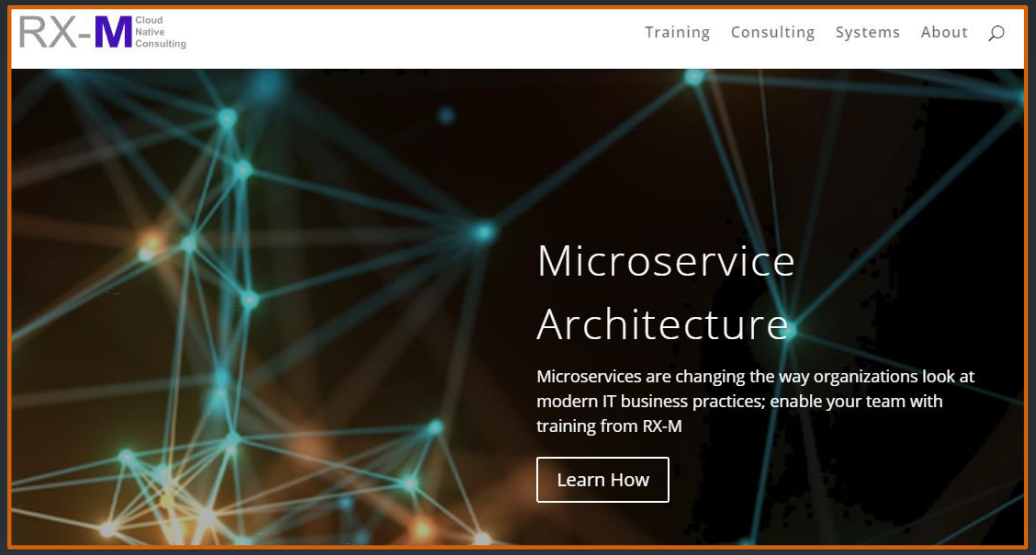
RX-M Cloud Native Consulting

Certified Kubernetes Administrator

Drills

RX-M Cloud Native Advisory, Consulting and Training

- Microservice Oriented
  - Microservices Foundation [3 Day]
  - Building Microservices on AWS [3 Day]
  - Building Microservices with Go [3 Day]
  - Building Microservices with Thrift [2 Day]
  - Building Microservices with gRPC [2 Day]
- Container Packaged
  - Docker Foundation [3 Day]
  - Docker Advanced [2 Day]
  - OCI [2 Day]
  - CNI [2 Day]
  - Containerd [2 Day]
  - Rocket [2 Day]
- Dynamically Managed
  - Kubernetes Foundation [2 Day]
  - Kubernetes Advanced [3 Day]
  - Securing Kubernetes [2 Day]
  - Kubernetes Day 2 Operations [2 Day]
  - Istio [2 Day]
  - Knative [2 Day]



kubernetes CERTIFIED SERVICE PROVIDER

RX-M Cloud Native Consulting

Overview

Day One

1. Containers & Orchestration
2. Kubernetes Architecture
3. Pods & Configs

Day Two

4. Controllers
5. Services & KubeProxy
6. Managing State

Day Three

7. Security
8. Metrics
9. Ingress

Day Four

10. Networking
11. etcd
12. Test Prep I

Day Five

13. Test Prep II
14. CKA Exam

Prerequisites:

RX-M Docker Foundation
or similar container experience

Equivalent experience:

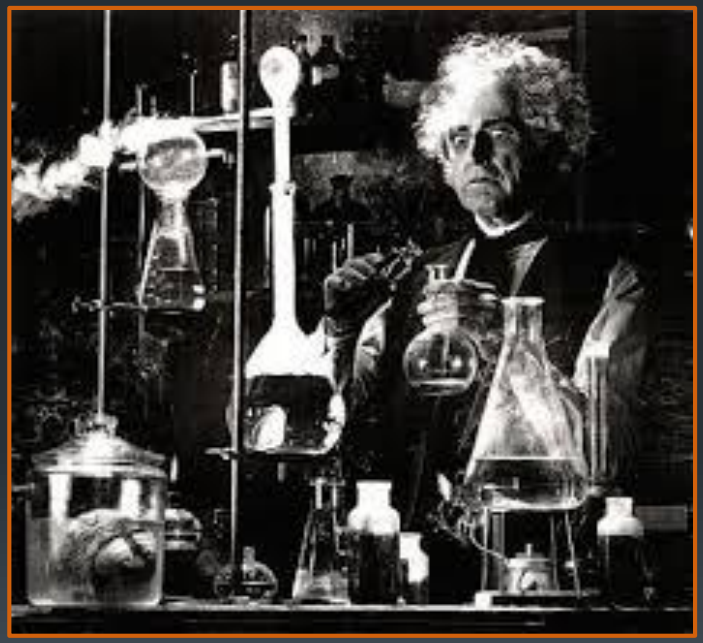
- Familiarity with container-based microservice packaging
- Understanding of container isolation and constraints
- Basic familiarity with Docker, RunC, Rocket, Garden or similar container engine

Administrative Info

- Course: Kubernetes CKA Boot Camp
- Length: 5 Days
- Format: Lecture/Labs/Discussion
- Schedule: 8:30AM – 5:30PM
  15 minute break, AM & PM
  1 hour lunch at noon
  Lab time at the end of each AM and PM session
- Location: Fire exits, Restrooms, Security, other matters
- Attendees: Name/Role/Experience/Goals for the Course

Lecture and Lab

- Our Goals in this class are two fold:
  1. Introduce concepts and ecosystems
     - Covering concepts and where things fit in the world is the primary purpose of the lecture/discussion sessions
     - The instructor will take you on a tour of the museum
       - Like a museum tour, you should interact with the instructor (tour guide), ask questions, discuss
       - Like a museum tour, you will not have time to read the slides during the tour, instead, the instructor will discuss and point out the highlights of the slides (exhibits) which will be waiting for you to read in depth later should you like to dig deeper
  2. Impart practical experience
     - This is the primary purpose of the labs
     - Classes rarely have time for complete real world projects so think of the labs as thought experiments
       - Like hands on exhibits at the museum



Kubectl Drills

Drill 1

1. Create a pod imperatively with the `redis:latest` image and label it app=`redis` with a single replica

2. List pods with the label app=`redis`

3. Get logs for all pods with the label app=`redis`

Drill 2

1. Set the worker node to unavailable and reschedule all the pods running on it

Kubectl Drills

1. Create a pod imperatively with the `redis:latest` image and label it `app=redis` with a single replica

```
kubectl run --generator=run-pod/v1 redis \
   --image=redis -l app=redis
```

2. List pods with the label `app=redis`

```
kubectl get pod -l app=redis
```

3. Get logs for all pods with the label `app=redis`

```
kubectl logs -l app=redis
```

4. Set the worker node to unavailable and reschedule all the pods running on it

`kubectl drain nodeb` (may need `--ignore-daemonsets` flag)

Kubectl Events Drills

- Retrieve the events from the kube-system namespace
- View events generated by the Replicaset controller and Default scheduler from the whole cluster
- Retrieve events generated by all kubelets and sort them by timestamp

Kubectl Events Drills

- Retrieve the events, if any, from the kube-system namespace

```
kubectl get events -n kube-system
```

- View events generated by the Replicaset controller and Default scheduler from the whole cluster

```
kubectl get events --all-namespaces --field-selector
source!=kubelet -o wide
```

- Retrieve events generated by all kubelets and sort them by timestamp

```
kubectl get events --all-namespaces --field-selector
source=kubelet --sort-by --sort-
by=.metadata.creationTimestamp
```

Pod Scheduling Drill

1. Label both of your Nodes with a unique label
   - `kubectl label node master node=master`
   - `kubectl label node worker node=worker`

2. Use the simplest method to schedule a single pod on each Node

Pod Scheduling Drill

1. Label both of your Nodes with a unique label
   - `kubectl label node ubuntu node=master`
   - `kubectl label node nodeb node=worker`

2. Use the simplest method to schedule a single pod on each Node

```
kubectl run --generator=run-pod/v1 \
  mypod --image=nginx --dry-run -o yaml
```

- Copy yaml output, pasting 2x into file (`vim mypods.yaml`)
- Place "`---`" between pods
- Set `nodeSelector` to master and worker
- Change the name of each pod
- `kubectl apply –f mypods.yaml`

Or:

```
kubectl run --generator=run-pod/v1 \
  mypod --image=nginx --overrides='{ "apiVersion": "v1",
"spec": { "nodeSelector": { "node": "master" } } }'
```

And:

```
kubectl run --generator=run-pod/v1 \
  mypod --image=nginx --overrides='{ "apiVersion": "v1",
"spec": { "nodeSelector": { "node": "worker" } } }'
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mypod
  name: mypod
spec:
  nodeSelector:
    node: master
  containers:
  - image: nginx
    name: mypod
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mypod1
  name: mypod1
spec:
  nodeSelector:
    node: worker
  containers:
  - image: nginx
    name: mypod1
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Deployment Drill

1. Create a Deployment with nginx version 1.14 with 3 replicas
2. Update the app to a new version 1.15.2
3. Rollback the update to the previous version

Note: make sure that you record each step

Deployment Drill

1. Create a Deployment with nginx version 1.14 with 3 replicas

```
kubectl create deploy mydep --image=nginx:1.14
kubectl scale deploy mydep --replicas=3
```

2. Update the app to a new version 1.15.2

```
kubectl set image deploy/mydep mydep=nginx:1.15.2 --record
```

3. Rollback the update to the previous version

```
kubectl rollout undo deploy/mydep
```

Make sure that you use the `--record option` for each step.

Jobs Drill

1. Create a Job that:
    1. Specifies 45 completions with 2 pods running in parallel
    2. Each pod should output the string "Hello" and not get restarted if it exits successfully

Jobs Drill

1. Create a Job that:
   1. Specifies 45 completions with 2 pods running in parallel
   2. Each pod should output the string "Hello" and not get restarted if it exits successfully

```
apiVersion: batch/v1
kind: Job
metadata:
  name: myjob
spec:
  parallelism: 2
  completions: 45
  template:
    metadata:
      name: myjob
    spec:
      containers:
      - name: myjob
        image: busybox
        command: ["sh","-c", "echo Hello"]
      restartPolicy: OnFailure
```

ConfigMap Drill

- Create a ConfigMap that contains this data:

```
droid=r2
shade=light
chastise=rebelscum
```

- Create and test a pod that mounts the ConfigMap as files with the path `/tmp/cmdata`

- The Pod should cat one of the files and exit

ConfigMap Drill

- Create a ConfigMap that contains this data:

```
droid=r2
shade=light
chastise=rebelscum
```

```
kubectl create cm mycm \
--from-literal=droid=r2 \
--from-literal=shade=light \
--from-literal=chastise=rebelscum
```

- Create and test a pod that mounts the ConfigMap as files with the path `/tmp/cmdata`
- The Pod should cat one of the files and exit when it runs
  - `kubectl run \ --generator=run-pod/v1 \ cmpod --image=busybox \ --dry-run \ -o yaml > cmpod.yaml`
  - Edit the file, adding the CM

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: cmpod
  name: cmpod
spec:
  containers:
  - image: busybox
    name: cmpod
    command: ["sh","-c","cat /tmp/cmdata/chastise"]
    volumeMounts:
    - mountPath: "/tmp/cmdata"
      name: data
  dnsPolicy: ClusterFirst
  restartPolicy: Never
  volumes:
  - name: data
    configMap:
      name: mycm
```

Secrets Drill

1. Create a secret that sets a username and password *from literals*
2. Create a Pod that uses the secret as environment variables
3. Check your work by exec-ing into the pod, getting the values of the env vars

Secrets Drill

1. Create a secret that sets a username and password *from literals*

```
kubectl create secret generic mysec \
--from-literal=username=user \
--from-literal=password=pass
```

2. Create a Pod that uses the secret as environment variables

```
kubectl run --generator=run-pod/v1 \
--env=USERNAME=user,PASSWORD=pass \
secpod --image=nginx --dry-run \
-o yaml > secpod.yaml
```

Edit the yaml, or omit the `--env=` flag and use envFrom in the yaml

3. Check your work by exec-ing into the pod, getting the values of the env vars

```
kubectl exec -it secpod -- /bin/bash
$ echo $USERNAME $PASSWORD
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: secpod
  name: secpod
spec:
  containers:
  - image: nginx
    name: secpod
    env:
    - name: USERNAME
      valueFrom:
        secretKeyRef:
          name: mysec
          key: username
    - name: PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysec
          key: password
--- OR ----
    envFrom:
    - secretRef:
        name: mysec
--- \OR ----
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Namespaces Drill

1. Create a namespace named myns

2. Apply a quota:
   - Persistentvolumeclaims: 1
   - nodePorts: 2
   - Pods: 5

3. Create a Role in your new namespace that allows the verbs "get" and "list" on Pods

4. Bind that role to a user named Paul

Namespaces Drill

1. Create a namespace named myns

```
kubectl create ns myns
```

2. Apply a quota:
   - Persistentvolumeclaims: 1
   - nodePorts: 2
   - Pods: 5

```
kubectl create quota myquota \
--hard=count/pvc=1,count/service.nodeport=2,count/pods=5 \
--namespace=myns
```

3. Create a Role in your new namespace that allows the verbs "get" and "list" on Pods

```
kubectl create role myrole --verb=get,list --resource=pod -n myns
```

4. Bind that role to a user named Paul

```
kubectl create rolebinding myrb --role=myrole --user=paul -n myns
```

Network Policy Drill

- Run a server pod in the default namespace

```
kubectl run server --generator=run-pod/v1 --image rxmllc/hostinfo --
port 80 -l app=info
```

- Run a client pod in another namespace called drill

```
kubectl create ns drill -l purpose=test

kubectl label ns drill purpose=test

kubectl run client -n drill --generator=run-pod/v1 --image busybox --
command -- tail -f /dev/null
```

1. Adjust your cluster so the client pod in the drill namespace can communicate with the server pod while a default-deny network policy is in place. Do not delete the network policy:

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

Network Policy Drill

1. Adjust your cluster so the client pod in the drill namespace can communicate with the server pod while a default-deny network policy is in place. Do not delete the network policy:

Create the following permissive network policy:

`nano np-client.yaml`:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-hostinfo
spec:
  podSelector:
    matchLabels:
      app: info
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: "test"
```

```
kubectl apply -f np-client.yaml
kubectl exec -it client -- wget -qO - http://<server pod IP>:9898
```

Kubelet Drill

1. Create a PodSpec that runs a single container
   - Deploy the pod w/o using the api-server

Kubelet Drill

1. Create a PodSpec that runs a single container
   - Deploy the pod w/o using the api-server

```
sudo cat /var/lib/kubelet/config.yaml | grep -i manifest

kubectl run --generator=run-pod/v1 kubeletpod --image=nginx \
--dry-run=true -o yaml
```

   - copy the file to `/etc/kubernetes/manifests` (or the manifest path)

Persistent Storage Drill

1. Statically provision a PV, create a matching PVC and a Pod that uses the PVC as a volume mount

Persistent Storage Drill

1. Statically provision a PV, create a matching PVC and a Pod that uses the PVC as a volume mount

```
$ mkdir /home/user/pv

$ kubectl run --generator=run-pod/v1 \
kubeletpod --image=nginx \
--dry-run=true -o yaml

$ kubectl exec -it persistentpod bash

root@persistentpod:/# touch /var/www/html/file

root@persistentpod:/# exit

$ ls -l pv

-rw-r--r-- 1 root root 0 Aug 17 12:16 file
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: hppv
spec:
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /home/user/pv
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: hppvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: persistentpod
spec:
  containers:
  - name: persistentcontainer
    image: nginx
    volumeMounts:
    - mountPath: "/var/www/html"
      name: persistence
  volumes:
    - name: persistence
      persistentVolumeClaim:
        claimName: hppvc
```

Shared Storage Drill

1. Create a pod with the following attributes:
   a. The pod must run two busybox containers
      i. Configure both busybox containers to run the "`tail -f /dev/null`" command
   b. The pod must have a shared host volume which will be removed by Kubernetes when the pod is deleted
   c. Create a file in the volume called "message"; make sure that this file is created before either of the two main containers run
   d. Mount the volume at "/vola" in one container and at "/data" in the other
   e. Exec into both containers to verify the "message" file is accessible on the appropriate path in both containers

Shared Storage Drill

1. Create a pod with the following attributes:
   a. The pod must run two busybox containers
      i. Configure both busybox containers to run the "`tail -f /dev/null`" command
   b. The pod must have a shared host volume which will be removed by Kubernetes when the pod is deleted
   c. Create a file in the volume called "message"; make sure that this file is created before either of the two main containers run
   d. Mount the volume at "/vola" in one container and at "/data" in the other
   e. Exec into both containers to verify the "message" file is accessible on the appropriate path in both containers

```
kubectl run --generator=run-pod/v1 \
kubeletpod --image=nginx \
--dry-run=true -o yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: drillpod
spec:
  restartPolicy: Always
  volumes:
  - name: shared-vol
    emptyDir: {}
  initContainers:
  - name: msg-creator
    image: busybox
    command: ["sh","-c"]
    args: ["touch /share/message"]
    volumeMounts:
    - mountPath: /share
      name: shared-vol
  containers:
  - name: msg-one
    image: busybox
    command: ["tail","-f", "/dev/null"]
    volumeMounts:
    - mountPath: /vola
      name: shared-vol
  - name: msg-two
    image: busybox
    command: ["tail","-f", "/dev/null"]
    volumeMounts:
    - mountPath: /data
      name: shared-vol
```

Etcd Drill

1. Backup etcd by copying the member/snap/db file
2. Backup an etcd cluster using etcdctl
3. Save the backups to /tmp/cka

Etcd Drill

1. Backup etcd by copying the member/snap/db file

```
sudo cp /var/lib/etcd/member/snap/db /tmp/cka/
```

2. Backup an etcd cluster using etcdctl

```
kubectl -n kube-system exec etcd-labsys -- \
/bin/sh -c "ETCDCTL_API=3 etcdctl \
--endpoints=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
snapshot save /var/lib/etcd/backup.db"
```

3. Save the backups to /tmp/cka

```
sudo cp /var/lib/etcd/backup.db /tmp/cka
```

Add Node Drill

1. Add a worker node with kubeadm
   - Install docker:
     - `wget -O - https://get.docker.com | sh`
   - Add the Kubernetes Apt Repo:
     - `curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -`
     - `echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list`
   - Use the quickest way to join a node to your cluster

Add Node Drill

1. Add a worker node with kubeadm
   - Install docker:
     - `wget -O - https://get.docker.com | sh`
   - Add the Kubernetes Apt Repo:
     - `curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -`
     - `echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list`
   - Use the quickest way to join a node to your cluster
     - On Master
       - `kubeadm token create --print-join-command`
     - On worker
       - ```
         kubeadm join master:6443 \
         --token 1ab2c3.v7zjj0oj39ypq0x8 \
         --discovery-token-ca-cert-hash sha256:...
         ```
         ```

Kubelet Service Drill

1. Identify the following settings for one of your Kubernetes nodes
2. Make the cka_kubelet service persists through reboots
3. Enable server certificate rotation on the Kubelet

Kubelet Service Drill

1. Identify the following settings for one of your Kubernetes nodes
   - Kubelet Service Status
   - Path to Kubelet Systemd Service
   - API Server IP Address
   - Kubelet Static Pod Manifest Path

```
sudo systemctl status kubelet
sudo systemctl status kubelet | grep "Loaded:"
sudo cat /etc/kubernetes/kubelet.conf | grep "server:"
sudo cat /var/kubelet/config.yaml| grep "static"
```

2. Make the cka_kubelet service persists through reboots

```
sudo systemctl enable cka_kubelet
```

3. Enable server certificate rotation on the Kubelet

Add the flag --rotate-server-certificates to the end of that file:

```
sudo nano /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

Or: append --rotate-server-certificates to the end of the kubelet systemd service ExecStart

```
sudo nano /lib/systemd/system/kubelet.service
```

Then reload the systemd service:

```
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

Kubelet Config Drill

1. Create a kubelet.conf file that connects to your cluster as a node named "new_worker"
2. Set up the new kubelet.conf to use certificates found /tmp/

Kubelet Config Drill

1. Create a kubelet.conf file that connects to your cluster as a node named "new worker"

   ```
   sudo cat /etc/kubernetes/kubelet.conf
   sudo cp /etc/kubernetes/kubelet.conf /tmp/cka_kubelet.conf
   nano /tmp/cka_kubelet.conf
   ```

   Change all references to `system:node:ubuntu` to `system:node:new_worker`

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ...
    server: https://<Cluster API Server>:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: system:node:new_worker
  name: system:node:new_worker@kubernetes
current-context: system:node:new_worker@kubernetes
kind: Config
preferences: {}
users:
- name: system:node:new_worker
  user:
    client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem
    client-key: /var/lib/kubelet/pki/kubelet-client-current.pem
```

TLS Bootstrap Drill

- Use Kubectl to generate a bootstrap-kubeconfig file under /tmp/bootstrap-kubeconfig.conf
  - Find out your current connection information
  - Your CA certificate is found under /etc/kubernetes/pki/ca.crt
  - Try to find a bootstrap token in your cluster
    - If you do not have one, use kubeadm to create a token

TLS Bootstrap Drill

Use Kubectl to create a bootstrap-kubeconfig file under /tmp/bootstrap-kubeconfig.conf

- Find out your current connection information

```
kubectl config view
```

  - Your CA certificate is found under /etc/kubernetes/pki/ca.crt

```
kubectl config set-cluster bootstrap \
--kubeconfig=/tmp/bootstrap-kubeconfig \
--server='https://172.31.6.52:6443' \
--certificate-authority=/etc/kubernetes/pki/ca.crt
```

- Try to find a bootstrap token in your cluster

```
kubectl get secrets -n kube-system | grep bootstrap
```

  - Use base64 to decode the token-id and token-secret

```
kubectl get secret -n kube-system $BOOTSTRAPSECRET -o jsonpath='{.data.token-id}' | base64 --
decode
kubectl get secret -n kube-system $BOOTSTRAPSECRET -o jsonpath='{.data.token-secret}' | base64
--decode
```

- If you do not have one, use kubeadm to create a token

```
kubeadm token create
```

TLS Bootstrap Drill (cont.)

- Associate the token with a user:

```
kubectl config set-credentials kubelet-bootstrap \
--kubeconfig=/tmp/bootstrap-kubeconfig \
--token=ppr919.z8k9u58ng23hqe5m
```

- Associate a user with a cluster using a context:

```
kubectl config set-context bootstrap \
--kubeconfig=/tmp/bootstrap-kubeconfig \
--user=kubelet-bootstrap \
--cluster=bootstrap
```

- Create the file by switching the context:

```
kubectl config use-context bootstrap \
--kubeconfig=/tmp/bootstrap-kubeconfig
```