A REPORT OF
MAJOR PROJECT
On
Framework for Deploying Containerized Application

Submitted in partial fulfillment of the requirements
for the award of the Degree of

Bachelor of Technology
of
Poornima University, Jaipur

Session: 2020-21

Submitted By:
Mohit Soni
[2018PUSETBCCX06516]
IVth Year, Computer Engineering (CT & IS)

Submitted To:
Department of Computer Engineering
School of Engineering & Technology, Poornima University
Ramchandrapura, Sitapura Ext., Jaipur, Rajasthan

CERTIFICATE

Certified that Major Project work entitled “Framework for Deploying Containerized Application**”** is a bonafide work carried out in the eighth semester by “Mohit Soni” in partial fulfilment for the award of Bachelor of Technology in Computer Science & Engineering with specialization in Cloud Technology & Information Security from Poornima University Jaipur, during the academic year 2020 - 2021.

SIGNATURE
(COORDINATOR)

SIGNATURE
(BRANCH COORDINATOR)

ACKNOWLEDGEMENT

I have undergone a Major Project which was meticulously planned and guided at every stage so that it became a life time experience for me. This could not be realized without the help from numerous sources and people in the Poornima University and Programming Express, Ajmer.

I am thankful to **Dr. Manoj Gupta, Provost, Poornima University** for providing us a platform to carry out this activity successfully.

I am also very grateful to **Mr. Ravi Kumar (HOD, Computer Engineering)** for his kind support and guidance.

I would like to take this opportunity to show our gratitude towards **Mr. Ishtiyaq Ahmad Khan** who helped me in successful completion of my Major Project. He has been a guide, motivator & source of inspiration for us to carry out the necessary proceedings for completing this training and related activities successfully and grateful for her/him guidance and support. I am thankful for their kind support and providing us expertise of the domain to develop the project.

I would also like to express our hearts felt appreciation to all of our friends whom direct or indirect suggestions help us to develop this project and to entire team members for their valuable suggestions.

Lastly, thanks to all faculty members of Department of Computer Engineering for their moral support and guidance.

Name: Mohit Soni

Page Index

Figure Index

Abstract

My project was based on DevOps technology in which I design a Kubernetes cluster. Which provides Kubernetes, at its basic level, is a system for running and coordinating containerized applications across a cluster of machines? It is a platform designed to completely manage the lifecycle of containerized applications and services using methods that provide predictability, scalability, and high availability. We have to make bash script to make a Kubernetes cluster.

As a Kubernetes user, you can define how your applications should run and the ways they should be able to interact with other applications or the outside world. You can scale your services up or down, perform graceful rolling updates, and switch traffic between different versions of your applications to test features or rollback problematic deployments. Kubernetes provides interfaces and compostable platform primitives that allow you to define and manage your applications with high degrees of flexibility, power, and reliability.

- Launching Kubernetes cluster
    - Installing Kubernetes
    - Configuring Networking
    - Setting up firewall
    - Loading Kernel Module
    - Making Registry
- Application Deployment on Kubernetes

1. INTRODUCTION

1.1 Aims and Objective:

The purpose of the Major Project is gaining exposure for the students on practical engineering fields. Through this, students will have better understanding of engineering practice in general and sense of frequent and possible problems.
The main objective of the Major Project is to provide an opportunity to undergraduates to identify, observe and practice how engineering is applicable in the real industry. It is not only to get experience on technical practices but also to observe management practices and to interact with fellow workers.

1.2 Scope:

One of the most necessary reasons for keeping a TO-DO List is that the organization. Organizing your tasks will create everything rather more manageable and cause you to feel grounded. Seeing a transparent define of your completed and uncompleted tasks can assist you feel organized and keep mentally centered.

As you cross things off your stir list, you may feel a way of progress and accomplishment which will be incomprehensible once speeding from one activity to successive. The affirmation that you just square measure creating progress can facilitate encourage you to stay moving forward instead of feeling engulfed.

Having an inventory of all of your tasks can permit you to sit down down and create a concept. One study showed that fifteen minutes spent coming up with might save associate hour of execution time!

1.3 Duration and Schedule

Duration of training: 01/02/2021 to 30/03/2021

Total Weeks – 8

Total Days - 58

Table 1.4.1 Schedule of Training

S No.	Learning
1.	Kubernetes
2.	Google Cloud Platform
3.	K3S
4.	Docker
5.	Bash

1.4 Conclusion

As an undergraduate student I would like to say that Major Project is an excellent opportunity for us to get to the ground level and experience the things that we would have never gained through going into a job straightly. Major Project was the first opportunity I got to apply the theories I learnt with the real industry for the real situations. This also gave me the chance to move with different types of people in the industry. Having exposed to such situations I was able to obtain lot of experiences which will be definitely helpful to success my future career as an Engineer.

Finally, I can say with a great pleasure that the 8 weeks of Major Project was a helpful period for me as an engineering undergraduate. I could meet my expectations of Major Project during the period I underwent training. The experiences I gained through this Major Project will be a strong foundation to my career as an engineer. I hope that the university would continue to conduct such Major Project for undergraduates even more effectively and efficient.

2. TECHNOLOGY LEARNED

2.1 KUBERNETES:

Kubernetes (also known as k8s or "kube") is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

In other words, you can cluster together groups of hosts running Linux® containers, and Kubernetes helps you easily and efficiently manage those clusters.

Kubernetes clusters can span hosts across on-premise, public, private, or hybrid clouds. For this reason, Kubernetes is an ideal platform for hosting cloud-native applications that require rapid scaling, like real-time data streaming through Apache Kafka.

Kubernetes was originally developed and designed by engineers at Google. Google was one of the early contributors to Linux container technology and has talked publicly about how everything at Google runs in containers. (This is the technology behind Google's cloud services.)

Google generates more than 2 billion container deployments a week, all powered by its internal platform, Borg. Borg was the predecessor to Kubernetes, and the lessons learned from developing Borg over the years became the primary influence behind much of Kubernetes technology.

``

kubernetes

The primary advantage of using Kubernetes in your environment, especially if you are optimizing app dev for the cloud, is that it gives you the platform to schedule and run containers on clusters of physical or virtual machines (VMs).

More broadly, it helps you fully implement and rely on a container-based infrastructure in production environments. And because Kubernetes is all about automation of operational tasks, you can do many of the same things other application platforms or management systems let you do—but for your containers.

Developers can also create cloud-native apps with Kubernetes as a runtime platform by using Kubernetes patterns. Patterns are the tools a Kubernetes developer needs to build container-based applications and services.

With Kubernetes you can:

- Orchestrate containers across multiple hosts.
- Make better use of hardware to maximize resources needed to run your enterprise apps.
- Control and automate application deployments and updates.
- Mount and add storage to run stateful apps.
- Scale containerized applications and their resources on the fly.
- Declaratively manage services, which guarantees the deployed applications are always running the way you intended them to run.
- Health-check and self-heal your apps with auto placement, auto restart, auto replication, and autoscaling.

However, Kubernetes relies on other projects to fully provide these orchestrated services. With the addition of other open-source projects, you can fully realize the power of Kubernetes. These necessary pieces include (among others):

- Registry, through projects like Docker Registry.
- Networking, through projects like Open Switch and intelligent edge routing.
- Telemetry, through projects such as Kibana, Hawkular, and Elastic.
- Security, through projects like LDAP, SE Linux, RBAC, and OAUTH with multitenancy layers.
- Automation, with the addition of Ansible playbooks for installation and cluster life cycle management.
- Services, through a rich catalog of popular app patterns.

As is the case with most technologies, language specific to Kubernetes can act as a barrier to entry. Let's break down some of the more common terms to help you better understand Kubernetes.

Control plane: The collection of processes that control Kubernetes nodes. This is where all task assignments originate.

Nodes: These machines perform the requested tasks assigned by the control plane.

Pod: A group of one or more containers deployed to a single node. All containers in a pod share an IP address, IPC, hostname, and other resources. Pods abstract network and storage from the underlying container. This lets you move containers around the cluster more easily.

Replication controller: This controls how many identical copies of a pod should be running somewhere on the cluster.

Service: This decouples work definitions from the pods. Kubernetes service proxies automatically get service requests to the right pod—no matter where it moves in the cluster or even if it’s been replaced.

Kubelet: This service runs on nodes, reads the container manifests, and ensures the defined containers are started and running.

kubectl: The command line configuration tool for Kubernetes.

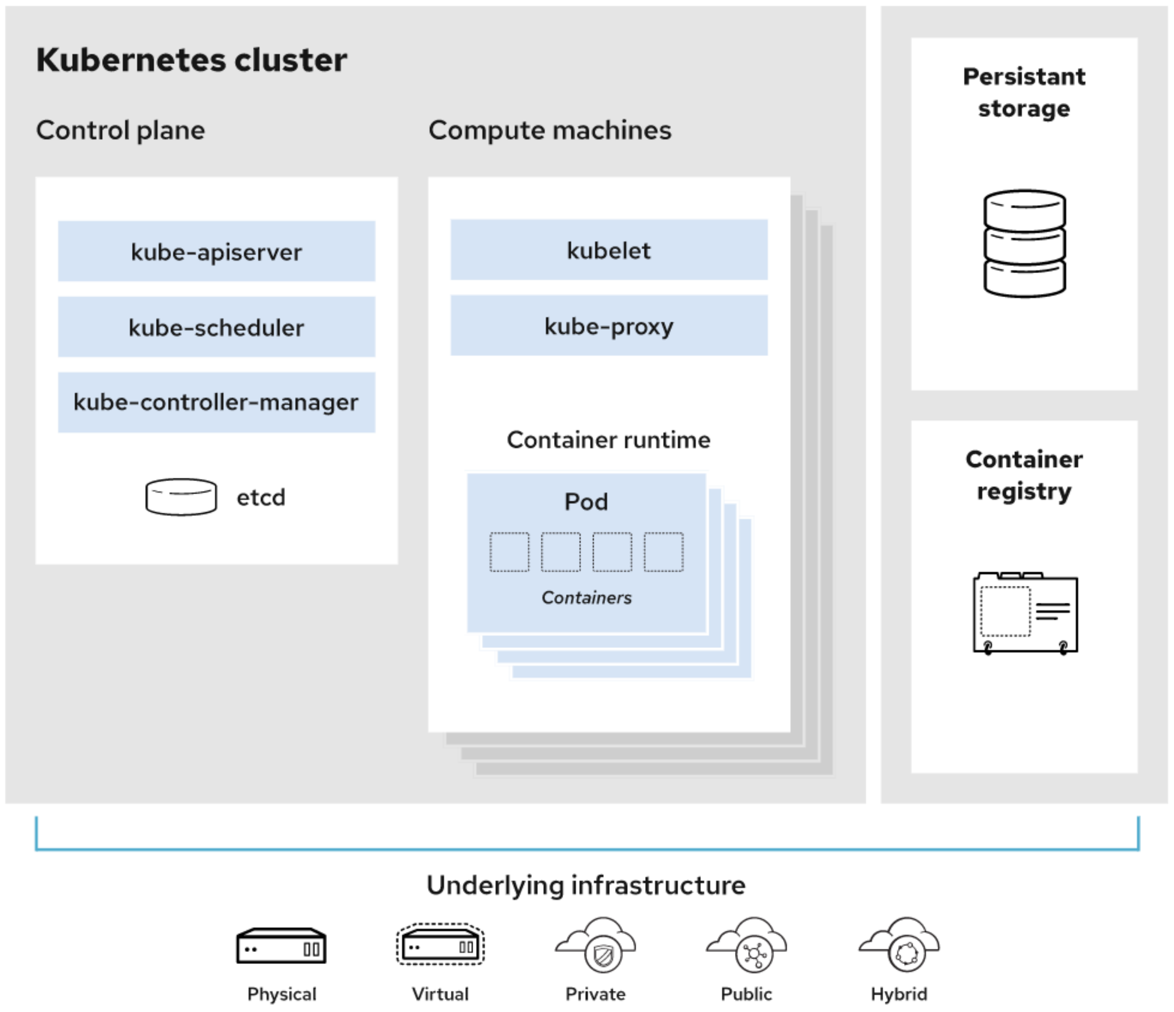


Figure 1 Kubernetes Cluster

A working Kubernetes deployment is called a cluster. You can visualize a Kubernetes cluster as two parts: the control plane and the compute machines, or nodes.

Each node is its own Linux® environment, and could be either a physical or virtual machine. Each node runs pods, which are made up of containers.

The control plane is responsible for maintaining the desired state of the cluster, such as which applications are running and which container images they use. Compute machines actually run the applications and workloads.

Kubernetes runs on top of an operating system (Red Hat® Enterprise Linux®, for example) and interacts with pods of containers running on the nodes.

The Kubernetes control plane takes the commands from an administrator (or DevOps team) and relays those instructions to the compute machines.

This handoff works with a multitude of services to automatically decide which node is best suited for the task. It then allocates resources and assigns the pods in that node to fulfill the requested work.

The desired state of a Kubernetes cluster defines which applications or other workloads should be running, along with which images they use, which resources should be made available to them, and other such configuration details.

From an infrastructure point of view, there is little change to how you manage containers. Your control over containers just happens at a higher level, giving you better control without the need to micromanage each separate container or node.

Your work involves configuring Kubernetes and defining nodes, pods, and the containers within them. Kubernetes handles orchestrating the containers.

Where you run Kubernetes is up to you. This can be on bare metal servers, virtual machines, public cloud providers, private clouds, and hybrid cloud environments. One of Kubernetes' key advantages is it works on many different kinds of infrastructure.

NEED:

Kubernetes can help you deliver and manage containerized, legacy, and cloud-native apps, as well as those being refactored into microservices.

In order to meet changing business needs, your development team needs to be able to rapidly build new applications and services. Cloud-native development starts with microservices in containers, which enables faster development and makes it easier to transform and optimize existing applications.

Watch this webinar series to get expert perspectives to help you establish the data platform on enterprise Kubernetes you need to build, run, deploy, and modernize applications.

Production apps span multiple containers, and those containers must be deployed across multiple server hosts. Kubernetes gives you the orchestration and management capabilities required to deploy containers, at scale, for these workloads.

Kubernetes orchestration allows you to build application services that span multiple containers, schedule those containers across a cluster, scale those containers, and manage the health of those containers over time. With Kubernetes you can take effective steps toward better IT security.

Kubernetes also needs to integrate with networking, storage, security, telemetry, and other services to provide a comprehensive container infrastructure.

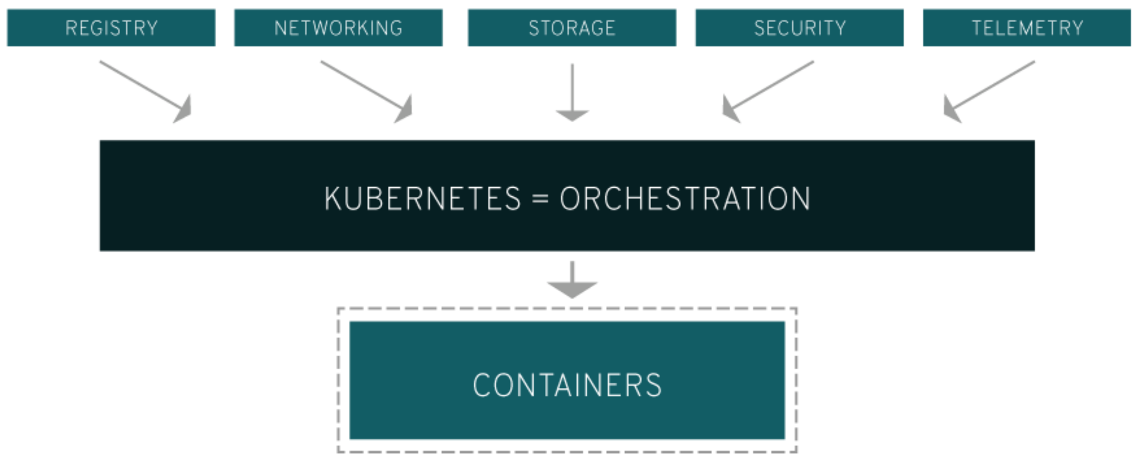


Figure 2 Kubernetes Orchestration

Once you scale this to a production environment and multiple applications, it's clear that you need multiple, colocated containers working together to deliver the individual services.

Linux containers give your microservice-based apps an ideal application deployment unit and self-contained execution environment. And microservices in containers make it easier to orchestrate services, including storage, networking, and security.

This significantly multiplies the number of containers in your environment, and as those containers accumulate, the complexity also grows.

Kubernetes fixes a lot of common problems with container proliferation by sorting containers together into "pods." Pods add a layer of abstraction to grouped containers, which helps you schedule workloads and provide necessary services—like networking and storage—to those containers.

Other parts of Kubernetes help you balance loads across these pods and ensure you have the right number of containers running to support your workloads.

With the right implementation of Kubernetes—and with the help of other open source projects like Open vSwitch, OAuth, and SELinux— you can orchestrate all parts of your container infrastructure.

2.2 GOOGLE CLOUD PLATFORM:

Google Cloud consists of a set of physical assets, such as computers and hard disk drives, and virtual resources, such as virtual machines (VMs), that are **contained in Google's** data centers around the globe. Each data center location is in a region.

Google Cloud Platform (**GCP**), offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail, file storage, and YouTube.

2.2.1 Compute

App Engine: App Engine enables you to build and host applications on the same systems that power Google applications. App Engine offers fast development and deployment; simple administration, with no need to worry about hardware, patches or backups; and effortless scalability.

Compute Engine: Compute Engine offers scalable and flexible virtual machine computing capabilities in the cloud, with options to utilize certain CPUs, GPUs, or Cloud TPUs. You can use Compute Engine to solve large-scale processing and analytic problems on Google's computing, storage, and networking infrastructure.

Google Cloud VMware Engine (GCVE) is a managed VMware-as-a-Service that is specifically designed for running VMware workloads on Google Cloud Platform. GCVE enables customers to run VMware virtual machines natively in a dedicated, private, software-defined data center.

2.2.2 Storage

Cloud Storage: Cloud Storage is a RESTful service for storing and accessing your data on Google's infrastructure. The service combines the performance and scalability of Google's cloud with advanced security and sharing capabilities.

Persistent Disk is durable and high performance block storage for Google Cloud Platform. Persistent Disk provides SSD and HDD storage that can be attached to instances running in either Compute Engine or Google Kubernetes Engine.

Cloud Filestore: Cloud Filestore is a scalable and highly available shared file service fully managed by Google. Cloud Filestore provides persistent storage ideal for shared workloads. It is best suited for enterprise applications requiring persistent, durable, shared storage which is accessed by NFS or requires a POSIX compliant file system.

***Cloud Storage for Firebase**: Cloud Storage for Firebase adds customizable Google security (via Firebase Security Rules for Cloud Storage) to file uploads and downloads for your Firebase apps, as well as robust uploads and downloads regardless of network quality through the Firebase SDK. Cloud Storage for Firebase is backed by Cloud Storage, a service for storing and accessing your data on Google's infrastructure.

2.2.3 Databases

Cloud Bigtable: Cloud Bigtable is a fast, fully managed, highly-scalable NoSQL database service. It is designed for the collection and retention of data from 1TB to hundreds of PB.

Datastore: Datastore is a fully managed, schemaless, non-relational datastore. It provides a rich set of query capabilities, supports atomic transactions, and automatically scales up and down in response to load. It can scale to support an application with 1,000 users or 10 million users with no code changes.

Firestore: Firestore is a NoSQL document database for storing, syncing, and querying data for mobile and web apps. Its client libraries provide live synchronization and offline support, while its security features and integrations with Firebase and Google Cloud Platform accelerate building serverless apps.

Memorystore: Memorystore, which includes Memorystore for Redis and Memorystore for Memcached, provides a fully managed in-memory data store service that allows customers to deploy distributed caches that provide sub-millisecond data access.

Cloud Spanner: Cloud Spanner is a fully managed, mission-critical relational database service. It is designed to provide a scalable online transaction processing (OLTP) database with high availability and strong consistency at global scale.

Cloud SQL: Cloud SQL is a web service that allows you to create, configure, and use relational databases that live in Google's cloud. It is a fully-managed service that maintains, manages, and administers your databases, allowing you to focus on your applications and services.

2.2.4 Networking

Cloud CDN: Cloud CDN uses Google's globally distributed edge points of presence to cache HTTP(S) load balanced content close to your users.

Cloud DNS: Cloud DNS is a high performance, resilient, global, fully managed DNS service that provides a RESTful API to publish and manage DNS records for your applications and services.

Cloud Interconnect: Cloud Interconnect offers enterprise-grade connections to Google Cloud Platform using Google Services for Dedicated Interconnect, Partner Interconnect and Cloud VPN. This solution allows you to directly connect your on-premises network to your Virtual Private Cloud.

Cloud Load Balancing: Cloud Load Balancing provides scaling, high availability, and traffic management for your internet-facing and private applications.

Cloud NAT (Network Address Translation): Cloud NAT enables instances in a private network to communicate with the internet.

Cloud Router: Cloud Router enables dynamic Border Gateway Protocol (BGP) route updates between your VPC network and your non-Google network.

Cloud VPN: Cloud VPN allows you to connect to your Virtual Private Cloud (VPC) network from your existing network, such as your on-premises network, another VPC network, or another cloud provider's network, through an IPsec connection using (i) Classic VPN, which supports dynamic (BGP) routing or static routing (route-based or policy-based), or (ii) HA (high-availability) VPN, which supports dynamic routing with a simplified redundancy setup, separate failure domains for the gateway interfaces, and a higher service level objective.

Google Cloud Armor: Google Cloud Armor offers a policy framework and rules language for customizing access to internet-facing applications and deploying defenses against denial of service attacks.

Network Intelligence Center: Network Intelligence Center is Google Cloud’s comprehensive network monitoring, verification, and optimization platform across the Google Cloud, multi-cloud, and on-prem environments.

Network Service Tiers: Network Service Tiers enable you to select different quality networks (tiers) for outbound traffic to the internet: the Standard Tier primarily utilizes third party transit providers while the Premium Tier leverages Google's private backbone and peering surface for egress.

Service Directory is a managed service that offers customers a single place to publish, discover and connect their services in a consistent way, regardless of their environment. Service Directory supports services in Google Cloud, multi-cloud, and on-prem environments and can scale up to thousands of services and endpoints for a single project.

Traffic Director: Traffic Director is Google Cloud Platform's traffic management service for open service meshes.

Virtual Private Cloud: Virtual Private Cloud provides a private network topology with IP allocation, routing, and network firewall policies to create a secure environment for your deployments.

2.2.5 Operations

Cloud Debugger: Cloud Debugger connects your application's production data to your source code by inspecting the state of your application at any code location in production without stopping or slowing down your requests.

Cloud Logging: Cloud Logging is a fully managed service that performs at scale and can ingest application and system log data, as well as custom log data from thousands of VMs and containers. Cloud Logging allows you to analyze and export selected logs to long-term storage in real time. Cloud Logging includes the Error Reporting feature, which analyzes and aggregates the errors in your cloud applications and notifies you when new errors are detected.

Cloud Monitoring: Cloud Monitoring provides visibility into the performance, uptime, and overall health of cloud-powered applications. Cloud Monitoring collects metrics, events, and metadata from certain Services, hosted uptime probes, application instrumentation, alert management, notifications and a variety of common application components.

Cloud Profiler: Cloud Profiler provides continuous profiling of resource consumption in your production applications, helping you identify and eliminate potential performance issues.

Cloud Trace: Cloud Trace provides latency sampling and reporting for App Engine, including per-URL statistics and latency distributions.

2.3 K3S:

Perfect for Edge

K3s is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances.

Simplified & Secure

K3s is packaged as a single <40MB binary that reduces the dependencies and steps needed to `

Optimized for ARM

Both ARM64 and ARMv7 are supported with binaries and multiarc images available for both. K3s works great from something as small as a Raspberry Pi to an AWS a1.4xlarge 32GiB server.

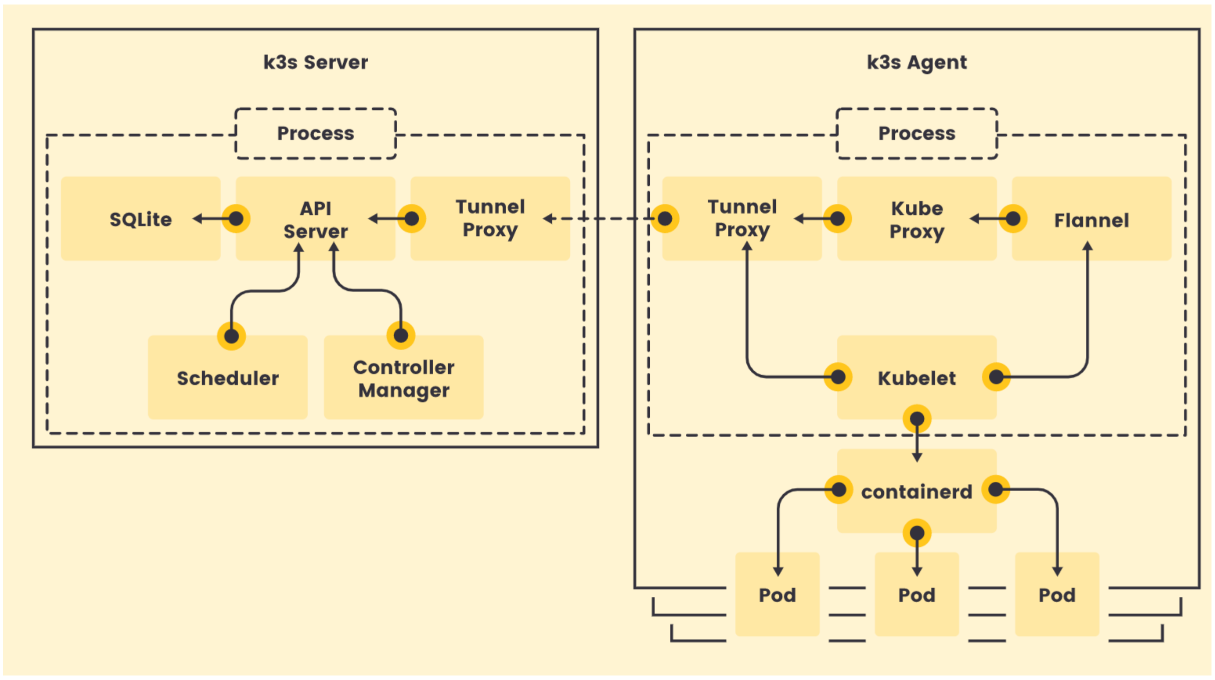


Figure 3 K3S Server-Agent Infrastructure

K3s is designed to be a single binary of less than 40MB that completely implements the Kubernetes API. In order to achieve this, they removed a lot of extra drivers that didn't need to be part of the core and are easily replaced with add-ons.

K3s is a fully CNCF (Cloud Native Computing Foundation) certified Kubernetes offering. This means that you can write your YAML to operate against a regular "full-fat" Kubernetes and they'll also apply against a k3s cluster.

Due to its low resource requirements, it's possible to run a cluster on anything from 512MB of RAM machines upwards. This means that we can allow pods to run on the master, as well as nodes.

And of course, because it's a tiny binary, it means we can install it in a fraction of the time it takes to launch a regular Kubernetes cluster! We generally achieve sub-two minutes to launch a k3s cluster with a handful of nodes, meaning you can be deploying apps to learn/test at the drop of a hat.

Both its reputation and adoption is growing rapidly too, with almost 13k GitHub stars since its launch in early 2019, whilst it was recently crowned the number 1 new developer tool of 2019 by Stack share.

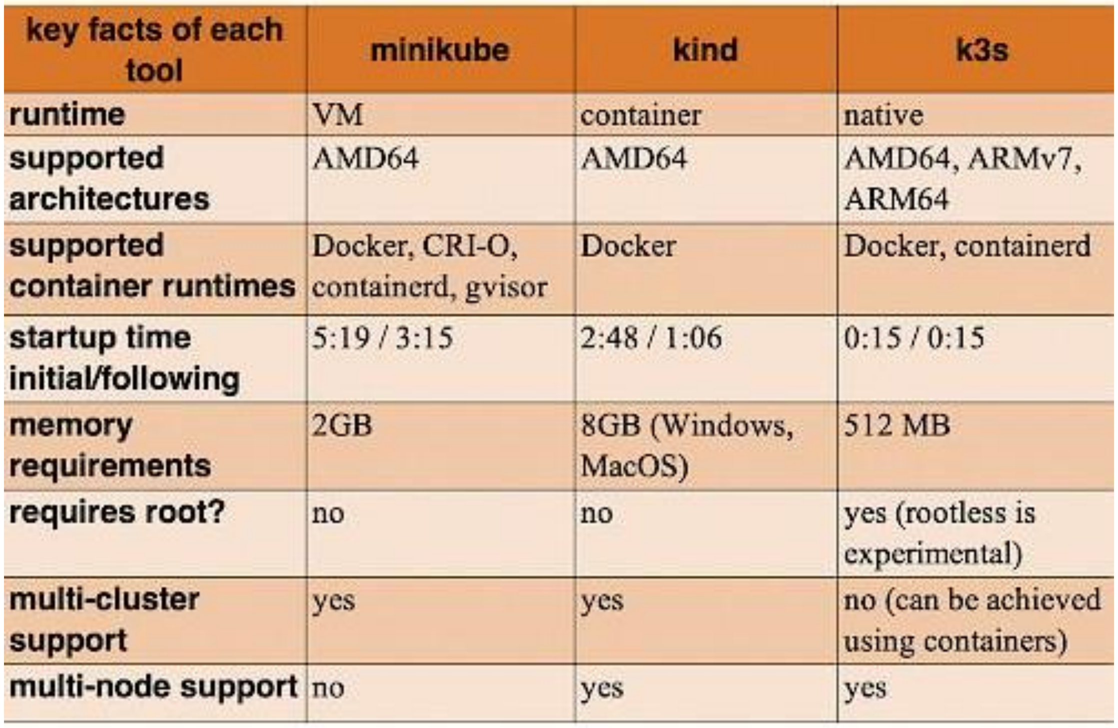
For a quick run through check out this video explanation on the differences between k3s and k8s.

Well, kind of. When most people think of Kubernetes, they think of containers automatically being brought up on other nodes (if the node dies), of load balancing between containers, of isolation and rolling deployments - and all of those advantages are the same between "full-fat" K8s vs k3s.

However, it's not all sunshine and roses, if that was the case everyone would be using k3s. So why not...

Firstly, currently (k3s v0.8.1) there is only the option to run a single master. This means if your master goes down then you lose the ability to manage your cluster any more (although all your existing containers will continue to run). If you want to run an external database platform, you can launch now with multiple masters. And there is some work being done for k3s v1.0 GA to support multiple masters natively within k3s.

Secondly, the default database in single master k3s clusters is SQLite. This is great for small databases not seeing much action, but can quickly become a major pain if they are being hammered! However, the changed happening in a Kubernetes control plane are more about frequently updating deployments, scheduling pods, etc - so the database load isn't too much for a small dev/test cluster.



key facts of each tool	minikube	kind	k3s
runtime	VM	container	native
supported architectures	AMD64	AMD64	AMD64, ARMv7, ARM64
supported container runtimes	Docker, CRI-O, containerd, gvisor	Docker	Docker, containerd
startup time initial/following	5:19 / 3:15	2:48 / 1:06	0:15 / 0:15
memory requirements	2GB	8GB (Windows, MacOS)	512 MB
requires root?	no	no	yes (rootless is experimental)
multi-cluster support	yes	yes	no (can be achieved using containers)
multi-node support	no	yes	yes

Figure 4 Different Kubernetes Frameworks

2.4 DOCKER:

Docker is a software platform for building applications based on *containers* — small and lightweight execution environments that make shared use of the operating system kernel but otherwise run-in isolation from one another. While containers as a concept have been around for some time, Docker, an open source project launched in 2013, helped popularize the technology, and has helped drive the trend towards *containerization* and *microservices* in software development that has come to be known as cloud-native development.

2.4.1 CONTAINERS:

One of the goals of modern software development is to keep applications on the same host or cluster isolated from one another so they don't unduly interfere with each other's operation or maintenance. This can be difficult, thanks to the packages, libraries, and other software components required for them to run. One solution to this problem has been *virtual machines*, which keep applications on the same hardware entirely separate, and reduce conflicts among software components and competition for hardware resources to a minimum. But virtual machines are bulky—each requires its own OS, so is typically gigabytes in size—and difficult to maintain and upgrade.

Containers, by contrast, isolate applications' execution environments from one another, but share the underlying OS kernel. They're typically measured in megabytes, use far fewer resources than VMs, and start up almost immediately. They can be packed far more densely on the same hardware and spun up and down *en masse* with far less effort and overhead. Containers provide a highly efficient and highly granular mechanism for combining software components into the kinds of application and service stacks needed in a modern enterprise, and for keeping those software components updated and maintained.

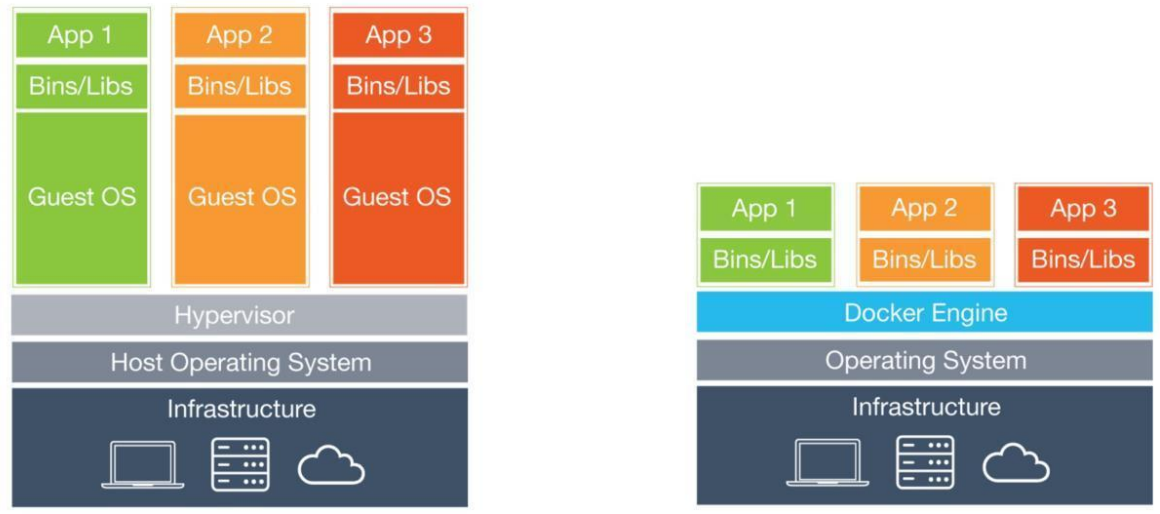


Figure 5 Virtual Machines vs Containers

Docker is an open-source project that makes it easy to create containers and container-based apps. Originally built for Linux, Docker now runs on Windows and MacOS as well. To understand how Docker works, let's take a look at some of the components you would use to create Docker-containerized applications.

2.4.2 Docker file

Each Docker container starts with a *Docker file*. A Docker file is a text file written in an easy-to-understand syntax that includes the instructions to build a Docker *image* (more on that in a moment). A Docker file specifies the operating system that will underlie the container, along with the languages, environmental variables, file locations, network ports, and other components it needs—and, of course, what the container will actually be doing once we run it.

2.4.3 Docker image

Once you have your Docker file written, you invoke the Docker build utility to create an *image* based on that Docker file. Whereas the Docker file is the set of instructions that tells build how to make the image, a Docker image is a portable file containing the specifications for which software components the container will run and how. Because a Docker file will probably include instructions about grabbing some software packages from online repositories, you should take care to explicitly specify the proper versions, or else your Docker file might produce inconsistent images depending on when it's invoked. But once an image is created, it's static. Code fresh offers a look at how to build an image in more detail.

2.4.4 Docker run

Docker's run utility is the command that actually launches a container. Each container is an *instance* of an image. Containers are designed to be transient and temporary, but they can be stopped and restarted, which launches the container into the same state as when it was stopped. Further, multiple container instances of the same image can be run simultaneously (as long as each container has a unique name). The Code Review has a great breakdown of the different options for the run command, to give you a feel for how it works.

2.4.5 Docker Hub

While building containers is easy, don't get the idea that you'll need to build each and every one of your images from scratch. Docker Hub is a SaaS repository for sharing and managing containers, where you will find official Docker images from open-source projects and software vendors and unofficial images from the general public. You can download container images containing useful code, or upload your own, share them openly, or make them private instead. You can also create a local Docker registry if you prefer. (Docker Hub has had problems in the past with images that were uploaded with backdoors built into them.)

2.4.6 Docker Engine

Docker Engine is the core of Docker, the underlying client-server technology that creates and runs the containers. Generally speaking, when someone says *Docker* generically and isn't talking about the company or the overall project, they mean Docker Engine. There are two different versions of Docker Engine on offer: Docker Engine Enterprise and Docker Engine Community.

2.4.7 Docker Community Edition

Docker released its *Enterprise Edition* in 2017, but its original offering, renamed Docker Community Edition, remains open source and free of charge, and did not lose any features in the process. Instead, Enterprise Edition, which costs $1,500 per node per year, added advanced management features including controls for cluster and image management, and vulnerability monitoring. The Box Boat blog has a rundown of the differences between the editions.

2.4.8 Docker Compose, Docker Swarm, and Kubernetes

Docker also makes it easier to coordinate behaviors *between* containers, and thus build application stacks by hitching containers together. Docker Compose was created by Docker to simplify the process of developing and testing multi-container applications. It's a command-line tool, reminiscent of the Docker client, that takes in a specially formatted descriptor file to assemble applications out of multiple containers and run them in concert on a single host. (Check out InfoWorld's Docker Compose tutorial to learn more.)

More advanced versions of these behaviors—what's called *container orchestration*—are offered by other products, such as Docker Swarm and Kubernetes. But Docker provides the basics. Even though Swarm grew out of the Docker project, Kubernetes has become the *de facto* Docker orchestration platform of choice.

2.4.9 Docker advantages

Docker containers provide a way to build enterprise and line-of-business applications that are easier to assemble, maintain, and move around than their conventional counterparts.

2.4.10 Docker containers enable isolation and throttling

Docker containers keep apps isolated not only from each other, but from the underlying system. This not only makes for a cleaner software stack, but makes it easier to dictate how a given containerized application uses system resources—CPU, GPU, memory, I/O, networking, and so on. It also makes it easier to ensure that data and code are kept separate. (See "Docker containers are stateless and immutable," below.)

2.4.11 Docker containers enable portability

A Docker container runs on any machine that supports the container's runtime environment. Applications don't have to be tied to the host operating system, so both the application environment and the underlying operating environment can be kept clean and minimal.

For instance, a MySQL for Linux container will run on most any Linux system that supports containers. All of the dependencies for the app are typically delivered in the same container.

Container-based apps can be moved easily from on-prem systems to cloud environments or from developers' laptops to servers, as long as the target system supports Docker and any of the third-party tools that might be in use with it, such as Kubernetes (see "Docker containers ease orchestration and scaling," below).

Normally, Docker container images must be built for a specific platform. A Windows container, for instance, will not run on Linux and vice versa. Previously, one way around this limitation was to launch a virtual machine that ran an instance of the needed operating system, and run the container in the virtual machine.

However, the Docker team has since devised a more elegant solution, called *manifests*, which allow images for multiple operating systems to be packed side-by-side in the same image. Manifests are still considered experimental, but they hint at how containers might become a cross-platform application solution as well as a cross-environment one.

2.4.12 Docker containers enable composability

Most business applications consist of several separate components organized into a stack—a web server, a database, an in-memory cache. Containers make it possible to compose these pieces into a functional unit with easily changeable parts. Each piece is provided by a different container and can be maintained, updated, swapped out, and modified independently of the others.

This is essentially the microservices model of application design. By dividing application functionality into separate, self-contained services, the microservices model offers an antidote to slow traditional development processes and inflexible monolithic apps. Lightweight and portable containers make it easier to build and maintain microservices-based applications.

2.4.13 Docker containers ease orchestration and scaling

Because containers are lightweight and impose little overhead, it's possible to launch many more of them on a given system. But containers can also be used to scale an application across clusters of systems, and to ramp services up or down to meet spikes in demand or to conserve resources.

The most enterprise-grade versions of the tools for deployment, managing, and scaling containers are provided by way of third-party projects. Chief among them is Google's Kubernetes, a system for automating how containers are deployed and scaled, but also how they're connected together, load-balanced, and managed. Kubernetes also provides ways to create and re-use multi-container application definitions or "Helm charts," so that complex app stacks can be built and managed on demand.

Docker also includes its own built-in orchestration system, Swarm mode, which is still used for cases that are less demanding. That said, Kubernetes has become something of the default choice; in fact, Kubernetes is bundled with Docker Enterprise Edition.

2.5 BASH:

Bash is the shell, or command language interpreter, for the GNU operating system. The name is an acronym for the ‘Bourne-Again Shell’, a pun on Stephen Bourne, the author of the direct ancestor of the current Unix shell sh, which appeared in the Seventh Edition Bell Labs Research version of Unix.

Bash is largely compatible with sh and incorporates useful features from the Korn shell ksh and the C shell csh. It is intended to be a conformant implementation of the IEEE POSIX Shell and Tools portion of the IEEE POSIX specification (IEEE Standard 1003.1). It offers functional improvements over sh for both interactive and programming use.

The Bash command syntax is a superset of the Bourne shell command syntax. Bash supports brace expansion, command line completion (Programmable Completion), basic debugging and signal handling (using trap) since bash 2.05 among other features. Bash can execute the vast majority of Bourne shell scripts without modification, with the exception of Bourne shell scripts stumbling into fringe syntax behaviour interpreted differently in Bash or attempting to run a system command matching a newer Bash built-in, etc. Bash command syntax includes ideas drawn from the KornShell (ksh) and the C shell (csh) such as command line editing, command history (history command), the directory stack, the $RANDOM and $PPID variables, and POSIX command substitution syntax $(…).

When a user presses the tab key within an interactive command-shell, Bash automatically uses command line completion, since beta version 2.04, to match partly typed program names, filenames and variable names. The Bash command-line completion system is very flexible and customizable, and is often packaged with functions that complete arguments and filenames for specific programs and tasks.

Bash's syntax has many extensions lacking in the Bourne shell. Bash can perform integer calculations ("arithmetic evaluation") without spawning external processes. It uses the ((…)) command and the $((…)) variable syntax for this purpose. Its syntax simplifies I/O redirection. For example, it can redirect standard output (stdout) and standard error (stderr) at the same time using the &> operator. This is simpler to type than the Bourne shell equivalent 'command > file 2>&1'. Bash supports process substitution using the <(command) and >(command)syntax, which substitutes the output of (or input to) a command where a filename is normally used. (This is implemented through /proc/fd/ unnamed pipes on systems that support that, or via temporary named pipes where necessary).

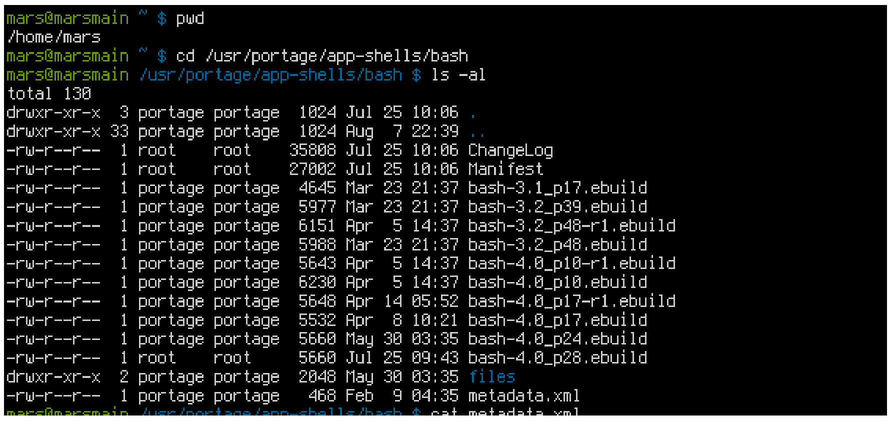
2.5.1 Bash Scripting

One reason Bash (and Linux in general) is considered so powerful is because it's scriptable. Anything you can type into Bash manually, you can also list in a plain-text file and have Bash run it for you. Instead of spending an afternoon manually running a hundred commands, you can script the commands and have your computer execute them while you tend to more important matters. Because nearly everything on Linux runs on top of the Bash shell, nearly everything on Linux can be scripted through Bash. While there are exceptions to this (graphical applications may have their own scripting language, for instance, or no scripting at all), scripting your OS opens up tens of thousands of possible functions you can make happen on your computer without doing them yourself.

The amount of work this saves Linux users each day is impossible to estimate. It's not the usual automation that makes the difference, though; it's the bespoke workflows that people invent for themselves, the things nobody else thinks need automation.

2.5.2 Advantages of Bash Scripting

Bash is as powerful as other shells but adds convenience functions like the double brackets ([[and]]) in the sample code. These "Bashisms" are much loved by Bash users because they avoid the sometimes verbose and awkward syntax in other shells like tcsh or ash. However, they are unique to Bash and are not POSIX-compliant, which could cause compatibility issues on systems not running Bash. Then again, Bash is open source free software, so most users can install it if they need it. The lack of compatibility only forces an extra dependency and does not exclude anyone from using a script.

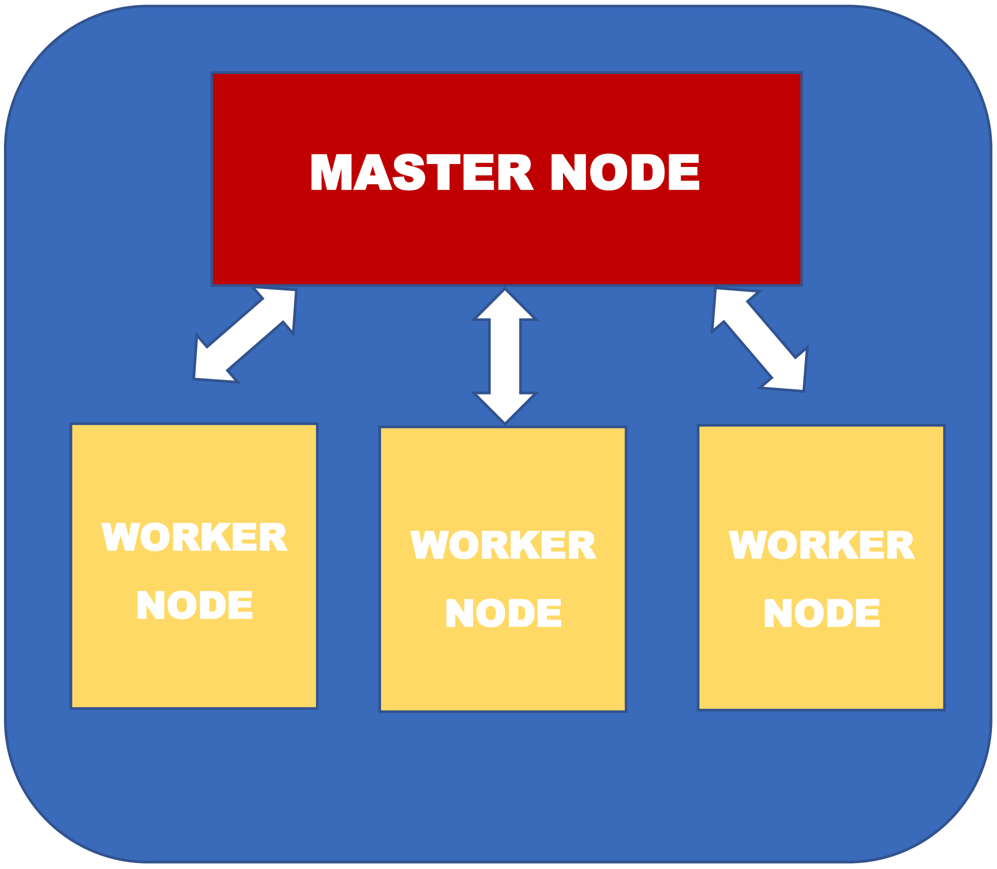


```
mars@marsmain ~ $ pwd
/home/mars
mars@marsmain ~ $ cd /usr/portage/app-shells/bash
mars@marsmain /usr/portage/app-shells/bash $ ls -al
total 130
drwxr-xr-x  3 portage portage  1024 Jul 25 10:06 .
drwxr-xr-x 33 portage portage  1024 Aug  7 22:39 ..
-rw-r--r--  1 root    root    35808 Jul 25 10:06 ChangeLog
-rw-r--r--  1 root    root    27002 Jul 25 10:06 Manifest
-rw-r--r--  1 portage portage  4645 Mar 23 21:37 bash-3.1_p17.ebuild
-rw-r--r--  1 portage portage  5977 Mar 23 21:37 bash-3.2_p39.ebuild
-rw-r--r--  1 portage portage  6151 Apr  5 14:37 bash-3.2_p48-r1.ebuild
-rw-r--r--  1 portage portage  5988 Mar 23 21:37 bash-3.2_p48.ebuild
-rw-r--r--  1 portage portage  5643 Apr  5 14:37 bash-4.0_p10-r1.ebuild
-rw-r--r--  1 portage portage  6230 Apr  5 14:37 bash-4.0_p10.ebuild
-rw-r--r--  1 portage portage  5648 Apr 14 05:52 bash-4.0_p17-r1.ebuild
-rw-r--r--  1 portage portage  5532 Apr  8 10:21 bash-4.0_p17.ebuild
-rw-r--r--  1 portage portage  5660 May 30 03:35 bash-4.0_p24.ebuild
-rw-r--r--  1 root    root     5660 Jul 25 09:43 bash-4.0_p28.ebuild
drwxr-xr-x  2 portage portage  2048 May 30 03:35 files
-rw-r--r--  1 portage portage   468 Feb  9 04:35 metadata.xml
mars@marsmain /usr/portage/app-shells/bash $ cat metadata.xml
```

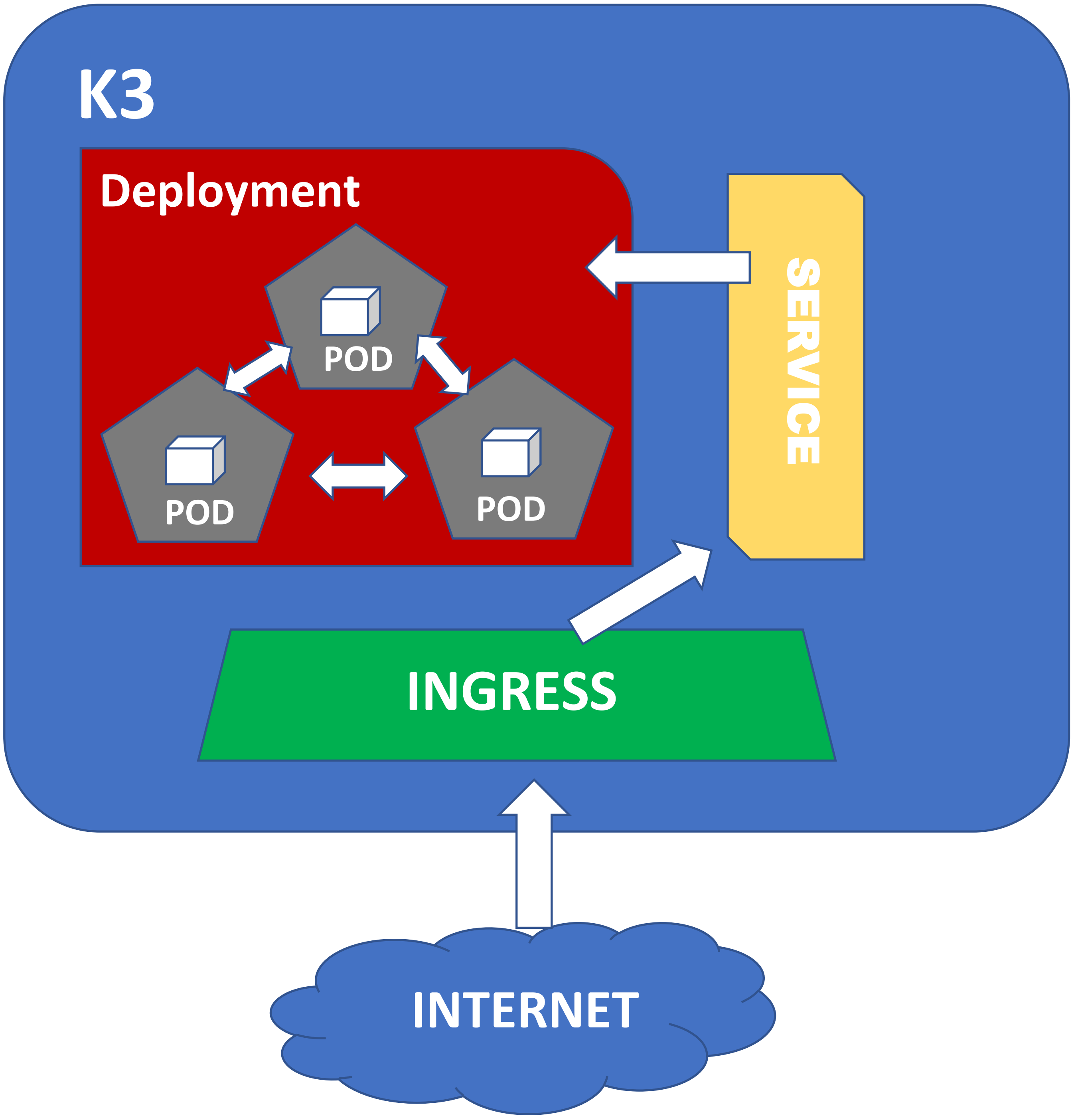
3. Project Development

3.1 Design:

- **Google Cloud Architecture**

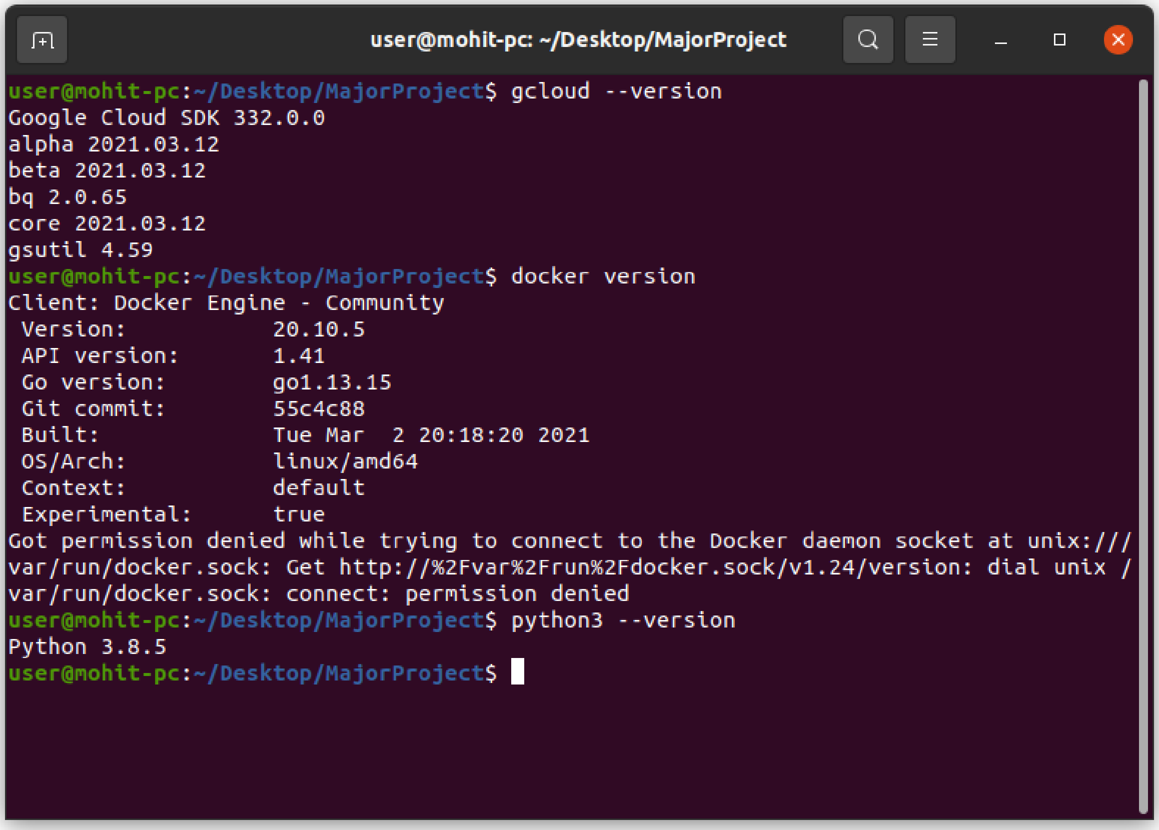


- **K3S Architecture**



3.2 Implementation:

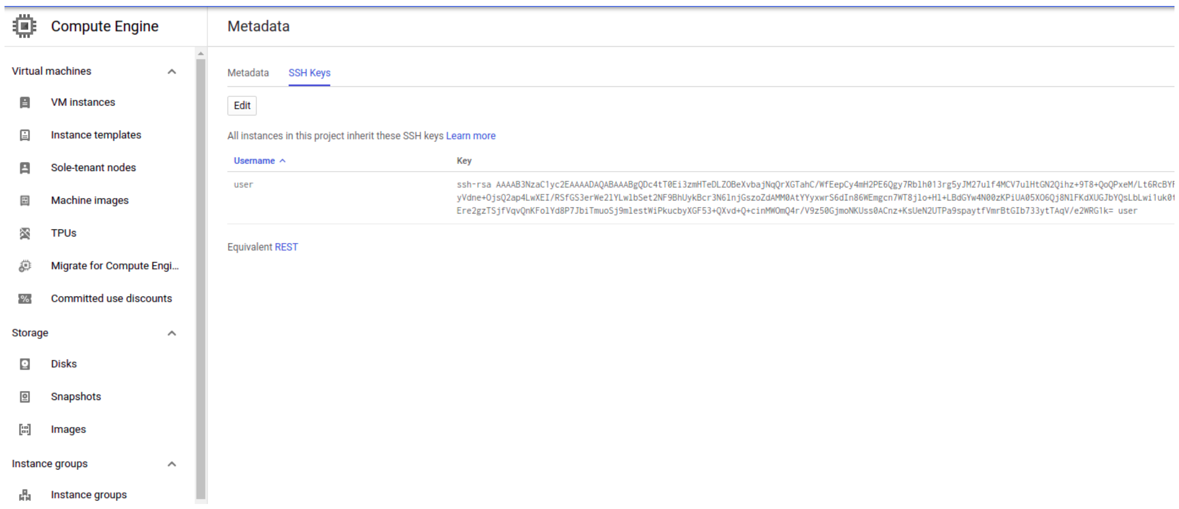
3.2.1 Pre-requisites



user@mohit-pc: ~/Desktop/MajorProject

```
user@mohit-pc:~/Desktop/MajorProject$ gcloud --version
Google Cloud SDK 332.0.0
alpha 2021.03.12
beta 2021.03.12
bq 2.0.65
core 2021.03.12
gsutil 4.59
user@mohit-pc:~/Desktop/MajorProject$ docker version
Client: Docker Engine - Community
 Version:           20.10.5
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        55c4c88
 Built:             Tue Mar  2 20:18:20 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true
Got permission denied while trying to connect to the Docker daemon socket at unix:///
var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.24/version: dial unix /
var/run/docker.sock: connect: permission denied
user@mohit-pc:~/Desktop/MajorProject$ python3 --version
Python 3.8.5
user@mohit-pc:~/Desktop/MajorProject$ 
```

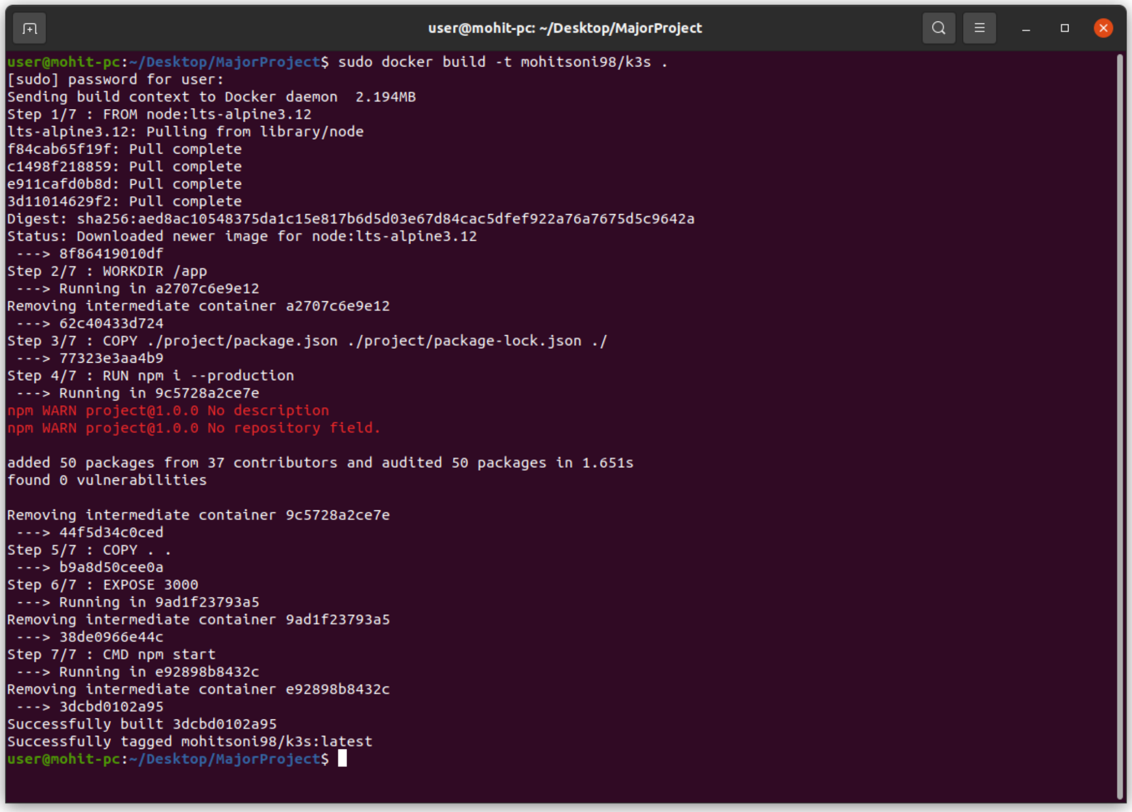
3.2.2 Setting up SSH Keys on Google Cloud



All instances in this project inherit these SSH keys Learn more

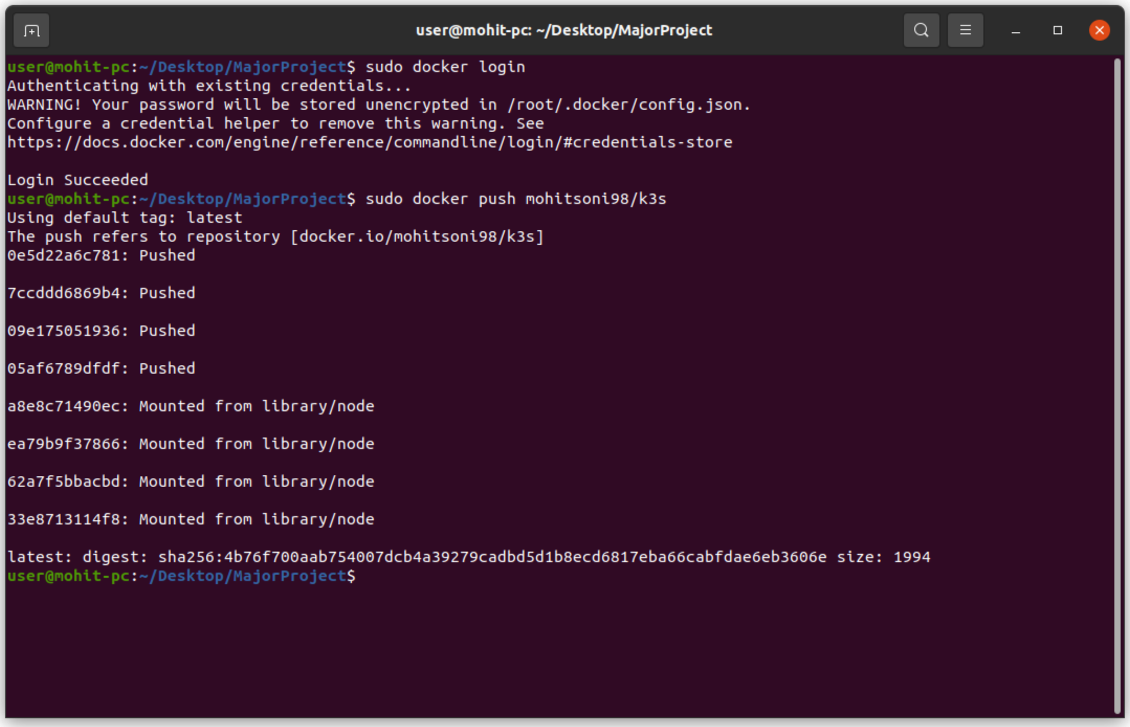
Username	Key
user	ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDc4tT0Ei3zmHTeDLZOBeXvbajNqQrXGTahC/WfEepCy4mH2PE6Qgy7Rblh013rg5yJM27ulf4MCV7ulHtGN2Qihz+9T8+QoQPxeM/Lt6RcBYF yVdne+OjsQ2ap4LwXEI/RSfGS3erWe2lYLwlbSet2NF9BhUykBcr3N6lnjGszoZdAMM0AtYYyxwrS6dIn86WEmgcn7WT8jlo+Hl+LBdGYw4N00zKPiUA05XO6Qj8NlFKdXUGJbYQsLbLwi1uk0 Ere2gzTSjfVqvQnKFolYd8P7JbiTmuoSj9mlestWiPkucbyXGF53+QXvd+Q+cinMWOmQ4r/V9z50GjmoNKUss0ACnz+KsUeN2UTPa9spaytfVmrBtGIb733ytTAqV/e2WRG1k= user

3.2.3 Building Docker Image



```
user@mohit-pc:~/Desktop/MajorProject$ sudo docker build -t mohitsoni98/k3s .
[sudo] password for user:
Sending build context to Docker daemon  2.194MB
Step 1/7 : FROM node:lts-alpine3.12
lts-alpine3.12: Pulling from library/node
f84cab65f19f: Pull complete
c1498f218859: Pull complete
e911cafd0b8d: Pull complete
3d11014629f2: Pull complete
Digest: sha256:aed8ac10548375da1c15e817b6d5d03e67d84cac5dfef922a76a7675d5c9642a
Status: Downloaded newer image for node:lts-alpine3.12
 ---> 8f86419010df
Step 2/7 : WORKDIR /app
 ---> Running in a2707c6e9e12
Removing intermediate container a2707c6e9e12
 ---> 62c40433d724
Step 3/7 : COPY ./project/package.json ./project/package-lock.json ./
 ---> 77323e3aa4b9
Step 4/7 : RUN npm i --production
 ---> Running in 9c5728a2ce7e
npm WARN project@1.0.0 No description
npm WARN project@1.0.0 No repository field.

added 50 packages from 37 contributors and audited 50 packages in 1.651s
found 0 vulnerabilities

Removing intermediate container 9c5728a2ce7e
 ---> 44f5d34c0ced
Step 5/7 : COPY . .
 ---> b9a8d50cee0a
Step 6/7 : EXPOSE 3000
 ---> Running in 9ad1f23793a5
Removing intermediate container 9ad1f23793a5
 ---> 38de0966e44c
Step 7/7 : CMD npm start
 ---> Running in e92898b8432c
Removing intermediate container e92898b8432c
 ---> 3dcbd0102a95
Successfully built 3dcbd0102a95
Successfully tagged mohitsoni98/k3s:latest
user@mohit-pc:~/Desktop/MajorProject$
```

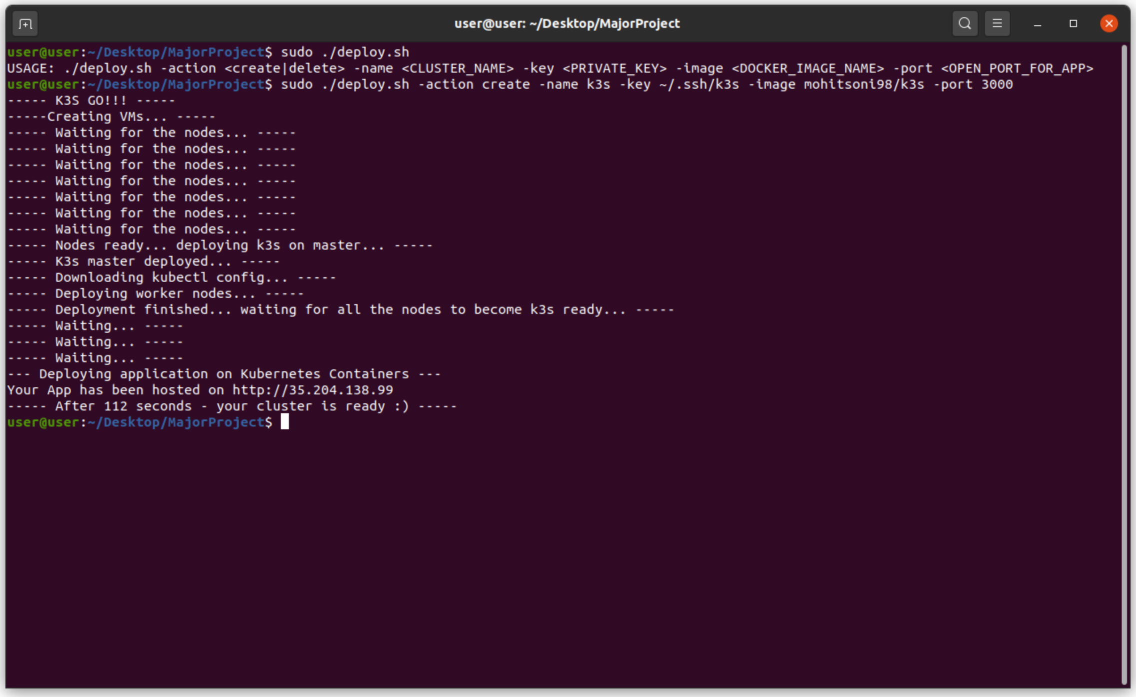
3.2.4 Pushing Image to Docker Hub



user@mohit-pc: ~/Desktop/MajorProject

```
user@mohit-pc:~/Desktop/MajorProject$ sudo docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
user@mohit-pc:~/Desktop/MajorProject$ sudo docker push mohitsoni98/k3s
Using default tag: latest
The push refers to repository [docker.io/mohitsoni98/k3s]
0e5d22a6c781: Pushed

7ccddd6869b4: Pushed

09e175051936: Pushed

05af6789dfdf: Pushed

a8e8c71490ec: Mounted from library/node

ea79b9f37866: Mounted from library/node

62a7f5bbacbd: Mounted from library/node

33e8713114f8: Mounted from library/node

latest: digest: sha256:4b76f700aab754007dcb4a39279cadbd5d1b8ecd6817eba66cabfdae6eb3606e size: 1994
user@mohit-pc:~/Desktop/MajorProject$
```

3.2.5 Running the script

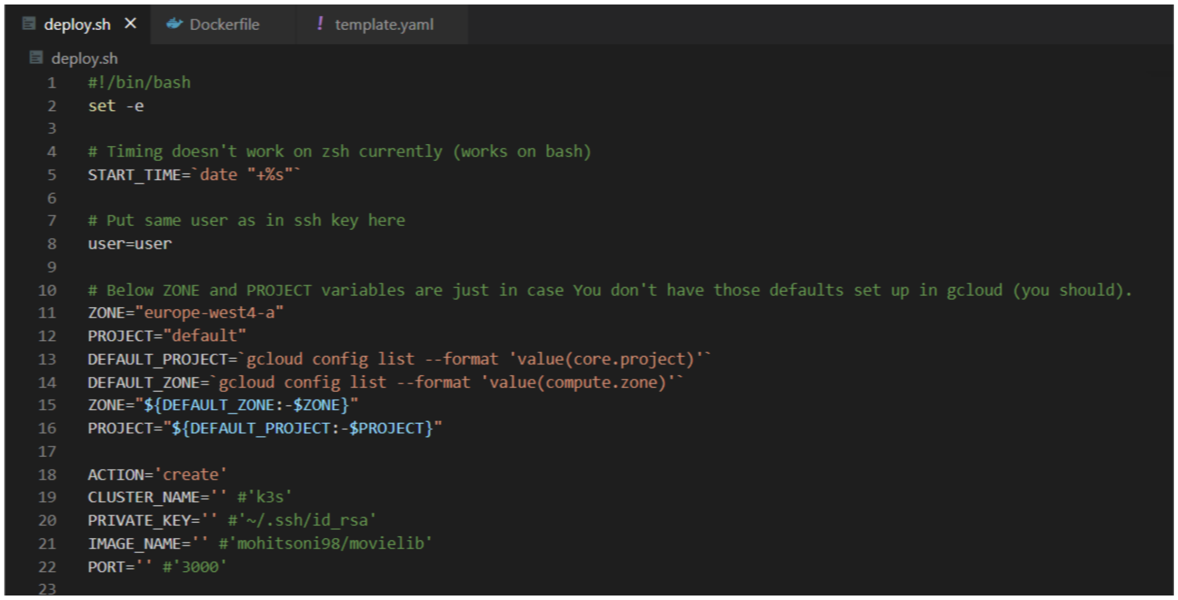


user@user: ~/Desktop/MajorProject

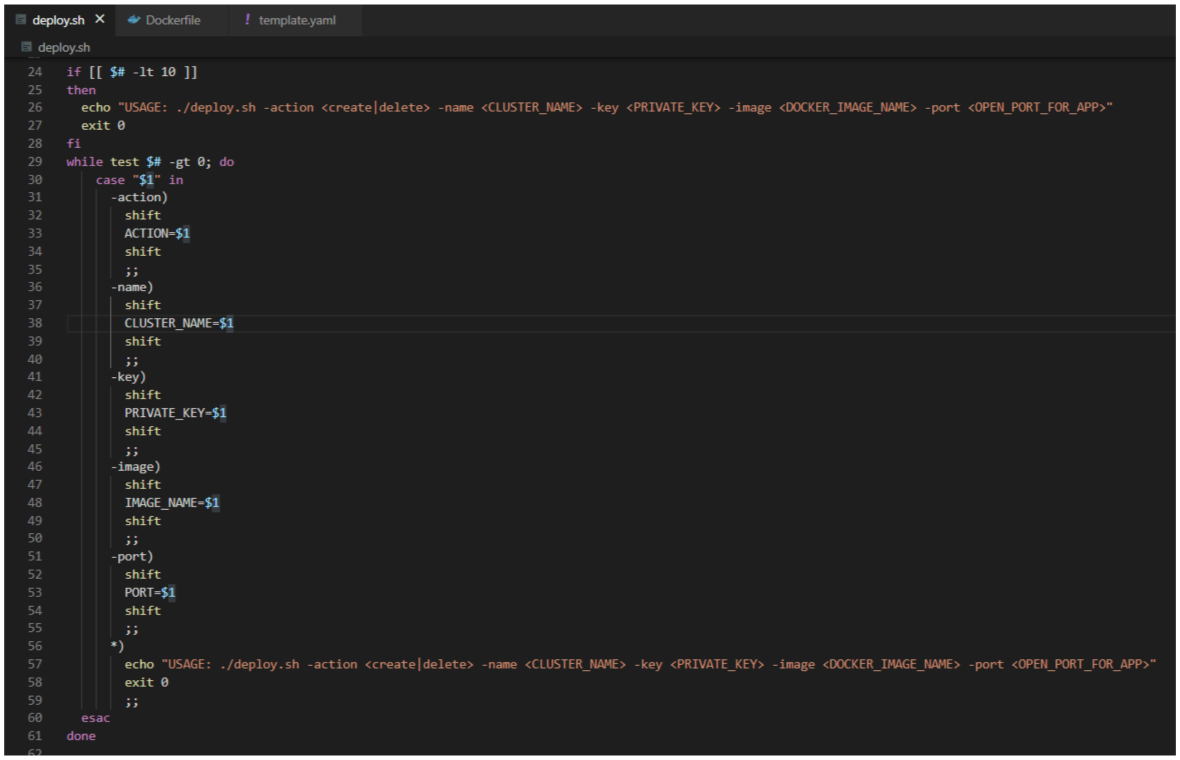
```
user@user:~/Desktop/MajorProject$ sudo ./deploy.sh
USAGE: ./deploy.sh -action <create|delete> -name <CLUSTER_NAME> -key <PRIVATE_KEY> -image <DOCKER_IMAGE_NAME> -port <OPEN_PORT_FOR_APP>
user@user:~/Desktop/MajorProject$ sudo ./deploy.sh -action create -name k3s -key ~/.ssh/k3s -image mohitsoni98/k3s -port 3000
----- K3S GO!!! -----
-----Creating VMs... -----
----- Waiting for the nodes... -----
----- Waiting for the nodes... -----
----- Waiting for the nodes... -----
----- Waiting for the nodes... -----
----- Waiting for the nodes... -----
----- Waiting for the nodes... -----
----- Waiting for the nodes... -----
----- Nodes ready... deploying k3s on master... -----
----- K3s master deployed... -----
----- Downloading kubectl config... -----
----- Deploying worker nodes... -----
----- Deployment finished... waiting for all the nodes to become k3s ready... -----
----- Waiting... -----
----- Waiting... -----
----- Waiting... -----
--- Deploying application on Kubernetes Containers ---
Your App has been hosted on http://35.204.138.99
----- After 112 seconds - your cluster is ready :) -----
user@user:~/Desktop/MajorProject$
```

3.3 Source Code:

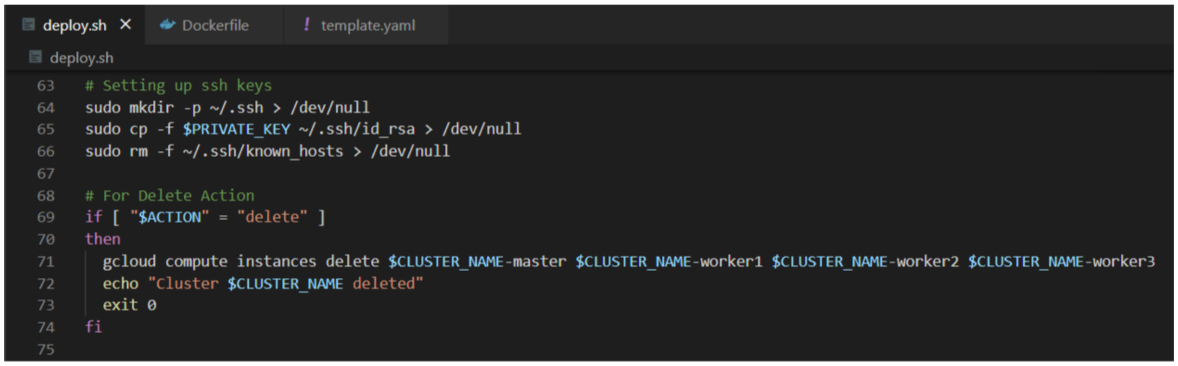
# deploy.sh



```bash
#!/bin/bash
set -e

# Timing doesn't work on zsh currently (works on bash)
START_TIME=`date "+%s"`

# Put same user as in ssh key here
user=user

# Below ZONE and PROJECT variables are just in case You don't have those defaults set up in gcloud (you should).
ZONE="europe-west4-a"
PROJECT="default"
DEFAULT_PROJECT=`gcloud config list --format 'value(core.project)'`
DEFAULT_ZONE=`gcloud config list --format 'value(compute.zone)'`
ZONE="${DEFAULT_ZONE:-$ZONE}"
PROJECT="${DEFAULT_PROJECT:-$PROJECT}"

ACTION='create'
CLUSTER_NAME='' #'k3s'
PRIVATE_KEY='' #'~/.ssh/id_rsa'
IMAGE_NAME='' #'mohitsoni98/movielib'
PORT='' #'3000'

```

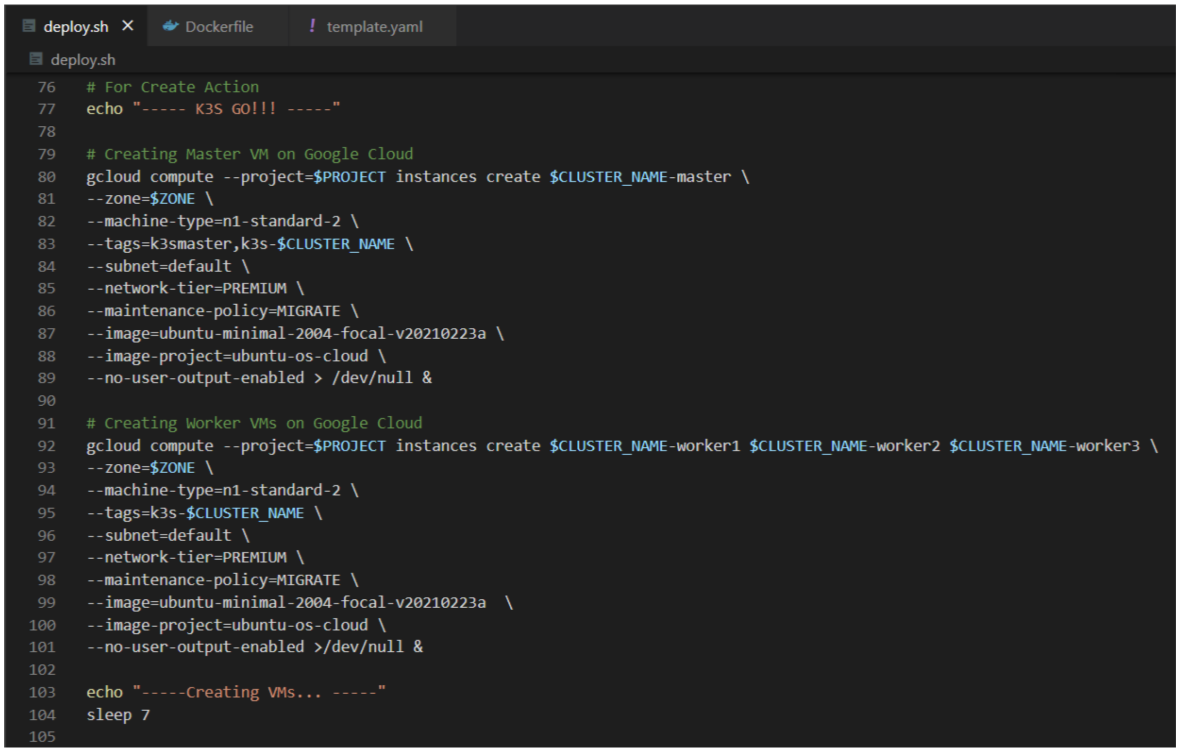


```bash
if [[ $# -lt 10 ]]
then
  echo "USAGE: ./deploy.sh -action <create|delete> -name <CLUSTER_NAME> -key <PRIVATE_KEY> -image <DOCKER_IMAGE_NAME> -port <OPEN_PORT_FOR_APP>"
  exit 0
fi
while test $# -gt 0; do
    case "$1" in
      -action)
        shift
        ACTION=$1
        shift
        ;;
      -name)
        shift
        CLUSTER_NAME=$1
        shift
        ;;
      -key)
        shift
        PRIVATE_KEY=$1
        shift
        ;;
      -image)
        shift
        IMAGE_NAME=$1
        shift
        ;;
      -port)
        shift
        PORT=$1
        shift
        ;;
      *)
        echo "USAGE: ./deploy.sh -action <create|delete> -name <CLUSTER_NAME> -key <PRIVATE_KEY> -image <DOCKER_IMAGE_NAME> -port <OPEN_PORT_FOR_APP>"
        exit 0
        ;;
  esac
done

```


deploy.sh × Dockerfile template.yaml

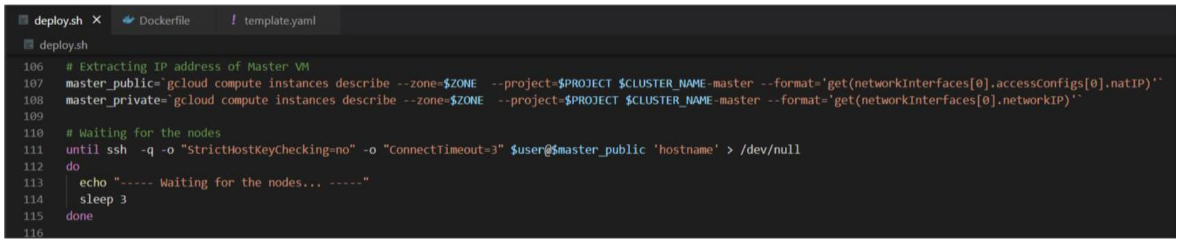
deploy.sh

```
# Setting up ssh keys
sudo mkdir -p ~/.ssh > /dev/null
sudo cp -f $PRIVATE_KEY ~/.ssh/id_rsa > /dev/null
sudo rm -f ~/.ssh/known_hosts > /dev/null

# For Delete Action
if [ "$ACTION" = "delete" ]
then
  gcloud compute instances delete $CLUSTER_NAME-master $CLUSTER_NAME-worker1 $CLUSTER_NAME-worker2 $CLUSTER_NAME-worker3
  echo "Cluster $CLUSTER_NAME deleted"
  exit 0
fi

```


deploy.sh × Dockerfile template.yaml

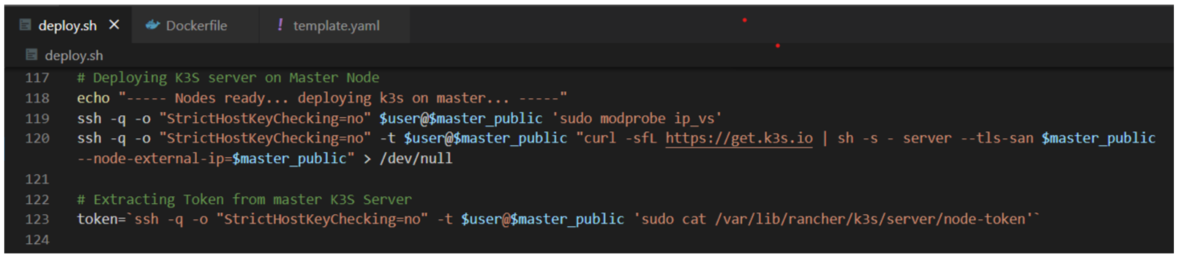
deploy.sh

```
# For Create Action
echo "----- K3S GO!!! -----"

# Creating Master VM on Google Cloud
gcloud compute --project=$PROJECT instances create $CLUSTER_NAME-master \
--zone=$ZONE \
--machine-type=n1-standard-2 \
--tags=k3smaster,k3s-$CLUSTER_NAME \
--subnet=default \
--network-tier=PREMIUM \
--maintenance-policy=MIGRATE \
--image=ubuntu-minimal-2004-focal-v20210223a \
--image-project=ubuntu-os-cloud \
--no-user-output-enabled > /dev/null &

# Creating Worker VMs on Google Cloud
gcloud compute --project=$PROJECT instances create $CLUSTER_NAME-worker1 $CLUSTER_NAME-worker2 $CLUSTER_NAME-worker3 \
--zone=$ZONE \
--machine-type=n1-standard-2 \
--tags=k3s-$CLUSTER_NAME \
--subnet=default \
--network-tier=PREMIUM \
--maintenance-policy=MIGRATE \
--image=ubuntu-minimal-2004-focal-v20210223a  \
--image-project=ubuntu-os-cloud \
--no-user-output-enabled >/dev/null &

echo "-----Creating VMs... -----"
sleep 7

```


deploy.sh × Dockerfile template.yaml

deploy.sh

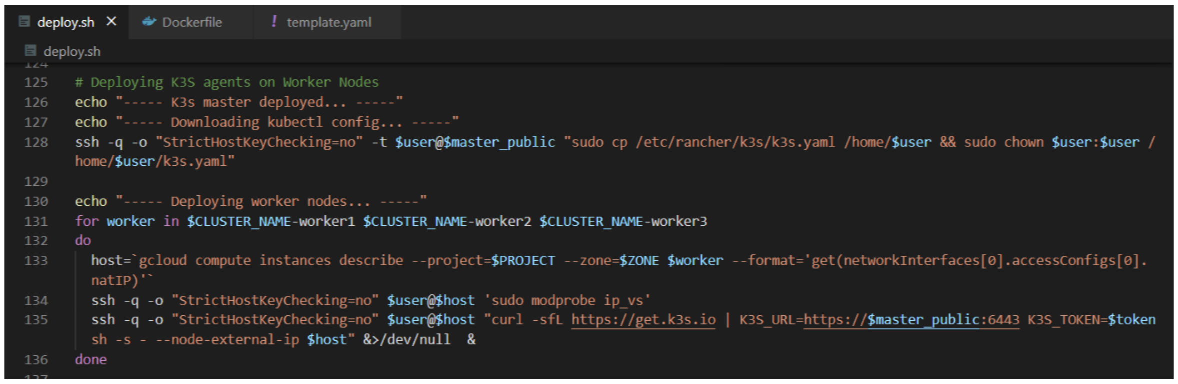
```
# Extracting IP address of Master VM
master_public=`gcloud compute instances describe --zone=$ZONE  --project=$PROJECT $CLUSTER_NAME-master --format='get(networkInterfaces[0].accessConfigs[0].natIP)'`
master_private=`gcloud compute instances describe --zone=$ZONE  --project=$PROJECT $CLUSTER_NAME-master --format='get(networkInterfaces[0].networkIP)'`

# Waiting for the nodes
until ssh  -q -o "StrictHostKeyChecking=no" -o "ConnectTimeout=3" $user@$master_public 'hostname' > /dev/null
do
  echo "----- Waiting for the nodes... -----"
  sleep 3
done

```



deploy.sh ✕ | Dockerfile | ! template.yaml

deploy.sh

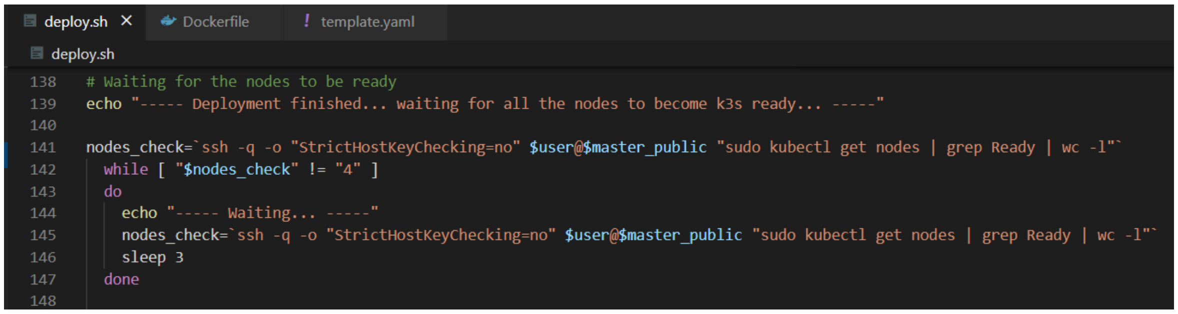
```bash
# Deploying K3S server on Master Node
echo "----- Nodes ready... deploying k3s on master... -----"
ssh -q -o "StrictHostKeyChecking=no" $user@$master_public 'sudo modprobe ip_vs'
ssh -q -o "StrictHostKeyChecking=no" -t $user@$master_public "curl -sfL https://get.k3s.io | sh -s - server --tls-san $master_public
--node-external-ip=$master_public" > /dev/null

# Extracting Token from master K3S Server
token=`ssh -q -o "StrictHostKeyChecking=no" -t $user@$master_public 'sudo cat /var/lib/rancher/k3s/server/node-token'`

```



deploy.sh ✕ | Dockerfile | ! template.yaml

deploy.sh

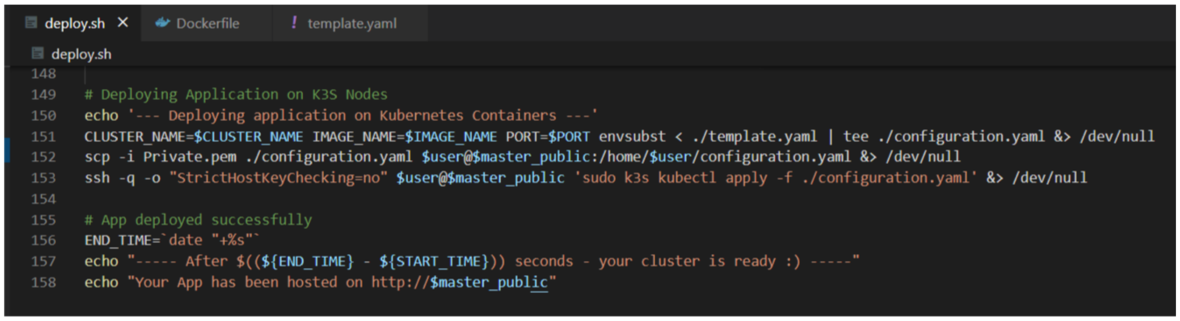
```bash
# Deploying K3S agents on Worker Nodes
echo "----- K3s master deployed... -----"
echo "----- Downloading kubectl config... -----"
ssh -q -o "StrictHostKeyChecking=no" -t $user@$master_public "sudo cp /etc/rancher/k3s/k3s.yaml /home/$user && sudo chown $user:$user /
home/$user/k3s.yaml"

echo "----- Deploying worker nodes... -----"
for worker in $CLUSTER_NAME-worker1 $CLUSTER_NAME-worker2 $CLUSTER_NAME-worker3
do
  host=`gcloud compute instances describe --project=$PROJECT --zone=$ZONE $worker --format='get(networkInterfaces[0].accessConfigs[0].
  natIP)'`
  ssh -q -o "StrictHostKeyChecking=no" $user@$host 'sudo modprobe ip_vs'
  ssh -q -o "StrictHostKeyChecking=no" $user@$host "curl -sfL https://get.k3s.io | K3S_URL=https://$master_public:6443 K3S_TOKEN=$token
  sh -s - --node-external-ip $host" &>/dev/null  &
done
```



deploy.sh ✕ | Dockerfile | ! template.yaml

deploy.sh

```bash
# Waiting for the nodes to be ready
echo "----- Deployment finished... waiting for all the nodes to become k3s ready... -----"

nodes_check=`ssh -q -o "StrictHostKeyChecking=no" $user@$master_public "sudo kubectl get nodes | grep Ready | wc -l"`
while [ "$nodes_check" != "4" ]
do
  echo "----- Waiting... -----"
  nodes_check=`ssh -q -o "StrictHostKeyChecking=no" $user@$master_public "sudo kubectl get nodes | grep Ready | wc -l"`
  sleep 3
done

```

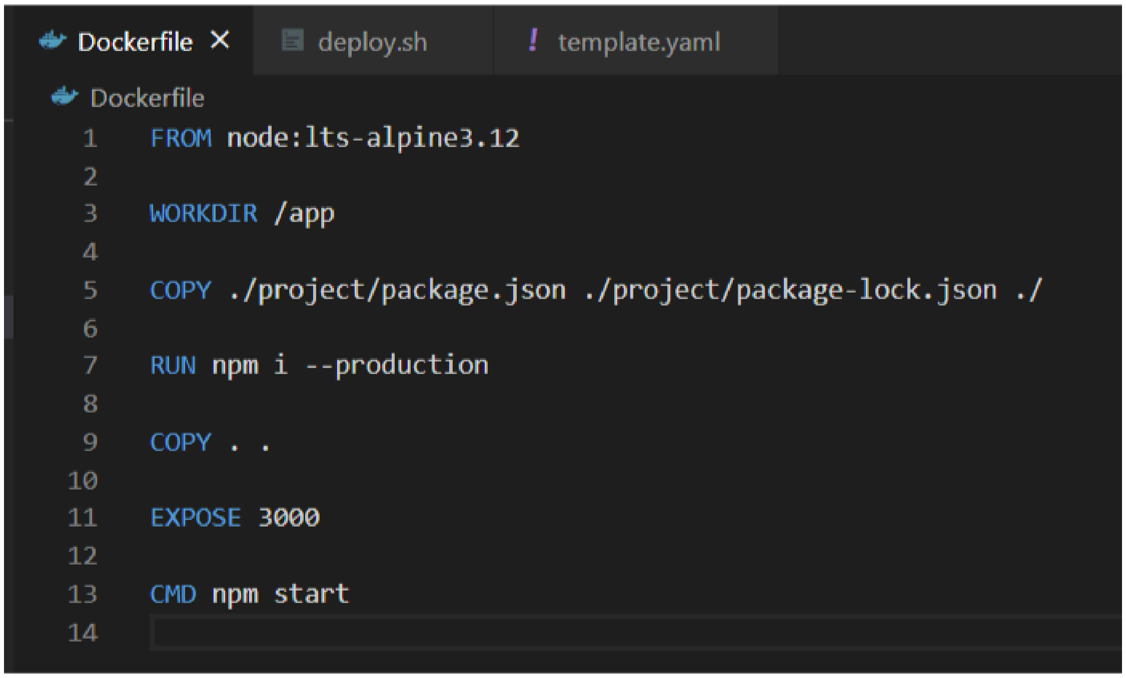


deploy.sh ✕ | Dockerfile | ! template.yaml

deploy.sh

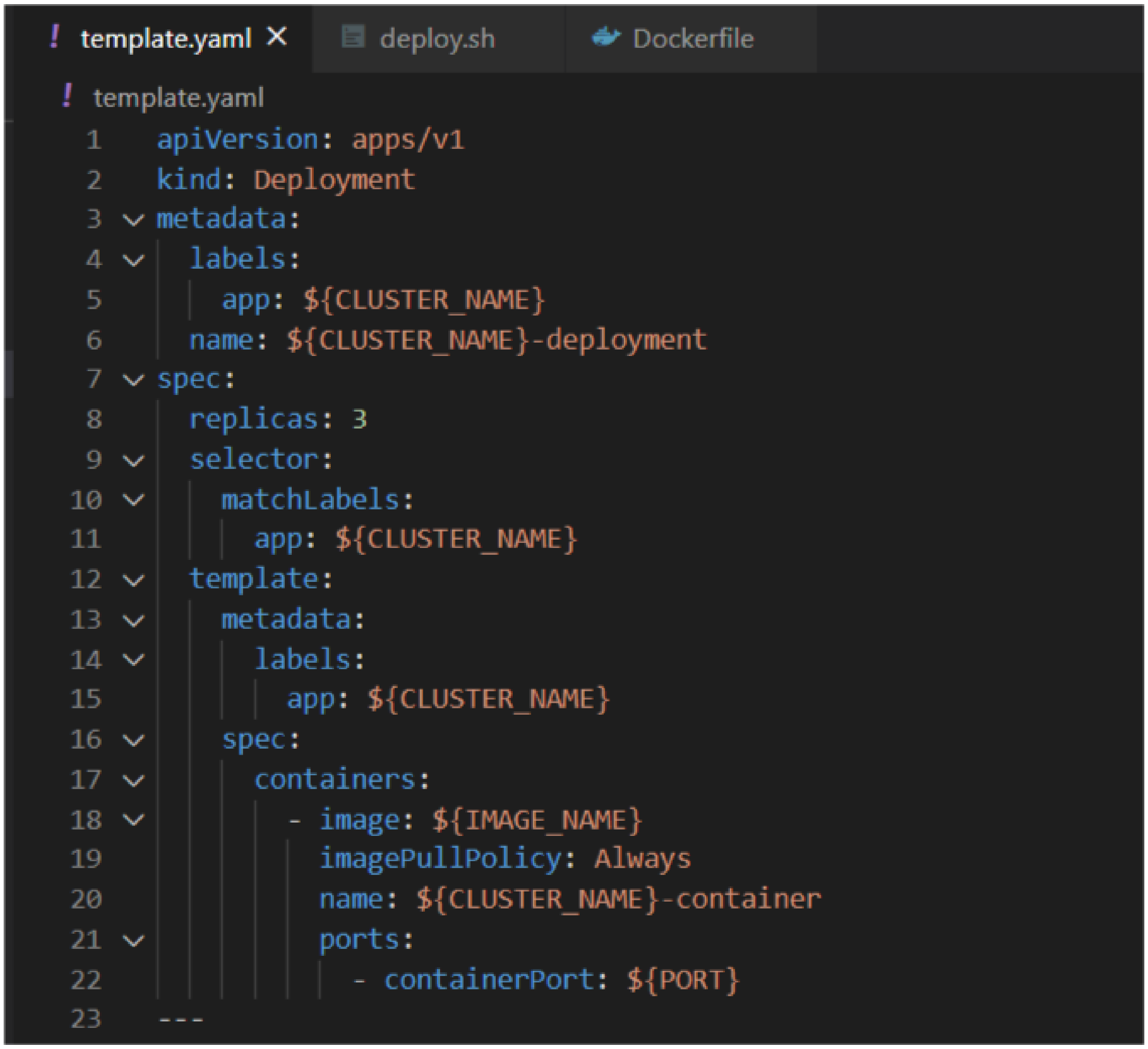
```bash

# Deploying Application on K3S Nodes
echo '--- Deploying application on Kubernetes Containers ---'
CLUSTER_NAME=$CLUSTER_NAME IMAGE_NAME=$IMAGE_NAME PORT=$PORT envsubst < ./template.yaml | tee ./configuration.yaml &> /dev/null
scp -i Private.pem ./configuration.yaml $user@$master_public:/home/$user/configuration.yaml &> /dev/null
ssh -q -o "StrictHostKeyChecking=no" $user@$master_public 'sudo k3s kubectl apply -f ./configuration.yaml' &> /dev/null

# App deployed successfully
END_TIME=`date "+%s"`
echo "----- After $((${END_TIME} - ${START_TIME})) seconds - your cluster is ready :) -----"
echo "Your App has been hosted on http://$master_public"
```

# DockerFile

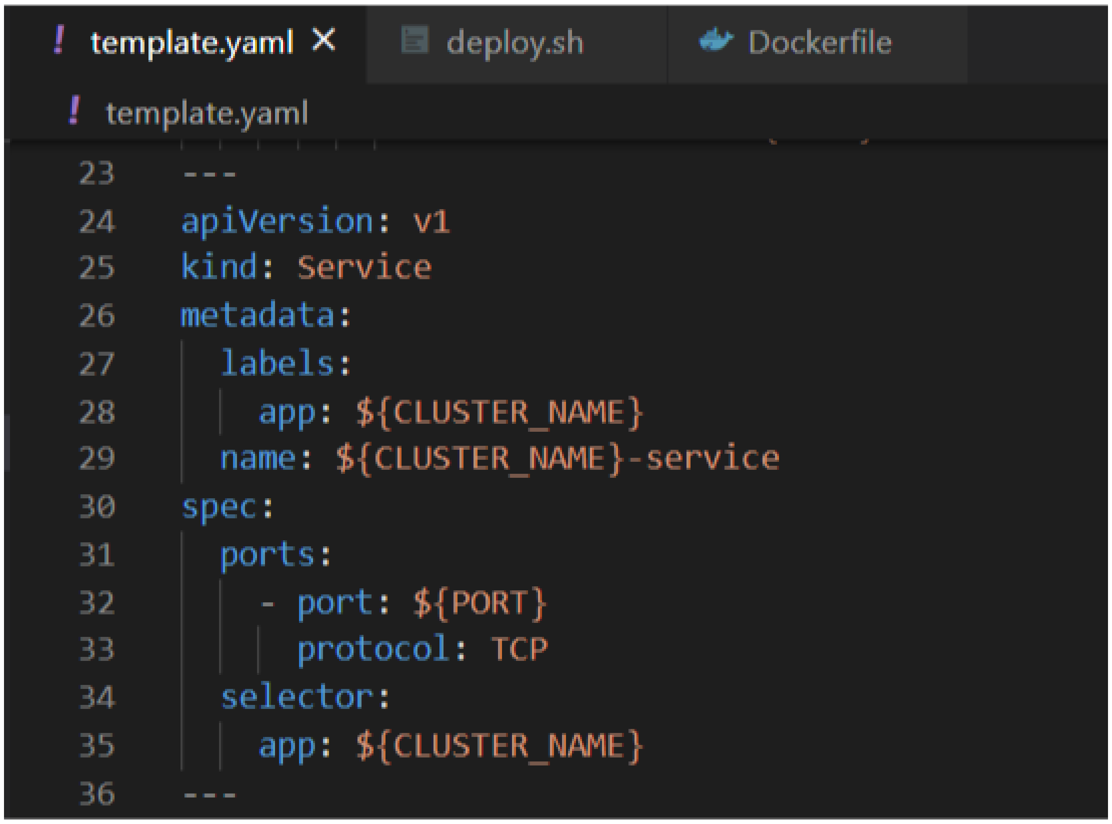


```
FROM node:lts-alpine3.12

WORKDIR /app

COPY ./project/package.json ./project/package-lock.json ./

RUN npm i --production

COPY . .

EXPOSE 3000

CMD npm start
```

# template.yaml



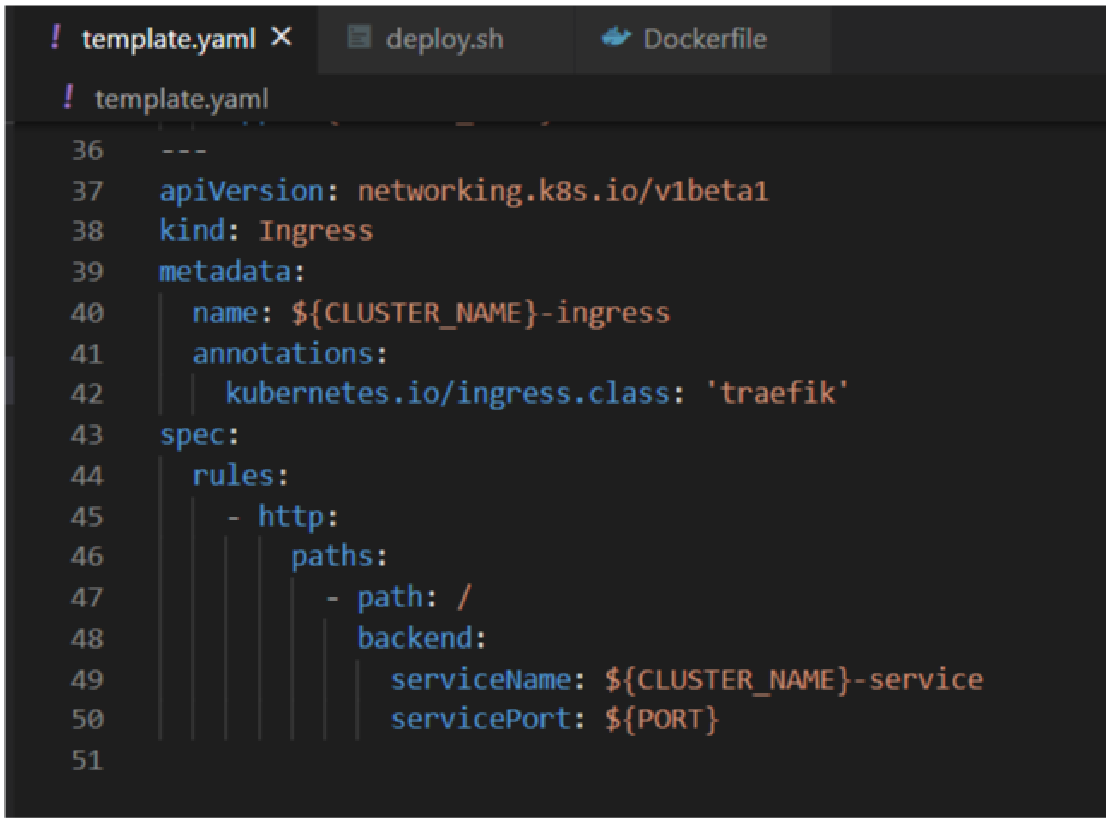
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: ${CLUSTER_NAME}
  name: ${CLUSTER_NAME}-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: ${CLUSTER_NAME}
  template:
    metadata:
      labels:
        app: ${CLUSTER_NAME}
    spec:
      containers:
        - image: ${IMAGE_NAME}
          imagePullPolicy: Always
          name: ${CLUSTER_NAME}-container
          ports:
            - containerPort: ${PORT}
---
```



template.yaml × deploy.sh Dockerfile

template.yaml

```yaml
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: ${CLUSTER_NAME}
  name: ${CLUSTER_NAME}-service
spec:
  ports:
    - port: ${PORT}
      protocol: TCP
  selector:
    app: ${CLUSTER_NAME}
---
```



```yaml
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ${CLUSTER_NAME}-ingress
  annotations:
    kubernetes.io/ingress.class: 'traefik'
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: ${CLUSTER_NAME}-service
          servicePort: ${PORT}

```

3.4 Conclusion:

As an undergraduate student I would like to say that Major Project is an excellent opportunity for us to get to the ground level and experience the things that we would have never gained through going into a job straightly. Major Project was the first opportunity I got to apply the theories I learnt with the real industry for the real situations. This also gave me the chance to move with different types of people in the industry. Having exposed to such situations I was able to obtain lot of experiences which will be definitely helpful to success my future career as an Engineer.

Finally, I can say with a great pleasure that the 8 weeks of Major Project was a helpful period for me as an engineering undergraduate. I could meet my expectations of Major Project during the period I underwent training. The experiences I gained through this Major Project will be a strong foundation to my career as an engineer. I hope that the university would continue to conduct such Major Project for undergraduates even more effectively and efficiently.

REFERENCES

- https://docs.docker.com/
- https://en.wikipedia.org/wiki/Google_Cloud_Platform
- https://medium.com/sysf/bash-scripting-everything-you-need-to-know-about-bash-shell-programming-cd08595f2fba
- https://github.com/k3s-io/k3s
- *https://kubernetes.io/docs/home/*