Tutorial : Getting Started with Kubernetes with Docker on Mac

Romin Irani Follow
Jan 10, 2018 · 9 min read

> *If you are looking for running Kubernetes on your Windows laptop, go to this tutorial.*

This blog post is related to Getting Started with Kubernetes on your Windows laptop with Minikube but this time with a Mac machine. The other big difference here is that this is not with Minikube, which you can still install on a Mac. It is with a Edge version of Docker on Mac.

> *This tutorial works on the Edge version of Docker on Mac and could undergo changes as it approaches a stable release. I will keep the article updated.*

We shall cover the following in this post:

- Installing Docker on Mac Edge version
- Go through the basic Kubernetes commands to validate our environment.

This tutorial assumes that you know about Docker and Kubernetes in general. To quote from my previous article, 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.

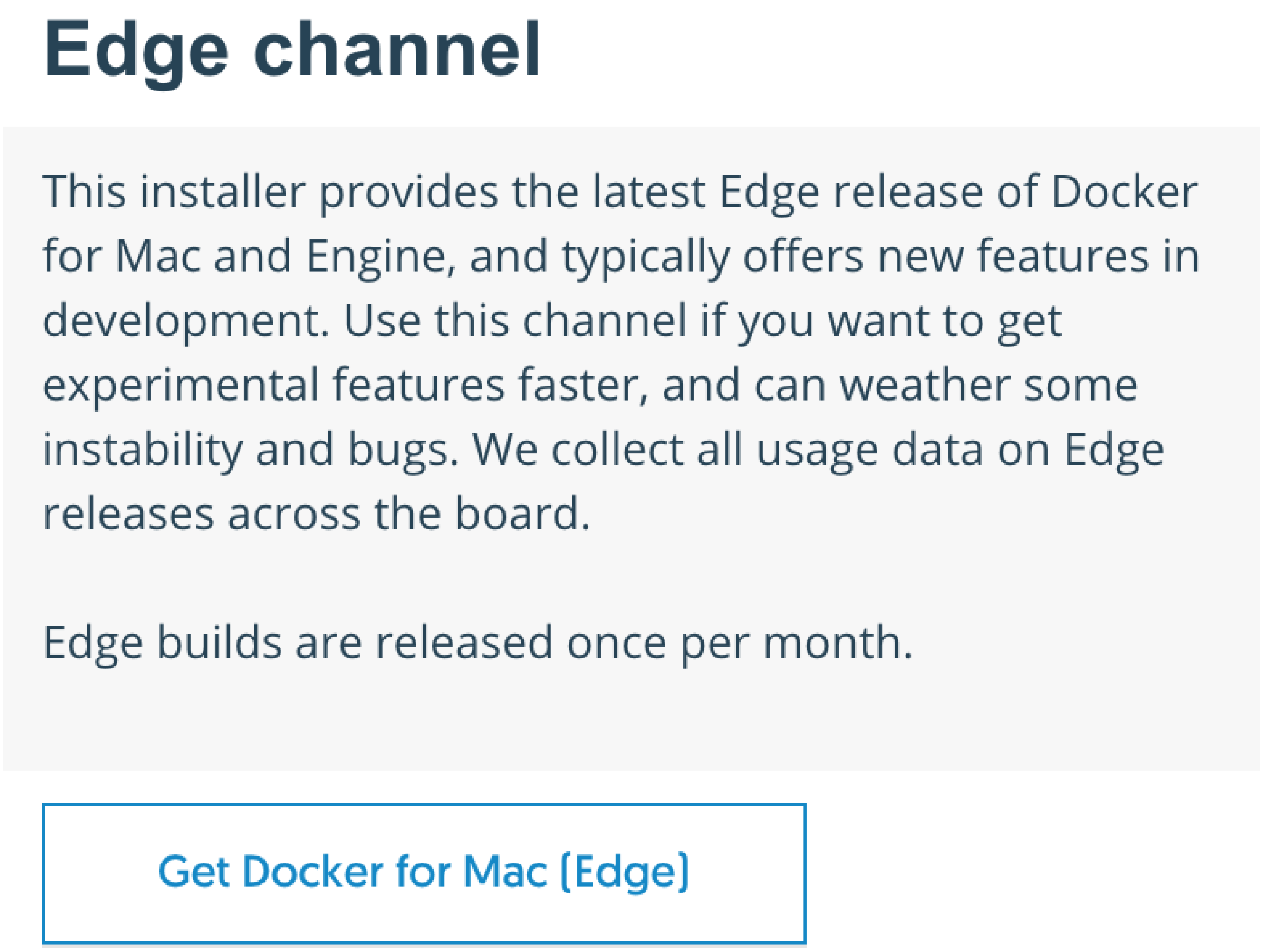
I have written a couple of other articles that go through a high level overview of Kubernetes:

- 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.

Docker for Mac installation

As per the official documentation, **Kubernetes is only available in Docker for Mac 17.12 CE Edge.** Go to the official download page and click on the Edge channel and not the Stable version.

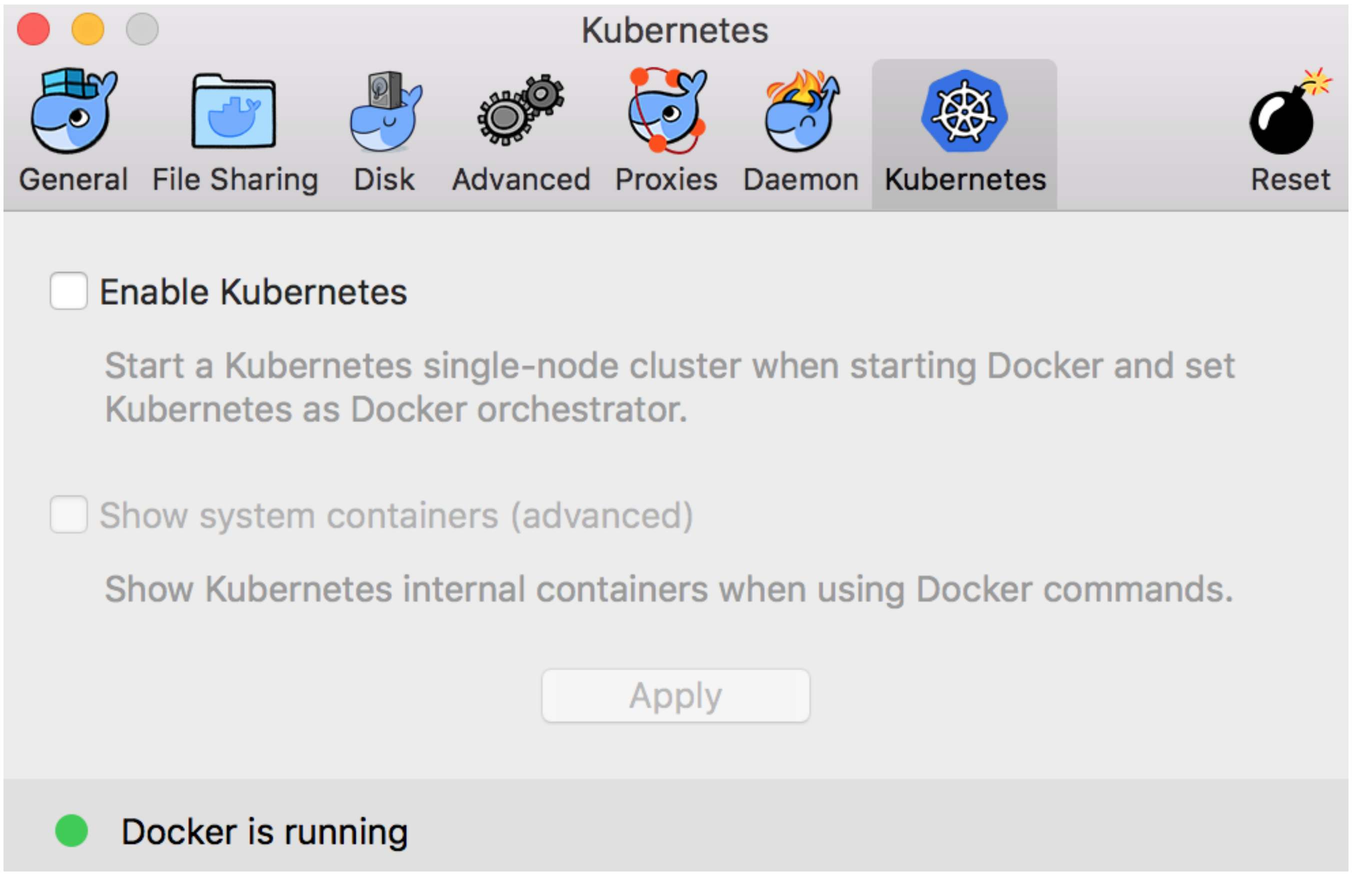


Edge channel

This installer provides the latest Edge release of Docker for Mac and Engine, and typically offers new features in development. Use this channel if you want to get experimental features faster, and can weather some instability and bugs. We collect all usage data on Edge releases across the board.

Edge builds are released once per month.

Download the .dmg file and go ahead with the standard installation steps. You can then launch Docker Edge. Click on the Docker icon and go to **Preferences** window as shown below. Click on the **Kubernetes** icon.



☐ Enable Kubernetes

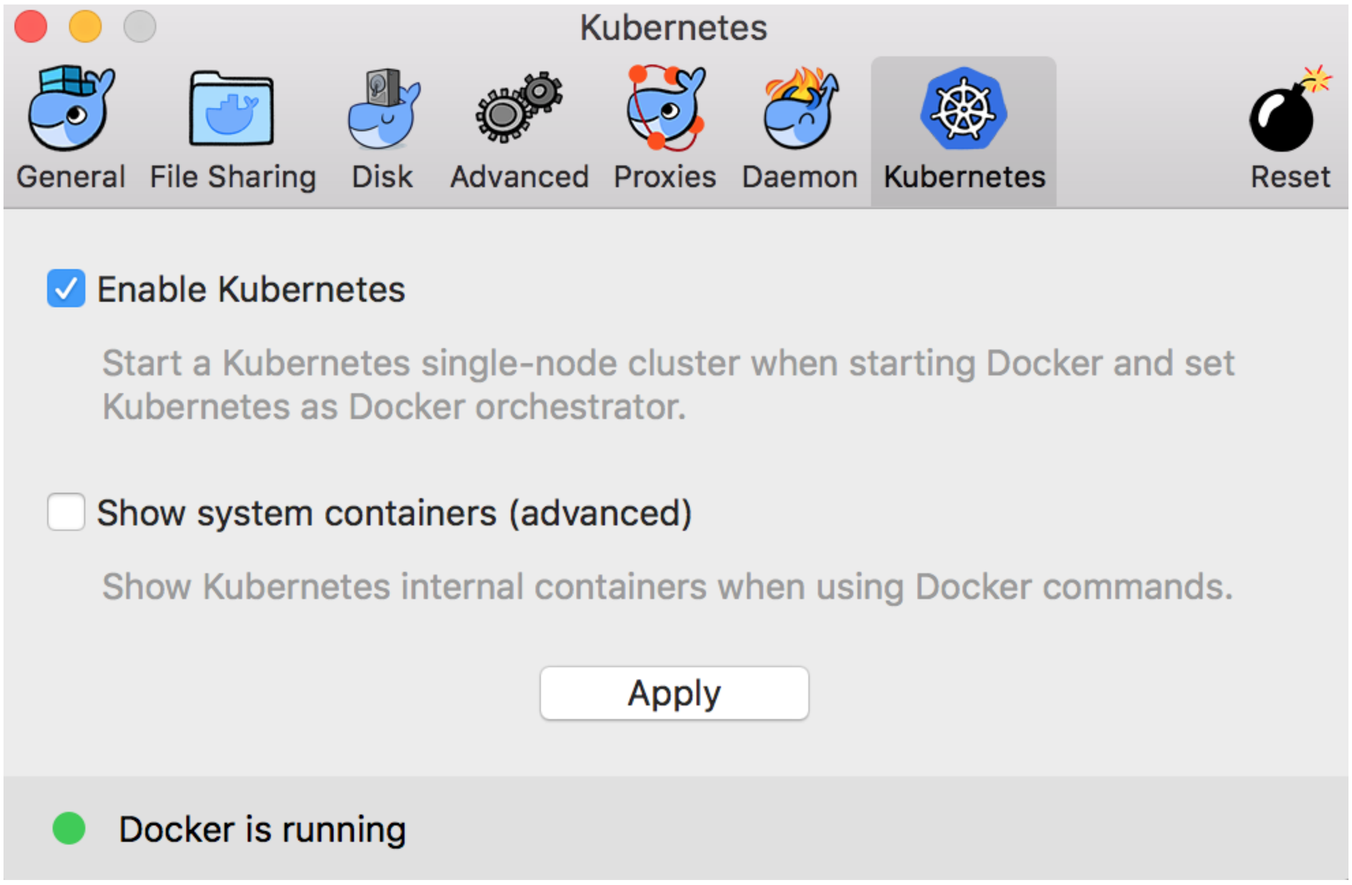
Start a Kubernetes single-node cluster when starting Docker and set Kubernetes as Docker orchestrator.

☐ Show system containers (advanced)

Show Kubernetes internal containers when using Docker commands.

● Docker is running

You will notice that Kubernetes is not enabled. Simply check on the **Enable Kubernetes** option and then hit the **Apply** button as shown below:



✓ Enable Kubernetes

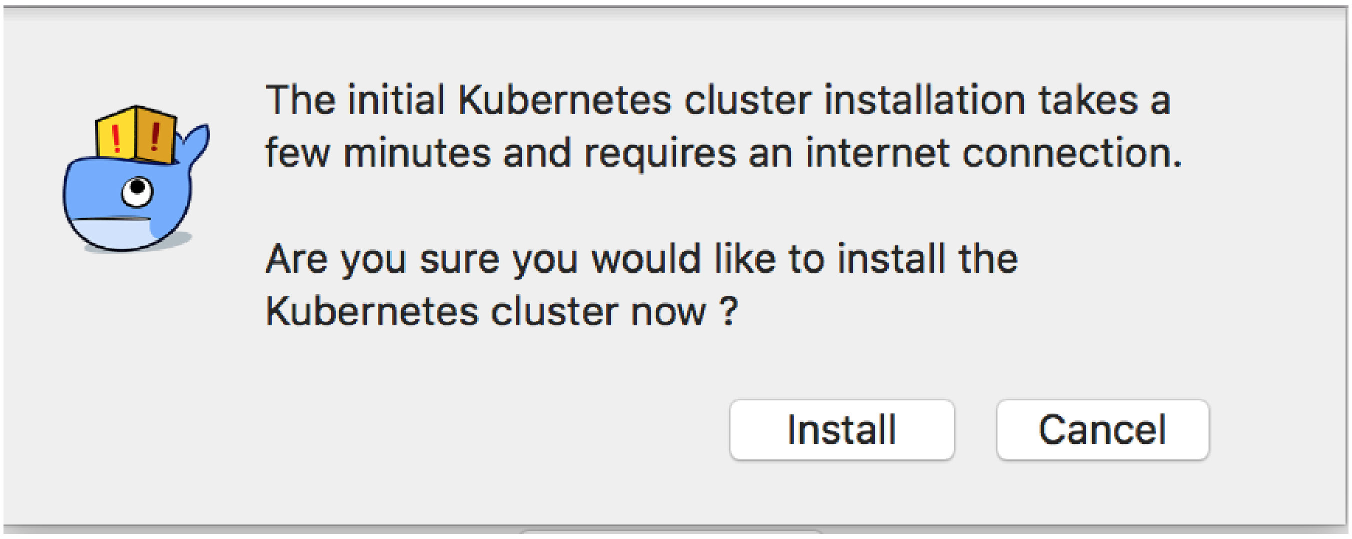
Start a Kubernetes single-node cluster when starting Docker and set Kubernetes as Docker orchestrator.

☐ Show system containers (advanced)

Show Kubernetes internal containers when using Docker commands.

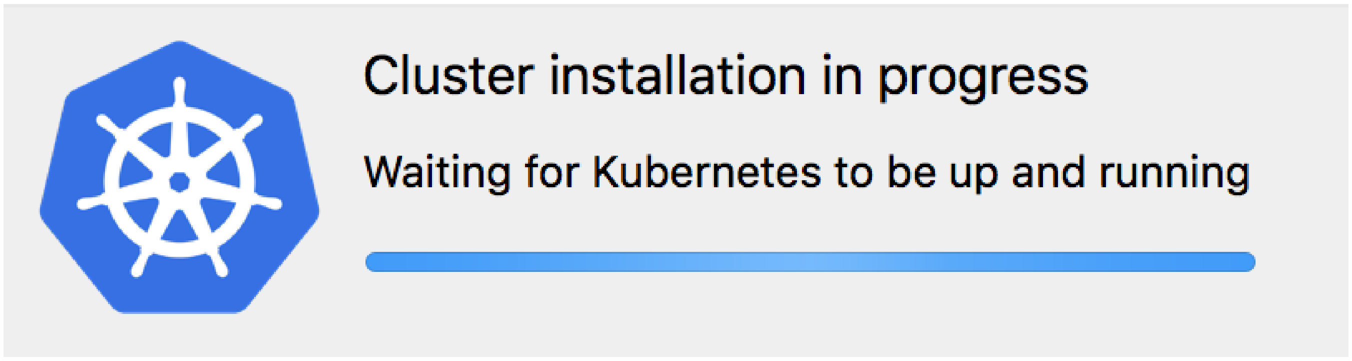
● Docker is running

This will display a message that the Kubernetes cluster needs to be installed. Make sure you are connected to the Internet and click on **Install**


The initial Kubernetes cluster installation takes a few minutes and requires an internet connection.

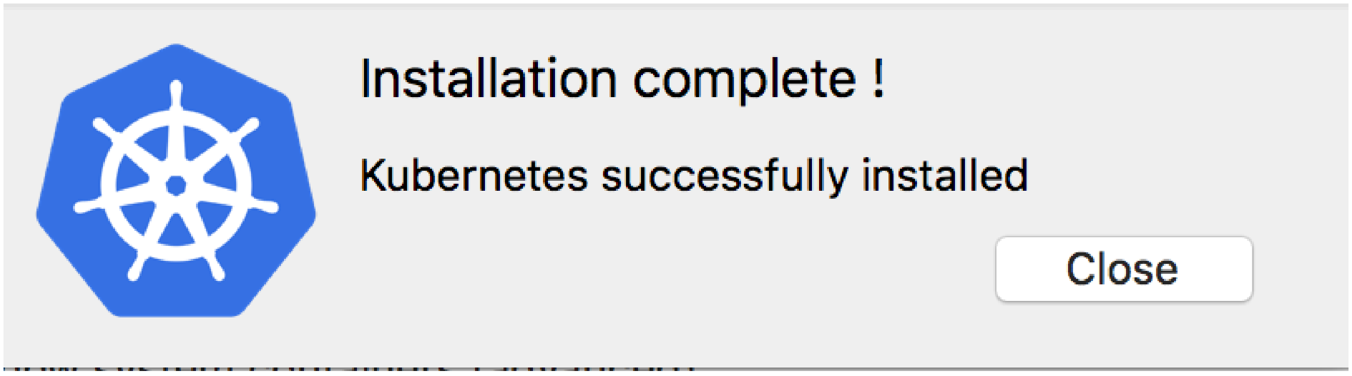
Are you sure you would like to install the Kubernetes cluster now ?

The installation starts. Please be patient since this could take a while depending on your network. *It would have been nice to see a small log window that shows a sequence of steps.*


Cluster installation in progress

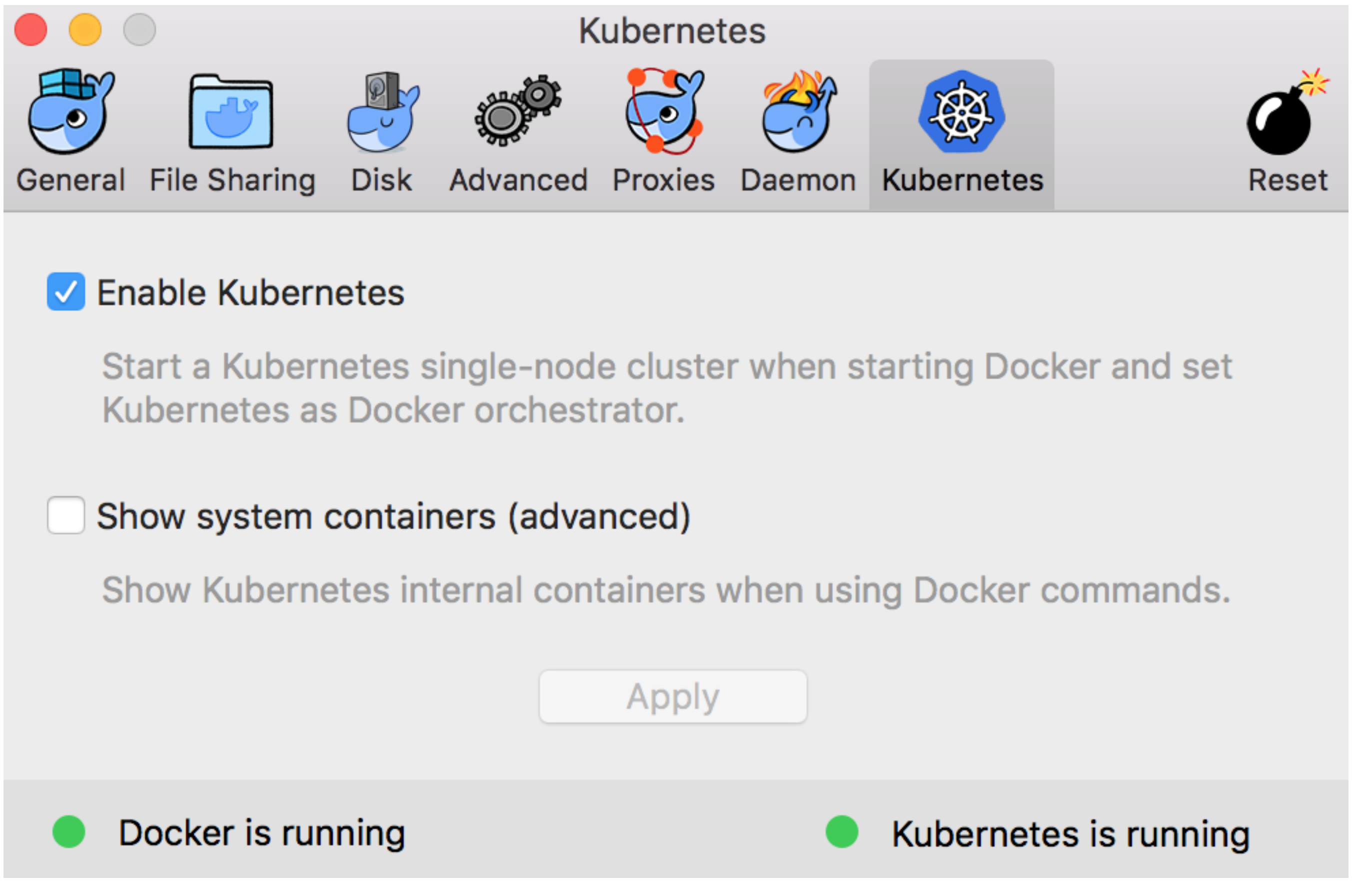
Waiting for Kubernetes to be up and running

Finally, you should see the following message:


Installation complete !

Kubernetes successfully installed

Click on Close. This will lead you back to the Preferences dialog and you should see the following screen:



☑ Enable Kubernetes

Start a Kubernetes single-node cluster when starting Docker and set Kubernetes as Docker orchestrator.

☐ Show system containers (advanced)

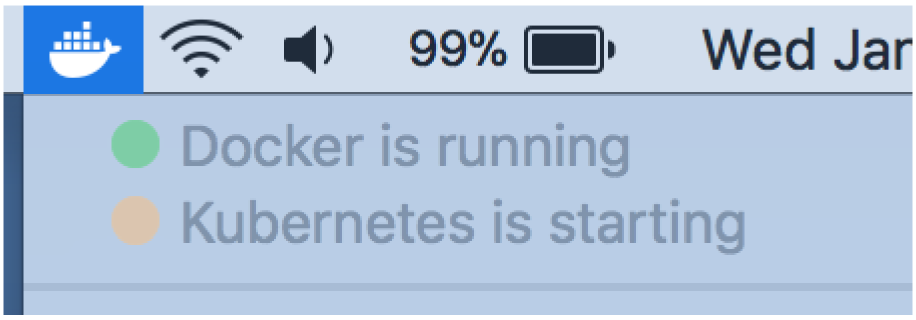
Show Kubernetes internal containers when using Docker commands.

● Docker is running ● Kubernetes is running

Note the two messages at the bottom of the window mentioning:

- Docker is running
- Kubernetes is running

In case you stop running and try to run Docker again, you will notice that both Docker and Kubernetes services are starting as shown below:

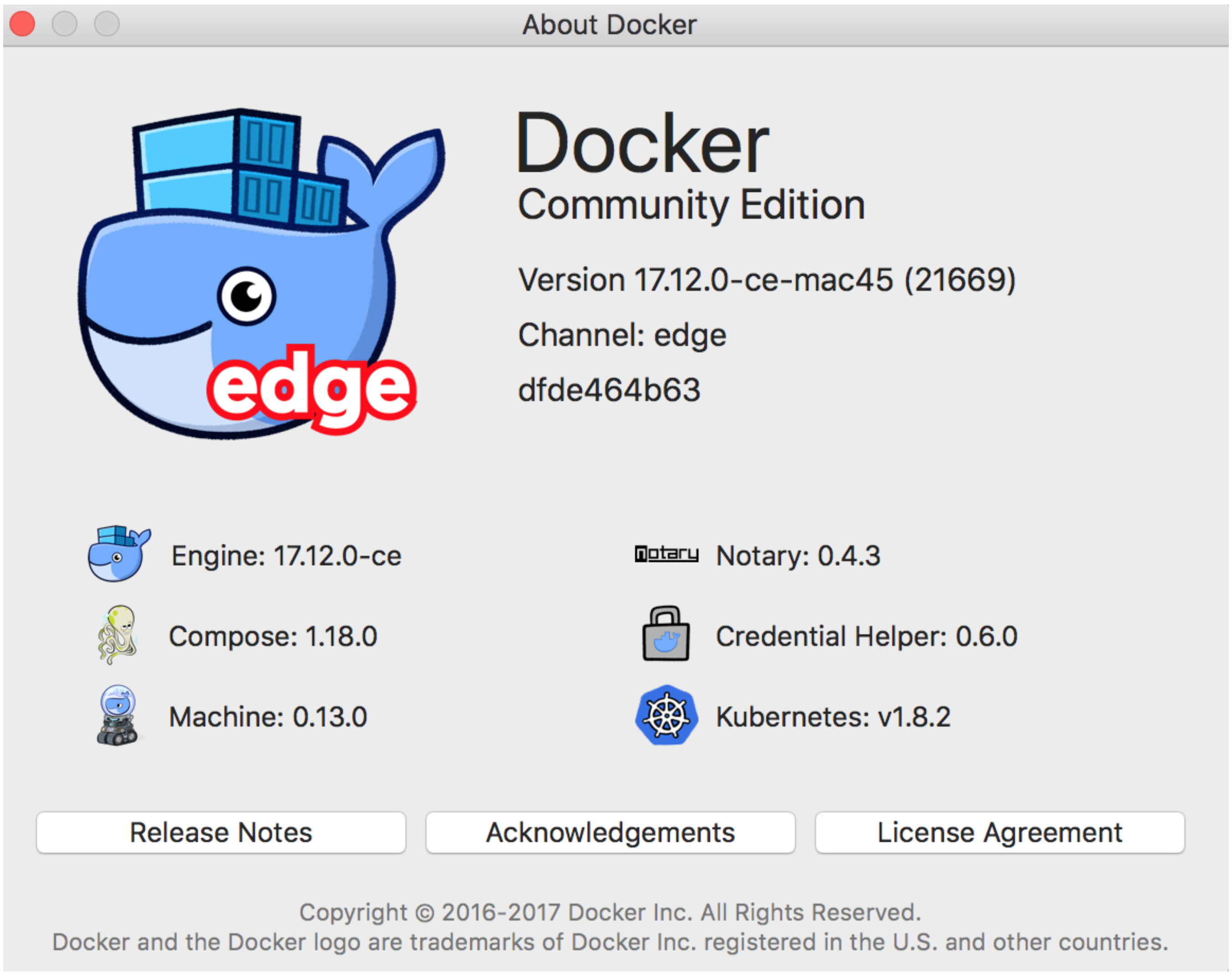


● Docker is running
● Kubernetes is starting

Congratulations! You now have the following:

- A standalone Kubernetes server and client, as well as Docker CLI integration.
- The Kubernetes server is a single-node cluster and is not configurable.

Just FYI … my About Docker shows the following:


About Docker

Docker
Community Edition

Version 17.12.0-ce-mac45 (21669)

Channel: edge

dfde464b63

Engine: 17.12.0-ce
Notary: 0.4.3
Compose: 1.18.0
Credential Helper: 0.6.0
Machine: 0.13.0
Kubernetes: v1.8.2

Release Notes | Acknowledgements | License Agreement

Copyright © 2016-2017 Docker Inc. All Rights Reserved.
Docker and the Docker logo are trademarks of Docker Inc. registered in the U.S. and other countries.

Check our installation

Let us try out a few things to ensure that we can make sense of what has got installed. Execute the following commands in a terminal:

```
$ kubectl version

Client Version: version.Info{Major:"1", Minor:"8",
GitVersion:"v1.8.4",
GitCommit:"9befc2b8928a9426501d3bf62f72849d5cbcd5a3",
GitTreeState:"clean", BuildDate:"2017–11–20T05:28:34Z",
GoVersion:"go1.8.3", Compiler:"gc", Platform:"darwin/amd64"}

Server Version: version.Info{Major:"1", Minor:"8",
GitVersion:"v1.8.2",
GitCommit:"bdaeafa71f6c7c04636251031f93464384d54963",
GitTreeState:"clean", BuildDate:"2017–10–24T19:38:10Z",
GoVersion:"go1.8.3", Compiler:"gc", Platform:"linux/amd64"}
```

You might have noticed that my server and client versions are different. I am using kubectl from my gCloud SDK tools and Docker for Mac, when it launched the

Kubernetes cluster has been able to set the cluster context for the kubectl utility for you. So if we fire the following command:

```
$ kubectl config current-context
docker-for-desktop
```

You can see that the cluster is set to **docker-for-desktop**.

> *Tip: In case you switch between different clusters, you can always get back using the following:*
>
> *$ kubectl config use-context docker-for-desktop*
> *Switched to context "docker-for-desktop"*

Let us get some information on the cluster.

```
$ kubectl cluster-info

Kubernetes master is running at https://localhost:6443
KubeDNS is running at https://localhost:6443/api/v1/namespaces/kube-
system/services/kube-dns/proxy
```

Let us check out the nodes in the cluster:

```
$ kubectl get nodes

NAME               STATUS  ROLES  AGE VERSION
docker-for-desktop Ready   master 7h  v1.8.2
```

Installating the Kubernetes Dashboard

The next step that we need to do here is to install the Kubernetes Dashboard. We can use the Kubernetes Dashboard YAML that is available and submit the same to the Kubernetes Master as follows:

```
$ kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/de
ploy/recommended/kubernetes-dashboard.yaml
```

```
secret “kubernetes-dashboard-certs” created
serviceaccount “kubernetes-dashboard” created
role “kubernetes-dashboard-minimal” created
rolebinding “kubernetes-dashboard-minimal” created
deployment “kubernetes-dashboard” created
service “kubernetes-dashboard” created
```

The Dashboard application will get deployed as a Pod in the **kube-system** namespace. We can get a list of all our Pods in that namespace via the following command:

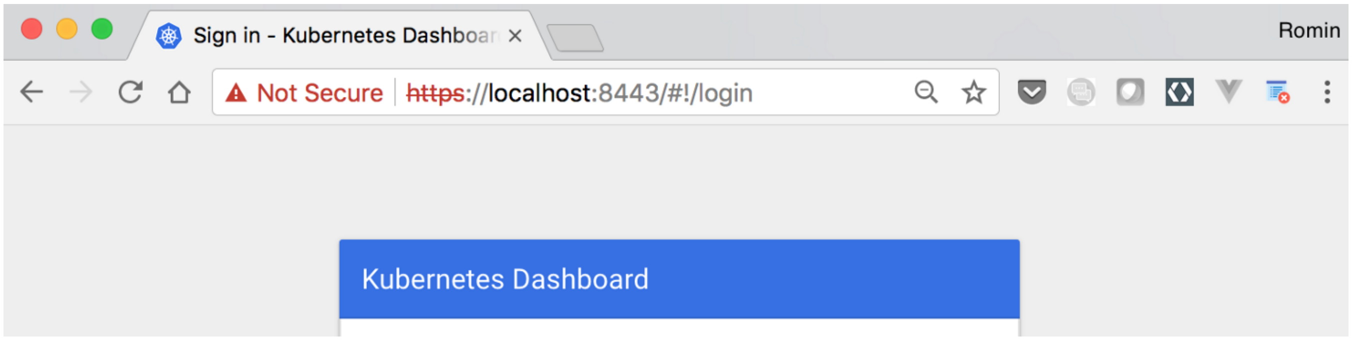
```
$ kubectl get pods — namespace=kube-system

NAME                                          READY STATUS RESTARTS AGE
etcd-docker-for-desktop                       1/1   Running 0       8h
kube-apiserver-docker-for-desktop             1/1   Running 0       7h
kube-controller-manager-docker-for-desktop    1/1   Running 0       8h
kube-dns-545bc4bfd4-l9tw9                     3/3   Running 0       8h
kube-proxy-w8pq7                              1/1   Running 0       8h
kube-scheduler-docker-for-desktop             1/1   Running 0       7h
kubernetes-dashboard-7798c48646-ctrtl         1/1   Running 0       3m
```

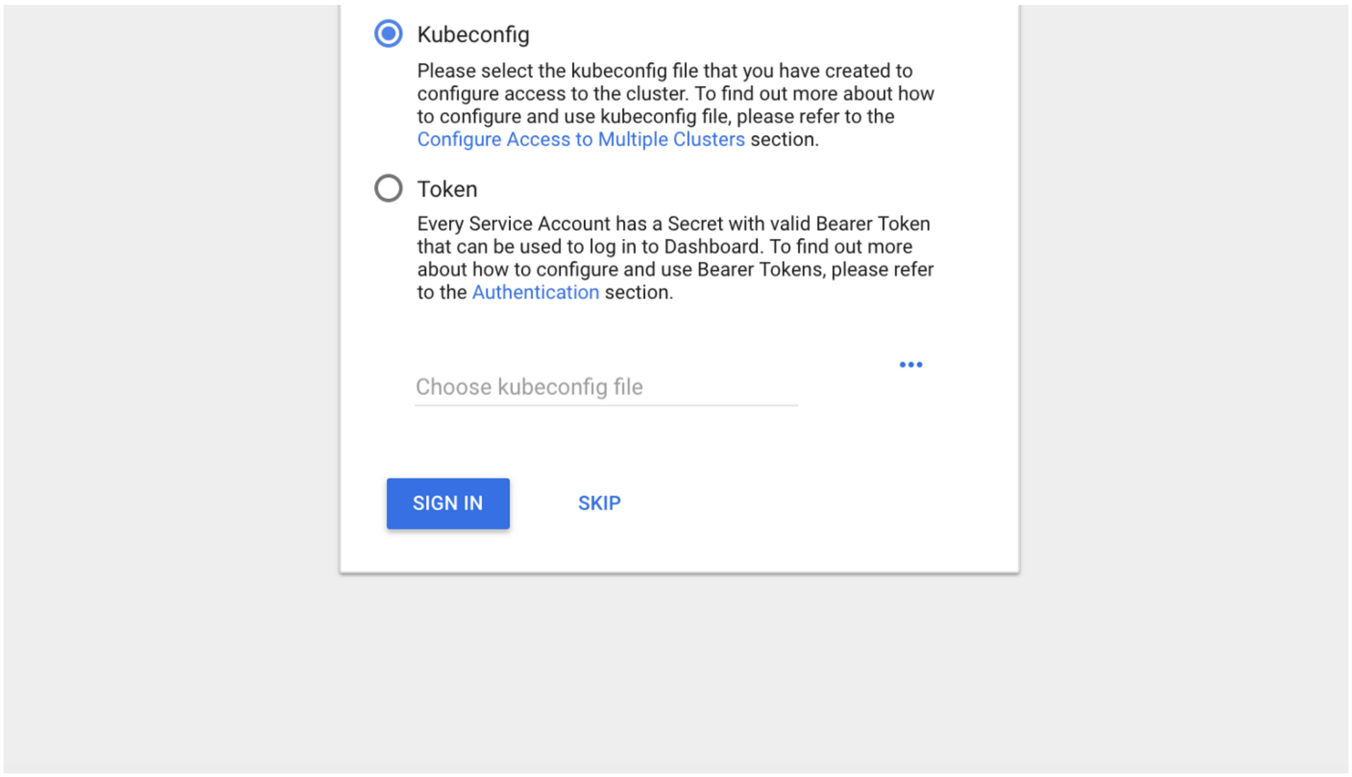
Ensure that the Pod shown in bold is in Running state. It could take some time to change from **ContainerCreating** to **Running**, so be patient.

Once it is in running state, you can setup a forwarding port to that specific Pod. So in our case, we can setup 8443 for the Pod Name as shown below:

```
$ kubectl port-forward kubernetes-dashboard-7798c48646-ctrtl
8443:8443 — namespace=kube-system

Forwarding from 127.0.0.1:8443 -> 8443
```

You can now launch a browser and go to https://localhost:8443. You might see some warnings but proceed. You will see the following screen:


Sign in - Kubernetes Dashboar
Romin
Not Secure
https://localhost:8443/#!/login
Kubernetes Dashboard



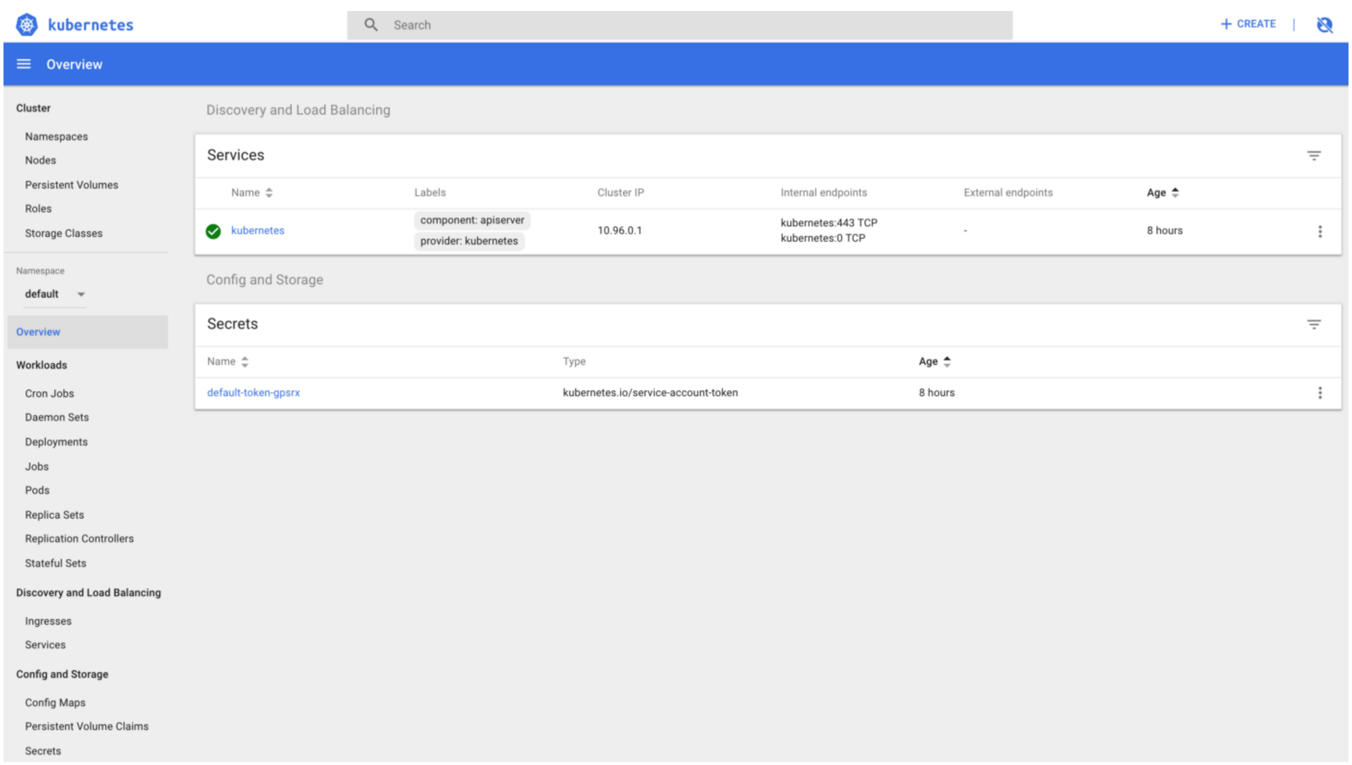
[✓ Kubeconfig]

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

Click on **SKIP** and you will be lead to the Dashboard as shown below:



Discovery and Load Balancing

Services

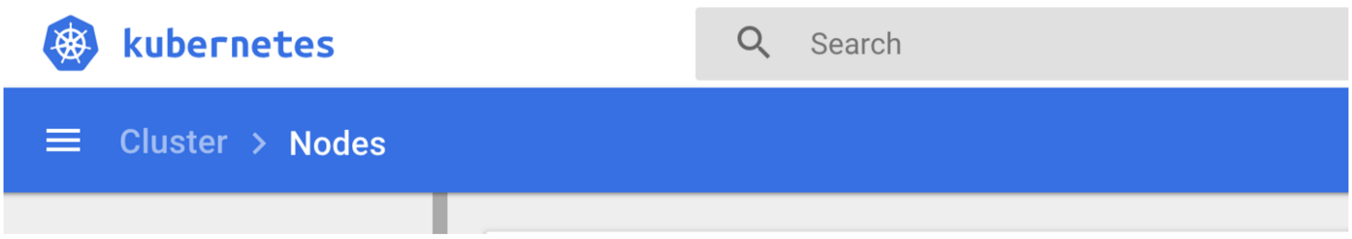
Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
✓ kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	8 hours

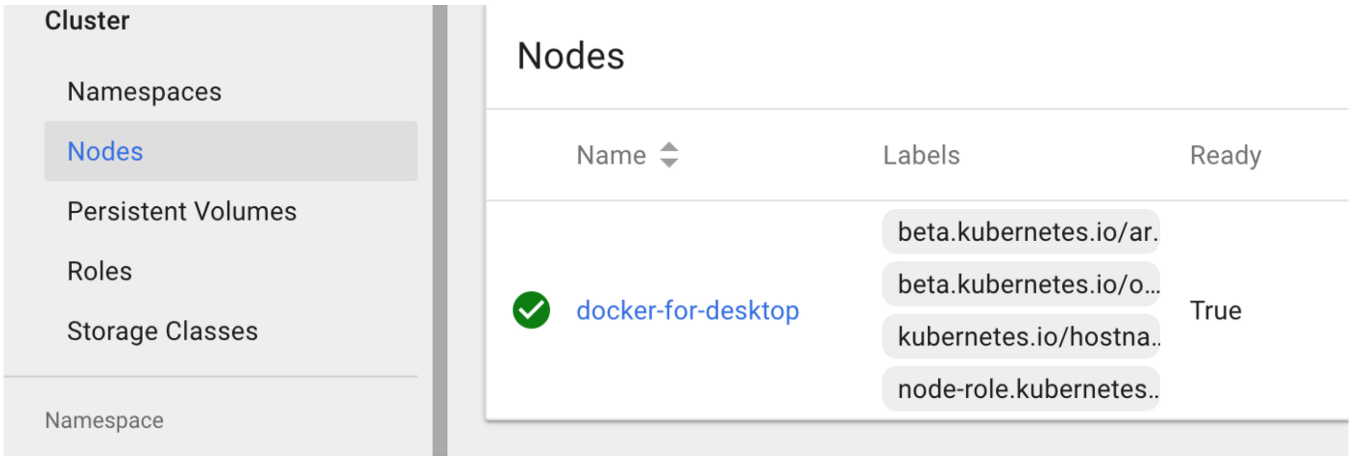
Config and Storage

Secrets

Name	Type	Age
default-token-gpsrx	kubernetes.io/service-account-token	8 hours

Click on Nodes and you will see the single node as given below:





Nodes

Name	Labels	Ready
docker-for-desktop	beta.kubernetes.io/ar. beta.kubernetes.io/o... kubernetes.io/hostna.. node-role.kubernetes..	True

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:

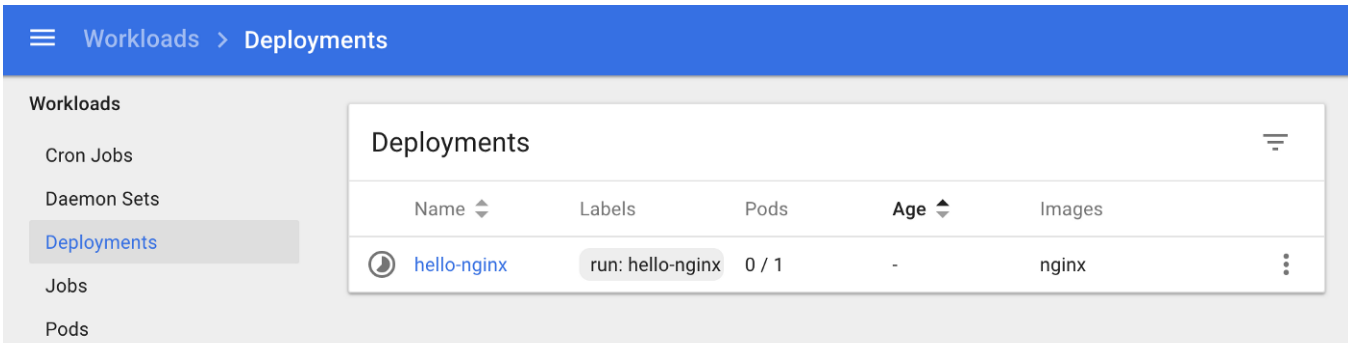
```
$ kubectl run hello-nginx --image=nginx --port=80

deployment “hello-nginx” created
```

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

```
$ kubectl get pods

NAME                          READY STATUS            RESTARTS AGE
hello-nginx-5d47cdc4b7-wxf9b 0/1   ContainerCreating 0        16s
```

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

Now, let us go back to the Dashboard and see the Deployments:



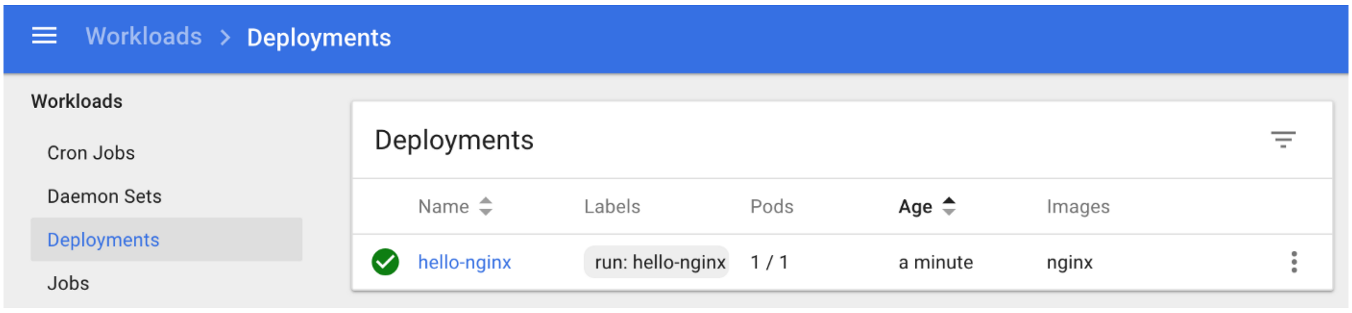
Deployments

Name	Labels	Pods	Age	Images
hello-nginx	run: hello-nginx	0 / 1	-	nginx

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:

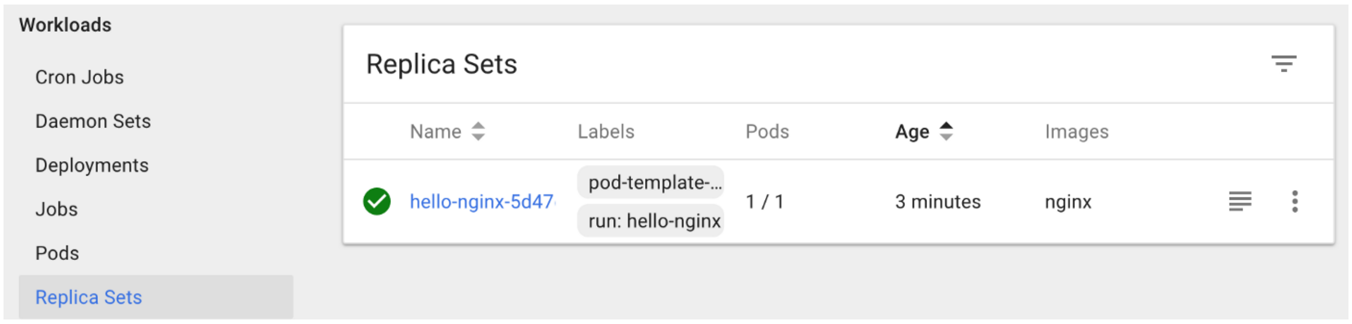
```
$ kubectl get pods

NAME                             READY STATUS  RESTARTS AGE
hello-nginx-5d47cdc4b7-wxf9b 1/1   Running 0        3m
```



Deployments

	Name	Labels	Pods	Age	Images
✓	hello-nginx	run: hello-nginx	1 / 1	a minute	nginx

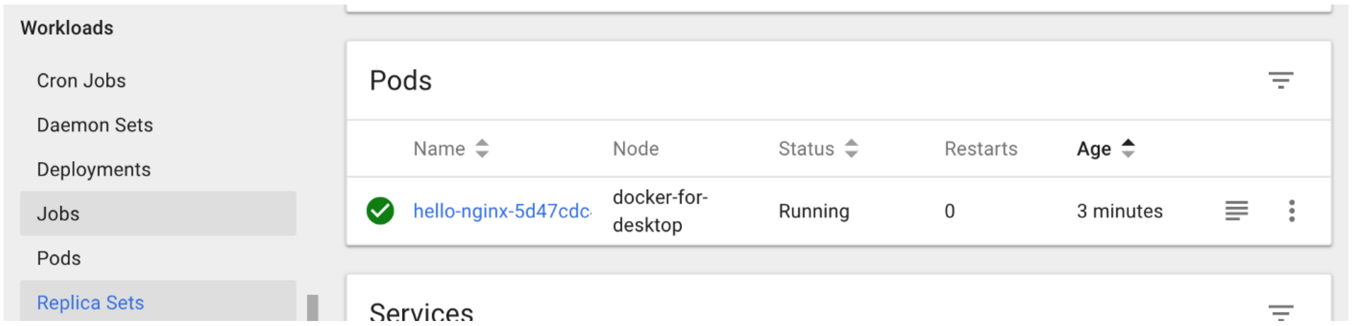
If we visit the Replica Sets now, we can see it:



Replica Sets

	Name	Labels	Pods	Age	Images
✓	hello-nginx-5d47	pod-template-… run: hello-nginx	1 / 1	3 minutes	nginx

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

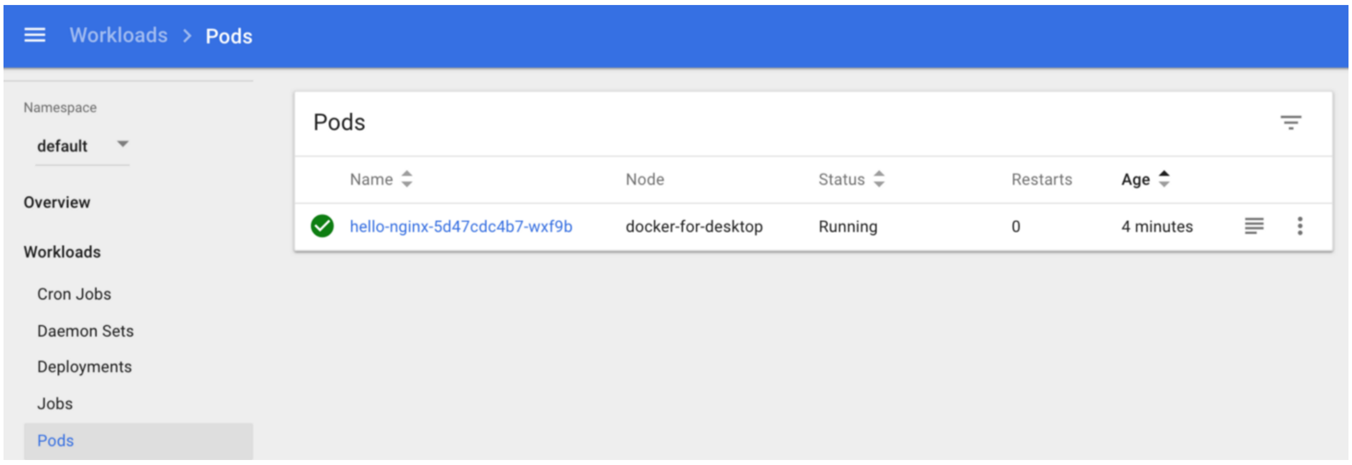


Pods

	Name	Node	Status	Restarts	Age
✓	hello-nginx-5d47cdc	docker-for-desktop	Running	0	3 minutes

Services

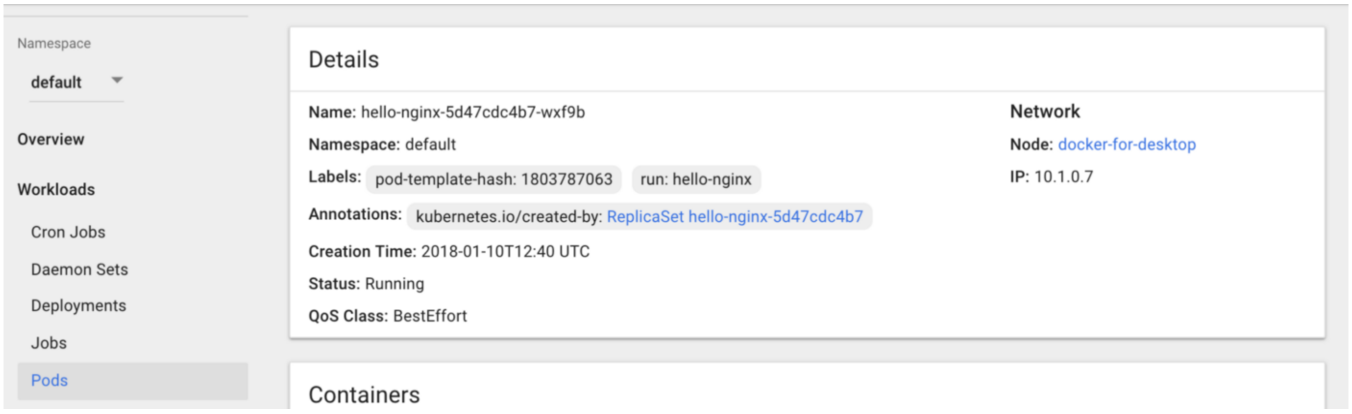
Alternately, you can also get to the Pods via the **Pods** link in the Workloads as shown below:



Pods

	Name	Node	Status	Restarts	Age
✓	hello-nginx-5d47cdc4b7-wxf9b	docker-for-desktop	Running	0	4 minutes

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



Details

Name: hello-nginx-5d47cdc4b7-wxf9b
Namespace: default
Labels: pod-template-hash: 1803787063 run: hello-nginx
Annotations: kubernetes.io/created-by: ReplicaSet hello-nginx-5d47cdc4b7
Creation Time: 2018-01-10T12:40 UTC
Status: Running
QoS Class: BestEffort

Network
Node: docker-for-desktop
IP: 10.1.0.7

Containers

You can see that it has been given some default labels. You can see its IP address. It is part of the node named **docker-for-desktop**.

There are some interesting links that you will find on this page as shown below, via which you can directly EXEC into the pods or see the logs too.

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:

```
$ kubectl get pods
```

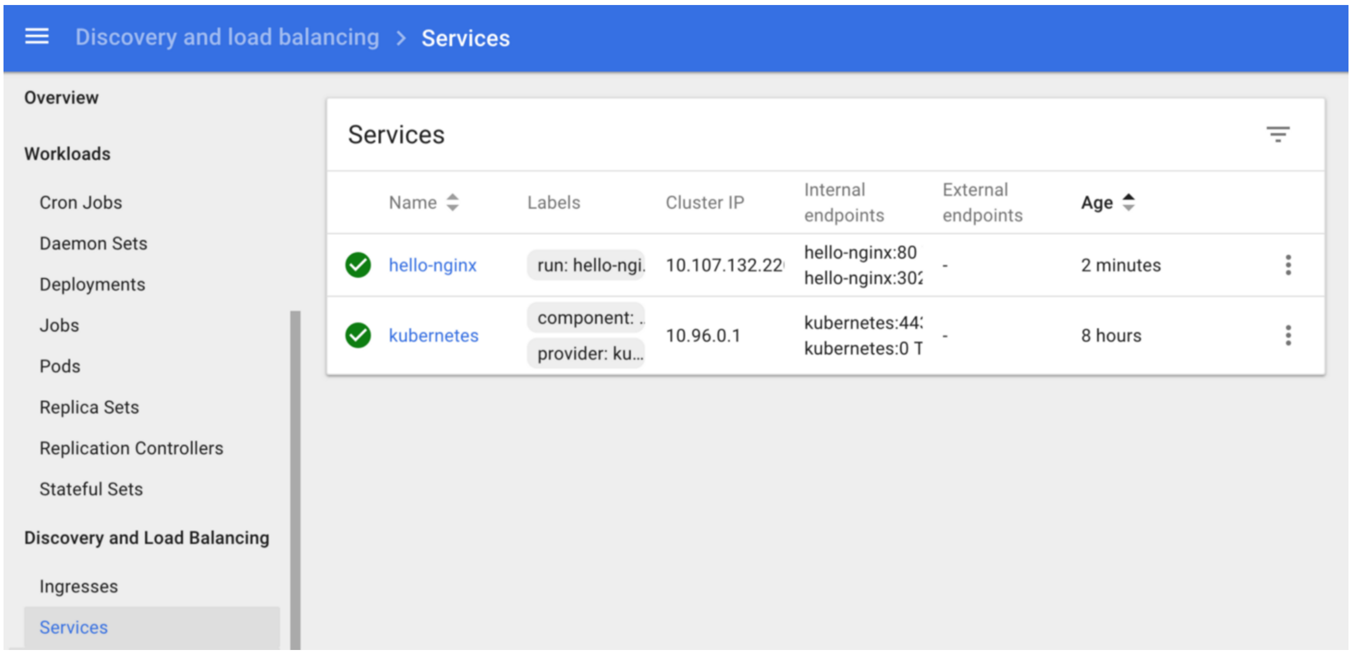
```
NAME                        READY STATUS  RESTARTS AGE
hello-nginx-5d47cdc4b7-wxf9b 1/1   Running 0        10m

$ kubectl describe pod hello-nginx-5d47cdc4b7-wxf9b

Name: hello-nginx-5d47cdc4b7-wxf9b
Namespace: default
Node: docker-for-desktop/192.168.65.3
Start Time: Wed, 10 Jan 2018 18:10:35 +0530
Labels: pod-template-hash=1803787063
run=hello-nginx

Annotations: kubernetes.io/created-by=
{“kind”:”SerializedReference”,”apiVersion”:”v1",”reference”:
{“kind”:”ReplicaSet”,”namespace”:”default”,”name”:”hello-nginx-
5d47cdc4b7",”uid”:”7415cff7-f603–11e7–9f7b-025000000…

Status: Running
IP: 10.1.0.7
Created By: ReplicaSet/hello-nginx-5d47cdc4b7
Controlled By: ReplicaSet/hello-nginx-5d47cdc4b7

Containers:
hello-nginx:
Container ID:
docker://a0c3309b61be4473bf6924ea2be9795de660f49bda36492785f94627690
cbdae
Image: nginx
Image ID: docker-
pullable://nginx@sha256:285b49d42c703fdf257d1e2422765c4ba9d3e37768d6
ea83d7fe2043dad6e63d
Port: 80/TCP
State: Running

...// 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:

```
$ kubectl get deployments

NAME        DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
hello-nginx 1       1       1          1         19m

$ kubectl 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.



Services

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
hello-nginx	run: hello-ngi.	10.107.132.22	hello-nginx:80 hello-nginx:302	-	2 minutes
kubernetes	component: . provider: ku...	10.96.0.1	kubernetes:44: kubernetes:0 T	-	8 hours

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

```
$ kubectl get services

NAME        TYPE      CLUSTER-IP     EXTERNAL-IP PORT(S)      AGE
hello-nginx NodePort  10.107.132.220 <none>      80:30259/TCP 1m
kubernetes  ClusterIP 10.96.0.1      <none>      443/TCP      8h
```

and

```
$ kubectl describe service hello-nginx

Name: hello-nginx
Namespace: default
Labels: run=hello-nginx
Annotations: <none>
Selector: run=hello-nginx
Type: NodePort
IP: 10.107.132.220
Port: <unset> 80/TCP
TargetPort: 80/TCP
NodePort: <unset> 30259/TCP
Endpoints: 10.1.0.7:80
Session Affinity: None
External Traffic Policy: Cluster
Events: <none>
```

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.

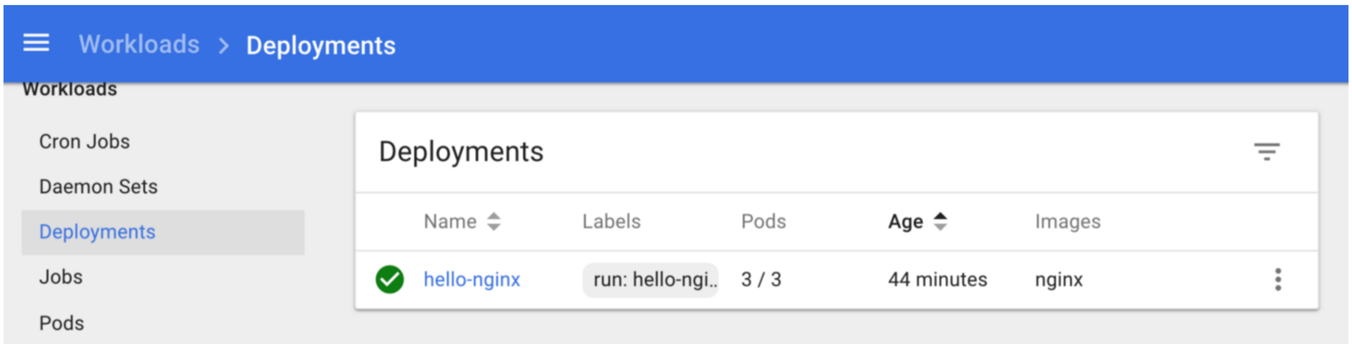
Let us go and see how we can scale this via the scale command. We want to scale it to 3 Pods.

```
$ kubectl scale --replicas=3 deployment/hello-nginx
deployment "hello-nginx" scaled
```

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

```
$ kubectl get deployment

NAME        DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
hello-nginx 3       3       3          3         45m
```

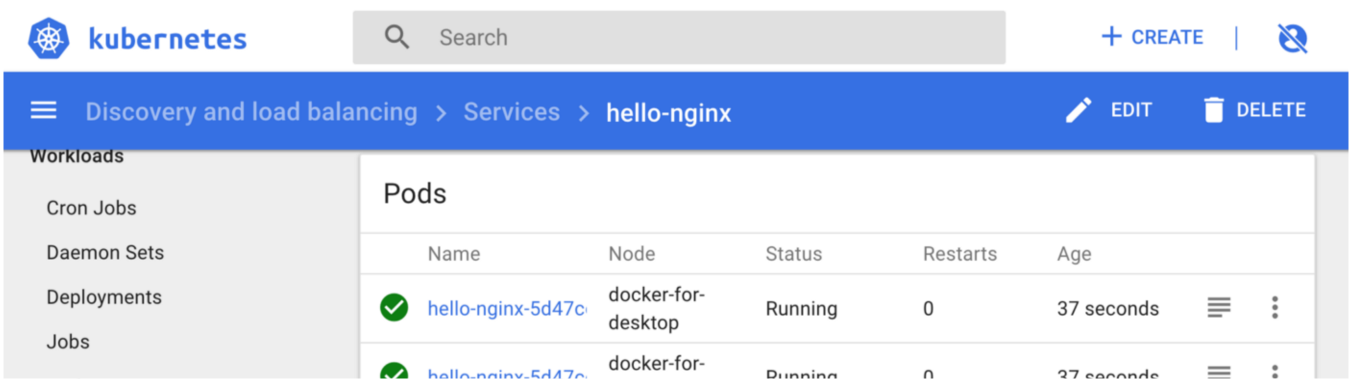
Now, if we visit the Dashboard for our Deployment:



Deployments

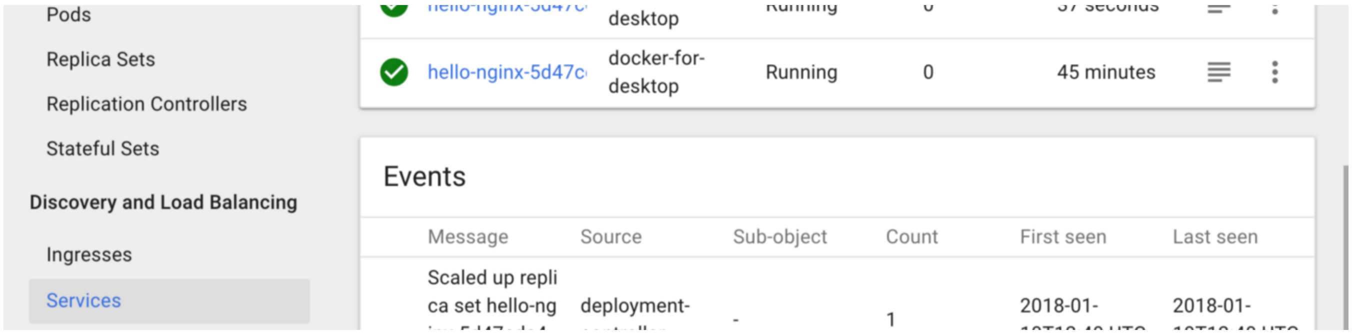
	Name	Labels	Pods	Age	Images
✓	hello-nginx	run: hello-ngi..	3 / 3	44 minutes	nginx

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



Pods

	Name	Node	Status	Restarts	Age
✓	hello-nginx-5d47c	docker-for-desktop	Running	0	37 seconds
✓	hello-nginx-5d47c	docker-for-	Running	0	37 seconds



Pods

Replica Sets

Replication Controllers

Stateful Sets

Discovery and Load Balancing

Ingresses

Services

	desktop			
hello-nginx-5d47c	docker-for-desktop	Running	0	45 minutes

Events

Message	Source	Sub-object	Count	First seen	Last seen
Scaled up repli ca set hello-ng	deployment-	-	1	2018-01-	2018-01-

Conclusion

Hope this blog post gets you started with Kubernetes with Docker for Mac. Please let me know about your experience in the comments. Now go forth and play the role of a helmsman.

Kubernetes Docker For Mac Docker Tutorial K8s