Hosting Student Applications on SNTC server via Docker Containerization

Group 5

Abhishek (B15103)

Sahil Singla (B15129)

Sai Tarun Reddy Palla (B15131)

Dhruv Patel (B15212)

IIT Mandi

CS307 - System Practicum Mini Project

Submitted to

Prof. Timothy A Gonsalves

Dr. Sriram Kailasam

May 27, 2018

Contents

Chapter 1

Introduction

The SNTC server intends to host many student projects so that the student community and the general public can have access to it. The projects will be heterogeneous in terms of the software components required. This can be realized by deploying shared docker containers in terms of the software and services required. The task is to have a load balancing mechanism between containers, so that sharing containers do not affect the performance of the application.

1.1 Need and Purpose

The SNTC wants to host several student projects, which are heterogeneous in terms of the software components requires. If one of the projects that share a container has a very high load, it will affect the others as well.

1.2 Intended Audience

The mini-project aims to provide a general documentation for the administrators of the SNTC server and members of IIT Mandi who may have to use the server for hosting their applications.

1.3 Scope of Project

The hosting architecture produced by the project aims to be dynamic in nature. This mini-project will establish a proper directive for hosting an application on the server. We aim to produce a directive which is long lasting and durable even if the technologies changes. The project allows a system for application hosting using containerization. The application can use any technology that can be supported by the Linux OS, and Docker containers. This will have specific directives for setting up containers, linking each containers with other containers, load balancing between instances of the containers, and easy access for a general user. In this project, we will be able to show load-balancing and container sharing using two application projects. We will be restricting ourselves to the constraints mentioned as follows -

- Each technology that an application developer uses can have one and only one version.
- Each docker image have only one instance of service running.
- Two applications cannot have the same application name.

Chapter 2

System Design

The overall blueprint of the server is shown in the figure 2.1

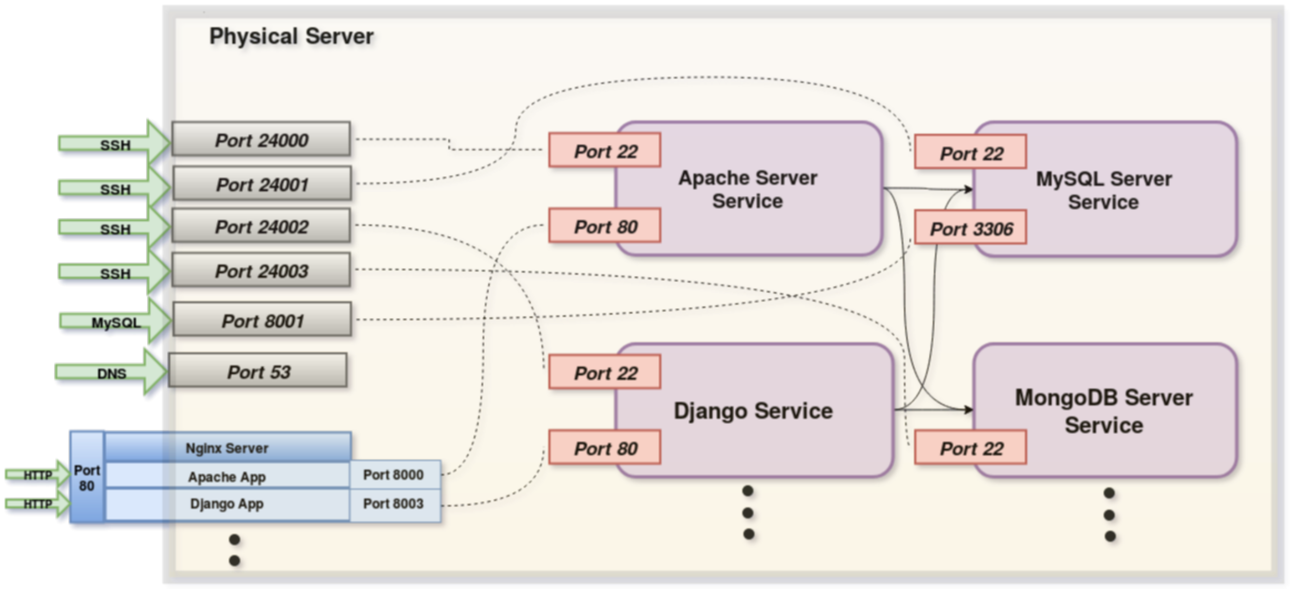

Figure 2.1: SNTC server design for hosting student application

The figure 2.1 shows four of the many services that can be hosted on the SNTC server - PHP-Apache, MySQL, Django, MongoDB. The services are hosted on a Docker Stack `webhosting` which is itself hosted on a swarm cluster containing only one node, i.e., the SNTC server. This type of design was implemented keeping in mind the objective that restarting containers will be minimal as possible.

2.1 General Service Design Specifications

- Each service is hosted on `webhosting` stack, resulting in each service's name as `webhosting_<service_name>`.
- Each service name will be corresponding to the software they host. Eg., Django service's name will be `webhosting_django`.

- Each service will have their port 22 mapped to a physical server's port which will be 24000 and up. This port will be used for SSH access to the container. This also means that each service will have an OpenSSH server installed on their containers.
- Each service can have one additional port on their containers exposed. This port will be used for accessing the main daemon of the software hosted by the service. This port will be tried to map to one of the physical server's 8000-9000 ports. If anyone of these ports are already in use, we can extend the range.
- All the service specification will be written in the main `docker-compose.yml` configuration file for the webhosting stack.
- Each service will have at least two volumes mapped to the physical server. One will contain the data of the applications that will use the software hosted by the service.
- Each container of the service will contain users corresponding to the applications using the software hosted by the service.
- Every service will be connected to the main overlay network `webhosting_main`. This will allow inter-networking between the services.

2.2 Specific Service Design Specification

2.2.1 Web Server services

- Each application will be hosted as a virtual host on the container.

2.2.2 PHP-Apache like webhosting services

These services will have their base-image derived from `phusion/baseimage`. This image allows running multiple daemons on the containers. Since such service might require some additional daemons, this image serves great purpose for these kinds of service. See [1] for more details. An additional image uploaded to the `sntcserver` Docker Cloud ID having `phusion/baseimage` as it's base image will server the base image for the service.

2.2.3 Services hosting Django, Nodejs, Ruby and similar softwares

These software have a different way of hosting web servers than Nginx or Apache. Here, each web server is bind to a port which serves the application requests. And each web server needs different ports and additional daemons running the server. To solve this problem, we utilise `phusion/passenger` images. Passenger is a deployment software which uses a middleman (`Nginx`) to host multiple web servers on different ports. See [2] and [3] for more details. An additional image uploaded to the `sntcserver` Docker Cloud ID having `phusion/passenger` as it's base image will server the base image for the service.

2.2.4 Database services

- Each database will have it's storage directory mounted on the physical server.

- These services will not have application-based Linux users, but will have specific user-like abstraction within the database. For eg., MySQL will have different users for each application with reduced privileges, and MongoDB will have different password-protected databases for each application using the service.
- SSH access to these containers will only be restricted to `root` user.
- The base images for these database will be from the Docker public image library. If you want ot add additional daemons, use `phusion/baseimage`.

2.2.5 Phusion Passenger Working

Phusion Passenger is an open source web application server. It handles HTTP requests, manages processes and resources, and enables administration, monitoring and problem diagnosis[3]. Passenger works in three integration modes - Apache, Nginx, and Standalone. We have used the Nginx method. Each app is hosted as a virtual host on Nginx server. Passenger daemon coordinates requests between the Nginx server and the application. The design is demonstrated in figure 2.2.

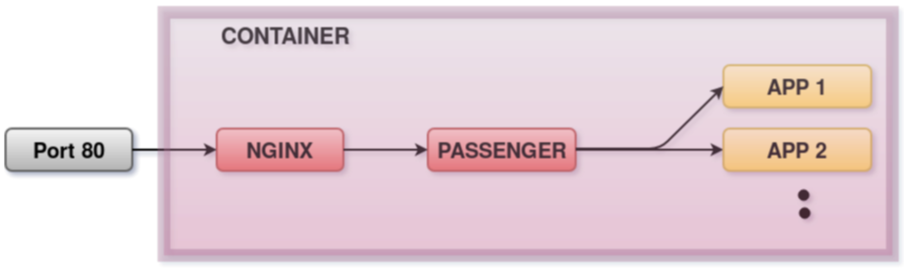

Figure 2.2: Design of a container with Phusion Passenger

Chapter 3

Configuration Details

This chapter deals with the current configuration on the server and lists all the configuration files used for deployment. The compose file is listed in listing

3.1 Docker Stack and Swarm

The Docker stack is deployed using a compose file listed in listing 3.1. Here, we use the YAML version 3.4 for writing the Dockerfile.

Listing 3.1: "Docker compose file for stack confiduration"

```
version: '3.4'
services:
  phpapache:
    image: sntcserver/phpapache:v1
    ports:
      - "8000:80"
      - "24000:22"
    volumes:
      - ./php-apache2/data:/home
      - ./php-apache2/apache2-conf:/etc/apache2
    networks:
      - main
  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: mysqlsntc
    logging:
      driver: "json-file"
    ports:
      - "24002:3306"
      - "24001:22"
    volumes:
      - ./mysql/data-dir:/var/lib/mysql
      - ./mysql/mysql-conf.d:/etc/mysql/conf.d
      - ./mysql/mysqld-conf.d:/etc/mysql/mysql.conf.d
#       - ./mysql/logs:/var/log/mysql
    networks:
      - main
  adminer:
    image: adminer
    ports:
      - "8002:8080"
    networks:
      - main
  django:
    image: sntcserver/django2:v1
    ports:
      - "8003:80"
      - "24003:22"
    volumes:
      - ./django-2/data:/home
      - ./django-2/nginx-conf:/etc/nginx
    networks:
      - main

volumes:
  data-phpapache:
    external: true
    name: /home/stweb/SysPrac2018-19/containers/php-apache2/data
  conf-phpapache:
    external: true
    name: /home/stweb/SysPrac2018-19/containers/php-apache2/apache2-conf
  data-mysql:
    external: true
    name: /home/stweb/SysPrac2018-19/containers/mysql/data-dir
  conf-mysql:
    external: true
    name: /home/stweb/SysPrac2018-19/containers/mysql/mysql-conf.d
  data-django:
  conf-django:

networks:
  main:
    driver: overlay
```

3.2 Dockerfiles

3.2.1 PHP-Apache Image

Listing 3.2: "Dockerfile for PHP-Apache image"

```
FROM phusion/baseimage

# OpenSSH-server
RUN cp /etc/ssh/sshd_config /etc/ssh/sshd_config.original_copy
RUN rm -f /etc/service/sshd/down
RUN echo "root:phpapache2" | chpasswd
RUN /etc/my_init.d/00_regen_ssh_host_keys.sh
RUN echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
RUN /etc/init.d/ssh restart

# Apache
RUN apt-get update
RUN apt-get -y install apache2

# PHP
RUN apt-get -y install php libapache2-mod-php php-mcrypt php-mysql

# Starting apache2
RUN mkdir /etc/service/apache
ADD apache.sh /etc/service/apache/run
RUN chmod +x /etc/service/apache/run
```

3.2.2 Django Image

Listing 3.3: "Dockerfile for Django image"

```
FROM phusion/passenger-customizable

CMD ["/sbin/my_init"]

# OpenSSH-server
RUN cp /etc/ssh/sshd_config /etc/ssh/sshd_config.original_copy
RUN rm -f /etc/service/sshd/down
RUN echo "root:phpapache2" | chpasswd
RUN /etc/my_init.d/00_regen_ssh_host_keys.sh
RUN echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
RUN /etc/init.d/ssh restart

# Nginx
```

```
RUN rm -f /etc/service/nginx/down

# Python and Django
RUN apt-get update
RUN apt-get install -y python3 python3-pip
RUN pip3 install django
```

3.3 Load Balancing Script

The load balancing script is in the listing 3.4. The basic principle on which the script is based on monitoring the CPU usage and memory usage percentages. The monitoring is used through a Docker CLI command - `docker stats`. This monitoring is done at predefined intervals implemented by the `sleep` function. After each sleep interval, we parse the docker statistics for getting the containers from our stack. We analyze if the average usage percentage of the service is greater or less than it's threshold. Depending on the comparison, we scale the service.

Listing 3.4: "Load Balancing Script"

```
import subprocess
import pandas as pd
import re
import time

import subprocess

command = "docker stats --no-stream > test1.txt"
upper_cpu_threshold = 90.0
upper_mem_threshold = 40.0
lower_cpu_threshold = 50.0
lower_mem_threshold = 20.0

def file_create():
        result = subprocess.run(command, shell=True)
        with open("test1.txt") as f:
            content = f.readlines()

        content = [x.strip('  ') for x in content]
        content1 = [re.sub('  +',',',x) for x in content]

        fo = open("csv_file.csv","w")
        for line in content1 :
                fo.write(line)
        fo.close()

def stats_query():
```

```
data_frame=pd.read_csv('csv_file.csv', sep=',')
a=dict()

for index, row in data_frame.iterrows():
        # print("csd\n")
        idx=row["NAME"].find('.')

        if idx>0:
                service_name = row["NAME"][:idx]
        else :
                service_name = row["NAME"]

        if "webhosting" not in service_name :
                continue

        if service_name in a.keys() :
                # print("csd\n")
                cpu = row["CPU %"]
                cpu = cpu.replace("%",'')
                a[service_name]["CPU %"] = ((a[service_name]["CPU
 %"]*a[service_name]["CONTAINER_NO."]) + float(cpu))/(a[service_name][
"CONTAINER_NO."]+1)

                mem = row["MEM %"]
                mem = mem.replace("%",'')
                a[service_name]["MEM %"] = ((a[service_name]["MEM
 %"]*a[service_name]["CONTAINER_NO."]) + float(mem))/(a[service_name][
"CONTAINER_NO."]+1)

                a[service_name]["CONTAINER_NO."]+=1

        else :
                a[service_name] = dict()
                # print("csd\n")

                cpu = row["CPU %"]
                cpu = cpu.replace("%",'')
                # print(cpu)
                a[service_name]["CPU %"] = float(cpu)

                mem = row["MEM %"]
                mem = mem.replace("%",'')
```

```
                        # print(mem)
                        a[service_name]["MEM %"] = float(mem)

                        a[service_name]["CONTAINER_NO."] = 1

        #print (a)
        return a

while True:
        file_create()
        a=stats_query()
        for service_name in a.keys() :
                inc_scaling_comand = "docker service scale " +
   service_name + "=" + str(a[service_name]["CONTAINER_NO."]+1)
                dec_scaling_comand = "docker service scale " +
   service_name + "=" + str(a[service_name]["CONTAINER_NO."]-1)
                if (a[service_name]["CPU %"] > upper_cpu_threshold or a[
   service_name]["MEM %"] > upper_mem_threshold) and a[service_name]["
   CONTAINER_NO."] <= 5  :
                        print("upscaling")
                        result = subprocess.run(inc_scaling_comand, shell
   =True)
                elif (a[service_name]["CPU %"] < lower_cpu_threshold or a
   [service_name]["MEM %"] < lower_mem_threshold) and a[service_name]["
   CONTAINER_NO."] > 1  :
                        print("downscaling")
                        result = subprocess.run(dec_scaling_comand, shell
   =True)
        print("sleeping")
        time.sleep(3)
```

Chapter 4

Deployment Example

In this chapter, we will deploy a Flask application which uses SQLite database. Since Flask is based on Python similar to Django, we will be building a base image using `phusion/passenger`.

4.1 Step 1: Build the base image for the service

Since, there is no existing service which hosts Flask, we will have to create a new service. The first step for creating the service is to start with a base image. Since this service will be based on `phusion/passenger`, we will have to manually install the Python dependencies and enable SSH in the Dockerfile. The Dockerfile for this new service is listed in listing 4.1.

Listing 4.1: "Dockerfile for Flask image"

```
FROM phusion/passenger-customizable

CMD ["/sbin/my_init"]

# OpenSSH-server
RUN cp /etc/ssh/sshd_config /etc/ssh/sshd_config.original_copy
RUN rm -f /etc/service/sshd/down
RUN echo "root:phpapache2" | chpasswd
RUN /etc/my_init.d/00_regen_ssh_host_keys.sh
RUN echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
RUN /etc/init.d/ssh restart

# Nginx
RUN rm -f /etc/service/nginx/down

# Python and Django
RUN apt-get update
RUN apt-get install -y python3 python3-pip
RUN pip3 install flask
```

1. Make a directory `flask` in the main `SysPrac2018-19/containers` directory on the server.

   ```
   > mkdir flask
   > cd flask
   ```

2. Save the Dockerfile listed in listing 4.1 as `Dockerfile`

3. Run the following commands to build and push the image to the `sntcserver` Docker ID.

   ```
   > docker build -t sntcserver/flask:v1 .
   > docker login
   > docker push sntcserver/flask:v1
   ```

4.2 Step 2: Make changes in the Stack configuration file

1. Append the following listing **??** in the main `docker-compose.yml` file.

   ```
   flask:
     image: sntcserver/flask:v1
     ports:
       - "8004:80"
       - "24004:22"
     volumes:
       - ./flask/data:/code
       - ./flask/nginx-conf:/etc/nginx
     networks:
       - main
   ```

   This specifies the ports that are to be exposed on the `flask` service. It also specifies the volumes to be mapped on the `flask` service and the base image to be taken to create the service. Lastly, it specifies the network to be attached to.

2. Update the stack by the following command.

   ```
   > docker stack deploy -c docker-compose.yml webhosting
   ```

4.3 Step 3: Check the service

Use the command `docker ps` to check whether a container of the service is running or not.

4.4 Step 4: Change the Nginx configuration file within the container

To do this, you can either make changes in the mapped volume drive OR execute the bash command on the container. Here, we will do it by executing the bash command on the container.

1. Find the container name using `docker ps`.

2. Execute the following command to open the bash terminal in the container.

   ```
   > docker exec -it <container_name> bash
   ```

3. Create a new file `/etc/nginx/sites-enabled/<app_name>.conf` and add the following lines in the file.

   ```
   server {
           listen 80;
           server_name <app_name>.iitmandi.ac.in;

           root <path to static files' folder>;

           passenger_enabled on;
           passenger_python /usr/bin/python3;
   }
   ```

4. Reload nginx by running the command `nginx -s reload`.

4.5 Step 5: Add DNS entries on the SNTC server

For this step, you have to add the DNS entry for `<app_name>.iitmandi.ac.in` in the DNS server. For this follow the updating configuration guide at [4].

4.6 Step 6: Change the Nginx configuration on SNTC server

Add the following server block in the `/etc/nginx/nginx.conf` -

```
server {
                listen 80;
                server_name <app\_name>.iitmandi.ac.in;

                location / {
                        proxy_pass http://127.0.0.1:8004;
                        proxy_set_header Host $host;
                        proxy_set_header X-Real-IP $remote_addr;
                        proxy_set_header X-Forwarded-For
                           $proxy_add_x_forwarded_for;
                        proxy_set_header X-Forwarded-Proto $scheme;
                }
        }
```

After adding the lines, reload the Nginx server. And your app is all set to go!

Test your app by `ping <app_name>.iitmandi.ac.in`.

References

[1] Phusion. Phusion baseimage-docker git repository and documentation. https://github.com/phusion/baseimage-docker.

[2] Phusion. Phusion passenger-docker git repository and documentation. https://github.com/phusion/passenger-docker.

[3] Phusion. Phusion passenger documentation. https://www.phusionpassenger.com/library/.

[4] Justin Ellingwood. How to configure bind as a private network dns server on ubuntu 16.04. https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-16-04.

[5] System Practicum 2018-19 Group 5. Software requirements specification document.

[6] System Practicum 2018-19 Group 5. Software design document.