

**Name: Rohit Jain**  
**Roll No.:333048**  
**Gr No.:22010315**  
**Division: TY-IT-C**

## **Assignment No 8**

**Title:** Setup Single Node Kubernetes Cluster with Minikube and Deploy an web app on kubernetes cluster .

### **Theory:**

#### ➤ **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

#### ➤ **Why 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**

#### ➤ **Kubernetes Components**

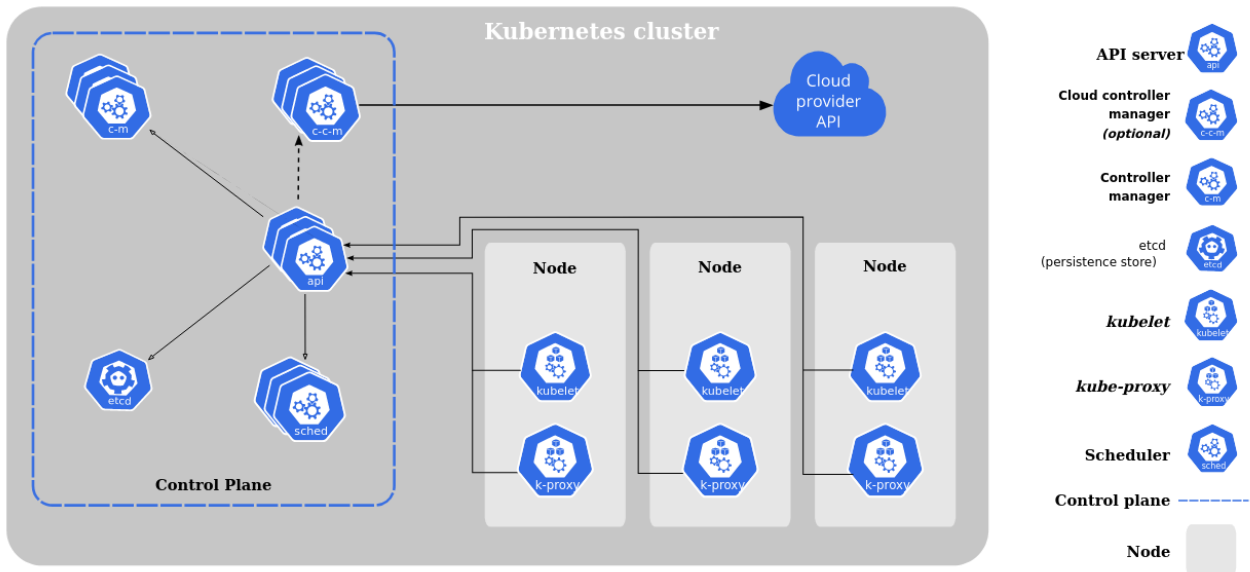
When you deploy Kubernetes, you get a cluster.

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

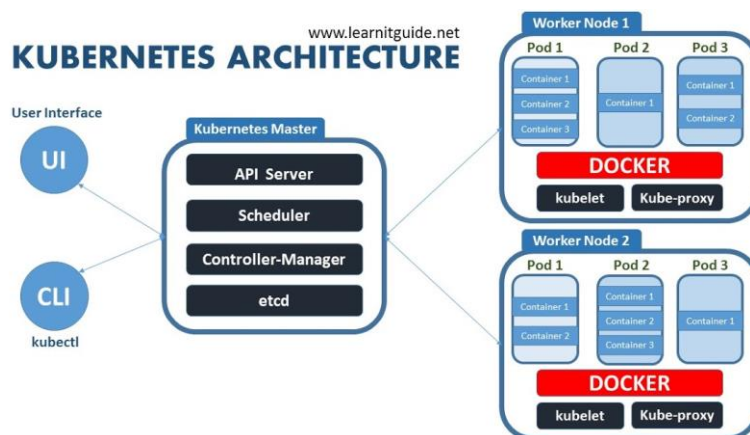
The worker node(s) host the [Pods](#) that are the components of the application workload. The [control plane](#) manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing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

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

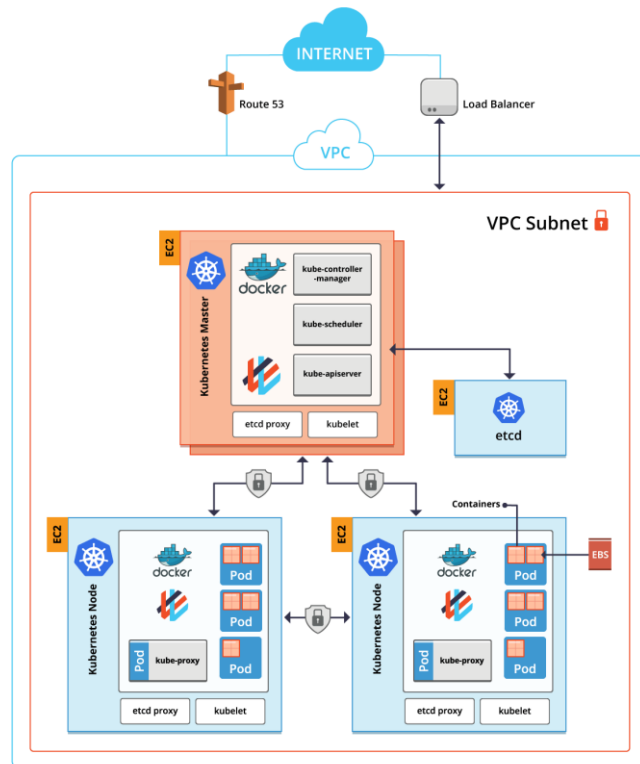
Here's the diagram of a Kubernetes cluster with all the components tied together.



## ➤ Kubernetes Architecture



## Kubernetes Architecture on AWS:



### ➤ Control Plane 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

#### ➤ 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

**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.

## ➤ 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.

## ➤ What is 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.

Here is a simple example where an Ingress sends all its traffic to one Service:



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](#).

## Try Kubernetes:

There are many ways to install Kubernetes.

**1) Manual** means that you manually set up everything from the networking bits over downloading, configuring and launching components such as etcd or kube-apiserver. You've got full control over what goes where, but it might take you some time and it's error-prone. The ultimate reference in this area is Kelsey Hightower's [Kubernetes The Hard Way](#) (KTHW).

**2) Installer** are CLI tools that leverage templates and/or automation tools such as Terraform and Ansible. You typically have a lot control over what is going on, but less than in the manual approach. Examples include:

- OpenCredo's Kubernetes from scratch to AWS with Terraform and Ansible
- [kubeadm](#) (bare metal installation, a building block for other installers)
- [kops](#), mainly AWS (AWS also provide service for kubernetes: EKS – Elastic Kubernetes Service)
- [kubicorn](#)
- [kubespray](#) (Ansible-based, both bare-metal and cloud)
- OpenShift [advanced install](#) based on Ansible

**3) Hosted** effectively means little to no installation effort on your end. For example:

- [Azure Container Service](#) (ACS) with Kubernetes
- [Google Container Engine](#) (GKE)
- [IBM Bluemix Container Service](#)
- [OpenShift Online](#) (OSO)

4) [Minikube](#) or [Minishift](#) – Virtualized Environment for Kubernetes

## ➤ **Setup Single Node Kubernetes Cluster with Minikube**

### **Implementation:**

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

Install Virtualbox latest edition.

#### 1 Installation

Click on the buttons that describe your target platform. For other architectures, see [the release page](#) for a complete list of minikube binaries.

Operating system	<a href="#">Linux</a>	<a href="#">macOS</a>	<a href="#">Windows</a>
Architecture	<a href="#">x86-64</a>		
Release type	<a href="#">Stable</a> <a href="#">Beta</a>		
Installer type	<a href="#">.exe download</a> <a href="#">Windows Package Manager</a> <a href="#">Chocolatey</a>		

To install the latest minikube **stable** release on **x86-64 Windows** using **.exe download**:

1. Download and run the installer for the **latest release**.

Or if using PowerShell, use this command:

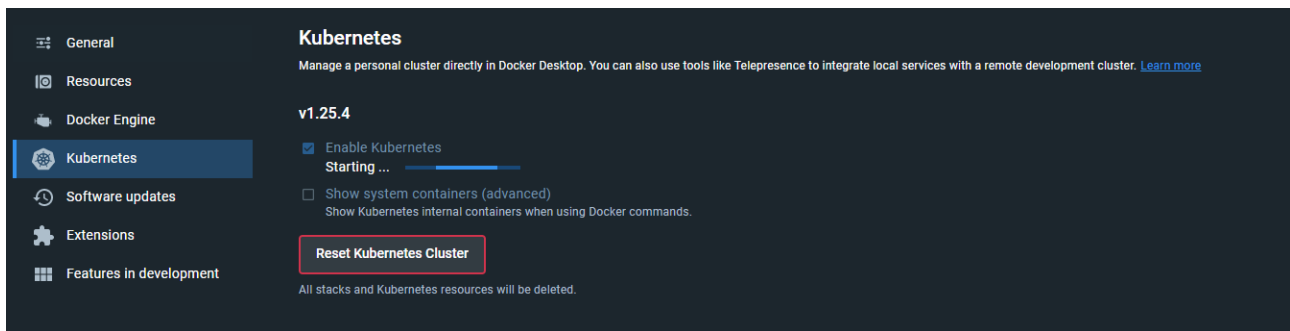
```
New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force
Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri 'https://github.com/kubernetes/minikube/releases/latest/download/minikube-windows-amd64.exe'
```

2. Add the `minikube.exe` binary to your `PATH`.

*Make sure to run PowerShell as Administrator.*

```
$oldPath = [Environment]::GetEnvironmentVariable('Path', [EnvironmentVariableTarget]::Machine)
if ($oldPath.Split(';') -notcontains 'C:\minikube'){
    [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube' -f $oldPath), [EnvironmentVariableTarget]::Machine)
}
```

If you used a terminal (like powershell) for the installation, please close the terminal and reopen it before running minikube.



```
PS C:\Users\user> New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force

Directory: C:\

Mode                LastWriteTime         Length Name
----                -
d-----          01-05-2023   02:03 PM             minikube
```

```
PS C:\Users\user> if ($oldPath.Split(';') -notcontains 'C:\minikube'){ `
>> [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube' -f $oldPath), [EnvironmentVariableTarget]::Machine)
>> |
```

- 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

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

minikube dashboard

```
PS C:\Users\user> minikube start
* minikube v1.30.1 on Microsoft Windows 11 Home Single Language 10.0.22621.1555 Build 22621.1555
* Automatically selected the docker driver
* Using Docker Desktop driver with root privileges
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.26.3 preload ...
  > preloaded-images-k8s-v18-v1...: 397.02 MiB / 397.02 MiB 100.00% 592.00
  > gcr.io/k8s-minikube/kicbase...: 373.53 MiB / 373.53 MiB 100.00% 533.43
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.26.3 on Docker 23.0.2 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
  - 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
PS C:\Users\user> |
```



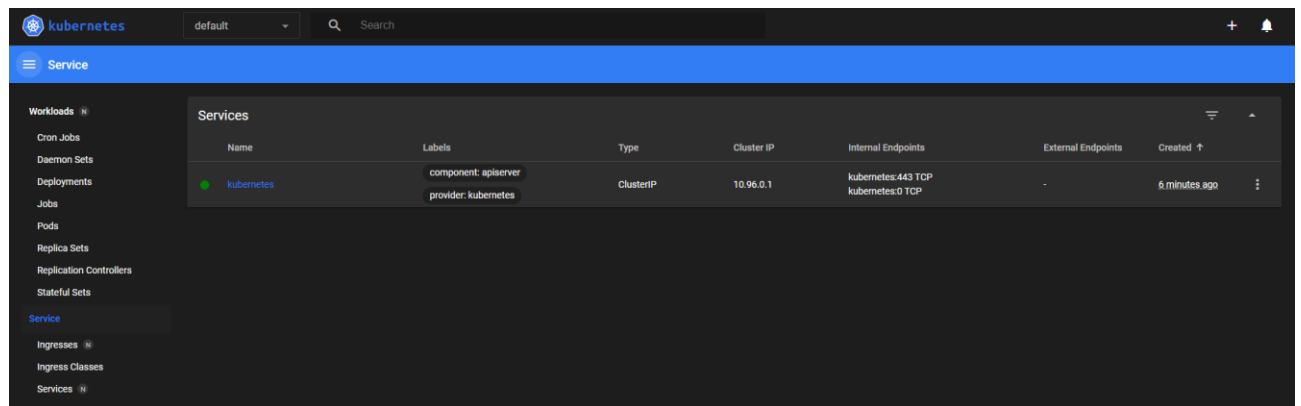
```
PS C:\Users\user> minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```


- 4) The dashboard will open in your default browser

```
PS C:\Users\user> minikube dashboard
* Enabling dashboard ...
- Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
- Using image docker.io/kubernetesui/dashboard:v2.7.0
* Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server

* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
* Opening http://127.0.0.1:55918/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```



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

Create from input

Create from file

Create from form

App name \*

0 / 24

Container image \*

Number of pods \*

1

Service \*

None

Namespace \*

default

Deploy

Preview

Cancel

Show advanced options

- 6) Enter the details as below  
App Name: nginx

Container image: nginx  
Number of pods: 1  
Service: External  
Port: 80  
Target Port: 80  
Protocol: TCP

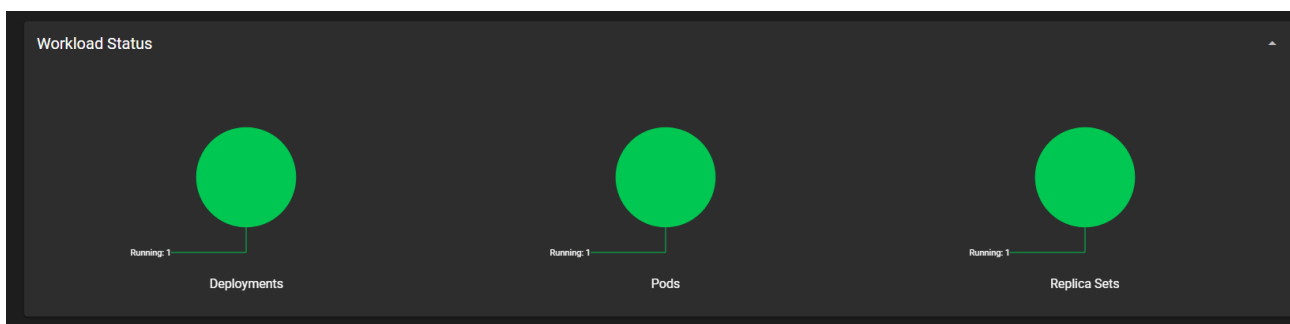
The screenshot shows the 'Create from form' tab in the Kubernetes Dashboard. The form fields are as follows:

- App name \***: nginx (A warning message below reads: 'Deployment or service with this name already exists within namespace.')
- Container image \***: nginx
- Number of pods \***: 1
- Service \***: External (dropdown menu)
- Port \***: 80
- Target port \***: 80 (dropdown menu)
- Protocol \***: TCP (dropdown menu)
- Namespace \***: default (dropdown menu)

At the bottom, there are four buttons: 'Deploy' (highlighted), 'Preview', 'Cancel', and 'Show advanced options'.

Click Deploy

7) Now you will see something like this,



You will see the details for deployment below.

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

minikube service nginx

```

PS C:\Users\user> minikube service nginx
-----|-----|-----|-----|
| NAMESPACE | NAME   | TARGET PORT | URL               |
|-----|-----|-----|-----|
| default   | nginx  | tcp-80-80-mldbn/80 | http://192.168.49.2:31325 |
|-----|-----|-----|-----|
* Starting tunnel for service nginx.
-----|-----|-----|-----|
| NAMESPACE | NAME   | TARGET PORT | URL               |
|-----|-----|-----|-----|
| default   | nginx  |             | http://127.0.0.1:56123 |
|-----|-----|-----|-----|
* Opening service default/nginx in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.

```

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

# 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](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

## ➤ References for further study:

- 1) <https://kubernetes.io/docs/home/>
- 2) you need to learn Kubernetes RIGHT NOW!! - Networkchuck
- 3) techworld with Nana
- 4) Kubernetes Concepts Explained in 9 minutes! - KodeKloud
- 5) Kubernetes Tutorial for Beginners | Kubernetes Tutorial | Intellipaat
- 6) Install Kubernetes | Setup Kubernetes Step by Step | Kubernetes Training | Intellipaat