**KUBERNETES & GOOGLE CONTAINER ENGINES**

Advance in Data Sciences and Architecture

INFO 7390 - SPRING 2017

PROFESSOR:

**SRIKANTH KRISHNAMURTHY**

TEAM MEMBERS:

**AASHRI TANDON**

**PRAGATI SHAW**

**SARTHAK AGARWAL**

Table of Contents

[**Introduction** 3](#_Toc474434326)

[What are Kubernetes? 3](#_Toc474434327)

[Why Docker is not enough? 3](#_Toc474434328)

[Why Kubernetes? 4](#_Toc474434329)

[Architecture 4](#_Toc474434330)

[**Kubernetes on Local Machine** 6](#_Toc474434331)

[Prerequisites & Installation 6](#_Toc474434332)

[Steps for running Kubernetes on Local Machine 8](#_Toc474434333)

[Running a Workload on the Cluster 11](#_Toc474434334)

[Expose a Service 14](#_Toc474434335)

[Scaling the Service 16](#_Toc474434336)

[**Kubernetes on Google Container Engine** 17](#_Toc474434337)

[Overview 17](#_Toc474434338)

[Steps for running Kubernetes on GCE 18](#_Toc474434339)

# **Introduction**

## What are Kubernetes?

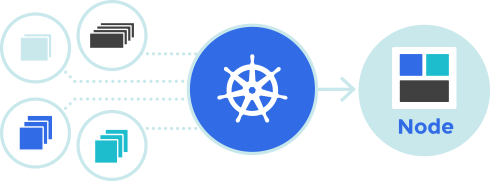
Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

With Kubernetes, we can quickly and efficiently respond to customer demand:

* Deploy your applications quickly and predictably.
* Scale your applications on the fly.
* Seamlessly roll out new features.
* Optimize use of your hardware by using only the resources you need.

## Why Docker is not enough?

Docker is great for managing images, and running containers in a specific host. Docker alone cannot manage containers across multiple nodes, or schedule and manage tasks to be completed across your cluster. In order to manage a Docker workload on a distributed cluster, Kubernetes is available to make working with Docker more efficient and streamlined. It is a solution for overseeing and managing multiple containers at scale, rather than just working with Docker on a manually-configured host.



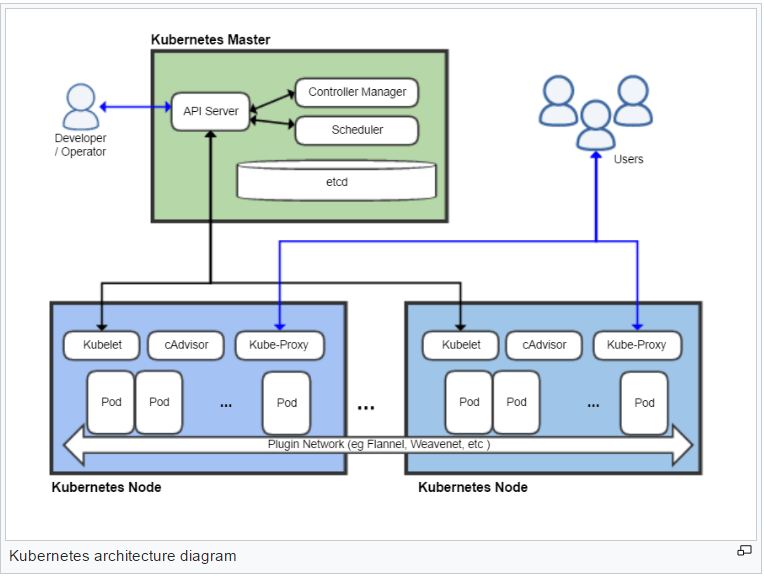
## Why Kubernetes?

Kubernetes can schedule and run application containers on clusters of physical or virtual machines. However, Kubernetes also allows developers to ‘cut the cord’ to physical and virtual machines, moving from a **host-centric** infrastructure to a **container-centric** infrastructure, which provides the full advantages and benefits inherent to containers. Kubernetes provides the infrastructure to build a truly **container-centric** development environment.

Kubernetes offers a few distinct advantages, first and foremost being that it packages all necessary tools -- orchestration, service discovery, load balancing, together in one nice package.

## Architecture

Kubernetes follows the master-slave architecture. The components of Kubernetes can be divided into those that manage an individual node and those that are part of the control plane.



While Kubernetes does work as a cohesive package, there are several components at play, each with a specific role. Kubernetes also has a specific collection of terms as mentioned below:

* ***Master*:** The managing machine, which oversees one or more minions.
* ***Node*:** A slave that runs tasks as delegated by the user and Kubernetes master. Also known as Worker or Minion is the single machine (or virtual machine) where containers(workloads) are deployed. Every node in the cluster must run the container runtime (such as Docker), as well as the below mentioned components, for communication with master for network configuration of these containers.
* ***Pod*:** An application (or part of an application) that runs on a minion. This is the basic unit of manipulation in Kubernetes.
* ***Replication Controller*:** It ensures that the requested number of pods are running on minions at all times.
* ***Label*:** An arbitrary key/value pair that the Replication Controller uses for service discovery
* ***kubectl*:** The command line config tool
* ***Service*:** An endpoint that provides load balancing across a replicated group of pods

To manage resources within Kubernetes, we interact with the Kubernetes API. Pulling down the Kubernetes binaries will give us all the services necessary to get your Kubernetes configuration up and running. Like most other cluster management solutions, Kubernetes works by creating a master, which exposes the Kubernetes API, allowing us to request certain tasks to be completed. The master then spawns containers to handle the workload we have asked for. Aside from running Docker, each node runs the Kubelet service, which is an agent that works with the container manifest and a proxy service. The Kubernetes control plane is comprised of many components, but they all run on the single Kubernetes master node.

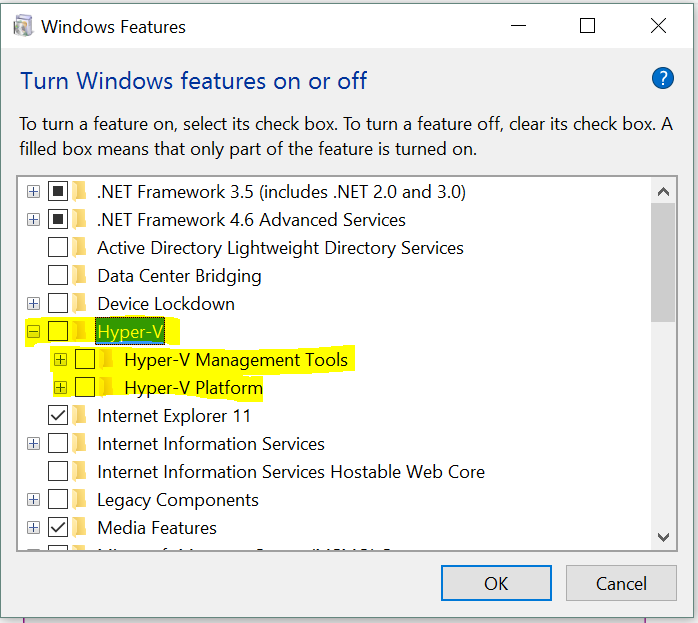
# **Kubernetes on Local Machine**

## Prerequisites & Installation

1. VirtualBox

Download [Docker Toolbox v1.12.0](https://www.docker.com/products/docker-toolbox), it comes packaged with VirtualBox and will automatically install it.

**Note:** Windows10 laptop ships with Hyperv and it does not go well with VirtualBox, hence you will need to disable it. Go to Turn Windows features on or off and you will see a dialog with list of Windows features as shown below. Navigate to the Hyper-V section and disable it completely.



1. Kubectl - command line utility

This is the CLI utility for the Kubernetes cluster and you need to install it and have it available in your PATH.

To install the latest 1.4 release, go the following URL:

<http://storage.googleapis.com/kubernetes-release/release/v1.4.0/bin/windows/amd64/kubectl.exe.>

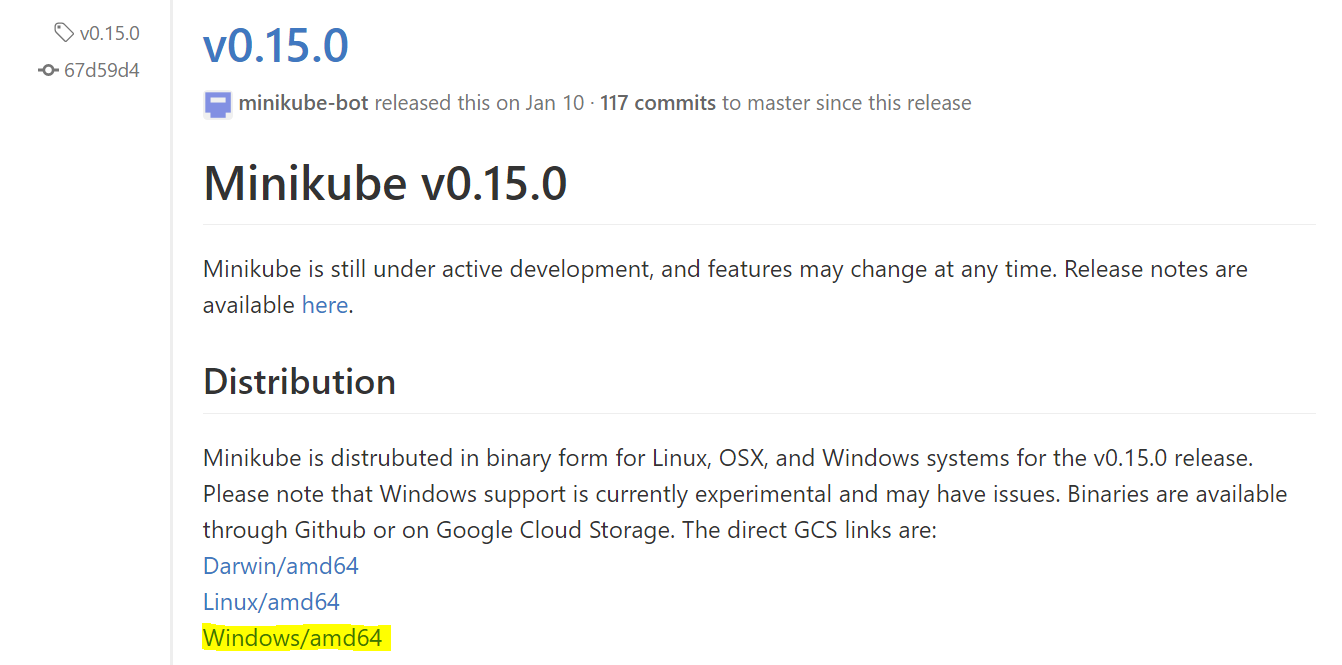
This will download the kubectl CLI executable. Place that in the C:\ folder and make it available in the environment PATH variable.

1. Minikube:

To install minikube go to the following URL:

<https://github.com/kubernetes/minikube/releases>

It will open a page like shown below:

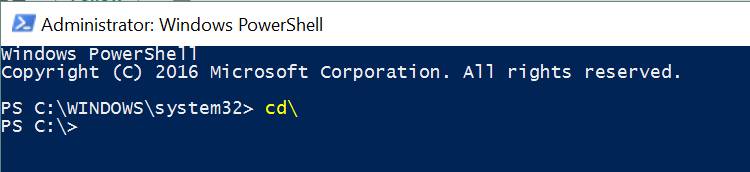


This will start downloading the executable. The file name is minikube-windows-amd64.exe. Just rename this to minikube.exe and place it in C:\ drive, alongside the kubectl.exe file from the previous section.

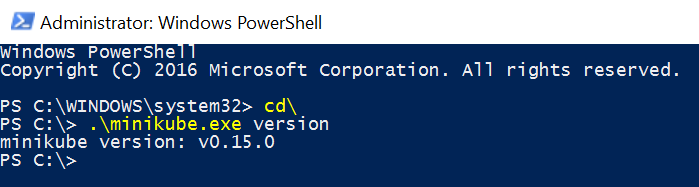
You are all set to launch a local Kubernets single node cluster but before let’s check if minikube is installed correctly.

**Check Minikube Installation**

Launch Powershell/Terminal in administrative mode and navigate to C:\ drive where the kubectl.exe and minikube.exe files are present.



Run the following command to check if minikube is working fine. It should be something like this.



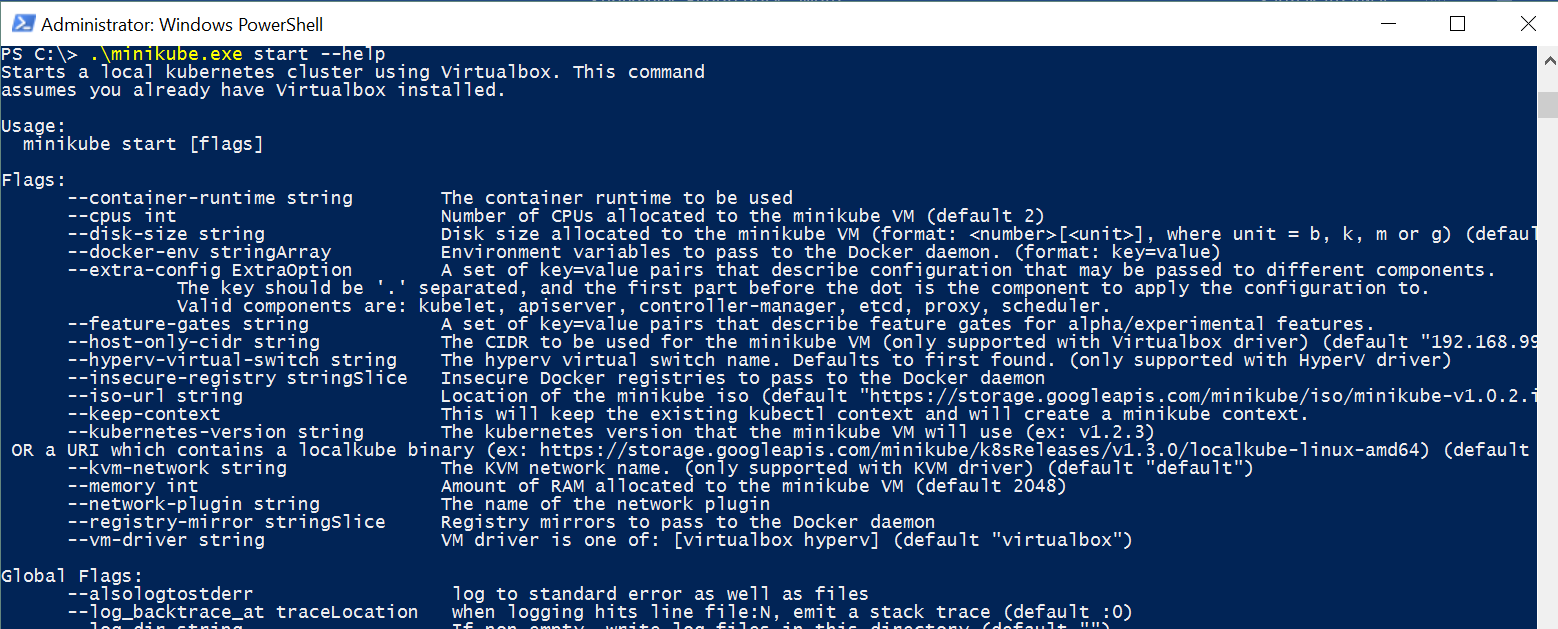
## Steps for running Kubernetes on Local Machine

1. **Start the Cluster**

Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.

To start the cluster, we will use START command.

You can check the usage of Start command by checking it help as shown below.



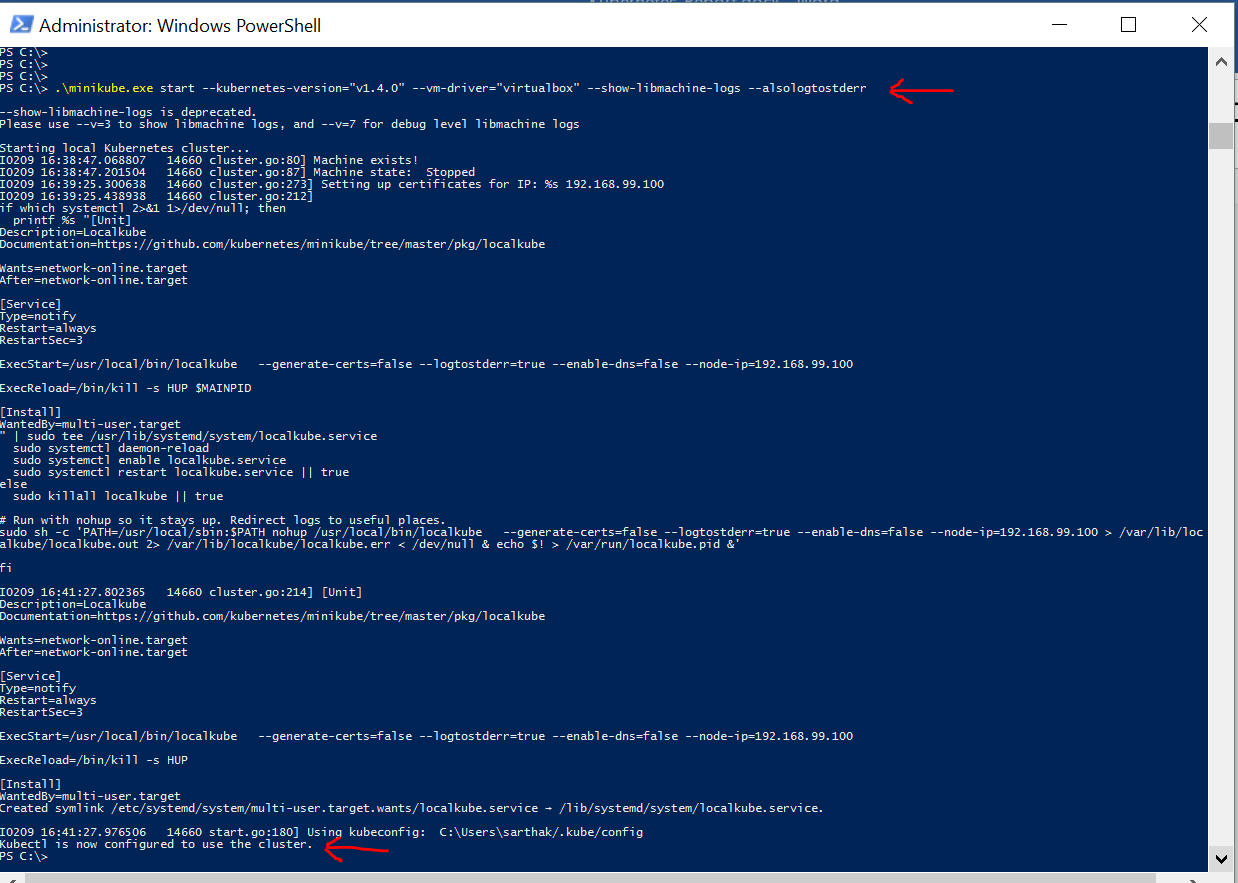
We will then start the cluster using the following command.

*.\minikube.exe start --kubernetes-version="v1.4.0" --vm-driver="virtualbox" --show-libmachine-logs --alsologtostderr*

This command does a couple of things:

* It generates the certificates and then proceeds to provision a local Docker host. This will result in a VM created inside of VirtualBox.
* That host is provisioned with the boot2Docker ISO image.
* It does its magic of setting it up, assigning it an IP and all the works.
* Finally, it prints out a message that kubectl is configured to talk to your local Kubernetes cluster.

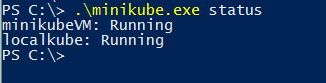
After the command has run successfully, it will look something like this and in the end it will say “Kubectl is now configured to use the cluster”:



1. **Check if the Cluster is running**

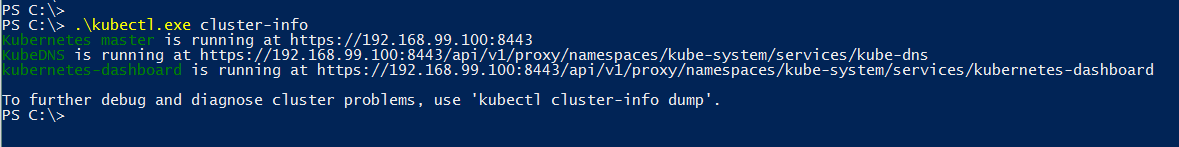
Execute the following command to check the status of the cluster.

*.\minikube.exe status*



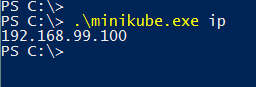
You can also use kubectl to get the cluster information like shown below.

*.\kubectl.exe cluster-info*



1. **Check the IP Address of the cluster**

*.\minikube.exe ip*

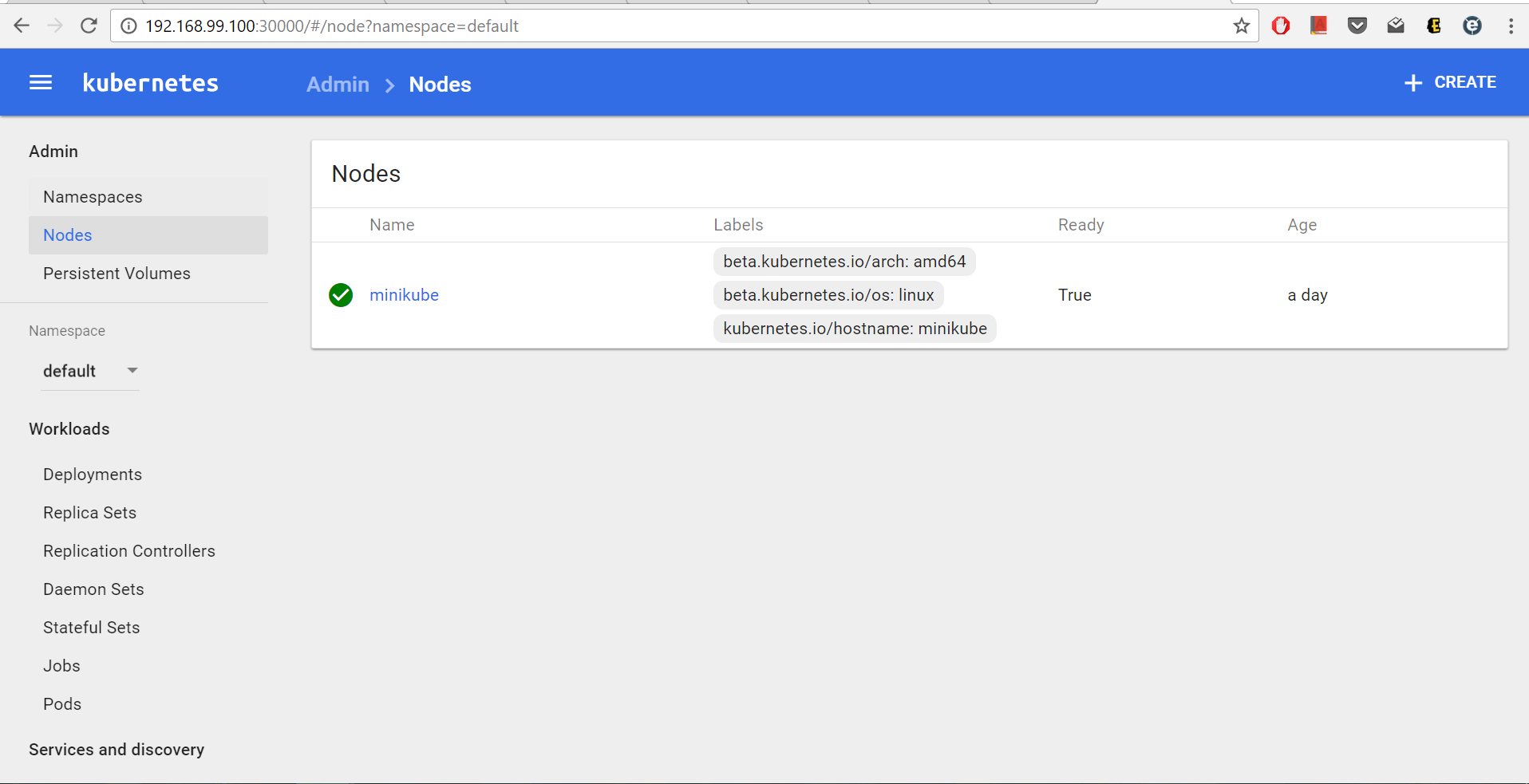


1. **Launch Kubernets Dashboard**

Execute the following command

*.\minikube.exe dashboard*

This will automatically launch the Dashboard in your local browser and it will look something like this:



## Running a Workload on the Cluster

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes Deployment. The Deployment is responsible for creating and updating instances of your application.

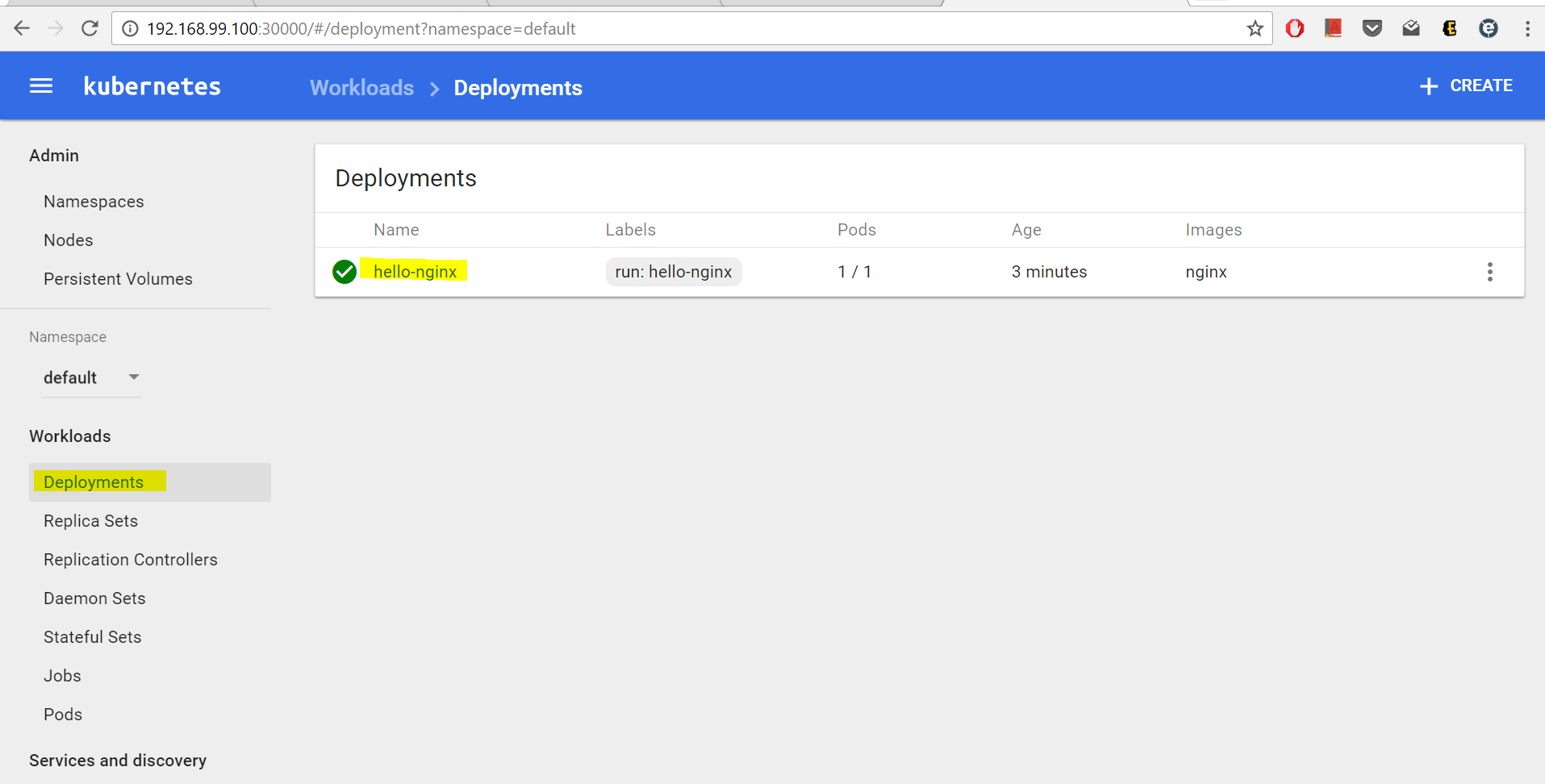
We will now run a simple [Nginx](https://hub.docker.com/_/nginx/) container on our cluster.

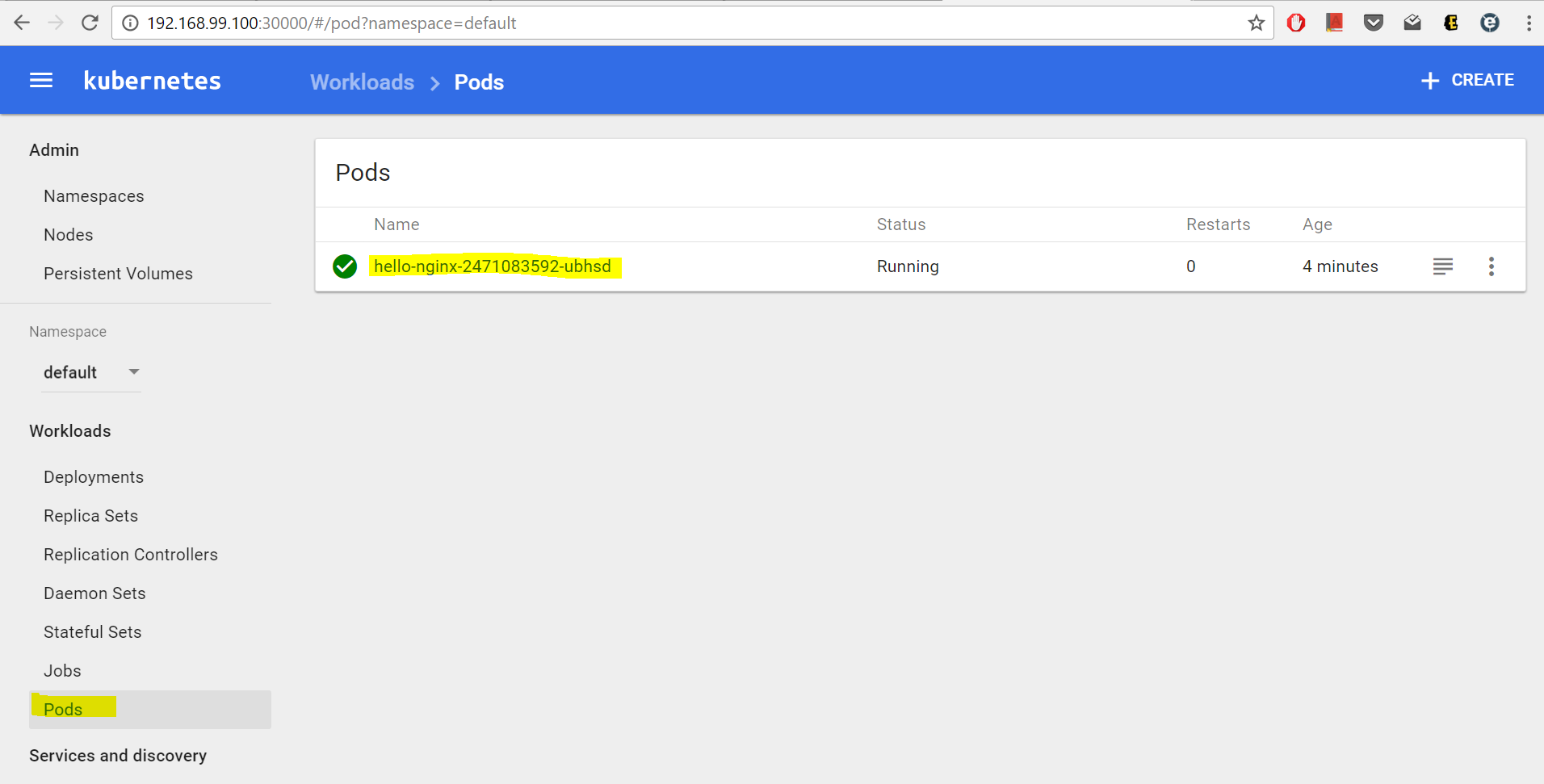
Execute the following command:

*.\kubectl.exe run hello-nginx --image=nginx --port=80*

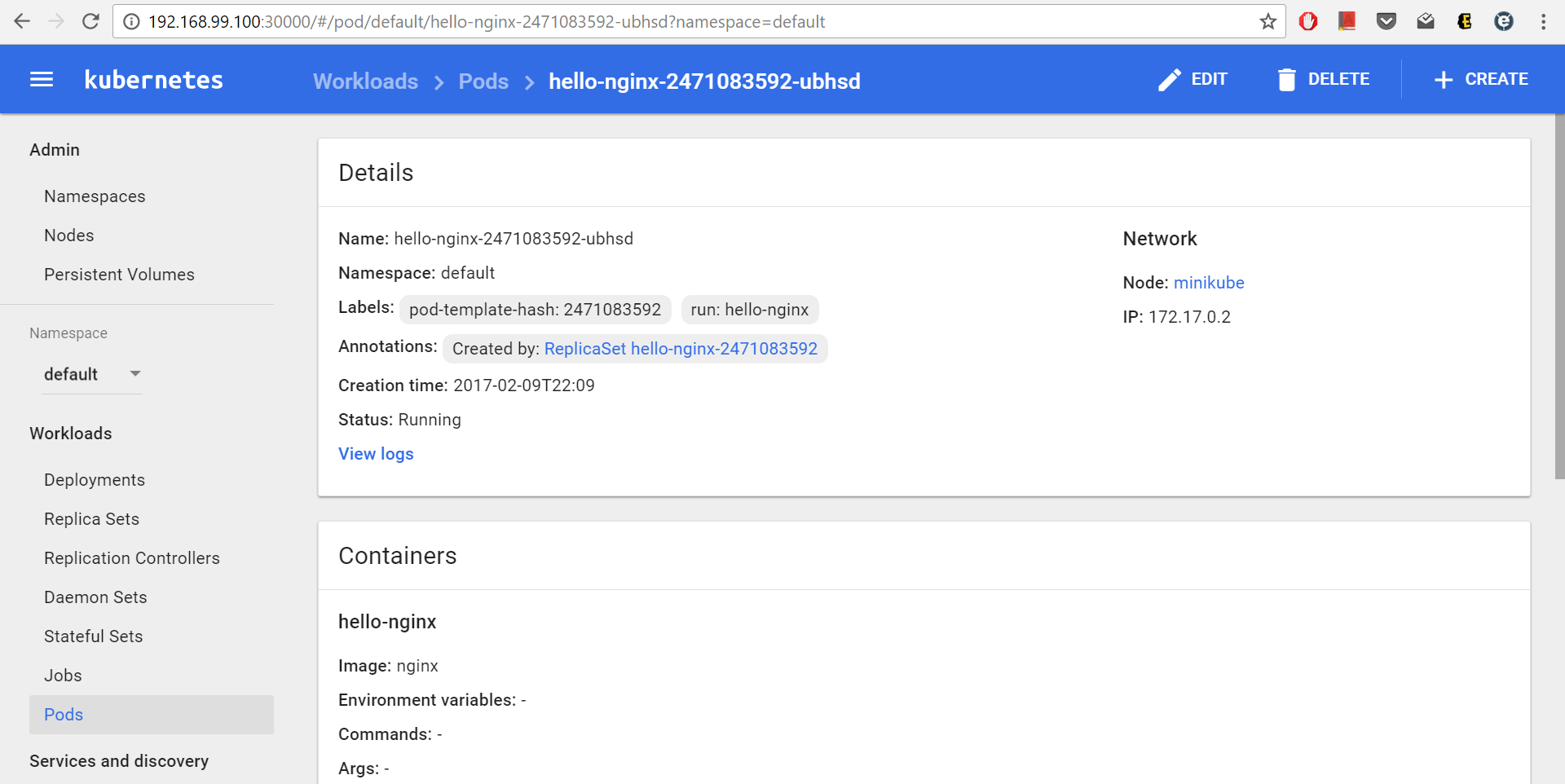


This will create a deployment and investigate into Pod that got created by checking the dashboard.



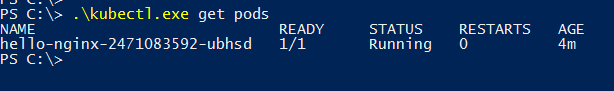


We can look at the details by clicking on the Name of the Pod and it will look something like this.



We can also check the Pod using the command line using the following command:

*.\kubectl.exe get pods*



## Expose a Service

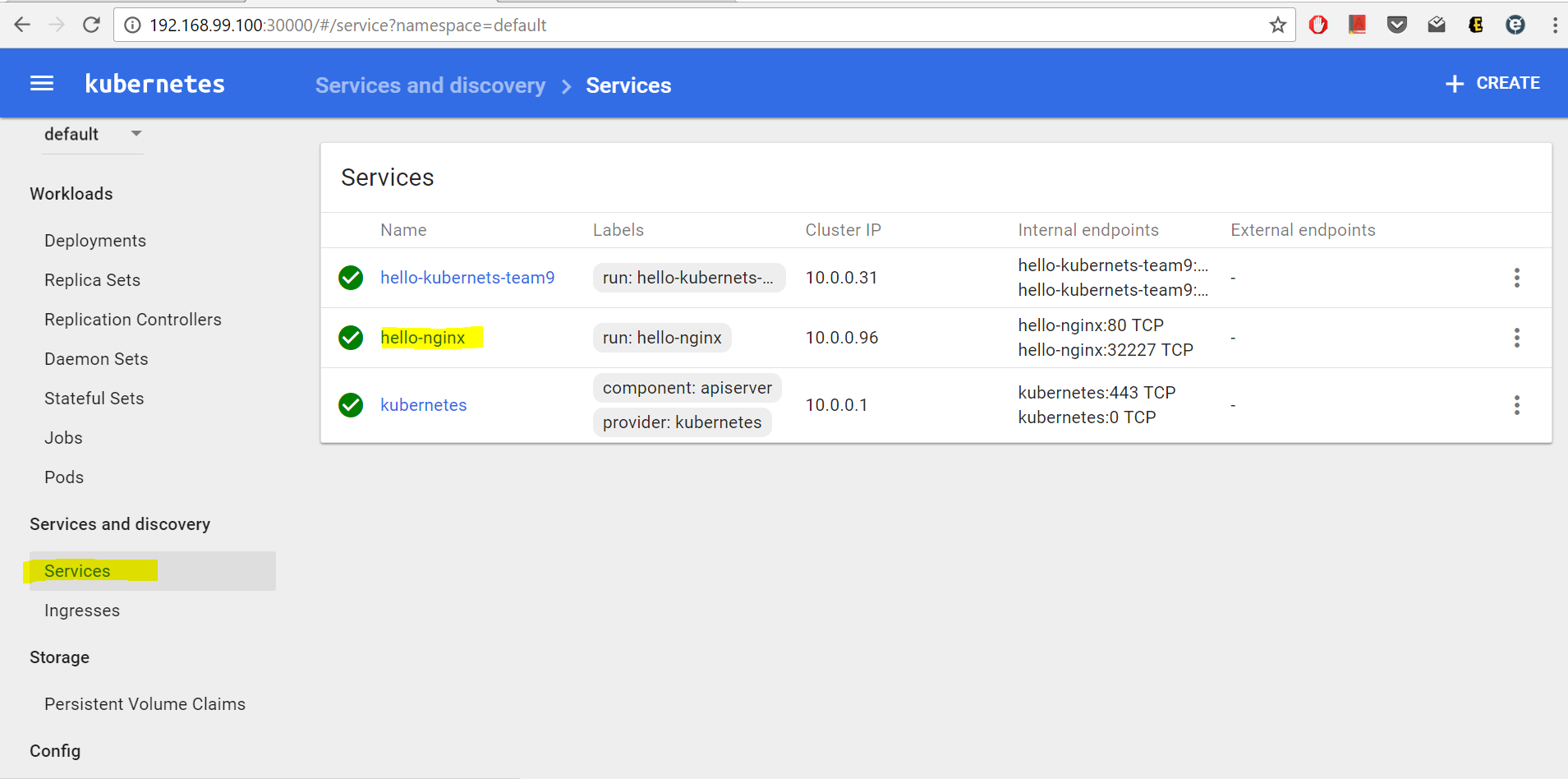
While Pods do have their own unique IP across the cluster, those IP's are not exposed outside Kubernetes. Taking into account that over time Pods may be terminated, deleted or replaced by other Pods, we need a way to let other Pods and applications automatically discover each other. Kubernetes addresses this by grouping Pods in Services. A Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods.

Execute the following command to expose our deployment:

*.\kubectl.exe expose deployment hello-nginx --type=NodePort*



If we visit the Dashboard at this point and go to the Services section, we can see out **hello-nginx** service entry.

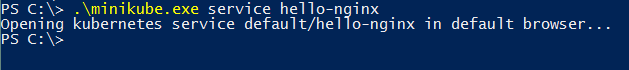


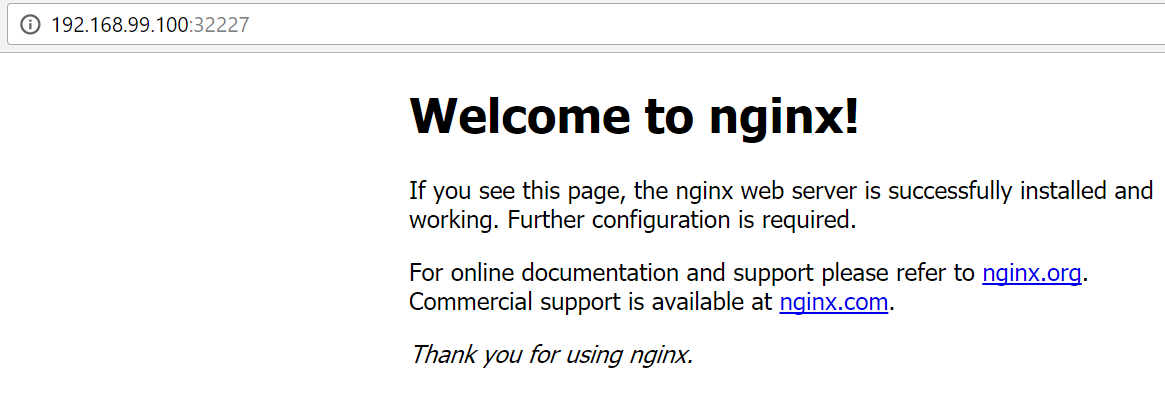
Alternately, we can use kubectl too, to check it out:

*.\kubectl.exe get services*

We can now use the minikube service to directly launch the browser and hit the service endpoint:

*.\minikube.exe service hello-nginx*





## Scaling the Service

The Deployment above created only one Pod for running our application. When traffic increases, we will need to scale the application to keep up with user demand. Scaling is accomplished by changing the number of replicas in a Deployment.

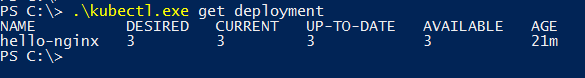
Suppose we want to scale it to 3 Pods. The command for it will be.

*.\kubectl scale --replicas=3 deployment/hello-nginx*

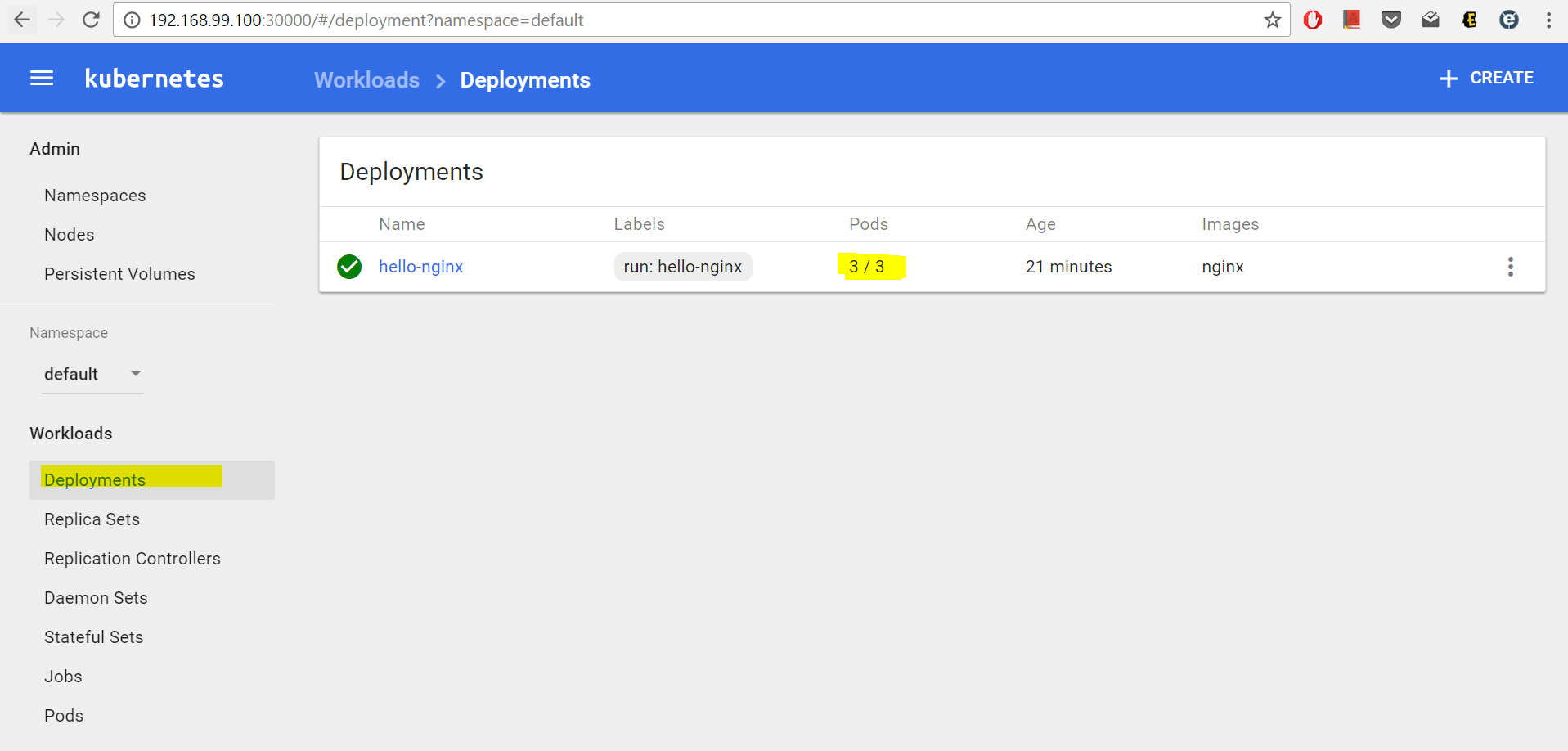


We can check the status of the deployment using the command line as:

**.\kubectl.exe get deployment**



Or, we can check the dashboard and it will look something like this:



# **Kubernetes on Google Container Engine**

## Overview

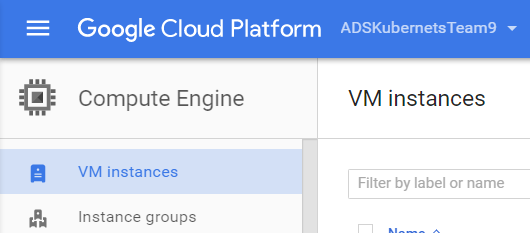
Google Container Engine makes it easy to run Docker containers in the cloud. It uses Kubernetes, an open source container scheduler, to ensure that your cluster is running exactly the way you want it to at all times.

A Container Engine cluster is a group of Compute Engine instances running Kubernetes. It consists of one or more node instances, and a managed Kubernetes master endpoint. Every container cluster has a single master endpoint, which is managed by Container Engine. The master provides a unified view into the cluster and, through its publicly-accessible endpoint, is the doorway for interacting with the cluster.

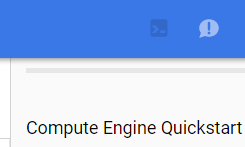
The managed master also runs the Kubernetes API server, which services REST requests, schedules pod creation and deletion on worker nodes, and synchronizes pod information (such as open ports and location) with service information.

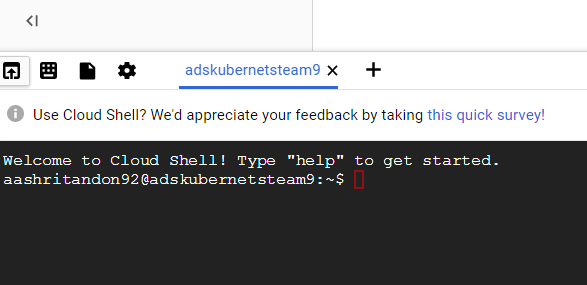
## Steps for running Kubernetes on GCE

* Sign-in to Google Cloud Platform console and create a new project
* Search for "Google Compute Engine" in the search box, and click on Enable Billing

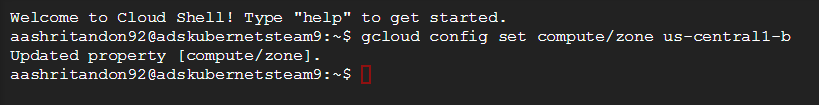


* Launch Google Cloud Shell by clicking on this icon



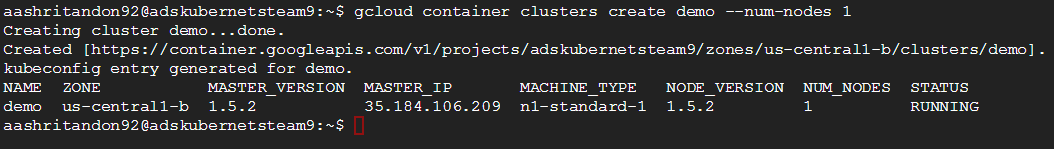


* We need to set the compute zone so that the virtual machines in our cluster are created in the correct region

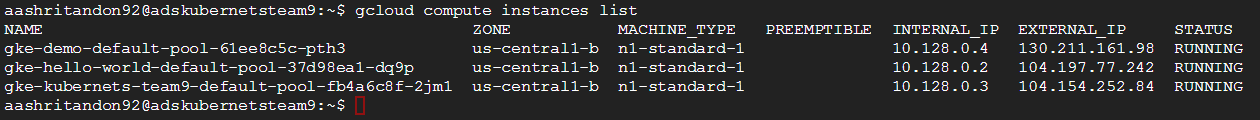


* We can create a new container cluster with the gcloud command like this:

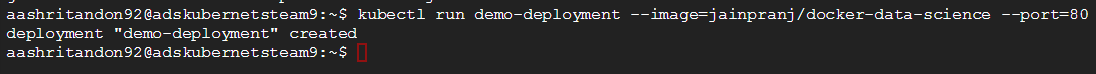




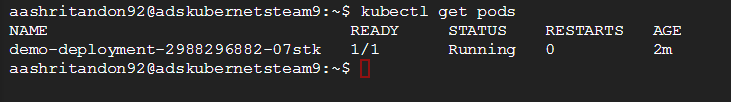
* We can also verify that the cluster started successfully by checking the instances list:



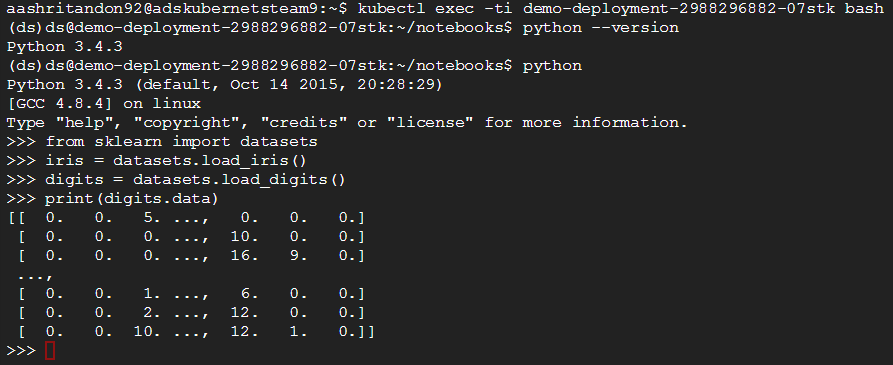
* We can create a pod by using the following command. This command starts up the docker image on one of the nodes in the cluster.



* We get the number of pods running using the following command



* Running the application through bash command on Google Cloud



Instances running on google cloud

