# Practical No. 03

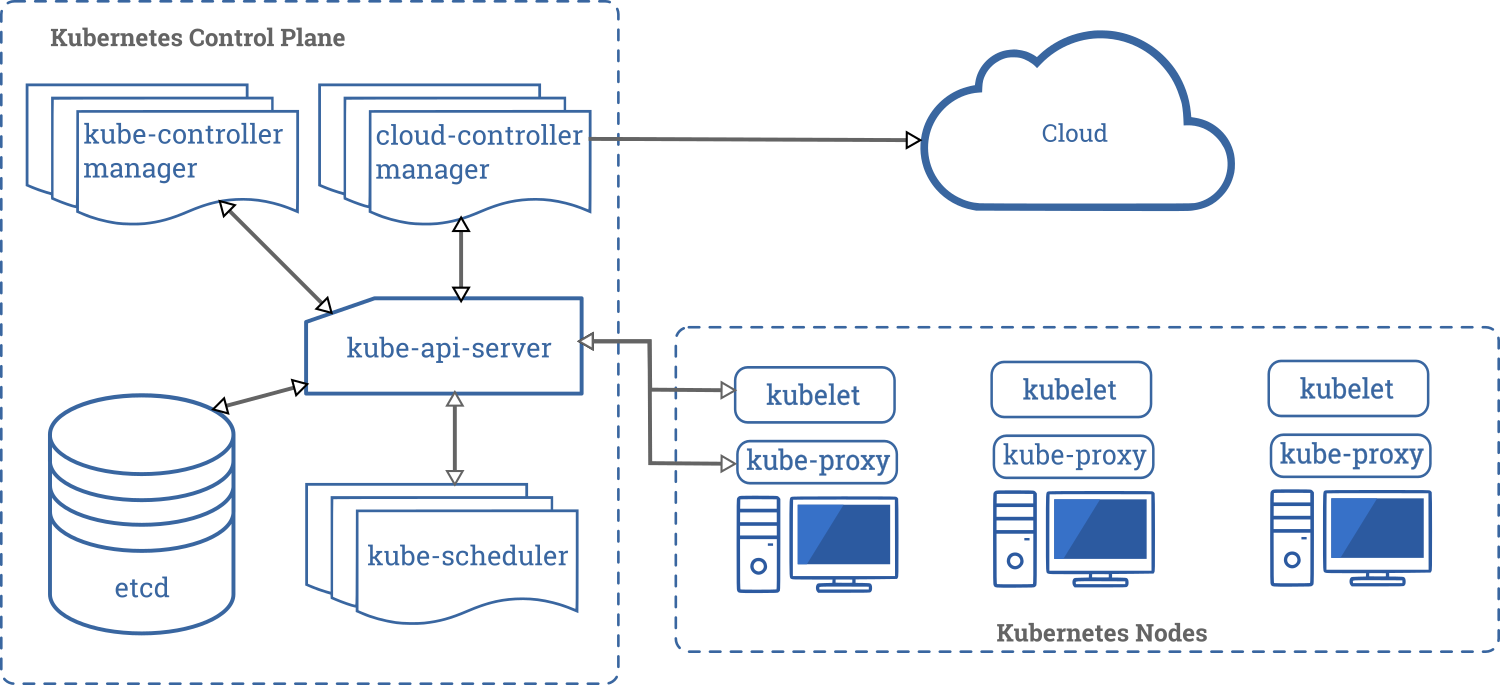
**Practical Name :** Understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

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

# Thoery :

* **What is Kubernetes?**

[Kubernetes](https://kubernetes.io/) is an open-source platform that was originally designed by Google and now maintained by the Cloud Native Computing Foundation. Kubernetes supports both declarative configuration and automation. It can help to automate deployment, scaling, and management of containerized workload and services.

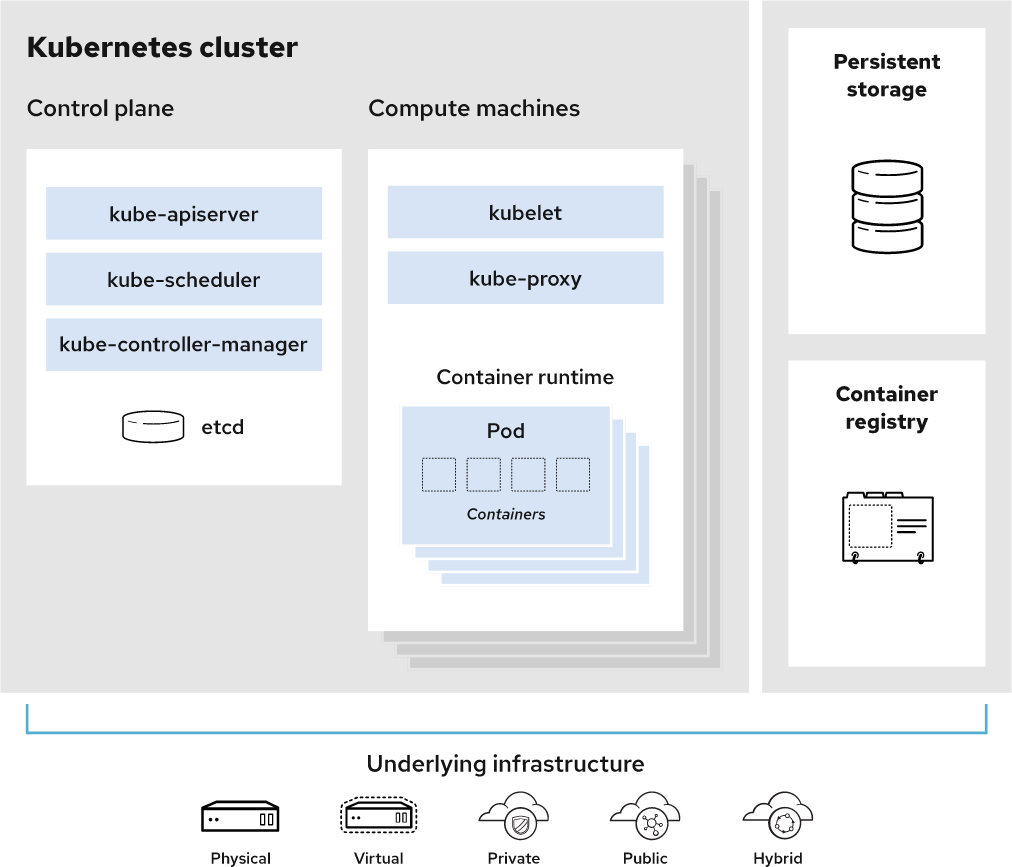


Kubernetes API helps to establish communication between users, cluster components, and external third-party components. Kubernetes control plane, and Nodes run on a group of nodes that together form the cluster. Application workload consists of one or more Pods which runs on Worker node(s). The control plane manages Pods and worker nodes.

Companies like Babylon, Booking.com, AppDirect extensively use [Kubernetes](https://geekflare.com/kubernetes-introduction/). Features

* + Service discovery and load balancing
  + Storage orchestration
  + Automated rollouts and rollbacks
  + Horizontal scaling
  + Secret and configuration management
  + Self-healing
  + Batch execution
  + IPv4/IPv6 dual-stack
  + Automatic bin packing

# Kubernetes Cluster Architecture :



* 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®](https://www.redhat.com/en/topics/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 and interacts with pods of containers running on the nodes.
* The Kubernetes control plane takes the commands from an administrator (or DevOps team) and relays those instructions to the compute machines.
* This handoff works with a multitude of services to automatically decide which node is best suited for the task. It then allocates resources and assigns the pods in that node to fulfill the requested work.
* The desired state of a Kubernetes cluster defines which applications or other workloads should be running, along with which images they use, which resources should be made available to them, and other such configuration details.
* From an infrastructure point of view, there is little change to how you manage containers. Your control over containers just happens at a higher level, giving you better control without the need to micromanage each separate container or node.
* Your work involves configuring Kubernetes and defining nodes, pods, and the containers within them. Kubernetes handles orchestrating the containers.
* Where you run Kubernetes is up to you. This can be on bare metal servers, virtual machines, public cloud providers, private clouds, and hybrid cloud environments. One of Kubernetes’ key advantages is it works on many different kinds of infrastructure.

# Kubernetes Services

Services are the Kubernetes way of configuring a proxy to forward traffic to a set of pods. Instead of static IP address-based assignments, Services use selectors (or labels) to define which pods uses which service. These dynamic assignments make releasing new versions or adding pods to a service really easy. Anytime a Pod with the same labels as a service is spun up, it’s assigned to the service.

# Kubernetes Networking

Networking Kubernetes has a distinctive networking model for cluster-wide, podto-pod networking. In most cases, the Container Network Interface (CNI) uses a simple overlay network (like Flannel) to obscure the underlying network from the pod by using traffic encapsulation (like VXLAN); it can also use a fully-routed solution like Calico. In both cases, pods communicate over a cluster-wide pod network, managed by a CNI provider like Flannel or Calico. Within a pod, containers can communicate without any restrictions. Containers within a pod exist within the same network namespace and share an IP. This means containers can communicate over localhost. Pods can communicate with each other using the pod IP address, which is reachable across the cluster. Moving from pods to services, or from external sources to services, requires going through kube-proxy.

# Steps to Install Kubernetes on Ubuntu Set up Docker

Step 1: Install Docker

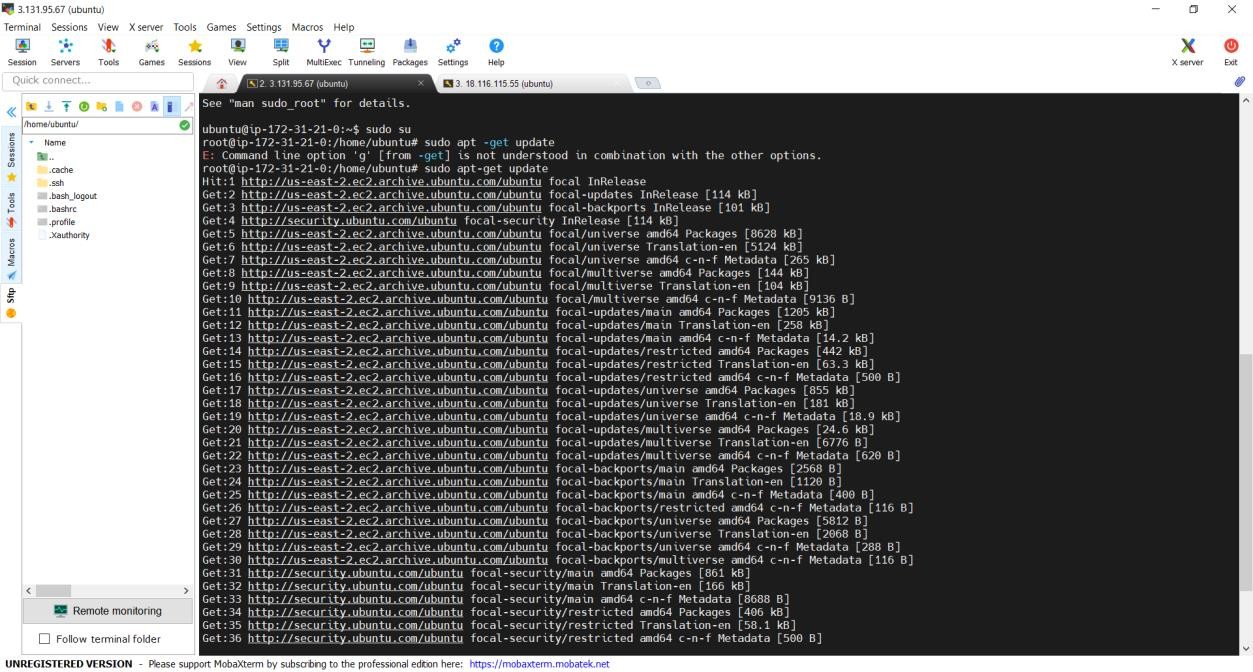
Kubernetes requires an existing Docker installation. If you already have Docker installed, skip ahead to Step 2.

If you do not have Kubernetes, install it by following these steps:

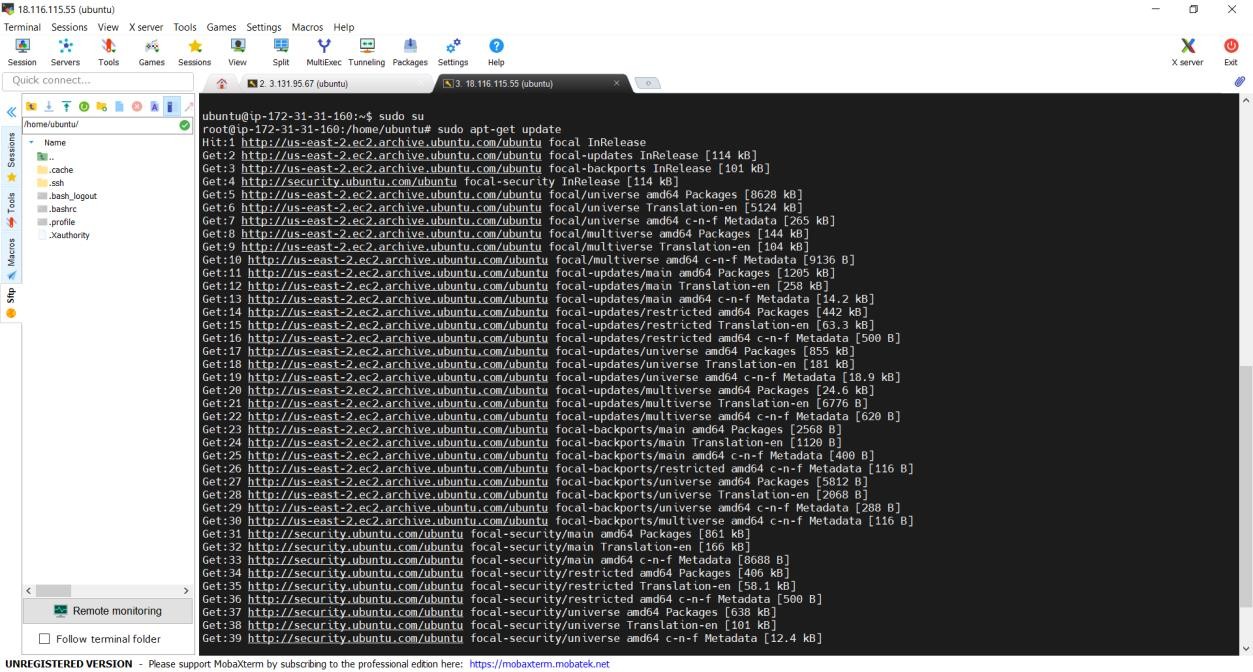
1. Update the package list with the command:

on-master&slave$sudo apt-get update

# Master :



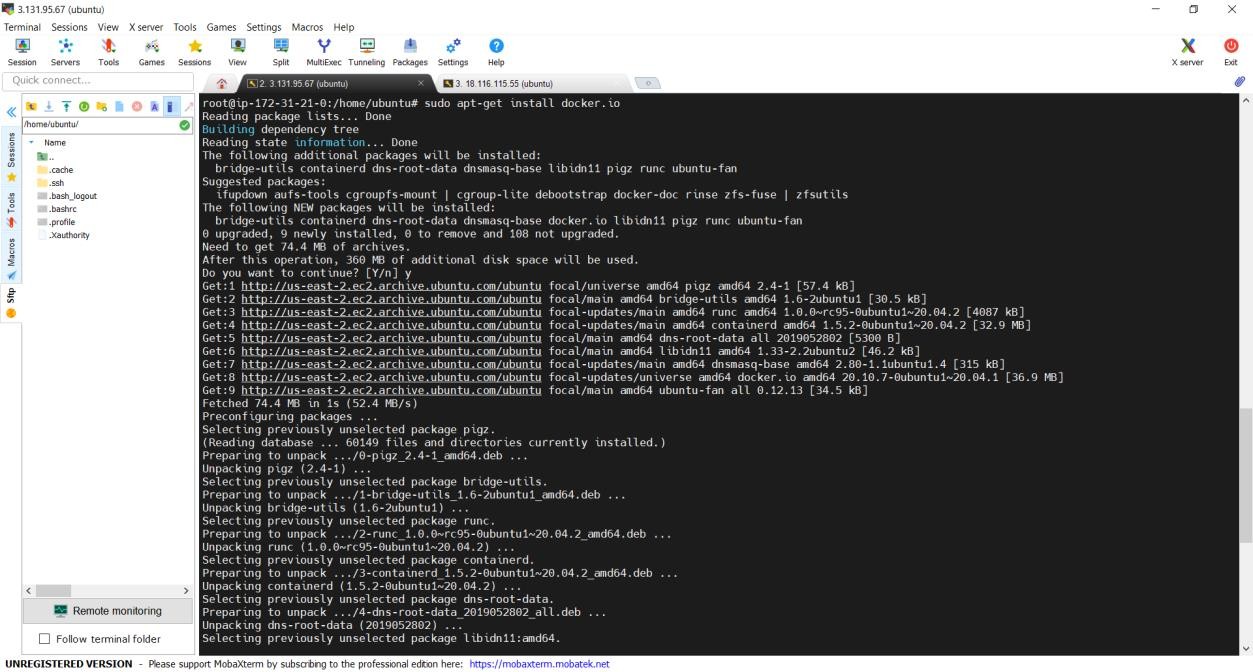
**Worker :**



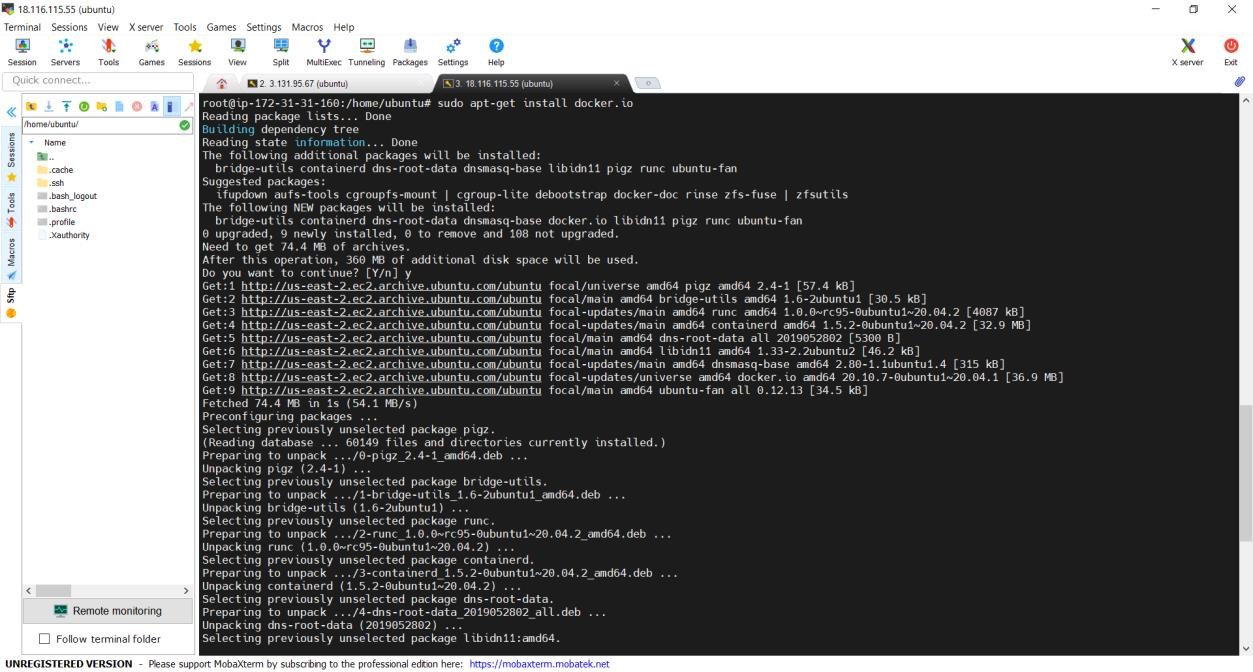
1. Next, install Docker with the command:

on-master&slave$sudo apt-get install docker.io

# Master :

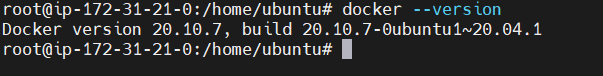


**Worker :**

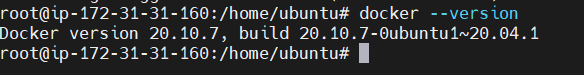


1. Check the installation (and version) by entering the following: on-master&slave$docker --version

# Master :



**Worker :**



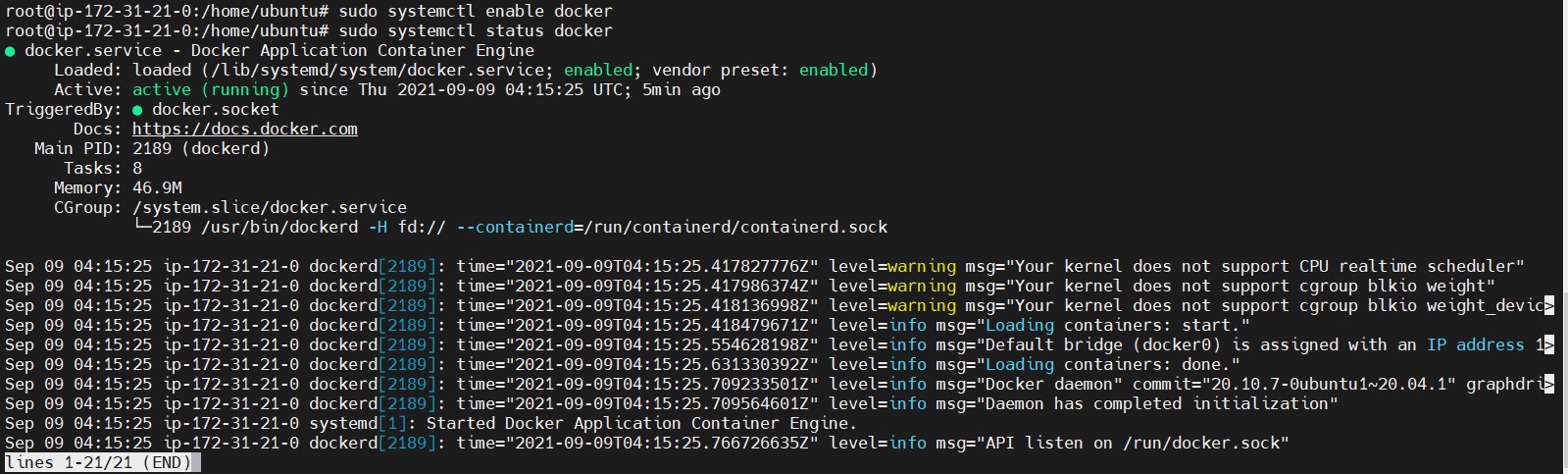
Step 2: Start and Enable Docker

1. Set Docker to launch at boot by entering the following: on-master&slave$sudo systemctl enable docker
2. Verify Docker is running:

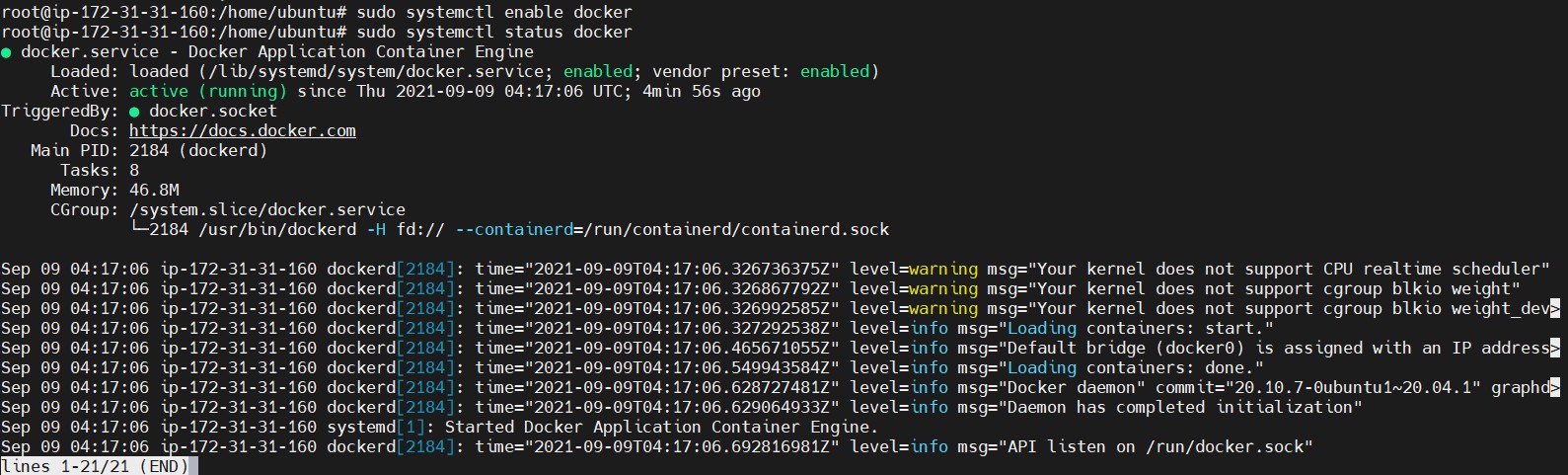
on-master&slave$sudo systemctl status docker To start Docker if it’s not running:

on-master&slave$sudo systemctl start docker

# Master :



**Worker :**



Install Kubernetes

Step 3: Add Kubernetes Signing Key

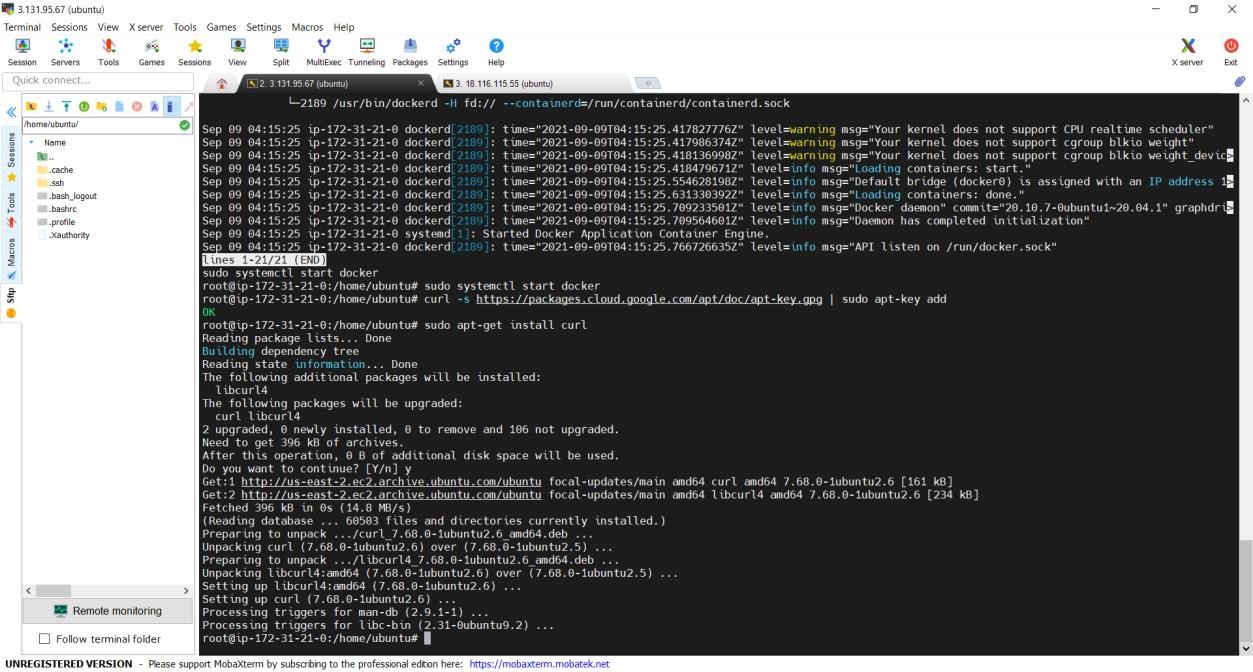
Since you are downloading Kubernetes from a non-standard repository, it is essential to ensure that the software is authentic. This is done by adding a signing key.

1.Enter the following to add a signing key:

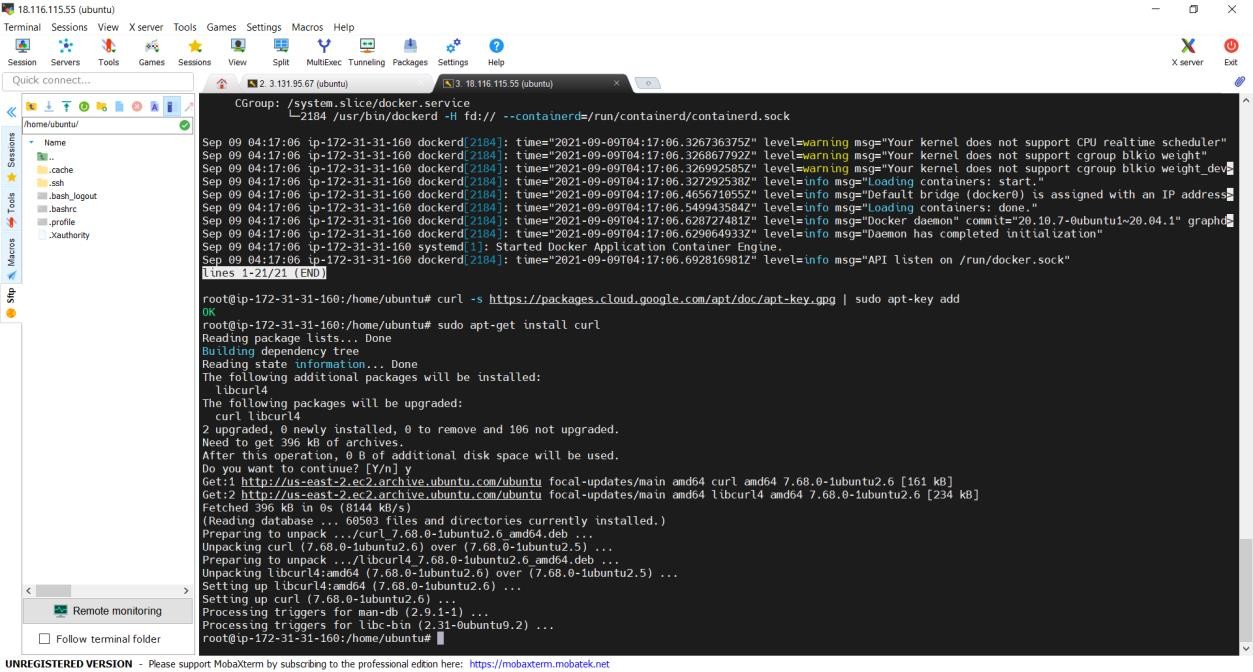
on-master&slave$curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt- key add

If you get an error that curl is not installed, install it with: on-master&slave$sudo apt-get install curl

# Master :



**Worker :**

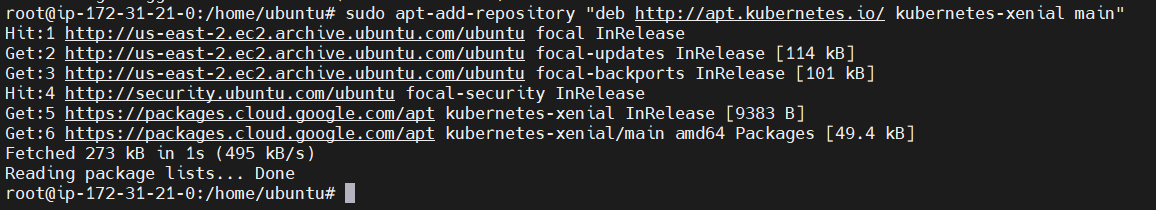


Step 4: Add Software Repositories

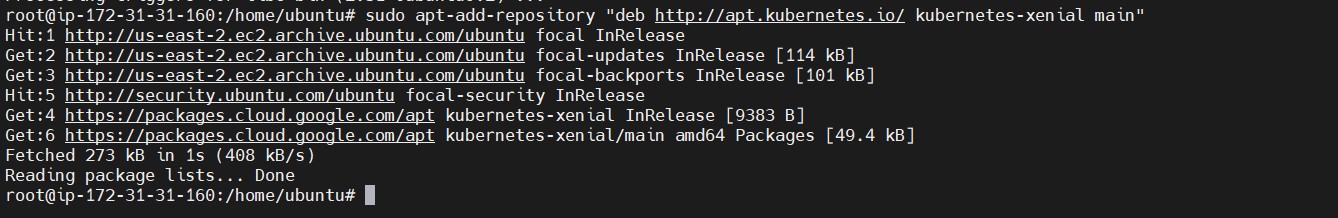
Kubernetes is not included in the default repositories. To add them, enter the following:

on-master&slave$sudo apt-add-repository "deb <http://apt.kubernetes.io/>kubernetes-xenial main"

# Master :



**Worker :**



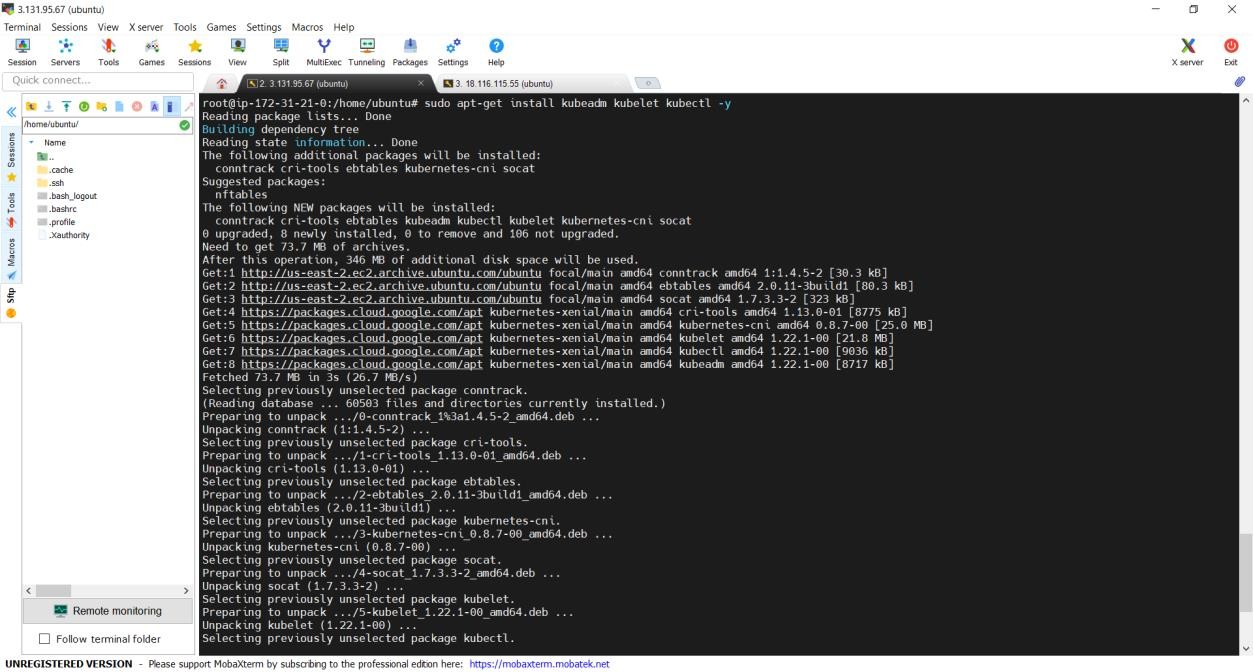
Step 5: Kubernetes Installation Tools

Kubeadm (Kubernetes Admin) is a tool that helps initialize a cluster. It fast-tracks setup by using community-sourced best practices. Kubelet is the work package, which runs on every node and starts containers. The tool gives you command-line access to clusters.

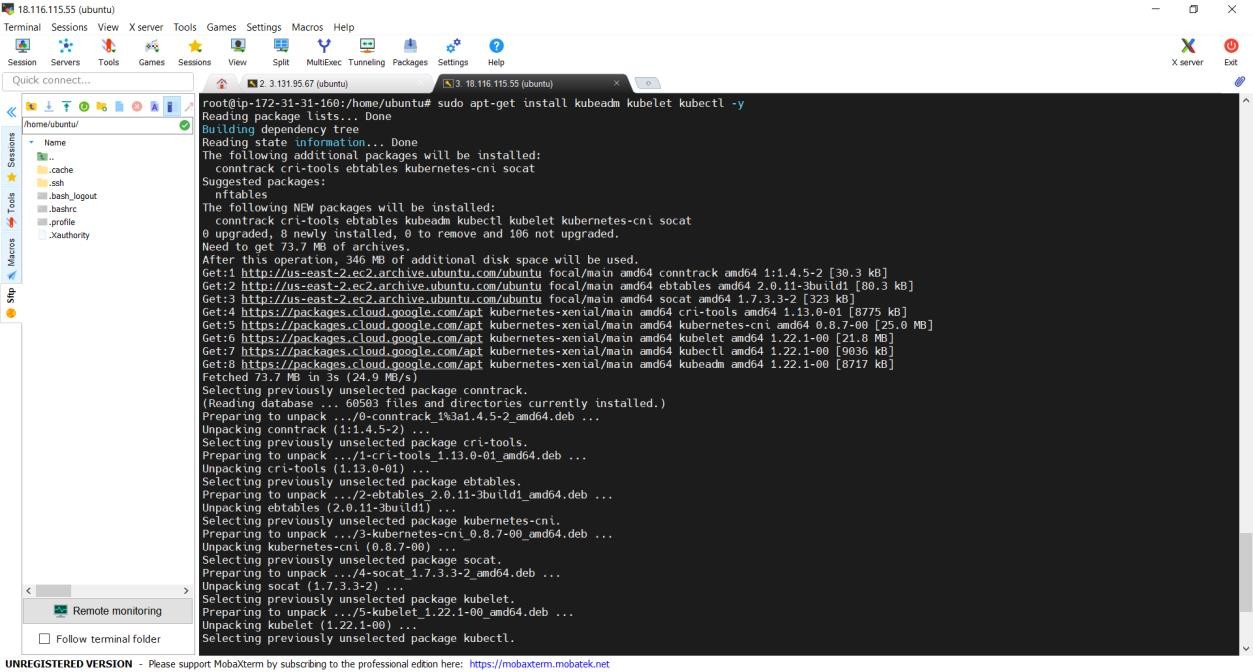
1. Install Kubernetes tools with the command:

on-master&slave$sudo apt-get install kubeadm kubelet kubectl -y on-master&slave$sudo apt-mark hold kubeadm kubelet kubectl Allow the process to complete.

# Master :



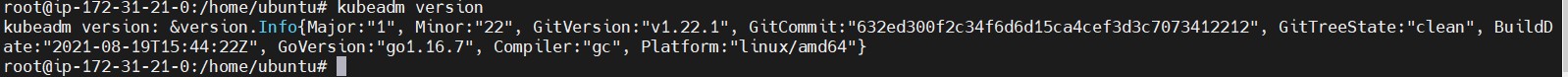
**Worker :**



1. Verify the installation with:

on-master&slave$kubeadm version

# Master :



**Worker :**



Kubernetes Deployment

Step 6: Begin Kubernetes Deployment

Start by disabling the swap memory on each server:

on-master&slave$sudo swapoff –a

Step 7: Assign Unique Hostname for Each Server Node

Decide which server to set as the master node. Then enter the command: on-master$sudo hostnamectl set-hostname master-node

Next, set a worker node hostname by entering the following on the worker server: on-slave$sudo hostnamectl set-hostname worker01

# Master :



**Worker :**



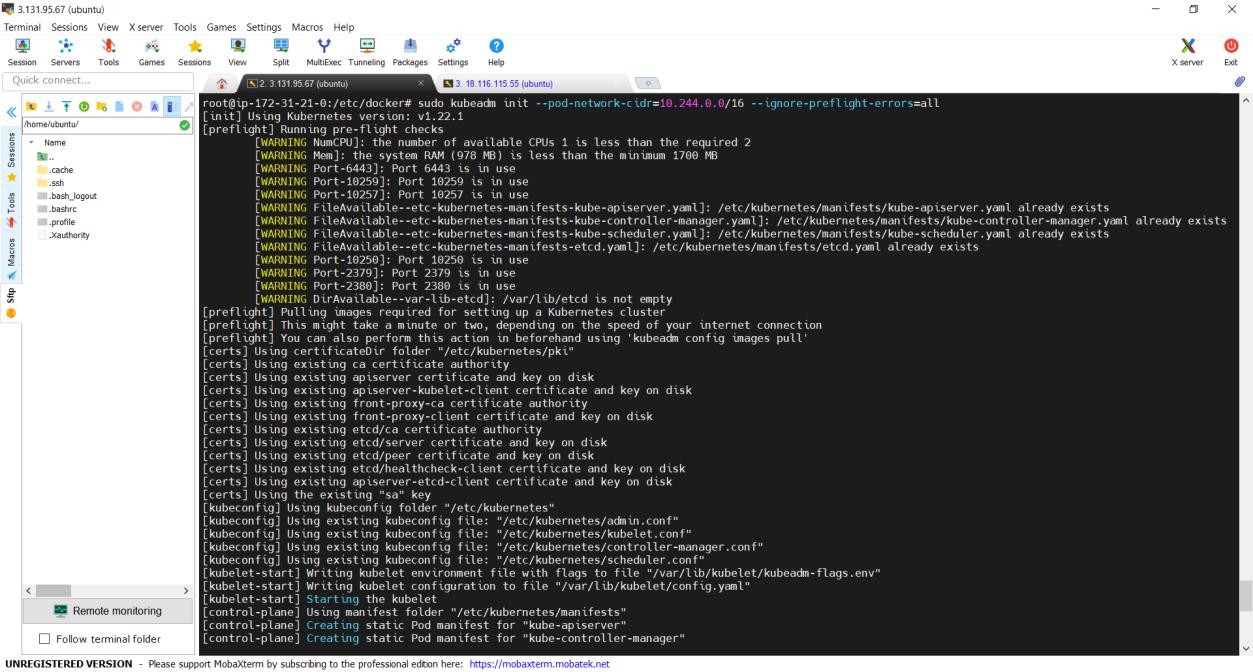
Step 8: Initialize Kubernetes on Master Node

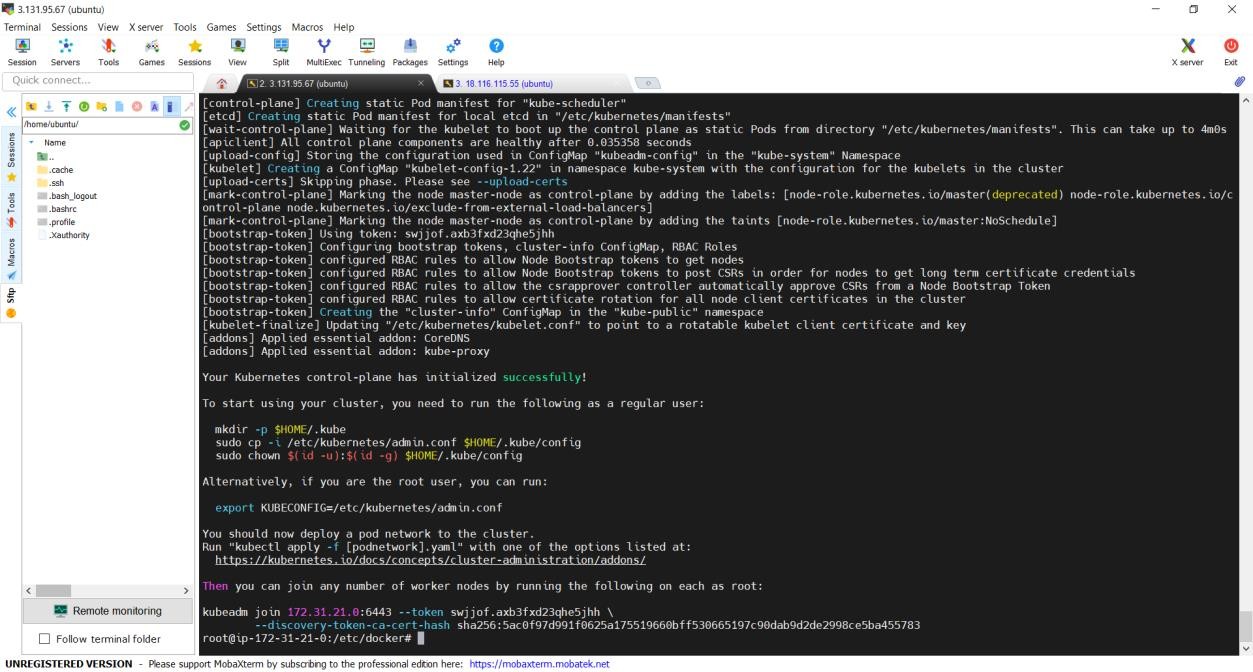
Switch to the master server node, and enter the following:

on-master$sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all

Once this command finishes, it will display a kubeadm join message at the end. Make a note of the whole entry. This will be used to join the worker nodes to the cluster.

# Master :

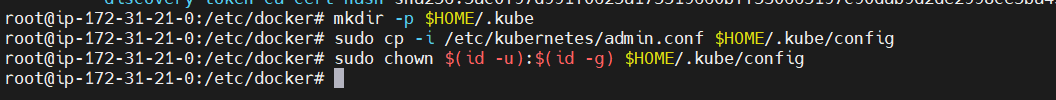




Next, enter the following to create a directory for the cluster:

kubernetes-master:~$ mkdir -p $HOME/.kube

kubernetes-master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config kubernetes-master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config



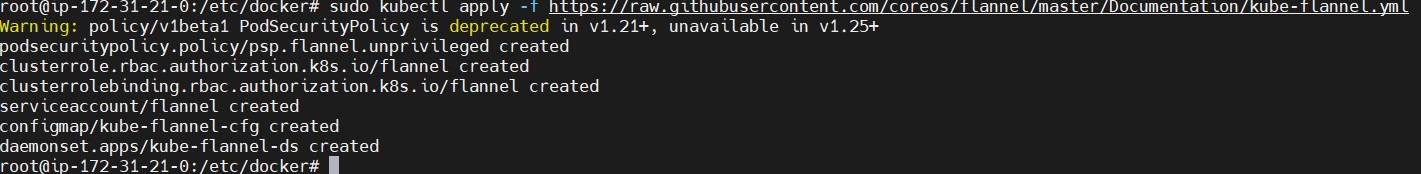
Step 9: Deploy Pod Network to Cluster

A Pod Network is a way to allow communication between different nodes in the cluster. This tutorial uses the flannel virtual network.

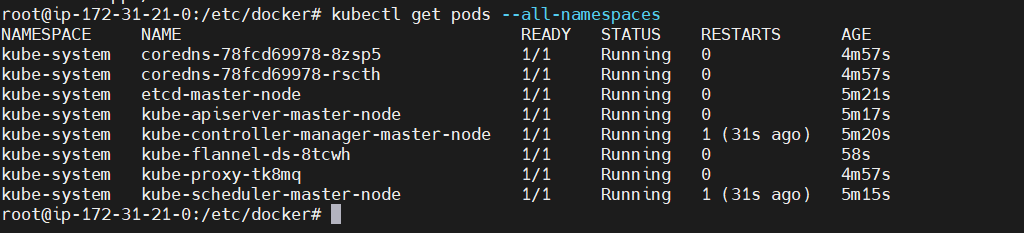
Enter the following:

kubernetes-master:~$ sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Allow the process to complete.



Verify that everything is running and communicating: kubernetes-master:~$ kubectl get pods --all-namespaces



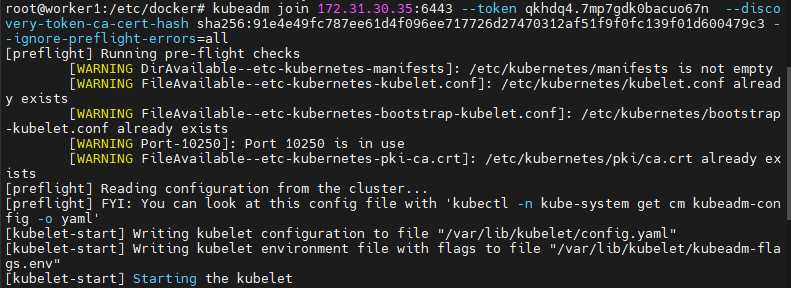
Step 10: Join Worker Node to Cluster

As indicated in Step 7, you can enter the kubeadm join command on each worker node to connect it to the cluster.

Switch to the worker01 system and enter the command you noted from Step 7:

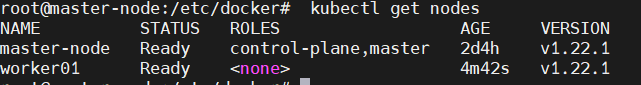
kubeadm join 172.31.30.35:6443 --token qkhdq4.7mp7gdk0bacuo67n --discovery-token-ca- cert-hash sha256:91e4e49fc787ee61d4f096ee717726d27470312af51f9f0fc139f01d600479c3

# Worker :



Wait a few minutes; then you can check the status of the nodes. Switch to the master server, and enter:

kubernetes-master:~$ kubectl get nodes



The system should display the worker nodes that you joined to the cluster.

Output

master Ready master 1d v1.14.0 worker1 Ready <none> 1d v1.14.0

If all of your nodes have the value Ready for STATUS, it means that they’re part of the cluster and ready to run workloads.

Now that your cluster is verified successfully, let’s schedule an example Nginx application on the cluster.

# Running An Application on the Cluster

You can now deploy any containerized application to your cluster. To keep things familiar, let’s deploy Nginx using Deployments and Services to see how this application can be deployed to the cluster. You can use the commands below for other containerized applications as well, provided you change the Docker image name and any relevant flags (such as ports and volumes).

Still within the master node, execute the following command to create a deployment named nginx:

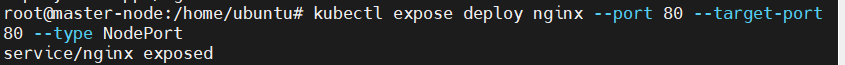
kubernetes-master:~$kubectl create deployment nginx --image=nginx



A deployment is a type of Kubernetes object that ensures there’s always a specified number of pods running based on a defined template, even if the pod crashes during the cluster’s lifetime. The above deployment will create a pod with one container from the Docker registry’s Nginx Docker Image.

Next, run the following command to create a service named nginx that will expose the app publicly. It will do so through a NodePort, a scheme that will make the pod accessible through an arbitrary port opened on each node of the cluster:

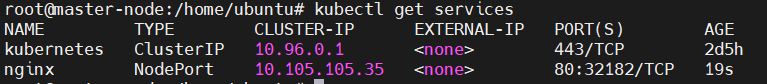
kubernetes-master:~$kubectl expose deploy nginx --port 80 --target-port 80 --type NodePort



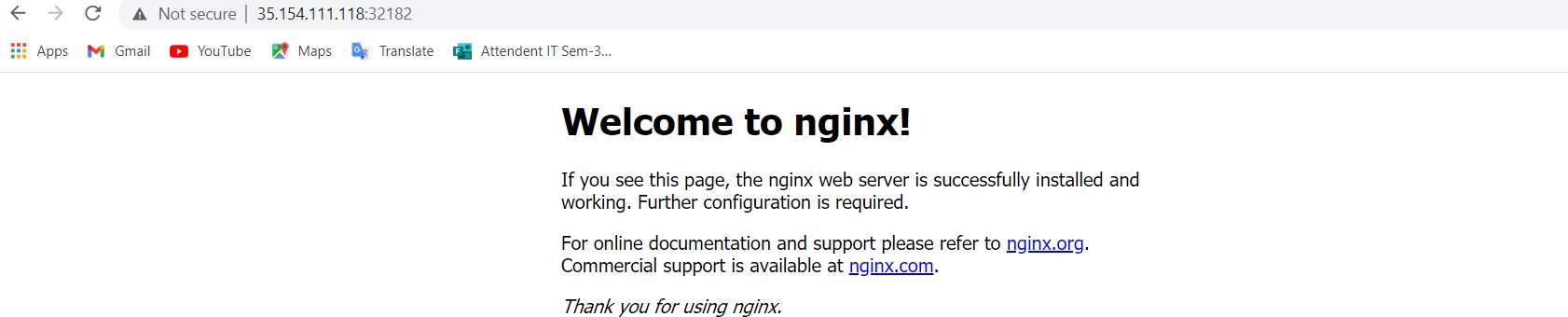
Services are another type of Kubernetes object that expose cluster internal services to clients, both internal and external. They are also capable of load balancing requests to multiple pods, and are an integral component in Kubernetes, frequently interacting with other components.

Run the following command:

kubernetes-master:~$kubectl get services



From the third line of the above output, you can retrieve the port that Nginx is running on. Kubernetes will assign a random port that is greater than 30000 automatically, while ensuring that the port is not already bound by another service.



# Conclusion :

**Successfully** understood the Kubernetes Cluster Architecture, installed and Spinned Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.