In this scenario, you'll learn how to use Kubectl to create and launch Deployments, Replication Controllers and expose them via Services without writing yaml definitions. This allows you to quickly launch containers onto the cluster.

Step 1 - Launch Cluster
To start we need to launch a Kubernetes cluster.
Execute the command below to start the cluster components and download the Kubectl CLI.

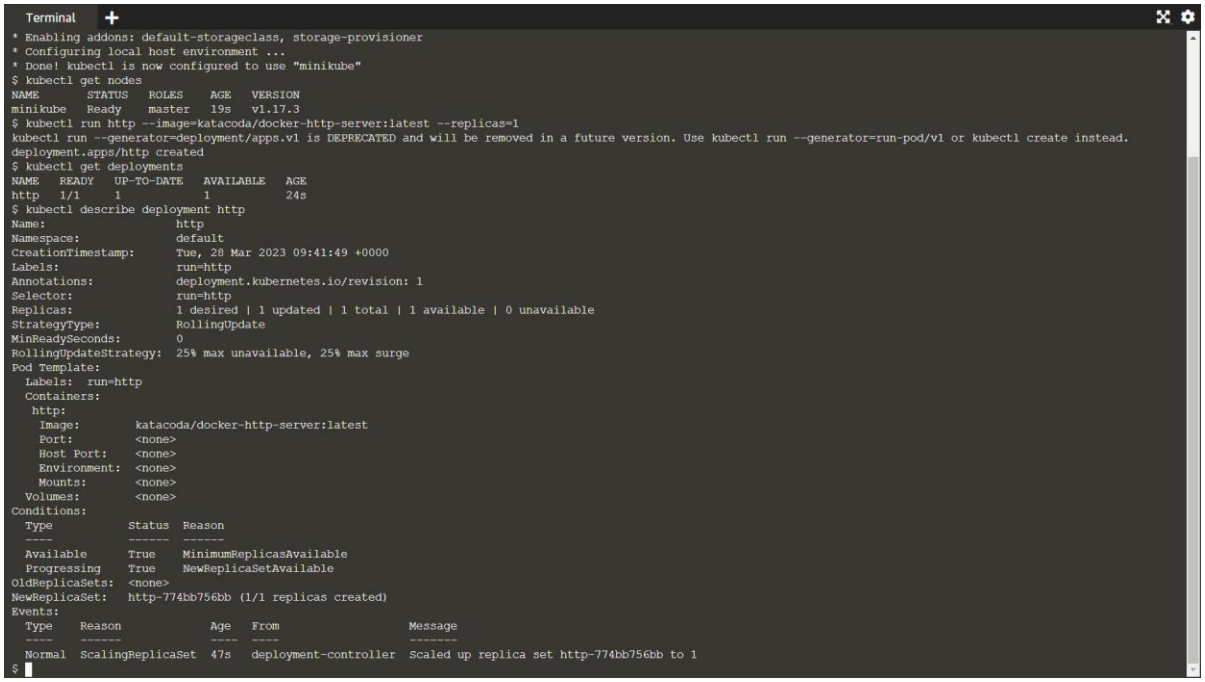
```
$ minikube start --wait=false
* minikube v1.8.1 on Ubuntu 18.04
* Using the none driver based on user configuration
* Running on localhost (CPUs=2, Memory=2460MB, Disk=145651MB) ...
* OS release is Ubuntu 18.04.4 LTS
* Preparing Kubernetes v1.17.3 on Docker 19.03.6 ...
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
* Launching Kubernetes ...
* Enabling addons: default-storageclass, storage-provisioner
* Configuring local host environment ...
* Done! kubectl is now configured to use "minikube"
$ kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   19s   v1.17.3
$
```

Step 2 - Kubectl Run
The run command creates a deployment based on the parameters specified, such as the image or replicas. This deployment is issued to the Kubernetes master which launches the Pods and containers required. Kubectl run_ is similar to docker run but at a cluster level.
The format of the command is kubectl run <name of deployment> <properties>

Task
The following command will launch a deployment called http which will start a container based on the Docker Image katacoda/docker-http-server:latest.
You can then use kubectl to view the status of the deployments

To find out what Kubernetes created you can describe the deployment process.

The description includes how many replicas are available, labels specified and the events associated with the deployment. These events will highlight any problems and errors that might have occurred.



In the next step we'll expose the running service.

Step 3 - Kubectl Expose
With the deployment created, we can use kubectl to create a service which exposes the Pods on a particular port.

Expose the newly deployed http deployment via kubectl expose. The command allows you to define the different parameters of the service and how to expose the deployment.
Task
Use the following command to expose the container port 80 on the host 8000 binding to the external-ip of the host.

```
$ kubectl expose deployment http --external-ip="10.0.0.7" --port=8000 --target-port=80
service/http exposed
```

You will then be able to ping the host and see the result from the HTTP service.

```
$ curl http://10.0.0.7:8000
<h1>This request was processed by host: http-774bb756bb-9tqrb</h1>
```

kubectl expose deployment http --external-ip="10.0.0.7" --port=8000 --target-port=80

The command "kubectl expose deployment" is used to create a service to expose a deployment.

In the given command, the service name is not specified, so it will be generated automatically based on the deployment name. The deployment name is "http".

The flag "--external-ip" specifies the external IP address to use for the service. In this case, it is set to "10.0.0.7". Note that this external IP address must be an IP address that is available in the cluster network.

The flag "--port" specifies the port on the service that will be exposed. In this case, it is set to "8000".

The flag "--target-port" specifies the port on the pods that the service should forward traffic to. In this case, it is set to "80".

Putting it all together, the command "kubectl expose deployment http --external-ip="10.0.0.7" --port=8000 --target-port=80" creates a service named "http" that exposes port 8000 on the service and forwards traffic to port 80 on the pods, using the external IP address "10.0.0.7".

Step 4 - Kubectl Run and Expose

With kubectl run it's possible to create the deployment and expose it as a single command.

Task

Use the command command to create a second http service exposed on port 8001.

```
$ kubectl run httpexposed --image=katacoda/docker-http-server:latest --replicas=1 --port=80 --hostport=8001
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run
 --generator=run-pod/v1 or kubectl create instead.
deployment.apps/httpexposed created
```

The command "kubectl run" is used to create a new deployment, and the command "kubectl run httpexposed --image=katacoda/docker-http-server:latest --replicas=1 --port=80 --hostport=8001" creates a new deployment with the following parameters:

The deployment name is "httpexposed"

The image used for the deployment is "katacoda/docker-http-server:latest"

The number of replicas is set to 1 with the "--replicas" flag.

The container will listen on port 80 inside the container using the "--port" flag.

The host port to use for accessing the container is set to 8001 with the "--hostport" flag.

This command creates a deployment with a single replica running the "katacoda/docker-http-server" image, which will listen on port 80 inside the container. The host port 8001 is exposed for accessing the container from the host machine.

Note that this command uses the deprecated "kubectl run" command with the "--hostport" flag, which is not recommended for production environments. In Kubernetes v1.22, the "kubectl run" command has been changed to only create a deployment by default, and the "--hostport" flag has been removed. Instead, you can use a service with a NodePort type to expose the deployment to the outside world.

You should be able to access it using

```
$ curl http://10.0.0.7:8001
<h1>This request was processed by host: httpexposed-68cb8c8d4-76pcr</h1>
$
```

Under the covers, this exposes the Pod via Docker Port Mapping. As a result, you will not see the service listed using

```
$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
http         ClusterIP   10.98.191.171   10.0.0.7      8000/TCP   18m
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    29m
```

Httpexposed not present .

To find the details you can use

```
$ docker ps | grep httpexposed
f2a762e62e11        katacoda/docker-http-server   "/app"                   7 minutes ago       Up 7 minutes
                        k8s_httpexposed_httpexposed-68cb8c8d4-76pcr_default_66f6d347-18d6-466d-afbf-1928c684676
_0
c016622c9fe8        k8s.gcr.io/pause:3.1          "/pause"                 7 minutes ago       Up 7 minutes
  0.0.0.0:8001->80/tcp   k8s_POD_httpexposed-68cb8c8d4-76pcr_default_66f6d347-18d6-466d-afbf-1928c6846761_0
$
```

Pause Containers

Running the above command you'll notice the ports are exposed on the Pod, not the http container itself. The Pause container is responsible for defining the network for the Pod. Other containers in the pod share the same network namespace. This improves network performance and allow multiple containers to communicate over the same network interface..

Step 5 - Scale Containers

With our deployment running we can now use kubectl to scale the number of replicas.

Scaling the deployment will request Kubernetes to launch additional Pods. These Pods will then automatically be load balanced using the exposed Service.

Task

The command kubectl scale allows us to adjust the number of Pods running for a particular deployment or replication controller.

```
$ kubectl scale --replicas=3 deployment http
deployment.apps/http scaled
$
```

Listing all the pods, you should see three running for the http deployment

```
$ kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
http-774bb756bb-9tqrb        1/1     Running   0          29m
http-774bb756bb-f7h5t        1/1     Running   0          74s
http-774bb756bb-kk8wn        1/1     Running   0          74s
httpexposed-68cb8c8d4-76pcr  1/1     Running   0          11m
```

Once each Pod starts it will be added to the load balancer service. By describing the service you can view the endpoint and the associated Pods which are included.

```
$ kubectl describe svc http
Name:              http
Namespace:         default
Labels:            run=http
Annotations:       <none>
Selector:          run=http
Type:              ClusterIP
IP:                10.98.191.171
External IPs:      10.0.0.7
Port:              <unset>  8000/TCP
TargetPort:        80/TCP
Endpoints:         172.18.0.4:80,172.18.0.6:80,172.18.0.7:80
Session Affinity:  None
Events:            <none>
```

Making requests to the service will request in different nodes processing the request.

```
$ curl http://10.0.0.7:8000
<h1>This request was processed by host: http-774bb756bb-9tqrb</h1>
$ curl http://10.0.0.7:8000
<h1>This request was processed by host: http-774bb756bb-9tqrb</h1>
$ curl http://10.0.0.7:8000
<h1>This request was processed by host: http-774bb756bb-kk8wn</h1>
$ curl http://10.0.0.7:8000
<h1>This request was processed by host: http-774bb756bb-9tqrb</h1>
$ curl http://10.0.0.7:8000
<h1>This request was processed by host: http-774bb756bb-f7h5t</h1>
$
```