



ROI Training

# Lab: Network Policies

# Enable Network Policies on the Cluster

- By default Minikube does not support Network Policies
  - To enable Network Policies in Minikube, a CNI that explicitly supports them, such as Calico, must be installed
  - To do this we must delete the cluster and create a new one
- Delete your existing minikube:  
`minikube delete`
- Start a new cluster with Calico:  
`minikube start --cni calico`

# Kubernetes config Folder

- In your codespaces terminal, change into the folder:

```
cd /workspaces/eventsapp/kubernetes-config
```

- Run the following commands to redeploy the app

```
helm install database-server oci://registry-1.docker.io/bitnamicharts/mariadb
```

- Wait for database pod to be running, then deploy the app

```
kubectl apply -f api-service.yaml
```

```
kubectl apply -f web-service.yaml
```

```
kubectl apply -f db_init_job.yaml
```

```
kubectl apply -f web-deployment.yaml
```

```
kubectl apply -f api-deployment.yaml
```

- Create the tunnel and port forward

```
minikube tunnel & kubectl port-forward service/events-web-svc 8080:80
```

# Create a Namespace

- Create a new namespace called test:

```
kubectl create namespace test
```

- List all namespaces:

```
kubectl get namespace
```

- You should see five namespaces

- View the pods in the default and test namespaces:

```
kubectl get pods
```

```
kubectl get pods -n test
```

- The Events App should be in the default namespace
  - Nothing is in the test namespace

# Testing Network Connectivity

- Run the following command to create a pod in the test namespace and open an interactive shell in the pod

```
kubectl run net-pol-demo --rm -i -t --image=curlimages/curl -n test -- sh
```

- The `--rm` option will cause the pod to automatically delete when you exit the shell
- The image used is an official docker image with curl
- Once deployed, you should see:

```
If you don't see a command prompt, try pressing enter.  
/ $
```

# Testing Network Connectivity

- At the pod prompt, verify you can get out of the cluster:  
`curl www.google.com`
  - You should see the google web page content returned
- At the pod prompt, verify you can get to the api service on port 8082 in the default namespace:  
`curl events-api-svc.default:8082/events`
  - You should see the events objects returned in JSON
  - The `.default` is because the events-api-svc is in the default namespace

# Network Policy to Deny Other Namespaces

- Open a new terminal by clicking the +
- Change into your kubernetes-config folder:  
`cd /workspaces/eventsapp/kubernetes-config`
- In the kubernetes-config folder, create a new file named  
`deny-from-other-ns.yaml`
  - You can create the file with the editor
  - Copy the contents from this slide:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: default
  name: deny-from-other-namespaces
spec:
  podSelector:
    matchLabels: {}
  ingress:
  - from:
    - podSelector: {}
```

# Network Policy to Deny Other Namespaces (continued)

- Apply the network policy:  
`kubectl apply -f deny-from-other-ns.yaml`
- Switch back to the terminal connected to the net-pol-demo pod
  - Try to reach the api service in the default namespace again:  
`curl events-api-svc.default:8082/events`
  - This time the command should just hang and not connect
  - You can no longer connect to anything in the default namespace from outside that namespace
  - Traffic within the default namespace is still allowed
- Switch back to the browser displaying the Events App
  - Try creating a new event
  - This also does not work because you are outside the namespace

# Allow Traffic to the Website Pods

- In the kubernetes-config folder, create a new file named [allow-web-traffic.yaml](#)
  - Copy the contents from this slide:
- Apply the file when ready
- Test the connectivity again
  - You should now be able to load the events app in the browser
  - Accessing the api-svc from the pod in the test namespace is still blocked

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: default
  name: allow-web-traffic
spec:
  podSelector:
    matchLabels:
      app: events-web
  ingress:
  - from:
    - ipBlock:
        cidr: '0.0.0.0/0'
```

# Block Egress Traffic

- In the kubernetes-config folder, create a new file named `block-egress.yaml`
  - Copy the contents from this slide:
- Apply the file when ready
- Switch to the terminal connected to the net-pol-demo pod, and try:  
`curl www.google.com`
  - This will no longer work
  - Even DNS is blocked:  
`nslookup www.google.com`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny-all-egress
  namespace: test
spec:
  policyTypes:
  - Egress
  podSelector: {}
  egress: []
```

# Allow Egress DNS

- In the kubernetes-config folder, create a new file named `allow-egress-dns.yaml`
  - Copy the contents from this slide:
- Apply the file when ready
- Switch to the terminal connected to the net-pol-demo pod, and try:  
`nslookup www.google.com`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-egress-dns
  namespace: test
spec:
  policyTypes:
    - Egress
  podSelector: {}
  egress:
    - ports:
        - port: 53
          protocol: UDP
        - port: 53
          protocol: TCP
    - to:
        - namespaceSelector: {}
```

# Clean Up

- Switch to the terminal connected to the net-pol-demo pod
  - Type exit to exit the shell
  - This will automatically delete the pod
- List all network policies:  
`kubectl get netpol`  
`kubectl get netpol -n test`
- Feel free to delete the network policies if you want  
`kubectl delete netpol <Name> [-n test]`

# Syncing the Changes to Git

- Commit these changes to your Git repository
  - On the left side, click the **Source Control** button 
  - Be sure ALL changes are staged by clicking in the + button
  - Type a commit message of: **Added Events app start code** and click the **Commit** button
  - Press the **Sync Changes** button and press **OK** to push the changes
- The code has now been saved to your **eventsapp** Git repository created earlier

# Success

- **Congratulations!** You have used network policies to implement pod level firewall rules
  - Restricted network traffic between pods
  - Isolated traffic based on namespaces