MINOR - I PROJECT
Synopsis On
Scaling of Web Application hosted on Containers with an automated build tool

Submitted By

Name	Roll No	Branch
Harsh Gupta	R110216070	CSE CCVT
Kunal Visoulia	R110216089	CSE CCVT
Vibhor Jain	R110216170	CSE CCVT

Under the Guidance of

Dr Monit Kapoor
Associate Professor & Head
Dept. of Cybernetics,School of Computer Science

School of Computer Science

University of Petroleum & Energy Studies, Dehradun

Bidholi Campus, Energy Acres, Dehradun-248007

School of Computer Science

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

Dehradun-248007 2018-19

Project Proposal Approval Form

Minor [I] **Major** [ ]

Project Title:

Scaling of Web Application hosted on Containers with an automated build tool.

Abstract

The world of cloud computing and enterprise software experiences an endless parade of new technologies, languages, and platforms.Containerization is a powerful way to package and deploy tools. The tools using containers gives us an environment that packages your code along with all the dependencies into an application container in such a way that it doesn't require a full fledged Virtual Machine to run. It has brought a revolutionary change in hosting cloud applications and microservices. The most incredible use case of cloud is its capability to provide on demand services. And containers has aided in enhancing that. The applications today are hosted on container rather than a virtual machine and those full fledged containers are shared to clients. But every revolution comes with a challenge. Here the challenge is to scale up and down the containers at run time. A container can also fail to run due to many reasons but that doesn't mean that a client's work should come at halt. A system is needed which can automatically detect a failure and run another hosting platform and resume the services. We have tried to build such a system.

Introduction

Today, the cloud industry is adopting the container technology both for internal usage and as commercial offering. The use of containers as base technology for large-scale systems opens many challenges in the area of management of these containers at run-time.Operating system and application virtualization, also known as container (e.g. Docker and LXC), became popular since 2013 with the launch of the Docker open source project (docker.com) and with the growing interest of cloud providers and Internet Service Providers (ISP).

Containers:

A container is a software environment where one can install an application or application component (the so called microservice) and all the library dependencies, the binaries, and a basic configuration needed to run the application. Containers provide a higher level of abstraction for the process life cycle management, with the ability not only to start/stop but also to upgrade and release a new version of a containerized application or of a microservice in a seamless way.Containers potentially may solve many distributed application challenges, e.g. portability and performance over-head. With containers, a microservice or an application can be executed on any platform running a container engines. Containers are lightweight and introduce lower overhead compared to Virtual Machines (VMs).That are some of the reasons that make the cloud computing industry to adopt container technologies and to contribute to the evolution of this technology.[1]

The use of containers as base technology for deploying large-scale applications opens many challenges in the area of their management at run-time. In production environment there exists a cluster of containers hosting an application and thus scaling of them as per the user demand is a challenge in industries.

Microservice:

A 'microservice' is a software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop, test, and more resilient to architecture erosion. It parallelizes development by enabling small autonomous teams to develop, deploy and scale their

respective services independently. Microservices-based architectures enable continuous delivery and deployment.[2]

make Utility:

The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to re compile them. you can use make with any programming language whose compiler can be run with a shell command. In fact, make is not limited to programs. You can use it to describe any task where some files must be updated automatically from others whenever the others change. [3]

To prepare to use make, you must write a file called the makefile that describes the relationships among files in your program, and the states the commands for updating each file. In a program, typically the executable file is updated from object files, which are in turn made by compiling source files.

Docker:

Docker is a tool that packages, provisions and runs containers independent of the OS. It is a open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools. Docker has emerged as a de facto standard platform that allows users to quickly compose, create, deploy, scale and oversee containers across Docker hosts. Docker allows a high degree of portability so that users can register and share containers over various hosts in private and public environments. Docker benefits include efficient application development, lower resource use and faster deployment compared to VMs. [4]

Container Orchestration:

Containers allow us to standardise the environment and abstract away the specifics of the underlying operating system and hardware. Container Orchestration can be thought as doing the same job for the data centre: it allows us the freedom not to think about what server will host a particular container or how that container will be started, monitored and killed.Container orchestration is all about managing the life cycles of containers, especially in large, dynamic environments. Software teams use container orchestration to control and automate many tasks like provisioning, scaling of containers etc.[5]

Motivating Example-Kubernetes:

Kubernetes is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications.It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation. It aims to provide a platform for automating deployment, scaling, and operations of application containers across clusters of hosts.It works with a range of container tools, including Docker.[6]

As you separate units of work into small batches, you need a way to manage those workloads within a management layer. This layer must let you share resources, schedule tasks, and treat many running processes as one unified, flood and fire a scalable, and well-behaved solution across all workloads. This is how transactional systems of years past have functioned.Compute clusters form a shared computing environment made up of servers (nodes), where resources have been clustered together to support the workloads and processes running within the cluster. As you combine processes within a cluster (called a task), they create the solution, which involves combining the tasks into a job.

Here, what we have is the cluster of containers which are working as a host to a web application. In an increased load of work this cluster needs to be scaled up or down as per the user demand and that requires a tool which can automatically detects the workload and do our management task.This can be achieved by reconfiguring the existing architecture of a single machine to increase available resources or by provisioning additional containers within a cluster of distributed machines.

Problem Statement

The common challenge that enterprise encounters is not being able to meet unpredictable, fluctuating computing demands on container hosted applications in an efficient manner. Businesses today need to be agile and need the ability to scale resources easily and quickly based on rapidly changing market needs. Because an enterprise's demand for computing resources can vary dramatically from one period of time to another, maintaining sufficient resources to meet peak requirements can be

costly. One way can be vertical scaling but that couldn't be cost friendly solution. Moreover, it would be more prone to hardware failures. However, horizontal scaling can cut costs by maintaining minimal computing resources until they run into the need to increase them meanwhile only paying for what they use.

Literature Review

A container may be only tens of megabytes in size, whereas a virtual machine with its own entire operating system may be several gigabytes in size. Because of this, a single server can host far more containers than virtual machines.

Another major benefit of using containers is that virtual machines may take several minutes to boot up their operating systems and begin running the applications they host, while containerized applications can be started almost instantly. That means containers can be instantiated in a "just in time" fashion when they are needed and can disappear when they are no longer required, freeing up resources on their hosts.

A third benefit is that containerization allows for greater modularity. Rather than run an entire complex application inside a single container, the application can be split in to modules (such as the database, the application front end, and so on). This is the so-called microservices approach. Applications built in this way are easier to manage because each module is relatively simple, and changes can be made to modules without having to rebuild the entire application. Because containers are so lightweight, individual modules (or microservices) can be instantiated only when they are needed and are available almost immediately.

We shall now cite some important research papers that led to this synopsis and become the guidance to our orchestration model-

In [7] the researchers find that Cloud computing makes extensive use of virtual machines (VMs) because they permit workloads to be isolated from one another and for the resource usage to be somewhat controlled. However, the extra levels of abstraction involved in virtualization reduce workload performance, which is passed on to customers as worse price/performance.Advancement in container-based virtualization simplifies the

deployment of applications while continuing to permit control of the resources allocated to different applications.

In [8] the authors say that,Containers as a lightweight technology to virtualise applications have recently been successful, particularly to manage applications in the cloud. Often, the management of clusters of containers becomes essential and the orchestration of the construction and deployment becomes a central problem. This emerging topic has been taken up by researchers, but there is currently no secondary study to consolidate this research.

Through this research they aimed to identify, taxonomically classify and systematically compare the existing research body on containers and their orchestration and specifically the application of this technology in the cloud. They had conducted a systematic mapping study of 46 selected studies. They classified and compared the selected studies based on a characterisation framework. This results in a discussion of agreed and emerging concerns in the container orchestration space, positioning it within the cloud context, but also moving it closer to current concerns in cloud platforms, microservices and continuous development.

In the citation [9] the authors write about design patterns to improve the performance of containers using a specific application.
This paper describes three types of design patterns that we have observed emerging in container-based distributed systems: single-container patterns for container management, single-node patterns of closely cooperating containers, and multi-node patterns for distributed algorithms. Like object-oriented patterns before them, these patterns for distributed computation encode best practices, simplify development, and make the systems where they are used more reliable.

In [10] paper, they consider how to best integrate container technology into an existing workflow system, using Makeflow, Work Queue, and Docker as examples of current technology. A brief performance study of Docker shows very little overhead in CPU and I/O performance, but significant costs in creating and deleting containers. Taking this into account, they described four different methods of connecting containers to different

points of the infrastructure and explain several methods of managing the container images that must be distributed to executing tasks. They explored the performance of a large bioinformatics workload on a Docker-enabled cluster, and observe the best configuration to be locally-managed containers that are shared between multiple tasks.

Objective

To demonstrate a system which can horizontally scale a web application hosted on containers and deploying those containers with multilevel queue scheduling algorithm?

Sub-objectives:

- Making a 3 tier web application
- Creating a container environment for the required web application.
- Commiting a full fledged container image for a web application.
- Using multilevel queue scheduling algorithm to orchestrate and deploy the container environment.
- Knitting of different services together as a stable automation using Make utility as a build tool.
- Extracting the state of the subsystems, so that while restarting,the image of the process can be reconstructed.

System Requirement

Hardware Requirements:

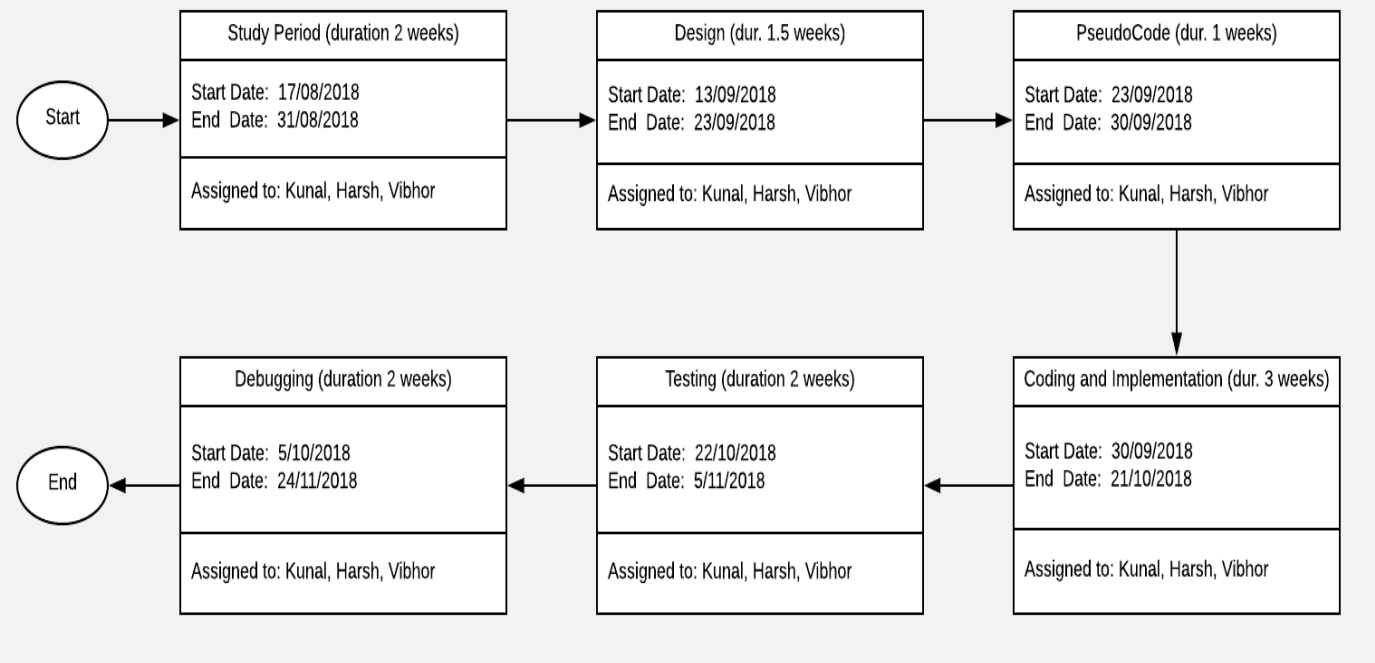
- x86_64/amd64 architecture
- 4.00 GB of RAM
- 30.00 GB of Disk Storage
- An Ethernet type Network Adapter

Software Requirements:

- Ubuntu (LTS 14.04 and above with Linux kernel version 3.10 or latest)
- Docker (10.0 and above)

- Container images of web server and web application
- GCC compiler

Schedule [PERT CHART]



References

[1] Docker Documentation
Available: https://www.docker.com/resources/what-container.

[2] Martin Flower, "Article on Microservices Architecture and Characteristics of microservices", 25th March 2014

[3] make (1) - Linux Man Pages
make: GNU make utility to maintain groups of programs

[4] Docker Official Resources
Available: https://www.docker.com/resources

[5] MatthewRevell,"Introduction to Container Orchestration-Docker, Kubernetes,and Mesos" a guide to compare strengths of orchestration tools available, Aug. 01, 16 · Cloud Zone · Opinion

[6] Kubernetes Documentation
Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[7] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio, IBM Research Report, An Updated Performance Comparison of Virtual Machines and Linux Containers.

[8] Pahl, Claus & Brogi, Antonio & Soldani, Jacopo & Jamshidi, Pooyan. (2017). Cloud Container Technologies: a State-of-the-Art Review. IEEE Transactions on Cloud Computing. PP. 1-1. 10.1109/TCC.2017.2702586.

[9] Brendan Burns,David Oppenheimer..Google..Design patterns for container based distributed systems.

[10] Chao Zheng and Douglas Thain ,Department of Computer Science and Engineering, University of Notre Dame. Integrating Containers into Workflows: A Case Study using Makeflow, Work Queue, and Docker

Approved By

(Dr Monit Kapoor)
Project Guide

(Dr Amit Agrawal)
Dept Head