Cloud Provider:AWS
Server:Ubuntu 18.04 x6
Server List: 2 Controller(Master), 2(Worker), 1 Nginx(Reverse Proxy), 1 Ubuntu 18.04(local)

Smoke Testing the Cluster

We will test the following features:

- Data encryption
- Deployments
- Port forwarding
- Logs
- Exec
- Services
- Untrusted workloads

Smoke Testing Data Encryption

Goal: Verify that we can encrypt secret data at rest.
Strategy:

- Create a generic secret in the cluster.
- Dump the raw data from etcd and verify that it is encrypted.

// Earlier in this course, we set up a data encryption config to allow Kubernetes to encrypt sensitive data. In this lesson, we will smoke test that functionality by creating some secret data and verifying that it is stored in an encrypted format in etcd. After completing this lesson, you will have verified that your cluster can successfully encrypt sensitive data.

// For this lesson, you will need to connect to the cluster using kubectl . You can log in to one of your controller servers and use kubectl there, or you can use kubectl from your local machine. To use kubectl from your local machine, you will need to open an SSH tunnel. You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

```
$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>
```

// Create a test secret:

```
$ kubectl create secret generic kubernetes-the-hard-way --from-literal="mykey=mydata"
```

// Log in to one of your controller servers, and get the raw data for the test secret from etcd:

```
$ sudo ETCDCTL_API=3 etcdctl get \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/etcd/ca.pem \
  --cert=/etc/etcd/kubernetes.pem \
  --key=/etc/etcd/kubernetes-key.pem\
  /registry/secrets/default/kubernetes-the-hard-way | hexdump -C
```

// Your output should look something like this:

```
00000000 2f 72 65 67 69 73 74 72 79 2f 73 65 63 72 65 74 |/registry/secret|
00000010 73 2f 64 65 66 61 75 6c 74 2f 6b 75 62 65 72 6e |s/default/kubern|
00000020 65 74 65 73 2d 74 68 65 2d 68 61 72 64 2d 77 61 |etes-the-hard-wa|
00000030 79 0a 6b 38 73 3a 65 6e 63 3a 61 65 73 63 62 63 |y.k8s:enc:aescbc|
00000040 3a 76 31 3a 6b 65 79 31 3a fc 21 ee dc e5 84 8a |:v1:key1:.!.....|
00000050 53 8e fd a9 72 a8 75 25 65 30 55 0e 72 43 1f 20 |S...r.u%e0U.rC. |
00000060 9f 07 15 4f 69 8a 79 a4 70 62 e9 ab f9 14 93 2e |...Oi.y.pb......|
00000070 e5 59 3f ab a7 b2 d8 d6 05 84 84 aa c3 6f 8d 5c |.Y?..........o.\|
00000080 09 7a 2f 82 81 b5 d5 ec ba c7 23 34 46 d9 43 02 |.z/.......#4F.C.|
00000090 88 93 57 26 66 da 4e 8e 5c 24 44 6e 3e ec 9c 8e |..W&f.N.\$Dn>...|
000000a0 83 ff 40 9a fb 94 07 3c 08 52 0e 77 50 81 c9 d0 |..@....<.R.wP...|
000000b0 b7 30 68 ba b1 b3 26 eb b1 9f 3f f1 d7 76 86 09 |.0h...&...?..v..|
000000c0 d8 14 02 12 09 30 b0 60 b2 ad dc bb cf f5 77 e0 |.....0.`......w.|
000000d0 4f 0b 1f 74 79 c1 e7 20 1d 32 b2 68 01 19 93 fc |O..ty.. .2.h....|
000000e0 f5 c8 8b 0b 16 7b 4f c2 6a 0a |.....{O.j.|
000000ea
```

// Look for k8s:enc:aescbc:v1:key1 on the right of the output to verify that the data is stored in an encrypted format.

Smoke Testing Deployments

Goal: Verify that we can create a deployment and that it can successfully create pods
Strategy:

- Create a simple deployment.
- Verify that the deployment successfully creates a pod.

// Deployments are one of the powerful orchestration tools offered by Kubernetes. In this lesson, we will make sure that deployments are working in our cluster. We will verify that we can create a deployment, and that the deployment is able to successfully stand up a new pod and container.

// For this lesson, you will need to connect to the cluster using kubectl . You can log in to one of your controller servers and use kubectl there, or you can use kubectl from your local machine. To use kubectl from your local machine, you will need to open an SSH tunnel. You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

```
$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>
```

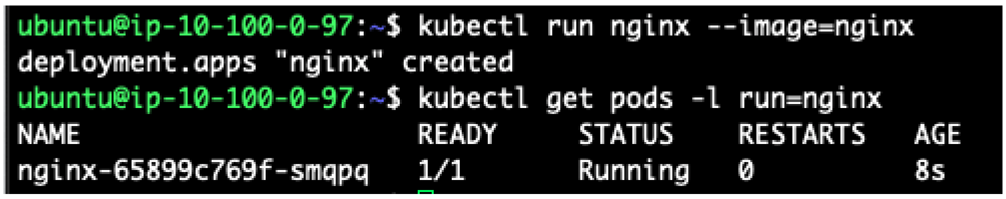
// Create a simple nginx deployment:

```
$ kubectl run nginx --image=nginx
```

// Verify that the deployment created a pod and that the pod is running:

$ kubectl get pods -l run=nginx

// Verify that the output looks something like this:



```
ubuntu@ip-10-100-0-97:~$ kubectl run nginx --image=nginx
deployment.apps "nginx" created
ubuntu@ip-10-100-0-97:~$ kubectl get pods -l run=nginx
NAME                     READY     STATUS    RESTARTS   AGE
nginx-65899c769f-smqpq   1/1       Running   0          8s
```

Smoke Testing Port Forwarding

Goal: Verify that we can use port forwarding to access pods remotely.
Strategy:

- Use kubectl port-forward to set up port forwarding for an Nginx pod.
- Access the pod remotely with curl.

// One way to get direct access to pods is the kubectl port-forward command. This command can be very useful in troubleshooting scenarios, so we want to make sure it works in our cluster. In this lesson, we will smoke test port forwarding in our cluster. We will set up port forwarding with kubectl to a pod in the cluster, and we will access the pod to verify that it is working.

// For this lesson, you will need to connect to your cluster using kubectl . You can log in to one of your controller servers and use kubectl there, or you can use kubectl from your local machine. To use kubectl from your local machine, you will need to open an SSH tunnel. You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>

// First, get the pod name of the nginx pod and store it an an environment variable:

$ POD_NAME=$(kubectl get pods -l run=nginx -o jsonpath="{.items[0].metadata.name}")

// Forward port 8081 to the nginx pod:

$ kubectl port-forward $POD_NAME 8081:80

// Open up a new terminal on the same machine running the kubectl port-forward command and verify that the port forward works.

$ curl --head http://127.0.0.1:8081

// You should get an http 200 OK response from the nginx pod.
// When you are done, you can stop the port-forward in the original terminal with control+c .

Smoke Testing Logs

Goal: Verify that we can get container logs with kubectl logs.
Strategy:

- Get the logs from the Nginx pod container

// When managing a cluster, it is often necessary to access container logs to check their health and diagnose issues. Kubernetes offers access to container logs via the kubectl logs command. In this lesson, we will smoke test our cluster's ability to provide logs with kubectl logs . In order to do this, we will access the logs from our nginx container.

// For this lesson, you will need to connect to the cluster using kubectl . You can log in to one of your controller servers and use kubectl there, or you can run it from your local machine. To use kubectl from your local machine, you will need to open an SSH tunnel. You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

```
$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>
```

// First, let's set an environment variable to the name of the nginx pod:

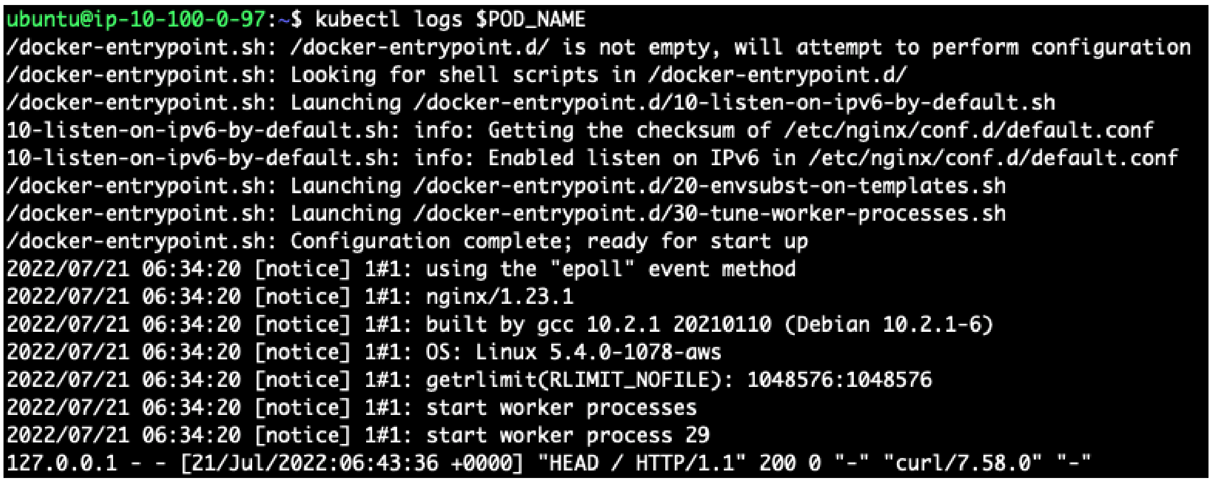
```
$ POD_NAME=$(kubectl get pods -l run=nginx -o jsonpath="{.items[0].metadata.name}")
```

// Get the logs from the nginx pod:

```
$ kubectl logs $POD_NAME
```

// This command should return the nginx pod's logs. It will look something like this (but there could be more lines):

```
127.0.0.1 - - [10/Sep/2018:19:29:01 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.47.0" "-"
```



```
ubuntu@ip-10-100-0-97:~$ kubectl logs $POD_NAME
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/07/21 06:34:20 [notice] 1#1: using the "epoll" event method
2022/07/21 06:34:20 [notice] 1#1: nginx/1.23.1
2022/07/21 06:34:20 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/07/21 06:34:20 [notice] 1#1: OS: Linux 5.4.0-1078-aws
2022/07/21 06:34:20 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/07/21 06:34:20 [notice] 1#1: start worker processes
2022/07/21 06:34:20 [notice] 1#1: start worker process 29
127.0.0.1 - - [21/Jul/2022:06:43:36 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/7.58.0" "-"
```

Smoke Testing Exec

Goal: Verify that we can run commands in a container with kubectl exec.
Strategy:

- Use kubectl exec to run a command in the Nginx pod container.

// The kubectl exec command is a powerful management tool that allows us to run commands inside of Kubernetes-managed containers. In order to verify that our cluster is set up correctly, we need to make sure that kubectl exec is working. In this lesson, we will smoke test kubectl exec against our cluster by using it to run a simple command inside our nginx container. If this works, then we know that our cluster is capable of running commands inside of its containers.

// For this lesson, you will need to connect to the cluster using kubectl . You can log in to one of your controller servers and use kubectl there, or you can use kubectl from your local machine. To use kubectl from your local machine, you will need to open an SSH tunnel. You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

```
$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>
```

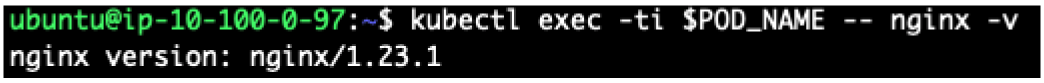
// First, let's set an environment variable to the name of the nginx pod:

```
$ POD_NAME=$(kubectl get pods -l run=nginx -o jsonpath="{.items[0].metadata.name}")
```

// To test kubectl exec , execute a simple nginx -v command inside the nginx pod:

```
$ kubectl exec -ti $POD_NAME -- nginx -v
```

// This command should return the nginx version output, which should look like this:



```
ubuntu@ip-10-100-0-97:~$ kubectl exec -ti $POD_NAME -- nginx -v
nginx version: nginx/1.23.1
```

Smoke Testing Services

Goal: Verify that we can create and access services.
Strategy:

- Create a NodePort service to expose the Nginx deployment.
- Access the service remotely using NodePort.

// In order to make sure that the cluster is set up correctly, we need to ensure that services can be created and accessed appropriately. In this lesson, we will smoke test our cluster's ability to create and access services by creating a simple testing service, and accessing it using a node port. If we can successfully create the service and use it to access our nginx pod, then we will know that our cluster is able to correctly handle services!

// For this lesson, you will need to connect to the cluster using kubectl . You can log in to one of your controller servers and use it there, or you can use it from your local

machine. To use kubectl from your local machine, you will need to open an SSH tunnel.

// You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

```
$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>
```

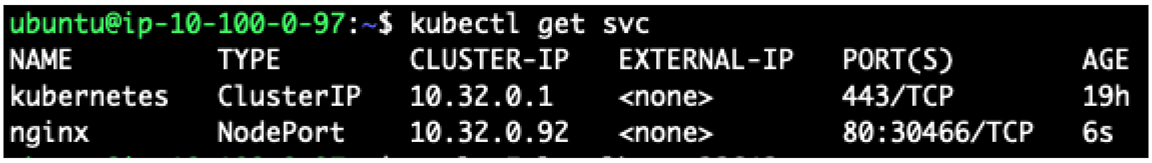
// First, create a service to expose the nginx deployment:

```
$ kubectl expose deployment nginx --port 80 --type NodePort
```

// Get the node port assigned to the newly-created service and assign it to an environment variable:

```
$ kubectl get svc
```

// The output should look something like this:



```
ubuntu@ip-10-100-0-97:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.32.0.1    <none>        443/TCP        19h
nginx        NodePort    10.32.0.92   <none>        80:30466/TCP   6s
```

// Look for the service called nginx in that output. Under PORT(S) , look for the second port, listed after 80: . In the example above, it is 32642. That is the node port, so make note of that value since you will need it in a moment.

// Next, log in to one of your worker servers and make a request to the service using the node port. Be sure to replace the placeholder with the actual node port:

```
$ curl -I localhost:<node port>
```

// You should get an http 200 OK response.

// Now that we have finished smoke testing the cluster, it is a good idea to clean up the objects that we created for testing. In this lesson, we will spend a moment removing the objects that were created in our cluster in order to perform the smoke testing.

// For this lesson, you will need to connect to the cluster using kubectl . You can log in to one of your controller servers and use it there, or you can use it from your local machine. To use kubectl from your local machine, you will need to open an SSH tunnel. You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

```
$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>
```

// You can clean up the objects that were created for smoke testing purposes like so:

```
$ kubectl delete secret kubernetes-the-hard-way
$ kubectl delete svc nginx
$ kubectl delete deployment nginx
$ kubectl delete pod untrusted
$ kubectl get pods
```