Scott Lowe: Hi, I'm Scott Lowe, and I'm a staff architect at VMware. In this video, part two of our series on cluster API, we're going to use cluster API to provision a cluster in AWS. Now if you haven't watched the first cluster API video, which covers concepts, components, and terminology, I recommend that you review that first before watching this video. Otherwise, this video uses some terminology that you may not be familiar with, and it may be confusing to you.

Scott Lowe: All right, let's get started with our demo here. We're just going to log in to an EC2 instance that we have. On this EC2 instance I have Kubernetes running, so if I do a, get nodes, you see that I have them running. This is a cluster. This is a management cluster, and I'm going to show how this has been cluster API enabled. I'll do a, get clusters, recall from the first video on cluster API that the cluster API uses custom resource definitions. A cluster is one of the custom resource definitions that a cluster API uses.

Scott Lowe: You can see I have an existing workload cluster provisioned. The fact that you didn't get an error when I ran this command means that this is a management cluster. It has been provisioned for cluster API, so it has the CRDs and the controllers installed. In fact, this is the same cluster that we used in the first video. We have a workload cluster provisioned, and if I flip over here to my EC2 console, I can go to instances, and you'll see that I have some instances here that are tagged capi-quickstart, which correspond to that cluster that I just showed you there.

Scott Lowe: Okay, so let's set up our own cluster. We're going to twitch into the demo directory, and in here you can see I have some yaml files. The first thing to do is to actually create the cluster itself, which will create the underlying AWS infrastructure. To do that, we're going to use this cluster yaml. Let's look at that real quick. You'll see here that it specifies configuration information about the cluster, the name of the cluster, the site or subnet that the pods should use, the region, so on, so forth.

Scott Lowe: Okay, so let's just do a kubectl apply, and boom. We'll see that it says, okay, I've created a cluster and then I've created awscluster in .infrastructure. This, these two objects are matched. The cluster object is for a cluster API, and then the AWS cluster object is for the cluster API provider for AWS. Recall from the first video, that cluster API uses providers to split platform specific information from the main cluster API. So that's how we can use cluster API on AWS or on vSphere or on Azure.

Scott Lowe: Now we're going to flip over to our console window again, and we're going to go here to the VPC dashboard. Let's refresh that. If you caught that, it went from three to four right there. Usually NAT gateways take the longest, so let's check on the status of the NAT gateway. We can see that it's pending. Once that goes available then we'll be able to proceed with the rest of creating the cluster.

Scott Lowe: While we're waiting on that, let's take a look at the other piece we have here for the control plane. Now this yaml defines the node that will be running in the cluster that has the control plane components installed. We use the kind machine here, which is another one of cluster APIs custom resource definitions. We have some associated

information, we link it with a label to the name of the demo that we specified... The name of the cluster rather, demo, that we specified in the cluster yaml.

Scott Lowe: Then you can see we have some infrastructure references, some config references here, that tell it how to bootstrap the node using Kubeadm, and then how to link this node object to the underlying EC2 instance, which is represented by this CRD AWSMachine. Then we define that down here, we give it a instance type, we give it a key name. Here is the bootstrap configuration that it should be using, and we have provided some Kubeadm configuration flags to enable the AWS cloud provider.

Scott Lowe: Now let's flip back over and see how our NAT gateway is doing. Okay. NAT gateway shows available now, and if we look at other objects, we'll see that we have some subnets that are created for demo. We have an internet gateway created for the demo environment. We have a NAT gateway that we just saw created for the demo environment. All of our infrastructure has been created, and we're now ready to go ahead and create the control plane. So let's create the control plane again with Kubectl.

Scott Lowe: This again, you see that it creates three objects here. The machine object, the AWSMachine object which represents the instance on the back end, and then the Kubeadm object, which represents how to bootstrap that node into the cluster. In the background, what's happening right now is the cluster API provider for AWS is speaking to the AWS API, and creating the machine.

Scott Lowe: If we flip over here to our console, we will see that we'll see a new machine running. Here we go demo-cp-zero. It's already up and running, it's still initializing. You'll notice also that it created a machine called demo-bastion. This part will probably be dropped in future revisions of cluster API, or at least made optional. But right now it automatically creates an SSH bastion host for improved security of the environment.

Scott Lowe: All right. Let's, while we're doing that, while we're waiting on that machine to come up, let's go ahead and pull the kubeconfig for the new cluster that we just created. To do that we're going to do a, kubectl get secrets, and you're going to see some secrets here. There's a secret that has the name of the cluster demo-kubeconfig. We're going to get that and then we're going to extract some information from it. We'll do a, get secret demo-kubeconfig. We're going to pipe this into a command called JQ, which is a JSON parser, to pull out only the value of the secret. Then we're going to decode it because it's base 64 encoded inside there. Then we're going to write it to a file called demo.kubeconfig.

Scott Lowe: And we get an error. Okay, so, oh that's right. Okay, I made a mistake here. We have to specify output as JSON, forgot to do that. Otherwise JQ doesn't know how to parse the output. There we go. All right. And now we have kubeconfig. Let's see what the status of our machine, our new cluster is. Kubeconfig=./demo get nodes. Okay.

Scott Lowe: All right, so we see the new node is up, it shows not ready yet. That's expected cause we haven't installed a CNI. So let's go ahead and install the CNI. We'll do, I have a prepared yaml file for Calico to install Calico into this cluster. Then let's go back and check our

nodes again, and it will take a couple of minutes while this workload cluster spins up the Calico pods. Once it's showing ready, then we will proceed with creating our worker node.

Scott Lowe: while we're waiting on that, let's look at the definition for the worker node. In this case we're using something called a machine deployment, which is like a deployment for pods, but instead for machines. We're going to give it, as you can see down here, a list of replicas. We're only using one in this case, but we could say replicas five or replicas ten, or whatever. Then cluster API will make sure that it's always running ten systems, or whatever this number of replicas is, much like a deployment would ensure that there's a certain number of pods always running.

Scott Lowe: All right, let's get out of here. Let's check our cluster. Okay. It shows ready. Great, so we'll do a, kubectl apply. Notice we're not specifying the kubeconfig here because in this case we want to speak to the management cluster. We don't want to speak to the workload cluster. We're going to apply the yaml. Notice it creates again the objects, the machine deployment, the AWSMachine template, which is how it creates configurations for AWSMachine objects. And then a kubeadm config template, which is how it creates kubeadm configs for the individual machines.

Scott Lowe: If we go back over to our AWS console and we refresh this, we will see a demo worker right here, showing up that it's running. Now I can go back and run the get nodes command again. We are specifying the kubeconfig here, because we want to talk to the workload cluster. We're asking the workload cluster, show me your nodes, right? So we do a get nodes, it's not ready yet, so it's still bootstrapping. We'll give this a couple more minutes. What we should see in a moment then, is that it will spin up this machine and bootstrap it using kubeadm.

Scott Lowe: Okay. We've given it some time now, and on the back end it has spun up the instance and bootstrapped it. Let's run our command again to do the get nodes. We see now that it shows that it is ready. We could also do, just to show you that this thing actually does have pods running, we could again specify the demo-kubeconfig. So we're talking to the workload cluster. We'll say, show us the cube system namespace, and let's get pods from the cube system namespace. Here you show the pods that are running in the cube system namespace on the workload cluster that we provisioned automatically with cluster API.

Scott Lowe: So that gives you an overview of how to use cluster API to create a cluster on AWS. You saw that we didn't really have to run any AWS specific commands. Everything was encoded in the cluster API yaml, and we used the management cluster to automatically spawn the infrastructure and configure that infrastructure to support a Kubernetes cluster. That covers cluster API in this video, thanks for watching.