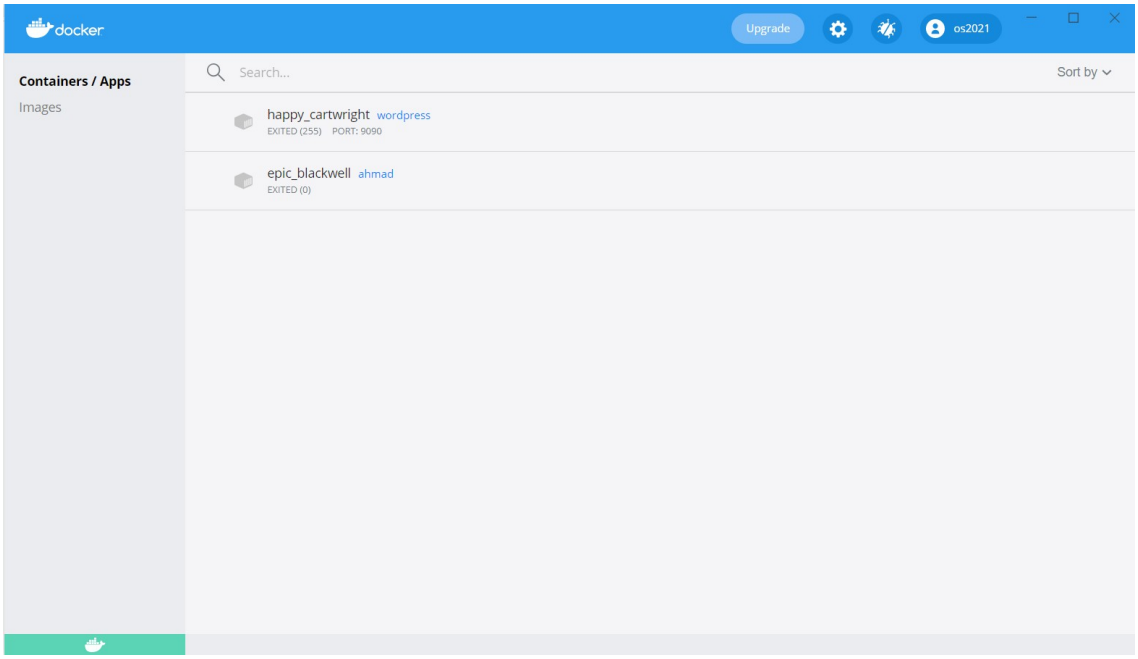
Before you start, make sure the following are installed on your laptop

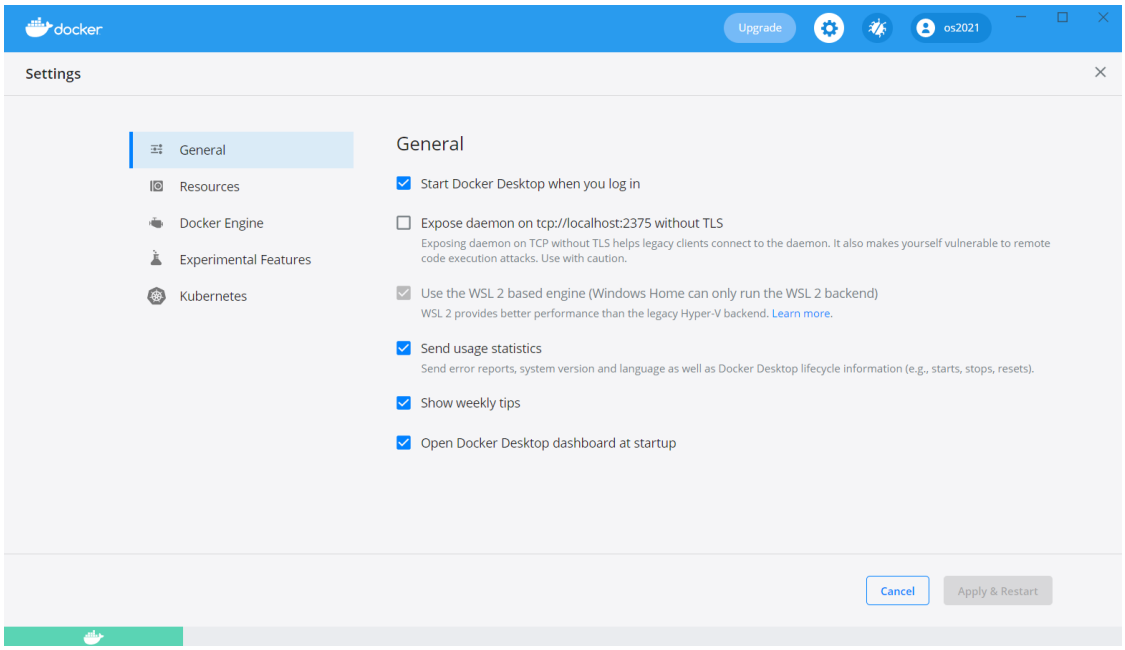
1. Kubernetes (***Minikube*** **-** https://www.youtube.com/watch?v=8h4FoWK7tIA)

    Can be installed easily if you have docker installed already on your system

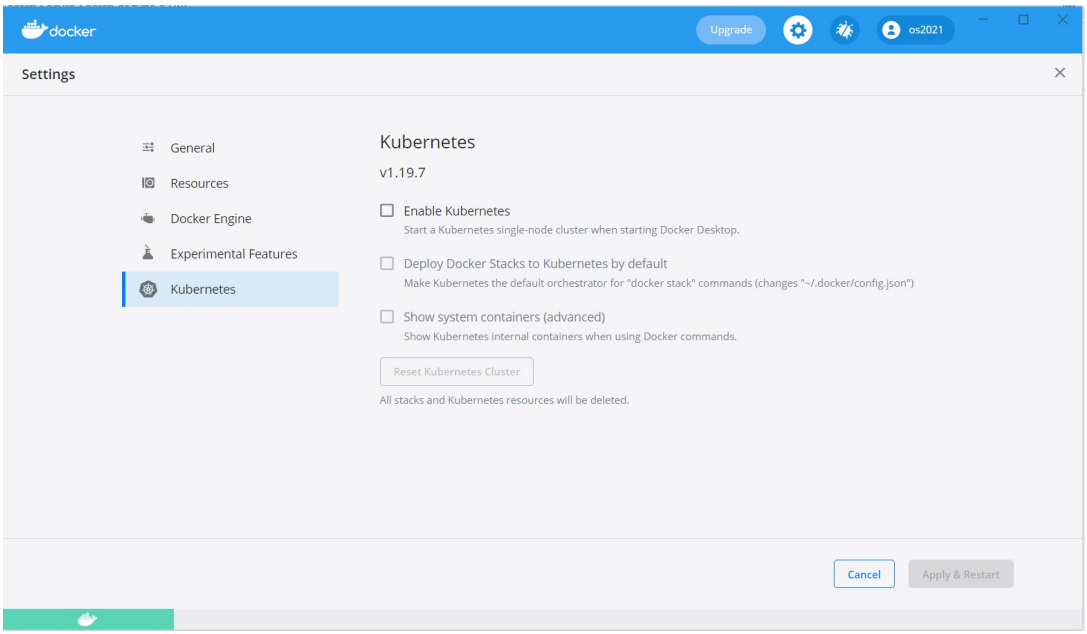
2. Kube controller (***kubectl*** - https://minikube.sigs.k8s.io/docs/start/)

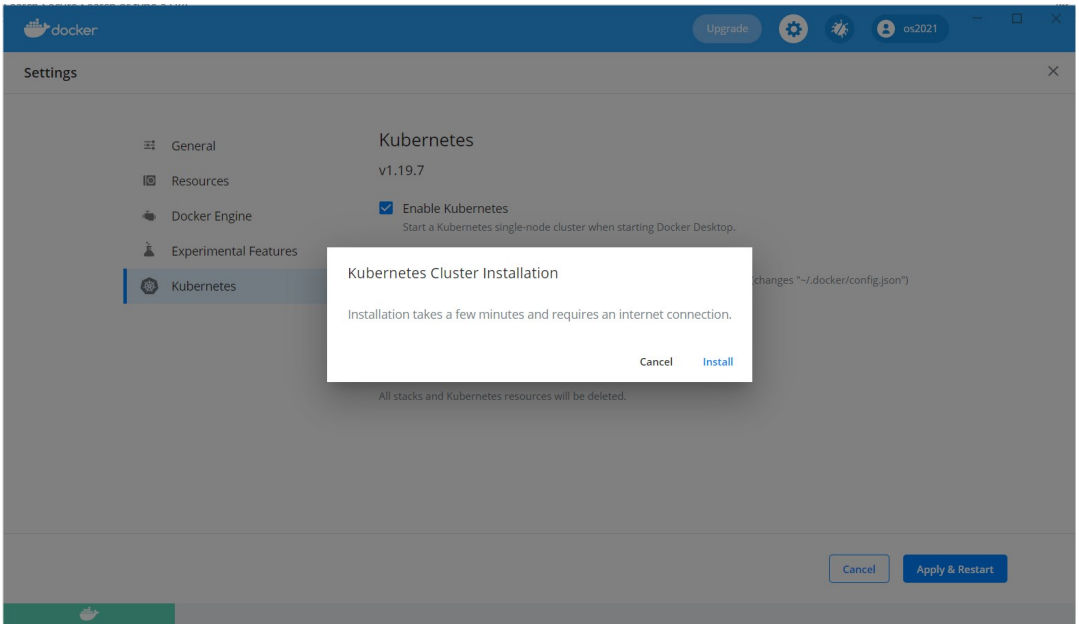
On ***Windows***, just follow the following pictures



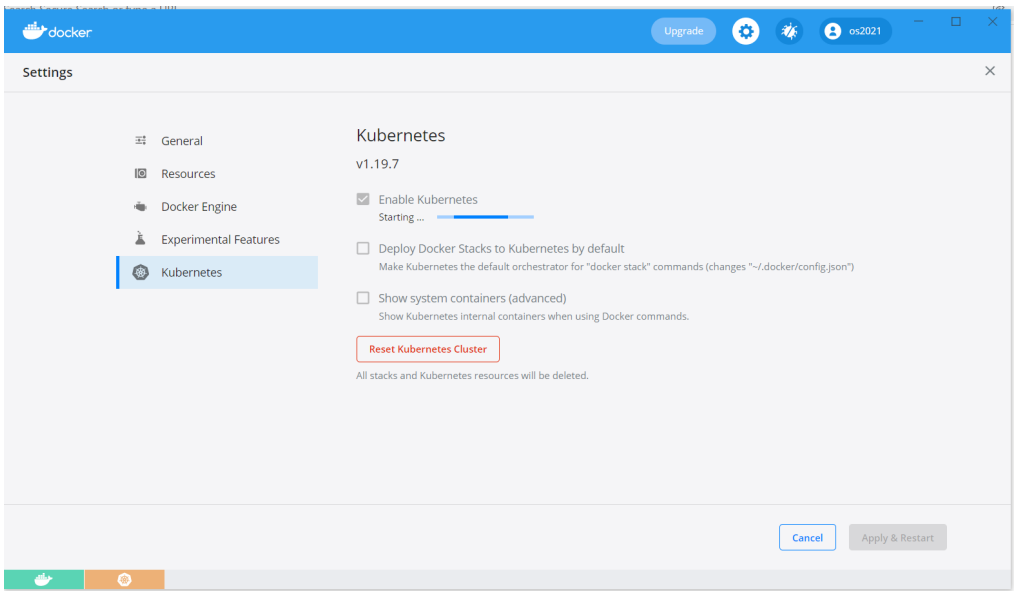


By: Ahmad ElBatanouni





By: [Ahmad ElBatanouni]



---

1. To see a list of nodes (currently we only have only one node since we used ***minikube***)

$ kubectl get nodes

```
ahmad@Batanouni:~$ kubectl get nodes
NAME       STATUS   ROLES    AGE     VERSION
minikube   Ready    master   6d23h   v1.19.4
```

if we installed ***kubeadmin*** instead, on ***multiple*** nodes we would get something like this

```
root@kubemaster:/home/osboxes# kubectl get nodes
NAME         STATUS    ROLES     AGE       VERSIO
kubemaster   Ready     master    7m        v1.9.4
kubenode1    Ready     <none>    4m        v1.9.4
kubenode2    Ready     <none>    4m        v1.9.4
```

2. To list all the ***pod***s

$ kubectl get pods

3. To create a new pod (pod is the smallest unit in Kubernetes which holds inside a docker container)

$ kubectl run nginx --image=nginx

4. To view more information about the pod … like showing on which node it was deployed, run

$ kubectl describe pods

By: Ahmad ElBatanouni

5. To delete a pod

$ kubectl delete pod podName

6. create a pod from a ***yaml*** file (files are needed for kubernetes to automatically restart failing pods)

$ kubectl create -f pod-definition.yml
All YAML files in kubernetes follow the following structure

```
apiVersion:
kind:
metadata:

spec:
```

1. apiVersion

Can be one of [“v1”, “apps/v1”] depending on what object we’re trying to create

Pod	v1
Deployment	apps/v1
Service	v1

2. kind

Can be one of [“Pod”, “Service”, “Deployment”]

3. metadata

add information about the item we are trying to create

4. spec

in case of creating a

4.1 ***POD***

you add information about the container/s you want to include in the pod and their resources (limits)

4.2 ***Deployment***

you specify the number of replicas (pods) that must be running all of the time, and also specify a template

Samples for ***POD*** & ***Deployment*** files are in the samples folder.

By: Ahmad ElBatanouni

Pods

Let's start by creating our first POD, go and fetch the template from the top of this page

```
apiVersion:
kind:
metadata:

spec:
```

Since we're trying to create a POD, the suitable ***apiVersion*** is ***v1*** … we know that from the table in the page above

```
apiVersion: v1
kind:
metadata:

spec:
```

Next, we set the ***kind*** to ***pod***

```
apiVersion: v1
kind: pod
metadata:

spec:
```

Next, we set the ***metadata*** .. here we can specify 2 attributes (***name*** & ***label***)
- *name* .. is just the name of the entity we're trying to create (the POD in this case)
- *label* .. allows us to give the created POD/s a label that we can use later to perform some kind of operation on ***all the PODs having this label***

```
apiVersion: v1
kind: pod
metadata:
  name: any-name-that-you-like
  labels:
    label-name-1: label-value
    another-label: my-app-name
spec:
```

Note that we can add more than one label.

By: Ahmad ElBatanouni

Finally, we move on to ***spec*** where we describe the kind of container/s we want to include in the pod

we do that by adding a child called ***containers***

```
apiVersion: v1
kind: pod
metadata:
  name: any-name-that-you-like
  labels:
    label-name-1: label-value
    another-label: my-app-name
spec:
  containers:
```

this attribute is in fact an ***array*** where you can add details of ***more than one*** container. Every item in this array is preceded with a dash (**-**) … for example

```
apiVersion: v1
kind: pod
metadata:
  name: any-name-that-you-like
  labels:
    label-name-1: label-value
    another-label: my-app-name
spec:
  containers:
    - details of the first container
    - details of the second container
    - details of the third container
```

Note that the above syntax is invalid … it is just used to explain how arrays in Yaml are created.

Next we specify the details of every item (container) like the ***image name***, ***container name***, ***resource limits (like how much ram and CPU can it use)***

```
apiVersion: v1
kind: pod
metadata:
  name: any-name-that-you-like
  labels:
    label-name-1: label-value
    another-label: my-app-name
spec:
  containers:
    - image: nginx
      name: nginx-container
      resources:
        limits:
          memory: "512Mi"
          cpu: "1"
```

Now that's we're done of the pod file … we can create the pod itself using the following command

$ kubectl create -f filename.yml

To make a pod accessible to the outside world, and not just withing the Kubernetes cluster, you'd use the ***port-forward*** command, giving a name of a deployment along with a mapping between a port on the host and a port inside the pod

$ kubectl port-forward [name-of-pod] 8080:80

By: Ahmad ElBatanouni

Deployments

Another concept/entity is the ***deployment controller*** is a way to ***manage*** pods, by introducing the following capabilities

- ***Deploy/Create*** multiple instances of your application
- ***Self-healing*** making sure the ***desired number*** of pods is always there, if a pod goes down, the deployment controller spins up a new one
- ***Update*** the pods/containers with newer images using what's called "rolling update" which doesn't stop all the pods immediately and pull the new image/s … ***instead*** it stops the existing pods either one by one or at a controlled rate resulting what's called a zero-downtime update.
- ***Go back/Rollback*** to an older deployment.
- ***Scale up/down*** the number of pods you have in the cluster.

To create a new deployment controller, we use the same template as before, in addition to that we ***specify*** 2 attributes under ***spec*** … the first is ***replicas*** and the second is ***template***

replicas specify the number of pods we need … the deployment then makes sure the specified number of pods are always up and running … and if for some reason one of the pods fails … it spins up a new pod (with the characteristics which we describe inside of the second attribute ***template***) … ***it also deletes pods if needed***.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-first-deployment

spec:
  replicas: 3
  template:

      Details of the pod to
      create
```

Again the above syntax (***details of the pod to create***) is not valid … it only shows that we use the value here to spin up new PODs when needed.

By: Ahmad ElBatanouni

Those details are exactly the same as the details we specified in the pod creation file .. except we ignore the first 2 attributes (apiVersion & kind)

So we can simply open the first pod file and ***copy all what's under (metadata & spec)*** and paste them here in the box

pod.yaml

```
apiVersion: v1
kind: pod
metadata:
  name: any-name-that-you-like
  labels:
    label-name-1: label-value
    another-label: my-app-name
spec:
  containers:
    - image: nginx
      name: nginx-container
      resources:
        limits:
          memory: "512Mi"
          cpu: "1"
```

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment-controller
spec:
  replicas: 3
  template:
      metadata:
        name: any-name-that-you-like
        labels:
          label-name-1: label-value
          another-label: my-app-name
      spec:
        containers:
          - image: nginx
            name: nginx-container
            resources:
              limits:
                memory: "512Mi"
                cpu: "1"
```

As usual, create the deployment using

$ kubectl create -f filename.yml

By: Ahmad ElBatanouni

To see the deployment controller we've just created

$ kubectl get deployment

The deployment in turn creates a replica set with the same name, check that by running

$ kubectl get rs

Another important attribute that we can specify under ***spec*** is called ***selector*** which is used by the deployment controller to track all the pods matching that label

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment

spec:
  selector:
    matchLabels:
      my-label: some-value
  replicas: 3
  template:

      metadata:
        name: any-name-that-you-like
        labels:
          my-label: some-value
      spec:
        containers:
          - image: nginx
            name: nginx-container
```

To ***update*** the existing pods with a different version of the container image, we have 2 strategies

1. Recreate: Stops all the pods at once, and recreates them using the new image .. the ***problem*** here is that users will be unable to access the application during the update time.

2. RollingUpdate (Default): stops and updates the existing pods either one by one or at a controlled rate so that users can access the application on other pods.

We can then update the yaml file with one of those 2 strategies ***along with the update we want to do to the container image***

By: Ahmad ElBatanouni

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment

spec:
  selector:
    matchLabels:
      my-label: some-value
  replicas: 3
  strategy:
    type: Recreate
  template:

      metadata:
        name: any-name-that-you-like
        labels:
          my-label: some-value
      spec:
        containers:
          - image: nginx:1.16.1
            name: nginx-container
```

Apply the changes using

$ kubectl apply -f deployment.yml

Now try to change the number of replicas (either increase or decrease them) and again run:

$ kubectl apply -f deployment.yml

Let's assume that updating to the new image wasn't the correct decision, something is just wrong with that image … we want to move back (***rollout***) to the old image (in Kubernetes world it's called the old ***revision***)

$ kubectl rollout undo deployment/my-deployment

If you now run ***$kubectl get pods*** you can see the pods are brought down one by one and new pods are created instead.

To ***scale up/down*** the number of replicas in a certain deployment, you'd simply do something like this

$ kubectl scale deployment [name-of-depl] --replicas=5

By: Ahmad ElBatanouni

Finally, to make a pod/deployment accessible to the outside world, and not just withing the Kubernetes cluster, you'd use the ***port-forward*** command, giving a name of a deployment along with a mapping between a port on the host and a port on

$ kubectl port-forward [name-of-pod] 8080:80

$ kubectl port-forward [name-of-deployment] 8080:80

8080 is the host port
80 is the pod/deployment port

Assignment

Create a Kubernetes deployment file that creates 2 replicas of ***wordpress***

1. Show the names of all the replicas
2. Show an example of how you can access wordpress from the browser (P.S. port forwarding).
3. Increase the number of replicas to 3 then show again the names of all of them.
4. Change the image from ***wordpress*** to ***nginx***

By: Ahmad ElBatanouni