**Learn to create apps on Kubernetes**

**A Project Work Synopsis**

*Submitted in the partial fulfilment for the award of the degree of*

**MASTERS OF ENGINEERING**

**IN**

**COMPUTER SCIENCE WITH SPECIALIZATION IN**

**CLOUD COMPUTING IN ASSOCIATION WITH VIRTUSA**

**Submitted by:**

22MCC10005 SYED IMRAN

22MCC10003 MOHD AMAAN KHAN

**Under the Supervision of:**

**<MR Digvijay Puri Sir>**



**CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413,**

**PUNJAB**

**March, 2023**

# Abstract

For managing containerized workloads and services, Kubernetes is a portable, extensible, open-source platform that supports declarative configuration and automation. It has a huge, expanding ecosystem. Services, assistance, and tools for Kubernetes are widely accessible. Pods are the fundamental Kubernetes units that are deployed in the cluster. The deployment of Kubernetes acts as an abstraction layer for the pods. Maintaining the resources stated in the deployment configuration in the appropriate condition is the primary function of the deployment object. The format of a deployment configuration can be either YAML or JSON. Technology for containerization is incredibly quick, dependable, effective, lightweight, and scalable. However, there are several issues with how its scalability is managed. In the event that containers are not manageable when they are scaled up, it is a very difficult task for servers to handle. The issue is not with growing the containers; rather, it is with how they would be managed and interacted with. Hence, we may say that standard containers are insufficient to provide additional scaling flexibility in specifics. Kubernetes is a different container management system. Users expect programs to be accessible 24 hours a day with current web services, and developers anticipate releasing updated versions of such applications frequently. Applications may be launched and upgraded quickly and easily without experiencing any downtime thanks to containerization, which enables package software to achieve these objectives. These containerized programs can be made to run where and when you want them to, and Kubernetes makes it easier for them to locate the tools and resources they require. With Google's extensive knowledge of container orchestration, Kubernetes is an open-source platform that is ready for production. A cluster of physical or virtual machines can be abstracted using Kubernetes orchestration. The Kubernetes API is used to access this abstraction layer, which is also known as the Kubernetes cluster. In order to run containers at scale, many tasks can now be automated. It makes it easier to manage resources associated with applications, such as configuration files, storage, and credentials. Kubernetes is crucial as businesses migrate to a microservices-based design. The intricacy associated with connecting these separate bits of functionality can be managed with Kubernetes.

Keywords: Kubernetes, Amazon Web Services (AWS), Amazon Elastic Kubernetes Service (EKS), operations automation, Performance Applications, Orchestration

# Table of Contents

|  |  |
| --- | --- |
| Title Page | i |
| Abstract | ii |
| 1.    Introduction |  |
| 1.1           Problem Definition |  |
| 1.2 Project Overview |  |
| 1.3 Hardware Specification |  |
| 1.4 Software Specification |  |
| 2.    Literature Survey |  |
| 2.1 Existing System |  |
| 2.2 Proposed System |  |
| 2.3 Literature Review Summary |  |
| 3.    Problem Formulation |  |
| 4.    Research Objective |  |
| 5.    Methodologies |  |
| 6.    Experimental Setup |  |
| 7.    Conclusion |  |
| 8.    Tentative Chapter Plan for the proposed work |  |
| 9.    Reference |  |

# INTRODUCTION

An open source container orchestration technology called Kubernetes aids in the large-scale management of dispersed, containerized applications. Kubernetes asks you where you want your software to run, and it essentially handles the rest. the history, purposes, and advantages of Kubernetes. the fundamentals of container management and orchestration, as well as modern application development. examine the fundamental Kubernetes architecture. Things to think about before implementing Kubernetes. Groups of hosts running Linux containers can be grouped together, and Kubernetes makes managing these clusters simple and effective. Hosts in on-premises, public, private, or hybrid clouds can be included in Kubernetes clusters. The platform it provides you with allows you to schedule and run containers on clusters of physical or virtual machines, which is very useful if you are optimizing app development for the cloud (VMs). More generally, it enables you to completely deploy and depend on a container-based architecture in production settings. You can perform many of the same operations that other application platforms or management systems permit you to perform, but for your containers, because Kubernetes is primarily about automating operational tasks. By utilizing Kubernetes patterns, developers can use Kubernetes as a runtime platform to build cloud-native apps. The tools a Kubernetes developer needs to create container-based services and applications are patterns. A functioning Kubernetes deployment is referred to as a cluster with Kubernetes. The cluster's desired state, including the applications that are active and the container images they utilise, must be maintained by the control plane. Applications and workloads are really run on compute machines. Red Hat Enterprise Linux, for instance, is the operating system that Kubernetes runs on top of. It communicates with pods of running containers on the nodes. The Kubernetes control plane delivers the commands to the compute machines after receiving them from an administrator (or DevOps team). This handoff uses a variety of services to determine which node is most appropriate for the task automatically. The desired task is subsequently assigned to the node's pods when resources have been allocated. Linux container servers can be grouped together into clusters, and Kubernetes makes managing these clusters simple and effective. On-premises, public, private, or hybrid clouds can all house hosts in a Kubernetes cluster. Kubernetes is the perfect hosting platform for cloud-native apps that need quick scaling, such as real-time data streaming through Apache Kafka. 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. 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, Haw Kular, and Elastic. Security, through projects like LDAP, Selina, RBAC, and OAUTH with multitenancy layers. Automation, with the addition of Ansible playbooks for installation and cluster life cycle management. Services, through a rich catalo 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. 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. However, 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. Kebele This service runs on nodes, reads the container manifests, and ensures the defined containers are started and running. Kubit The command line configuration tool for Kubernetes. 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 fulfil 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 kinds of infrastructure. Docker can be used as a container runtime that Kubernetes orchestrates. When Kubernetes schedules a pod to a node, the kubelet on that node will instruct Docker to launch the specified containers. The kubelet then continuously collects the status of those containers from Docker and aggregates that information in the control plane. Docker pulls containers onto that node and starts and stops those containers. The difference when using Kubernetes with Docker is that an automated system asks Docker to do those things instead of the admin doing so manually on all nodes for all containers. Today, the majority of on-premises Kubernetes deployments run on top of existing virtual infrastructure, with a growing number of deployments on bare metal servers. This is a natural evolution in data centres. Kubernetes serves as the deployment and lifecycle management tool for containerized applications, and separate tools are used to manage infrastructure resources. But what if you designed the datacentre from scratch to support containers, including the infrastructure layer? You would start directly with bare metal servers and software-defined storage, deployed and managed by Kubernetes to give the infrastructure the same self-installing, self-scaling, and self-healing benefits as containers enjoy. This is the vision of Kubernetes-native infrastructure. Developing modern applications requires different processes than the approaches of the past. DevOps speeds up how an idea goes from development to deployment. At its core, DevOps relies on automating routine operational tasks and standardizing environments across an app’s lifecycle. Containers support a unified environment for development, delivery, and automation, and make it easier to move apps between development, testing, and production environments. A major outcome of implementing DevOps is a continuous integration and continuous deployment pipeline (CI/CD). CI/CD helps you deliver apps to customers frequently and validate software quality with minimal human intervention. Managing the lifecycle of containers with Kubernetes alongside a DevOps approach helps to align software development and IT operations to support a CI/CD pipeline. With the right platforms, both inside and outside the container, you can best take advantage of the culture and process changes you’ve implemented. Kubernetes was designed to support the features required by highly available distributed systems, such as (auto-)scaling, high availability, security and portability. Scalability – Kubernetes provides horizontal scaling of pods on the basis of CPU utilization. The threshold for CPU usage is configurable and Kubernetes will automatically start new pods if the threshold is reached. For example, if the threshold is 70% for CPU but the application is actually growing up to 220%, then eventually 3 more pods will be deployed so that the average CPU utilization is back under 70%. When there are multiple pods for a particular application, Kubernetes provides the load balancing capacity across them. Kubernetes also supports horizontal scaling of stateful pods, including NoSQL and RDBMS databases through Stateful sets. A Stateful set is a similar concept to a Deployment, but ensures storage is persistent and stable, even when a pod is removed. High Availability – Kubernetes addresses highly availability both at application and infrastructure level. Replica sets ensure that the desired (minimum) number of replicas of a stateless pod for a given application are running. Stateful sets perform the same role for stateful pods. At the infrastructure level, Kubernetes supports various distributed storage backends like AWS EBS, Azure Disk, Google Persistent Disk, NFS, and more. Adding a reliable, available storage layer to Kubernetes ensures high availability of stateful workloads. Also, each of the master components can be configured for multi-node replication (multi-master) to ensure higher availability. Security – Kubernetes addresses security at multiple levels: cluster, application and network. The API endpoints are secured through transport layer security (TLS). Only authenticated users (either service accounts or regular users) can execute operations on the cluster (via API requests). At the application level, Kubernetes secrets can store sensitive information (such as passwords or tokens) per cluster (a virtual cluster if using namespaces, physical otherwise). Note that secrets are accessible from any pod in the same cluster. Network policies for access to pods can be defined in a deployment. A network policy specifies how pods are allowed to communicate with each other and with other network endpoints. Portability – Kubernetes portability manifests in terms of operating system choices (a cluster can run on any mainstream Linux distribution), processor architectures (either virtual machines or bare metal), cloud providers (AWS, Azure or Google Cloud Platform), and new container runtimes, besides Docker, can also be added. Through the concept of federation, it can also support workloads across hybrid (private and public cloud) or multi-cloud environments. This also supports availability zone fault tolerance within a single cloud provider. VMware has extended support to Kubernetes since 2019. This means you can use the virtualization platform vSphere, including the ESXi hypervisor, to run containers. To manage your standard clusters, you can also use Tanzu, which helps ensure compatibility with your Kubernetes development later on. The Tanzu Kubernetes Grid is a platform that lets you run Kubernetes in a production environment, manage multiple Kubernetes clusters throughout your onsite servers, public cloud deployments, and VMware infrastructure. The Tanzu stack is designed to facilitate simpler operations and development. Here are several of its key features: Faster release cycles—Tanzu offers a collection of container images built and maintained by administrators that enables organizations. The use of these images can help teams increase the velocity of application development and delivery. Full-stack observability—Tanzu provides a single view for all stakeholders responsible for monitoring and analysing cluster infrastructure and application metrics. Compliant with various runtimes—Tanzu supports OCI-compliant and CRI-compliant runtimes to allow teams to leverage containers created using any runtime engine.  Ephemeral and persistent storage—Tanzu uses vSphere to manage storage. vSphere comes with a CNS-CSI driver that enables it to support any Kubernetes storage solution that follows persistent and ephemeral storage. Full-stack networking capabilities—Tanzu employs VMWare’s NSX Container Networking Solution to provide full-stack networking capabilities. Teams can leverage this feature to implement Kubernetes-native networking solutions. Several software vendors have created their own distributions of the original Kubernetes project. Each vendor offers their Kubernetes distribution packaged with other technologies that provide added value. Here are a few popular Kubernetes distributions. OpenShift Container Platform OpenShift Container Platform is a hybrid cloud platform designed by Red Hat to help organizations build and scale containerized applications. It is built on top of Kubernetes and utilizes several additional technologies, including Docker-style Linux containers and Red Hat Enterprise Linux (RHEL). Here are several key technologies employed by OpenShift Container Platform: OKD—a community distribution of Kubernetes that powers OpenShift. It is built around OCI container packaging and Kubernetes clusters. Red Hat OpenShift Container Storage—provides available and dynamic persistent storage for containerize-based applications. Software Defined Networking (SDN)—offers plugins you can use to configure overlay networks for your Kubernetes clusters. Rancher is an open source platform that lets you run containers in production across several environments, including public clouds and on-premises infrastructure. Rancher does that by capturing computing resources from private or public clouds and then seamlessly deploying Kubernetes resources on the captured computing resources. Notable features include container load balancing, cross-host networks, persistent storage services, user management, multi-tenancy, built-in security for Kubernetes clusters, and multi-cloud management. Merantis is a cloud computing company that offers various services, including a Kubernetes distributor for Red Hat OpenStack Platform. OpenStack is an open source platform commonly used to host Infrastructure as a Service (IaaS) operations on physical or virtual machines (VMs). The Merantis Kubernetes Engine is delivered as a stack consisting of custom databases, staging components, orchestration functionality, and message queueing. It enables unified cluster operations for multi-cloud applications, helping organizations reduce complexities in infrastructure and operations. DevOps teams can leverage the engine to build and ship code to public and private clouds quickly. Here are key features of Mirante’s Kubernetes Engine: Complies with OCI. Comes with built-in support for Dockers him, a Kubernetes component that allows it to run Docker containers. Offers Calico as a default CNI plugin to support highly scalable networks as well as multiple networking models. Relies mainly on software-defined storage. Offers Caph for object and block storage.

**Kubernetes Alternatives**

Kubernetes is a popular container orchestrator, but there are worthy alternatives. We’ll cover two common alternatives, Docker Swarm and Nomad.

In most cases, these Kubernetes alternatives are selected for smaller-scale use cases, where Kubernetes is considered “overkill” and overly complex. Docker Swarm is an orchestration platform commonly used as an alternative to Kubernetes. Swarm (also known as Swarm mode) is a native Docker orchestrating feature designed especially for Docker engine clusters. A Swarm cluster includes the following: Manager nodes—manages the cluster. Worker nodes receives instructions from manager nodes and then performs the requested tasks. All nodes must be deployed using Docker Engine. Both Swarm and Kubernetes enable you to automate application management and scaling, by partitioning your application into containers. However, there are key differences between the two, including: Kubernetes focuses on modular orchestration—it is ideal for demanding applications that require complex configurations. Docker Swarm focuses on ease of use—it is ideal for simple applications that do not require advanced workflow automation or complex resource provisioning. Nomad is a container orchestrator developed by Hashi Corp. It enables organizations to deploy and manage containers alongside legacy applications—using the same workflow for all components. The orchestrator is particularly designed for ease of use.  Nomad lets you use infrastructure as code (IaC) to declaratively deploy applications. It then runs workloads, such as Docker, microservices, batch applications, and non-container applications side by side. Notable features of Nomad include GPU support, device plugins, multi-region federation, and multi-cloud management. It also integrates with Hashi Corp products, including Consul, Vault, and Terraform.

#### **Docker**

Docker is not really a Kubernetes alternative, but newcomers to the space often ask what is the difference between them. The primary difference is that Docker is a container runtime, while Kubernetes is a platform for running and managing containers across multiple container runtimes. Docker is one of many container runtimes supported by Kubernetes. You can think of Kubernetes as an “operating system” and Docker containers as one type of application that can run on the operating system. Docker is hugely popular and was a major driver for the adoption of containerized architecture. Docker solved the classic “works on my computer” problem, and is extremely useful for developers, but is not sufficient to manage large-scale containerized applications. If you need to handle the deployment of a large number of containers, networking, security, and resource provisioning become important concerns. Standalone Docker was not designed to address these concerns, and this is where Kubernetes comes in. Kubernetes Security Kubernetes provides numerous security features and capabilities. However, out of the box, Kubernetes is not securely configured. Kubernetes applications also make extensive use of container images, which may contain security vulnerabilities. Due to its complexity and flexibility, it can be difficult to fully secure a Kubernetes cluster and maintain security over time. Let’s consider three perspectives of Kubernetes security: securing container images during the build process, securing network communication between containers in production environments, and securely configuring

**Kubernetes infrastructure.**

Securing Images in the Build Process In modern CI/CD development lifecycles, developers write code, which is pushed to a CI/CD pipeline and becomes part of a container image. Application security techniques are critical here—it is important to introduce testing as early as possible during the build process, so any code created or included by developers, or images pulled from repositories, are scanned for security vulnerabilities.  Keep in mind that container images are built of multiple layers, each of which may contain a large number of components. Automated container scanning technologies are essential to vet images, ensure they do not contain vulnerable components and have not been tampered with. This is the first step to ensuring a secure Kubernetes environment. Network Security in Production Networking structures within a Kubernetes cluster are another major attack surface. Most production deployments of Kubernetes use the container network interface (CNI) to create a secure networking layer, supporting network segmentation, with a private subnet for each Kubernetes namespace, and security policies you can use to restrict communication between containers. Secure Kubernetes Configuration Kubernetes is enormously flexible, but this makes it more difficult to achieve a secure configuration. It is important to understand that Kubernetes is not secure by default and that you must implement a list of secure configuration best practices to prevent exposure. Security researchers have published formal best practices for securing Kubernetes—possibly the most commonly used is the Kubernetes CIS benchmark. While the benchmark has over 250 pages of instructions for securing Kubernetes, you can use automated tools like kube-bench to scan your cluster according to the CIS benchmark and identify insecure configurations. A Kubernetes node is a worker machine that runs Kubernetes workloads. It can be a physical (bare metal) machine or a virtual machine (VM). Each node can host one or more pods. Kubernetes nodes are managed by a control plane, which automatically handles the deployment and scheduling of pods across nodes in a Kubernetes cluster. When scheduling pods, the control plane assesses the resources available on each node. Each node runs two main components—a kubelet and a container runtime. The kubelet is in charge of facilitating communication between the control plane and the node. The container runtime is in charge of pulling the relevant container image from a registry, unpacking containers, running them on the node, and communicating with the operating system kernel. The Kubelet is responsible for managing the deployment of pods to Kubernetes nodes. It receives commands from the API server and instructs the container runtime to start or stop containers as needed. A network proxy running on each Kubernetes node. It is responsible for maintaining network rules on each node. Network rules enable network communication between nodes and pods. Kube-proxy can directly forward traffic or use the operating system packet filter layer. The software layer responsible for running containers. There are several container runtimes supported by Kubernetes, including Containerd, CRI-O, Docker, and other Kubernetes Container Runtime Interface (CRI) implementations. You can manually add nodes to a Kubernetes cluster, or let the kubelet on that node self-register to the control plane. Once a node object is created manually or by the kubelet, the control plane validates the new node object The example below is a JSON manifest that creates a node object. Kubernetes then checks that the kubelet registered to the API server matches the node’s metadata.name field. Only healthy nodes running all necessary services are eligible to run the pod. If the check fails, the node is ignored until it becomes healthy. Nodes that self-register with the API Server report their CPU and memory volume capacity after the node object is created. However, When manually creating the node, administrators need to set up capacity demands. Once this information is defined, the Kubernetes scheduler assigns resources to all pods running on a node. The scheduler is responsible for ensuring that requests do not exceed node capacity. You can use kubectl to manually create or modify node objects, overriding the settings defined in --register-node. You can, for example: Use labels on nodes and node selectors to control scheduling. You can, for example, limit a pod to be eligible only for running on a subset of available nodes.Mark a node as unschedulable to prevent the scheduler from adding new pods to the node. This action does not affect pods running on the node. You can use this option in preparation for maintenance tasks like node reboot. To mark the node as unschedulable, you can run: kubectl cordon $NODENAME.There are three primary commands you can use to determine the status of a node and the resources running on it. kubectl describe nodes Run the command kubectl describe nodes my-node to get node information including: Host Name—reported by the node operating system kernel. You can report a different value for Host Name using the kubelet flag --hostname-override. InternalIP—enables traffic to be routed to the node within the cluster. ExternalID—an IP that can be used to access the node from outside the cluster. Conditions—system resource issues including CPU and memory utilization. This section shows error conditions like Out Of Disk, Memory Pressure, and Disk Pressure. Events—this section shows issues occurring in the environment, such as eviction of pods. kubectl describe pods You can use this command to get information about pods running on a node: Pod information—labels, resource requirements, and containers running in the pod Pod ready state—if a pod appears as READY, it means it passed the last readiness check. Container state—can be Waiting, Running, or Terminated. Restart count—how often a container has been restarted. Log events—showing activity on the pod, indicating which component logged the event, for which object, a Reason and a Message explaining what happened. Understanding the Node Controller The node controller is the control plane component responsible for managing several aspects of the node’s lifecycle. Here are the three main roles of the node controller: Assigning CIDR addresses When the node is registered, the node controller assigns a Cross Inter-Domain Routing (CIDR) block (if CIDR assignment is enabled).Updating internal node lists The node controller maintains an internal list of nodes. It needs to be updated constantly with the list of machines available by the cloud provider. This list enables the node controller to ensure capacity is met. When a node is unhealthy, the node controller checks if the host machine for that node is available. If the VM is not available, the node controller deletes the node from the internal list. If Kubernetes is running on a public or private cloud, the node controller can send a request to create a new node, to maintain cluster capacity. Monitoring the health of nodes Here are several tasks the node controller is responsible for: Checking the state of all nodes periodically, with the period determined by the --node-monitor-period flag. Updating the Node Ready condition to Condition Unknown if the node becomes unreachable and the node controller no longer receives heartbeats. Evicting all pods from the node. If the node remains unreachable, the node controller uses graceful termination to evict the pods. Timeouts are set by default to 40 seconds, before reporting Condition Unknown. Five minutes later, the node controller starts evicting pods. This component is the main node agent for managing individual containers that run in a pod. Vulnerabilities associated with the kubelet are constantly discovered, meaning that you need to regularly upgrade the kubelet versions and apply the latest patches. Access to the kubelet is not authenticated by default, so you should implement strong authentication measures to restrict access. This component handles request forwarding according to network rules. It is a network proxy that supports various protocols (i.e. TCP, UDP) and allows Kubernetes services to be exposed. There are two ways to secure kube-proxy: If proxy configuration is maintained via the kube config file, restrict file permissions to ensure unauthorized parties cannot tamper with proxy settings. Ensure that communication with the API server is only done over a secured port, and always require authentication and authorization. Hardened Node Security You can harden your noder security by following these steps: Ensure the host is properly configured and secure—check your configuration to ensure it meets the CIS Benchmarks standards. Control access to sensitive ports—ensure the network blocks access to ports that kubelet uses. Limit Kubernetes API server access to trusted networks. Limit administrative access to nodes—ensure your Kubernetes nodes have restricted access. You can handle tasks like debugging and without having direct access to a node. Isolation of Sensitive Workloads You should run any sensitive workload on dedicated machines to minimize the impact of a breach. Isolating workloads prevents an attacker from accessing sensitive applications through lower-priority applications on the same host or with the same container runtime. Attackers can only exploit the kubelet credentials of compromised nodes to access secrets that are mounted on those nodes. You can use controls such as node pools, namespaces, tolerations and taints to isolate your workloads. Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux containers and hosts can benefit from container orchestration. Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it. And microservices in containers make it easier to orchestrate services, including storage, networking, and security. Containers give your microservice-based apps an ideal application deployment unit and self-contained execution environment. They make it possible to run multiple parts of an app independently in microservices, on the same hardware, with much greater control over individual pieces and life cycles. Managing the lifecycle of containers with orchestration also supports DevOps teams who integrate it into CI/CD workflows. Along with application programming interfaces (APIs) and DevOps teams, containerized microservices are the foundation for cloud-native applications.

* Provisioning and deployment
* Configuration and scheduling
* Resource allocation
* Container availability
* Scaling or removing containers based on balancing workloads across your infrastructure
* Load balancing and traffic routing
* Monitoring container health
* Configuring applications based on the container in which they will run
* Keeping interactions between containers secure

Container orchestration tools provide a framework for managing containers and microservices architecture at scale. There are many container orchestration tools that can be used for container lifecycle management. Some popular options are Kubernetes, Docker Swarm, and Apache Mesos. Kubernetes is an open source container orchestration tool that was originally developed and designed by engineers at Google. Google donated the Kubernetes project to the newly formed Cloud Native Computing Foundation in 2015.Kubernetes orchestration allows you to build application services that span multiple containers, schedule containers across a cluster, scale those containers, and manage their health over time. Kubernetes eliminates many of the manual processes involved in deploying and scaling containerized applications. You can cluster together groups of hosts, either physical or virtual machines, running Linux containers, and Kubernetes gives you the platform to easily and efficiently manage those clusters. More broadly, it helps you fully implement and rely on a container-based infrastructure in production environments. These clusters can span hosts across public, private, or hybrid clouds. For this reason, Kubernetes is an ideal platform for hosting cloud-native apps that require rapid scaling. Kubernetes also assists with workload portability and load balancing by letting you move applications without redesigning them. When you use a container orchestration tool, such as Kubernetes, you will describe the configuration of an application using either a YAML or JSON file. The configuration file tells the configuration management tool where to find the container images, how to establish a network, and where to store logs. When deploying a new container, the container management tool automatically schedules the deployment to a cluster and finds the right host, taking into account any defined requirements or restrictions. The orchestration tool then manages the container’s lifecycle based on the specifications that were determined in the compose file.You can use Kubernetes patterns to manage the configuration, lifecyle, and scale of container-based applications and services. These repeatable patterns are the tools needed by a Kubernetes developer to build complete systems. Container orchestration can be used in any environment that runs containers, including on-premise servers and public cloud or private cloud environments.

## Problem Definition

The deployment of applications on Kubernetes can sometimes present challenges, including:

1.Complexity: Kubernetes is a complex system with many moving parts, including pods, services, and controllers. Deploying an application on Kubernetes can be challenging for developers who are not familiar with the system.

2.Configuration: Kubernetes requires developers to configure various settings such as resource limits, volumes, and environment variables. This can be time-consuming and error-prone.

3.Scalability: Kubernetes is designed to scale applications easily, but achieving this requires careful planning and design. Developers must ensure that their applications are designed to take advantage of Kubernetes' scaling features.

4.Networking: Kubernetes manages networking between pods and services, which can be complex. Developers must ensure that their applications are configured correctly to take advantage of Kubernetes networking features.

5.Security: Kubernetes has many security features, including role-based access control and network policies. However, developers must ensure that their applications are configured correctly to take advantage of these features.

6.Monitoring and debugging: Kubernetes provides many tools for monitoring and debugging applications, but developers must be familiar with these tools and know how to use them effectively.

Overall, deploying applications on Kubernetes requires a deep understanding of the system and careful planning and design. However, the benefits of using Kubernetes, including scalability, resilience, and flexibility, make it a popular choice for modern application development.

Traditional application deployment typically involves deploying an application on a single server or a group of servers. The application is installed and configured manually on each server, and then the servers are connected to create a functioning system.

On the other hand, deploying an application on Kubernetes involves creating a container image of the application and running that image on a cluster of servers managed by Kubernetes. Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

Here are some key differences between traditional application deployment and deploying on Kubernetes:

1.Scalability: Kubernetes is designed to scale applications horizontally by adding or removing nodes, while traditional deployment is limited by the capacity of the individual servers.

2.Resilience: Kubernetes offers features like automatic self-healing and load balancing that help ensure high availability and uptime of the application. Traditional deployment requires manual intervention to recover from failures or perform load balancing.

3.Configuration management: Kubernetes uses declarative configuration files to define the desired state of the application, while traditional deployment typically involves manual configuration and updates.

4.Resource utilization: Kubernetes can optimize resource utilization by dynamically allocating resources to different parts of the application, while traditional deployment may result in underutilized or overutilized resources.

5.Complexity: Deploying on Kubernetes may require additional skills and knowledge compared to traditional deployment, as it involves working with containers, networking, and Kubernetes-specific tools.

Overall, deploying an application on Kubernetes offers many benefits over traditional deployment, including improved scalability, resilience, and resource utilization. However, it may also require more expertise and resources to implement and maintain.

## Problem Overview

Before deploying an application on Kubernetes, there are several challenges and considerations that must be addressed.

Firstly, organizations must decide whether containerization is the right approach for their application. While containerization provides numerous benefits such as portability, scalability, and consistency, it may not be suitable for all types of applications. For example, applications with complex dependencies or specialized hardware requirements may not be well-suited for containerization.

Secondly, organizations must ensure that their application is designed with containerization in mind. This involves breaking the application down into smaller, independent services that can be deployed and scaled independently. Additionally, the application must be packaged into a container image that includes all of the necessary dependencies and configuration.

Thirdly, organizations must ensure that their infrastructure is capable of supporting containerized applications. This includes setting up a container registry, configuring a container runtime such as Docker, and ensuring that the underlying infrastructure has sufficient resources to support containerized workloads.

Fourthly, organizations must consider how they will manage and orchestrate their containerized application. While Kubernetes provides a powerful set of features for managing containers and services, organizations must still invest in the necessary skills and expertise to manage the platform effectively.

Finally, organizations must consider the security implications of deploying a containerized application. Containers can introduce new security risks, such as container escape vulnerabilities, and must be properly secured to ensure that the application and the underlying infrastructure are not compromised.

In summary, before deploying an application on Kubernetes, organizations must carefully consider factors such as application suitability, containerization readiness, infrastructure capabilities, orchestration and management, and security. By addressing these challenges upfront, organizations can ensure that their Kubernetes deployment is successful and meets their business requirements.

## Hardware Specification

When deploying applications on Kubernetes, it is important to consider the hardware specification for the nodes that will be part of the Kubernetes cluster. The hardware specification will determine the amount of compute resources available for the applications to run and the performance of the cluster.

Here are some hardware specifications to consider:

1. CPU: The number of CPU cores and clock speed is important for the performance of the applications. It is recommended to have at least 2 CPU cores per node for a small cluster and increase the number of cores as the cluster grows.

2. Memory: The amount of memory available on each node will determine the number of containers that can be run on the node. It is recommended to have at least 4 GB of memory per node for a small cluster and increase the memory as the cluster grows.

3. Storage: The amount of storage available on each node will determine the amount of data that can be stored locally. It is recommended to have at least 20 GB of storage per node for a small cluster and increase the storage as the cluster grows.

4. Network: The network bandwidth and latency between the nodes in the cluster are important for the performance of the applications. It is recommended to have a high-speed network with low latency for a smooth running cluster.

5. GPUs: If the applications require GPU resources for machine learning or other data-intensive tasks, it is recommended to have nodes with dedicated GPUs.

## Software Specification

When deploying applications on Kubernetes, it is important to consider the hardware specification for the nodes that will be part of the Kubernetes cluster. The hardware specification will determine the amount of compute resources available for the applications to run and the performance of the cluster.

Here are some hardware specifications to consider:

1. CPU: The number of CPU cores and clock speed is important for the performance of the applications. It is recommended to have at least 2 CPU cores per node for a small cluster and increase the number of cores as the cluster grows.

2. Memory: The amount of memory available on each node will determine the number of containers that can be run on the node. It is recommended to have at least 4 GB of memory per node for a small cluster and increase the memory as the cluster grows.

3. Storage: The amount of storage available on each node will determine the amount of data that can be stored locally. It is recommended to have at least 20 GB of storage per node for a small cluster and increase the storage as the cluster grows.

4. Network: The network bandwidth and latency between the nodes in the cluster are important for the performance of the applications. It is recommended to have a high-speed network with low latency for a smooth running cluster.

5. GPUs: If the applications require GPU resources for machine learning or other data-intensive tasks, it is recommended to have nodes with dedicated GPUs.

# LITERATURE SURVEY

[1]. L. Lin, Y. Gao, and H. Shen delivered this work titled "An Empirical Study on the Use of Kubernetes for DevOps" at the International Conference on Cloud Computing and Big Data (CCBD 2021). The study's objective was to conduct an empirical investigation to learn more about how Kubernetes is used in DevOps practises. The adoption of Kubernetes by organisations, as well as the advantages and difficulties of using it in a DevOps environment, are examined in this article. The report also identifies the critical elements that are necessary for Kubernetes to be implemented successfully in a DevOps environment. The findings of a poll the authors performed among DevOps experts showed that Kubernetes is extensively used and that it aids in automating the deployment, scaling, and administration of containerized applications. According to the report, organisations' primary obstacles to adopting Kubernetes for DevOps are a lack of experience and the software's complexity. Overall, the report offers insightful information about the use of Kubernetes in DevOps and can be a helpful resource for businesses considering including Kubernetes in their DevOps workflows.

[2]. C. Xu and Y. Sun The paper "DevOps for Kubernetes: A Survey and Future Research Directions," by C. Xu and Y. Sun, offers an overview of the state of DevOps practises and tools for Kubernetes at the moment. The complexity of the technology and the requirement for specialised skills are just two of the difficulties the authors mention while adopting DevOps for Kubernetes. Additionally, they examine the current Continuous Integration/Continuous Delivery (CI/CD), monitoring, and logging DevOps tools and procedures for Kubernetes. The discussion of potential future research topics in DevOps for Kubernetes, including enhancing the interaction of Kubernetes and DevOps technologies and creating new tools and methods for managing Kubernetes at scale, finishes the study.

[3]. T. Zhang and H. Zhang In this paper, a brand-new DevOps framework built on the microservices architecture and Kubernetes is proposed. The authors talk about the problems with conventional DevOps methods and how the suggested framework might solve them. The framework makes use of Kubernetes features like service discovery, automated scaling, and container orchestration to speed up the deployment and administration of microservices-based applications. The framework's implementation is also discussed, and the authors offer test results that show how effective it is in terms of scalability and fault tolerance. The report ends by offering potential future research topics for enhancing the suggested framework. It appears that the authors suggested a brand-new Kubernetes- and microservices-based DevOps framework. The framework was probably designed and put into use, tested in a simulated or real-world setting, and then its efficiency in terms of enhancing DevOps operations was assessed.

[4]. Y. Zhang and H. Yu A DevOps pipeline based on Kubernetes and Jenkins is suggested in the work by Y. Zhang and H. Yu. The methodology includes developing and executing a DevOps pipeline that combines the Jenkins continuous integration and continuous deployment (CI/CD) tool with the Kubernetes container orchestration platform. Source code management, building and packaging, testing, and deployment are the four primary stages of the pipeline as it is currently being proposed. The proposed DevOps pipeline is shown to be effective by the authors in terms of lowering manual labour requirements, boosting productivity, and guaranteeing code quality through the inclusion of a case study. Software development teams employ a DevOps pipeline, which is a series of automated steps, to build, test, and reliably deliver software applications. The pipeline begins with the development stage, during which code alterations are produced and entered into version control databases. The code is then automatically compiled, tested, and merged with the main branch as part of continuous integration (CI), which follows the code changes. The code is then put through continuous delivery (CD), where the production environment is automatically deployed. Automated testing, code review, and monitoring are also included in the pipeline to guarantee that the software is of the highest quality and that any flaws are found early in the development process. Software development teams can speed up and enhance the quality of their application delivery by adopting a DevOps pipeline to optimise their development process.

[5]. H. Cheng and X. Li The obstacles and best practises of implementing DevOps for Kubernetes are covered in the paper "DevOps for Kubernetes: Challenges and Best Practises" by H. Cheng and X. Li, which was published in the Proceedings of the International Conference on Information and Computer Technologies (ICICT 2021). When adopting DevOps on Kubernetes, organisations encounter difficulties with scalability, complicated infrastructure management, and continuous deployment, among others. They also offer a number of best practises, such as employing automation tools, building a microservices architecture, and putting security measures in place, for deploying DevOps on Kubernetes. The significance of collaboration and communication between development and operations teams is emphasised in the paper's conclusion in order to ensure a successful DevOps implementation on Kubernetes. Microservices are a design strategy for creating software systems that are made up of a number of tiny, independent services that connect with one another via well defined APIs. In a microservices architecture, each service is in charge of a particular business capability and has its own autonomous development, deployment, and scaling capabilities. With this method, software development is encouraged to be modular, agile, and scalable, and organisations are better able to adapt to shifting business needs.

[6]. J. Li, W. Li, and X. Zou The work describes a Kubernetes-based DevOps pipeline for microservices applications. Development, testing, deployment, and monitoring of microservices applications are all automated by the pipeline. The authors describe the pipeline's architecture and the numerous parts that make it up. In the year 2021, J. Li, W. Li, and X. Zou published their work titled "A DevOps Pipeline for Microservices Applications Based on Kubernetes" in the Proceedings of the International Conference on Internet of Things and Intelligent Applications (ITIA 2021). The pipeline is built on a variety of DevOps-related tools and technologies, such as Git, Jenkins, Docker, Kubernetes, and Prometheus. The authors explain how these tools are combined and utilised individually in the pipeline.A case study that illustrates the pipeline's effectiveness in a practical setting is also included in the paper. The case study focuses on the creation and implementation of a microservices application for an online store. The authors describe how the pipeline was used to automate the creation, testing, and distribution of the application and how this improved the process's overall effectiveness.

[7]. M. Zhang, Q. Li In 2021, M. Zhang, Q. Li, and Y. Feng published their article titled "A DevOps Platform for Kubernetes with Automated Testing and Deployment" in the Proceedings of the 2021 IEEE International Conference on Software Quality, Reliability, and Security (QRS 2021).The paper describes an automated testing and deployment DevOps framework for Kubernetes. The platform's goal is to make developing, testing, and deploying apps on Kubernetes simpler.The platform's architecture and the numerous parts that make it up are described by the creators. Git, Jenkins, Docker, Kubernetes, and Helm are just a few of the DevOps-related tools and technologies that are included in the platform. The platform's use for automating application creation, testing, and deployment on Kubernetes is well described in the paper. The platform aids in lowering the time and effort needed for these tasks, as well as enhancing the dependability and quality of the apps, according to the authors.A case study that illustrates the platform's effectiveness in a practical setting is also included in the paper. The case study focuses on the creation and implementation of a microservices application for an online store. The platform was used to automate the creation, testing, and distribution of the application, and the authors describe how this increased the overall effectiveness of the development process.

[8]. L. Zhao and H. Wang The authors of this research suggest a Kubernetes-based DevOps pipeline for distributed deep learning. The pipeline consists of a number of steps, such as model training, model deployment, and data preparation. To manage the distributed resources and automate the deployment and scaling of deep learning models, the authors employ Kubernetes. For continuous integration and continuous deployment, they further employ GitLab and Jenkins. The performance and scalability of the deep learning models can be greatly improved by the suggested pipeline, according to the findings of an evaluation of the pipeline using a case study on the CIFAR-10 dataset. In-depth information about an automated DevOps pipeline for Kubernetes deployments is provided in the paper. Software development teams aiming to simplify their development procedures and raise the calibre and dependability of their apps on Kubernetes may find the pipeline outlined in the article to be a useful resource.

[9]. X. Li and X. Liu In the Proceedings of the International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI 2021), X. Li and X. Liu's work titled "An Automated DevOps Pipeline for Kubernetes Deployments" was released in 2021.The goal of the article is to simplify the process of developing, testing, and deploying apps on Kubernetes by describing an automated DevOps pipeline for Kubernetes deployments.The authors describe the pipeline's architecture and the numerous parts that make it up. Git, Jenkins, Docker, Kubernetes, and Helm are just a few of the DevOps-related tools and technologies that are part of the pipeline. The pipeline's use for automating the development, testing, and deployment of applications on Kubernetes is covered in great detail in the paper. The authors describe how the pipeline helps to decrease the time and effort needed for these tasks as well as how it helps to enhance the applications' quality and dependability.A case study that illustrates the pipeline's effectiveness in a practical setting is also included in the paper. The case study focuses on the creation and implementation of a microservices application for an online store. The authors describe how the pipeline was used to automate the creation, testing, and distribution of the application and how this improved the process's overall effectiveness. Overall, the article offers a thorough overview of a DevOps process that is automated for Kubernetes deployments. Software development teams aiming to simplify their development procedures and raise the calibre and dependability of their apps on Kubernetes may find the pipeline outlined in the article to be a useful resource.

[10]. X. Yu, Y. Zhang In the year 2021, the authors X. Yu, Y. Zhang, and X. Huang published their work titled "A Container-Based DevOps Pipeline for Kubernetes Deployments" in the Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE 2021).In order to streamline the procedure for developing, testing, and deploying apps on Kubernetes, the paper outlines a container-based DevOps pipeline for Kubernetes deployments.The authors describe the pipeline's architecture and the numerous parts that make it up. Git, Jenkins, Docker, Kubernetes, and Helm are just a few of the DevOpsrelated tools and technologies that are part of the pipeline. The pipeline's use for automating the development, testing, and deployment of applications on Kubernetes is covered in great detail in the paper. The authors describe how the pipeline helps to decrease the time and effort needed for these tasks as well as how it helps to enhance the applications' quality and dependability A case study that illustrates the pipeline's effectiveness in a practical setting is also included in the paper. The case study focuses on the creation and implementation of a microservices application for an online store. The authors describe how the pipeline was used to automate the creation, testing, and distribution of the application and how this improved the process's overall effectiveness. The work offers a thorough description of a container-based DevOps pipeline for Kubernetes deployments as a whole. Software development teams aiming to simplify their development procedures and raise the calibre and dependability of their apps on Kubernetes may find the pipeline outlined in the article to be a useful resource.

## 2.1 Existing System

There are several existing systems for deploying applications on Kubernetes, including:

1. Kubernetes Dashboard: Kubernetes Dashboard is a web-based user interface for managing Kubernetes clusters. It provides a graphical representation of the resources running in the cluster and allows administrators to manage the cluster and its resources through an intuitive interface.

2. Helm: Helm is a package manager for Kubernetes that helps developers and operators manage the deployment of applications and services on Kubernetes. It provides a way to package, share, and deploy applications as Helm charts, which are a collection of YAML files that describe a set of Kubernetes resources needed to run an application.

3. Istio: Istio is an open-source service mesh platform that provides a way to connect, secure, and manage microservices running on Kubernetes. It provides a way to manage traffic routing, load balancing, and service-level security between microservices.

4. Kustomize: Kustomize is a tool for customizing Kubernetes resources. It allows developers to manage configuration files in a declarative way and generate different configurations based on the environment they are deploying to.

5. GitOps: GitOps is a methodology for deploying and managing applications on Kubernetes using Git as the single source of truth. It allows developers to use Git as the primary tool for managing the configuration of the cluster and deploying applications.

## 

## 2.2 Proposed System

To deploy an application on Kubernetes, there are several steps involved. Here is a high-level overview of the process:

1. Define the application's container image and other necessary resources (such as config files, environment variables, etc.).

2. Create a Kubernetes deployment object that specifies the desired state of the application.

3. Create a Kubernetes service object that exposes the application to other pods in the cluster.

4. Create any necessary Kubernetes objects, such as ingress or persistent volume claims, to support the application.

5. Apply the Kubernetes objects to the cluster using the kubectl apply command or a similar tool.

Here are some more detailed steps for deploying an application on Kubernetes:

1. Create a Dockerfile that defines the application's container image. The Dockerfile should include all the necessary dependencies and configurations for the application to run.

2. Build the Docker image and push it to a container registry such as Docker Hub or Google Container Registry.

3. Create a Kubernetes deployment object. The deployment object specifies the desired state of the application, including the number of replicas, the container image to use, and any necessary environment variables or configuration files.

4. Create a Kubernetes service object. The service object exposes the application to other pods in the cluster by creating a stable DNS name and IP address that can be used to access the application.

5. If the application requires additional Kubernetes objects, such as ingress or persistent volume claims, create those objects as well.

6. Apply the Kubernetes objects to the cluster using the kubectl apply command or a similar tool.

7. Monitor the deployment using Kubernetes tools such as kubectl or the Kubernetes dashboard. Troubleshoot any issues that arise and make any necessary changes to the deployment or Kubernetes objects.

Overall, deploying an application on Kubernetes requires careful planning and attention to detail, but once the deployment is up and running, Kubernetes provides powerful tools for scaling, updating, and managing the application.

**2.3 Literature Review Summary**

Deploying applications on Kubernetes is a popular approach for managing containerized applications at scale. A literature review on this topic highlights several key insights.

Firstly, Kubernetes offers several benefits for application deployment, such as enhanced scalability, fault tolerance, and resource utilization. Additionally, its flexible architecture allows for the deployment of a wide range of applications and services.

Secondly, there are several approaches to deploying applications on Kubernetes, including manual deployment, using YAML files, using Helm charts, and using Kubernetes operators. Each approach has its own advantages and disadvantages, and the choice of approach depends on the specific requirements of the application and the preferences of the development team.

Thirdly, there are several challenges associated with deploying applications on Kubernetes, such as security concerns, complexity, and performance issues. To overcome these challenges, best practices have emerged, such as using optimized container images, implementing proper network and security configurations, and using monitoring and logging tools to detect and address issues.

# PROBLEM FORMULATION

As businesses move towards adopting cloud-native applications, they need to deploy and manage their applications effectively. One popular solution for application deployment and management is Kubernetes, an open-source container orchestration platform. However, deploying applications on Kubernetes can be complex and time-consuming, especially for teams that are new to the technology.

The problem is how to deploy an application on Kubernetes effectively and efficiently, especially for teams that are new to the technology. This involves understanding the architecture of Kubernetes and its components, creating container images for the application, defining Kubernetes resources such as pods, services, and deployments, configuring networking, scaling, and monitoring, and troubleshooting any issues that arise during deployment. Additionally, teams need to understand the best practices for deploying and managing applications on Kubernetes to ensure that they are scalable, resilient, and secure.

Overall, the problem formulation is how to deploy an application on Kubernetes while minimizing the complexity and time required for deployment, ensuring the application is scalable, resilient, and secure, and enabling teams to manage the application effectively.

# OBJECTIVES

The primary objectives of deploying applications on Kubernetes are:

Scalability: Kubernetes provides automatic scaling of applications based on their resource usage, which allows for efficient utilization of computing resources and ensures that applications can handle increased traffic.

High Availability: Kubernetes ensures high availability of applications by automatically managing replicas of application instances across multiple nodes, allowing for fault tolerance and resiliency in the face of node failures or application crashes.

Flexibility: Kubernetes allows for flexibility in deploying and managing applications across different environments, such as on-premises data centers or public cloud platforms.

Easy Deployment and Rollback: Kubernetes provides a declarative approach to deploying applications, making it easy to define and manage application configurations. This approach also enables easy rollback to previous application versions in case of issues or errors.

Resource Optimization: Kubernetes allows for efficient utilization of computing resources by scheduling and managing application instances based on their resource requirements and available resources.

Continuous Delivery: Kubernetes provides a platform for continuous delivery of applications by enabling automated deployments, rolling updates, and canary releases.

Overall, deploying applications on Kubernetes provides a powerful platform for managing and scaling applications across different environments, while also ensuring high availability, flexibility, and resource optimization.

# METHODOLOGY

Here are the general steps involved:

Write your application code in a programming language of your choice. Some popular languages for Kubernetes applications include Go, Python, and Java.

Use a version control system like Git to manage your codebase. Create a Git repository for your application code.

Create a Dockerfile to package your application into a container image. The Dockerfile should include all the necessary dependencies and configurations for your application to run on Kubernetes.

Build the container image using a container registry like Docker Hub or Google Container Registry.

Create a Kubernetes deployment configuration file that specifies how to deploy your containerized application. This file should include details like the container image location, number of replicas, and any necessary environment variables.

Push your deployment configuration file to your Git repository.

Use a Kubernetes CLI tool like kubectl to apply your deployment configuration to your Kubernetes cluster. This will create the necessary Kubernetes objects like pods, services, and deployments.

Monitor your application and make any necessary changes to your deployment configuration file to ensure your application is running smoothly.

By following these steps, you can deploy your application on Kubernetes using Git and a programming language of your choice.

Deploying an application on Kubernetes can involve multiple steps, but here is a high-level overview of the process:

Containerize your application: First, you need to containerize your application by creating a Docker image. This involves creating a Dockerfile that describes the environment and dependencies needed for your application to run.

Create a Kubernetes deployment: Next, you need to create a Kubernetes deployment that describes how many instances of your application should be running and how they should be configured. You'll also specify the Docker image that you created in step 1.

Create Kubernetes service: To expose your application to the internet or other services running in your cluster, you need to create a Kubernetes service. The service will create a stable endpoint that other services can use to communicate with your application.

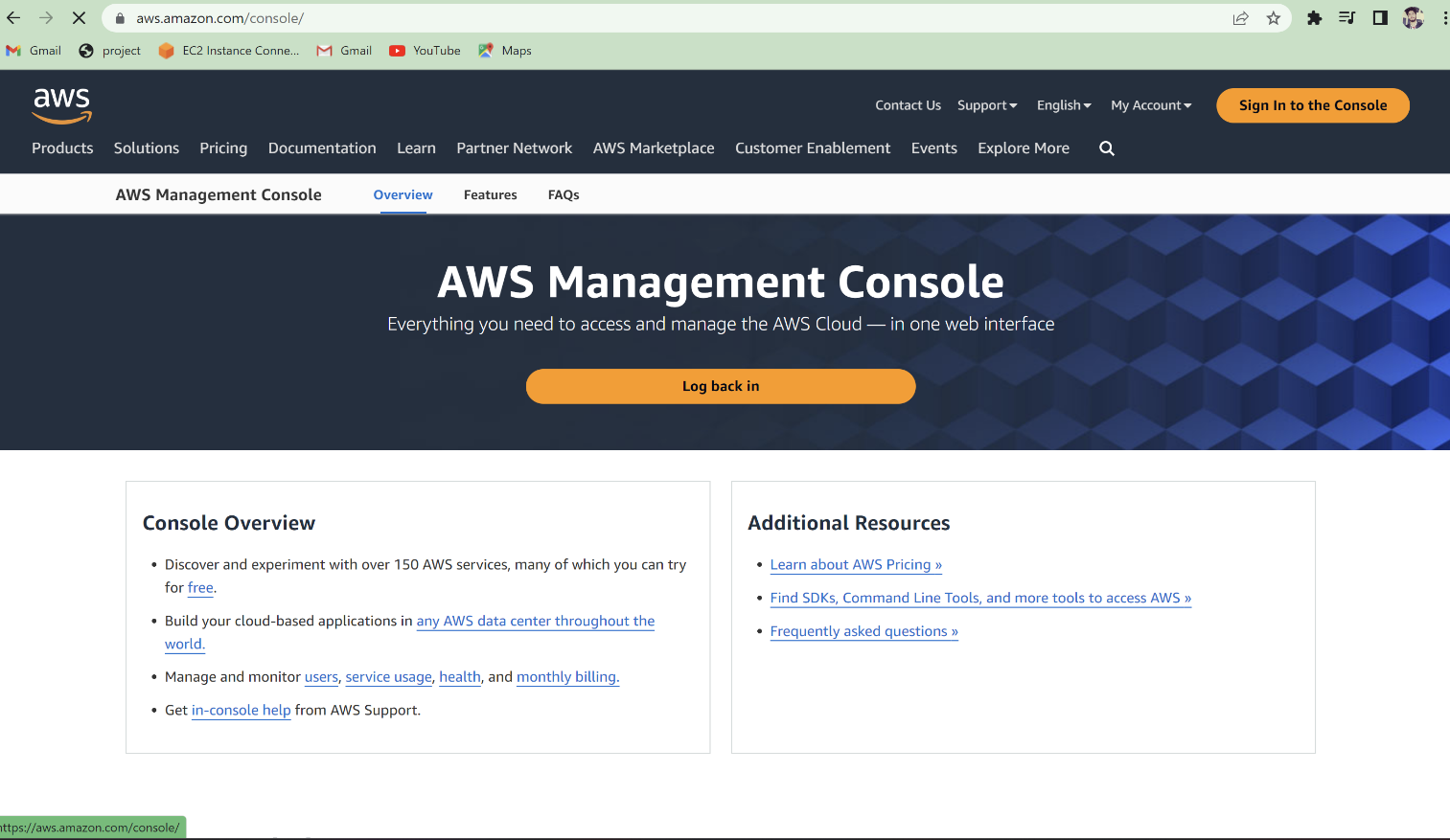
Define Kubernetes ingress: If you want to expose your application to the internet, you'll need to define a Kubernetes ingress. This will create a routing layer that can route traffic to your service.

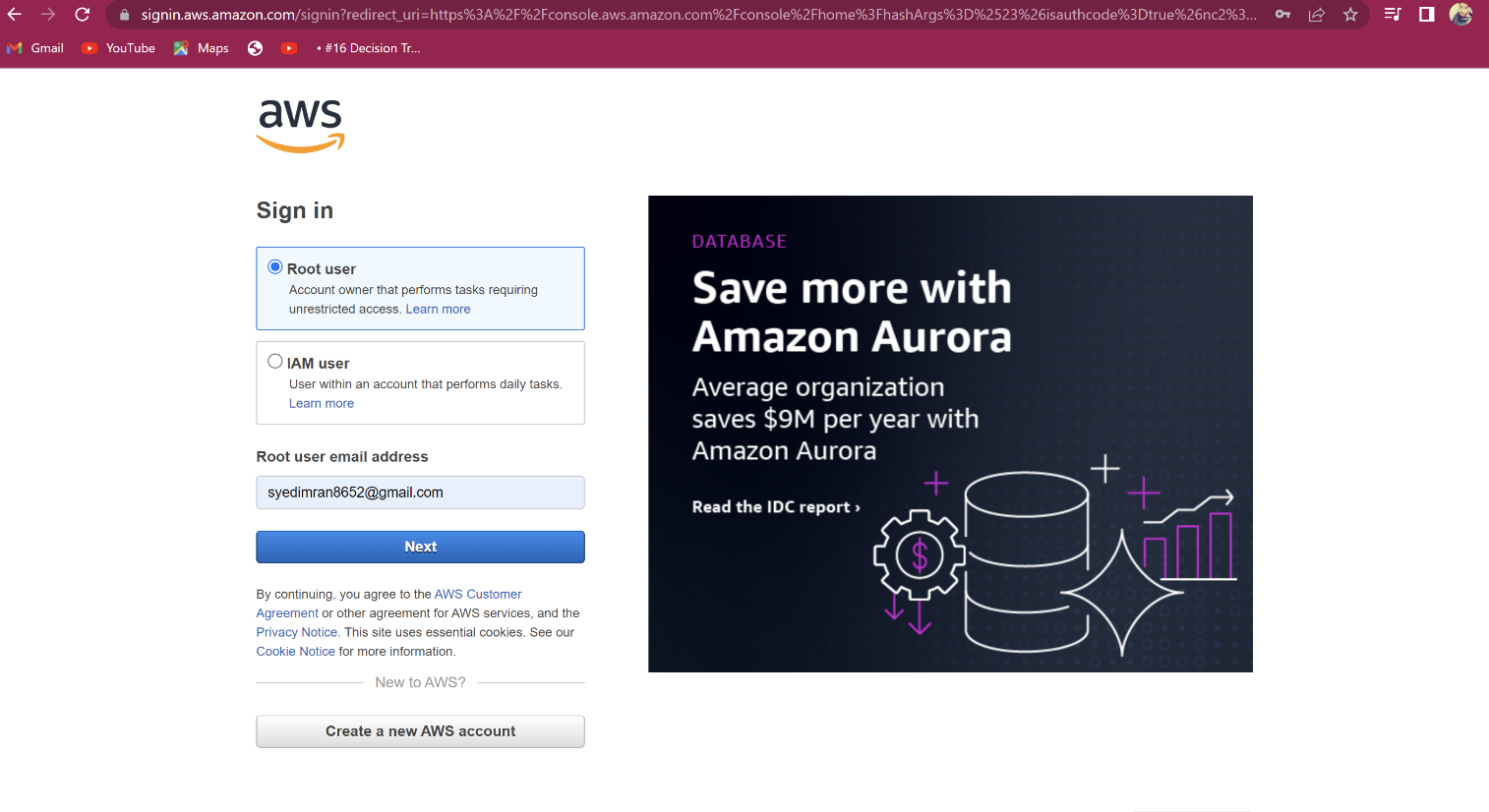
Deploy your application: Once you've created your deployment, service, and ingress, you can deploy your application to your Kubernetes cluster. This will create the necessary resources in the cluster to run your application.

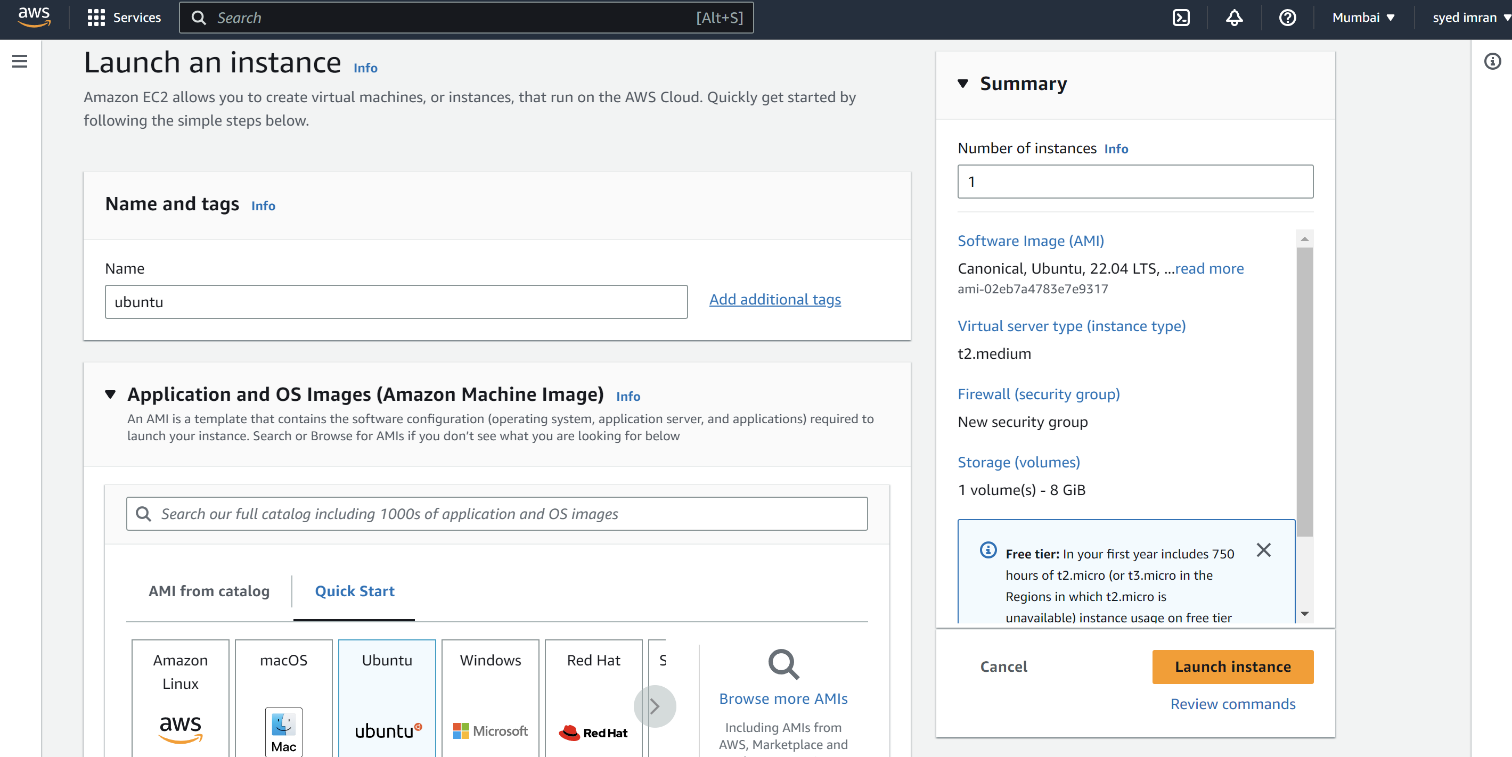
Monitor your application: Finally, you'll want to monitor your application to ensure that it's running smoothly. You can use Kubernetes tools like kubectl or Grafana to monitor the health of your application and troubleshoot any issues that arise.’

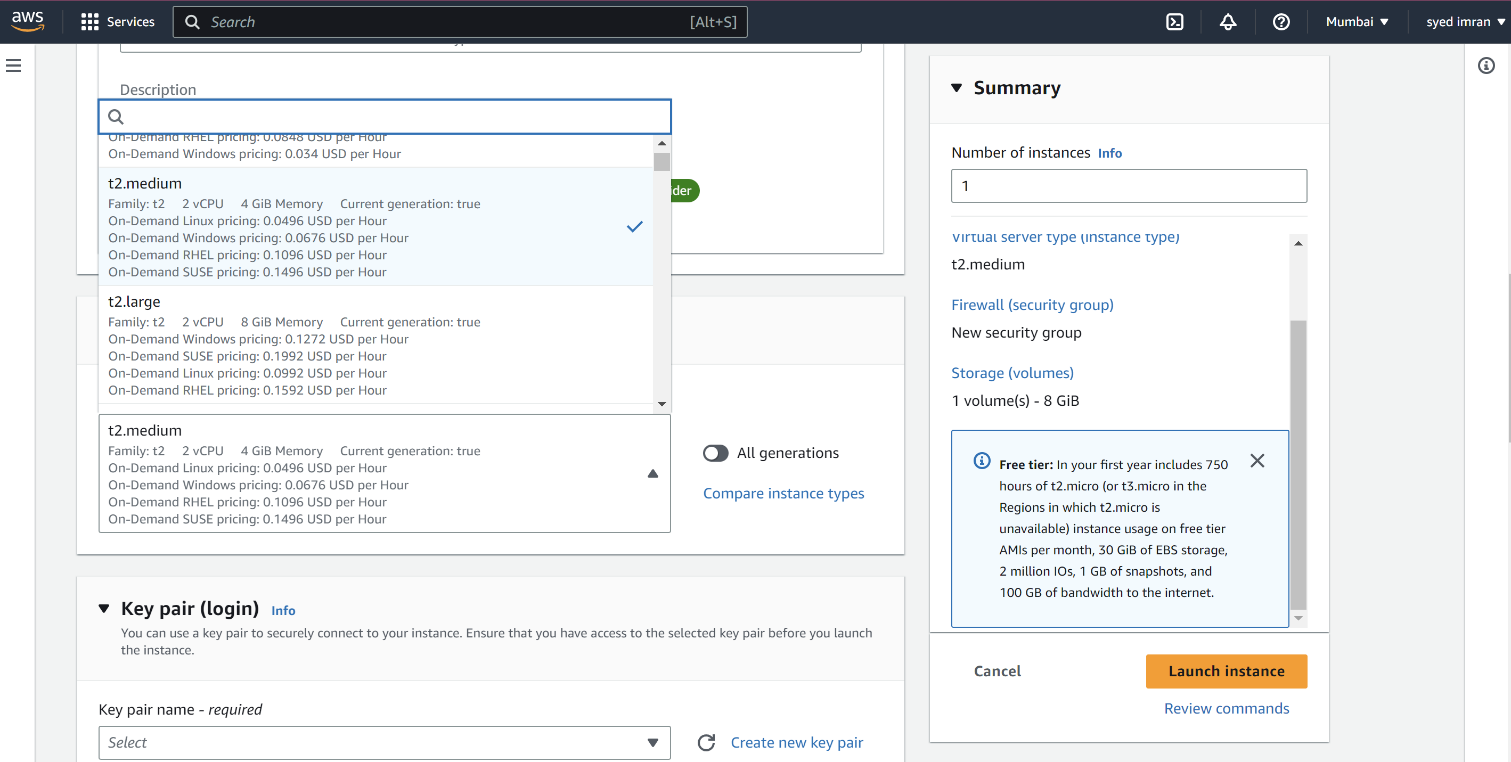
**Experimental Setup**

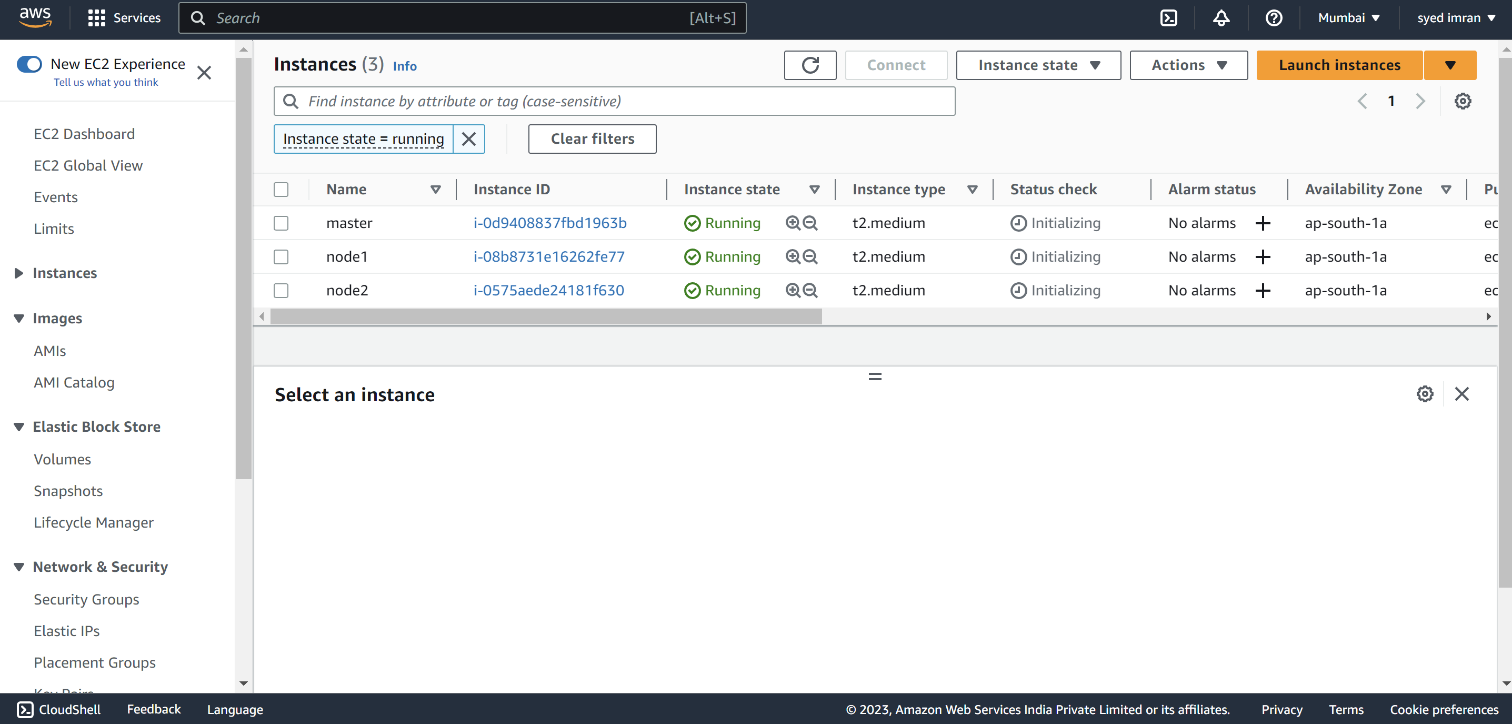


****

****

****

****

****

**Steps for Installing Docker:**

1. Open the terminal on Ubuntu.

2. Remove any docker file that are running in the system, using the following command:

|  |
| --- |
| **$ sudo apt-get remove docker docker-engine docker.io** |
|  |

After entering the above command, you will need to enter the password of the root and press enter.

3. Check if the system is up-to-date using the following command:

|  |
| --- |
| **$ sudo apt-get update** |
|  |

4. Install Docker using the following command:

|  |
| --- |
| **$ sudo apt install docker.io** |
|  |

You’ll then get a prompt asking you to choose between y/n - choose y

5. Install all the dependency packages using the following command:

|  |
| --- |
| **$ sudo snap install docker** |
|  |

6. Before testing Docker, check the version installed using the following command:

|  |
| --- |
| **$ docker --version** |
|  |

7. Pull an image from the Docker hub using the following command:

|  |
| --- |
| **$ sudo docker run hello-world** |
|  |

Here, hello-world is the docker image present on the Docker hub.

8. Check if the docker image has been pulled and is present in your system using the following command:

|  |
| --- |
| **$ sudo docker images** |
|  |

9. To display all the containers pulled, use the following command:

|  |
| --- |
| **$ sudo docker ps -a** |
|  |

10. To check for containers in a running state, use the following command:

|  |
| --- |
| **$ sudo docker ps** |
|  |

You’ve just successfully installed Docker on Ubuntu!

# 

**DEFINITION*:*** *Kubernetes is an open source container management tool, which automates container deployment, container scaling and load balancing.*

It schedules runs and manages isolated containers which are running on virtual/physical/cloud machines.

**Online platforms:** Kubernetes playground,

Play with K8s classroom

Play with Kubernetes k8s

**K8s installation tools:** kubeadm & minicube

**FEATURES:**

Orchestration (clustering no of containers running on different network)

Auto Scaling

Auto-healing

Load balancing

Platform Independent (cloud/virtual/physical)

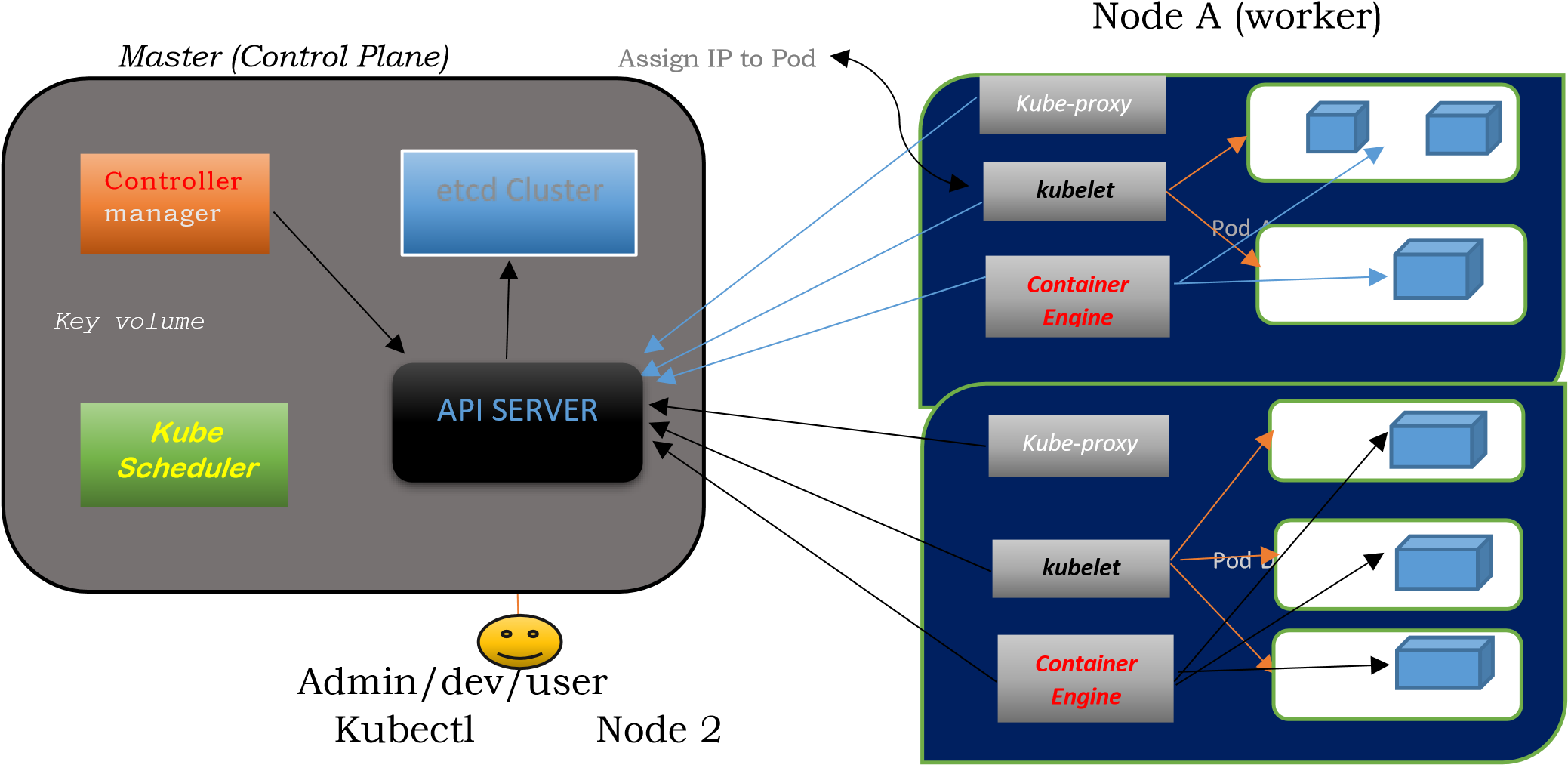
Fault tolerance (node/pod failure)

Roll Back (going back to previous version)

Healthy monitoring & Containers

Batch execution (one time sequential parallel).

**Comparisons these according to their features:**



***Node is going to run to 3 important piece of software process.***

**KUBELET:**

Agent running on the node

Listen to k8s master (ex: pod creation request)

Use port 10255

Sends success/fail report to master.

**CONTAINER ENGINE:**

Works with kubeles

Pulling images

Start /stop containers

Exposing container on port specified in manifest.

**KUBE-PROXY:-**

Assign IP to each pod

It is required to assign IP address to Pods (dynamic)

Kube-proxy runs on each node and this make sure

that each pod will get its own unique IP address.

***Working with Kubernetes:***

We create manifest (Jason .yml)

Apply this cluster to master (to master) to bring into desired state.

Pods runs on node which is controlled by master.

***Role of master node:***

Kubernetes cluster contains running on bare metal/VM instance /Cloud instances/All mix.

Kubernetes designates one or more of these as master and all others are workers.

The master is now going to run set of k8s process. These process will insure smooth functioning of cluster these process are called Control plane.

It can be multi master for high availability.

Master runs control plane to run cluster smoothly.

***Components of Control Plane master:***

Kube API server: - (*for all communications)*

This interacts directly with user

(*If we applied Jason or .YML manifest to kube API server).*

The kubeAPI server is meant to scale automatically as per load.

Kube API server is front end of control-plane.

etcd:-

It stores metadata and status of cluster.

It is consistent and H-A store (*key volume store*)

*Source of Touch* for cluster state. (*Information about cluster’s state*).

*etcd features:-*

*Fully replicated,*

*Secure > Implements automatic TLS with optional client certificate*

*Fast > Benchmarked at 10,000 writes per second.*

**Kube-Scheduler:-**

When user make request for the creation and management of PODS kube scheduler is going to take actions on these requests.

Handled POD creation and management.

It match/assign any node to create and run pods.

A scheduler watches for newly created pods that have no node assigned for every pod that the scheduler discovers, the scheduler becomes responsible for finding best node for that POD to run ON.

Scheduler gets the information for hardware configuration from configuration files and schedules the PODS on nodes accordingly.

**Controller manager:-**

It makes sure actual state of cluster matches to desired state. Two possible choices for controller manager:

If k8s on cloud then it will be cloud controller manager

If k8s on non-cloud, then it will be kube-controller manager.

***Components on Master that runs controller:***

**Node–controller: -** For checking the cloud provider to determine if a node has been detected in the cloud after it stops responding.

Route Controller: - Responsible for setting up network routes on your cloud.

**Service controller: -** Responsible for load balancers on your cloud against service of type load balancer.

Volume controller: - for creating attaching and maintaining volumes and interacting with the cloud provider to orchestrate volume.

***POD: -***

Smallest unit in kubernetes (*usually contains 1 container*).

It is a group of one or more container that are deployed together on the dame host.

A cluster is a group of nodes which has at least 1 master and 2 worker nodes.

In K8s Pod is the control unit not the container.

Pod runs on node which is control by master.

K8s communicates with pods not container.

Without POD we cannot start containers.

***Multi-Container Pod:-***

Share access to memory space.

Connect to each other using local host <*container host*>

Share access to the same volume

Container within pod are deployed in An, All or Nothing manner.

Entire pod is hosted on the same node (*scheduler will decide about node*).

***Pod limitations: -***

No auto healing and scaling

Pod creases

***Higher Level K8s Objects:-***

Deployment: Versioning and Rollback

Replication set: Scaling and healing

Service: Static (*non-ephemeral)* IP networking

Volume: Non ephemeral storage

***Set up of K8S master and worker node on AWS:***

Minimum requirement for master is 4 GB RAM and 2CPU.

Create 3 instances (Ubuntu 16.04 t2 medium) 1 for master 2 for nodes.

***Commands for master and nodes:***

*sudo su apt-get update apt-get install apt-transport-https*

This https is needed for intra cluster communication (Particularly from control plane to individual pods).

**Now install Docker on all 3 instances:**

*apt install docker.io -y docker –version*

**To check whether Docker is installed or not?**

*systemctl start docker systemctl enable docker*

Set up open GPG key this is required for intra cluster communication it will be added to source key on this node when K8s sends singed info’s to our host, it is going to accept those information because this open GPG key is present in the source key.

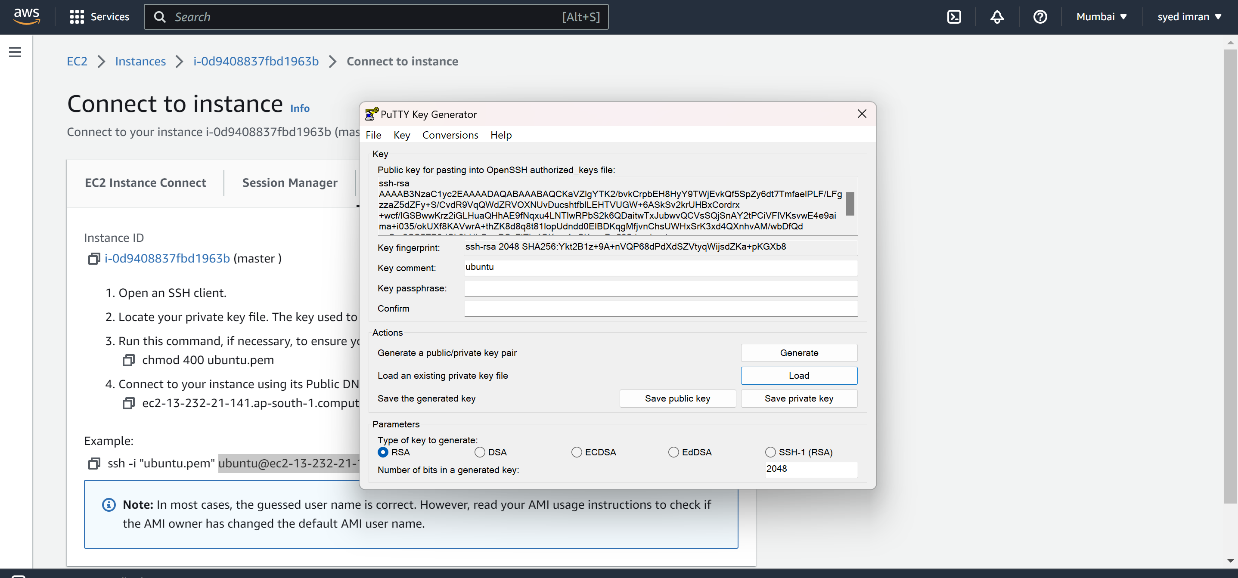
*sudo curl -s https://packages.cloud.google.com/apt... | sudo apt-key add*

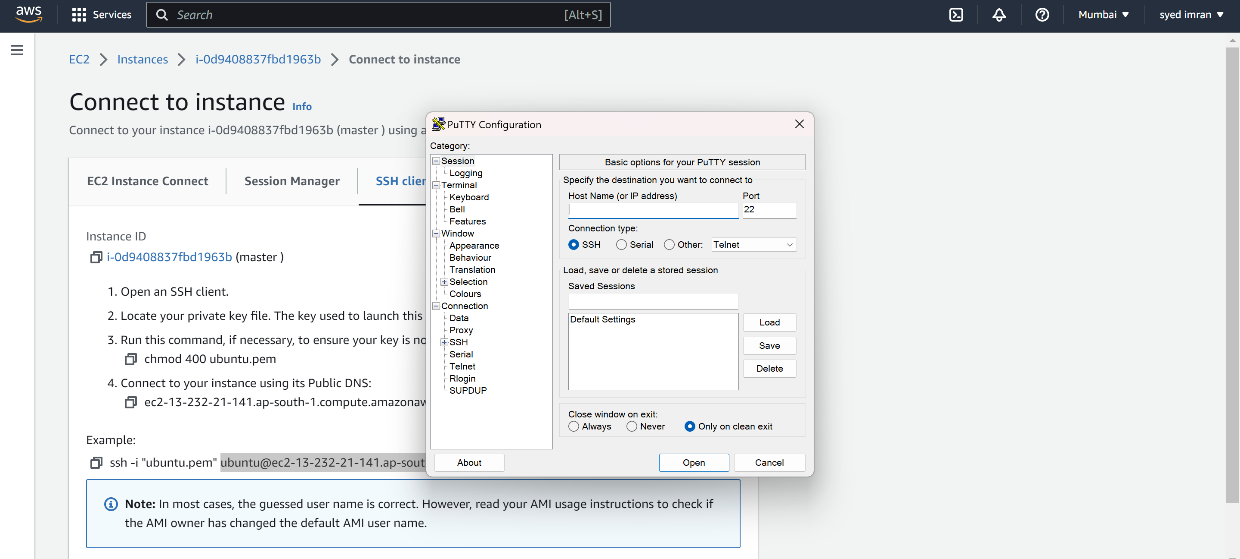
*Paste this on all three instances master node, node 1 and node 2.*

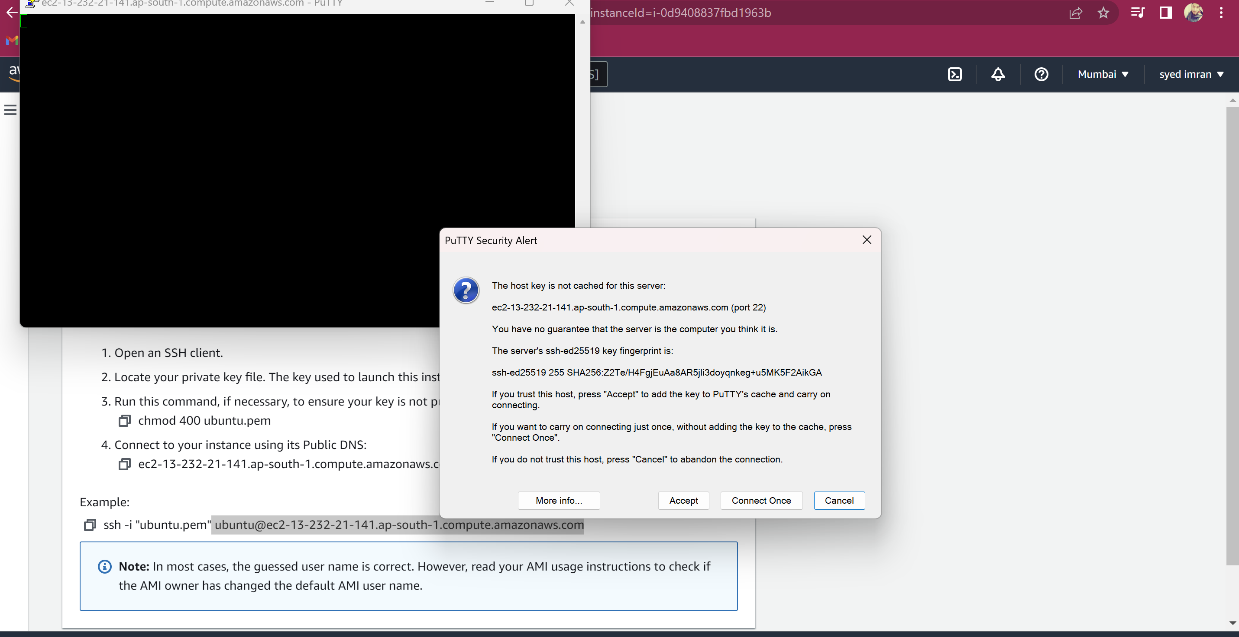
Edit source list file (apt-get-install nano)

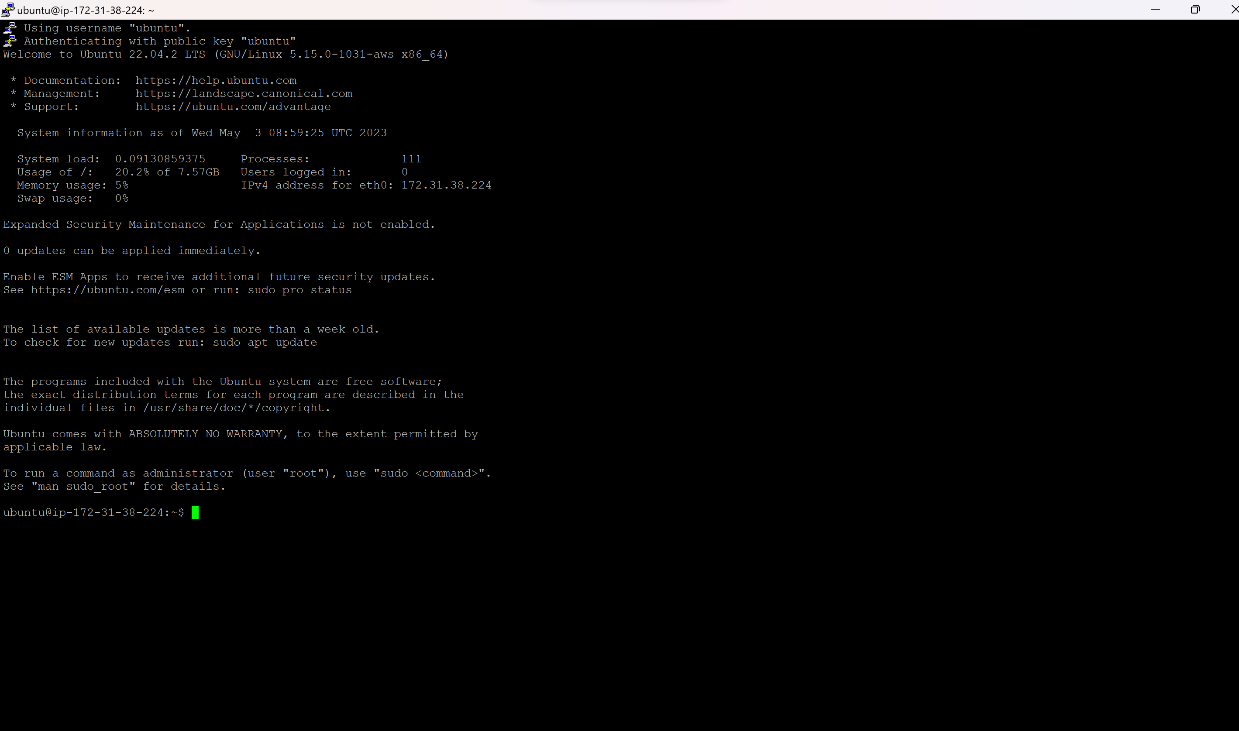
# 

### 









*Commands for master and nodes:*

*sudo su apt-get update*

*apt-get install apt-transport-https*

This https is needed for intra cluster communication (Particularly from control plane to individual pods).

**Now install Docker on all 3 instances:**

**To check whether Docker is installed or not?**

*Systemctlstart docker*

*systemctl enable docker*



**Set up open GPG key this is required for intra cluster communication** it will be added to source key on this node when K8s sends singed info’s to our host, it is going to accept those information because this open GPG key is present in the source key.

### *sudo curl -s https://packages.cloud.google.com/apt... | sudo apt-key add*

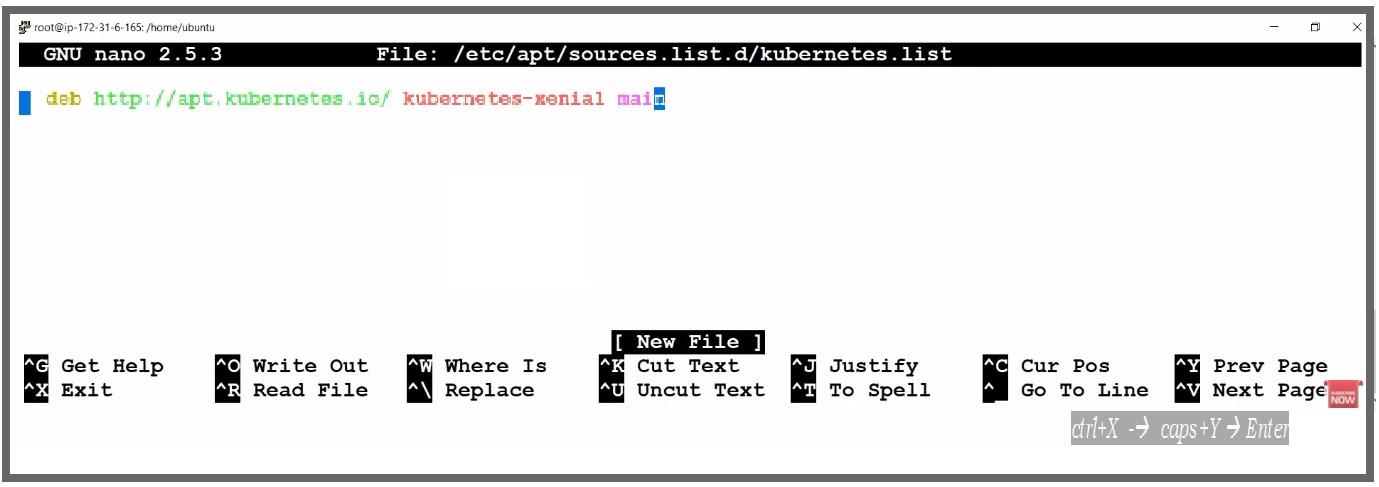
***Paste this on all three instances master node, node 1 and node 2.***

**Edit source list file** (**apt-get-install nano**)

*Create nano file, Go inside and paste this (Xenial) command in side all nodes*

# nano /etc/apt/sources.list.d/kubernetes.list

# deb http://apt.kubernetes.io/ kubernetes-**xenial main**



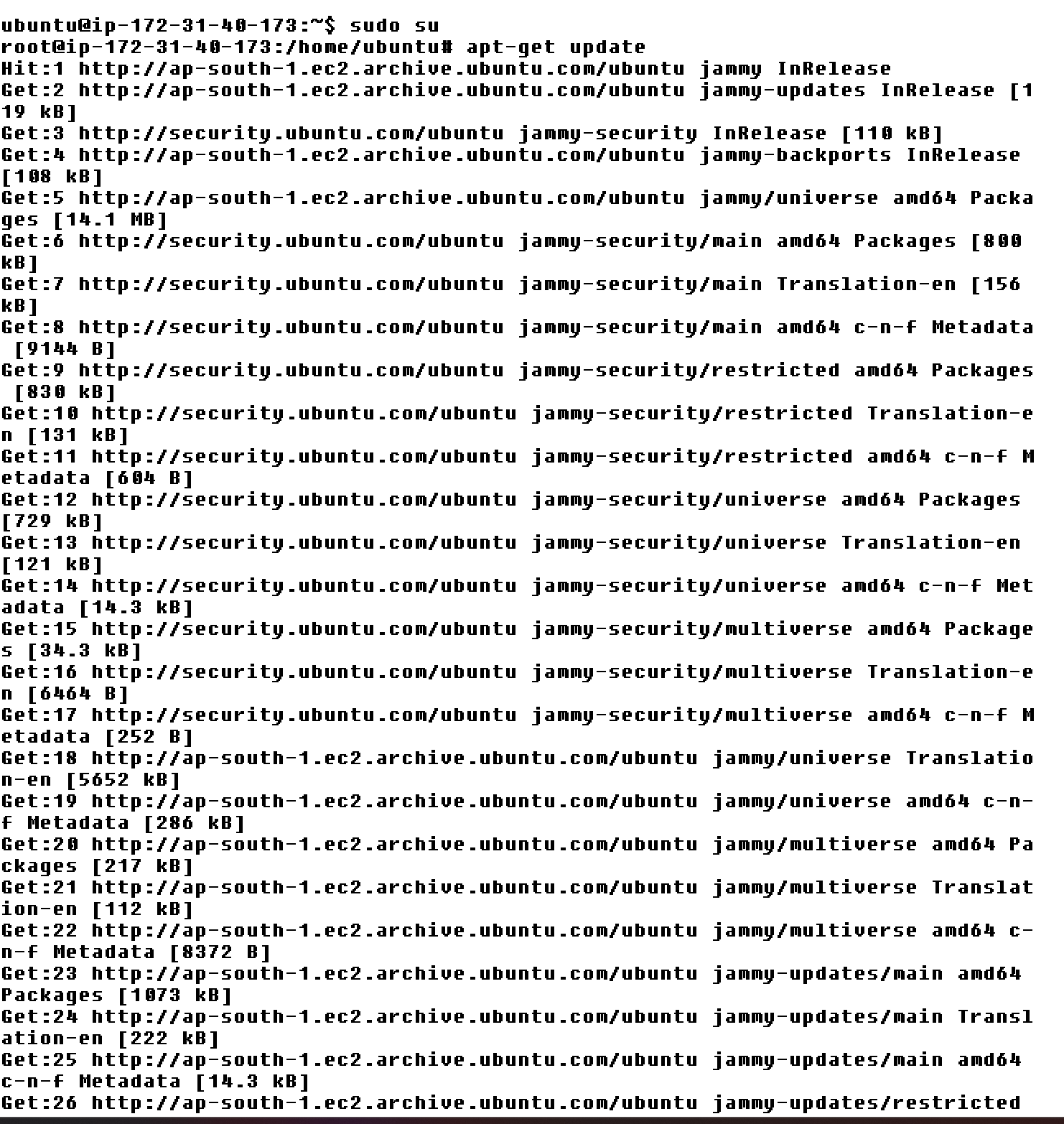
***Exit from nano ..***

***ctrl+X -*→ *caps+Y* → *Enter***



For getting update after closing the Nano editor.

# apt-get update



**Install all package on All 3 nodes**

# apt-get install -y kubelet kubeadm kubectl kubernetes-cni

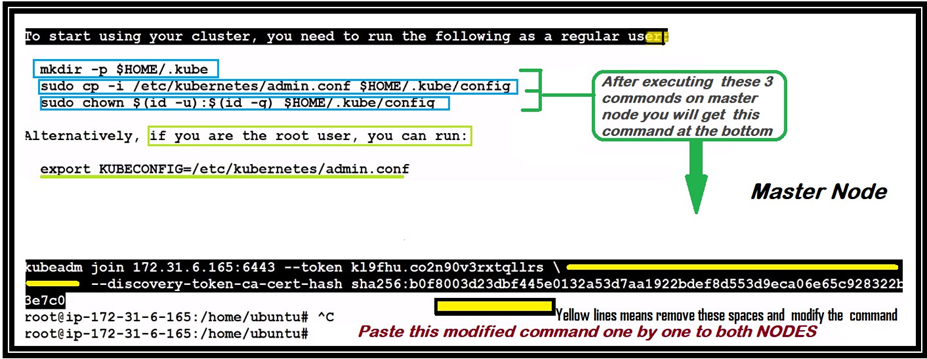
### *BOOTSTRAPPING IN THE MASTERNODE (in Master)*

**To initialize kubernetes cluster:**

**#** *kubeadm init*

Then you will get one long command started from **“kubeadm join**

**172.31.6.265:6443**  ….…… ***Copy the command and save on notepad***



#### Run this command in to nodes, then nodes will connect to the master

Create both **.Kube** and its parent directories (-p)

**#** Mkdir –p $HOME/.kube

**Copy configuration to kube directory (un-configured file):**

***#*** *mkdir -p $HOME/.kube*

***#*** *cp -i /etc/kubernetes/admin.conf $HOME/.kube/config*

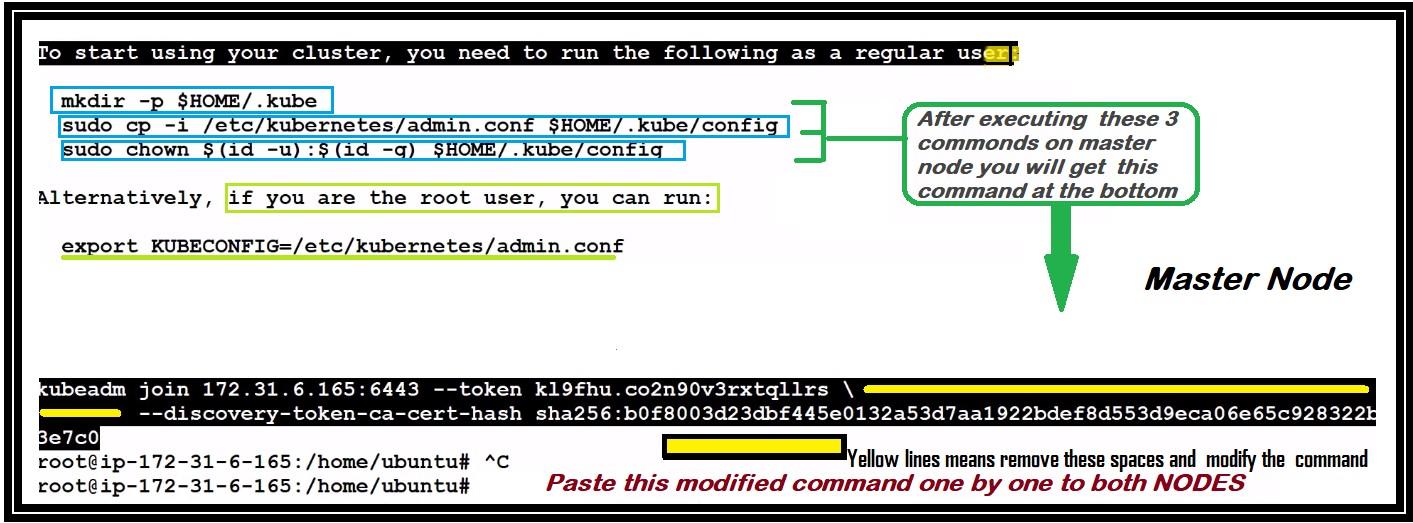
**Provide user permission to config file:**

**#***chown $(id -u):$(id -g) $HOME/.kube/config*

**Deploy FLANNEL node network for its repository Path.**

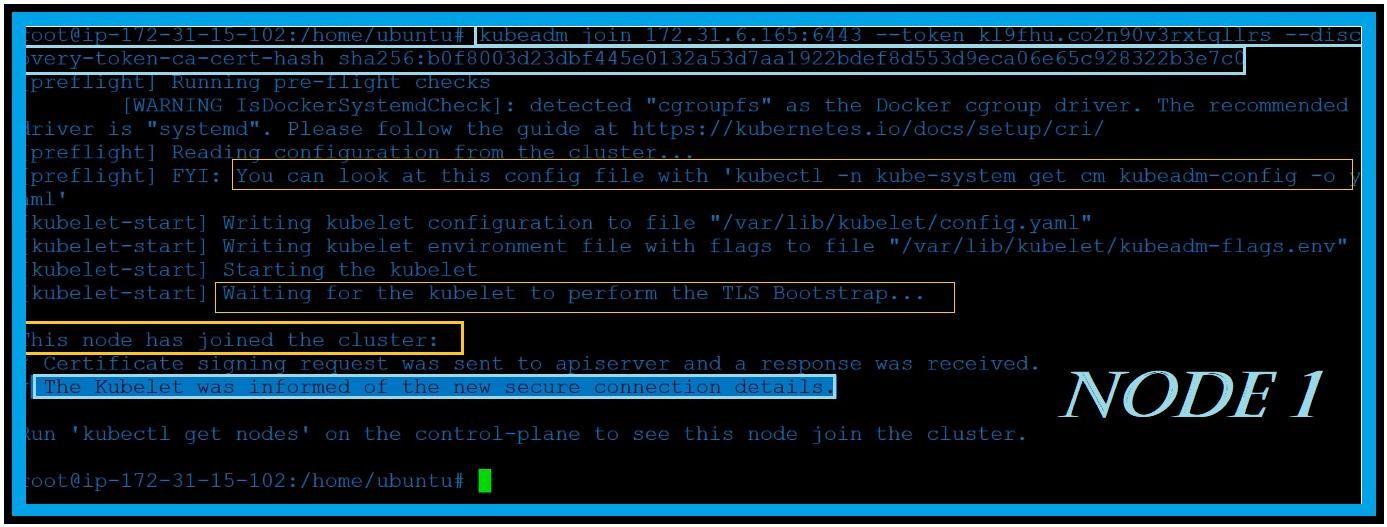
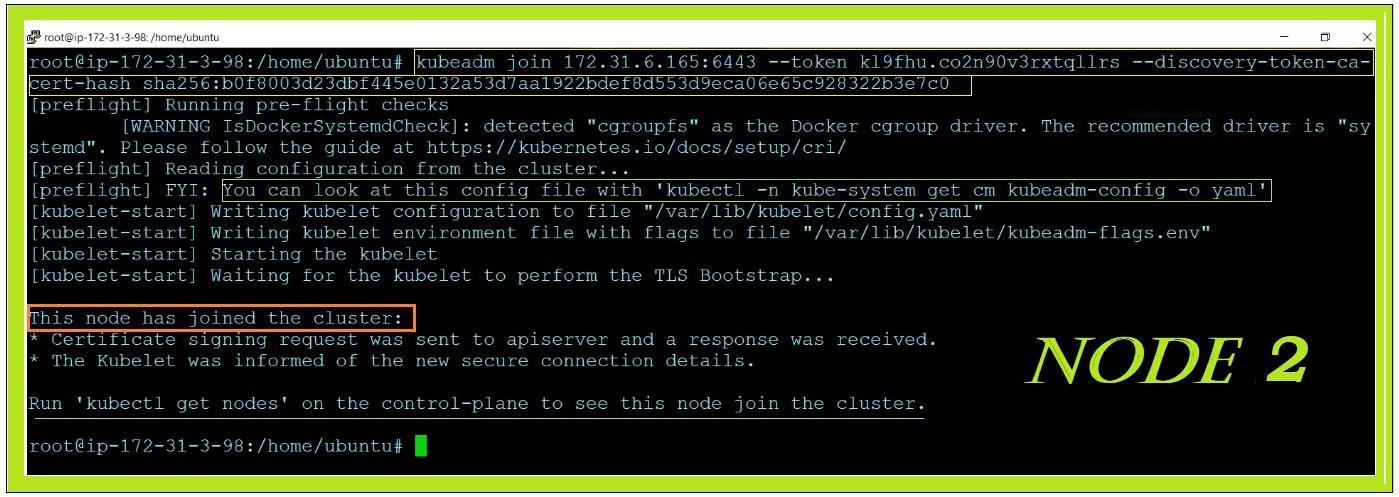
**Flannel is going to place a binary in each node.**

**Cluster role binding /flannel creation/flannel Configured.**



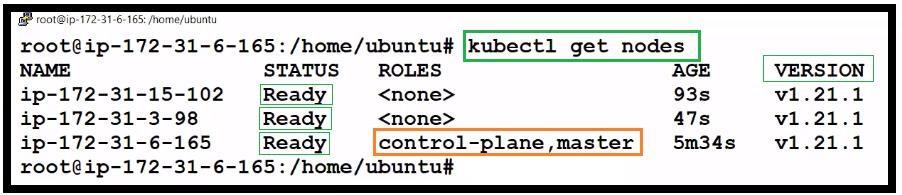
**Configuration worker node**

**Paste long command (provided by master) in both the nodes** e.g- kubeadm join 172.31.6.165:6443 --token kl9fhu.co2n90v3rxtqllrs --discovery-token-ca-cert-hash sha256:b0f8003d23dbf445e0132a53d7aa1922bdef8d553d9eca06e65c92832 2b

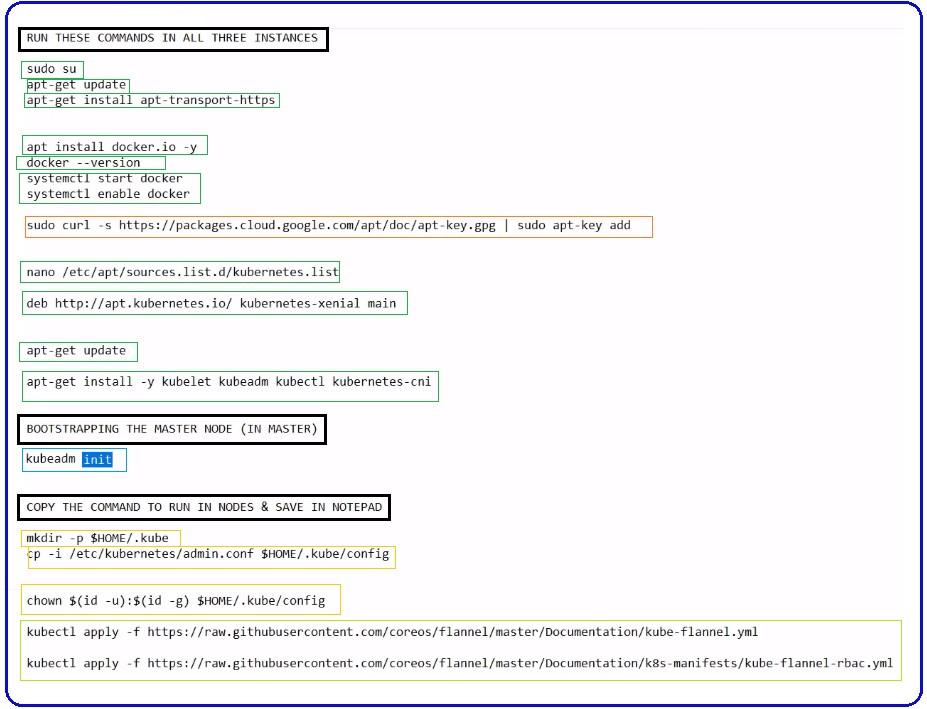


**To check the status of nodes, Go to master and run this command**

**Kubectl get nod**



***All commands in one frame***



***apt-get install apt-transport-https***

