Bansilal Ramnath Agarwal Charitable Trust's

VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY,

PUNE-48 Department of Information Technology

ITUA32202: CLOUD COMPUTING
Assignment-8

Shreyas Shripad Kulkarni
C2 Batch
Roll No. 333030
PRN: 22010443

AIM: Setup Single Node Kubernetes Cluster with Minikube and Deploy a web app on Kubernetes cluster.

THEORY:

1) 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

2) 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 behaviour was handled by a system?

That is 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

3) Kubernetes Components

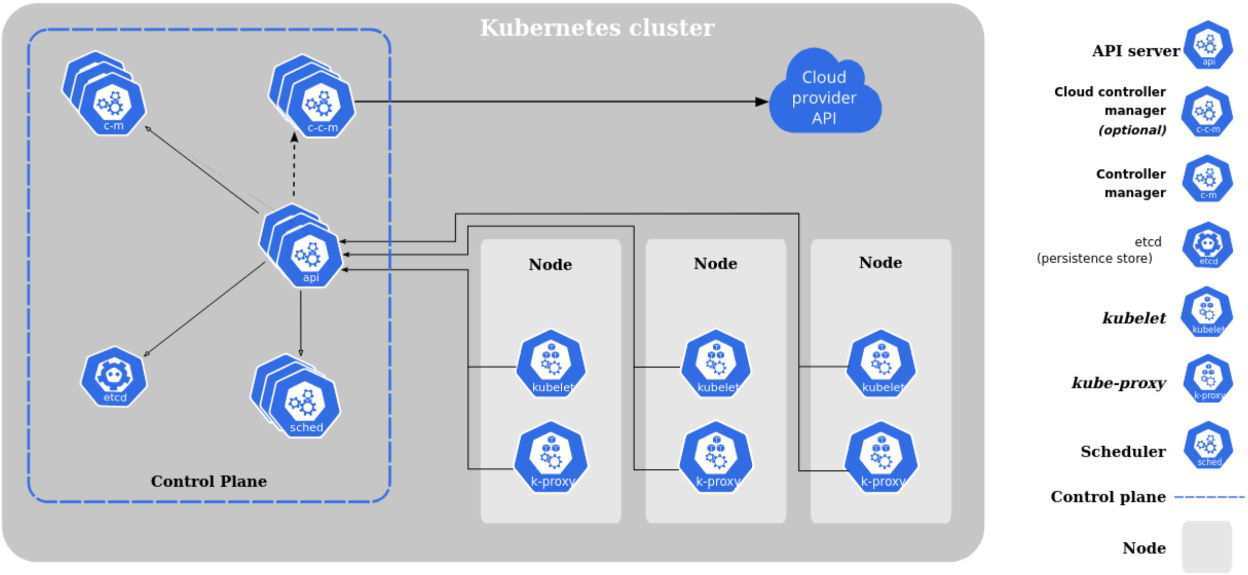
When you deploy Kubernetes, you get a cluster.

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

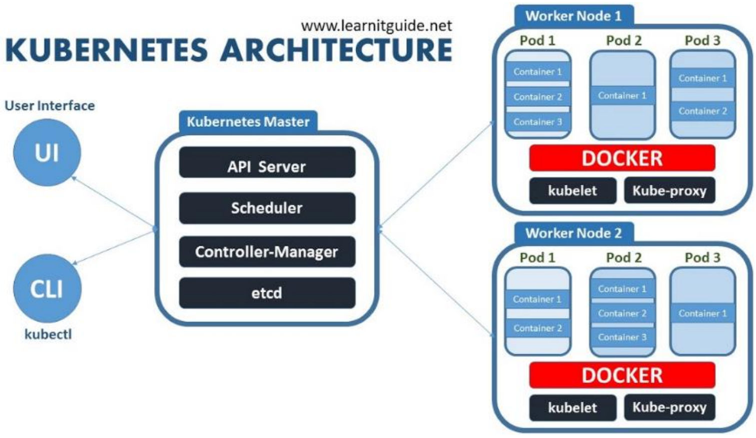
The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance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

This document outlines the various components you need to have a complete and working Kubernetes cluster.

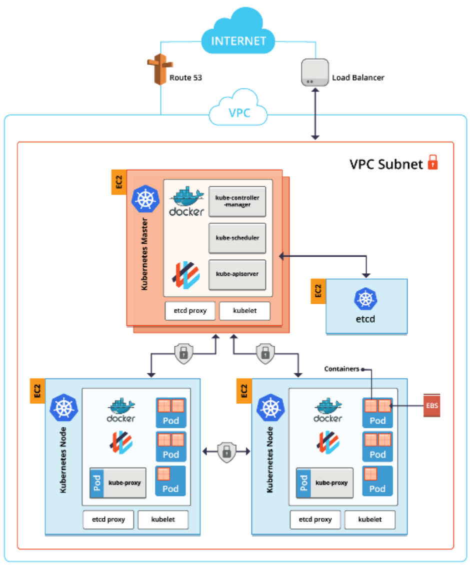
Diagram of a Kubernetes cluster with all the components tied together.



4) Kubernetes Architecture



5) Kubernetes Architecture on AWS



6) Control Pane Components

The control plane'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 back up 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 taken into account 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 controller: 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

7) 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, container, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

8) 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.

9) Using Pods

Usually, we 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:

- <u>Pods that run a single container.</u> 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.
- <u>Pods that run multiple containers that need to work together</u>. 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.

10) 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.

11) Ingress

FEATURE STATE: Kubernetes v1.19 [stable] An API object that manages external access to the services in a cluster, typically HTTP. Ingress may provide load balancing, SSL termination and name-based virtual hosting.

12) Terminology

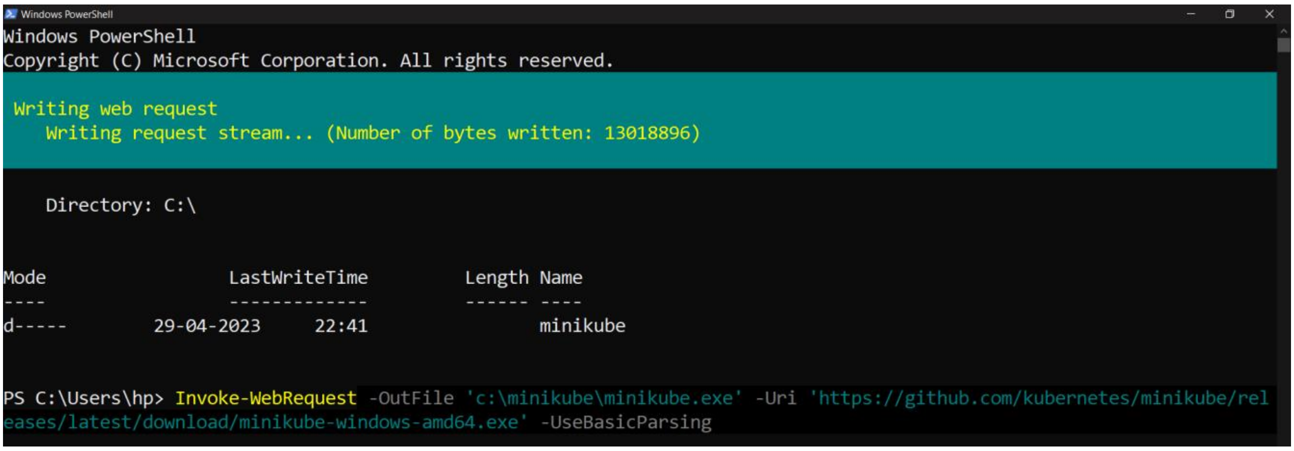
For clarity, this guide defines the following terms:

- 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.

STEPWISE PROCESS to setup Single Node Kubernetes Cluster with Minikube

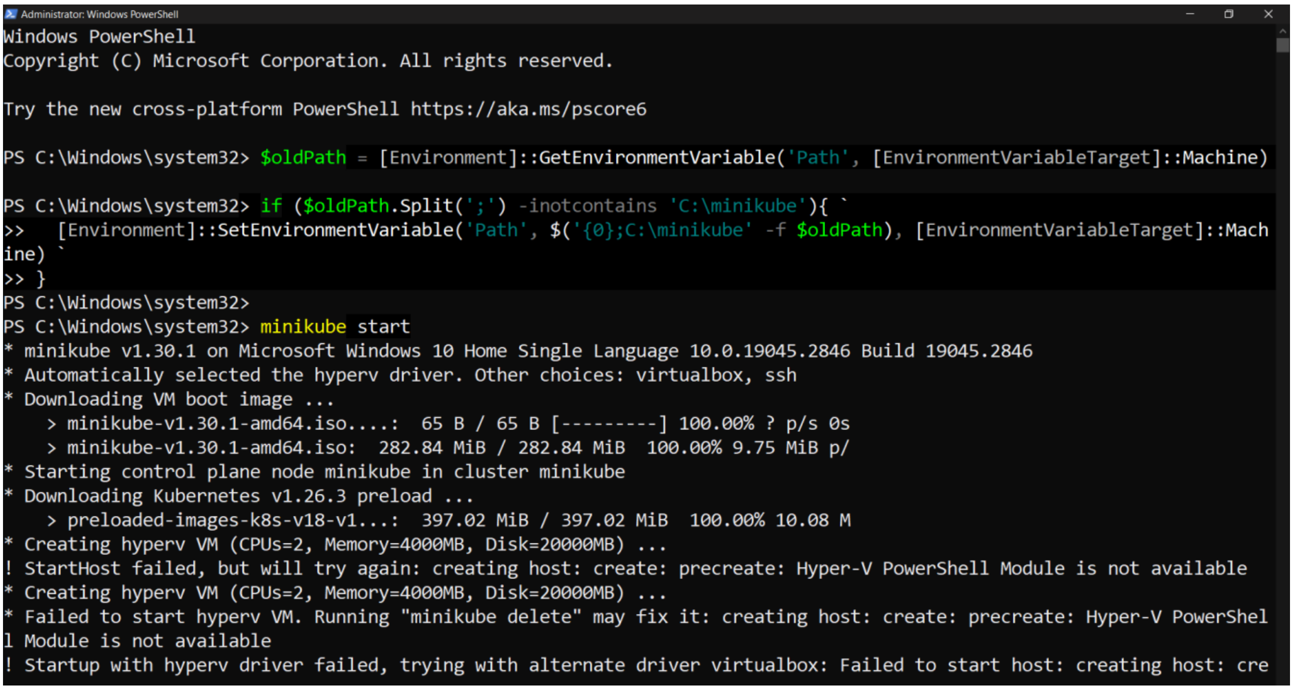
1) Installing Minikube -> https://kubernetes.io/docs/tasks/tools/install-minikube/

Install Virtualbox latest edition



2) Setting up Minikube on virtualbox -> https://kubernetes.io/docs/tasks/tools/install-minikube/

You will need to keep the minikube in the PATH both on Windows/Linux



Use the following instruction to setup single node Kubernetes cluster

minikube start --driver=virtualbox

You can also set custom configuration like

minikube start --driver=virtualbox --cpus=2 --memory=4096m

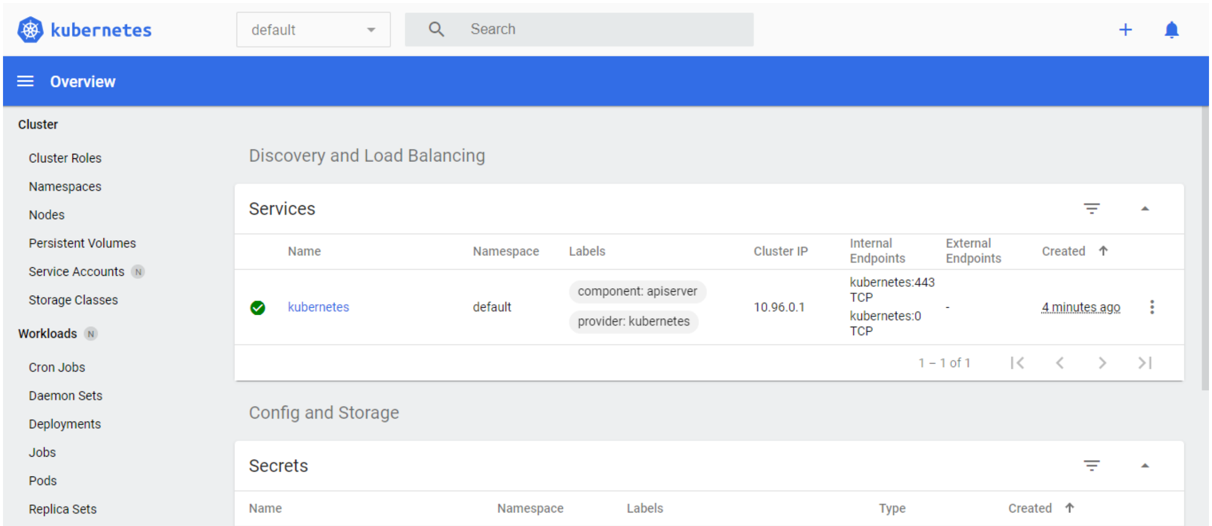
```
PS C:\Windows\system32> minikube start --driver=virtualbox
* minikube v1.30.1 on Microsoft Windows 10 Home Single Language 10.0.19045.2846 Build 19045.2846
* Using the virtualbox driver based on existing profile
* Starting control plane node minikube in cluster minikube
* Creating virtualbox VM (CPUs=2, Memory=4000MB, Disk=20000MB) ...
! StartHost failed, but will try again: creating host: create: precreate: This computer doesn't have VT-X/AMD-v enable
d. Enabling it in the BIOS is mandatory
* Creating virtualbox VM (CPUs=2, Memory=4000MB, Disk=20000MB) ...
* Failed to start virtualbox VM. Running "minikube delete" may fix it: creating host: create: precreate: This computer
 doesn't have VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory

X Exiting due to HOST_VIRT_UNAVAILABLE: Failed to start host: creating host: create: precreate: This computer doesn't
have VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory
* Suggestion: Virtualization support is disabled on your computer. If you are running minikube within a VM, try '--dri
ver=docker'. Otherwise, consult your systems BIOS manual for how to enable virtualization.
* Related issues:
  - https://github.com/kubernetes/minikube/issues/3900
  - https://github.com/kubernetes/minikube/issues/4730
```

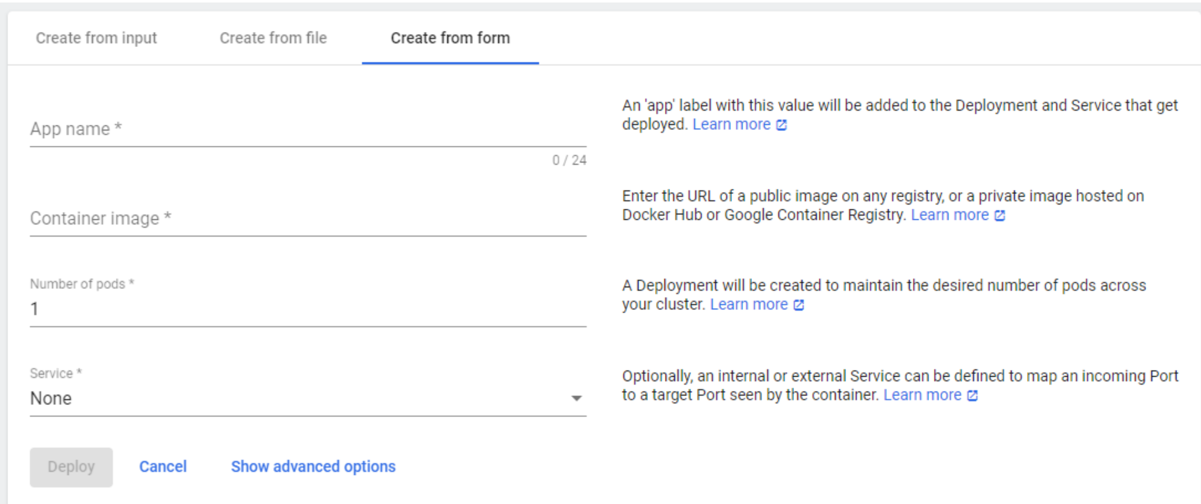
3) Open the minikube dashboard with following command. It will take 2-10 mins depending on your bandwidth.

minikube dashboard

The dashboard will open in your default browser



3) On the right side corner you will see a symbol for +, click on it, and go to -> Create from form



4) Enter the details as below

App Name: nginx

Container image: nginx

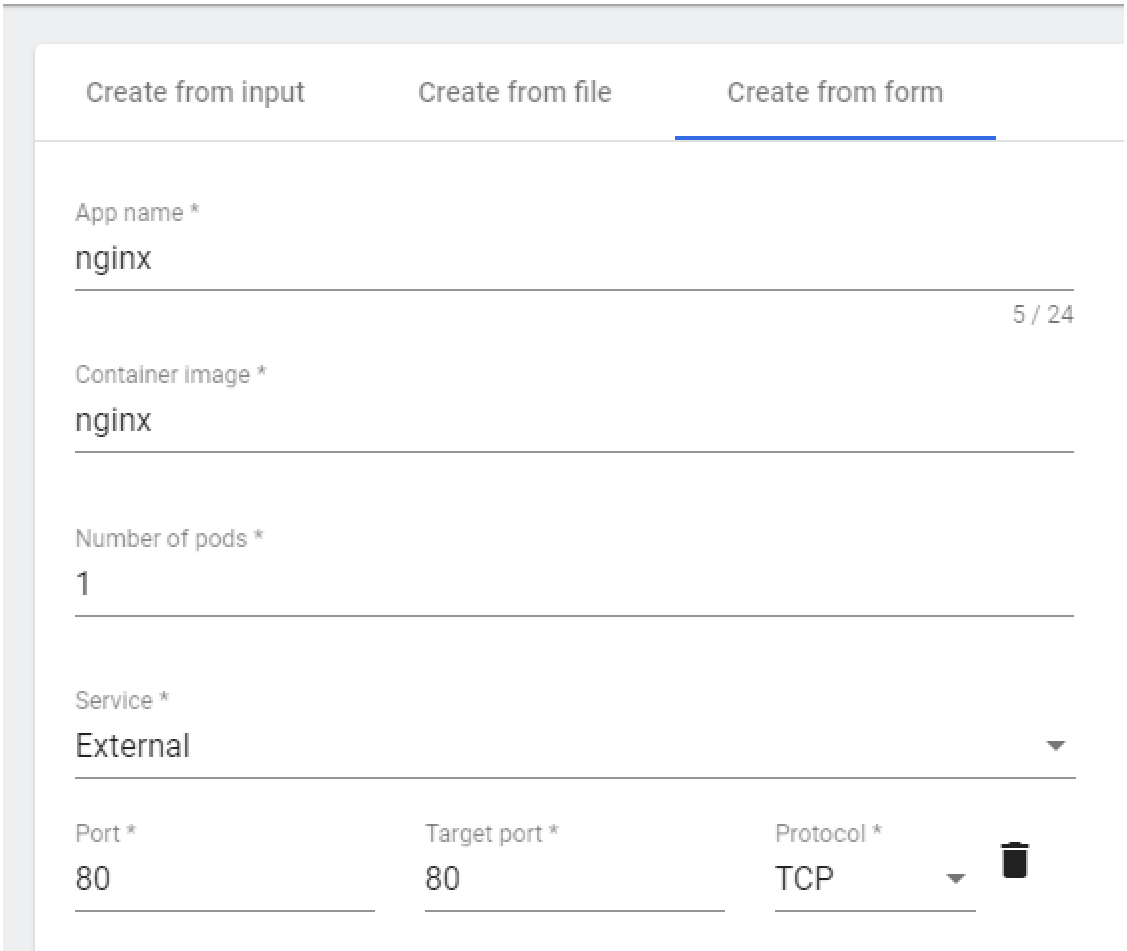
Number of pods: 1

Service: External

Port: 80

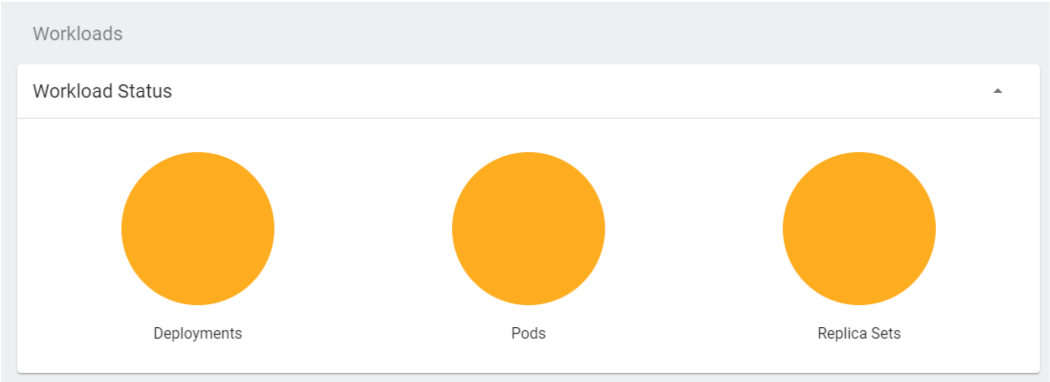
Target Port: 80

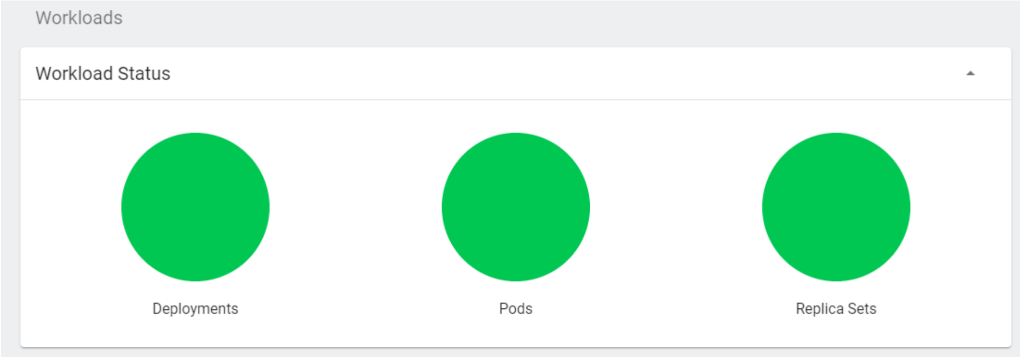
Protocol: TCP



Click Deploy

7) This will show the Work load Status. Wait till it turns green

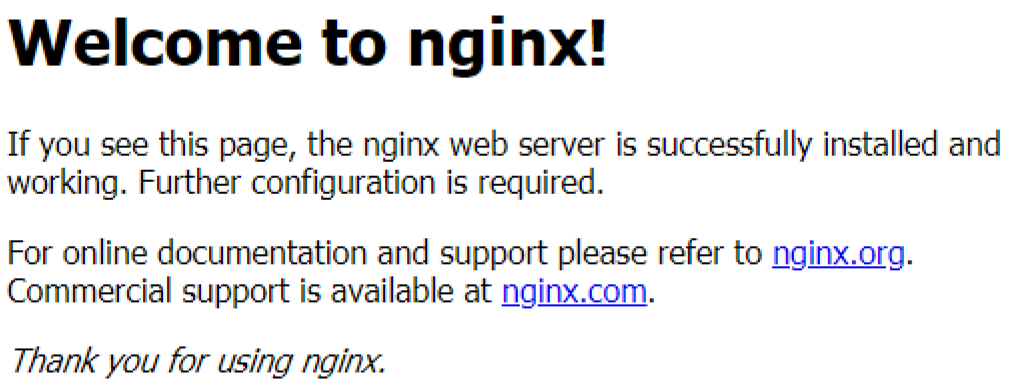




8)Access the nginx application in your browser with following command.

minikube service nginx

The nginx default page will open in browser and you will see the service details as well.



<u>Conclusion:</u> We have successfully Setup Single Node Kubernetes Cluster with Minikube and Deploy a web app on Kubernetes cluster.