

Tutorial : Getting Started with Kubernetes on your Windows Laptop with Minikube

I personally like the simplicity of Docker Swarm and have found in my teaching experience with developers, that it was easier for most people to understand what Container Management solutions are all about when they see a few simple commands in Docker Swarm and are able to relate to stuff like scaling up/down, rolling updates, etc.



Personally I think if you are looking for a container management solution in today's world, you have to invest your time in [Kubernetes](#) (k8s). There is no doubt about that because of multiple factors. To the best of my understanding, these points include:

- Kubernetes is [Open Source](#)
- Great momentum in terms of activities & contribution at its [Open Source Project](#)
- Decades of experience running its [predecessor](#) at Google
- Support of multiple OS and infrastructure software vendors
- Rate at which features are being released
- Production readiness (Damn it, [Pokemon Go](#) met its scale due to Kubernetes)
- Number of features available. Check out the list of features at the [home page](#).

What this post is about?

I do not want to spend time explaining about what Kubernetes is and its building blocks like Pods, Replication Controllers, Services, Deployments

and more. There are multiple articles on that and I suggest that you go through it.

I have written a couple of other articles that go through a high level overview of Kubernetes:

- [Introduction to Kubernetes](#)
- [Kubernetes Building Blocks](#)

It is important that you go through some basic material on its concepts, so that we can directly get down into its commands.

The general perception about a management solution like Kubernetes is that it would require quite a bit of setup for you to try it out locally. What this means is that it would take some time to set it up but more than setting it up, you might probably get access to it only during staging phase or something like that. Ideally you want a similar environment in your development too, so that you are as close to what it takes to run your application. The implications of this is that you want it running on your laptop/desktop, where you are likely to do your development.

This was the goal behind the [minikube](#) project and the team has put in fantastic effort to help us setup and run Kubernetes on our development machines. This is as simple and portable as it can get. The tagline of minikube project says it all: **“Run Kubernetes locally”**.



minikube

This post is going to take you through setting up Minikube on your Windows development machine and then taking it for a Hello World spin to see a local Kubernetes cluster in action. Along the way, I will highlight my environment and what I had to do to get the experimental build of

minikube working on my Windows machine. Yes, it is experimental software, but it works!

If you are not on Windows, the instructions to setup minikube on either your Linux machine or Mac machine are also available [here](#). Check it out. You can then safely skip over the setup and go to the section where we do a quick Hello World to test drive Kubernetes locally.

Keep in mind that Minikube gives you a single node cluster that is running in a VM on your development machine.

Of course, once you are done with what you see in this blog, I strongly recommend that you also look at Managed Container Orchestration solutions like Google Container Engine.

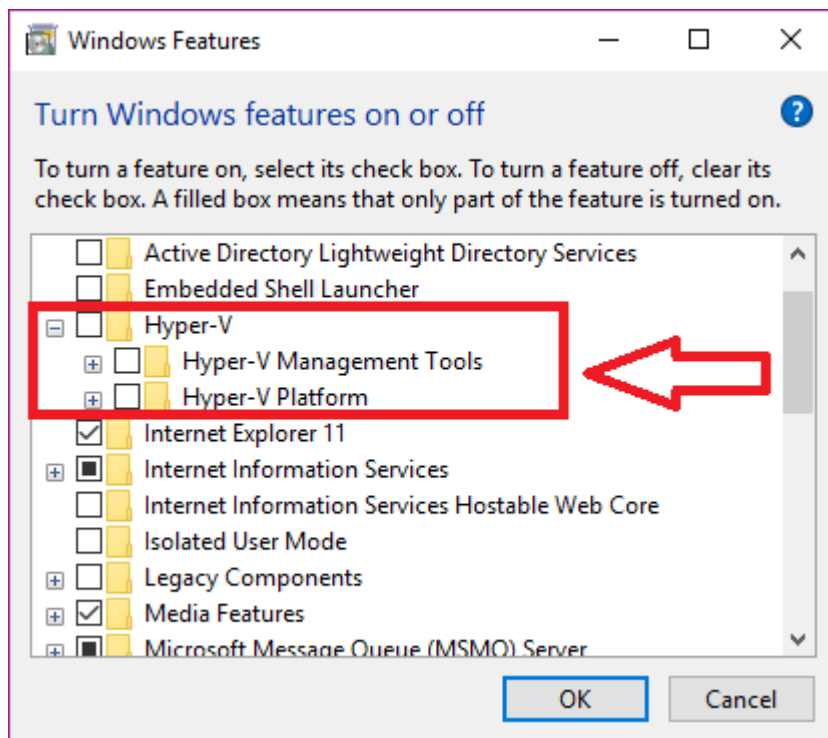
Let's get started now with installation of minikube. But first, we must make sure that our development machine has some of the pre-requisites required to run it. Do not ignore that!

Using VirtualBox and not Hyper-V

VirtualBox and Hyperv (which is available on Windows 10) do not make a happy pair and you are bound to run into situations where the tools get confused. I preferred to use VirtualBox and avoid all esoteric command-line switches that we need to provide to enable creation of the underlying Docker hosts, etc.

To disable Hyper-V, 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.

This will require a restart to the machine to take effect and on my machine, it even ended up doing a Windows Update, configuring it and a good 10 minutes later, it was back up.



Great! We have everything now to get going.

Development Machine Environment

I am assuming that you have a setup that is similar to this. I believe, you should be fine on Windows 7 too and it would not have the HyperV stuff, instructions of which I will give in a while.

- Windows 10 Laptop. VT-x/AMD-v virtualization must be enabled in BIOS.
- [Docker Toolbox v1.12.0](#). The toolbox sets up VirtualBox and I have gone with that.
- **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, do the following: Go to the browser and give the following URL : <http://storage.googleapis.com/kubernetes-release/release/v1.4.0/bin/windows/amd64/kubect.exe>. This will download the kubectl CLI executable. Please make it available in the environment PATH variable.

Note: kubectl versions are available at a generic location as per the following format: [https://storage.googleapis.com/kubernetes-release/release/\\${K8S_VERSION}/bin/\\${GOOS}/\\${GOARCH}/\\${K8S_BINARY}](https://storage.googleapis.com/kubernetes-release/release/${K8S_VERSION}/bin/${GOOS}/${GOARCH}/${K8S_BINARY})

Minikube installation

The first step is to take the **kubectl.exe** file that you downloaded in the previous step and place that in the **C:** folder.

The next step is to download the minikube binary from the following location: <https://github.com/kubernetes/minikube/releases>

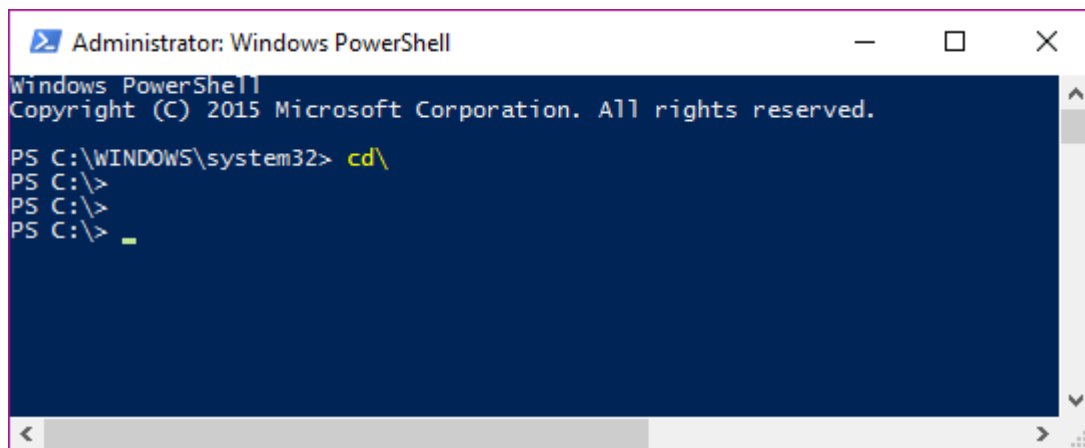
Go to the Windows download link as shown below:



This will start downloading the **v0.10.0** release of 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 now to launch a local Kubernetes one -node cluster!

All the steps moving forward are being done in Powershell. Launch Powershell in Administrative mode (Ctrl-Shift-Enter) and navigate to C:\ drive where the kubectl.exe and minikube.exe files are present.

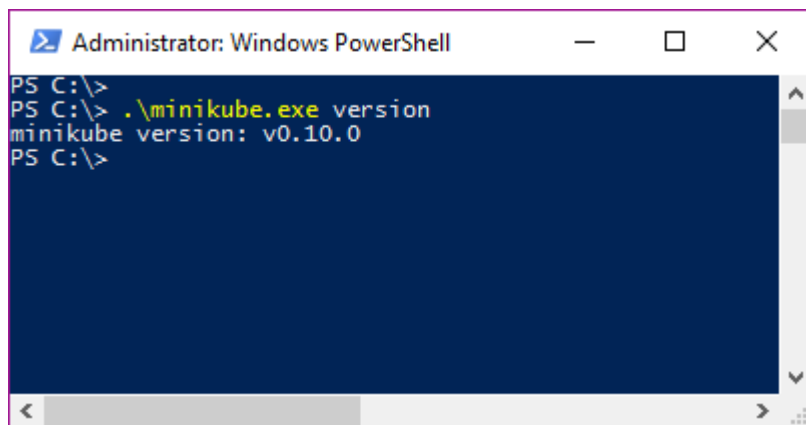
A screenshot of a Windows PowerShell terminal window titled "Administrator: Windows PowerShell". The terminal shows the following commands and output:

```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> cd\
PS C:\>
PS C:\>
PS C:\> _
```

A few things to note

Let's do our standard testing to validate our utilities.

A screenshot of a Windows PowerShell terminal window titled "Administrator: Windows PowerShell". The terminal shows the following commands and output:

```
PS C:\>
PS C:\> .\minikube.exe version
minikube version: v0.10.0
PS C:\>
```

If you go to your %HOMEPATH%\minikube folder now, you will notice that several folders got created. Take a look!

There are multiple commands that Minikube supports. You can use the standard `--help` option to see the list of commands that it has:

```
PS C:\> .\minikube --help
Minikube is a CLI tool that provisions and manages
single-node Kubernetes clusters optimized for
development workflows
Usage:
  minikube [command]
Available Commands:
  dashboard      Opens/displays the kubernetes
dashboard URL for your local cluster
```

`delete` Deletes a local kubernetes cluster.
`docker-env` sets up docker env variables; similar to '`$(docker-machine env)`'
`get-k8s-versions` Gets the list of available kubernetes versions available for minikube.
`ip` Retrieve the IP address of the running cluster.
`logs` Gets the logs of the running localkube instance, used for debugging minikube, not user code.
`config` Modify minikube config
`service` Gets the kubernetes URL for the specified service in your local cluster
`ssh` Log into or run a command on a machine with SSH; similar to '`docker-machine ssh`'
`start` Starts a local kubernetes cluster.
`status` Gets the status of a local kubernetes cluster.
`stop` Stops a running local kubernetes cluster.
`version` Print the version of minikube.

Flags:

`--alsologtostderr[=false]`: log to standard error as well as files
`--log-flush-frequency=5s`: Maximum number of seconds between log flushes
`--log_backtrace_at=:0`: when logging hits line file:N, emit a stack trace
`--log_dir=""`: If non-empty, write log files in this directory
`--logtostderr[=false]`: log to standard error instead of files
`--show-libmachine-logs[=false]`: Whether or not to show logs from libmachine.
`--stderrthreshold=2`: logs at or above this threshold go to stderr
`--v=0`: log level for V logs
`--vmodule=`: comma-separated list of pattern=N settings for file-filtered logging

Use "minikube [command] --help" for more information about a command.

I have highlighted a couple of **Global flags** that you can use in all the commands for minikube. These flags are useful to see what is going on inside the hood at times and also for seeing the output on the standard output (console/command).

Minikube supports multiple versions of Kubernetes and the latest version is v1.4.0. To check out the different versions supported try out the following command:

```
PS C:\> .\minikube get-k8s-versions
The following Kubernetes versions are available:
    - v1.4.0
    - v1.3.7
    - v1.3.6
    - v1.3.5
    - v1.3.4
    - v1.3.3
    - v1.3.0
```

Starting our Cluster

We are now ready to launch our Kubernetes cluster locally. We will use the **start** command for it.

Note: You might run into multiple issues while starting a cluster the first time. I have several of them and have created a section at the end of this blog post on Troubleshooting. Take a look at it, in case you run into any issues.

You can check out the help and description of the command/flags/options via the help option as shown below:

```
PS C:\> .\minikube.exe start --help
```


You will notice several **Flags** that you can provide to the **start** command and while there are some useful defaults, we are going to be a bit specific, so that we can better understand things.

We want to use Kubernetes v1.4.0 and while the VirtualBox driver is default on windows, we are going to be explicit about it. At the same time, we are going to use a couple of the Global Flags that we highlighted earlier, so that we can see what is going on under the hood.

All we need to do is give the following command (I have separated the flags on separate line for better readability). The output is also attached.

```
PS C:\> .\minikube.exe start --kubernetes-  
version="v1.4.0"  
  
--vm-  
driver="virtualbox"  
  
--show-libmachine-  
logs --alsologtostderr  
W1004 13:01:30.429310      9296 root.go:127] Error  
reading config file at  
C:\Users\irani_r\.minikube\config\config.json: o  
pen C:\Users\irani_r\.minikube\config\config.json:  
The system cannot find the file specified.  
I1004 13:01:30.460582      9296 notify.go:103]  
Checking for updates...  
Starting local Kubernetes cluster...  
Creating CA:  
C:\Users\irani_r\.minikube\certs\ca.pem  
Creating client certificate:  
C:\Users\irani_r\.minikube\certs\cert.pemRunning  
pre-create checks...  
Creating machine...  
(minikube) Downloading  
C:\Users\irani_r\.minikube\cache\boot2docker.iso  
from file://C:/Users/irani_r/.minikube/cache/iso  
/minikube-0.7.iso...  
(minikube) Creating VirtualBox VM...  
(minikube) Creating SSH key...  
(minikube) Starting the VM...
```

```

(minikube) Check network to re-create if needed...
(minikube) Waiting for an IP...
Waiting for machine to be running, this may take a
few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote
daemon...
Checking connection to Docker...
Docker is up and running!
I1004 13:03:06.480550      9296 cluster.go:389]
Setting up certificates for IP: %s 192.168.99.100
I1004 13:03:06.567686      9296 cluster.go:202] sudo
killall localkube || true
I1004 13:03:06.611680      9296 cluster.go:204]
killall: localkube: no process killed
I1004 13:03:06.611680      9296 cluster.go:202]
# Run with nohup so it stays up. Redirect logs to
useful places.
sudo sh -c 'PATH=/usr/local/sbin:$PATH nohup
/usr/local/bin/localkube --generate-certs=false
--logtostderr=true --node
-ip=192.168.99.100 >
/var/lib/localkube/localkube.err 2>
/var/lib/localkube/localkube.out < /dev/null &
echo $! > /var/r
un/localkube.pid &'
I1004 13:03:06.658605      9296 cluster.go:204]
Kubectl is now configured to use the cluster.
PS C:\>

```

Let us understand what it is doing behind the scenes in brief. I have also highlighted some of the key lines in the output above:

1. It generates the certificates and then proceeds to provision a local Docker host. This will result in a VM created inside of VirtualBox.

2. That host is provisioned with the boot2Docker ISO image.
3. It does its magic of setting it up, assigning it an IP and all the works.
4. Finally, it prints out a message that kubectl is configured to talk to your local Kubernetes cluster.

You can now check on the status of the local cluster via the **status** command:

```
PS C:\> .\minikube.exe status
minikubeVM: Running
localkube: Running
```

You can also use the kubectl CLI to get the cluster information:

```
PS C:\> .\kubectl.exe cluster-info
Kubernetes master is running at
https://192.168.99.100:8443
kubernetes-dashboard is running at
https://192.168.99.100:8443/api/v1/proxy/namespaces/kube-system/services/kubernetes-d
ashboard
To further debug and diagnose cluster problems,
use 'kubectl cluster-info dump'.
```

Kubernetes Client and Server version

Let us do a quick check of the Kubernetes version at the client and server level. Execute the following command:

```
PS C:\> .\kubectl version
Client Version: version.Info{Major:"1", Minor:"4",
GitVersion:"v1.4.0",
GitCommit:"a16c0a7f71a6f93c7e0f222d961f4675cd97a
46b", GitTreeState:"clean", BuildDate:"2016-09-
26T18:16:57Z", GoVersion:"go1.6.3", Compiler:"gc",
Platform:"windows/amd6
4"}
Server Version: version.Info{Major:"1", Minor:"4",
GitVersion:"v1.4.0",
GitCommit:"a16c0a7f71a6f93c7e0f222d961f4675cd97a
```

```
46b", GitTreeState:"dirty", BuildDate:"1970-01-01T00:00:00Z", GoVersion:"go1.7.1", Compiler:"gc", Platform:"linux/amd64"
}
```

You will notice that both client and server are at version **1.4**.

Cluster IP Address

You can get the IP address of the cluster via the **ip** command:

```
PS C:\> .\minikube.exe ip
192.168.99.100
```

Kubernetes Dashboard

You can launch the Kubernetes Dashboard at any point via the **dashboard** command as shown below:

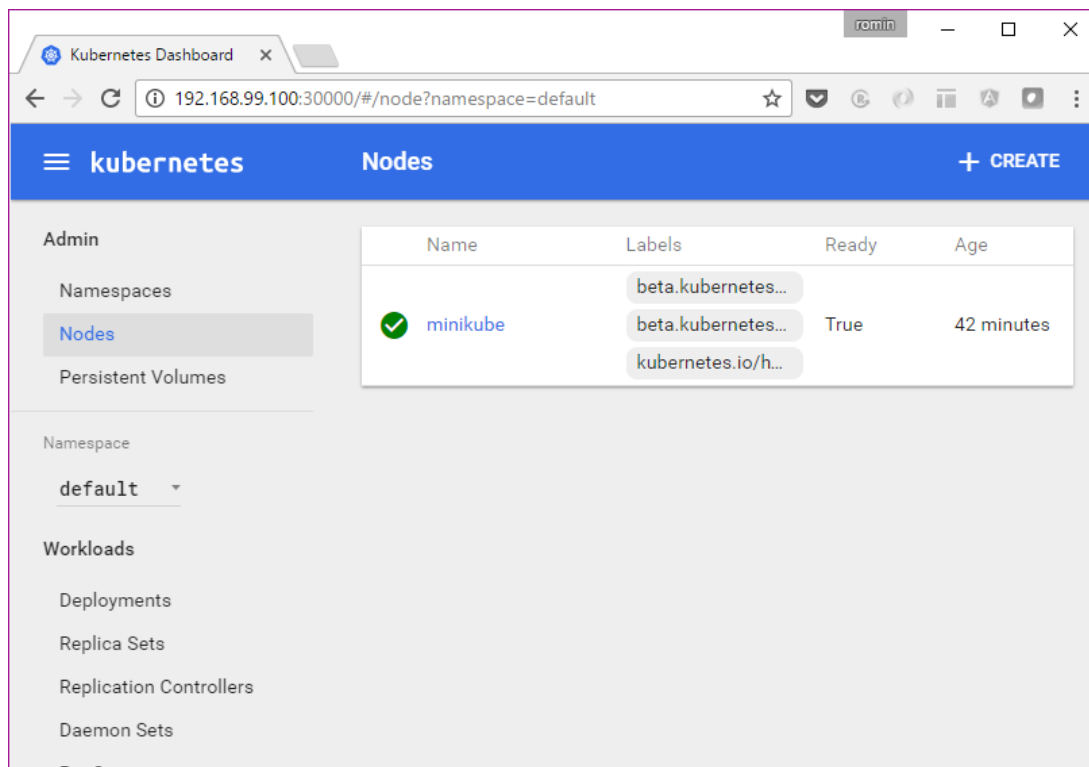
```
PS C:\> .\minikube.exe dashboard
```

This will automatically launch the Dashboard in your local browser. However if you just want to nab the Dashboard URL, you can use the following flag:

```
PS C:\> .\minikube.exe dashboard --url=true
http://192.168.99.100:30000
```

There is a [great post](#) on how the Kubernetes Dashboard underwent a design change in version 1.4. It explains how the information is split up into respective sections i.e. Workloads , Services and Discovery, Storage and Configuration, which are present on the left -side menu and via which you can sequentially introspect more details of your cluster. All of this is provided by a nifty filter for the Namespace value above.

If you look at the Kubernetes dashboard right now, you will see that it indicates that nothing has been deployed. Let us step back and think what we have so far. We have launched a **single-node cluster** .. right? Click on the Node link and you will see that information:



The above node information can also be obtained by using the kubectl CLI to get the list of nodes.

```
PS C:\> .\kubectl.exe get nodes
NAME          STATUS    AGE
minikube      Ready     51m
```

Hopefully, you are now able to relate how some of the CLI calls are reflected in the Dashboard too. Let's move forward. But before that, one important tip!

Tip: use-context minikube

If you had noticed closely when we started the cluster, there is a statement in the output that says **"Kubectl is now configured to use the cluster."** What this is supposed to do is to eventually set the current context for the kubectl utility so that it knows which cluster it is talking to. Behind the scenes in your %HOMEPATH%\kube directory, there is a config file that contains information about your Kubernetes cluster and the details for connecting to your various clusters is present over there.

In short, we have to be sure that the kubectl is pointing to the right cluster. In our case, the cluster name is minikube.

In case you see an error like the one below (I got it a few times), then you need to probably set the context again.

```
PS C:\> kubectl get nodes
error: You must be logged in to the server (the
server has asked for the client to provide
credentials)
```

The command for that is:

```
PS C:\> kubectl config use-context minikube
switched to context "minikube".
```

Running a Workload

Let us proceed now to running a simple [Nginx container](#) to see the whole thing in action:

We are going to use the run command as shown below:

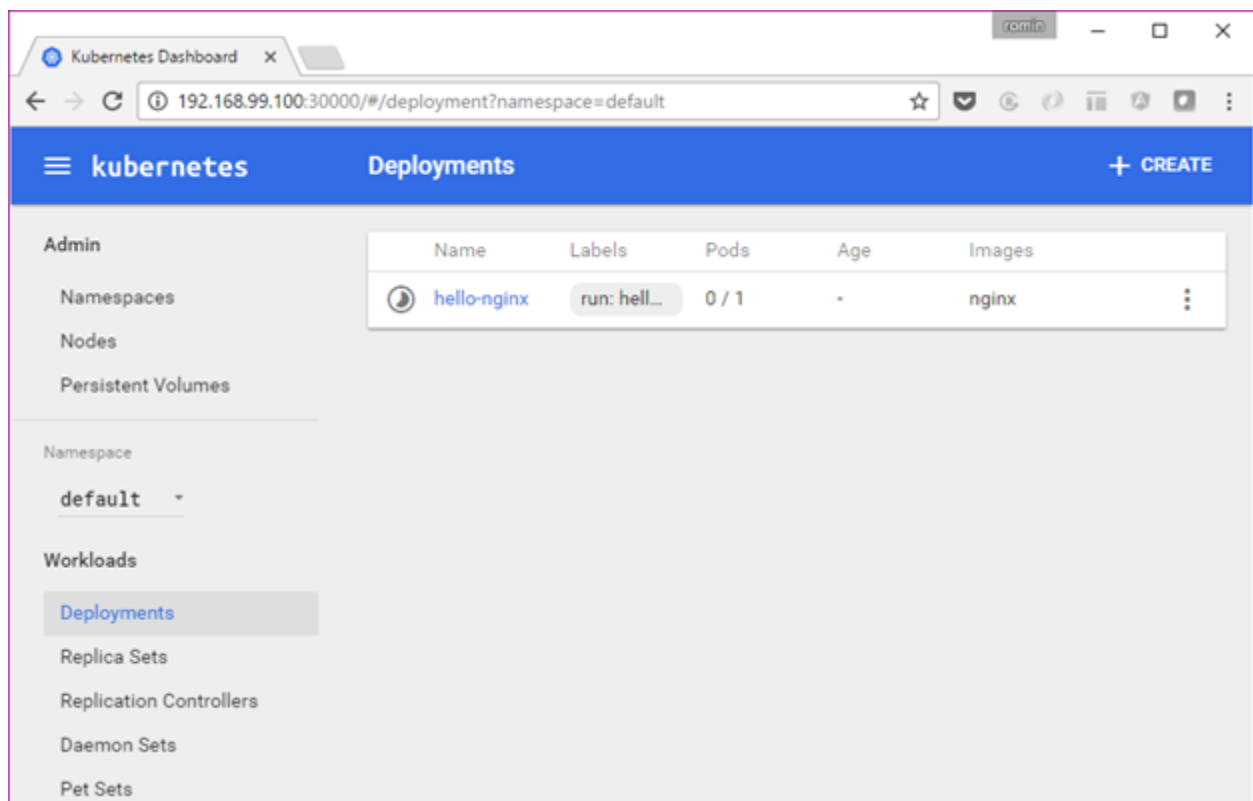
```
PS C:\> .\kubectl.exe run hello-nginx --
image=nginx --port=80
deployment "hello-nginx" created
```

This creates a deployment and we can investigate into the Pod that gets created, which will run the container:

```
PS C:\> .\kubectl.exe get pods
NAME                                READY   STATUS
RESTARTS   AGE
hello-nginx-24710...    0/1     ContainerCreating
0                2m
```

You can see that the STATUS column value is **ContainerCreating**.

Now, let us go back to the Dashboard (I am assuming that you either have it running or can launch it again via the **minikube dashboard** command):

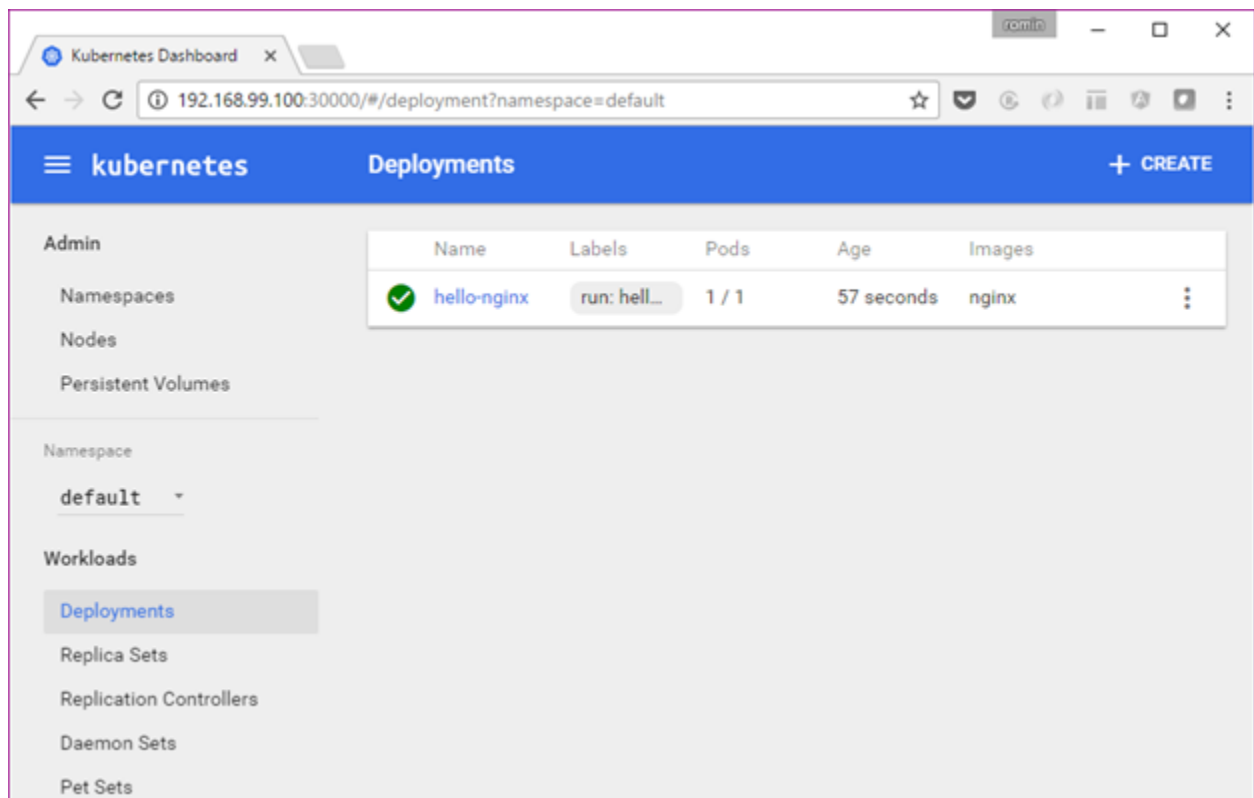


You can notice that if we go to the Deployments option, the Deployment is listed and the status is still in progress. You can also notice that the Pods value is 0/1.

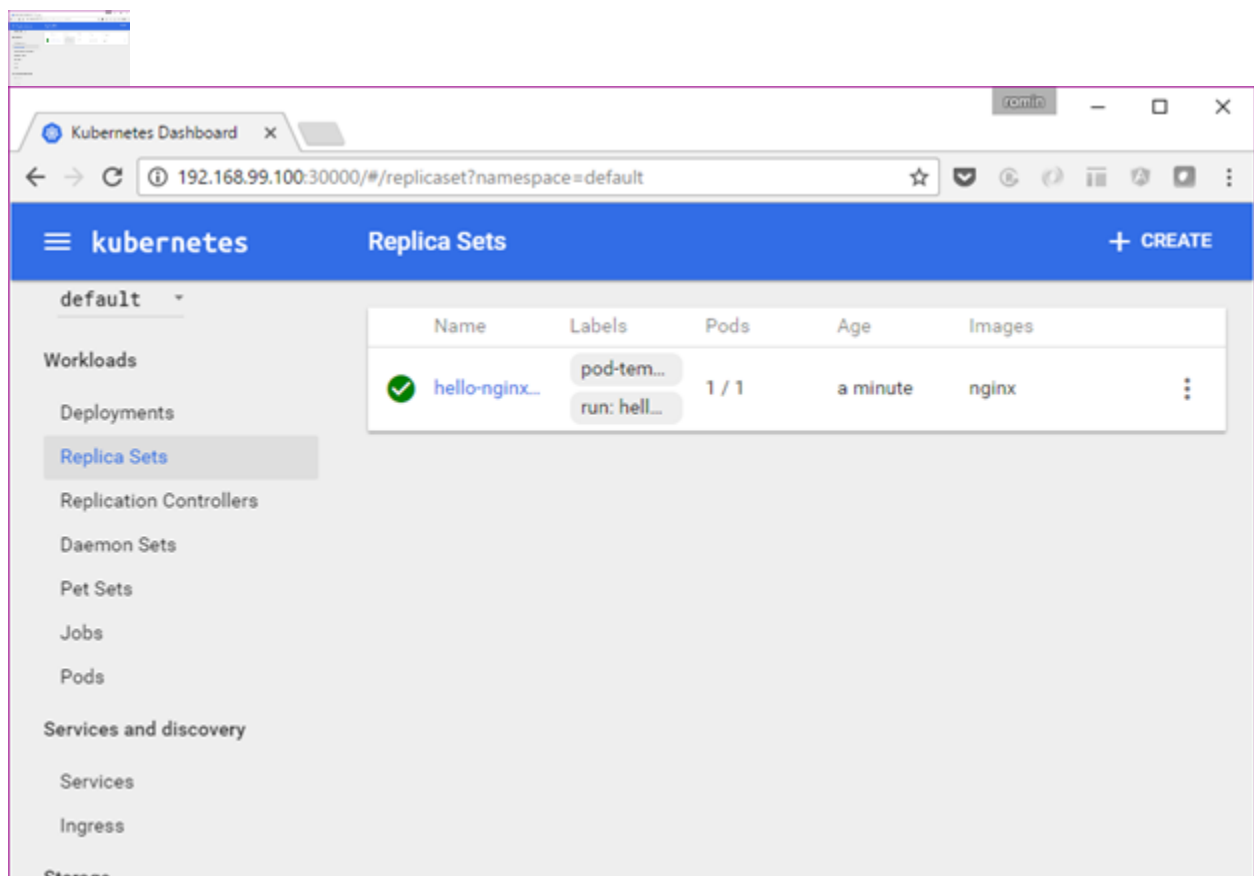
If we wait for a while, the Pod will eventually get created and it will ready as the command below shows:

```
PS C:\> .\kubectl.exe get pods
NAME                                READY    STATUS    RESTARTS   AGE
hello-nginx-24710...               1/1     Running   0          3m
```

If we see the Dashboard again, the Deployment is ready now:



If we visit the Replica Sets now, we can see it:



Click on the Replica Set name and it will show the Pod details as given below:

The screenshot shows the Kubernetes Dashboard interface. The browser address bar displays the URL: `192.168.99.100:30000/#/replicaset/default/hello-nginx-2471083592?namespace=default`. The dashboard header includes the 'kubernetes' logo, the breadcrumb 'Replica Sets > hello-nginx-2471083592', and action buttons for 'EDIT', 'DELETE', and '+ CREATE'.

The left sidebar contains a navigation menu with categories: 'Workloads' (Deployments, **Replica Sets**, Replication Controllers, Daemon Sets, Pod Sets, Jobs, Pods), 'Services and discovery' (Services, Ingress), 'Storage' (Persistent Volume Claims), and 'Config' (Secrets, Config Maps).

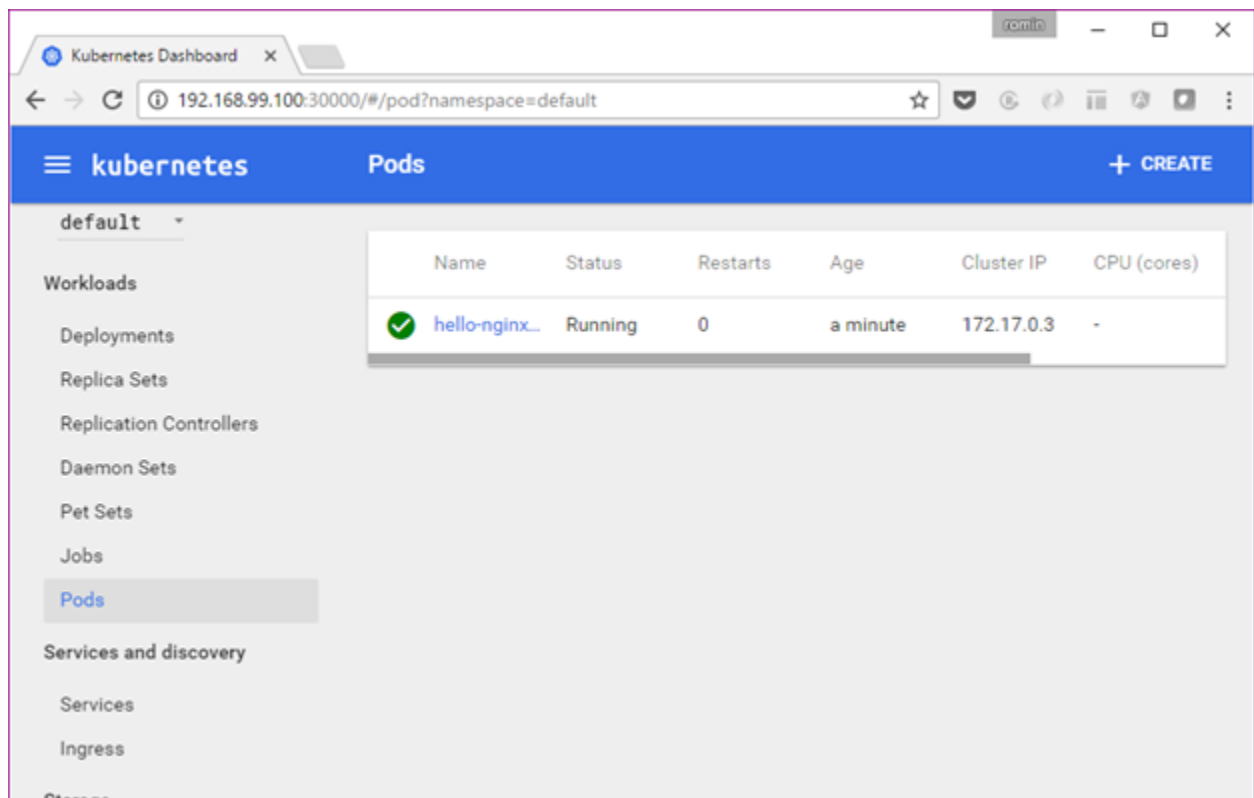
The main content area displays the following information:

- Details:**
 - Name: hello-nginx-2471083592
 - Namespace: default
 - Labels: pod-template-hash: 2471083592, run: hello-nginx
 - Images: nginx
- Status:** Pods: 1 running
- Pods:** A table showing one pod in a 'Running' state.
- Events:** A table showing two events related to the pod's creation and assignment.

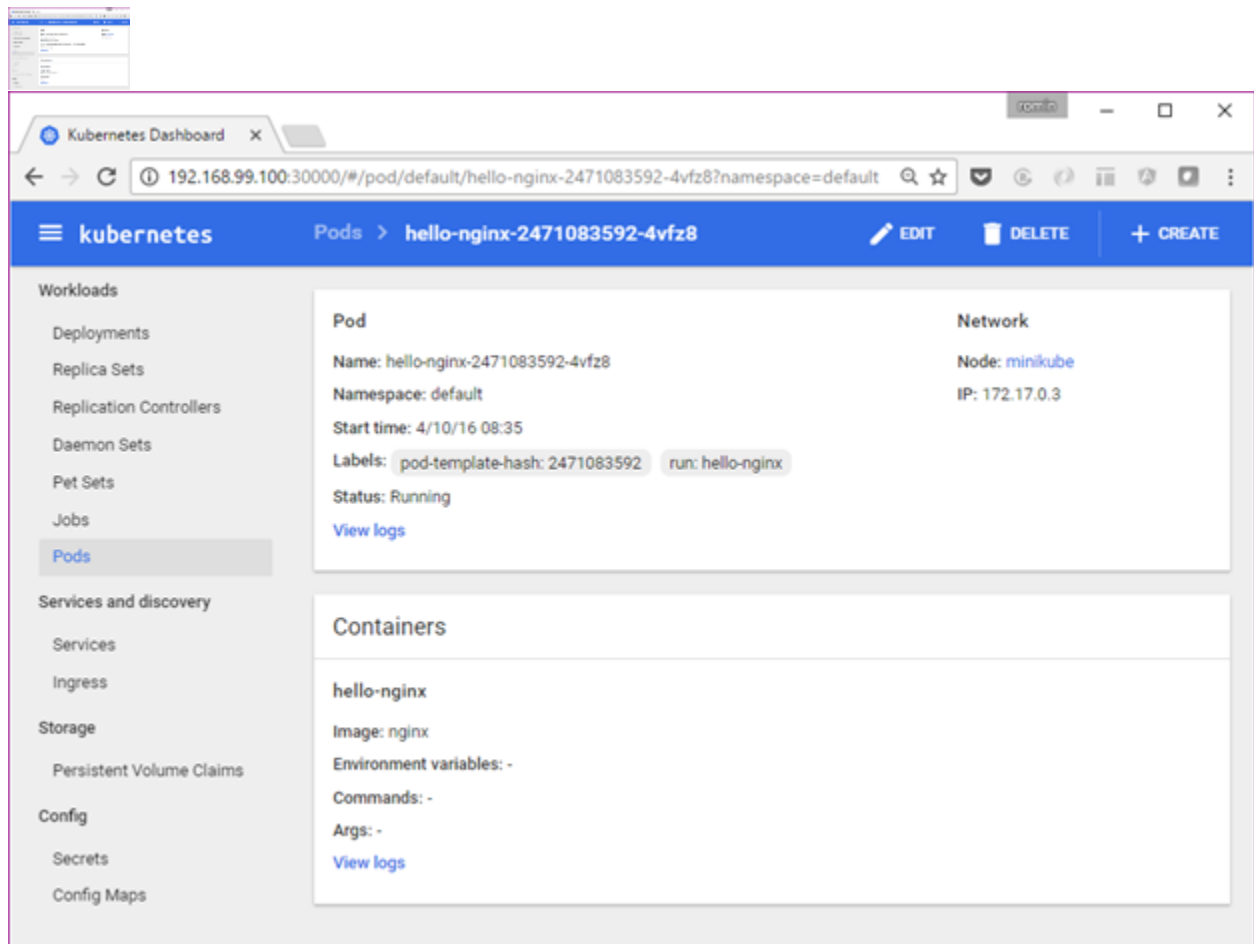
Name	Status	Restarts	Age	Cluster IP	CPU (cores)	Memory (bytes)
hello-nginx...	Running	0	a minute	172.17.0.3	-	-

Message	Source	Sub-object	Count	First seen	Last seen
Created pod: hello-nginx-2471083592-4vfz8	replicaset-controller	-	1	4/10/16 08:35 UTC	4/10/16 08:35 UTC
Successfully assigned pod hello-nginx...				4/10/16 08:35	4/10/16 08:35

Alternately, you can also get to the Pods via the **Pods** link in the Workloads as shown below:



Click on the Pod and you can get various details on it as given below:



You can see that it has been given some default labels. You can see its IP address. It is part of the node named minikube. And most importantly, there is a link for **View Logs** too.

The 1.4 dashboard greatly simplifies using Kubernetes and explaining it to everyone. It helps to see what is going on in the Dashboard and then the various commands in kubectl will start making sense more.

We could have got the Node and Pod details via a variety of **kubectl describe node/pod** commands and we can still do that. An example of that is shown below:

```
PS C:\> .\kubectl.exe describe pod hello-nginx-2471083592-4vfz8
```

```
Name:                hello-nginx-2471083592-4vfz8
Namespace:           default
Node:                minikube/192.168.99.100
Start Time:          Tue, 04 Oct 2016 14:05:15 +0530
Labels:              pod-template-hash=2471083592
                    run=hello-nginx
Status:              Running
IP:                  172.17.0.3
Controllers:         ReplicaSet/hello-nginx-2471083592
Containers:
  hello-nginx:
    Container ID:      docker://98a9e303f0dbf21db80a20aea744725c9bd64f6b2ce2764379151e3ae422fc18
    Image:              nginx
    Image ID:           docker://sha256:ba6bed934df2e644fdd34e9d324c80f3c615544ee9a93e4ce3cfddfcf84bdbbc2
    Port:               80/TCP
    State:              Running
      Started:          Tue, 04 Oct 2016 14:06:02 +0530
    Ready:              True
    Restart Count:      0
    Volume Mounts:
```

```
/var/run/secrets/kubernetes.io/serviceaccount from
default-token-rie7t (ro)
```

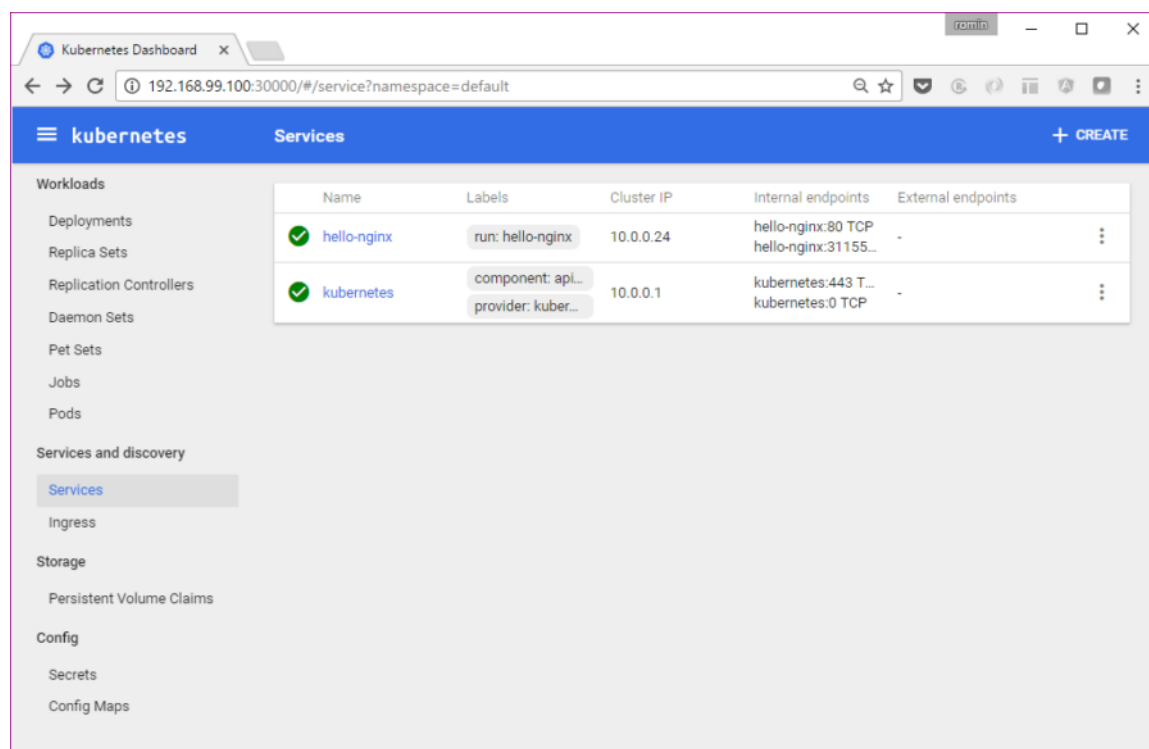
```
    Environment Variables:      <none>
..... /// REST OF THE OUTPUT ///
```

Expose a Service

It is time now to expose our basic Nginx deployment as a service. We can use the command shown below:

```
PS C:\> .\kubectl.exe expose deployment hello-
nginx --type=NodePort
service "hello-nginx" exposed
```

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:

```
PS C:\> .\kubectl.exe get services
NAME                CLUSTER-IP    EXTERNAL-IP    PORT(S)
AGE
```

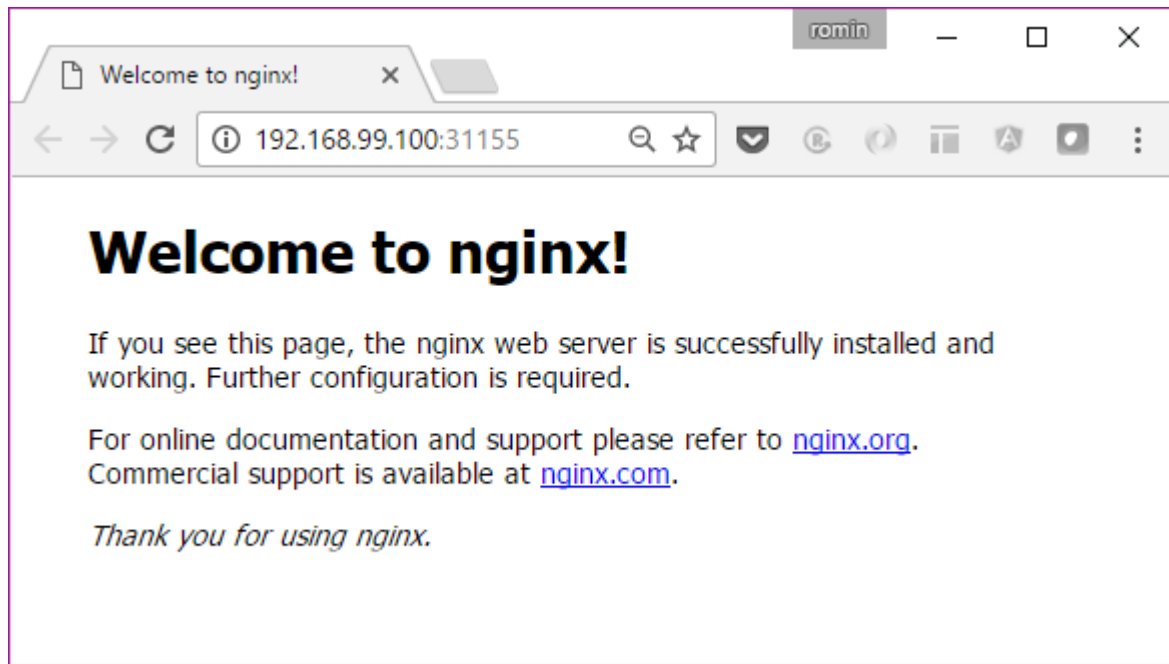
```
hello-nginx    10.0.0.24    <nodes>        80/TCP
3m
kubernetes     10.0.0.1    <none>         443/TCP
1h
PS C:\> .\kubectl.exe describe service hello-nginx
Name:                hello-nginx
Namespace:           default
Labels:              run=hello-nginx
Selector:            run=hello-nginx
Type:               NodePort
IP:                 10.0.0.24
Port:               <unset> 80/TCP
NodePort:           <unset> 31155/TCP
Endpoints:          172.17.0.3:80
Session Affinity:    None
No events.
```

We can now use the minikube service to understand the URL for the service as shown below:

```
PS C:\> .\minikube.exe service --url=true hello-nginx
http://192.168.99.100:31155
```

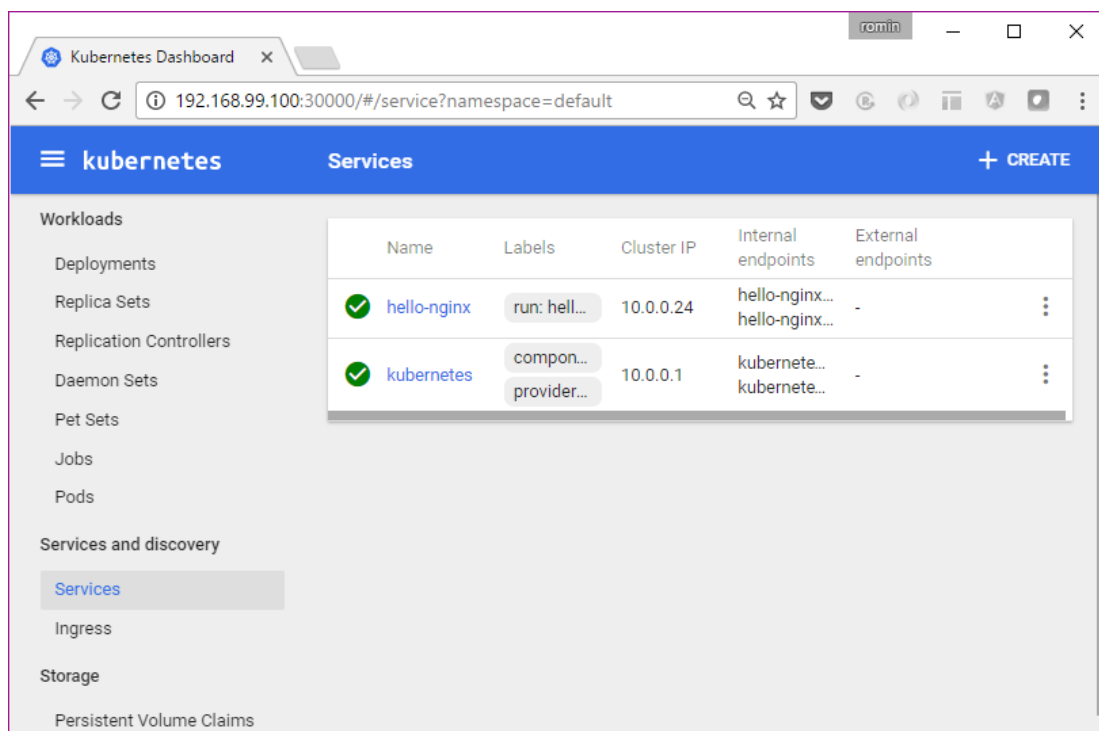
Alternately, if we do not use the url flag, then it can directly launch the browser and hit the service endpoint:

```
PS C:\> .\minikube.exe service hello-nginx
Opening kubernetes service default/hello-nginx in
default browser...
```



View Logs

Assuming that you have accessed the service once in the browser as shown above, let us look at an interesting thing now. Go to the Service link in the Dashboard.



Click on the **hello-nginx** service. This will also show the list of Pods (single) as shown below. Click on the icon for Logs as highlighted below:

The screenshot shows the Kubernetes Dashboard interface. The left sidebar contains navigation links for Workloads, Services and discovery, and Storage. The main content area is titled 'hello-nginx' and includes a 'Resource Details' section with the following information:

- Name:** hello-nginx
- Namespace:** default
- Label selector:** run: hello-nginx
- Labels:** run: hello-nginx
- Type:** NodePort

The 'Connection' section shows the Cluster IP (10.0.0.24) and Internal endpoints (hello-nginx:80 TCP, hello-nginx:31155 TCP). Below this, the 'Pods' section displays a table with the following data:

Name	Status	Restarts	Age	Cluster IP	CPU (cores)	Memory (bytes)
hello-nginx...	Running	0	28 minutes	172.17.0.3	-	-

This will show the logs for that particular Pod and with HTTP Request calls that was just made.

The screenshot shows the 'Logs' section of the Kubernetes Dashboard. The left sidebar contains navigation links for Admin, Namespace, and Workloads. The main content area is titled 'Logs' and displays the following information:

Logs from hello-nginx in hello-nginx-2471083592-4vfz8

```

2016-10-04T09:00:33.300266903Z 172.17.0.1 - - [04/Oct/2016:09:00:33 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36" "-"
2016-10-04T09:00:33.548308476Z 2016/10/04 09:00:33 [error] 5#5: *1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 172.17.0.1, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "192.168.99.100:31155", referer: "http://192.168.99.100:31155/"
2016-10-04T09:00:33.548358270Z 172.17.0.1 - - [04/Oct/2016:09:00:33 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "http://192.168.99.100:31155/" "Mozilla/5.0 (Windows NT 10.0; Win64;
  
```

Logs from 10/4/16 2:30 PM to 10/4/16 2:30 PM

You could do the same by using the logs <podname> command for the kubectl CLI:

```

PS C:\> .\kubectl logs hello-nginx-2471083592-4vfz8
172.17.0.1 - - [04/Oct/2016:09:00:33 +0000] "GET /
  
```

```

HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT
10.0; Win64; x64) Appl
eWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 Safari/537.36" "-"
2016/10/04 09:00:33 [error] 5#5: *1 open()
"/usr/share/nginx/html/favicon.ico" failed (2: No
such file or directory), cl
ient: 172.17.0.1, server: localhost, request: "GET
/favicon.ico HTTP/1.1", host:
"192.168.99.100:31155", referer: "http
://192.168.99.100:31155/"
172.17.0.1 - - [04/Oct/2016:09:00:33 +0000] "GET
/favicon.ico HTTP/1.1" 404 571
"http://192.168.99.100:31155/" "Mozilla/
5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 Safari/537.36" "-"
PS C:\>

```

Scaling the Service

OK, I am not yet done!

When we created the deployment, we did not mention about the number of instances for our service. So we just had one Pod that was provisioned on the single node.

Let us go and see how we can scale this via the scale command. We want to scale it to 3 Pods.

```

PS C:\> .\kubectl scale --replicas=3
deployment/hello-nginx
deployment "hello-nginx" scaled

```

We can see the status of the deployment in a while:

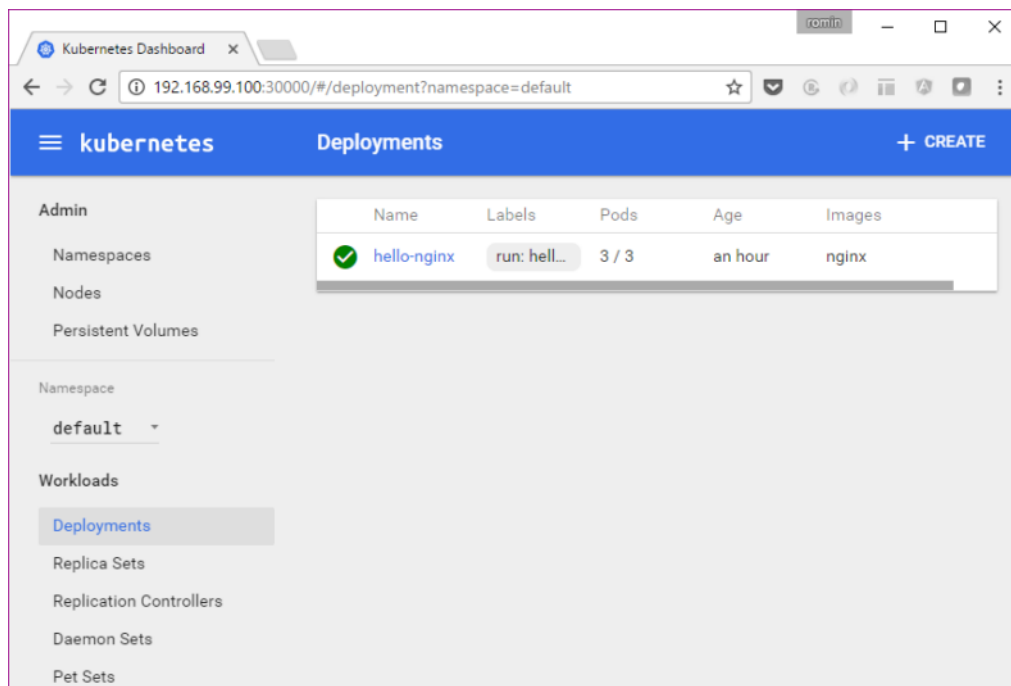
```

PS C:\> .\kubectl.exe get deployment
NAME                DESIRED    CURRENT    UP-TO-DATE
AVAILABLE    AGE

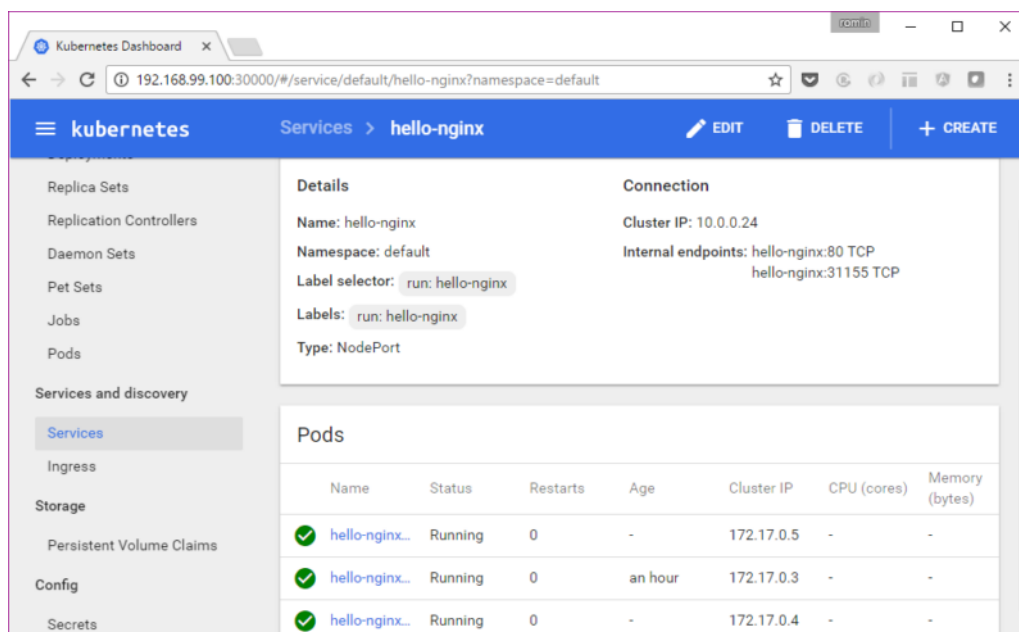
```


hello-nginx 3 3 3 3
1h

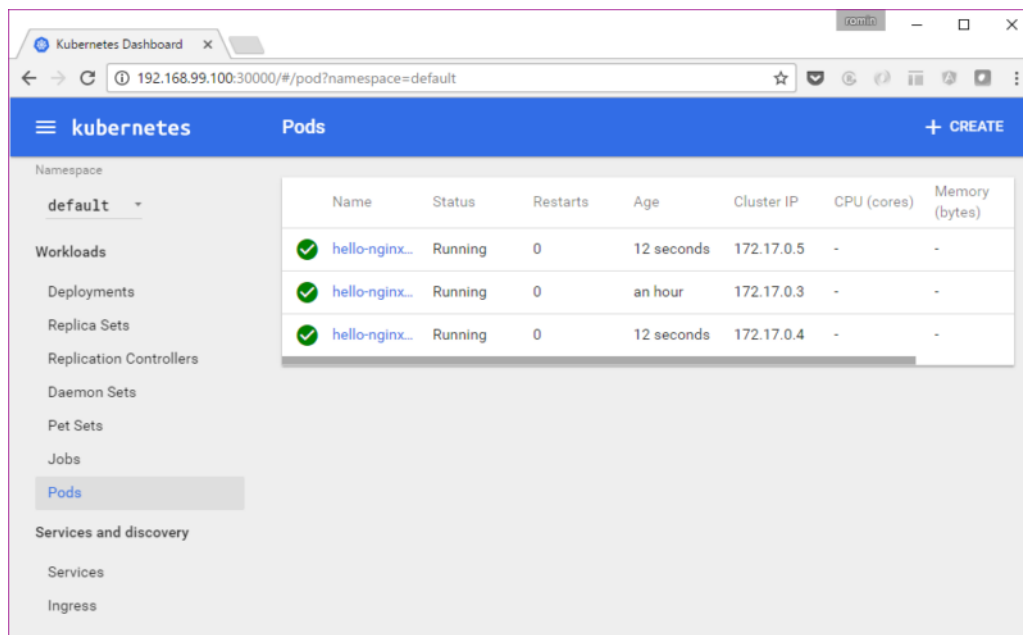
Now, if we visit the Dashboard for our Deployment:



We have the 3/3 Pods available. Similarly, we can see our Service or Pods.



or the Pod list:



Stopping and Deleting the Cluster

This is straightforward. You can use the **stop** and **delete** commands for minikube utility.

Limitations

Minikube is a work in progress at this moment and it does not support all the features of Kubernetes. Please [refer to the minikube documentation](#), where it clearly states what is currently supported.

A Note on Google Container Engine

Now that you have been able to see the basic building blocks of Kubernetes run on your machine, the next step for you might be to try it out on an actual cluster of machines (VMs) and see it all work.

One of the best managed solutions out there for k8s is Google Container Engine. A single command and you are up and running with a fully managed Kubernetes cluster with a few VMs in a matter of minutes. You can then set the configuration for the kubectl command line utility to work with that cluster and perform the operations. Give it a try.



CONTAINER ENGINE

One-click Kubernetes clusters, managed by Google

<https://cloud.google.com/container-engine/>

Google hosts some excellent codelabs on Google Container Engine to get you started. I have learnt a lot through them. Check them out:

- [Running a Container in Kubernetes with Container Engine](#)
- [Orchestrating the Cloud with Kubernetes](#)
- [Hello Node Kubernetes Codelab](#)

Troubleshooting Issues on Windows 10

My experience to get the experimental build of minikube working on Windows was not exactly a smooth one, but that is to be expected from anything that calls itself experimental.

I faced several issues and hope that it will save some time for you. I do not have the time to investigate deeper into why some of the stuff worked for me since my focus is to get it up and running on Windows. So if you have some specific comments around that, that will be great and I can add to this blog post.

In no order of preference, here you go:

Use Powershell

I used Powershell and not command line. Ensure that Powershell is launched in Administrative mode. This means Ctrl + Shift + Enter.

Put minikube.exe file in C:\ drive

I saw some issues that mentioned to do that. I did not experiment too much and went with C:\ drive.

Clear up .minikube directory

If there were some issues starting up minikube first time and then you try to start it again, you might see errors that say stuff like “Starting Machine” or “Machine exists” and then a bunch of errors before it gives up. I suggest that you clear up the .minikube directory that is present in %HOMEPATH%\minikube directory. In my case, it is C:\Users\irani_r\minikube. You will see a bunch of folders there. Just delete them all and start all over again.

To see detailed error logging, give the following flags while starting up the cluster:

```
--show-libmachine-logs --alsologtostderr
```

Example of the error trace for me was as follows:

```
PS C:\> .\minikube start --show-libmachine-logs --
alsologtostderr
W1003 15:59:52.796394    12080 root.go:127] Error
reading config file at
C:\Users\irani_r\minikube\config\config.json: o
pen C:\Users\irani_r\minikube\config\config.json:
The system cannot find the file specified.
I1003 15:59:52.800397    12080 notify.go:103]
Checking for updates...
Starting local Kubernetes cluster...
I1003 15:59:53.164759    12080 cluster.go:75]
Machine exists!
I1003 15:59:54.133728    12080 cluster.go:82]
Machine state: Error
E1003 15:59:54.133728    12080 start.go:85] Error
starting host: Error getting state for host:
machine does not exist. Re
trying.
I1003 15:59:54.243132    12080 cluster.go:75]
Machine exists!
I1003 15:59:54.555738    12080 cluster.go:82]
Machine state: Error
E1003 15:59:54.555738    12080 start.go:85] Error
starting host: Error getting state for host:
```

```
machine does not exist. Re
trying.
I1003 15:59:54.555738    12080 cluster.go:75]
Machine exists!
I1003 15:59:54.790128    12080 cluster.go:82]
Machine state: Error
E1003 15:59:54.790128    12080 start.go:85] Error
starting host: Error getting state for host:
machine does not exist. Re
trying.
E1003 15:59:54.790128    12080 start.go:91] Error
starting host: Error getting state for host:
machine does not exist
Error getting state for host: machine does not
exist
Error getting state for host: machine does not
exist
```

Disable Hyper-V

As earlier mentioned, VirtualBox and Hyper-V are not the happiest of co-workers. Definitely disable one of them on your machine. As per the documentation of minikube, both virtualbox and hyperv drivers are supported on Windows. I will do a test of Hyper-V someday but I went with disabling Hyper-V and used VirtualBox only. The steps to disable Hyper-V correctly were shown earlier in this blog post.

Conclusion

Hope this blog post gets you started with Kubernetes on your Windows development machine by using minikube.