Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

Step 1 - Start Minikube
Minikube has been installed and configured in the environment. Check that it is properly installed, by running the minikube version command:

minikube version

Start the cluster, by running the minikube start command:

minikube start --wait=false

Great! You now have a running Kubernetes cluster in your online terminal. Minikube started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

```
$ minikube version
minikube version: v1.8.1
commit: cbda04cf6bbe65e987ae52bb393c10099ab62014
$ minikube start --wait=false
* minikube v1.8.1 on Ubuntu 18.04
* Using the none driver based on user configuration
* Running on localhost (CPUs=2, Memory=2460MB, Disk=145651MB) ...
* OS release is Ubuntu 18.04.4 LTS
* Preparing Kubernetes v1.17.3 on Docker 19.03.6 ...
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
* Launching Kubernetes ...
```

The minikube start command is used to start a local Kubernetes cluster using Minikube, which is a tool that enables you to run a single-node Kubernetes cluster on your local machine.

The --wait=false option is used to disable waiting for Kubernetes components to be ready before returning control to the user. By default, when you start a Minikube cluster, the command will wait until all the necessary components are up and running before returning control to the user. This can take a few minutes depending on your system configuration.

Disabling the wait option can be useful if you want to start the cluster quickly and don't want to wait for all the components to be fully operational before running your Kubernetes commands. However, if you disable the wait option, you may encounter errors if you try to run Kubernetes commands before all the necessary components are up and running.

Step 2 - Cluster Info
The cluster can be interacted with using the kubectl CLI. This is the main approach used for managing Kubernetes and the applications running on top of the cluster.

Details of the cluster and its health status can be discovered via
kubectl cluster-info

To view the nodes in the cluster using
kubectl get nodes

If the node is marked as NotReady then it is still starting the components.

This command shows all nodes that can be used to host our applications. Now we have only one node, and we can see that it's status is ready (it is ready to accept applications for deployment).

```
* Done! kubectl is now configured to use "minikube"
$ kubectl cluster-info
Kubernetes master is running at https://10.0.0.10:8443
KubeDNS is running at https://10.0.0.10:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
$ kubectl get nodes
NAME       STATUS   ROLES    AGE     VERSION
minikube   Ready    master   2m52s   v1.17.3
$
```

Step 3 - Deploy Containers
With a running Kubernetes cluster, containers can now be deployed.

Using kubectl run, it allows containers to be deployed onto the cluster –
kubectl create deployment first-deployment --image=katacoda/docker-http-server

The status of the deployment can be discovered via the running Pods - kubectl get pods

Once the container is running it can be exposed via different networking options, depending on requirements. One possible solution is NodePort, that provides a dynamic port to a container.

kubectl expose deployment first-deployment --port=80 --type=NodePort

The command below finds the allocated port and executes a HTTP request.

export PORT=$(kubectl get svc first-deployment -o go-template='{{range.spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}')
echo "Accessing host01:$PORT"
curl host01:$PORT

The result is the container that processed the request.

```
minikube   Ready    master   2m52s   v1.17.3
$ kubectl create deployment first-deployment --image=katacoda/docker-http-server
deployment.apps/first-deployment created
$ kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
first-deployment-666c48b44-gdsg8    0/1     ContainerCreating   0          2s
$ kubectl expose deployment first-deployment --port=80 --type=NodePort
service/first-deployment exposed
$ export PORT=$(kubectl get svc first-deployment -o go-template='{{range.spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}')
$ echo "Accessing host01:$PORT"
Accessing host01:31474
$ curl host01:$PORT
<h1>This request was processed by host: first-deployment-666c48b44-gdsg8</h1>
$ kubectl get nodes
NAME       STATUS   ROLES    AGE     VERSION
minikube   Ready    master   6m46s   v1.17.3
$
```

Step 4 - Dashboard
Enable the dashboard using Minikube with the command minikube addons enable dashboard

Make the Kubernetes Dashboard available by deploying the following YAML definition. This should only be used on Katacoda.

kubectl apply -f /opt/kubernetes-dashboard.yaml

The Kubernetes dashboard allows you to view your applications in a UI. In this deployment, the dashboard has been made available on port 30000 but may take a while to start.

To see the progress of the Dashboard starting, watch the Pods within the kube-system namespace using kubectl get pods -n kubernetes-dashboard -w

```
$ minikube addons enable dashboard
* The 'dashboard' addon is enabled
$ kubectl apply -f /opt/kubernetes-dashboard.yaml
namespace/kubernetes-dashboard configured
service/kubernetes-dashboard-katacoda created
$ cat ubectl get svc first-deployment
cat: ubectl: No such file or directory
cat: get: No such file or directory
cat: svc: No such file or directory
cat: first-deployment: No such file or directory
$ cat /opt/kubernetes-dashboard.yaml
apiVersion: v1
kind: Namespace
metadata:
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/minikube-addons: dashboard
  name: kubernetes-dashboard
  selfLink: /api/v1/namespaces/kubernetes-dashboard
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: kubernetes-dashboard
  name: kubernetes-dashboard-katacoda
  namespace: kubernetes-dashboard
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 9090
    nodePort: 30000
  selector:
    k8s-app: kubernetes-dashboard
  type: NodePort
$ kubectl get pods -n kubernetes-dashboard -w
NAME                                         READY   STATUS    RESTARTS   AGE
dashboard-metrics-scraper-7b64584c5c-fk572   1/1     Running   0          43s
kubernetes-dashboard-79d9cd965-lvwxl         1/1     Running   0          43s
```