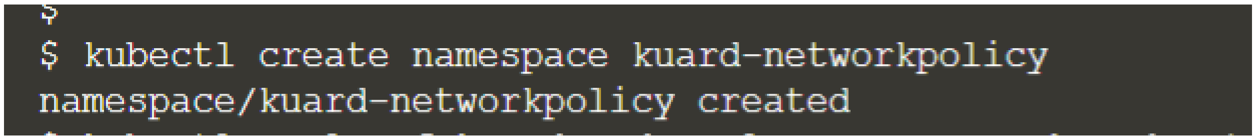
Kubernetes has a NetworkPolicy API that allows you to create both ingress and egress network policies for your workload. Network policies are configured using labels, which allow you to select specific Pods and define how they can communicate with other Pods and endpoints.

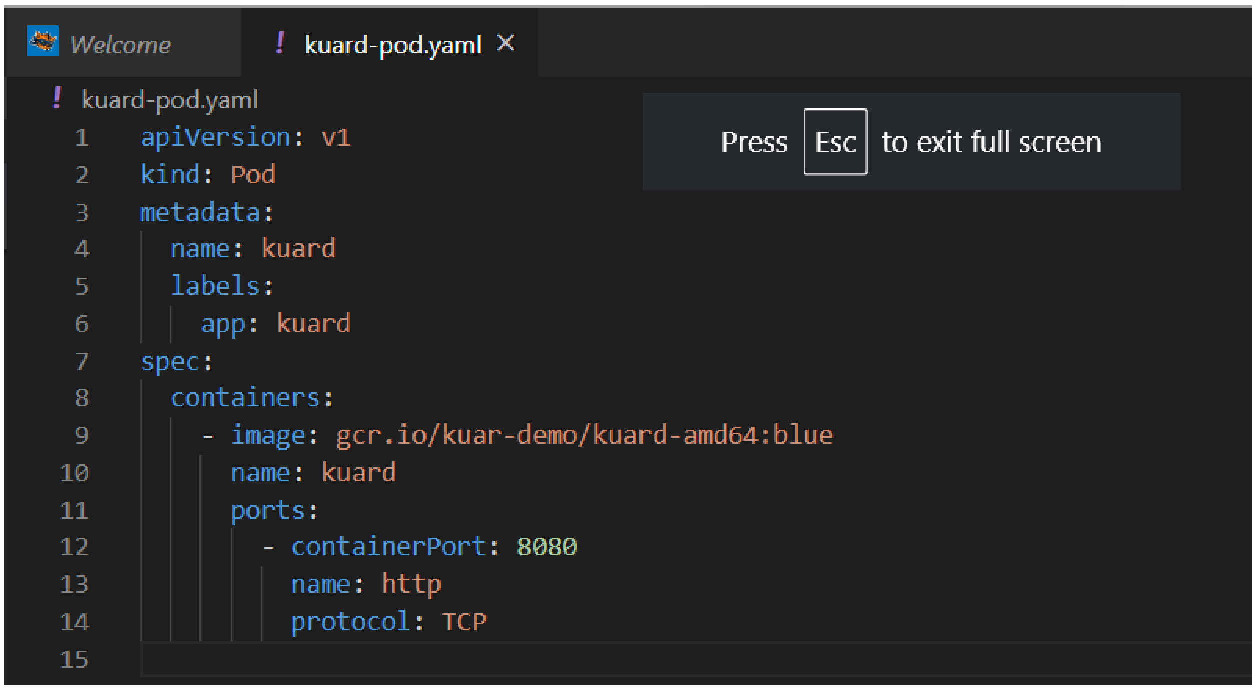
In this lab, you will create network policies that demonstrate denying and allowing a network call to a Pod. Cilium has been installed in the cluster to evaluate the network policy rules.

Accessing a Pod Without any Network Policy
Create a new namespace kuard-networkpolicy:

kubectl create namespace kuard-networkpolicy



```
$ kubectl create namespace kuard-networkpolicy
namespace/kuard-networkpolicy created
```



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: kuard
  labels:
    app: kuard
spec:
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:blue
      name: kuard
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP

```

Create the kuard Pod in the kuard-networkpolicy namespace:

kubectl apply -f kuard-pod.yaml --namespace kuard-networkpolicy

kubectl get pod kuard --namespace kuard-networkpolicy

Expose the kuard Pod as a service:

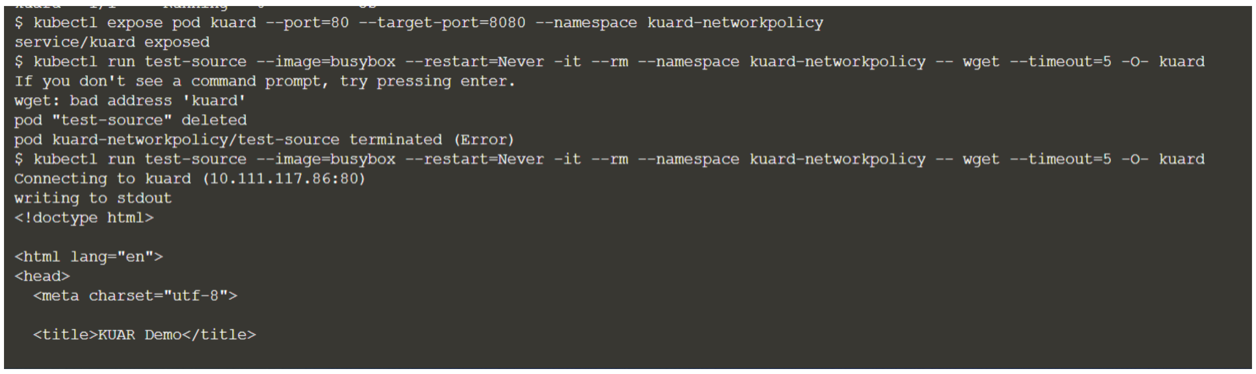
kubectl expose pod kuard --port=80 --target-port=8080 --namespace kuard-networkpolicy

Now we can use kubectl run to spin up a Pod to test as our source and test access to the kuard Pod without applying any NetworkPolicy:

kubectl run test-source --image=busybox --restart=Never -it --rm --namespace kuard-networkpolicy -- wget --timeout=5 -O- kuard

The kubectl run command you provided creates a temporary Pod named test-source in the kuard-networkpolicy namespace. This Pod runs a BusyBox container using the busybox image.

kubectl run test-source: Creates a Pod with the name test-source.
--image=busybox: Specifies that the container should use the busybox image.
--restart=Never: Sets the restart policy of the Pod to "Never," which means it won't be automatically restarted.
-it: Allocates a pseudo-TTY and attaches stdin for interactive communication.
--rm: Removes the Pod after it completes execution.
--namespace kuard-networkpolicy: Specifies the namespace kuard-networkpolicy for creating the Pod.
--: Indicates the end of the kubectl run command and the beginning of the container's command.
wget --timeout=5 -O- kuard: Executes the wget command inside the BusyBox container, attempting to retrieve the content of the kuard URL with a timeout of 5 seconds and printing the output to the console (-O-).
This command is useful for testing connectivity and retrieving the content of the kuard URL from within the kuard-networkpolicy namespace using the BusyBox container. The Pod will be automatically deleted after the wget command completes.



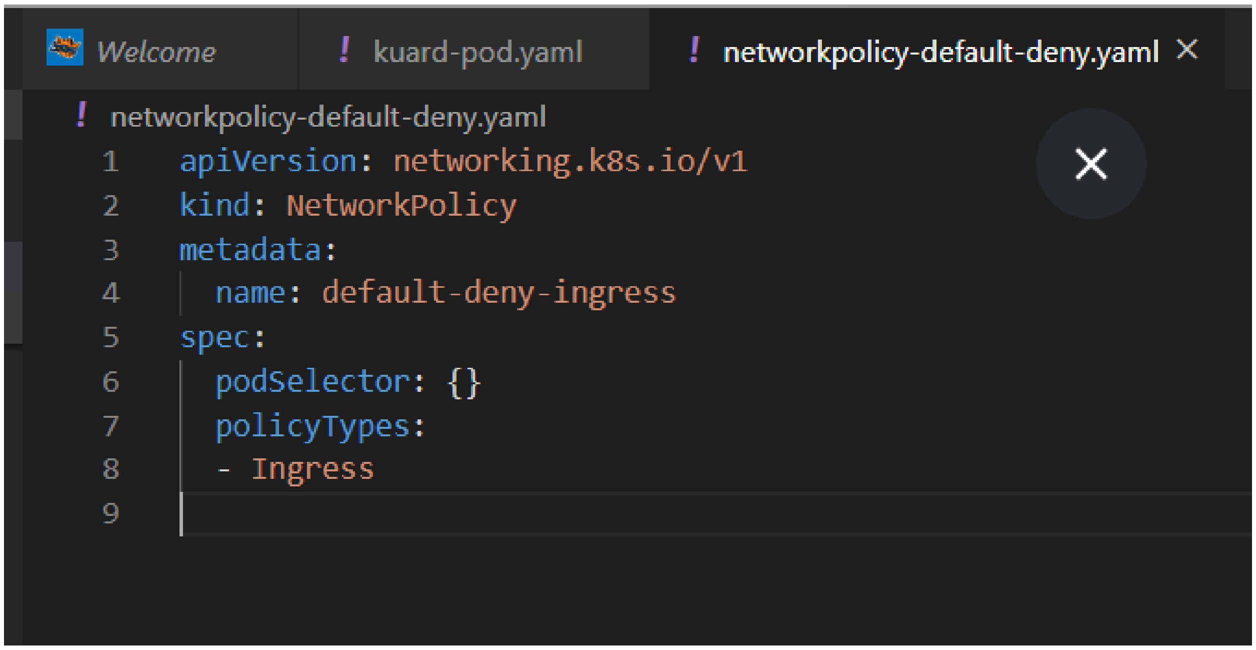
```
$ kubectl expose pod kuard --port=80 --target-port=8080 --namespace kuard-networkpolicy
service/kuard exposed
$ kubectl run test-source --image=busybox --restart=Never -it --rm --namespace kuard-networkpolicy -- wget --timeout=5 -O- kuard
If you don't see a command prompt, try pressing enter.
wget: bad address 'kuard'
pod "test-source" deleted
pod kuard-networkpolicy/test-source terminated (Error)
$ kubectl run test-source --image=busybox --restart=Never -it --rm --namespace kuard-networkpolicy -- wget --timeout=5 -O- kuard
Connecting to kuard (10.111.117.86:80)
writing to stdout
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">

  <title>KUAR Demo</title>
```

Seeing the Effects of a Default Deny Rule

There are several idiosyncrasies when using NetworkPolicy that you need to be aware of. If a Pod is matched by any NetworkPolicy resource, then any ingress or egress communication must be explicitly defined; otherwise, it will be blocked. If a Pod matches multiple NetworkPolicy resources, then the policies are additive. If a Pod isn't matched by any NetworkPolicy, then traffic is allowed. This decision was intentionally made to ease onboarding of new workloads. If you do, however, want all traffic to be blocked by default, you can create a default deny rule per namespace. The following YAML manifest is an example default deny rule that can be applied per namespace:


Welcome
kuard-pod.yaml
networkpolicy-default-deny.yaml
networkpolicy-default-deny.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Now apply the default deny network policy:

kubectl apply -f networkpolicy-default-deny.yaml --namespace kuard-networkpolicy

Now let's test access to the kuard Pod from the test-source Pod:

kubectl run test-source --image=busybox --restart=Never -it --rm --namespace kuard-networkpolicy -- wget --timeout=5 -O- kuard

We can no longer access the kuard pod from the test-source Pod due to the default deny NetworkPolicy.

The YAML manifest you provided describes a Kubernetes NetworkPolicy named default-deny-ingress. This NetworkPolicy is configured to deny all incoming network traffic to pods in the namespace where the NetworkPolicy is applied.

Here's a breakdown of the NetworkPolicy manifest:

API Version: networking.k8s.io/v1

Kind: NetworkPolicy

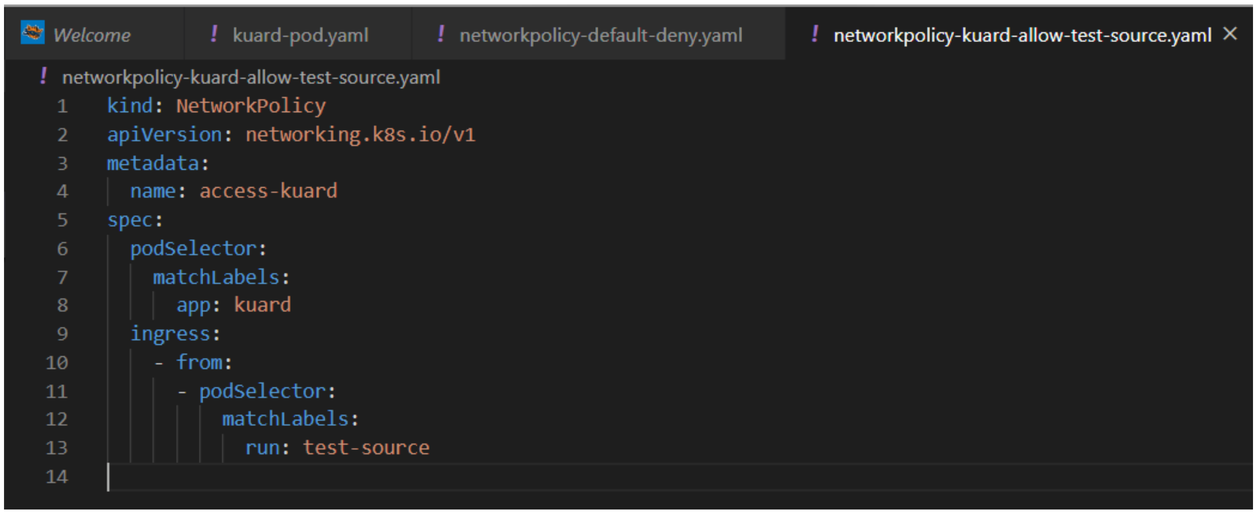
Metadata:

The name of the NetworkPolicy is set to default-deny-ingress.
Spec:

podSelector: This field is empty ({}), indicating that the NetworkPolicy applies to all pods in the namespace where it is applied. It does not select specific pods based on labels.
policyTypes: This field specifies the policy type for the NetworkPolicy. In this case, only Ingress traffic is allowed. This means that incoming network traffic to the pods is denied, while egress (outgoing) traffic is allowed.
The default-deny-ingress NetworkPolicy enforces a default deny rule for incoming traffic to pods in the namespace. This means that unless explicitly allowed by other NetworkPolicy rules, incoming traffic will be blocked. Outgoing traffic from the pods will still be allowed by default.

By applying this NetworkPolicy to a namespace, you can restrict incoming network access to the pods within that namespace, providing an additional layer of security for your applications.

Seeing the Effects of an Ingress Rule
Create a NetworkPolicy that allows access from the test-source to the kuard Pod. Create a file called networkpolicy-kuard-allow-test-source.yaml with the following contents:


Welcome
kuard-pod.yaml
networkpolicy-default-deny.yaml
networkpolicy-kuard-allow-test-source.yaml
networkpolicy-kuard-allow-test-source.yaml

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-kuard
spec:
  podSelector:
    matchLabels:
      app: kuard
  ingress:
    - from:
      - podSelector:
          matchLabels:
            run: test-source
```

```
kubectl apply -f networkpolicy-kuard-allow-test-source.yaml --namespace kuard-networkpolicy
```

Again, verify that the test-source Pod can indeed access the kuard Pod:

```
kubectl run test-source --image=busybox --restart=Never -it --rm --namespace
kuard-networkpolicy -- wget --timeout=5 -O- kuard
```

We can successfully connect to the kuard Pod, as the preceding NetworkPolicy allows incoming traffic from the test-source Pod via label selection.

The YAML manifest you provided describes a Kubernetes NetworkPolicy named access-kuard. This NetworkPolicy allows incoming network traffic to pods labeled with app: kuard from pods labeled with run: test-source.

Here's a breakdown of the NetworkPolicy manifest:

Kind: NetworkPolicy

API Version: networking.k8s.io/v1

Metadata:

The name of the NetworkPolicy is set to access-kuard.
Spec:

podSelector:
matchLabels: This field specifies the labels used to select the pods to which the NetworkPolicy applies. In this case, the NetworkPolicy applies to pods with the label app: kuard.
ingress:
from:
podSelector:
matchLabels: This field specifies the labels used to select the source pods from which incoming traffic is allowed. In this case, the NetworkPolicy allows incoming traffic from pods with the label run: test-source.
The access-kuard NetworkPolicy allows incoming network traffic to pods labeled with app: kuard only from pods labeled with run: test-source. This means that pods labeled with run: test-source are granted access to communicate with pods labeled with app: kuard via network connections.

By applying this NetworkPolicy to your cluster, you can enforce specific network access rules between different pods, providing fine-grained control over network traffic within your Kubernetes environment.

Applying NetworkPolicy provides an extra layer of security for your workloads and continues to build on the "defense in depth" and "principle of least privilege" concepts. Adding a deny rule helps with restricting network access to a bare minimum, or even none at all. Network policy rules will be applied in an additive fashion. In this lab, a second network policy was used to open up the deny rule a bit more.