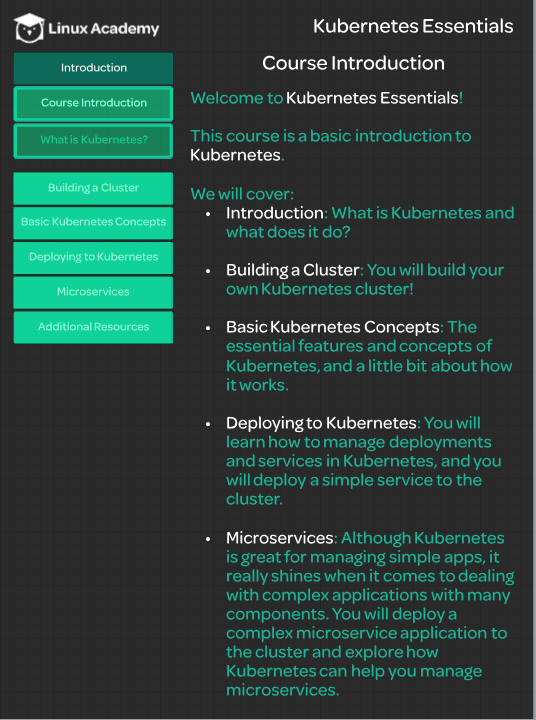
## What is Kubernetes?

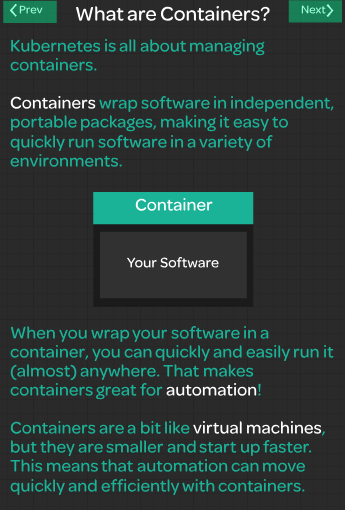
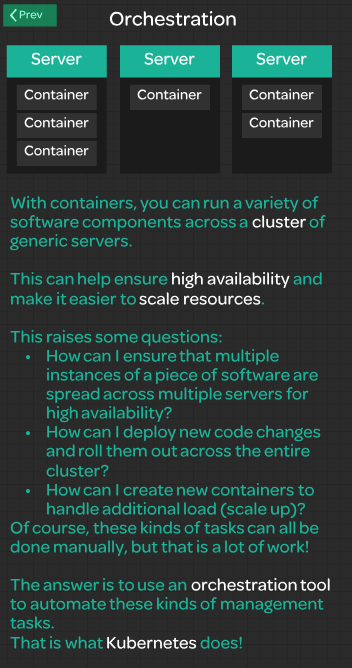
In order to proceed further in understanding how to install and use Kubernetes, it is important to have some foundational knowledge about Kubernetes and containers. This lesson discusses Kubernetes from a high-level perspective. After completing this lesson, you will have a general idea of what containers are, as well as how container orchestration tools such as Kubernetes can help you take full advantage of containers.

Check out the official Kubernetes site for documentation and more info!

[http://kubernetes.io](http://kubernetes.io/)



## 

## 

## Installing Docker

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

## Installing kubeadm, kubelet, kubectl

## 

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 the Cluster

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.

## Configuring Networking with Flannel

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 & Pods

In order to run and manage containers with Kubernetes, you will need to use pods. In this lesson, we discuss the basics of what pods are and how they are related to containers within the world of Kubernetes. We will create a simple pod and then we will look at some ways to explore and interact with pods in your Kubernetes cluster.

Here are the commands used in this lesson:

* Create a simple pod running an nginx container:
* cat << EOF | kubectl create -f -
* apiVersion: v1
* kind: Pod
* metadata:
* name: nginx
* spec:
* containers:
* - name: nginx
* image: nginx
* EOF
* Get a list of pods and verify that your new nginx pod is in the Running state:
* kubectl get pods
* Get more information about your nginx pod:
* kubectl describe pod nginx
* Delete the pod:

kubectl delete pod nginx

## Clustering and Nodes

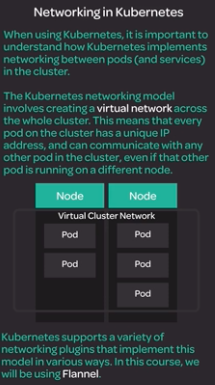


Nodes are an essential part of the Kubernetes cluster. They are the machines where your cluster's container workloads are executed. In this lesson, we will discuss what nodes are in Kubernetes, and we will explore some ways in which you can find information about nodes in your cluster.

Here are the commands used in this lesson:

* Get a list of nodes:
* kubectl get nodes
* Get more information about a specific node:
* kubectl describe node $node\_name

## Networking in Kubernetes



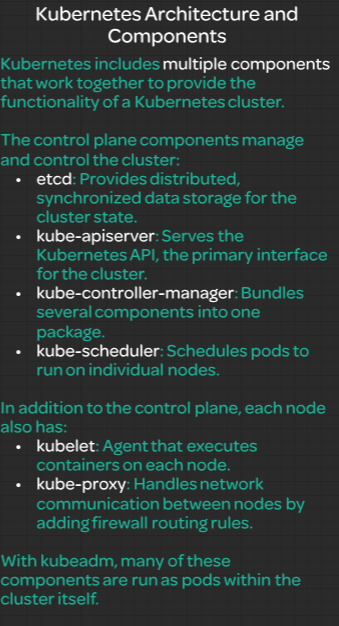
Virtual Cluster Network is very similar to Amazon VPC – Virtual Private Cloud

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 contacting 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

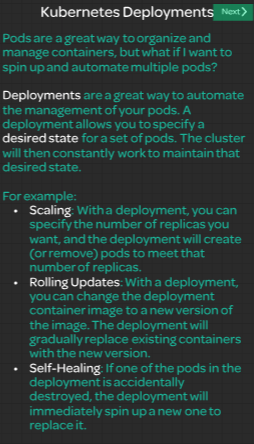


A Kubernetes cluster is made up of multiple individual components running on the various machines that are part of the cluster. In this lesson, we will briefly discuss the major Kubernetes software components and what each of them do. We will also look into how these components are actually running in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

## Kubernetes Deployments



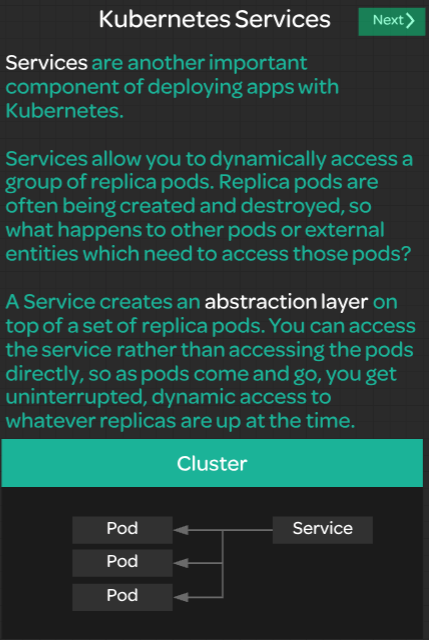
Deployments are an important tool if you want to take full advantage of the automation capabilities provided by Kubernetes. In this lesson, we will discuss what deployments are and briefly mention some common use cases for Kubernetes deployments. We will also create a simple deployment in our cluster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

## Kubernetes Services



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!

## Microservices

Microservices provide many benefits in the design and management of applications, but they also introduce a lot of complexity. In this lesson, we will talk about what microservices are and why so many people today are excited about them. We will also talk about the relationship between microservices and Kubernetes, and how Kubernetes can help manage the additional complexity that microservices bring.

