Cloud Computing Lab

Assignment - 8: Deploy a **Web Application** using **Kubernetes**

Name – Ansfred Ryan D'Souza

Roll Number – 322002 (TY B)

PR Number – 22010961

Aim:

Set up Single Node Kubernetes Cluster with Minikube and Deploy a Web App on Kubernetes Cluster.

Theory:

Q) What is Kubernetes?

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.

Q) Why do you need Kubernetes and what it can do?

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

Kubernetes provides you with:

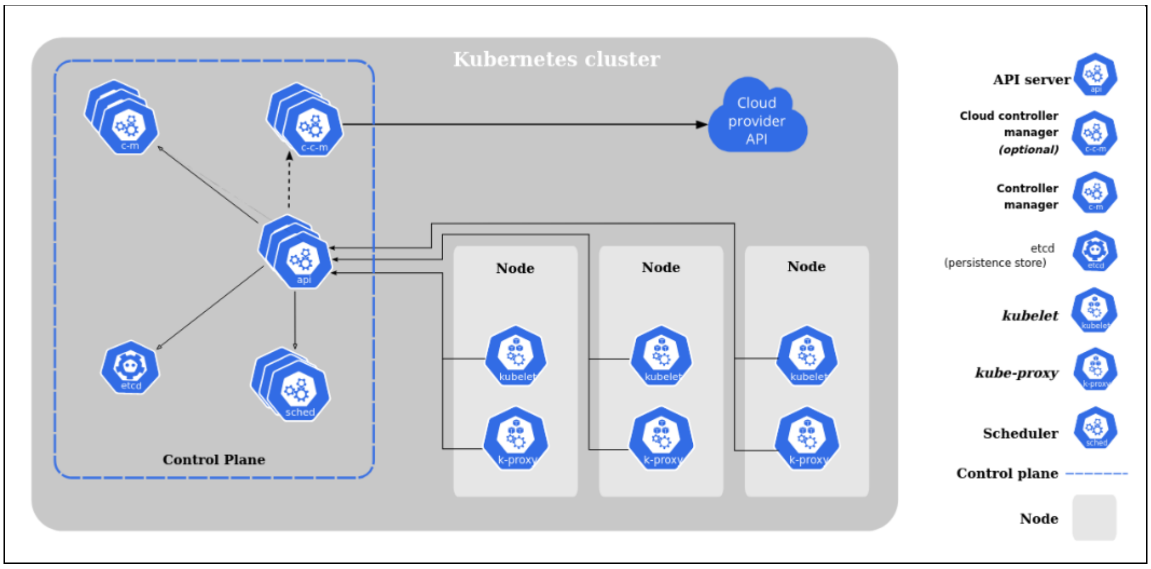
- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

Q) Kubernetes Components

When you deploy Kubernetes, you get a cluster. A Kubernetes cluster consists of a set of worker machines, called nodes , that run containerized applications. Every cluster has at least one worker node.

The worker node(s) hosts the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

This document outlines the various components you need to have a complete and working Kubernetes cluster. Here's the diagram of a Kubernetes cluster with all the components tied together.



Q) Kubernetes Architecture on AWS

- **Control Panel Components:**

The control panel's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied). Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine. See Building High-Availability Clusters for an example multi-master-VM setup.

1. **kube-apiserver**

   The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane. The main implementation of a Kubernetes API server is kube-apiserver . kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

2. **etcd**

   Consistent and highly available key value store used as Kubernetes' backing store for all cluster data. If your Kubernetes cluster uses etcd as its backing store, make sure you have a backup plan for those data. You can find in-depth information about etcd in the official documentation.

3. **kube-scheduler**

   Control plane component that watches for newly created Pods with no assigned node and selects a node for them to run on. Factors considered for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

4. **kube-controller-manager**

   Control Plane component that runs controller processes. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

   These controllers include:

   - **Node controll**er: Responsible for noticing and responding when nodes go down.

- **Replication controller**: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- **Endpoints controller**: Populates the Endpoints object (that is, joins Services & Pods).
- **Service Account & Token controllers**: Create default accounts and API access tokens for new namespaces.

5. **cloud-controller-manager**

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API and separates out the components that interact with that cloud platform from components that just interact with your cluster.

The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.

As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process. You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

The following controllers can have cloud provider dependencies:

- **Node controller**: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding.
- **Route controller**: For setting up routes in the underlying cloud infrastructure.
- **Service controller**: For creating, updating, and deleting cloud provider load balancers.

- **Node Components:**

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

1. **kubelet**

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

2. **kube-proxy**

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

3. **Container runtime**

The container runtime is the software that is responsible for running containers. Kubernetes supports several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

- **Pods:**

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers , with shared storage/network resources, and a specification for how to run the containers. A Pod's contents are always co-located and co-scheduled and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are analogous to cloud applications executed on the same logical host.

- **Using Pods:**

Usually, you don't need to create Pods directly, even singleton Pods. Instead, create them using workload resources such as Deployment or Job . If your Pods need to track state, consider the StatefulSet resource.

Pods in a Kubernetes cluster are used in two main ways:

- Pods that run a single container. The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.

  - Pods that run multiple containers that need to work together. A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit of service—for example, one container serving data stored in a shared volume to the public, while a separate sidecar container refreshes or updates those files. The Pod wraps these containers, storage resources, and an ephemeral network identity together as a single unit.

- **Service:**

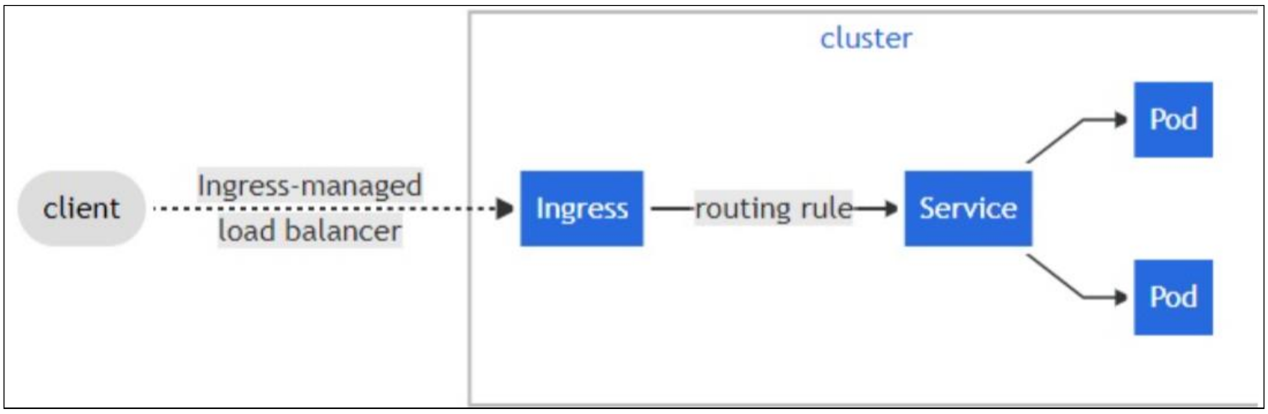
An abstract way to expose an application running on a set of Pods as a network service.
With Kubernetes you don't need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods and can load-balance across them.

- **Ingress:**

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

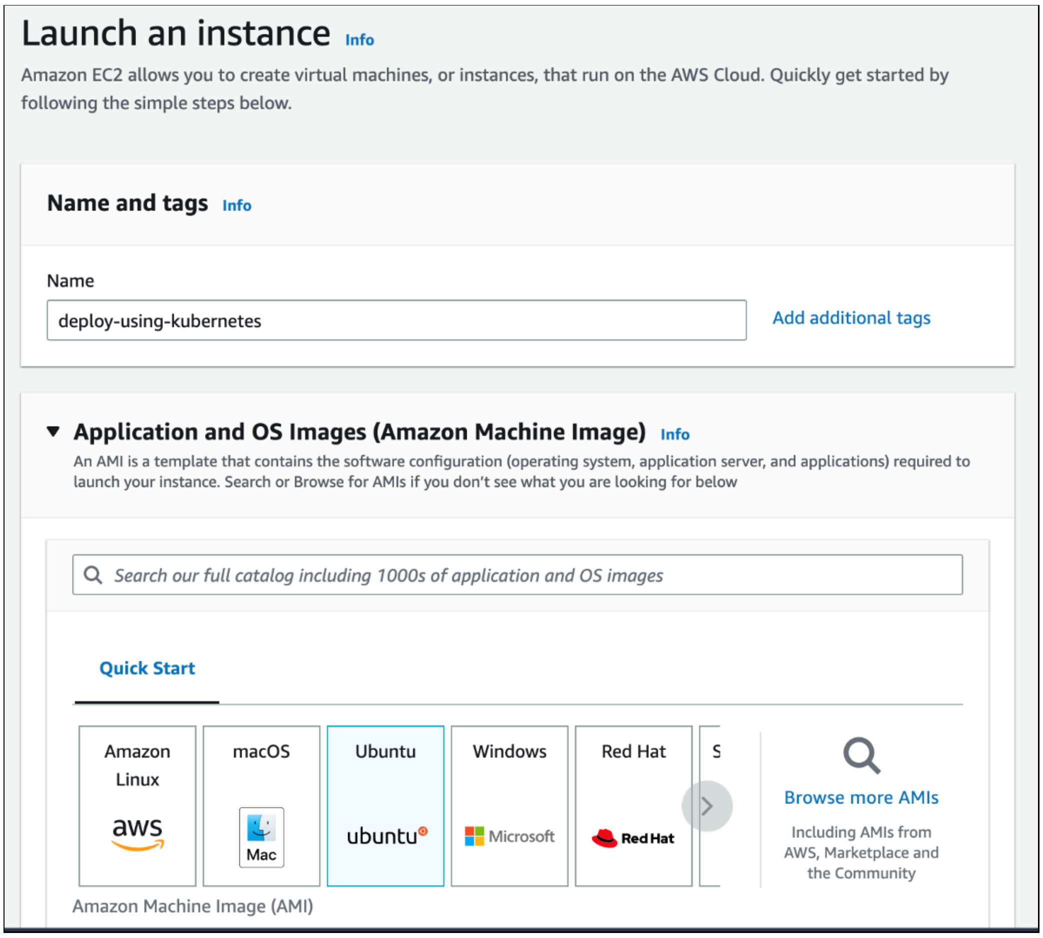
Some terminologies:

  - Node: A worker machine in Kubernetes, part of a cluster.
  - Cluster: A set of Nodes that run containerized applications managed by Kubernetes. For this example, and in most common Kubernetes deployments, nodes in the cluster are not part of the public internet.
  - Edge router: A router that enforces the firewall policy for your cluster. This could be a gateway managed by a cloud provider or a physical piece of hardware.
  - Cluster network: A set of links, logical or physical, that facilitate communication within a cluster according to the Kubernetes networking model .
  - Service: A Kubernetes Service that identifies a set of Pods using label selectors. Unless mentioned otherwise, Services are assumed to have virtual IPs only routable within the cluster network.



An Ingress may be configured to give Services externally reachable URLs, load balance traffic, terminate SSL / TLS, and offer name-based virtual hosting. An Ingress controller is responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic.
An Ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer.

Implementation Screenshots:



Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: deploy-using-kubernetes

Add additional tags

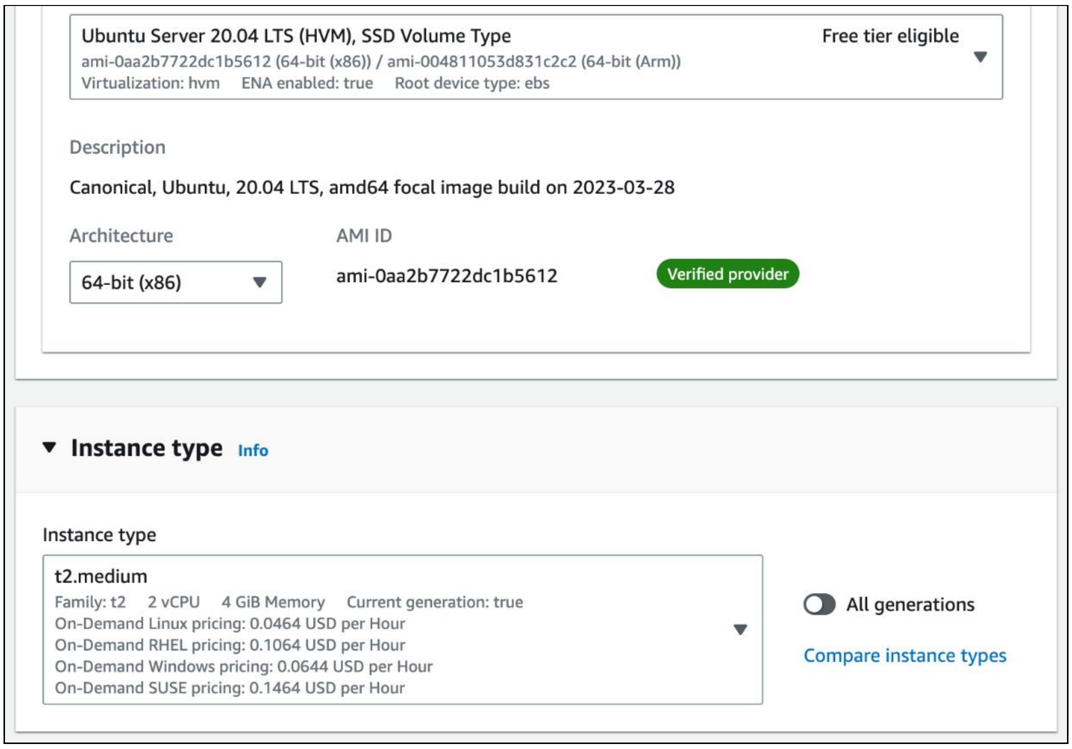
Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)

Launching EC2 Instance (With Required Specifications)


Ubuntu Server 20.04 LTS (HVM), SSD Volume Type Free tier eligible
ami-0aa2b7722dc1b5612 (64-bit (x86)) / ami-004811053d831c2c2 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Canonical, Ubuntu, 20.04 LTS, amd64 focal image build on 2023-03-28

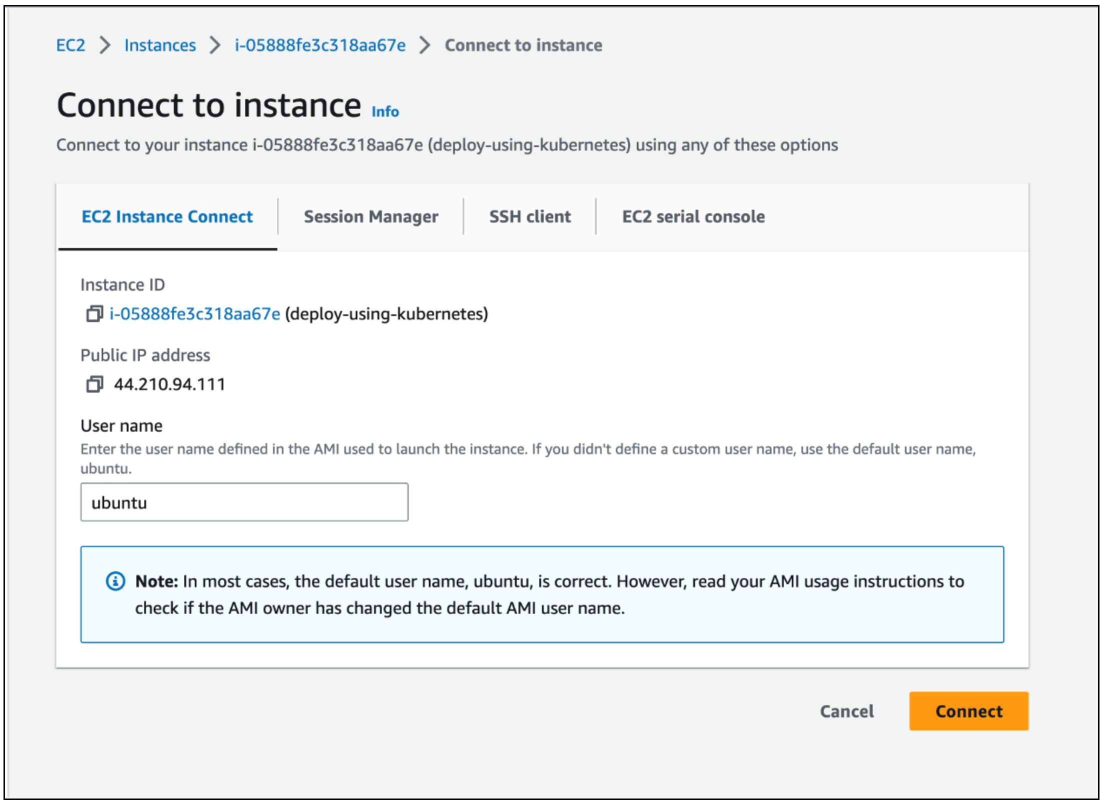
Architecture	AMI ID
64-bit (x86)	ami-0aa2b7722dc1b5612 Verified provider

▼ Instance type Info

Instance type

t2.medium
Family: t2 2 vCPU 4 GiB Memory Current generation: true
On-Demand Linux pricing: 0.0464 USD per Hour
On-Demand RHEL pricing: 0.1064 USD per Hour
On-Demand Windows pricing: 0.0644 USD per Hour
On-Demand SUSE pricing: 0.1464 USD per Hour

Choosing `t2.medium` as Instance Type


EC2 > Instances > i-05888fe3c318aa67e > Connect to instance

Connect to instance Info

Connect to your instance i-05888fe3c318aa67e (deploy-using-kubernetes) using any of these options

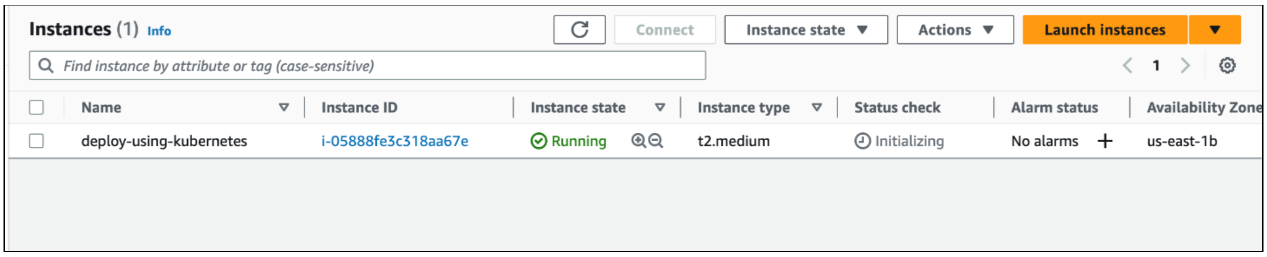
Instance ID
i-05888fe3c318aa67e (deploy-using-kubernetes)

Public IP address
44.210.94.111

User name
Enter the user name defined in the AMI used to launch the instance. If you didn't define a custom user name, use the default user name, ubuntu.
ubuntu

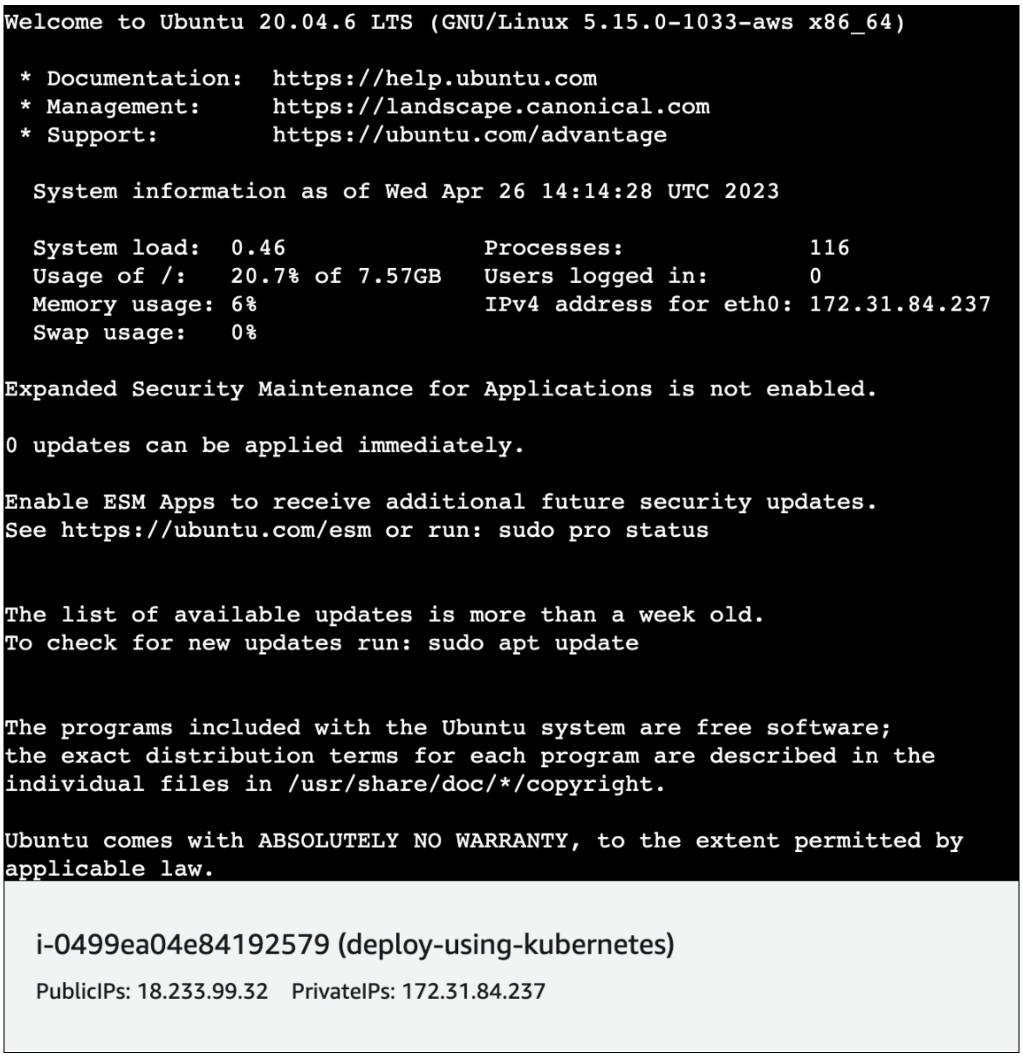
Note: In most cases, the default user name, ubuntu, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

Connecting to Instance


Instances (1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
deploy-using-kubernetes	i-05888fe3c318aa67e	Running	t2.medium	Initializing	No alarms	us-east-1b

Instance Created Successfully

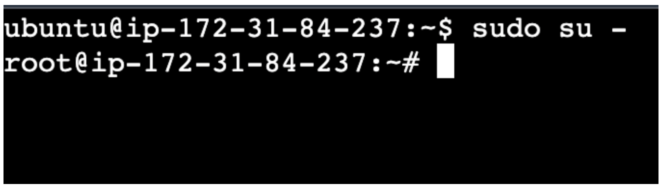


```
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1033-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed Apr 26 14:14:28 UTC 2023

  System load:  0.46              Processes:             116
  Usage of /:   20.7% of 7.57GB   Users logged in:       0
  Memory usage: 6%                IPv4 address for eth0: 172.31.84.237
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

i-0499ea04e84192579 (deploy-using-kubernetes)

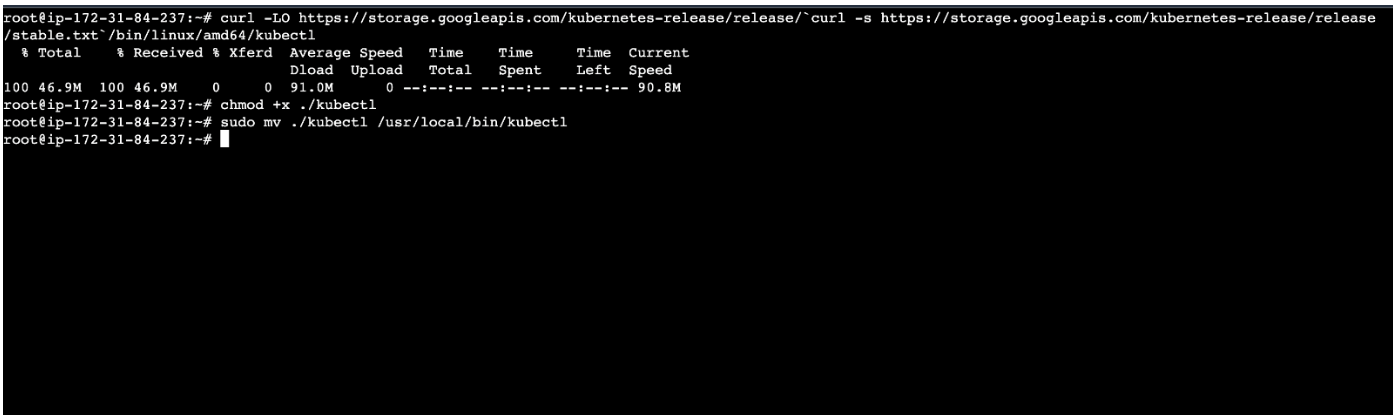
PublicIPs: 18.233.99.32   PrivateIPs: 172.31.84.237

Connecting to EC2 Instance



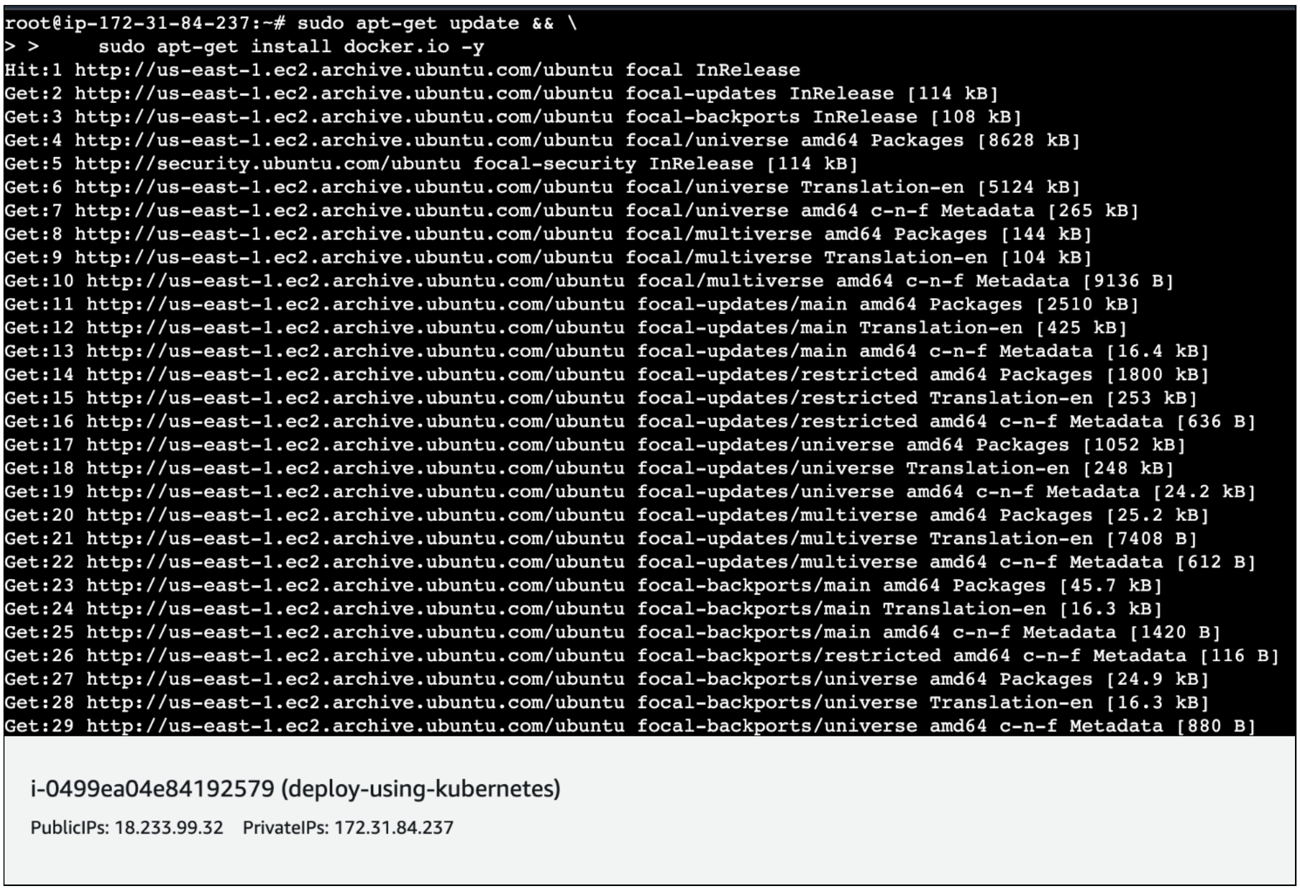
```
ubuntu@ip-172-31-84-237:~$ sudo su -
root@ip-172-31-84-237:~#
```

Accessing Root Privilege



```
root@ip-172-31-84-237:~# curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release
/stable.txt`/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 46.9M  100 46.9M    0     0  91.0M      0 --:--:-- --:--:-- --:--:-- 90.8M
root@ip-172-31-84-237:~# chmod +x ./kubectl
root@ip-172-31-84-237:~# sudo mv ./kubectl /usr/local/bin/kubectl
root@ip-172-31-84-237:~#
```

Installing `kubectl` onto EC2 Machine

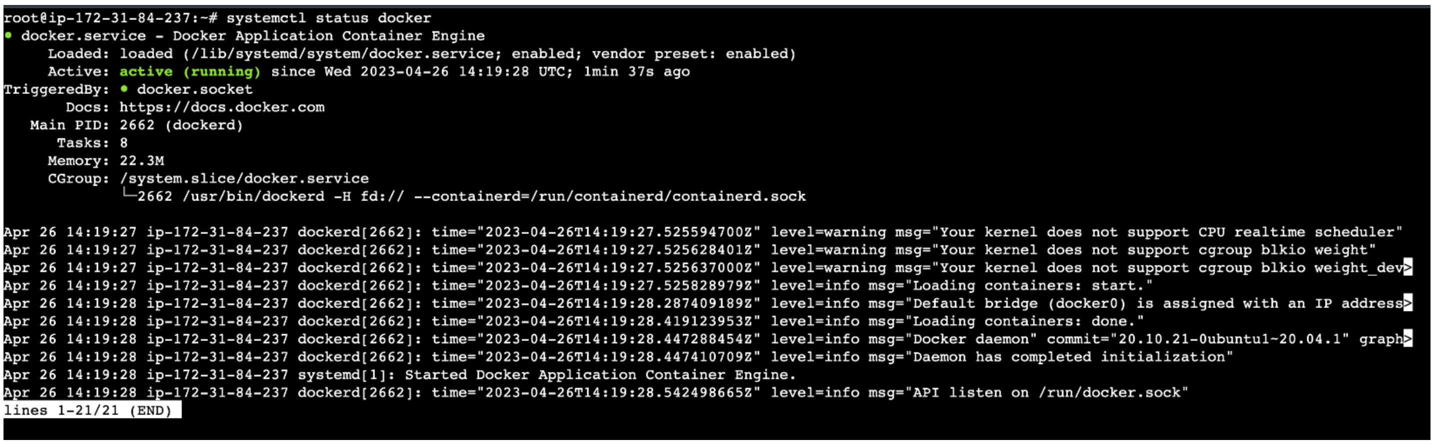


```
root@ip-172-31-84-237:~# sudo apt-get update && \
> >     sudo apt-get install docker.io -y
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 Packages [8628 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/universe Translation-en [5124 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 c-n-f Metadata [265 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [144 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/multiverse Translation-en [104 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/multiverse amd64 c-n-f Metadata [9136 B]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [2510 kB]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [425 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [16.4 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [1800 kB]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [253 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 c-n-f Metadata [636 B]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1052 kB]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/universe Translation-en [248 kB]
Get:19 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [24.2 kB]
Get:20 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [25.2 kB]
Get:21 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/multiverse Translation-en [7408 B]
Get:22 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 c-n-f Metadata [612 B]
Get:23 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports/main amd64 Packages [45.7 kB]
Get:24 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports/main Translation-en [16.3 kB]
Get:25 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports/main amd64 c-n-f Metadata [1420 B]
Get:26 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports/restricted amd64 c-n-f Metadata [116 B]
Get:27 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [24.9 kB]
Get:28 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports/universe Translation-en [16.3 kB]
Get:29 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports/universe amd64 c-n-f Metadata [880 B]
```

i-0499ea04e84192579 (deploy-using-kubernetes)

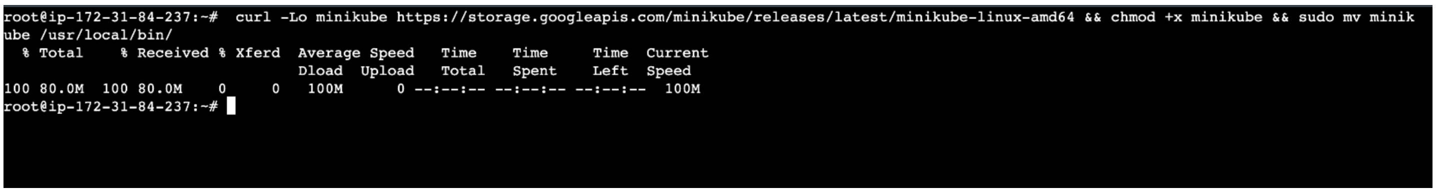
PublicIPs: 18.233.99.32   PrivateIPs: 172.31.84.237

Installing Docker onto EC2 Machine



```
root@ip-172-31-84-237:~# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-04-26 14:19:28 UTC; 1min 37s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 2662 (dockerd)
      Tasks: 8
     Memory: 22.3M
     CGroup: /system.slice/docker.service
             └─2662 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Apr 26 14:19:27 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:27.525594700Z" level=warning msg="Your kernel does not support CPU realtime scheduler"
Apr 26 14:19:27 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:27.525628401Z" level=warning msg="Your kernel does not support cgroup blkio weight"
Apr 26 14:19:27 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:27.525637000Z" level=warning msg="Your kernel does not support cgroup blkio weight_dev>
Apr 26 14:19:27 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:27.525828979Z" level=info msg="Loading containers: start."
Apr 26 14:19:28 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:28.287409189Z" level=info msg="Default bridge (docker0) is assigned with an IP address>
Apr 26 14:19:28 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:28.419123953Z" level=info msg="Loading containers: done."
Apr 26 14:19:28 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:28.447288454Z" level=info msg="Docker daemon" commit="20.10.21-0ubuntu1~20.04.1" graph>
Apr 26 14:19:28 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:28.447410709Z" level=info msg="Daemon has completed initialization"
Apr 26 14:19:28 ip-172-31-84-237 systemd[1]: Started Docker Application Container Engine.
Apr 26 14:19:28 ip-172-31-84-237 dockerd[2662]: time="2023-04-26T14:19:28.542498665Z" level=info msg="API listen on /run/docker.sock"
lines 1-21/21 (END)
```

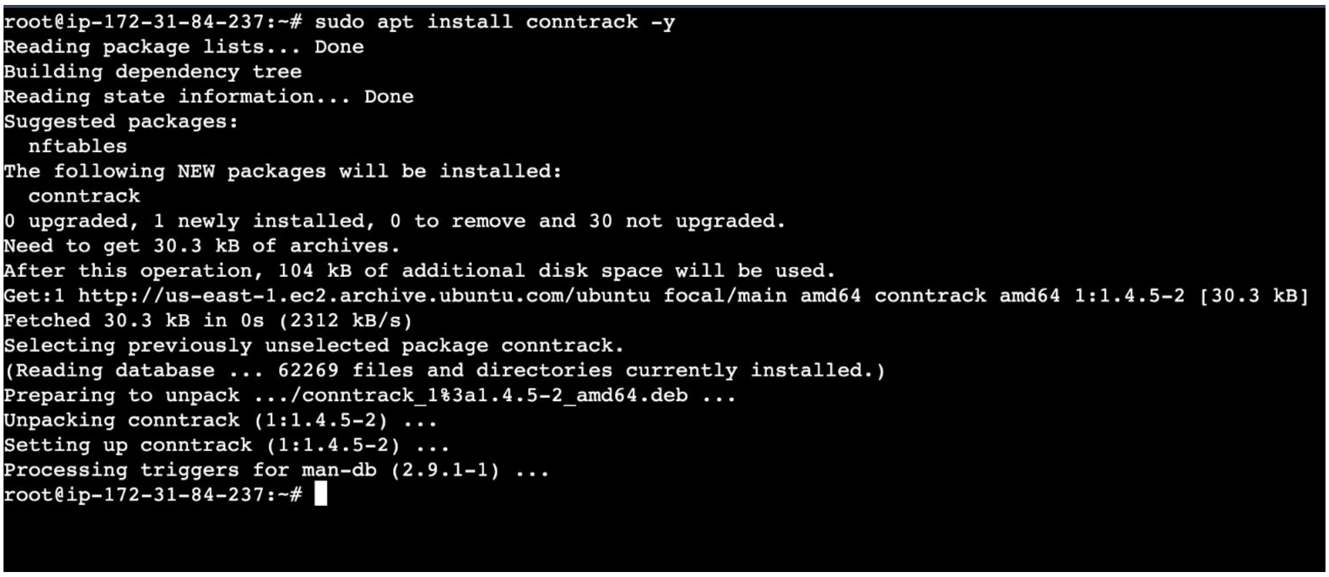
Checking System Status of Installed Docker



```
root@ip-172-31-84-237:~#  curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube && sudo mv minik
ube /usr/local/bin/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.0M  100 80.0M    0     0   100M      0 --:--:-- --:--:-- --:--:--  100M
root@ip-172-31-84-237:~# 
```

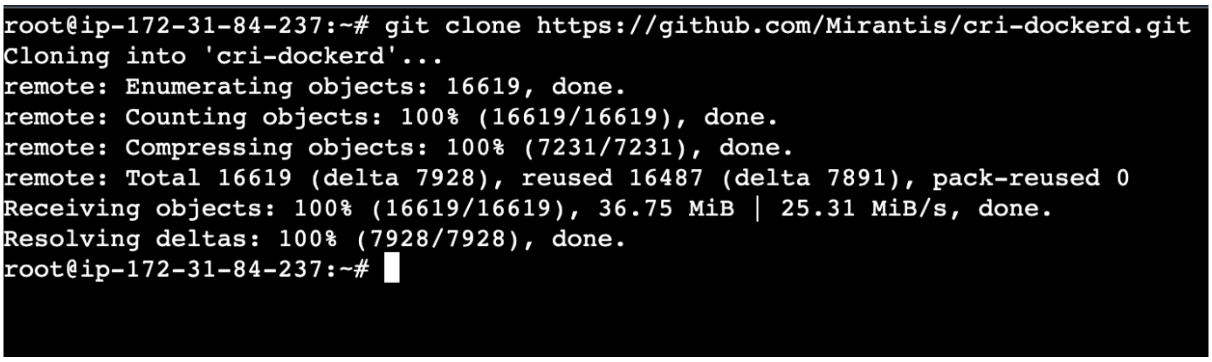
Installing MiniKube onto EC2 Machine

P.T.O

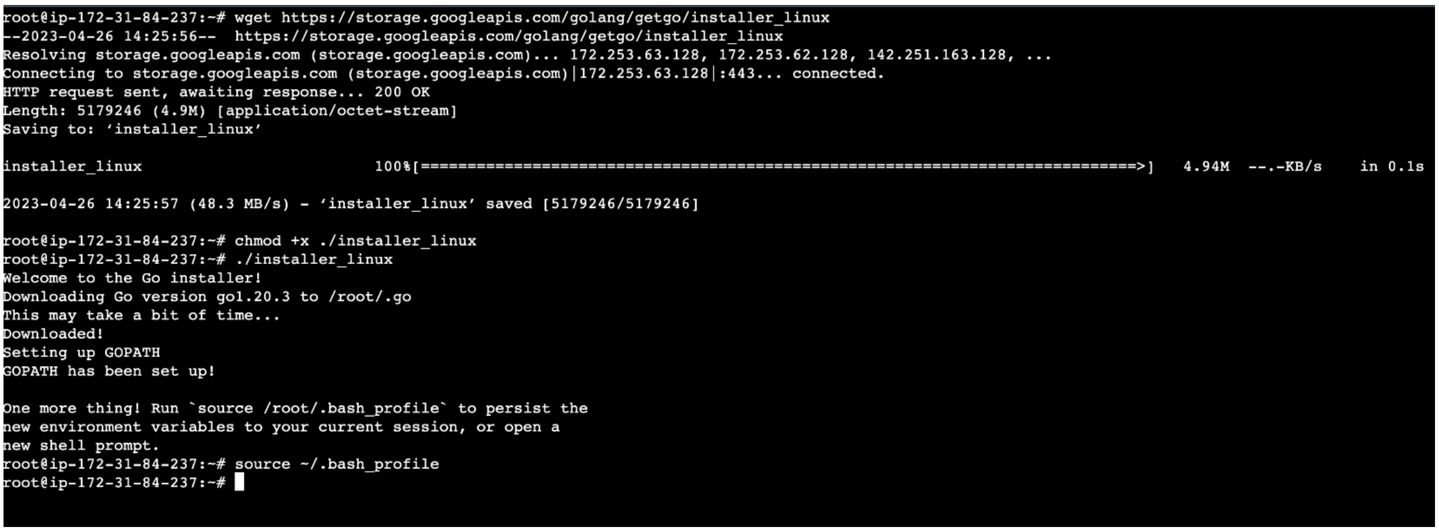


```
root@ip-172-31-84-237:~# sudo apt install conntrack -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  nftables
The following NEW packages will be installed:
  conntrack
0 upgraded, 1 newly installed, 0 to remove and 30 not upgraded.
Need to get 30.3 kB of archives.
After this operation, 104 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 conntrack amd64 1:1.4.5-2 [30.3 kB]
Fetched 30.3 kB in 0s (2312 kB/s)
Selecting previously unselected package conntrack.
(Reading database ... 62269 files and directories currently installed.)
Preparing to unpack .../conntrack_1%3a1.4.5-2_amd64.deb ...
Unpacking conntrack (1:1.4.5-2) ...
Setting up conntrack (1:1.4.5-2) ...
Processing triggers for man-db (2.9.1-1) ...
root@ip-172-31-84-237:~#
```

Installing Conntrack on EC2 Machine



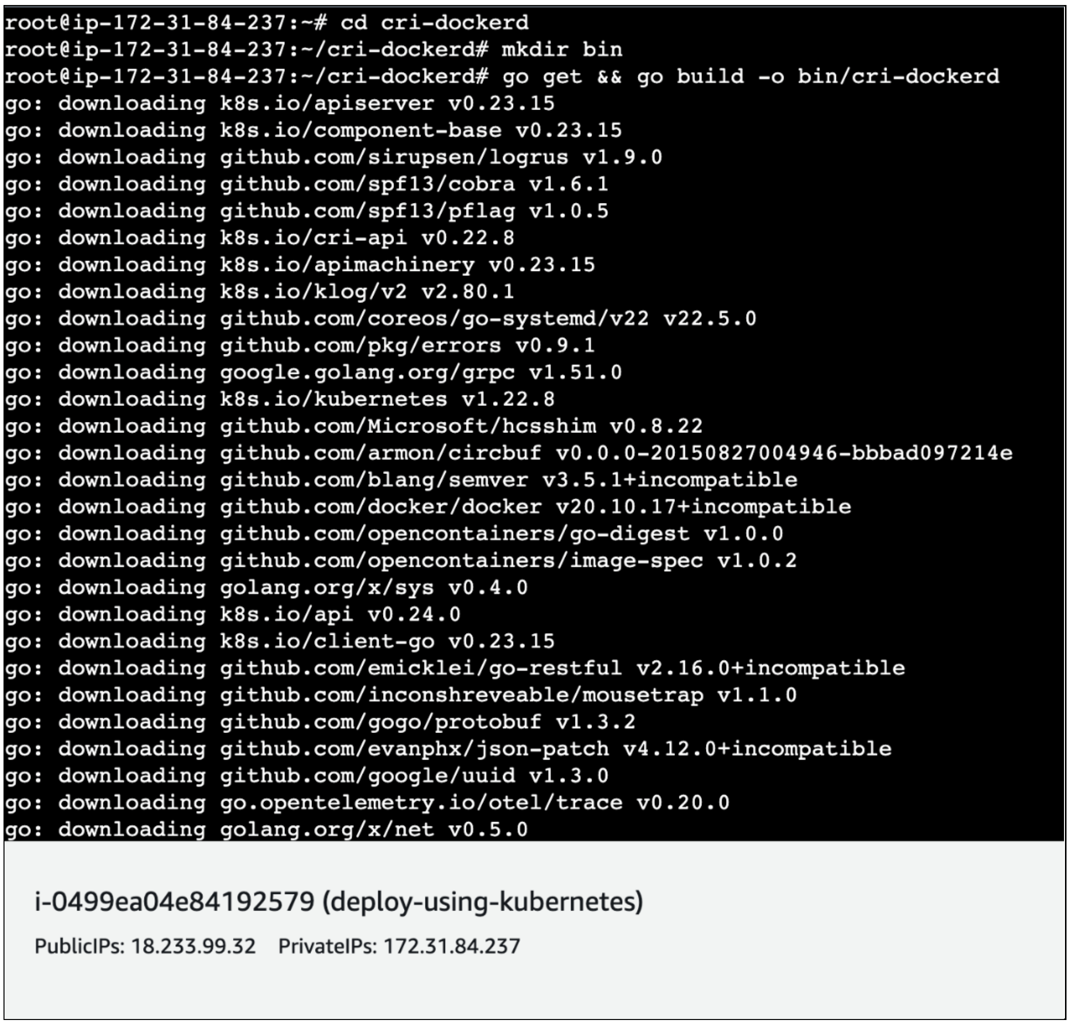
```
root@ip-172-31-84-237:~# git clone https://github.com/Mirantis/cri-dockerd.git
Cloning into 'cri-dockerd'...
remote: Enumerating objects: 16619, done.
remote: Counting objects: 100% (16619/16619), done.
remote: Compressing objects: 100% (7231/7231), done.
remote: Total 16619 (delta 7928), reused 16487 (delta 7891), pack-reused 0
Receiving objects: 100% (16619/16619), 36.75 MiB | 25.31 MiB/s, done.
Resolving deltas: 100% (7928/7928), done.
root@ip-172-31-84-237:~#
```



```
root@ip-172-31-84-237:~# wget https://storage.googleapis.com/golang/getgo/installer_linux
--2023-04-26 14:25:56--  https://storage.googleapis.com/golang/getgo/installer_linux
Resolving storage.googleapis.com (storage.googleapis.com)... 172.253.63.128, 172.253.62.128, 142.251.163.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.253.63.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5179246 (4.9M) [application/octet-stream]
Saving to: ‘installer_linux’

installer_linux                    100%[==============================================================================>]   4.94M  --.-KB/s    in 0.1s

2023-04-26 14:25:57 (48.3 MB/s) - ‘installer_linux’ saved [5179246/5179246]

root@ip-172-31-84-237:~# chmod +x ./installer_linux
root@ip-172-31-84-237:~# ./installer_linux
Welcome to the Go installer!
Downloading Go version go1.20.3 to /root/.go
This may take a bit of time...
Downloaded!
Setting up GOPATH
GOPATH has been set up!

One more thing! Run `source /root/.bash_profile` to persist the
new environment variables to your current session, or open a
new shell prompt.
root@ip-172-31-84-237:~# source ~/.bash_profile
root@ip-172-31-84-237:~#
```

Cloning `dockered.git` Repository and Installing it

P.T.O

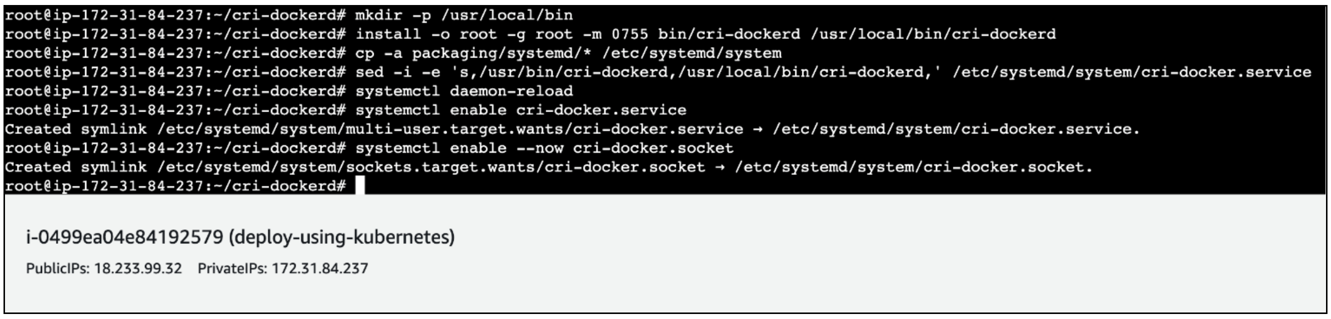


```
root@ip-172-31-84-237:~# cd cri-dockerd
root@ip-172-31-84-237:~/cri-dockerd# mkdir bin
root@ip-172-31-84-237:~/cri-dockerd# go get && go build -o bin/cri-dockerd
go: downloading k8s.io/apiserver v0.23.15
go: downloading k8s.io/component-base v0.23.15
go: downloading github.com/sirupsen/logrus v1.9.0
go: downloading github.com/spf13/cobra v1.6.1
go: downloading github.com/spf13/pflag v1.0.5
go: downloading k8s.io/cri-api v0.22.8
go: downloading k8s.io/apimachinery v0.23.15
go: downloading k8s.io/klog/v2 v2.80.1
go: downloading github.com/coreos/go-systemd/v22 v22.5.0
go: downloading github.com/pkg/errors v0.9.1
go: downloading google.golang.org/grpc v1.51.0
go: downloading k8s.io/kubernetes v1.22.8
go: downloading github.com/Microsoft/hcsshim v0.8.22
go: downloading github.com/armon/circbuf v0.0.0-20150827004946-bbbad097214e
go: downloading github.com/blang/semver v3.5.1+incompatible
go: downloading github.com/docker/docker v20.10.17+incompatible
go: downloading github.com/opencontainers/go-digest v1.0.0
go: downloading github.com/opencontainers/image-spec v1.0.2
go: downloading golang.org/x/sys v0.4.0
go: downloading k8s.io/api v0.24.0
go: downloading k8s.io/client-go v0.23.15
go: downloading github.com/emicklei/go-restful v2.16.0+incompatible
go: downloading github.com/inconshreveable/mousetrap v1.1.0
go: downloading github.com/gogo/protobuf v1.3.2
go: downloading github.com/evanphx/json-patch v4.12.0+incompatible
go: downloading github.com/google/uuid v1.3.0
go: downloading go.opentelemetry.io/otel/trace v0.20.0
go: downloading golang.org/x/net v0.5.0
```

i-0499ea04e84192579 (deploy-using-kubernetes)

PublicIPs: 18.233.99.32   PrivateIPs: 172.31.84.237

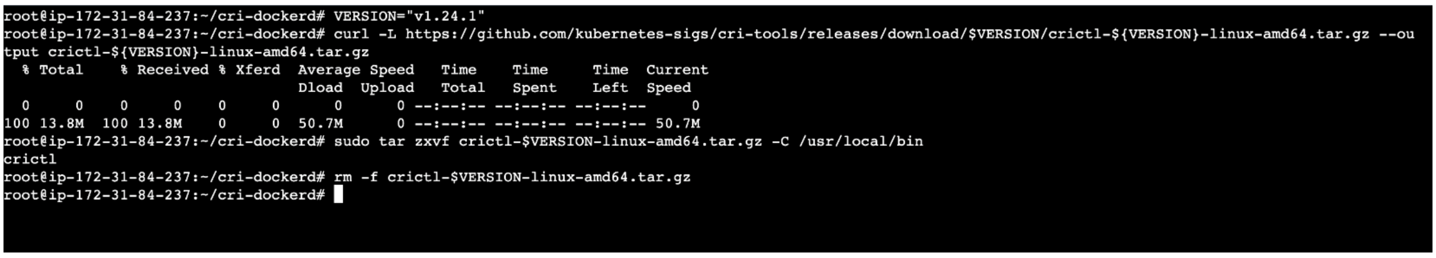
Building GO Environment to Run the Application



```
root@ip-172-31-84-237:~/cri-dockerd# mkdir -p /usr/local/bin
root@ip-172-31-84-237:~/cri-dockerd# install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-dockerd
root@ip-172-31-84-237:~/cri-dockerd# cp -a packaging/systemd/* /etc/systemd/system
root@ip-172-31-84-237:~/cri-dockerd# sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service
root@ip-172-31-84-237:~/cri-dockerd# systemctl daemon-reload
root@ip-172-31-84-237:~/cri-dockerd# systemctl enable cri-docker.service
Created symlink /etc/systemd/system/multi-user.target.wants/cri-docker.service → /etc/systemd/system/cri-docker.service.
root@ip-172-31-84-237:~/cri-dockerd# systemctl enable --now cri-docker.socket
Created symlink /etc/systemd/system/sockets.target.wants/cri-docker.socket → /etc/systemd/system/cri-docker.socket.
root@ip-172-31-84-237:~/cri-dockerd#
```

i-0499ea04e84192579 (deploy-using-kubernetes)

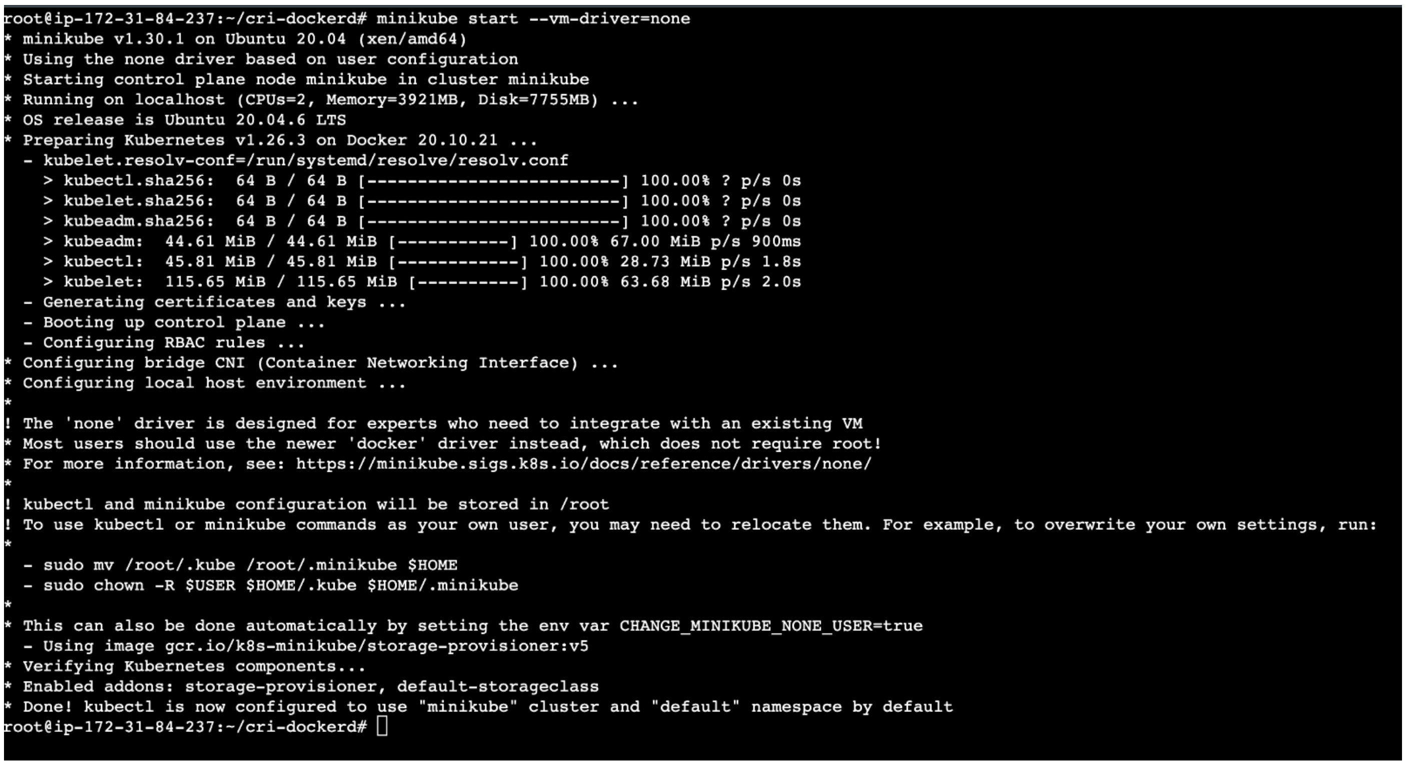
PublicIPs: 18.233.99.32   PrivateIPs: 172.31.84.237



```
root@ip-172-31-84-237:~/cri-dockerd# VERSION="v1.24.1"
root@ip-172-31-84-237:~/cri-dockerd# curl -L https://github.com/kubernetes-sigs/cri-tools/releases/download/$VERSION/crictl-${VERSION}-linux-amd64.tar.gz --ou
tput crictl-${VERSION}-linux-amd64.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 13.8M  100 13.8M    0     0  50.7M      0 --:--:-- --:--:-- --:--:-- 50.7M
root@ip-172-31-84-237:~/cri-dockerd# sudo tar zxvf crictl-$VERSION-linux-amd64.tar.gz -C /usr/local/bin
crictl
root@ip-172-31-84-237:~/cri-dockerd# rm -f crictl-$VERSION-linux-amd64.tar.gz
root@ip-172-31-84-237:~/cri-dockerd#
```

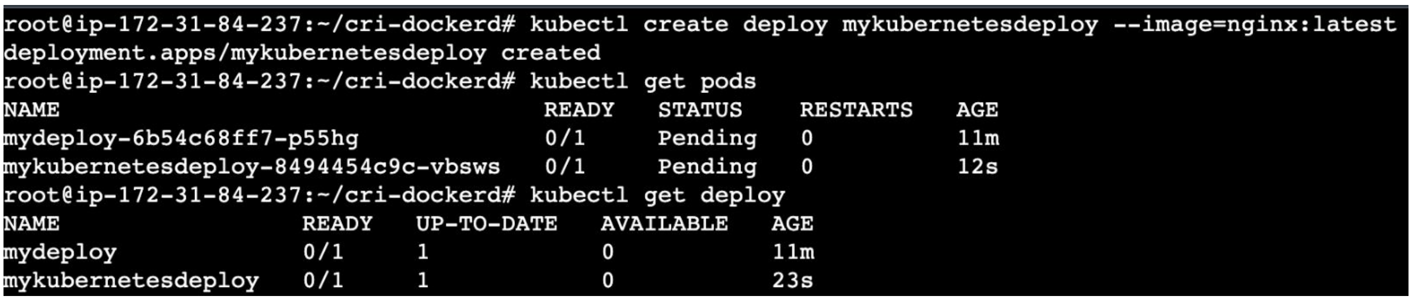
Installing `crictl` Module for Minikube

P.T.O



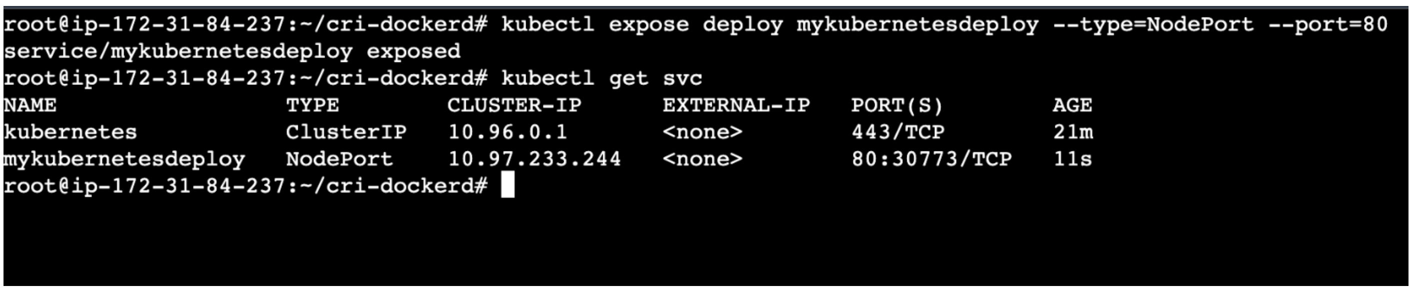
```
root@ip-172-31-84-237:~/cri-dockerd# minikube start --vm-driver=none
* minikube v1.30.1 on Ubuntu 20.04 (xen/amd64)
* Using the none driver based on user configuration
* Starting control plane node minikube in cluster minikube
* Running on localhost (CPUs=2, Memory=3921MB, Disk=7755MB) ...
* OS release is Ubuntu 20.04.6 LTS
* Preparing Kubernetes v1.26.3 on Docker 20.10.21 ...
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
    > kubectl.sha256:  64 B / 64 B [-------------------------] 100.00% ? p/s 0s
    > kubelet.sha256:  64 B / 64 B [-------------------------] 100.00% ? p/s 0s
    > kubeadm.sha256:  64 B / 64 B [-------------------------] 100.00% ? p/s 0s
    > kubeadm:  44.61 MiB / 44.61 MiB [-----------] 100.00% 67.00 MiB p/s 900ms
    > kubectl:  45.81 MiB / 45.81 MiB [------------] 100.00% 28.73 MiB p/s 1.8s
    > kubelet:  115.65 MiB / 115.65 MiB [----------] 100.00% 63.68 MiB p/s 2.0s
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Configuring local host environment ...
*
! The 'none' driver is designed for experts who need to integrate with an existing VM
* Most users should use the newer 'docker' driver instead, which does not require root!
* For more information, see: https://minikube.sigs.k8s.io/docs/reference/drivers/none/
*
! kubectl and minikube configuration will be stored in /root
! To use kubectl or minikube commands as your own user, you may need to relocate them. For example, to overwrite your own settings, run:
*
  - sudo mv /root/.kube /root/.minikube $HOME
  - sudo chown -R $USER $HOME/.kube $HOME/.minikube
*
* This can also be done automatically by setting the env var CHANGE_MINIKUBE_NONE_USER=true
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Verifying Kubernetes components...
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
root@ip-172-31-84-237:~/cri-dockerd# 
```

Starting Minikube Service



```
root@ip-172-31-84-237:~/cri-dockerd# kubectl create deploy mykubernetesdeploy --image=nginx:latest
deployment.apps/mykubernetesdeploy created
root@ip-172-31-84-237:~/cri-dockerd# kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
mydeploy-6b54c68ff7-p55hg             0/1     Pending   0          11m
mykubernetesdeploy-8494454c9c-vbsws   0/1     Pending   0          12s
root@ip-172-31-84-237:~/cri-dockerd# kubectl get deploy
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
mydeploy             0/1     1            0           11m
mykubernetesdeploy   0/1     1            0           23s
```

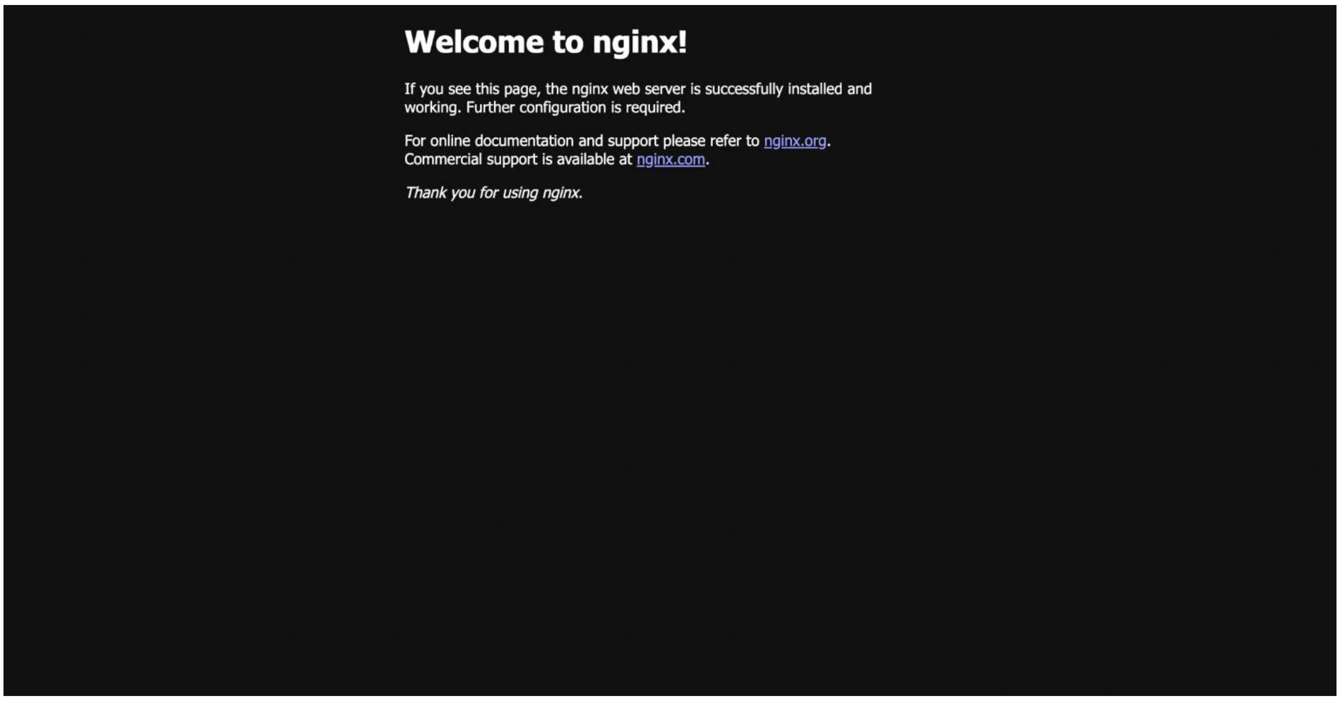
Deploying the Nginx Image to the Container and Checking Pods



```
root@ip-172-31-84-237:~/cri-dockerd# kubectl expose deploy mykubernetesdeploy --type=NodePort --port=80
service/mykubernetesdeploy exposed
root@ip-172-31-84-237:~/cri-dockerd# kubectl get svc
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP        21m
mykubernetesdeploy   NodePort    10.97.233.244   <none>        80:30773/TCP   11s
root@ip-172-31-84-237:~/cri-dockerd# 
```

Deploying

P.T.O



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Successfully Deployed on Desired Port

Conclusion:

Therefore, we have successfully deployed our Web Application on Kubernetes.