sudo service cassandra status
sudo service cassandra start
sudo service cassandra stop
```

Now we need to edit some configurations:

```
vim /etc/cassandra/cassandra.yaml
# find start_rpc and ensure it is set to true
# make sure listen_address is set to the host ip
```

If the server was already running make sure to restart it after editing the configurations.

To query database, begin with:

```
cqlsh <host.ip>
```

Then you can run any cql commands you wish.

If you encounter other errors, check the log files located in /var/log/cassandra

5 Spark

5.1 What is Spark?

Spark is a cluster-computing software that allows for fast data analytics in a distributed system. This is the main focus of our research and what we are interested in scaling.

5.2 Setup

Installation: Both master and workers need to run this step if they are on different virtual machines.

```
wget https://archive.apache.org/dist/spark/spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz
tar xvf spark-2.3.0-bin-hadoop2.7.tgz
```

On the master, run the following:

```
cd spark-2.3.0-bin-hadoop2.7
./sbin/start-master.sh
```

On the worker, run the following

```
cd spark-2.3.0-bin-hadoop2.7
./sbin/start-slave.sh spark://<spark-master-hostname>:7077 (optional) --cores <num cores>
    (optional) --memory <memory>
```

When submitting a job, it the following needs to be run only on the master:

```
./bin/spark-submit --class <class-name> --master spark://<spark-master-hostname>:6066
    --deploy-mode cluster <path-to-jar>
```

NOTE: it is important to run on port 6066 as this is the port for cluster deployment.

5.3 Writing a Spark Streaming Job from Kafka to Cassandra

The main steps to writing a spark streaming job are as follows:

(a) Start a sparkConf. Make sure to specify an AppName and the spark.cassandra.connection.host ip at the very least.

(b) Initialize a streaming context with the sparkConf and a time for each iteration of the job.

(c) Create a Map[String,Object] for Kafka input parameters. Also have an Array of the topics you are pulling from.

(d) Create a Properties object for output parameters if you are going to send any data back to Kafka.

(e) Create a direct stream from Kafka

(f) Process the stream

(g) Perform operations you wish to on the processed stream

(h) Save the data to Cassandra

(i) Start the streaming context and await termination

In my github there is an example file of both the basic processing and the security jobs.

5.4 Building the JAR

It is important that when you are building a jar that you package all of the dependencies from the xml file with the jar, so the job can run. To do so you should go to the maven menu in intellij, run clean, scala:compile, assembly:single and it should package properly.

6 Docker

6.1 What is Docker?

Docker is a container program that allows you to run and scale services in a lightweight, isolated environment. Essentially, a Docker container is like a virtual machine, but it relies on its host operating system, allowing it to be easier to deploy and less resource demanding.

6.2 Setup

Download docker
You will need to download docker in both you development environment as well as your runtime environment. It is advisable to use the same version in both places, so please do the following on both machines. Feel free to download the latest version.
Development environment: Follow the link to Docker and download your respective version
Runtime environment:

```
sudo apt-get update
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
sudo apt-get update
sudo apt-get install docker-ce
```

Now that you have docker installed in both locations it is time to write a Dockerfile for a Docker image. A Dockerfile is a set of instructions telling Docker how to deploy your container, it may copy jar files or other executables that you may then run in the container, expose ports that are necessary, set environment variables, etc. A Docker image is a snapshot of the container, if you start an image, you are running a container of this image, but you can have as many containers of this image as you would like.
NOTE: the Dockerfile must be named Dockerfile, that is how docker locates it.

After writing a Dockerfile you must construct the docker image. Generally you want to make a directory with the Dockerfile and all of its dependencies first.

Build the image

```
docker build -t <image-name>:<optional-tag> <Path-to-dockerfile-directory>
```

Now you have created your docker image, but wouldn't it be great if you could share with other people? Good news, you can! Create an account on docker hub, then you can log in.

```
docker login
# You will be prompted for username and password
# Now you can tag your image and push it to docker hub
docker tag <image-name> username/repository:tag
docker push username/repository:tag
```

Now people are able to run your services from remotely!
Docker-compose is a higher level set of instructions that allow you to deploy multiple services at once, along with networks that may or may not connect any two containers.
NOTE: Spacing is key in a docker-compose file, if the spacing is wrong you will get an error.
NOTE: Many more options are available at the docker-compose reference

7 Putting it all together

Now that you have created all of the services individually and confirmed that they run as you expect, it is time to run them together! Ensure that Kafka and Zookeeper are running in one VM and that Cassandra is running on the other VM. Now we will deploy the docker spark job in another container. Assuming you have created docker images for the spark master and spark workers that are available from docker hub and you have a docker-compose file on the VM you wish to deploy,

use the following commands.

```
docker swarm init
docker stack deploy -c <path-to-docker-compose> <stack-name>
# You can check the status of the stack with
docker stack ps <stack-name>
# You can check the port mappings with
docker service ls
# If you need to go into a container run
docker exec -it <container-id> /bin/bash
```

Now your processes should deploy and in a minute or two they should be running properly. You are ready to connect your run your sensors and gateway and start analytics. Good luck!