Kubernetes Linux Acedemy

Docker installation:
We are installing it on 3 nodes.
1 master:
bakumar2c.mylabserver.com
2 worker nodes:
bakumar3c.mylabserver.com,
bakumar4c.mylabserver.com
Run below commands in all 3 nodes

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"


apt-get update

 apt-get install -y docker-ce=18.06.1~ce~3-0~ubuntu

 apt-mark hold docker-ce

docker ps
```

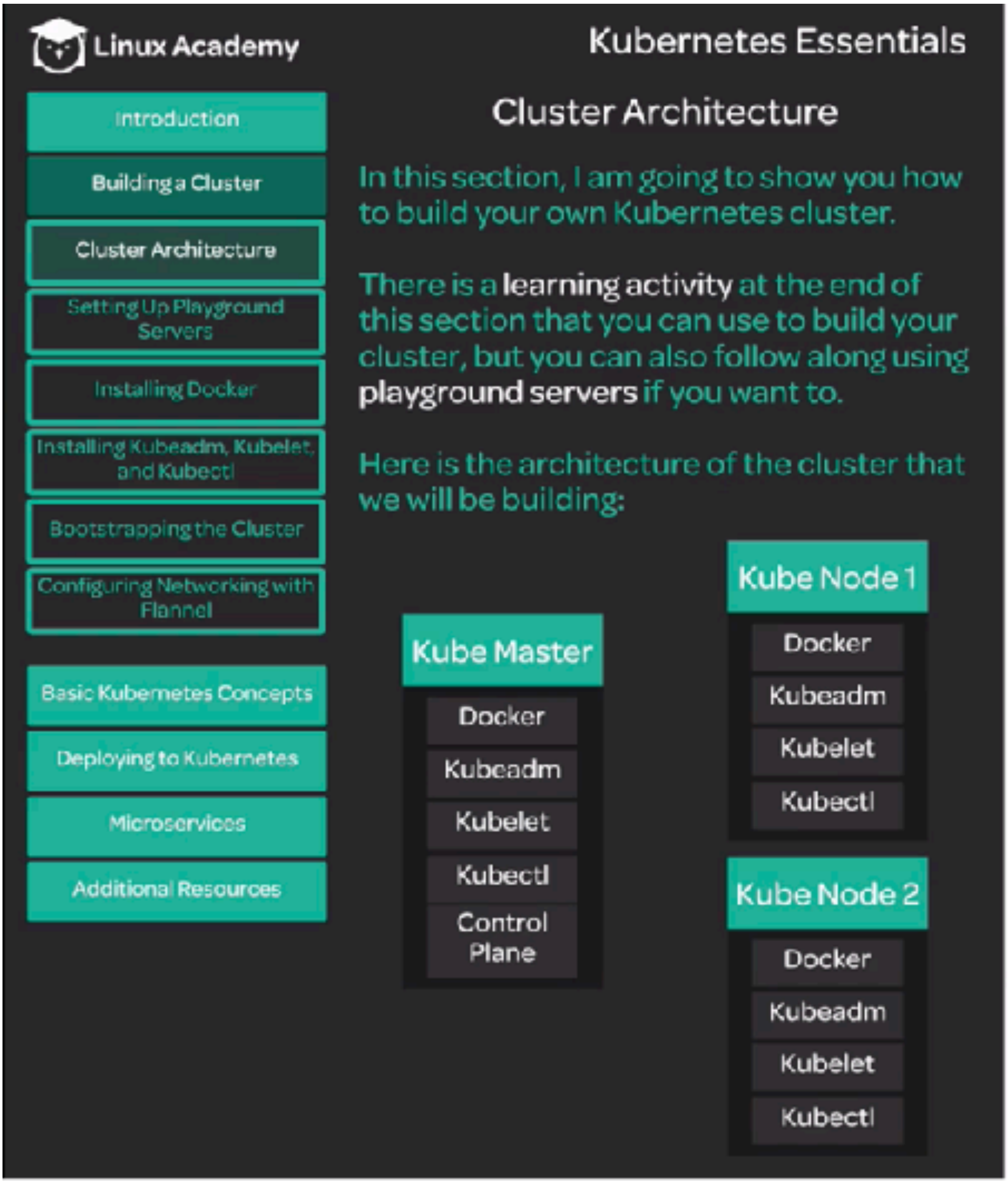
The first step in setting up a new cluster is to install a container runtime such as Docker. In this lesson, we will be installing Docker on our three servers in preparation for standing up a Kubernetes cluster. After completing this lesson, you should have three playground servers, all with Docker up and running.

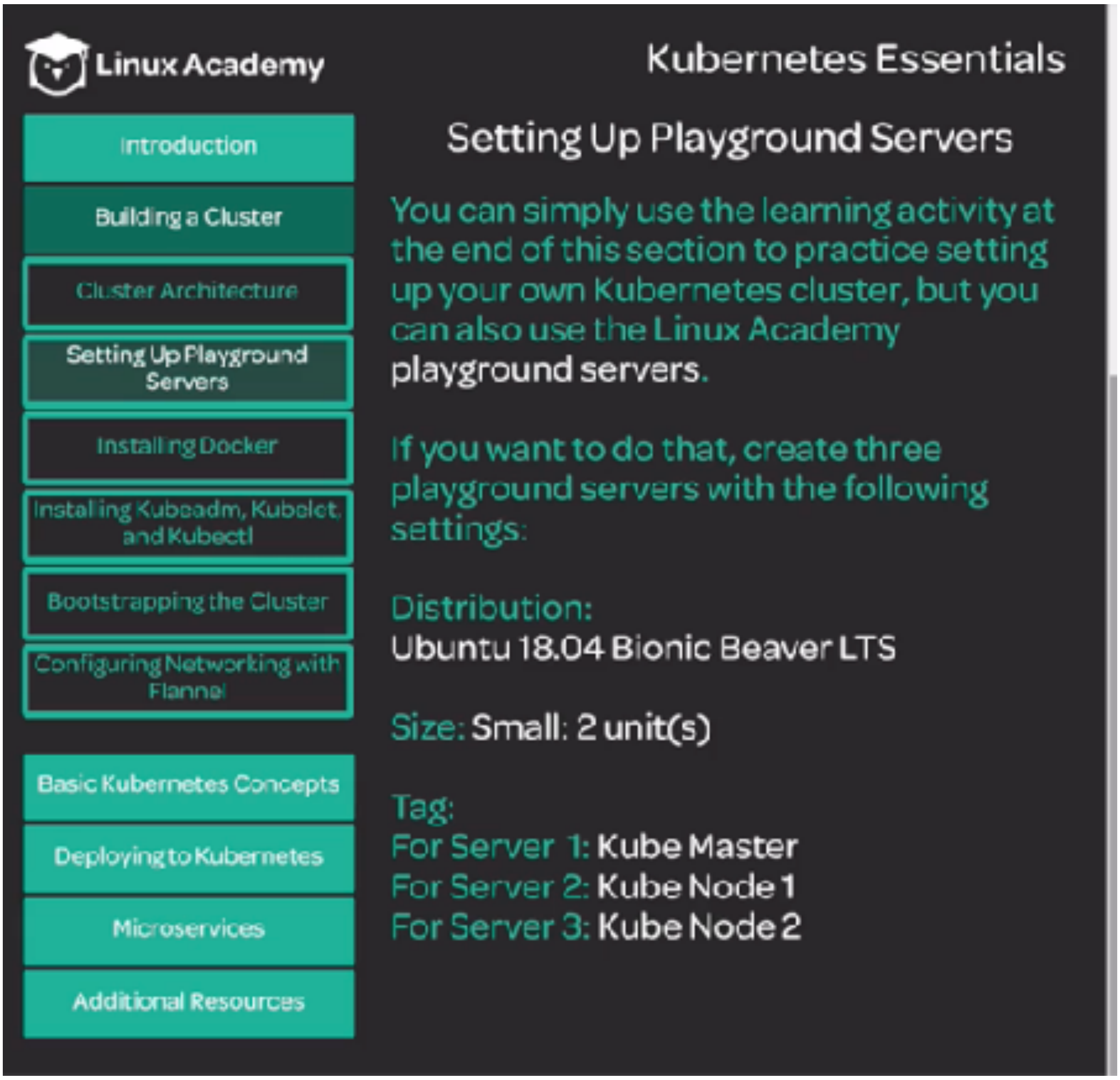
Here are the commands used in this lesson:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

sudo apt-get update

sudo apt-get install -y docker-ce=18.06.1~ce~3-0~ubuntu

sudo apt-mark hold docker-ce
```

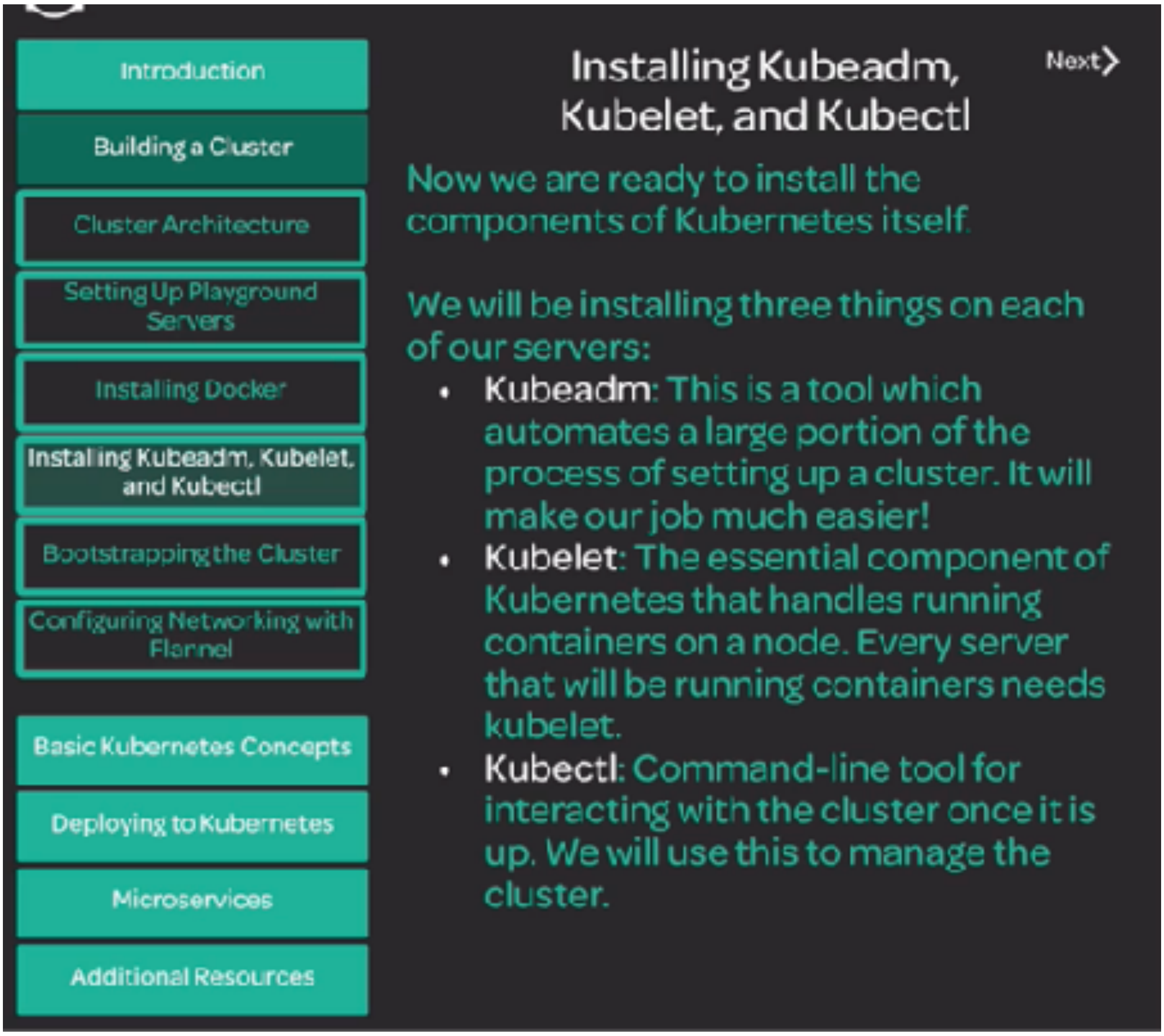
You can verify that docker is working by running this command:

```
sudo docker version
```


Linux Academy
Kubernetes Essentials
Introduction
Building a Cluster
Cluster Architecture
Setting Up Playground Servers
Installing Docker
Installing Kubeadm, Kubelet, and Kubectl
Bootstrapping the Cluster
Configuring Networking with Flannel
Basic Kubernetes Concepts
Deploying to Kubernetes
Microservices
Additional Resources
Cluster Architecture
In this section, I am going to show you how to build your own Kubernetes cluster.
There is a learning activity at the end of this section that you can use to build your cluster, but you can also follow along using playground servers if you want to.
Here is the architecture of the cluster that we will be building:
Kube Master
Docker
Kubeadm
Kubelet
Kubectl
Control Plane
Kube Node 1
Docker
Kubeadm
Kubelet
Kubectl
Kube Node 2
Docker
Kubeadm
Kubelet
Kubectl



Linux Academy
Kubernetes Essentials
Introduction
Building a Cluster
Cluster Architecture
Setting Up Playground Servers
Installing Docker
Installing Kubeadm, Kubelet, and Kubectl
Bootstrapping the Cluster
Configuring Networking with Flannel
Basic Kubernetes Concepts
Deploying to Kubernetes
Microservices
Additional Resources
Setting Up Playground Servers
You can simply use the learning activity at the end of this section to practice setting up your own Kubernetes cluster, but you can also use the Linux Academy playground servers.
If you want to do that, create three playground servers with the following settings:
Distribution:
Ubuntu 18.04 Bionic Beaver LTS
Size: Small: 2 unit(s)
Tag:
For Server 1: Kube Master
For Server 2: Kube Node 1
For Server 3: Kube Node 2



Install below Kubernetes steps In all 3 nodes

```
 curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF

apt-get update

apt-get install -y kubelet=1.12.7-00 kubeadm=1.12.7-00 kubectl=1.12.7-00

apt-mark hold kubelet kubeadm kubectl

 kubeadm  version
```

Now that Docker is installed, we are ready to install the Kubernetes components. In this lesson, I will guide you through the process of installing Kubeadm, Kubelet, and Kubectl on all three playground servers. After completing this lesson, you should be ready for the next step, which is to bootstrap the cluster.

Here are the commands used to install the Kubernetes components in this lesson. Run these on all three servers.

NOTE: There are some issues being reported when installing version 1.12.2-00 from the Kubernetes ubuntu repositories. You can work around this by using version 1.12.7-00 for kubelet, kubeadm, and kubectl.

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF

sudo apt-get update

sudo apt-get install -y kubelet=1.12.7-00 kubeadm=1.12.7-00 kubectl=1.12.7-00

sudo apt-mark hold kubelet kubeadm kubectl
```

After installing these components, verify that Kubeadm is working by getting the version info.

```
kubeadm version
```

Bootstrapping Kubernetes cluster

Run Below commands in Master node.

```
kubeadm init --pod-network-cidr=10.244.0.0/16
```

<It will display set of further commands to execute on same master node along with token key to join worker nodes into cluster.>

Example:

[

To start using your cluster, you need to run the following as a regular user:

```
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You can now join any number of machines by running the following on each node
as root:

```
  kubeadm join 172.31.41.71:6443 --token ocf5ib.xzicjfjwt7slt271 --discovery-token-ca-cert-hash sha256:e6332502487041e1314afec51b7a31486edc12d1a41861c1380d8791816bf117
```

]

Run above shown commands in master node:

```
mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Then for checking kubectl

#kubectl version

output:

Client Version: ...
Server Version:..

Run above master node shown output of “toker join “in worker node

From bakumar3c.mylabserver.com,

```
sudo kubeadm join 172.31.41.71:6443 --token ocf5ib.xzicjfjwt7slt271 --discovery-token-ca-cert-hash
sha256:e6332502487041e1314afec51b7a31486edc12d1a41861c1380d8791816bf117
```

From bakumar4c.mylabserver.com

```
sudo kubeadm join 172.31.41.71:6443 --token ocf5ib.xzicjfjwt7slt271 --discovery-token-ca-cert-hash
sha256:e6332502487041e1314afec51b7a31486edc12d1a41861c1380d8791816bf117
```

Once ran above commands, we run below command from Master node to see list of joined nodes

```
root@bakumar2c:~# kubectl  get nodes
NAME                       STATUS     ROLES    AGE    VERSION
bakumar2c.mylabserver.com  NotReady   master   8m5s   v1.12.7
bakumar3c.mylabserver.com  NotReady   <none>   3m46s  v1.12.7
bakumar4c.mylabserver.com  NotReady   <none>   3m31s  v1.12.7
```

Till we setup networking for cluster, “STATUS” will show up in “Not Ready”

Now we are ready to get a real Kubernetes cluster up and running! In this lesson, we will bootstrap the cluster on the Kube master node. Then, we will join each of the two worker nodes to the cluster, forming an actual multi-node Kubernetes cluster.

Here are the commands used in this lesson:

- On the Kube master node, initialize the cluster:

  ```
  sudo kubeadm init --pod-network-cidr=10.244.0.0/16
  ```

  That command may take a few minutes to complete.
- When it is done, set up the local kubeconfig:

  ```
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
  ```

- Verify that the cluster is responsive and that Kubectl is working:

  ```
  kubectl version
  ```

  You should get `Server Version` as well as `Client Version`. It should look something like this:

  ```
  Client Version: version.Info{Major:"1", Minor:"12", GitVersion:"v1.12.2", GitCommit:"17c77c7898218073f14c8d573582e8d2313dc740", GitTreeState:"clean", BuildDate:"2018-10-24T06:54:59Z", GoVersion:"go1.10.4", Compiler:"gc", Platform:"linux/amd64"}
  Server Version: version.Info{Major:"1", Minor:"12", GitVersion:"v1.12.2", GitCommit:"17c77c7898218073f14c8d573582e8d2313dc740", GitTreeState:"clean", BuildDate:"2018-10-24T06:43:59Z", GoVersion:"go1.10.4", Compiler:"gc", Platform:"linux/amd64"}
  ```

- The `kubeadm init` command should output a `kubeadm join` command containing a token and hash. Copy that command and run it with `sudo` on both worker nodes. It should look something like this:

  ```
  sudo kubeadm join $some_ip:6443 --token $some_token --discovery-token-ca-cert-hash $some_hash
  ```

- Verify that all nodes have successfully joined the cluster:

  ```
  kubectl get nodes
  ```

  You should see all three of your nodes listed. It should look something like this:

  ```
  NAME                      STATUS     ROLES    AGE     VERSION
  wboyd1c.mylabserver.com   NotReady   master   5m17s   v1.12.2
  wboyd2c.mylabserver.com   NotReady   <none>   53s     v1.12.2
  wboyd3c.mylabserver.com   NotReady   <none>   31s     v1.12.2
  ```

  Note: The nodes are expected to have a STATUS of `NotReady` at this point.

Kubenetes- Setting up networking with Flannel

Run below commands in all nodes-master & workers in cluster

```
root@bakumar2c:~# echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/
sysctl.conf
net.bridge.bridge-nf-call-iptables=1
root@bakumar2c:~# sysctl  -p
net.bridge.bridge-nf-call-iptables = 1
root@bakumar2c:~#
```

Install Flannel in the cluster by running this only on the Master node:

```
root@bakumar2c:~# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/
bc79dd1505b0c8681ece4de4c0d86c5cd2643275/Documentation/kube-flannel.yml
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.extensions/kube-flannel-ds-amd64 created
daemonset.extensions/kube-flannel-ds-arm64 created
daemonset.extensions/kube-flannel-ds-arm created
daemonset.extensions/kube-flannel-ds-ppc64le created
daemonset.extensions/kube-flannel-ds-s390x created
```

Now we setted up networking with flannel, check the nodes status.

```
root@bakumar2c:~# kubectl  get nodes
NAME                       STATUS   ROLES    AGE   VERSION
bakumar2c.mylabserver.com  Ready    master   16m   v1.12.7
bakumar3c.mylabserver.com  Ready    <none>   12m   v1.12.7
bakumar4c.mylabserver.com  Ready    <none>   12m   v1.12.7
root@bakumar2c:~#
```

Check the namespace of “kube-system”, which by default all containers are created under this.

```
root@bakumar2c:~# kubectl  get pods -n kube-system
NAME                                                READY   STATUS    RESTARTS   AGE
coredns-bb49df795-b4hzd                             1/1     Running   0          19m
coredns-bb49df795-fgj4j                             1/1     Running   0          19m
etcd-bakumar2c.mylabserver.com                      1/1     Running   0          18m
kube-apiserver-bakumar2c.mylabserver.com            1/1     Running   0          18m
kube-controller-manager-bakumar2c.mylabserver.com   1/1     Running   0          18m
kube-flannel-ds-amd64-jmdxn                         1/1     Running   0          3m9s
kube-flannel-ds-amd64-rn6xj                         1/1     Running   0          3m9s
kube-flannel-ds-amd64-smn2n                         1/1     Running   0          3m9s
kube-proxy-hxxbk                                    1/1     Running   0          19m
kube-proxy-l4n6s                                    1/1     Running   0          14m
kube-proxy-sqrkh                                    1/1     Running   0          15m
kube-scheduler-bakumar2c.mylabserver.com            1/1     Running   0          18m
```

The kube-flannel is responsible for setting up networking layer in kebernetes.

Once the Kubernetes cluster is set up, we still need to configure cluster networking in order to make the cluster fully functional. In this lesson, we will walk through the process of configuring a cluster network using Flannel. You can find more information on Flannel at the official site: https://coreos.com/flannel/docs/latest/.

Here are the commands used in this lesson:

- On all three nodes, run the following:

```
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

- Install Flannel in the cluster by running this only on the Master node:

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/bc79dd1505b0c8681ece4de4c0d86c5cd2643275/Documentation/kube-flannel.yml
```

- Verify that all the nodes now have a STATUS of `Ready`:

```
kubectl get nodes
```

You should see all three of your servers listed, and all should have a STATUS of `Ready`. It should look something like this:

```
NAME                      STATUS   ROLES    AGE     VERSION
wboyd1c.mylabserver.com   Ready    master   5m17s   v1.12.2
wboyd2c.mylabserver.com   Ready    <none>   53s     v1.12.2
wboyd3c.mylabserver.com   Ready    <none>   31s     v1.12.2
```

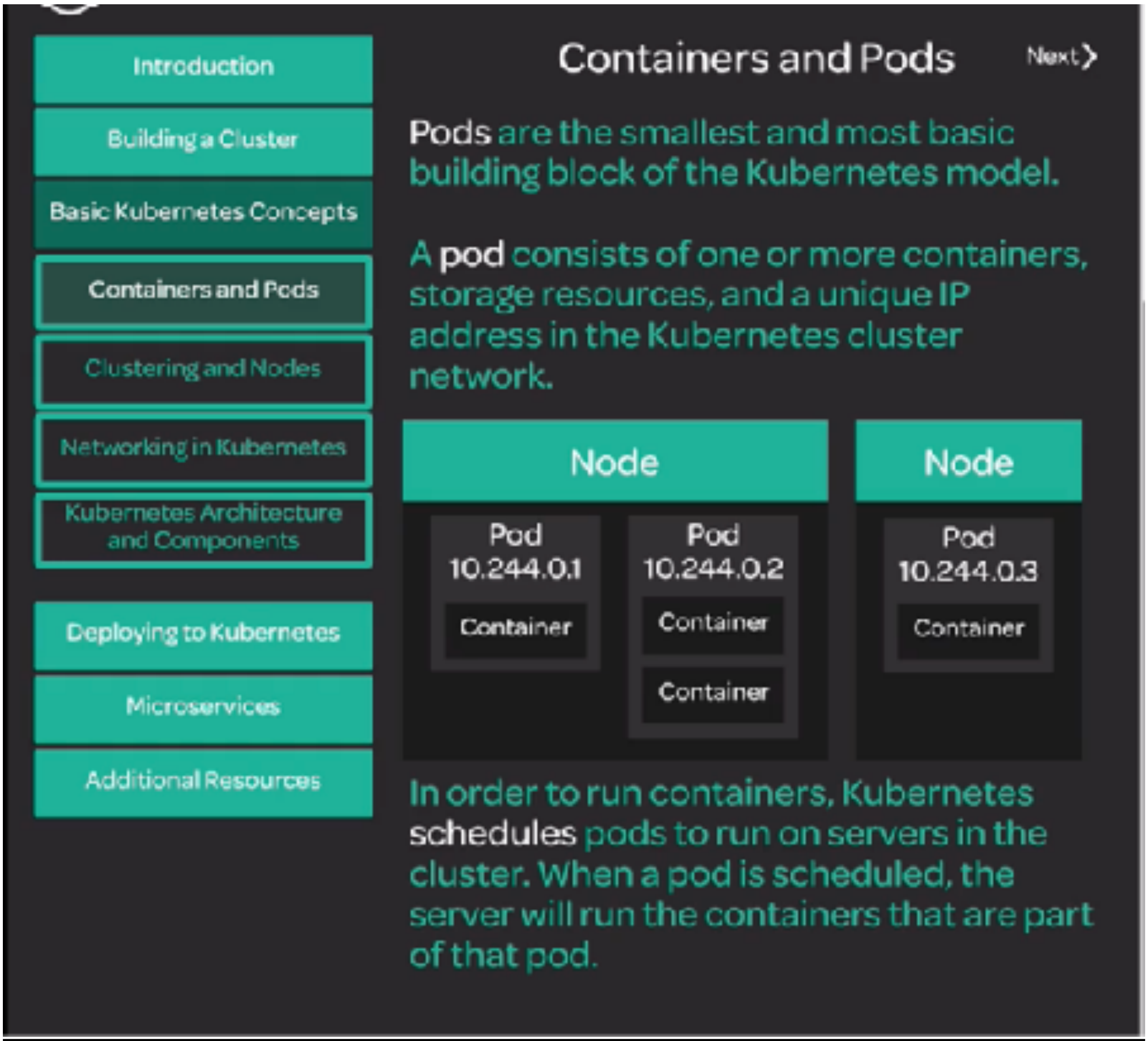
Note: It may take a few moments for all nodes to enter the `Ready` status, so if they are not all `Ready`, wait a few moments and try again.

- It is also a good idea to verify that the Flannel pods are up and running. Run this command to get a list of system pods:

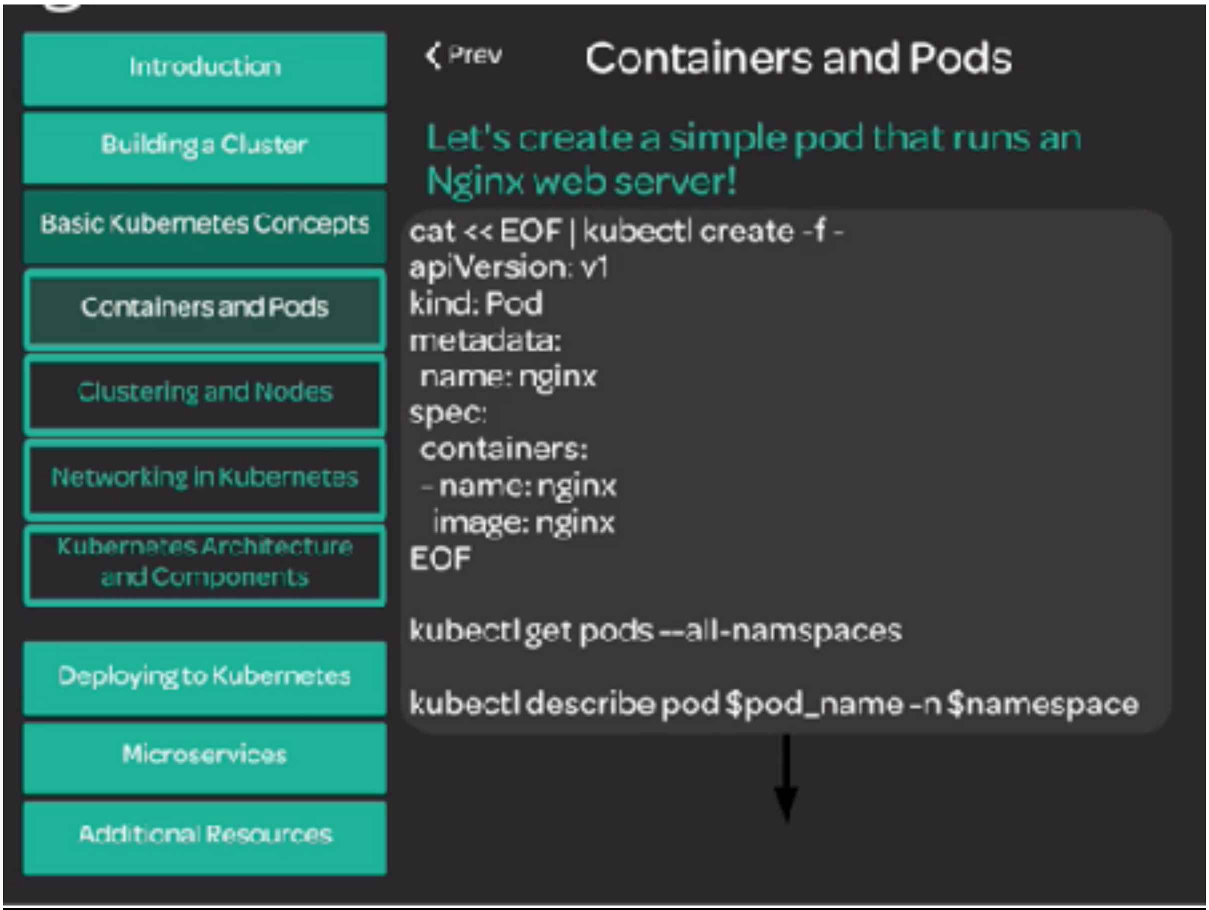
```
kubectl get pods -n kube-system
```

You should have three pods with `flannel` in the name, and all three should have a status of `Running`.

Containers and Pods



Sample YAML file to create nginx image


Introduction
Building a Cluster
Basic Kubernetes Concepts
Containers and Pods
Clustering and Nodes
Networking in Kubernetes
Kubernetes Architecture and Components
Deploying to Kubernetes
Microservices
Additional Resources
Prev
Containers and Pods
Let's create a simple pod that runs an Nginx web server!
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
name: nginx
spec:
containers:
- name: nginx
image: nginx
EOF
kubectl get pods --all-namspaces
kubectl describe pod $pod_name -n $namespace

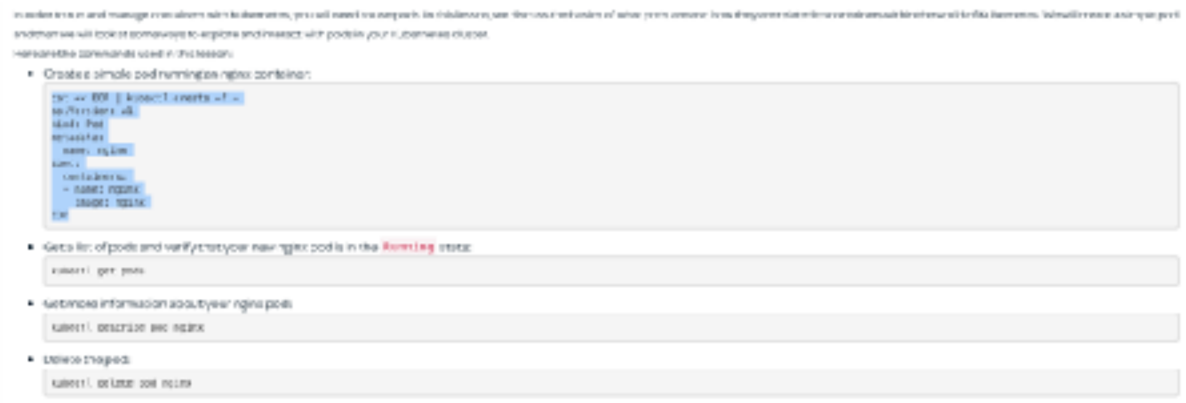
Run below in Master node to create a nginx image

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
EOF
```

```
root@bakumar2c:/opt# kubectl  get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          24s
```

more info about created pod “nginx”

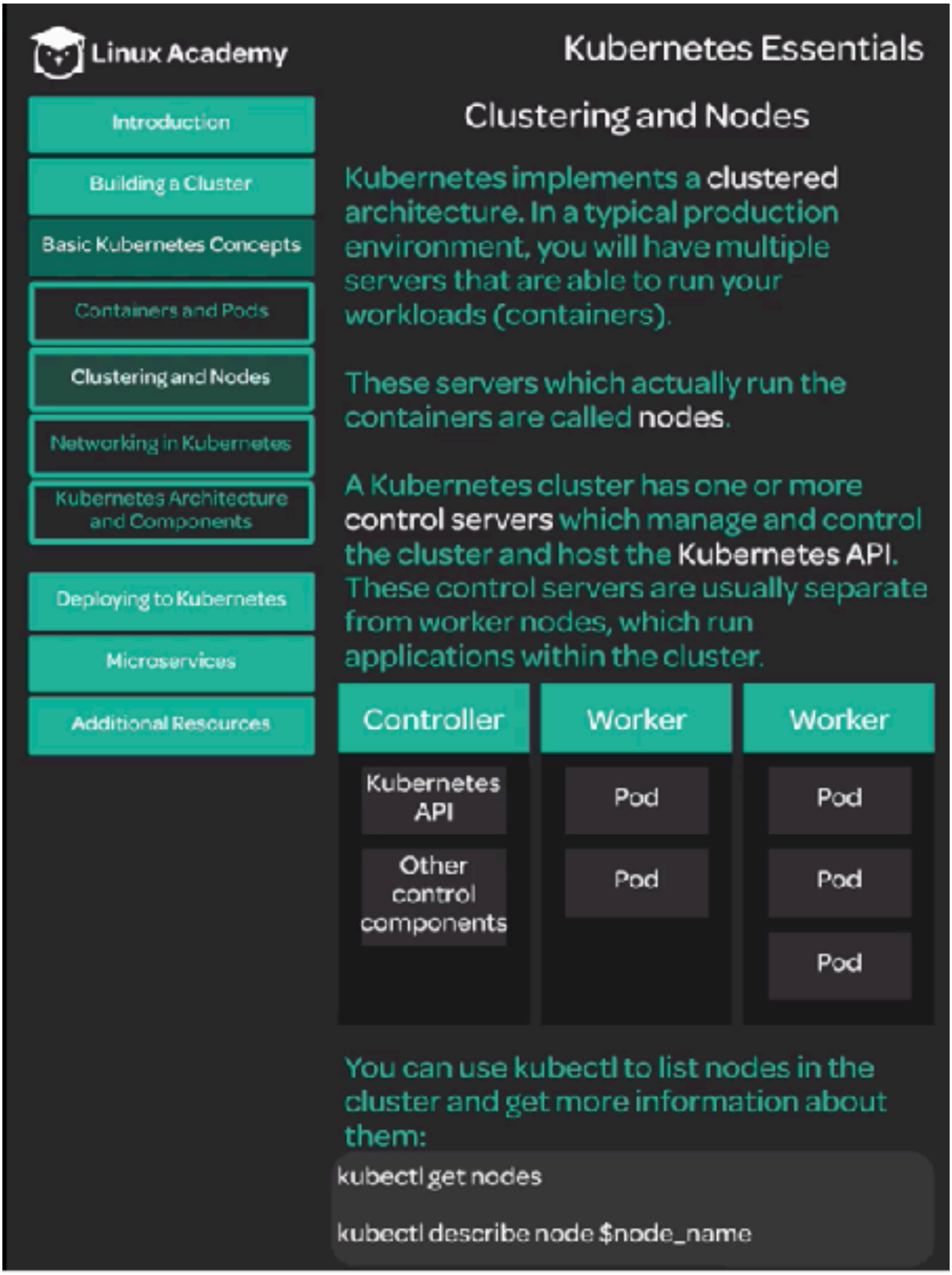
```
root@bakumar2c:/opt# kubectl  describe pod nginx
Name:               nginx
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               bakumar4c.mylabserver.com/172.31.47.144
Start Time:         Mon, 14 Oct 2019 06:46:27 +0000
Labels:             <none>
Annotations:        <none>
Status:             Running
IP:                 10.244.2.2
Containers:
  nginx:
    Container ID:   docker://13c158c33b40ac51c3091b6e76662984c214d6ab5caf3bcd040a44431b9039a0
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:aeded0f2a861747f43a01cf1018cf9efe2bdd02afd57d2b11fcc7fcadc16ccd1
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 14 Oct 2019 06:46:38 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-chmls (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-chmls:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-chmls
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age    From                                Message
  ----    ------     ----   ----                                -------
  Normal  Scheduled  4m43s  default-scheduler                   Successfully assigned default/nginx to bakumar4c.mylabserver.com
  Normal  Pulling    4m42s  kubelet, bakumar4c.mylabserver.com  pulling image "nginx"
  Normal  Pulled     4m34s  kubelet, bakumar4c.mylabserver.com  Successfully pulled image "nginx"
  Normal  Created    4m32s  kubelet, bakumar4c.mylabserver.com  Created container
  Normal  Started    4m32s  kubelet, bakumar4c.mylabserver.com  Started container
root@bakumar2c:/opt#
```



Delete pod

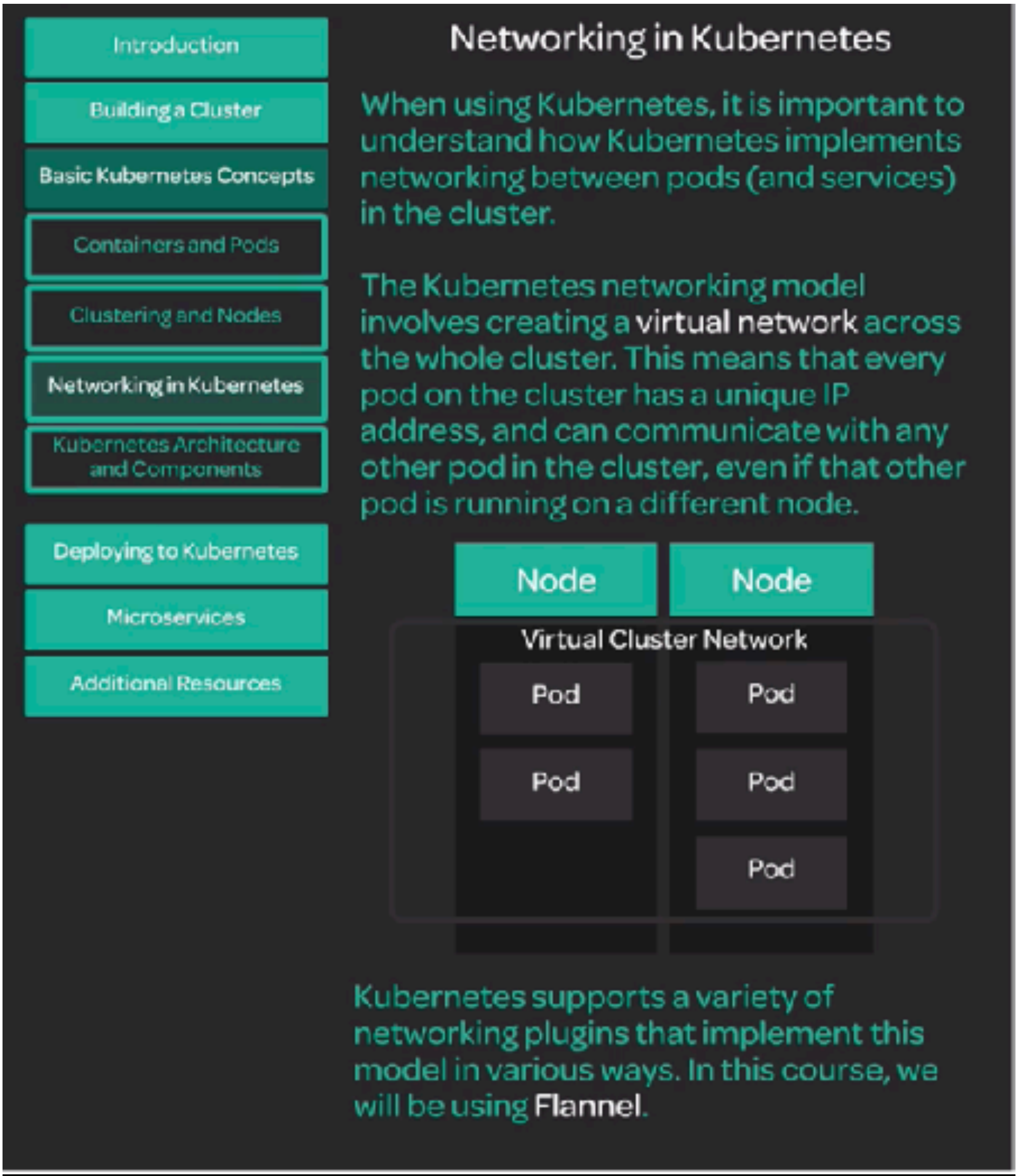
```
root@bakumar2c:/opt#
root@bakumar2c:/opt# kubectl  delete pod nginx
pod "nginx" deleted
root@bakumar2c:/opt#
```

<u>Clustering and nodes</u>



Master node -> Kubernetes Control nodes
Worker nodes are responsible for running actual applications

Networking in Kuberenetes



Run it in Master node

```
cat << EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.4
        ports:
        - containerPort: 80
EOF
```

---

root@bakumar2c:/opt# kubectl get pods

```
NAME                   READY   STATUS    RESTARTS   AGE
nginx-d55b94fd-r25rc   1/1     Running   0          73s
nginx-d55b94fd-w2rmv   1/1     Running   0          73s
```

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
  - name: busybox
    image: radial/busyboxplus:curl
    args:
    - sleep
    - "1000"
EOF
```

--

```
root@bakumar2c:/opt# kubectl   get pods
NAME                  READY   STATUS    RESTARTS   AGE
busybox               1/1     Running   0          37s
nginx-d55b94fd-r25rc  1/1     Running   0          2m41s
nginx-d55b94fd-w2rmv  1/1     Running   0          2m41s
```

we can get more wider output:

```
root@bakumar2c:/opt# kubectl   get pods -o wide
NAME                   READY   STATUS    RESTARTS   AGE     IP           NODE                        NOMINATED NODE
busybox                1/1     Running   0          70s     10.244.2.4   bakumar4c.mylabserver.com   <none>
nginx-d55b94fd-r25rc   1/1     Running   0          3m14s   10.244.1.4   bakumar3c.mylabserver.com   <none>
nginx-d55b94fd-w2rmv   1/1     Running   0          3m14s   10.244.2.3   bakumar4c.mylabserver.com   <none>
```

Now we will try to reach one Pod(Ex: Master node-busybox-10.244.2.4) to another Pod(worker node2- nginx-10.244.2.3)

Note:Here 10.244.2.3 is nginx node3 and busybox is master node1.

```
root@bakumar2c:/opt# kubectl  exec  busybox -- curl 10.244.2.3
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   612  100   612    0     0   525k      0 --:--:-- --:--:-- --:--:--  597k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
   body {
      width: 35em;
      margin: 0 auto;
      font-family: Tahoma, Verdana, Arial, sans-serif;
   }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

It is possible to interact one POD to another POD using virtual network of Kubernetes.(As shown in “Networking in Kubernees ” doc above)

Networking is an important part of understanding the basics of Kubernetes. This lesson provides a high-level overview of what a Kubernetes virtual cluster network looks like. We will also demonstrate how the network functions by connecting one pod from another pod over the virtual network.

- Create a deployment with two nginx pods:

```
cat << EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.4
        ports:
        - containerPort: 80
EOF
```

- Create a busybox pod to use for testing:

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
  - name: busybox
    image: radial/busyboxplus:curl
    args:
    - sleep
    - "1000"
EOF
```

- Get the IP addresses of your pods:

```
kubectl get pods -o wide
```

- Get the IP address of one of the nginx pods, then contact that nginx pod from the busybox pod using the nginx pod's IP address:

```
kubectl exec busybox -- curl $nginx_pod_ip
```

Kubernetes Architecture and Components

A Kubernetes cluster is made up of multiple individual components running on the various machines that are part of the cluster. In this lesson, we will briefly discuss the major Kubernetes software components and what each of them do. We will also look into how these components are actually running in our cluster currently.

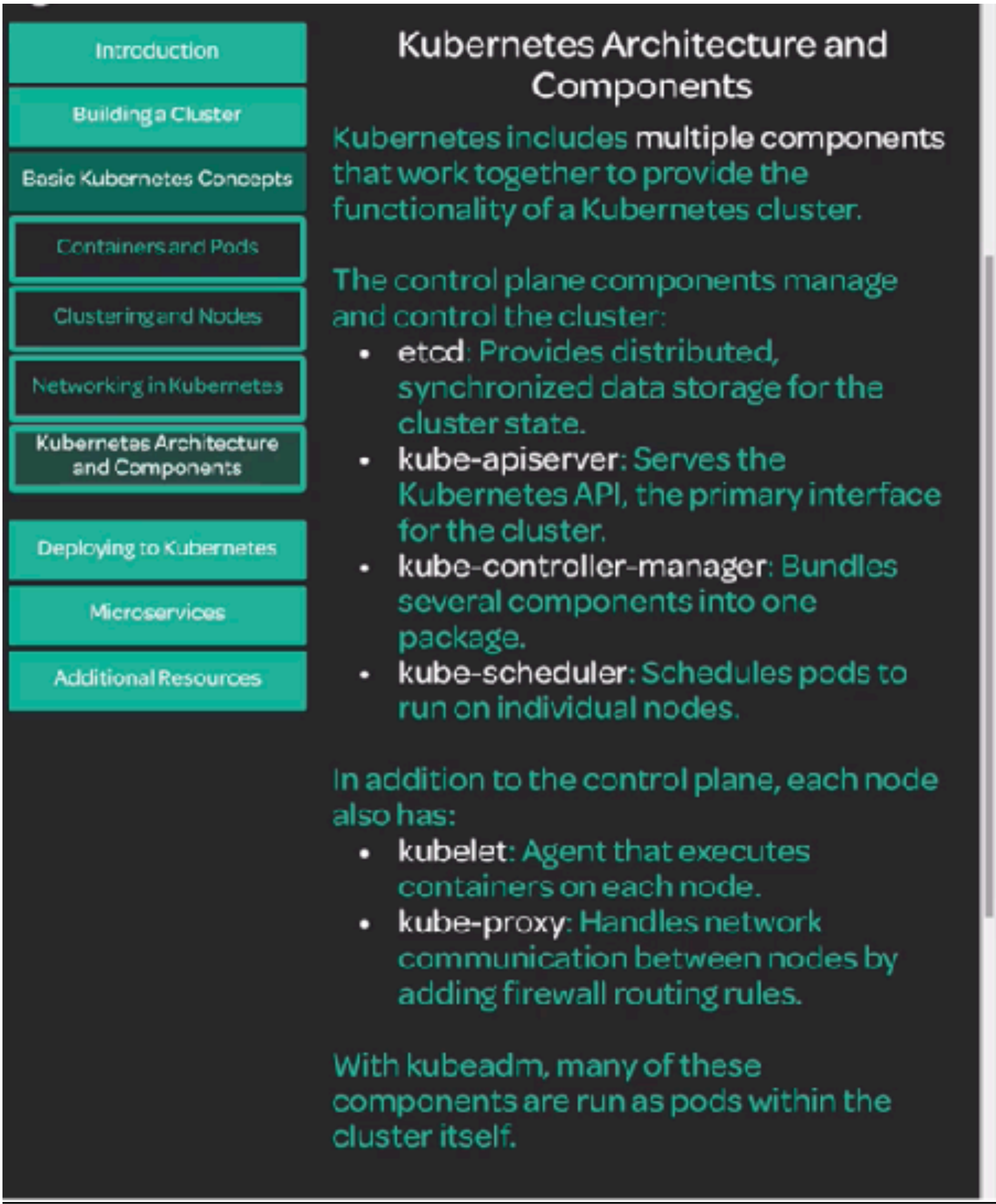
Here are the commands used in this lesson:

- Get a list of system pods running in the cluster:

```
kubectl get pods -n kube-system
```

- Check the status of the `kubelet` service:

```
sudo systemctl status kubelet
```



<u>Kubelet</u>

Its is an agent run in each kuberenetes cluster node and acts as a middle man between container(docker) and Kuberenetes api

```
root@bakumar2c:~# systemctl  status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: active (running) since Mon 2019-10-14 06:16:35 UTC; 1h 19min ago
     Docs: https://kubernetes.io/docs/home/
 Main PID: 27212 (kubelet)
    Tasks: 17 (limit: 2318)
   CGroup: /system.slice/kubelet.service
           └─27212 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/
kubelet.conf --config=/
```

deployments

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.4
        ports:
        - containerPort: 80
EOF
```

---

root@bakumar2c:~# kubectl get deployments

```
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx            2         2         2            2           39m
nginx-deployment 2         2         2            2           22s
```

root@bakumar2c:~# kubectl get deployments -o wide

```
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES         SELECTOR
nginx            2         2         2            2.          39m   nginx        nginx:1.15.4   app=nginx
nginx-deployment 2         2         2            2           27s   nginx        nginx:1.15.4   app=nginx
```

Wide description

oot@bakumar2c:~# kubectl describe deployments nginx-deployment

```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Mon, 14 Oct 2019 07:43:11 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.15.4
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-d55b94fd (2/2 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  9m54s  deployment-controller  Scaled up replica set nginx-deployment-d55b94fd to 2
root@bakumar2c:~#
```

- Introduction
- Building a Cluster
- Basic Kubernetes Concepts
  - Containers and Pods
  - Clustering and Nodes
  - Networking in Kubernetes
  - Kubernetes Architecture and Components
- Deploying to Kubernetes
- Microservices
- Additional Resources

Kubernetes Architecture and Components

Kubernetes includes **multiple components** that work together to provide the functionality of a Kubernetes cluster.

The control plane components manage and control the cluster:

- **etcd**: Provides distributed, synchronized data storage for the cluster state.
- **kube-apiserver**: Serves the Kubernetes API, the primary interface for the cluster.
- **kube-controller-manager**: Bundles several components into one package.
- **kube-scheduler**: Schedules pods to run on individual nodes.

In addition to the control plane, each node also has:

- **kubelet**: Agent that executes containers on each node.
- **kube-proxy**: Handles network communication between nodes by adding firewall routing rules.

With kubeadm, many of these components are run as pods within the cluster itself.

Introduction
Building a Cluster
Basic Kubernetes Concepts
Deploying to Kubernetes
Kubernetes Deployments
Kubernetes Services
Microservices
Additional Resources

Kubernetes Deployments

Next〉

Pods are a great way to organize and manage containers, but what if I want to spin up and automate multiple pods?

Deployments are a great way to automate the management of your pods. A deployment allows you to specify a **desired state** for a set of pods. The cluster will then constantly work to maintain that desired state.

For example:

- **Scaling**: With a deployment, you can specify the number of replicas you want, and the deployment will create (or remove) pods to meet that number of replicas.
- **Rolling Updates**: With a deployment, you can change the deployment container image to a new version of the image. The deployment will gradually replace existing containers with the new version.
- **Self-Healing**: If one of the pods in the deployment is accidentally destroyed, the deployment will immediately spin up a new one to replace it.

Linux Academy

Kubernetes Essentials

- Introduction
- Building a Cluster
- Basic Kubernetes Concepts
- Deploying to Kubernetes
- Kubernetes Deployments
- Kubernetes Services
- Microservices
- Additional Resources

‹ Prev

Kubernetes Deployments

Let's create a simple deployment!

We'll make a deployment that includes two replicas running basic Nginx containers.

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.4
        ports:
        - containerPort: 80
EOF

kubectl get deployments

kubectl describe deployment nginx-deployment

kubectl get pods
```

Deployments are an important tool if you want to take full advantage of the automation capabilities provided by Kubernetes. In this lesson, we will discuss what deployments are and briefly mention some common use cases for Kubernetes deployments. We will also create a simple deployment in our cluster and explore how we can interact with it.

Here are the commands used in this lesson:

- Create a deployment:

```
cat <<EOF | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.4
        ports:
        - containerPort: 80
EOF
```

- Get a list of deployments:

```
kubectl get deployments
```

- Get more information about a deployment:

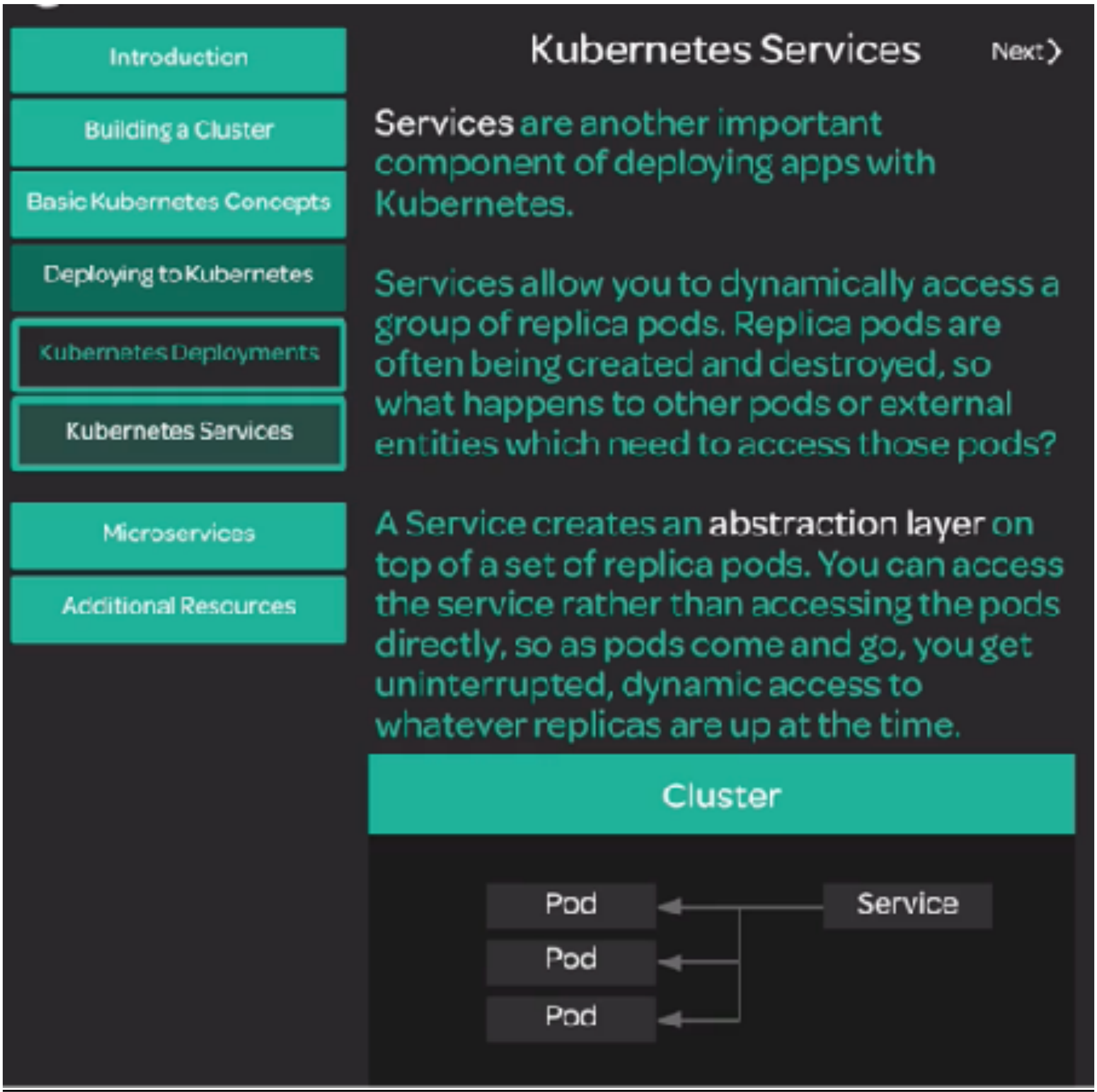
```
kubectl describe deployment nginx-deployment
```

- Get a list of pods:

```
kubectl get pods
```

You should see two pods created by the deployment.

Kuberenets services



<u>Kubenetes services</u>

While deployments provide a great way to automate the management of your pods, you need a way to easily communicate with the dynamic set of replicas managed by a deployment. This is where services come in. In this lesson, we will discuss what services are in Kubernetes, demonstrate how to create a simple service, and explore that service in our own cluster.

Here are the commands used in the demonstration:

- Create a NodePort service on top of your nginx pods:

```
cat << EOF | kubectl create -f -
kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30080
  type: NodePort
EOF
```

- Get a list of services in the cluster.

```
kubectl get svc
```

  You should see your service called `nginx-service`.

- Since this is a NodePort service, you should be able to access it using port 30080 on any of your cluster's servers. You can test this with the command:

```
curl localhost:30080
```

  You should get an HTML response from nginx!

```
root@bakumar2c:~# cat << EOF | kubectl create -f -
> kind: Service
> apiVersion: v1
> metadata:
>   name: nginx-service
> spec:
>   selector:
>     app: nginx
>   ports:
>   - protocol: TCP
>     port: 80
>     targetPort: 80
>     nodePort: 30080
>   type: NodePort
> EOF
service/nginx-service created
root@bakumar2c:~# kubectl  get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        111m
nginx-service   NodePort    10.97.179.188   <none>        80:30080/TCP   14s
root@bakumar2c:~# kubectl  get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        111m
nginx-service   NodePort    10.97.179.188   <none>        80:30080/TCP   17s
root@bakumar2c:~#
root@bakumar2c:~# curl localhost:30080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@bakumar2c:~#
```

or

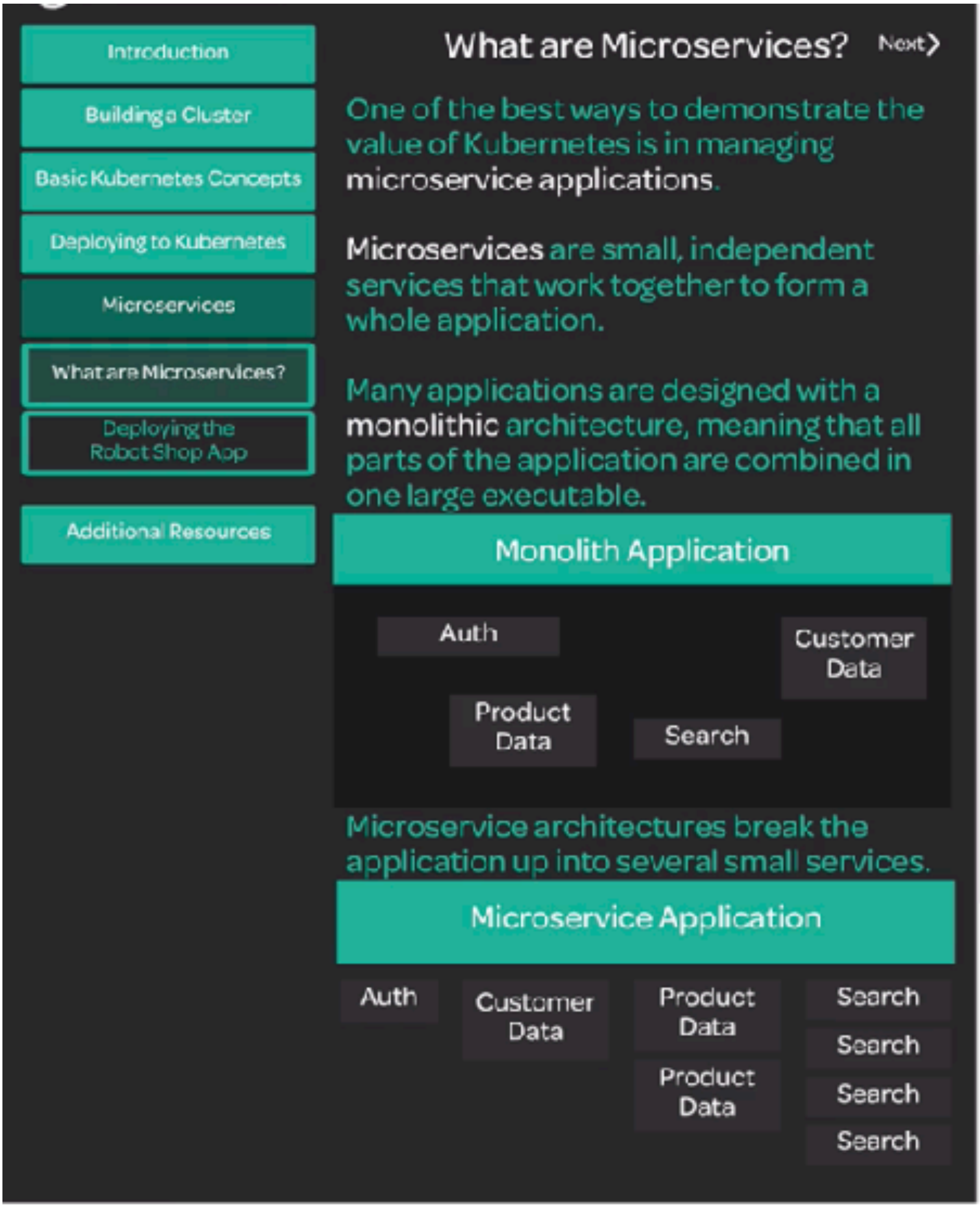
```
root@bakumar2c:~# curl 172.31.40.125:30080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<body>
<h1>Welcome to nginx!</h1>
<
```

Here "172.31.40.125" is the Master node private IP.
Similary using Private IP Addresses of worker nodes ,we can connect to pods/containers.

from workernode2

```
root@bakumar2c:~# curl 172.31.41.71:30080

<head>
<title>Welcome to nginx!</title>

<h1>Welcome to nginx!</h1>
```

Kuberenetes Microsercvices and Deployments


Introduction
Building a Cluster
Basic Kubernetes Concepts
Deploying to Kubernetes
Microservices
What are Microservices?
Deploying the Robot Shop App
Additional Resources
What are Microservices?
Next
One of the best ways to demonstrate the value of Kubernetes is in managing microservice applications.
Microservices are small, independent services that work together to form a whole application.
Many applications are designed with a monolithic architecture, meaning that all parts of the application are combined in one large executable.
Monolith Application
Auth
Customer Data
Product Data
Search
Microservice architectures break the application up into several small services.
Microservice Application
Auth
Customer Data
Product Data
Product Data
Search
Search
Search
Search

Introduction
Building a Cluster
Basic Kubernetes Concepts
Deploying to Kubernetes
Microservices
What are Microservices?
Deploying the Robot Shop App
Additional Resources

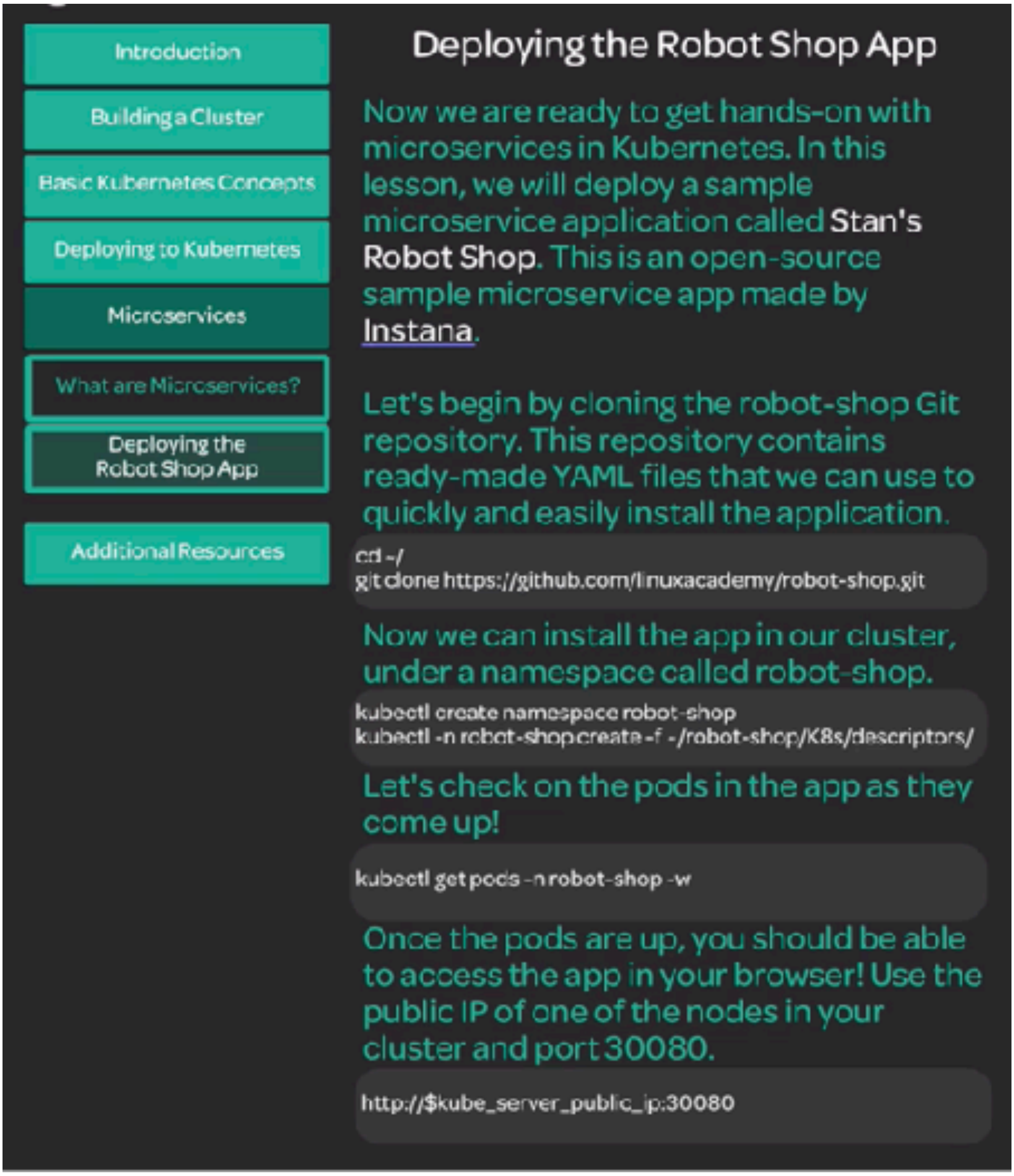
< Prev

What are Microservices?

Here are a few advantages of microservices:

- **Scalability:** Individual microservices are **independently scalable**. If your search service is under a large amount of load, you can scale that service by itself, without scaling the whole application.
- **Cleaner code**: When services are relatively independent, it is easier to make a change in one area of the application without breaking things in other areas.
- **Reliability**: Problems in one area of the application are less likely to affect other areas.
- **Variety of tools**: Different parts of the application can be built using different tools, languages, and frameworks. This means that the right tool can be used for every job!

Implementing microservices means deploying, scaling, and managing a lot of individual components! **Kubernetes** is a great tool for accomplishing all of this. In the world of microservices, the benefits of Kubernetes really shine!



http://18.140.237.207:30080/

Note: here 18.140.237.207 is the public IP address of master node.s

You can see a UI for signup and registration.

Scaling up:

Deploy the Stan's Robot Shop app to the cluster.

1. Clone the Git repo that contains the pre-made descriptors:

```
cd ~/
git clone https://github.com/linuxacademy/robot-
shop.git
```

2. Since this application has many components, it is a good idea to create a separate namespace for the app:

```
kubectl create namespace robot-shop
```

3. Deploy the app to the cluster:

```
kubectl -n robot-shop create -f ~/robot-
shop/K8s/descriptors/
```

4. Check the status of the application's pods:

```
kubectl get pods -n robot-shop
```

5. You should be able to reach the robot shop app from your browser using the Kube master node's public IP:
`http://$kube_master_public_ip:30080`

Scale up the MongoDB deployment to two replicas instead of just one.

1. Edit the deployment descriptor:

```
kubectl edit deployment mongodb -n robot-shop
```

2. You should see some YAML describing the deployment object.
   - Under `spec:`, look for the line that says `replicas: 1` and change it to `replicas: 2`.
   - Save and exit.
3. Check the status of the deployment with:

```
kubectl get deployment mongodb -n robot-shop
```

After a few moments, the number of available replicas should be 2.

Deploying a Simple Service to Kubernetes

Create a deployment for the store-products service with four replicas.

```
cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: store-products
  labels:
    app: store-products
spec:
  replicas: 4
  selector:
    matchLabels:
      app: store-products
  template:
    metadata:
      labels:
        app: store-products
    spec:
      containers:
      - name: store-products
        image: linuxacademycontent/store-products:1.0.0
        ports:
        - containerPort: 80
EOF
```

Create a store-products service and verify that you can access it from the busybox testing pod.

```
cat << EOF | kubectl apply -f -
kind: Service
apiVersion: v1
metadata:
  name: store-products
spec:
  selector:
    app: store-products
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
EOF
```

```
root@bakumar2c:~# kubectl  get services
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP   3h34m
store-products ClusterIP   10.110.241.128   <none>        80/TCP    6m46s
root@bakumar2c:~# kubectl  get deployments
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx             2         2         2            2           166m
nginx-deployment  2         2         2            2           128m
store-products    4         4         4            4           7m48s



root@bakumar2c:~# kubectl  exec busybox -- curl -s  store-products
{
	"Products":[
		{
			"Name":"Apple",
			"Price":1000.00,
		},
		{
			"Name":"Banana",
			"Price":5.00,
		},
		{
			"Name":"Orange",
			"Price":1.00,
		},
		{
			"Name":"Pear",
			"Price":0.50,
		}
	]
}
```