Technique For Load Balancing with Kubernetes and Python

A Project report submitted in partial fulfilment
of the requirements for the degree of B. Tech in Electrical Engineering

By

Name of the Students (Roll No.)
Abhro Roy (11701619034)

Deepanjan Mondal (11701619025)

Anish Chakraborty (11701619015)

Under the supervision of

Shilpi Bhattacharya, Associate Professor

Department of Electrical Engineering

Department of Electrical Engineering
RCC INSTITUTE OF INFORMATION TECHNOLOGY
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA – 700015, WEST BENGAL
Maulana Abul Kalam Azad University of Technology (MAKAUT)

ACKNOWLEDGEMENT

It is my great fortune that I have got opportunity to carry out this project work under the supervision of **Prof. (Dr.) Shilpi Bhattacharya** in the Department of Electrical Engineering, RCC Institute of Information Technology (RCCIIT), Canal South Road, Beliaghata, Kolkata-700015, affiliated to Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India. I express my sincere thanks and deepest sense of gratitude to my guide for his constant support, unparalleled guidance and limitless encouragement.

I wish to convey my gratitude to Prof. (Dr.) Shilpi Bhattacharya, HOD, Department of Electrical Engineering, RCCIIT and to the authority of RCCIIT for providing all kinds of infrastructural facility towards the research work.

I would also like to convey my gratitude to all the faculty members and staffs of the Department of Electrical Engineering, RCCIIT for their whole hearted cooperation to make this work turn into reality.

Signature of the Students

Place:

Date:

Department of Electrical Engineering
RCC INSTITUTE OF INFORMATION TECHNOLOGY
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA – 700015, WEST BENGAL

CERTIFICATE
To whom it may concern

This is to certify that the project work entitled **Technique For Load Balancing with Kubernetes and Python** is the bona fide work carried out by **Abhro Roy (11701619034), Deepanjan Mondal (11701619025)** , **Anish Chakraborty (11701619015)** students of B.Tech in the Dept. of Electrical Engineering, RCC Institute of Information Technology (RCCIIT), Canal South Road, Beliaghata, Kolkata-700015, affiliated to Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India, during the academic year 2021-22, in partial fulfillment of the requirements for the degree of Bachelor of Technology in Electrical Engineering and this project has not submitted previously for the award of any other degree, diploma and fellowship.

_____ _____

Signature of the Guide
Name:
Designation:

Signature of the HOD, EE
Name:
Designation:

_____

Signature of the External Examiner
Name:
Designation:

TABLE OF CONTENTS

Technique For Load Balancing with Kubernetes and Python

Abhro Roy, Deepanjan Mondal, Anish Chakraborty, Shilpi Bhattacharya
Electrical Engineering
RCC Institute of Information Technology, Kolkata
e-Mail: abirabhroroy@gmail.com

ABSTRACT

Often we find servers across organizations and institutes having servers which have minimal to no load on normal days but peak during certain times of the year and are slow and overload rendering the site to little use. Kubernetes enables versatile deployment of applications which are auto-scaled, auto-maintained, and auto-managed depending on variety of parameters including peak load.

Keywords: **Kubernetes, Python, Docker, Linux**

1. INTRODUCTION

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Kubernetes is an open-source platform renowned for its versatility in managing containerized workloads and services. It offers portability, enabling seamless deployment and management across various cloud environments, including public, private, and hybrid setups. With its extensible architecture, Kubernetes allows users to incorporate additional functionalities through a wide range of plugins and extensions available in its thriving ecosystem.

At the heart of Kubernetes lies the concept of declarative configuration, empowering users to define the desired state of their applications and infrastructure. Through automated orchestration, Kubernetes handles the complexities of deploying, scaling, and updating applications, freeing developers and operators to focus on higher-level tasks.

The name "Kubernetes" derives its significance from Greek, meaning "helmsman" or "pilot." This metaphor aptly represents Kubernetes' role as a guiding force, steering applications and services towards smooth operation and optimal performance. The abbreviation "K8s" simplifies pronunciation while emphasizing the eight letters between "K" and "s."

In 2014, Google's decision to open-source Kubernetes marked a pivotal moment in container orchestration. Leveraging Google's extensive experience in managing production workloads at scale, Kubernetes incorporates industry best practices and community-driven innovations. Today, it has become the industry standard for container orchestration due to its robust architecture and feature set.

Kubernetes owes its success and popularity to its vibrant ecosystem. An active community of developers, operators, and vendors contribute to its development, offering a wealth of tools, frameworks, and services that complement and enhance Kubernetes' capabilities. This extensive ecosystem ensures comprehensive support, documentation, and resources, simplifying adoption and utilization of Kubernetes for various use cases.

In conclusion, Kubernetes is a dynamic and evolving platform that empowers organizations to efficiently manage and scale their containerized workloads and services. Its portability, extensibility, and declarative nature, along with its thriving ecosystem, make it the preferred choice for modern application deployment and orchestration.[1]

2. LITERATURE SURVEY

Deploying a Kubernetes cluster is cost-effective as it leverages open-source code freely available to everyone. The software itself is open source and does not require any licensing fees. However, it is important to note that while the software is free, there are associated costs for hardware and infrastructure.

To host a Kubernetes architecture, a relatively modest bare-metal system is typically sufficient. It does not necessarily require high-end or expensive hardware. A moderately capable system can effectively run a Kubernetes cluster without significant issues. The hardware requirements may vary depending on the scale and complexity of the workload and the number of nodes in the cluster.

It is worth mentioning that although the software cost is minimal, there may still be expenses related to networking, storage, and other infrastructure components required to support the Kubernetes cluster. Additionally, there might be costs associated with maintenance, monitoring, and ongoing operational support.

Overall, while the software itself is open source and freely available, the cost of deploying a Kubernetes cluster includes considerations for hardware and infrastructure, as well as potential ongoing operational expenses.[2]

3. TECHNOLOGIES AND METHODOLOGIES USED IN THIS PROJECT

The resources used to deploy this entire architecture are less in number, but implemented in depth. They are as follows:

1. KUBERNETES

The actual platform which hosts our code, application and all the bells and whistles required to seamlessly expose the application as a service outside Kubernetes.

2. PYTHON

The application is developed using a specific programming language, and it is complemented by a Python-Flask server to facilitate hosting the application within a containerized environment. The choice of programming language depends on various factors, including the requirements, preferences, and expertise of the development team. Python-Flask, a popular web framework, is utilized as the server to provide a lightweight and flexible platform for running the application. By combining the chosen programming language with the Python-Flask server, the application can be effectively encapsulated and deployed within a container, ensuring portability, scalability, and ease of management.

3. DOCKER

Docker is a containerization platform used to deploy containers, which are nothing but a shell of a base image of an OS, on which one can install all the resources one installs on a normal bare-metal OS. Containers are surprisingly light and fast and are used during mass deployment of similar applications for milking out the most of the resources available at hand.

4. TECHNIQUES USED FOR REAL-LIFE IMPLEMENTATION AND WORKING

1. Writing the python script

Write a python code with Flask web server to print a basic html page with ip of the host it is running on and push the image to DockerHub.

Below is the python code implemented. A rather simple code stating our application be broadcast on all IPs (ref. Last line ‘host=0.0.0.0’). On going to our default route which is ‘/’ in ‘webgen.americaniche.com/’, the below code is executed. The code gets the host IP the application is running on and displays it has HTML content on the webpage. [3]-[4]

root@JEET: /

```
from flask import Flask
import subprocess
import socket

app = Flask(__name__)


@app.route('/')
def main():
    # cmd = "ifconfig | grep -E \"([0-9]{1,3}\.){3}[0-9]{1,3}\" | grep -v 127.0.0.1 | awk '{ print $2 }' | cut -f2 -d:"
    # result = subprocess.run(cmd.split(), stdout=subprocess.PIPE)
    print(socket.gethostbyname(socket.gethostname()))
    return '''
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<h5>THIS IP '''+socket.gethostbyname(socket.gethostname())+'''</h5>
<p>For online documentation and support please refer to
<a href=http://nginx.org/>nginx.org</a>.<br/>
Commercial support is available at
<a href=http://nginx.com/>nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
'''


if __name__ == '__main__':
    app.run(host='0.0.0.0',debug=True)
~
~
~
```

2. Creating the custom image with Dockerfile

We first build a custom Docker image with base OS ‘ubuntu:latest’ and we setup the image to do the following:
Expose port 5000 outside container where our server will be running.
Copy our Python code from host os to image os.
Install python and its dependencies.
Create entry-point script to run our python code whenever the image is deployed.[5]

```
from ubuntu:latest

EXPOSE 5000

COPY /app.py /

RUN apt update

RUN apt install -y python3 && \
    apt install -y python3-pip && \
    pip3 install Flask && \
    apt install -y net-tools && \
    apt install -y vim && \
    apt install -y cron

#RUN echo "until python3 /app.py; do  echo "Server 'flask' crashed with exit code $?.  Respawning.." >&2;  sleep 1; done" >> /keepalive.sh
#RUN chmod 644 /keepalive.sh

#RUN touch /var/log/cron.log

#RUN echo '3 * * * * kill -9 $(ps ax|grep "app.py"|head -1|awk "{print $1}"' >> /etc/cron.d/runka
#RUN echo '* * * * * python3 /app.py '>> /etc/cron.d/runka
#RUN chmod 644 /etc/cron.d/runka

# Apply cron job
#RUN crontab /etc/cron.d/runka
ENTRYPOINT ["python3", "/app.py"]
~
~
```

3. Setting up the Kubernetes cluster

Using Vagrant and Ansible we set up a 5 node Kubernetes cluster of which one is the master node (responsible for managing worker nodes and for deploying apps and services) and four are the worker/slave nodes.
The Vagrantfile is responsible for spinning up the required number of virtual machines and naming them, provisioning resources, disk space, assign IP etc. , and the ansible playbook, which is embedded within the Vagrantfile, and is responsible for installing required applications, dependencies, features etc. The first image below is the Vagrantfile and the second image is the trimmed ansible playbook for the VM that will act as our master node in our Kubernetes cluster.[6]

```
Vagrant.configure("2") do |config|
    config.ssh.insert_key = false

    config.vm.provider "virtualbox" do |v|
        v.memory = 4096
        v.cpus = 2
    end

    config.vm.define "k8s-master" do |master|
        master.vm.box = IMAGE_NAME
        master.vm.network "private_network", ip: "192.168.56.10"
        master.vm.hostname = "k8s-master"
        master.vm.provision "ansible" do |ansible|
            ansible.playbook = "master-playbook.yaml"
            ansible.extra_vars = {
                node_ip: "192.168.56.10",
            }
        end
    end

    (1..N).each do |i|
        config.vm.define "node-#{i}" do |node|
            node.vm.box = IMAGE_NAME
            node.vm.network "private_network", ip: "192.168.56.#{i + 10}"
            node.vm.hostname = "node-#{i}"
            node.vm.provision "ansible" do |ansible|
                ansible.playbook = "node-playbook.yaml"
                ansible.extra_vars = {
                    node_ip: "192.168.56.#{i + 10}",
                }
            end
```

```
[abhro@backendserver vaganskube]$ cat master-playbook.yaml
---
- hosts: all
  become: true
  tasks:
  - name: Install packages that allow apt to be used over HTTPS
    apt:
      name: "{{ packages }}"
      state: present
      update_cache: yes
    vars:
      packages:
      - apt-transport-https
      - ca-certificates
      - curl
      - gnupg-agent
      - software-properties-common

  - name: Add an apt signing key for Docker
    apt_key:
      url: https://download.docker.com/linux/ubuntu/gpg
      state: present

  - name: Add apt repository for stable version
    apt_repository:
      repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu xenial stable
      state: present

  - name: Install docker and its dependecies
    apt:
      name: "{{ packages }}"
      state: present
      update_cache: yes
    vars:
      packages:
      - docker-ce
      - docker-ce-cli
      - containerd.io
    notify:
      - docker status

  - name: Add vagrant user to docker group
    user:
      name: vagrant
      group: docker

  - name: Remove swapfile from /etc/fstab
    mount:
      name: "{{ item }}"
```

4. Assigning Node and Pods

To deploy our application on Kubernetes, we begin by pulling our custom Docker image from DockerHub. This image contains all the necessary dependencies and configurations for our application.

Next, we set up a pod, which is a single instance of a Docker container, using the pulled image. Kubernetes then assigns an IP address to the pod and determines the node where the pod will run. We can access this IP address from within the Kubernetes cluster to verify the pod's availability.

Once the pod is successfully deployed, we create a NodePort type service. This service acts as a load balancer and is associated with our pod. It exposes the application outside of the Kubernetes cluster using a port that is opened on the node where the pod is hosted. This allows external traffic to reach our application.

In order to ensure high availability, we employ a DaemonSet. This configuration allows us to replicate our pod and deploy one copy on each node within the Kubernetes cluster. By doing so, we distribute our application across multiple nodes, making it highly available and resilient to node failures.

The combination of the pod, service, and DaemonSet enables us to efficiently deploy and manage our application on Kubernetes. It provides scalability, fault tolerance, and ease of access to our application both within and outside of the Kubernetes cluster.[7]-[9]

5. Exposing outside Kubernetes

To enable access to our Pod from anywhere in the world while ensuring security and convenience, we utilize an Apache server virtual host with a reverse proxy. This setup allows us to prevent the leaking of the underlying port information and provides a pre-configured AWS Route53 domain for easy access.

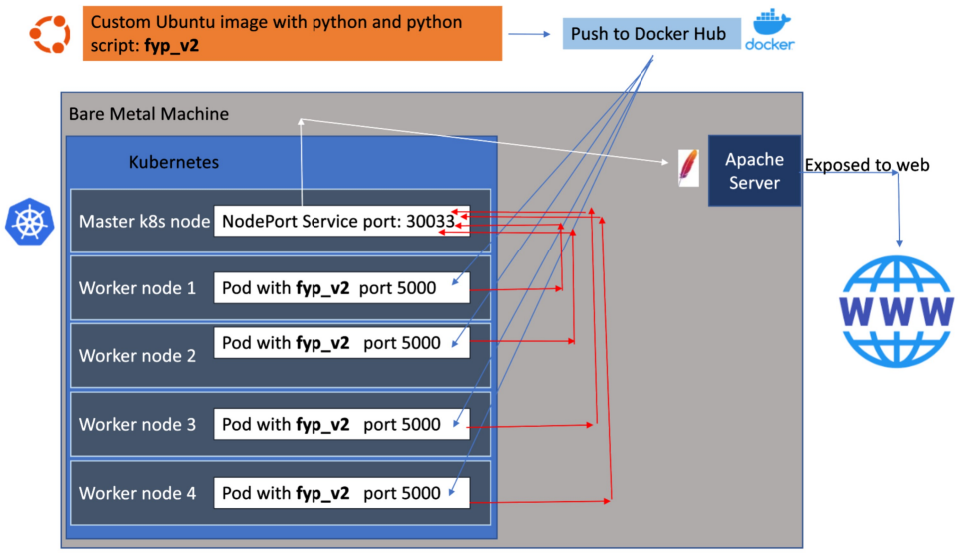
In this configuration, the <VirtualHost *:80> directive indicates that we are creating a virtual host to handle incoming requests on port 80. The ServerName parameter is set to our fully qualified domain name (FQDN), which in this case is webgen.americaniche.com.

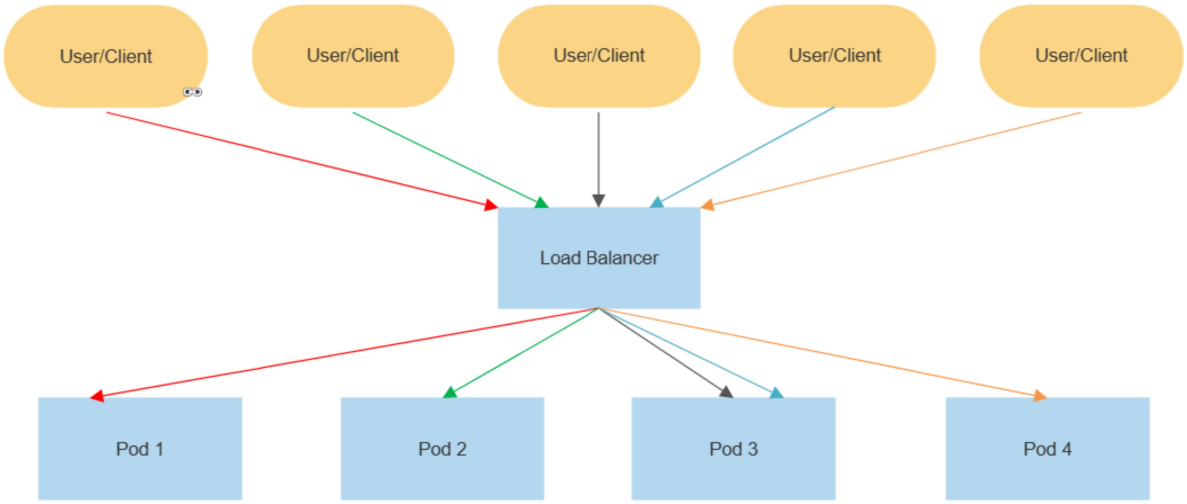
The most crucial part of this setup is the ProxyPass configuration. It acts as a reverse proxy, forwarding requests to the specified port where our application is running, which is represented by <port>. By doing so, the actual port where our application is running remains hidden, preventing exposure and minimizing the risk of compromise or attacks.

With this setup, when users access http://webgen.americaniche.com, the requests are redirected to the Apache server, which then proxies the traffic to the underlying application running on the designated port. This provides a seamless and secure way to access our application from anywhere in the world without revealing the underlying infrastructure details.[10]

```
<VirtualHost *:80>
    Servername webgen.americaniche.com
Proxypass / http://webgen.americaniche.com:30034/
ProxypassReverse / http://webgen.americaniche.com:30034/
</VirtualHost>
```

5. VISUAL REPRESENTATION





6. FUTURE WORK

Our scope to this project ends here and goes on to show the capabilities offered, however, it can be extended to include sticky sessions to bind to users to save login data or session data, other enhancements and whatnot. For this we would need to find a requirement to satisfy, and that supports this development.[6]-[9]

7. CONCLUSION

Thus with this implementation we prove the ease of usability, low cost and setup of a deployment platform that can host entire infrastructures with the proper resources anywhere in the world, for a fraction of the cost required by mainstream cloud platforms like AWS and Azure.

REFERENCES

[1] D. Ashley, “Using Flask and Jinja,” Foundation Dynamic Web Pages with Python, pp. 159–181, 2020, doi: 10.1007/978-1-4842-6339-6_5.

[2] D. Ashley, “Introduction to Web Servers,” Foundation Dynamic Web Pages with Python, pp. 1–27, 2020, doi: 10.1007/978-1-4842-6339-6_1.

[3] O. Yilmaz, “Introduction,” Extending Kubernetes, pp. 1–19, 2021, doi: 10.1007/978-1-4842-7095-0_1.

[4] M. Lukša, “Kubernetes erweitern,” Kubernetes in Action, pp. 553–578, Jul. 2018, doi: 10.3139/9783446456020.018

[5] Q. Li and B. Moon, “Distributed cooperative Apache web server,” Proceedings of the tenth international conference on World Wide Web

[6] M. Shahinpoor, Y. Bar-Cohen, T. Xue, J.O. Simpson and J. Smith, “Ionic Polymer-Metal Composites (IPMCs) as Biomimetic Sensors, Actuators and Artificial Muscles: A Review”, Proceedings of SPIE's 5th Annual International Symposium on Smart Structures and Materials, 1-5 March, 1998, San Diego, CA DOI: 10.1088/0964-1726/7/6/001

[7] Zheng Chen, Yantao Shen, Jason Malinak, Ning Xi, Xiaobo Tan,“Hybrid IPMC/PVDF Structure for Simultaneous Actuation and Sensing”, Proceedings of SPIE Vol. 6168, Smart Structures and Materials, pp. 61681L 1 – 61681L9, 2006. DOI: 10.1109/ICIEV.2014.6850840

[8] R.K. Jain, S. Datta and S. Majumder, “Design and Control of an EMG Driven IPMC Based Artificial Muscle Finger”

[9] S. Bhat, “Introduction to Containerization,” Practical Docker with Python, pp. 1–8, 2018, doi: 10.1007/978-1-4842-3784-7_1.

[10] K. Jangla, “Docker Compose,” Accelerating Development Velocity Using Docker, pp. 77–98, 2018, doi: 10.1007/978-1-4842-3936-0_6.