Experiment: Kind tcpdump image load

Create Cluster

For MacOS and Windows

Launch a terminal in your Projects folder

Create a kind cluster with a here document for the cluster config.

```
$ kind create cluster --name tcpdump-cluster --image kindest/node:v1.21.12

Γ£ô Preparing nodes ≡ƒôª Creating cluster "tcpdump-cluster" ...
 ΓÇó Ensuring node image (kindest/node:v1.21.1) ≡ƒû╝  ...
 Γ£ô Ensuring node image (kindest/node:v1.21.1) ≡ƒû╝
 ΓÇó Preparing nodes ≡ƒôª   ...
 Γ£ô Preparing nodes ≡ƒôª
 ΓÇó Writing configuration ≡ƒô£  ...
 Γ£ô Writing configuration ≡ƒô£
 ΓÇó Starting control-plane ≡ƒò╣∩╕Å  ...
 Γ£ô Starting control-plane ≡ƒò╣∩╕Å
 ΓÇó Installing CNI ≡ƒöî  ...
 Γ£ô Installing CNI ≡ƒöî
 ΓÇó Installing StorageClass ≡ƒÆ╛  ...
 Γ£ô Installing StorageClass ≡ƒÆ╛
Set kubectl context to "kind-tcpdump-cluster"
You can now use your cluster with:

kubectl cluster-info --context kind-tcpdump-cluster

Have a nice day! ≡ƒæï
```

$ kind get clusters

```
tcpdump-cluster
```

Note: remember that for **kubectl cluster-info** command, we need to prepend our cluster name with **`kind-`**

`$ kubectl cluster-info --context kind-tcpdump-cluster`

Similarly unless we used the default cluster name “kind” we need to pass in **--name** for invocations to the kind cli.

$ **kind get kubeconfig**

```
ERROR: could not locate any control plane nodes
```

$ kind get kubeconfig --name tcpdump-cluster

```
apiVersion: v1
clusters:
. . .
```

Build tcpdump image

We'll create a container image for our tcpdump image. This can be useful in many circumstances when there are issues and we want to have a container loaded into our cluster for troubleshooting.

In this experiment, we'll build the image with a "**here**" document, similar to the way we built the cluster in a previous experiment.

Paste the following into our terminal

```
docker build -t tcpdump - <<EOF
FROM ubuntu
RUN apt-get update && apt-get install -y tcpdump
CMD tcpdump -i eth0
EOF
```

```
$ docker build -t tcpdump - <<EOF
> FROM ubuntu
> RUN apt-get update && apt-get install -y tcpdump
> CMD tcpdump -i eth0
> EOF
#2 [internal] load .dockerignore
#2
sha256:6d12e025fda653c659974c1f3ffa6c64eeec150e37e3f3d45512a7a7b
45bdfc9
#2 transferring context: 2B 0.0s done
#2 DONE 0.1s

#1 [internal] load build definition from Dockerfile
#1
sha256:a3d4f9355937a661add8618395d3740cfc88354763fdb1424719c309f
f3564e7
#1 transferring dockerfile: 121B 0.0s done
#1 DONE 0.1s
. . .
#6 naming to docker.io/library/tcpdump done
```

```
#6 DONE 0.2s

Use 'docker scan' to run Snyk tests against images to find
vulnerabilities and learn how to fix them
```

Verify the image

```
$ docker images
```

```
REPOSITORY      TAG       IMAGE ID       CREATED         SIZE
tcpdump         latest    57d9ee3ae218   2 hours ago     109MB
kindest/node    <none>    32b8b755dee8   2 months ago    1.12GB
```

Since this is a latest we would need to tag that or have to set image pull policy. If we post a latest to the kind cluster it would not behave as we expect due to the default pull policy for latest to always pull.

Tag the image with a unique valid tag.

$ docker tag tcpdump:latest tcpdump:new

Validate our images

```
$ docker images
```

```
REPOSITORY      TAG       IMAGE ID       CREATED         SIZE
tcpdump         latest    57d9ee3ae218   2 hours ago     109MB
tcpdump         new       57d9ee3ae218   2 hours ago     109MB
kindest/node    <none>    32b8b755dee8   2 months ago    1.12GB
```

Load the image to our cluster.

```
kubernetes@DESKTOP-1M2VN7E MINGW64 /c/projects/kind
$ kind load docker-image tcpdump:new

Image: "tcpdump:new" with ID
"sha256:57d9ee3ae218cd2a637a83a4ad8b5a1de208d52d933358e60c60ca2f
fa4950d7" not yet present on node "kind-control-plane",
loading...
```

View the images on the cluster with the **crictl** tool

```
kubernetes@DESKTOP-1M2VN7E MINGW64 /c/projects/kind
$ docker exec -it kind-control-plane crictl images
```

Note: If we receive an error we can prepend the command with winpty on Windows environments for git bash.

```
kubernetes@DESKTOP-1M2VN7E MINGW64 /c/projects/kind
```
$ winpty docker exec -it kind-control-plane crictl images

```
IMAGE                                       TAG
IMAGE ID
    SIZE
docker.io/kindest/kindnetd                  v20210326-1e038dc5
6de166512aa22
    54MB
docker.io/library/tcpdump                   new
57d9ee3ae218c
    112MB
docker.io/rancher/local-path-provisioner    v0.0.14
e422121c9c5f9
    13.4MB
k8s.gcr.io/build-image/debian-base          v2.1.0
c7c6c86897b63
    21.1MB
k8s.gcr.io/coredns/coredns                  v1.8.0
296a6d5035e2d
    12.9MB
k8s.gcr.io/etcd                             3.4.13-0
0369cf4303ffd
    86.7MB
k8s.gcr.io/kube-apiserver                   v1.21.1
94ffe308aeff9
    127MB
k8s.gcr.io/kube-controller-manager          v1.21.1
96a295389d472
    121MB
k8s.gcr.io/kube-proxy                       v1.21.1
0e124fb3c695b
    133MB
k8s.gcr.io/kube-scheduler                   v1.21.1
1248d2d503d37
    51.9MB
k8s.gcr.io/pause                            3.5
ed210e3e4a5ba
    301kB
```

Note: We will also see the kubernetes control plane images, the kindest node, and the image we loaded.

This would enable us to execute the tcpdump to link output the communication occurring between containers.

Experiment Cleanup

Delete the cluster

$ kind delete cluster --name tcpdump-cluster

```
Deleting cluster "tcpdump-cluster" ...
```