[DevOps](https://www.c-sharpcorner.com/technologies/deployment)

Understanding Docker Compose

Introduction

Docker Compose is a tool for running multi-container applications, I found it an extremely convenient mechanism for deploying all the distributed components of an application on the local environment.

Imagine you are working on a Microservices development project, where you have various microservices and components in the architecture, say Microservices are designed using a Choreography pattern where one service (service A) communicates to another by publishing an Event to the Kafka Topic and service B consumes this event for further processing and in the further processing service B may invoke another REST service or save the event info to the Database or publish a message to another Kafka Topic or Queue. For Kafka to work Zookeeper is also required, for other REST services to work on our local some other component may be required.

It’s going to be extremely difficult for us to set up all these components manually some may work some may not, that’s where Docker Compose Scales and simplify these steps for us significantly, all we need is one .yml file where we will specify all our services, their dependencies and with one docker-compose up command, the components will be deployed, and the multi-container apps are ready for us to use.

Setup

In general, setting up Docker Compose Is a two-step process.

**Step One**

In step one, we will set up the Docker Engine using the command:  "sudo apt-get install docker.io"

*Verify docker setup: "sudo docker ps"*

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

**Step Two**

Step two is setting up the docker-compose tool using command: "sudo apt install docker-compose"

That’s it, Docker Compose setup is ready for us to use.

**Example**

For understanding the fundamentals, I am using one of the examples from the link <https://docs.docker.com/compose/samples-for-compose/>, the repository contains many docker-compose samples for us to use, it’s an amazing repository

I am using one of the samples with multiple integrated services

* [Elasticsearch / Logstash / Kibana](https://github.com/docker/awesome-compose/tree/master/elasticsearch-logstash-kibana) - Sample Elasticsearch, Logstash, and Kibana stack

This sample has three services, the ELK stack is one of my favorites for microservices debugging purposes. The services required are,

1. Elasticsearch
2. Logstash
3. Kibana

Let’s understand briefly what they are,

1. Elasticsearch is a search and analytics engine.
2. Logstash is server-side data processing pipeline that ingests data.
3. Kibana is for user debugging, helps us with error/data Visualization with nice graphs and charts.

Simply use the same docker-compose.yml file from the repository.

Commands

*docker-compose up*

All the services mentioned in the .yml file will be available at once.

*docker-compose up -d*

For running the command in the background ‘-d’ flag should be used, the services will be kept running in the background, -d stands for Detached Mode.

*docker-compose ps*

The ps command lists the containers and provides details like Ports, State, whether services are healthy or not, whether they are Up or Exited.

Name  Command                          State                        Ports  
-----------------------------------------------------------------------------------------------------  
es   /bin/tini -- /usr/local/bi ...        Up (healthy)     0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp  
kib   /bin/tini -- /usr/local/bi ...       Up                    0.0.0.0:5601->5601/tcp  
log   /usr/local/bin/docker-entr ... Exit                   0

*docker-compose stop*

The stop command stops the running containers, and it can be started again using the ‘up’ command.

*docker-compose logs*

Once the containers are started in the detached mode using the (-d) flag, we can check the logs using the ‘logs’ command with docker-compose.

*docker-compose -d –scale kibana = 3*

scale command scales the specific component to whatever the number is provided, in this case, we are scaling it to 3.

Visualizing Compose

As I mentioned in the Introduction section, in the classic Microservice use-case, there can be many services, apps, components (Kafka, DB, Queue) required for the microservices to communicate. Sometimes, with these many components, if we are creating all of them in one compose file, we may create confusion for ourselves, in my opinion identifying issues in the .yml file is not going to be straightforward.

An open-source utility should be used to visualize the compose known as “docker-compose-viz”

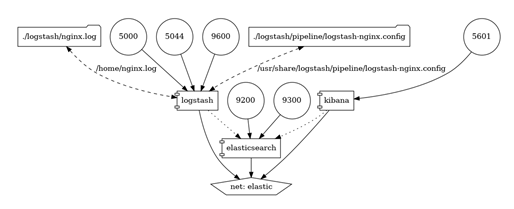
To set up the utility, execute: "sudo apt-get install graphviz"

to generate the .png use,

*docker run -u root --rm -it --name dcv -v $(pwd):/input pmsipilot/docker-compose-viz render -m image docker-compose.yml*

The command generates a .png file, which should be used for debugging or identifying what all components are taken care of by the specific compose file.

.png of our application is,



Summary

As mentioned, few times in the article, docker-compose simplifies the entire containerization process for our application, specially when we are working in distributed architecture. In the next article, we will explore in detail the docker-compose.yml file.