Creating a Scalable Containerized IoT Network Topology

Nathaniel Holeksa, Simon Nadeau

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 Setup Savi Testbed

If you want to have all the topology and the IoT Virtual Lab in VMs on the Savi testbed, it is simple to create and setup VMs on Savi. Everything is explained in this documentation. You first need to import your key pair (1.2.1) following Appendix A.1 to be able to use it at the moment of launching a VM. You don't need to create a security group (1.2.2) unless you want to use something else than the default one.

After that, you can launch a VM following step 1.3. Choose the "Instance Name" and the "Flavor" you want. For the "Image name", if it is a VM where you want to use Docker, select "Ubuntu-16-04-OVS-DOCKER". If you do not intend to use Docker, you can select "Ubuntu-16-04". Anyway, you always can install Docker following this documentation. To be able to run docker commands without sudo, you can follow those quick steps (the "Manage Docker as a non-root user" only): There is more explanation about Docker in section 6 of this document.
To give access to your VM to someone else, you only have to add his public key to the file /.ssh/authorized_keys in the VM right under yours.

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 architecture 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
```

8 IoT Lab Setup

Now that you have the three VMs (one (without Docker) for Kafka, one with Docker for Spark and one (without Docker) for Cassandra) to process the data, you are ready to setup the IoT Virtual Lab. You can already create two more VMs: one with Docker for the sensors and one with Docker for the gateways. *after you create all your VMs you may need to manually add the server hostnames and ips to each of the /etc/hosts file in each VM to make them discoverable if you don't have a DNS server in your network On all VMs with Docker, enable the Docker remote API using these commands:

```
sudo vim /lib/systemd/system/docker.service
# Replace the line beginning with ExecStart for: ExecStart=/usr/bin/dockerd -H fd:// -H
    tcp://0.0.0.0:2375
sudo systemctl daemon-reload
# After you reload the daemon, you will need to restart and docker services that were
    running
sudo systemctl restart docker.service
# If everything worked, this command should return a json file
curl http://localhost:2375/images/json
```

8.1 Gateway Setup

This is the node-red image that connects the sensors to the Kafka broker. You need to change the IP address, the port and the topic to point to the correct Kafka broker in the following file before you build. To build the image, use this command on the VM for the gateways:

```
docker build --no-cache=true -f latest/Dockerfile
    https://github.com/SimonNadeau/node-red-docker.git -t brianr82/multinodered:latest
```

8.2 Virtual Sensor Setup

This is the image that allows to generate light and temperature data. To build the image, use this command on the VM for the sensors:

```
docker build --no-cache=true -f Dockerfile https://github.com/SimonNadeau/sensorsim.git -t
    brianr82/sensorsim:latest
```

8.3 The Experiment Tool

You are now almost ready to run some experiments. Clone this repository on your local machine: https://github.com/SimonNadeau/Sensor-Manager.git The Experiment Tool generates load for the application by creating new sensors and gateways (IoTDevice.py, IoTVirtualGateway.py) following a predetermined pattern that you have to specify in an IoTExperiment object such as IoTExperimentLinear. This way you can execute different workloads and run different experiments. The Experiment Tool also allows you to monitor the Docker containers in the VMs where you enable the Docker remote API. In our case, we use it to monitor containers running the Spark job.

8.3.1 Install PyCharm and Setup the Project

Go to this link to download and install PyCharm: https://www.jetbrains.com/pycharm/download/section=windows Go to this link to download and install Python 3.6.6: https://www.python.org/downloads/release/python-366/ Open PyCharm and select File → Open, then select the Sensor-Manager repository you cloned on your local machine and click OK.

Select File → Settings, then Project: Sensor-Manager → Project Interpreter. Click on the Settings symbol next to the Project Interpreter selection bar and select Add:

In the window that opens, create a new Virtual Environment and browse to the location of the python.exe file for the Base interpreter, then click OK:

In the terminal of PyCharm, enter these commands:

```
pip install docker
pip install pandas
```

You are now ready to run the project, but the tool has to be configured so it can connect to the VMs and access to the Docker remote API.

8.3.2 Create a new experiment and configure network and monitors

To create a new experiment, you only have to create a new class that inherits from IoTExperiment (take the example of IoTExperimentLinear.py). The first thing to do is to enter the correct IPs and ports in the configureNetwork function. It is there that you have to enter the information of the VMs where you want to create containers (sensors, gateways) or where you want to do some monitoring and scaling (Spark). The second thing to do is to create the right monitors for the appropriate VMs

you want to monitor. For that, use the configureMonitor function. There should not be a lot of changes to do in this function.

8.3.3 Configure the Workload

In the experiment, you can configure different type of workload using the configureExperiment function where it is possible to specify how much of each sensor you want. In the executeWorkload function, you can configure how the number of sensors vary in time.

8.3.4 Extra Configuration Aspects

The light and temperature sensors both use the same image on the sensor VM where they are created. To change the data creation rate of those sensors or their id, you have to change the producer_device_delay and the IoTDeviceName in the addIoTVirtualDevice function of their specific IoTDeviceService class (IoTDeviceServiceLight, IoTDeviceServiceTemperature). These configuration aspects could be used to simulates security attacks.

8.4 Scaling the Application Using the Experiment Tool

Eventually, the experiment tool and his monitoring capability could be used to scale the application following the MAPE-K process. We started something in the IoTScalingManager class, but it uses docker-cloud since docker-compose does not give access to an API that allows us to do scaling. Since we do not deploy the application on docker-cloud, it is not sure that this class is the right way to do scaling.

9 Bring it All Together

Now you have the tools to start your own experiments. Good luck!