ASSIGNMENT - 5

AIM: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms

LO2. To deploy single and multiple container applications and manage application deployments with rollouts in Kubernetes.

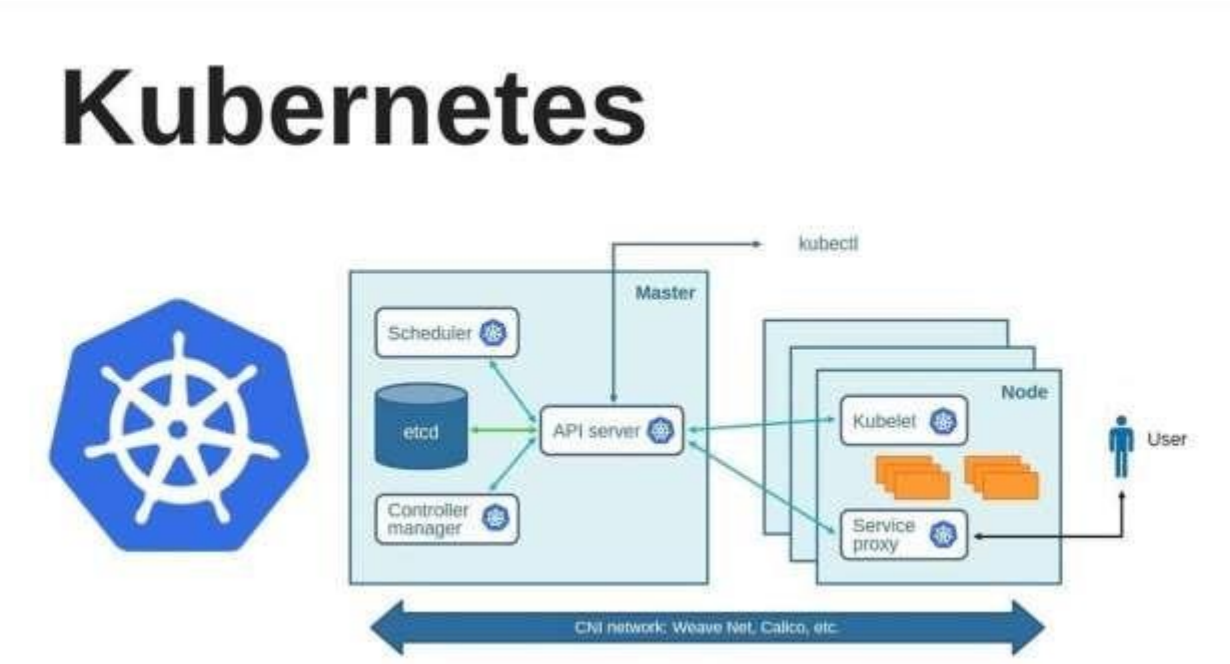
THEORY:

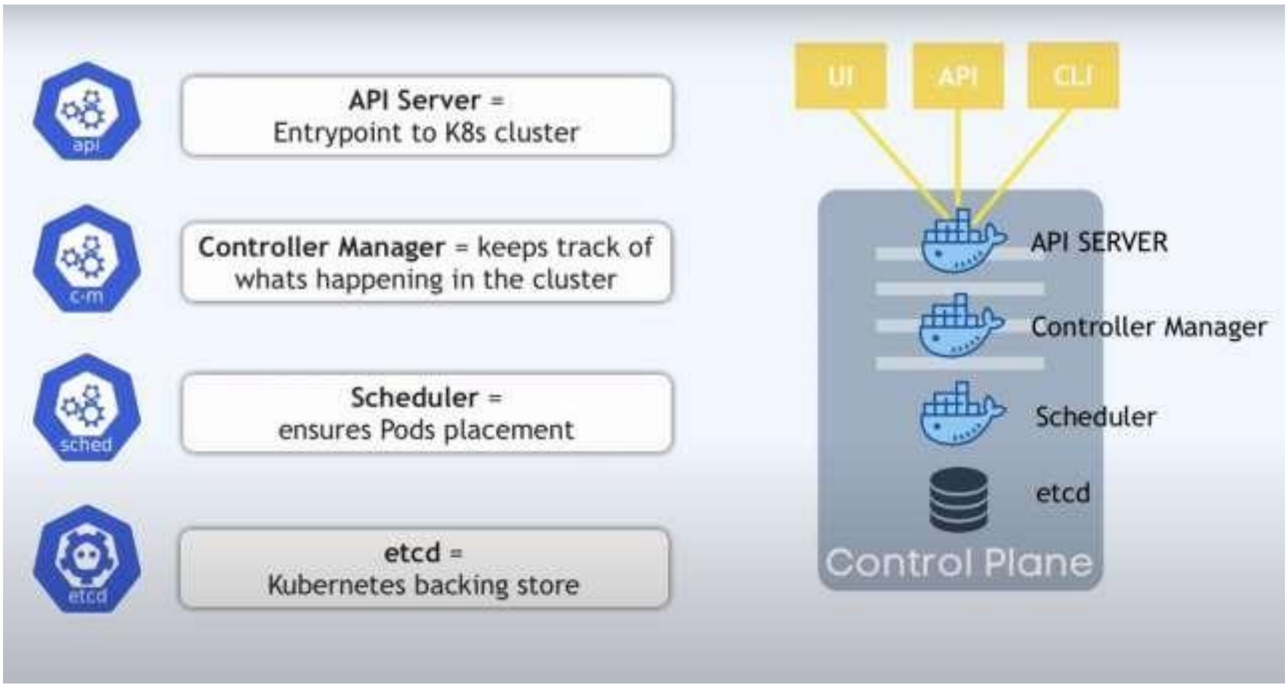
Kubernetes is an open-source container management tool that automates container deployment, scaling & load balancing.

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

All top cloud providers support Kubernetes. One popular name for Kubernetes is K8s.

ARCHITECTURE





Working with Kubernetes

- We create a Manifest (.yml) file
- Apply those to cluster (to master) to bring it into the desired state.
- POD runs on a node, which is controlled by the master.

Q.1 What are the various Kubernetes services running on nodes? Describe the role of each service.

In a Kubernetes cluster, several essential services run on nodes to manage containerized applications and maintain the overall health and functionality of the system. These services play crucial roles in orchestrating and maintaining container workloads. Here, we'll describe some of the core Kubernetes services running on nodes and their roles:

1. Kubelet:

- The Kubelet is one of the most critical components in a Kubernetes node. It is responsible for ensuring that containers are running in a Pod (the smallest deployable unit in Kubernetes).

- Kubelet communicates with the Kubernetes master to receive Pod specifications, which it then works to realize by starting, stopping, and managing containers.

2. Container Runtime:

- Kubernetes is designed to be container runtime-agnostic, which means it can work with various container runtimes such as Docker, containerd, or CRI-O.

- The container runtime is responsible for pulling container images, creating containers, and managing their lifecycles.

3. Kube Proxy:

- Kube Proxy is responsible for managing network connectivity and ensuring that network rules (such as Service LoadBalancer and Ingress) are correctly implemented.

- It maintains network rules on each node and facilitates communication between Pods.

4. Node Status:

- The Node Status service is responsible for reporting the health of the node to the master. It provides information about the node's resources, capacity, and conditions.

- The master can make scheduling decisions based on this information, ensuring that Pods are placed on nodes with adequate resources.

5. cAdvisor (Container Advisor):

- cAdvisor is an open-source resource usage and performance analysis agent. It is integrated into the Kubelet and provides valuable information about resource consumption for each container.

- This information is used for monitoring and scaling decisions, helping to ensure optimal resource allocation.

6. Node Problem Detector:

- This service is responsible for detecting and reporting node-level problems, such as hardware or OS issues. It can detect issues like disk failures, memory problems, or kernel crashes.

- Node Problem Detector helps in identifying problematic nodes and allows for proactive maintenance.

7. Cluster DNS:

- Kubernetes clusters usually have a built-in DNS service to provide DNS resolution for Pods. This service ensures that Pods can communicate with each other using their names.

- It maintains DNS records for all active Services and Pods in the cluster.

8. Node-local DNS Cache:

- This service caches DNS queries on a node to improve DNS resolution performance. It reduces the load on the Cluster DNS service.

9. Container Logging:

- While not a single service, container logging is a crucial part of node-level services. Containers often generate logs that need to be collected and centralized for monitoring, troubleshooting, and auditing purposes.

- Common solutions for container logging include Fluentd, Elasticsearch, and Kibana (EFK) or Fluentd, Logstash, and Kibana (ELK) stacks.

These are the fundamental services running on Kubernetes nodes. Each plays a distinct role in managing containers, networking, resource monitoring, and system health. Together, they ensure the reliable operation of containerized workloads in a Kubernetes cluster.

Q.2 What is Pod Disruption Budget (PDB)?

A Pod Disruption Budget (PDB) is a policy in Kubernetes that allows you to define constraints on how many Pods of a particular type (often referred to as a "workload") can be unavailable simultaneously during voluntary disruptions. These voluntary disruptions can be caused by actions like scaling down or draining nodes, which may result from maintenance tasks, upgrades, or other operational requirements.

The primary purpose of a Pod Disruption Budget is to ensure that critical or sensitive workloads are not disrupted beyond a specified limit, thus maintaining application stability and reliability. Here's a more detailed explanation of PDB:

Key Concepts:

1. Target Workload: A Pod Disruption Budget is associated with a specific set of Pods, typically defined by a label selector. These Pods make up the "target workload" for which the PDB is applied.

2. Min Available: The PDB specifies a minimum number (or percentage) of Pods from the target workload that must remain available during disruptions. For example, you can set a PDB to ensure that at least 50% of the Pods are always available.

3. Max Unavailable: This parameter defines the maximum number (or percentage) of Pods from the target workload that can be unavailable during disruptions. For instance, you might limit the unavailability to 1 Pod at a time.

Use Cases and Benefits:

- High Availability: PDBs are crucial for maintaining high availability for stateful applications and other workloads where individual Pods may have data that needs to be preserved.

- Rolling Updates: During rolling updates or node maintenance, PDBs help control the rate at which Pods are taken down and new ones are started. This ensures that your application continues to function within defined availability constraints.

- Stateful Workloads: StatefulSets and other controllers for stateful workloads use PDBs to ensure that scaling and updates do not violate data integrity and the order of operations.

How PDB Works:

1. Pod Deletion: When a disruption event occurs (e.g., during node maintenance or scaling down), Kubernetes checks the PDB associated with the target workload to see if the disruption is allowed.

2. Evaluation: The PDB evaluates the current state of the target workload and compares it to the defined Min Available and Max Unavailable constraints.

3. Decision: If the disruption event would result in violating these constraints, Kubernetes prevents the action (e.g., node drain) until it's safe to proceed without breaching the PDB.

Example:

Suppose you have a critical application with a PDB set to allow a maximum of 1 Pod to be unavailable. When a node running some of your Pods goes offline, Kubernetes will prevent additional Pods from being taken down until there is only 1 Pod remaining on the node. This guarantees that your application remains operational while still allowing maintenance to take place.

Q.3 What is the role of Load Balance in Kubernetes?

Load balancing is a fundamental component in Kubernetes that plays a crucial role in distributing network traffic across multiple Pods to ensure high availability, scalability, and reliability of applications. The main objectives of load balancing in Kubernetes are as follows:

1. Distribution of Incoming Traffic:

- When multiple instances of a Pod or a Service are running in a Kubernetes cluster, a load balancer evenly distributes incoming network traffic across these instances.

- This distribution ensures that no single instance becomes overwhelmed with requests, which improves the performance and reliability of the application.

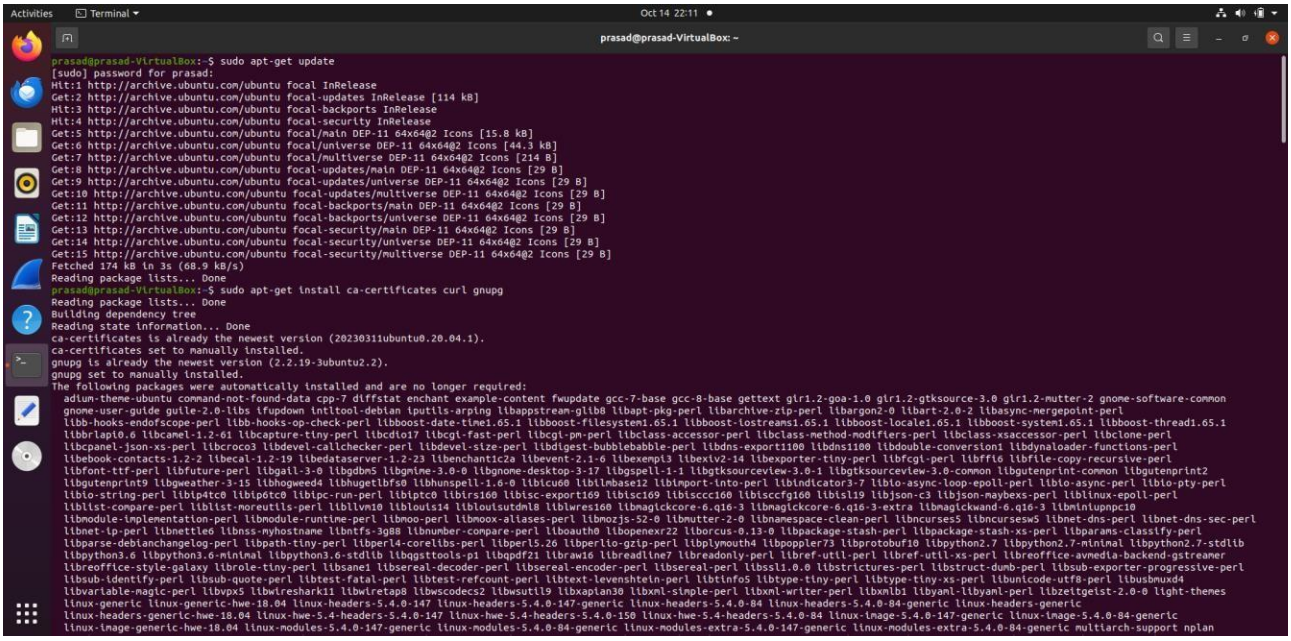
2. High Availability:

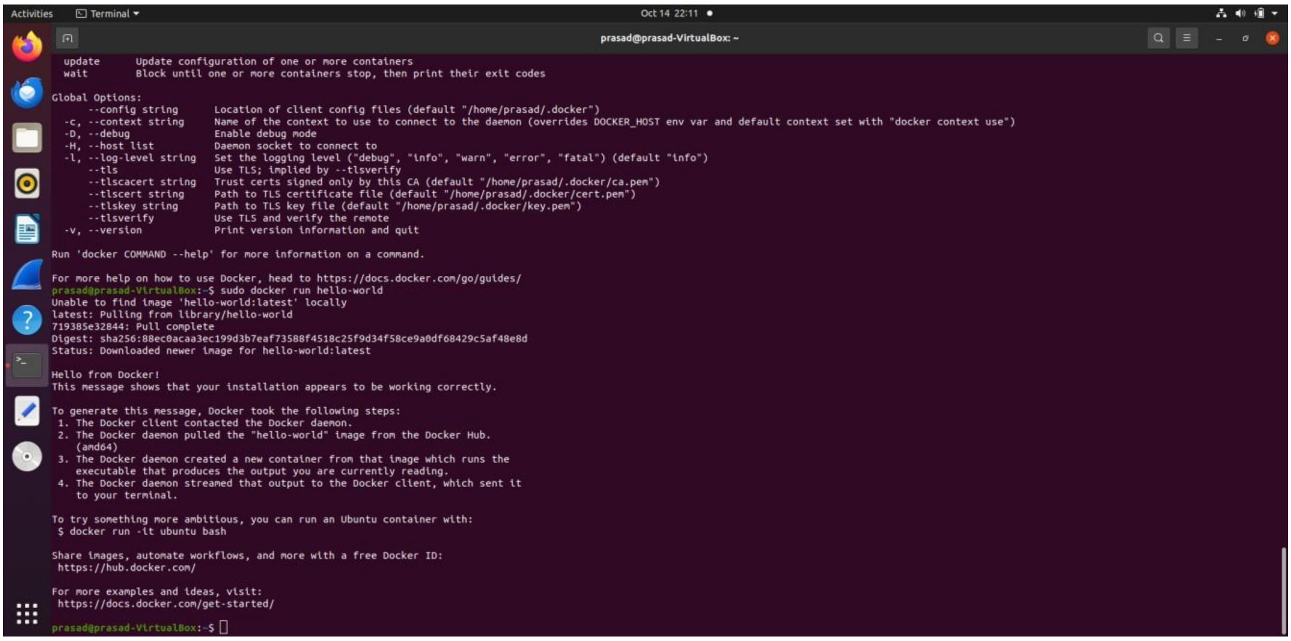
- Load balancing enhances the high availability of applications by distributing traffic to healthy Pods or nodes. If one Pod becomes unhealthy or a node fails, the load balancer redirects traffic to the remaining healthy instances.

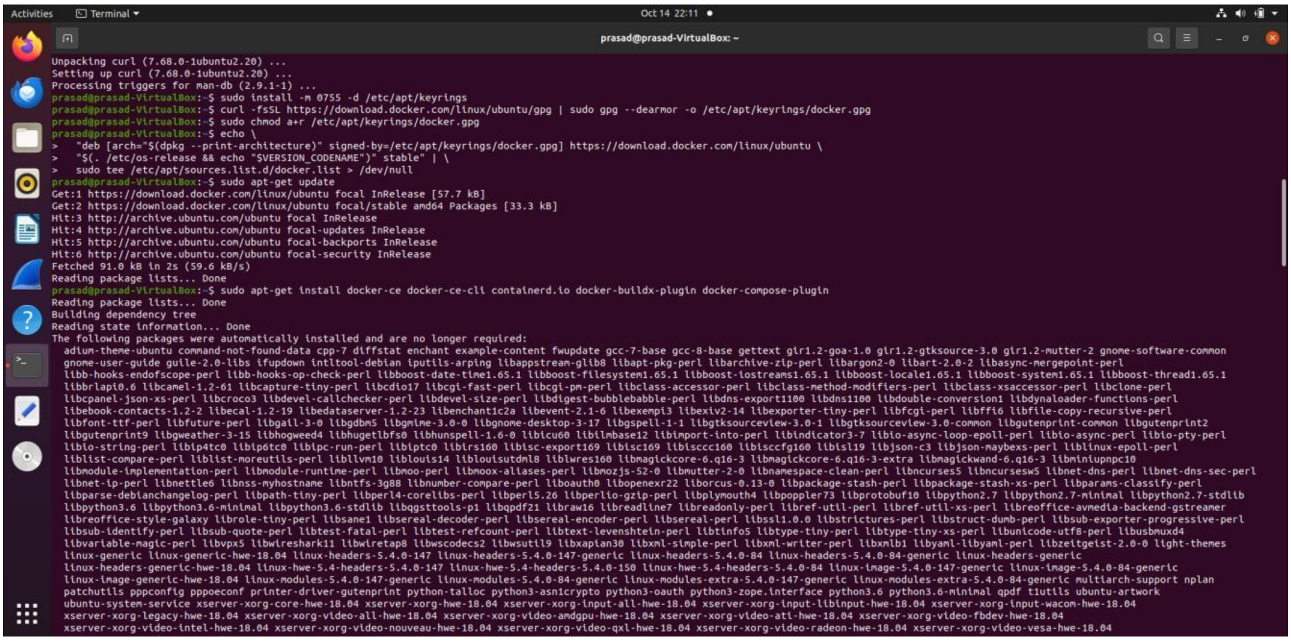
- This feature is essential for applications that require continuous availability, as it minimizes downtime due to failures.

INSTALLATION:

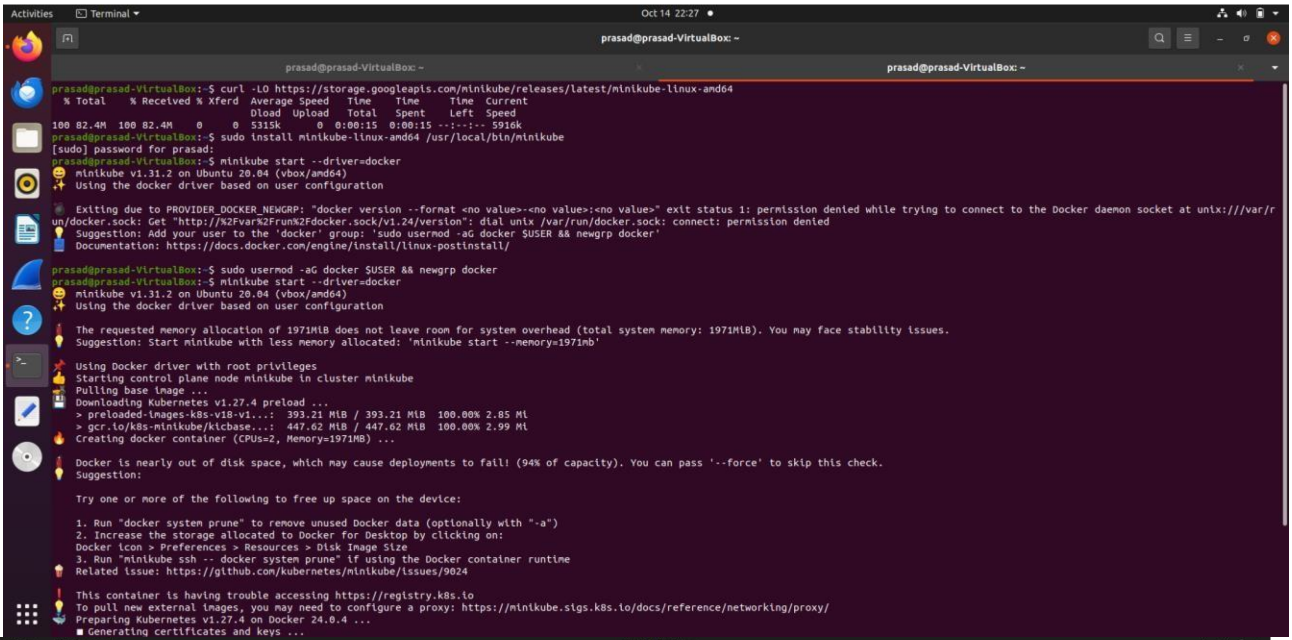
1. Install Docker

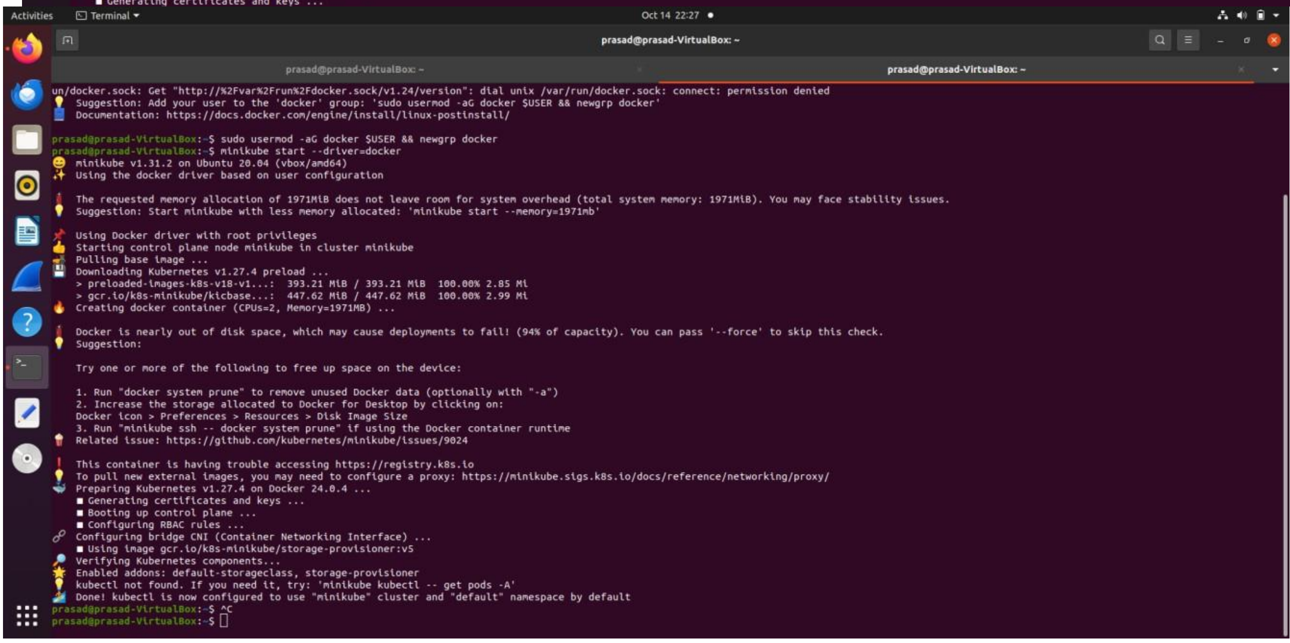




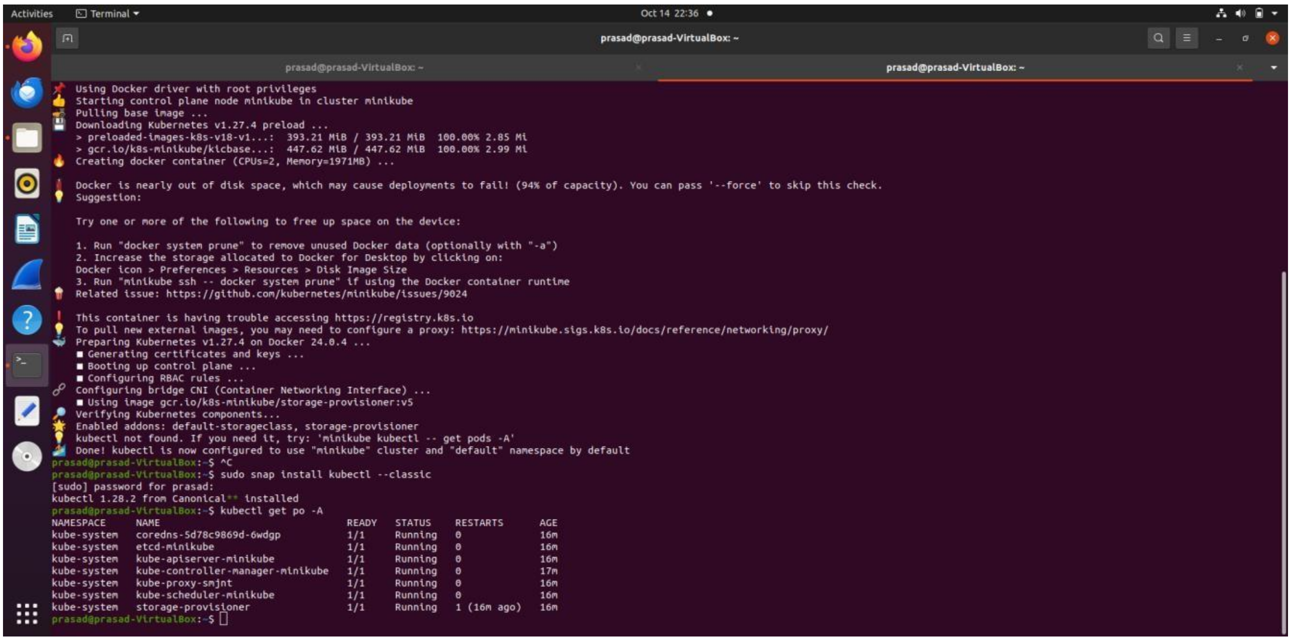


2. Install minikube using following commands

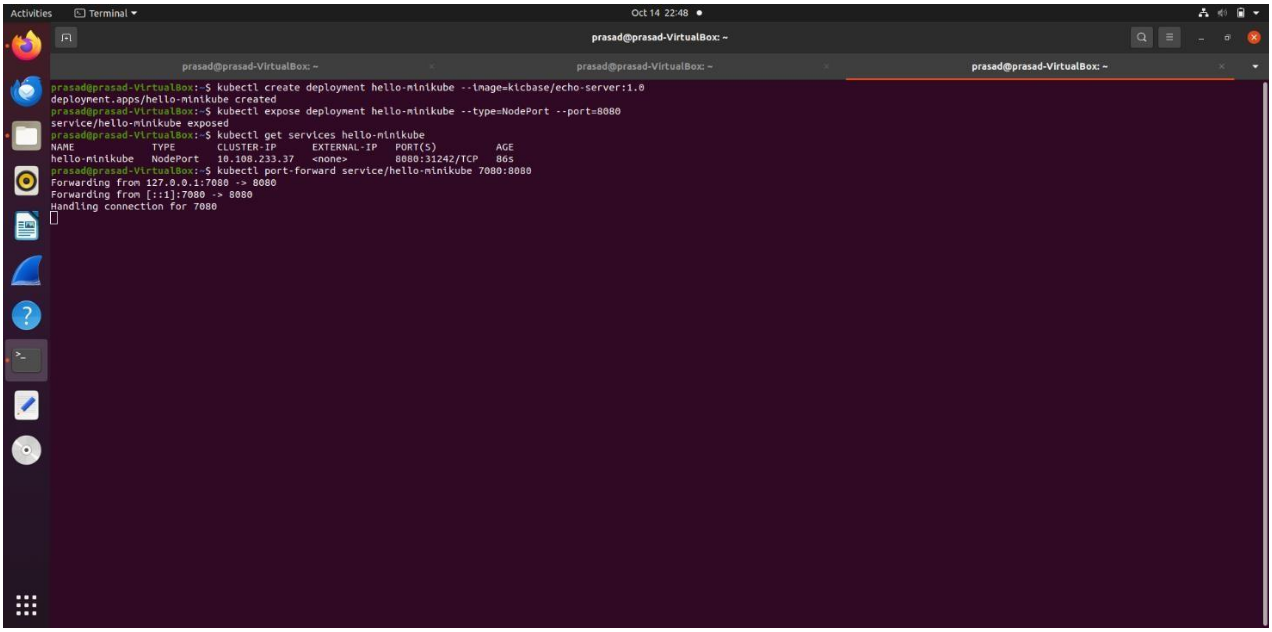


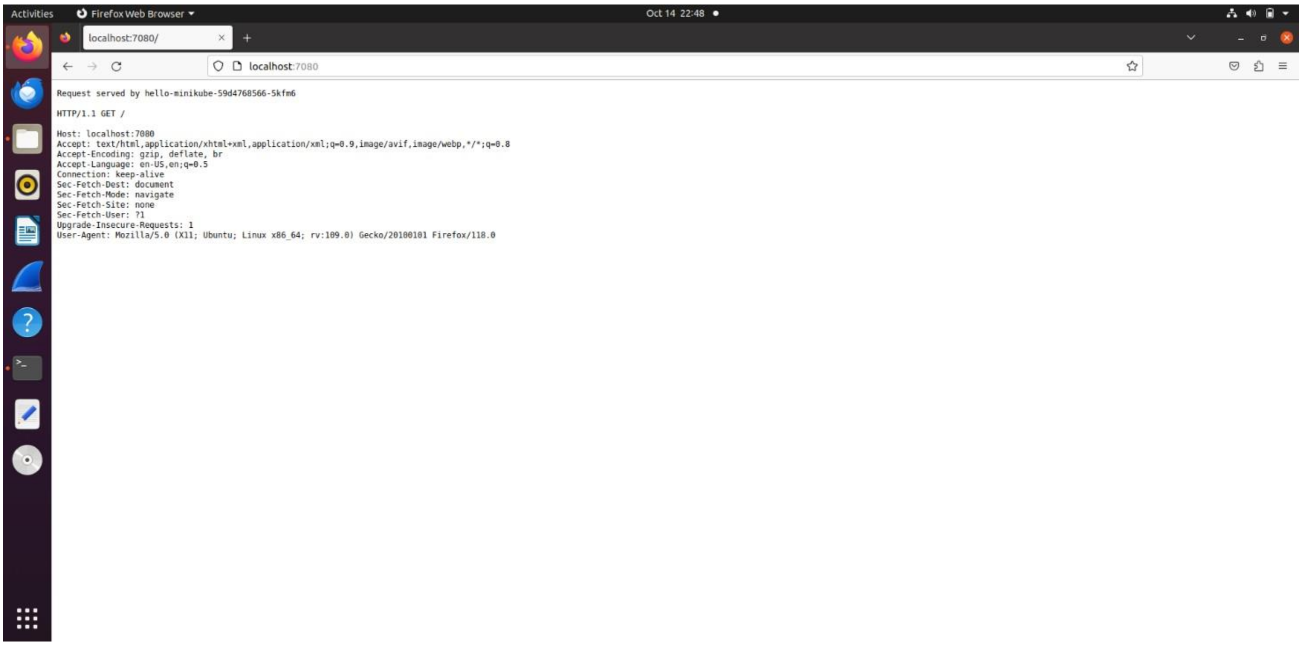


3. Install kubectl



4. Create a sample deployment.





CONCLUSION:

Here we studied Kubernetes cluster architecture in detail. Also we installed Kubernetes in ubuntu machine and created a sample deployment.