RX-M Cloud Native Consulting

Advanced Kubernetes

Lab 7 – TLS and Access Control

The Kubernetes API server validates and configures data for the API objects which include pods, services, replicasets, deployments, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

Privacy

Typically the API server public IP interface is secured using TLS. In many configurations intra-cluster traffic is also secured using TLS. This means that end users and kubelets/schedulers/controller-managers/proxys need to have appropriate TLS configuration to access the API server. TLS ensures sensitive data stays private on the wire. In a typical Kubernetes cluster, the public API is served on port 443. The API server presents a certificate to clients connecting on 443 (which is self signed or otherwise). Clients can store the apiserver's CA certificate in `~/.kube/config` to enable client trust (kube-up.sh and other installation tools may do this for you).

Authentication

Once TLS is established, the HTTP request is authenticated. Kubernetes can run one or more Authenticator Modules. The authentication step typically examines headers and/or the client certificate to determine whether the user is authorized to access the API. Authentication modules include Client Certificates, Password, Plain Tokens, and JWT Tokens. Multiple authentication modules can be specified, in which case each one is tried in sequence, until one of them succeeds.

If the request cannot be authenticated, it is rejected with HTTP status code 401. Otherwise, the user is authenticated as a specific username. Some authenticators may also provide the group memberships of the user, while other authenticators do not. While Kubernetes uses "usernames" for access control decisions and in request logging, it does not have a user object nor does it store usernames or other information about users in its object store.

Authorization

Authorization happens as a separate step from authentication. Authorization applies to all HTTP accesses on the main (secure) apiserver port. The authorization check for any request compares attributes of the context of the request, (such as user, resource, and namespace) with access policies. An API call must be allowed by some policy in order to proceed.

The following implementations are available, and are selected by flag:

- `--authorization-mode=AlwaysDeny` blocks all requests (used in tests)

- `--authorization-mode=AlwaysAllow` allows all requests (disables authorization)
- `--authorization-mode=ABAC` (Attribute-Based Access Control) a simple local-file-based user-configured authorization policy
- `--authorization-mode=RBAC` (Role-Based Access Control) beta implementation allowing for authorization driven by the Kubernetes API
- `--authorization-mode=Webhook` allows for authorization driven by a remote service using REST

If multiple modes are provided, the set is unioned, and only a single authorizer is required to admit the action. For example, `--authorization-mode=AlwaysDeny,AlwaysAllow` will always allow.

In this lab we will setup a secure apiserver and test authorization.

0. Tear down any existing cluster components

To complete this lab we will setup a new cluster, be sure to shutdown any existing kubernetes apiservers, kubelets and other components.

The standard kubelet directory, /var/lib/kubelet, contains the kubelet's operating state. To avoid picking up the state from the previous kubelet in your new cluster, remove this directory:

```
ubuntu@nodea:~$ sudo rm -rf /var/lib/kubelet
```

If you can not remove this directory due to files in use, the easiest thing to do may be to reboot your lab system and then remove it.

1. Start etcd

Shut down any etcd instances you have running and clear the etcd storage:

```
ubuntu@nodea:~$ sudo rm -rf default.etcd/
```

Start etcd again:

```
ubuntu@nodea:~$ etcd --listen-client-urls 'http://0.0.0.0:2379' \
--advertise-client-urls 'http://172.31.7.235:2379'

2018-01-17 14:31:41.325817 I | etcdmain: etcd Version: 3.3.10
...
```

Create your flannel subnet in etcd and start flanneld:

```
ubuntu@nodea:~$ etcdctl set /coreos.com/network/config '{"Network":"10.1.0.0/16"}'

ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ sudo ./dist/flanneld --etcd-
endpoints=http://localhost:2379
```

nodeb:

```
ubuntu@nodeb:~$ sudo ./dist/flanneld --etcd-endpoints=http://nodea:2379
```

2. Generate CA certs and keys

Production Kubernetes systems generally use TLS to protect all intra-cluster communications.

Set the Kubernetes version to run and the architecture (be sure the version matches the version of kubernetes you have prepared in previous labs):

```
ubuntu@nodea:~$ export K8S_VERSION=v1.14 && echo $K8S_VERSION

v1.14
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ export ARCH=amd64 && echo $ARCH

amd64
ubuntu@nodea:~$
```

To secure our cluster we will need signed certificates. In this lab we'll generate our own CA. Create a new certificate authority which will be used to sign the rest of our certificates:

```
ubuntu@nodea:~$ openssl genrsa -out ca-key.pem 2048

Generating RSA private key, 2048 bit long modulus
..................................................+++
......+++
e is 65537 (0x10001)
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl req -x509 -new -nodes -key ca-key.pem -days 10000 -out ca.pem -subj "/CN=kube-ca"

ubuntu@nodea:~$
```

We will need to create an openssl configuration file to generate the api-server certificate (some options can't be specified as flags). Create the `openssl.cnf` as follows:

```
ubuntu@nodea:~$ ip a show eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:29:37:1d:cc:3e brd ff:ff:ff:ff:ff:ff
    inet 172.31.7.235/20 brd 172.31.15.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::29:37ff:fe1d:cc3e/64 scope link
       valid_lft forever preferred_lft forever
ubuntu@nodea:~$
```

The last two fields of the `openssl.cnf` are the first cluster IP used by our cluster to host the kubernetes (API) service followed by the IP of eth0 on our lab system (**be sure to replace IP.2 with your lab system IP**).

```
ubuntu@nodea:~$ vi openssl.cnf
ubuntu@nodea:~$ cat openssl.cnf

[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
```

```
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = kubernetes
DNS.2 = kubernetes.default
DNS.3 = kubernetes.default.svc
DNS.4 = kubernetes.default.svc.cluster.local
IP.1 = 10.0.0.1
IP.2 = 172.31.7.235
ubuntu@nodea:~$
```

Also be sure to replace the 172.31.7.235 example address with your actual VM address *throughout* this lab.

Now we can create the apiserver keypair:

```
ubuntu@nodea:~$ openssl genrsa -out apiserver-key.pem 2048

Generating RSA private key, 2048 bit long modulus
..................................................+++
.............................................................+++
e is 65537 (0x10001)
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl req -new -key apiserver-key.pem -out apiserver.csr \
-subj "/CN=kube-apiserver" -config openssl.cnf

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl x509 -req -in apiserver.csr -CA ca.pem -CAkey ca-key.pem \
-CAcreateserial -out apiserver.pem -days 365 -extensions v3_req -extfile openssl.cnf

Signature ok
subject=/CN=kube-apiserver
Getting CA Private Key
```

```
ubuntu@nodea:~$
```

3. Generate node certs and key

Now that we have the apiserver keys and cert prepared we will need to create a cert for each node that will connect to the master. In this example we will generate keys and certs for a single kubelet.

The certificate output will be customized per worker by worker IP. Create the file `node-openssl.cnf`:

```
ubuntu@nodea:~$ vi nodea-openssl.cnf
ubuntu@nodea:~$ cat nodea-openssl.cnf

[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
IP.1 = 172.31.7.235
ubuntu@nodea:~$
```

Now we can generate the worker node keypairs:

```
ubuntu@nodea:~$ openssl genrsa -out nodea-worker-key.pem 2048

Generating RSA private key, 2048 bit long modulus
..............................................+++
..........................................................................+++
e is 65537 (0x10001)
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl req -new -key nodea-worker-key.pem -out nodea-worker.csr \
-subj "/CN=nodea" -config nodea-openssl.cnf

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl x509 -req -in nodea-worker.csr -CA ca.pem -CAkey ca-key.pem \
-CAcreateserial -out nodea-worker.pem -days 365 -extensions v3_req -extfile nodea-openssl.cnf

Signature ok
subject=/CN=nodea
Getting CA Private Key
ubuntu@nodea:~$
```

Step 4 - Generate admin certs and key

Our final cert will identify the admin user. Generate keys and a cert for the administrative user:

```
ubuntu@nodea:~$ openssl genrsa -out admin-key.pem 2048

Generating RSA private key, 2048 bit long modulus
..............................+++
...............................................................................+++
e is 65537 (0x10001)
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl req -new -key admin-key.pem -out admin.csr -subj "/CN=kube-admin"

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl x509 -req -in admin.csr -CA ca.pem -CAkey ca-key.pem \
-CAcreateserial -out admin.pem -days 365
```

```
Signature ok
subject=/CN=kube-admin
Getting CA Private Key
ubuntu@nodea:~$
```

Verify that you now have a private key and a certificate for:

- The admin user
- The api server
- The CA
- nodea

```
ubuntu@nodea:~$ ls -l *.pem

-rw-rw-r-- 1 ubuntu ubuntu 1679 Aug 30 21:24 admin-key.pem
-rw-rw-r-- 1 ubuntu ubuntu  977 Aug 30 21:24 admin.pem
-rw-rw-r-- 1 ubuntu ubuntu 1679 Aug 30 21:19 apiserver-key.pem
-rw-rw-r-- 1 ubuntu ubuntu 1188 Aug 30 21:19 apiserver.pem
-rw-rw-r-- 1 ubuntu ubuntu 1675 Aug 30 21:14 ca-key.pem
-rw-rw-r-- 1 ubuntu ubuntu 1090 Aug 30 21:15 ca.pem
-rw-rw-r-- 1 ubuntu ubuntu 1675 Aug 30 21:22 nodea-worker-key.pem
-rw-rw-r-- 1 ubuntu ubuntu 1038 Aug 30 21:22 nodea-worker.pem
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl x509 -in admin.pem -text -noout | head

Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 16934609578823312917 (0xeb03d1b043b0e615)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=kube-ca
        Validity
            Not Before: Aug 31 04:24:36 2017 GMT
            Not After : Aug 31 04:24:36 2018 GMT
        Subject: CN=kube-admin
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ openssl rsa -in admin-key.pem -text -noout | head

Private-Key: (2048 bit)
modulus:
    00:c6:57:4f:0f:ef:ee:1a:1d:2e:00:fa:c8:cf:69:
    ef:53:c2:43:92:d9:ea:40:f8:c5:0e:04:b3:1f:ae:
    c6:52:a0:3c:e7:cf:d6:c1:12:db:a1:71:c7:f5:b5:
    d5:ea:32:c8:29:03:f2:b4:58:1a:31:b6:e4:86:0b:
    b8:2e:70:4e:53:ed:eb:9f:00:2f:8e:fc:7d:63:ef:
    1e:f0:c2:96:ec:d6:bb:86:03:04:3d:31:0f:0d:d1:
    4a:68:72:b2:c3:d2:20:98:16:88:7e:49:83:91:85:
    d8:cc:b4:30:47:23:d1:93:d9:44:cb:56:63:3a:72:
ubuntu@nodea:~$
```

Step 5 - Install the keys and certs

Kubernetes keys and certs are generally stored in `/etc/kubernetes/ssl`. Move the apiserver security assets to this directory:

```
ubuntu@nodea:~$ sudo su

root@nodea:/home/ubuntu#
```

```
root@nodea:/home/ubuntu# mkdir -p /etc/kubernetes/pki

root@nodea:/home/ubuntu#
```

```
root@nodea:/home/ubuntu# cp ca.pem apiserver.pem apiserver-key.pem /etc/kubernetes/pki/

root@nodea:/home/ubuntu#
```

Now restrict access to the private key to root only:

```
root@nodea:/home/ubuntu# sudo chmod 600 /etc/kubernetes/pki/*-key.pem

root@nodea:/home/ubuntu#
```

```
root@nodea:/home/ubuntu# sudo chown root:root /etc/kubernetes/pki/*-key.pem

root@nodea:/home/ubuntu#
```

```
root@nodea:/home/ubuntu# exit

exit
ubuntu@nodea:~$
```

Step 6 - Launch a secure API server without authorization control

Now we can run a secure API server. Exit any root shell you may have active and from the user shell execute the below command changing the `--advertise-address` flag to the address of your lab system's ens33.

```
ubuntu@nodea:~$ ip a show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:ef:63:d5:3b:be brd ff:ff:ff:ff:ff:ff
    inet 172.31.7.235/20 brd 172.31.31.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::ef:63ff:fed5:3bbe/64 scope link
       valid_lft forever preferred_lft forever

ubuntu@nodea:~$ export IP=172.31.7.235 && echo $IP

172.31.7.235

ubuntu@nodea:~$ sudo $HOME/k8s/_output/bin/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--service-cluster-ip-range=10.0.0.0/16 \
--bind-address=0.0.0.0 \
--disable-admission-plugins=ServiceAccount \
--allow-privileged=true \
--secure-port=443 \
--advertise-address=$IP \
--tls-cert-file=/etc/kubernetes/pki/apiserver.pem \
--tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem \
--client-ca-file=/etc/kubernetes/pki/ca.pem \
--service-account-key-file=/etc/kubernetes/pki/apiserver-key.pem

I0714 15:47:03.766028   72844 server.go:703] external host was not specified, using 172.31.7.235
W0331 05:00:22.976572    9759 authentication.go:415] AnonymousAuth is not allowed with the AlwaysAllow authorizer.
Resetting AnonymousAuth to false. You should use a different authorizer
I0331 05:00:22.976813    9759 server.go:146] Version: v1.14.0
I0331 05:00:23.802832    9759 plugins.go:158] Loaded 7 mutating admission controller(s) successfully in the
following order:
NamespaceLifecycle,LimitRanger,TaintNodesByCondition,Priority,DefaultTolerationSeconds,DefaultStorageClass,Mutatin
gAdmissionWebhook.
I0331 05:00:23.802858    9759 plugins.go:161] Loaded 5 validating admission controller(s) successfully in the
following order: LimitRanger,Priority,PersistentVolumeClaimResize,ValidatingAdmissionWebhook,ResourceQuota.
E0331 05:00:23.804084    9759 prometheus.go:138] failed to register depth metric admission_quota_controller:
duplicate metrics collector registration attempted
E0331 05:00:23.804113    9759 prometheus.go:150] failed to register adds metric admission_quota_controller:
duplicate metrics collector registration attempted
E0331 05:00:23.804160    9759 prometheus.go:162] failed to register latency metric admission_quota_controller:
duplicate metrics collector registration attempted
E0331 05:00:23.804193    9759 prometheus.go:174] failed to register work_duration metric
admission_quota_controller: duplicate metrics collector registration attempted
E0331 05:00:23.804223    9759 prometheus.go:189] failed to register unfinished_work_seconds metric
admission_quota_controller: duplicate metrics collector registration attempted
E0331 05:00:23.804245    9759 prometheus.go:202] failed to register longest_running_processor_microseconds metric
admission_quota_controller: duplicate metrics collector registration attempted
I0331 05:00:23.804269    9759 plugins.go:158] Loaded 7 mutating admission controller(s) successfully in the
following order:
NamespaceLifecycle,LimitRanger,TaintNodesByCondition,Priority,DefaultTolerationSeconds,DefaultStorageClass,Mutatin
gAdmissionWebhook.
I0331 05:00:23.804281    9759 plugins.go:161] Loaded 5 validating admission controller(s) successfully in the
following order: LimitRanger,Priority,PersistentVolumeClaimResize,ValidatingAdmissionWebhook,ResourceQuota.

...
```

This command line runs the kube-apiserver with the following switches:

- **--etcd-servers=http://localhost:2379** - Identifies the etcd cluster to use for kubernetes cluster state
- **--service-cluster-ip-range=10.0.0.0/16** - sets the Cluster IP service range
- **--bind-address=0.0.0.0** - bind securly (TLS) to all host interfaces
- **--disable-admission-plugins=ServiceAccount** - disables the Service Account admission controller
- **--allow-privileged=true** - allows kubernetes to run privileged containers
- **--secure-port=443** - sets the secure listening port to 443
- **--tls-cert-file=/etc/kubernetes/ssl/apiserver.pem** - the server certificate
- **--advertise-address=** - sets the address advertised by the server to the host's eth IP
- **--tls-private-key-file=/etc/kubernetes/ssl/apiserver-key.pem** - the server's private key
- **--client-ca-file=/etc/kubernetes/ssl/ca.pem** - the CA certificate
- **--service-account-key-file=/etc/kubernetes/ssl/apiserver-key.pem** - the server's private key file

Test your api server by retrieving the list of available namespaces using the insecure interface:

```
ubuntu@nodea:~$ curl -s http://127.0.0.1:8080/api/v1/namespaces | jq '.items[] | .metadata.name' -r

default
kube-node-lease
kube-public
kube-system
ubuntu@nodea:~$
```

Now test it over the secure interface using the nodea kubelet credentials:

```
ubuntu@nodea:~$ export IP=172.31.7.235 && echo $IP

172.31.7.235

ubuntu@nodea:~$ curl https://$IP:443 \
--cacert ca.pem \
--key nodea-worker-key.pem \
--cert nodea-worker.pem && echo

{
  "paths": [
    "/api",
    "/api/v1",
      "/apis",
      "/apis/",
      "/apis/admissionregistration.k8s.io",
      "/apis/admissionregistration.k8s.io/v1beta1",
      "/apis/apiextensions.k8s.io",
      "/apis/apiextensions.k8s.io/v1beta1",
      "/apis/apiregistration.k8s.io",
      "/apis/apiregistration.k8s.io/v1",
      "/apis/apiregistration.k8s.io/v1beta1",
      "/apis/apps",
      "/apis/apps/v1",
      "/apis/apps/v1beta1",
      "/apis/apps/v1beta2",
      "/apis/authentication.k8s.io",
      "/apis/authentication.k8s.io/v1",
      "/apis/authentication.k8s.io/v1beta1",
      "/apis/authorization.k8s.io",
      "/apis/authorization.k8s.io/v1",
      "/apis/authorization.k8s.io/v1beta1",
      "/apis/autoscaling",
      "/apis/autoscaling/v1",
      "/apis/autoscaling/v2beta1",
      "/apis/autoscaling/v2beta2",
      "/apis/batch",
      "/apis/batch/v1",
      "/apis/batch/v1beta1",
      "/apis/certificates.k8s.io",
      "/apis/certificates.k8s.io/v1beta1",
      "/apis/coordination.k8s.io",
      "/apis/coordination.k8s.io/v1",
      "/apis/coordination.k8s.io/v1beta1",
      "/apis/events.k8s.io",
      "/apis/events.k8s.io/v1beta1",
      "/apis/extensions",
      "/apis/extensions/v1beta1",
      "/apis/networking.k8s.io",
      "/apis/networking.k8s.io/v1",
      "/apis/networking.k8s.io/v1beta1",
      "/apis/node.k8s.io",
      "/apis/node.k8s.io/v1beta1",
      "/apis/policy",
      "/apis/policy/v1beta1",
      "/apis/rbac.authorization.k8s.io",
    "/apis/rbac.authorization.k8s.io/v1",
    "/apis/rbac.authorization.k8s.io/v1beta1",
    "/apis/scheduling.k8s.io",
    "/apis/scheduling.k8s.io/v1",
    "/apis/scheduling.k8s.io/v1beta1",
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1",
    "/apis/storage.k8s.io/v1beta1",
    "/healthz",
    "/healthz/autoregister-completion",
    "/healthz/etcd",
    "/healthz/log",
    "/healthz/ping",
    "/healthz/poststarthook/apiservice-openapi-controller",
    "/healthz/poststarthook/apiservice-registration-controller",
    "/healthz/poststarthook/apiservice-status-available-controller",
    "/healthz/poststarthook/bootstrap-controller",
    "/healthz/poststarthook/ca-registration",
    "/healthz/poststarthook/crd-informer-synced",
    "/healthz/poststarthook/generic-apiserver-start-informers",
    "/healthz/poststarthook/kube-apiserver-autoregistration",
    "/healthz/poststarthook/scheduling/bootstrap-system-priority-classes",
    "/healthz/poststarthook/start-apiextensions-controllers",
    "/healthz/poststarthook/start-apiextensions-informers",
    "/healthz/poststarthook/start-kube-aggregator-informers",
    "/healthz/poststarthook/start-kube-apiserver-admission-initializer",
    "/logs",
    "/metrics",
    "/openapi/v2",
    "/version"
  ]
}
ubuntu@nodea:~$
```

Finally try without credentials:

```
ubuntu@nodea:~$ curl -k https://$IP:443

{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401
}
ubuntu@nodea:~$
```

The `-k` switch tells curl to allow connections to unknown hosts but the API server rejects us because we do not have a certificate signed by the trusted CA.

Step 7 - Run a secure kubelet

Each node in the cluster will require a signed certificate to connect to the API server over the secure interface.

Copy the kubelet cert and key to the k8s ssl directory:

```
ubuntu@nodea:~$ sudo cp nodea-worker-key.pem nodea-worker.pem /etc/kubernetes/pki/

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ sudo chmod 600 /etc/kubernetes/pki/*-key.pem

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ sudo chown root:root /etc/kubernetes/pki/*-key.pem

ubuntu@nodea:~$
```

The kubelet uses a kubeconfig file to define the client cert and key to use against the API server. The standard kubelet directory, `/var/lib/kubelet`, contains the kubelet's operating state. Create the following kubeconfig and copy it to the standard kubeconfig directory, `/var/lib/kubelet`, ensuring that you **change the IP in the file to your VM's IP**:

```
ubuntu@nodea:~$ curl https://s3.us-east-2.amazonaws.com/rx-m-kubernetes/secure.kubeconfig -o secure.kubeconfig

ubuntu@nodea:~$ vim secure.kubeconfig
ubuntu@nodea:~$ cat secure.kubeconfig

apiVersion: v1
kind: Config
clusters:
- cluster:
    server: https://172.31.7.235:443
    certificate-authority: /etc/kubernetes/pki/ca.pem
  name: local
users:
- name: kubelet
  user:
    client-certificate: /etc/kubernetes/pki/nodea-worker.pem
    client-key: /etc/kubernetes/pki/nodea-worker-key.pem
contexts:
- context:
    cluster: local
    user: kubelet
  name: kubelet-context
current-context: kubelet-context
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ sudo mkdir -p /var/lib/kubelet/

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ sudo cp secure.kubeconfig /var/lib/kubelet/

ubuntu@nodea:~$
```

> N.B. Like the apiserver, the kubelet has --tls-cert-file and --tls-private-key-file switches. These set certs for the kubelet REST end point not the kubelet's client connection to the apiserver, only the kubeconfig provides kubelet settings for apiserver access.

Now that we have the kubelet configured we can start it.

```
ubuntu@nodea:~$ sudo $HOME/k8s/_output/bin/kubelet \
--kubeconfig=/var/lib/kubelet/secure.kubeconfig \
--config=nodea.yaml \
--allow-privileged=true \
--runtime-cgroups=/systemd/machine.slice \
--kubelet-cgroups=/systemd/machine.slice \
--pod-infra-container-image=k8s.gcr.io/pause:3.1

...

I0331 05:11:34.334768   10104 kubelet_node_status.go:72] Attempting to register node nodea
I0331 05:11:34.338509   10104 kubelet_node_status.go:75] Successfully registered node nodea
I0331 05:11:34.377863   10104 cpu_manager.go:155] [cpumanager] starting with none policy
I0331 05:11:34.377886   10104 cpu_manager.go:156] [cpumanager] reconciling every 10s
I0331 05:11:34.377896   10104 policy_none.go:42] [cpumanager] none policy: Start
W0331 05:11:34.378657   10104 manager.go:538] Failed to retrieve checkpoint for "kubelet_internal_checkpoint":
checkpoint is not found
W0331 05:11:34.379820   10104 container_manager_linux.go:818] CPUAccounting not enabled for pid: 10104
W0331 05:11:34.379919   10104 container_manager_linux.go:821] MemoryAccounting not enabled for pid: 10104
I0331 05:11:34.546456   10104 reconciler.go:154] Reconciler: start to sync state
```

We now have a secure one node cluster running with SSL between the kubelet and the apiserver.

Step 8 - Add RBAC authorization to your API server

Stop the previous apiserver and the kubelet (^C) and rerun apiserver with RBAC authorization enabled (`--authorization-mode=RBAC` added to the end of the previous command), remember to change the example IP address to your own VM ip address if you use the example below:

```
ubuntu@nodea:~$ sudo $HOME/k8s/_output/bin/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--service-cluster-ip-range=10.0.0.0/16 \
--bind-address=0.0.0.0 \
--allow-privileged=true \
--secure-port=443 \
--advertise-address=$IP \
--tls-cert-file=/etc/kubernetes/pki/apiserver.pem \
--tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem \
--client-ca-file=/etc/kubernetes/pki/ca.pem \
--service-account-key-file=/etc/kubernetes/pki/apiserver-key.pem \
--authorization-mode=RBAC

I0714 15:59:30.831434   72985 server.go:703] external host was not specified, using 172.31.7.235
I0331 05:19:15.582103   10628 server.go:146] Version: v1.14.0
I0331 05:19:16.205651   10628 plugins.go:158] Loaded 8 mutating admission controller(s) successfully in the
following order:
NamespaceLifecycle,LimitRanger,ServiceAccount,TaintNodesByCondition,Priority,DefaultTolerationSeconds,DefaultStora
geClass,MutatingAdmissionWebhook.
I0331 05:19:16.205679   10628 plugins.go:161] Loaded 6 validating admission controller(s) successfully in the
following order:
LimitRanger,ServiceAccount,Priority,PersistentVolumeClaimResize,ValidatingAdmissionWebhook,ResourceQuota.
E0331 05:19:16.206521   10628 prometheus.go:138] failed to register depth metric admission_quota_controller:
duplicate metrics collector registration attempted
E0331 05:19:16.206602   10628 prometheus.go:150] failed to register adds metric admission_quota_controller:
duplicate metrics collector registration attempted
E0331 05:19:16.206655   10628 prometheus.go:162] failed to register latency metric admission_quota_controller:
duplicate metrics collector registration attempted
E0331 05:19:16.206741   10628 prometheus.go:174] failed to register work_duration metric
admission_quota_controller: duplicate metrics collector registration attempted
E0331 05:19:16.206761   10628 prometheus.go:189] failed to register unfinished_work_seconds metric
admission_quota_controller: duplicate metrics collector registration attempted
E0331 05:19:16.206781   10628 prometheus.go:202] failed to register longest_running_processor_microseconds metric
admission_quota_controller: duplicate metrics collector registration attempted
I0331 05:19:16.206799   10628 plugins.go:158] Loaded 8 mutating admission controller(s) successfully in the
following order:
NamespaceLifecycle,LimitRanger,ServiceAccount,TaintNodesByCondition,Priority,DefaultTolerationSeconds,DefaultStora
geClass,MutatingAdmissionWebhook.
I0331 05:19:16.206811   10628 plugins.go:161] Loaded 6 validating admission controller(s) successfully in the
following order:
LimitRanger,ServiceAccount,Priority,PersistentVolumeClaimResize,ValidatingAdmissionWebhook,ResourceQuota.

...

I0331 05:19:19.730492   10628 storage_scheduling.go:122] all system priority classes are created successfully or
already exist.
I0331 05:19:19.741897   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/cluster-admin
I0331 05:19:19.744149   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:discovery
I0331 05:19:19.746332   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:basic-user
I0331 05:19:19.748195   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:public-
info-viewer
```

```
I0331 05:19:19.751161   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/admin
I0331 05:19:19.753120   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/edit
I0331 05:19:19.755146   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/view
I0331 05:19:19.757640   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:aggregate-
to-admin
I0331 05:19:19.760064   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:aggregate-
to-edit
I0331 05:19:19.761977   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:aggregate-
to-view
I0331 05:19:19.764853   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:heapster
I0331 05:19:19.767070   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:node
I0331 05:19:19.769526   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:node-
problem-detector
I0331 05:19:19.771482   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:node-
proxier
I0331 05:19:19.774367   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:kubelet-
api-admin
I0331 05:19:19.776417   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:node-
bootstrapper
I0331 05:19:19.778804   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:auth-
delegator
I0331 05:19:19.780819   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:kube-
aggregator
I0331 05:19:19.782754   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:kube-
controller-manager
I0331 05:19:19.785175   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:kube-
scheduler
I0331 05:19:19.787374   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:kube-dns
I0331 05:19:19.789806   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:persistent-volume-provisioner
I0331 05:19:19.792756   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:csi-
external-attacher
I0331 05:19:19.794876   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:aws-cloud-
provider
I0331 05:19:19.797159   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:certificates.k8s.io:certificatesigningrequests:nodeclient
I0331 05:19:19.799171   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:certificates.k8s.io:certificatesigningrequests:selfnodeclient
I0331 05:19:19.801200   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:volume-
scheduler
I0331 05:19:19.803360   10628 storage_rbac.go:195] created clusterrole.rbac.authorization.k8s.io/system:csi-
external-provisioner
I0331 05:19:19.805728   10628 storage_rbac.go:195] created
```

```
clusterrole.rbac.authorization.k8s.io/system:controller:attachdetach-controller
I0331 05:19:19.808409   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:clusterrole-aggregation-controller
I0331 05:19:19.811730   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:cronjob-controller
I0331 05:19:19.814102   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:daemon-set-controller
I0331 05:19:19.816292   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:deployment-controller
I0331 05:19:19.818857   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:disruption-controller
I0331 05:19:19.821383   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:endpoint-controller
I0331 05:19:19.823810   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:expand-controller
I0331 05:19:19.825692   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:generic-garbage-collector
I0331 05:19:19.828319   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:horizontal-pod-autoscaler
I0331 05:19:19.830250   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:job-controller
I0331 05:19:19.833298   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:namespace-controller
I0331 05:19:19.835547   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:node-controller
I0331 05:19:19.840160   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:persistent-volume-binder
I0331 05:19:19.842249   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:pod-garbage-collector
I0331 05:19:19.844635   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:replicaset-controller
I0331 05:19:19.847126   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:replication-controller
I0331 05:19:19.849335   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:resourcequota-controller
I0331 05:19:19.852281   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:route-controller
I0331 05:19:19.854435   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:service-account-controller
I0331 05:19:19.873625   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:service-controller
I0331 05:19:19.913668   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:statefulset-controller
```

```
I0331 05:19:19.953881   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:ttl-controller
I0331 05:19:19.993720   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:certificate-controller
I0331 05:19:20.033759   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:pvc-protection-controller
I0331 05:19:20.073581   10628 storage_rbac.go:195] created
clusterrole.rbac.authorization.k8s.io/system:controller:pv-protection-controller
I0331 05:19:20.113714   10628 storage_rbac.go:223] created clusterrolebinding.rbac.authorization.k8s.io/cluster-
admin
I0331 05:19:20.153686   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:discovery
I0331 05:19:20.193812   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:basic-user
I0331 05:19:20.233458   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:public-info-viewer
I0331 05:19:20.273849   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:node-proxier
I0331 05:19:20.313746   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:kube-controller-manager
I0331 05:19:20.353688   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:kube-dns
I0331 05:19:20.393681   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:kube-scheduler
I0331 05:19:20.433777   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:aws-cloud-provider
I0331 05:19:20.473860   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:volume-scheduler
I0331 05:19:20.513740   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:node
I0331 05:19:20.553694   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:attachdetach-controller
I0331 05:19:20.593601   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:clusterrole-aggregation-controller
I0331 05:19:20.633861   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:cronjob-controller
I0331 05:19:20.673688   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:daemon-set-controller
I0331 05:19:20.713918   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:deployment-controller
I0331 05:19:20.726335   10628 controller.go:102] OpenAPI AggregationController: Processing item
k8s_internal_local_delegation_chain_0000000001
I0331 05:19:20.726413   10628 controller.go:102] OpenAPI AggregationController: Processing item
```

```
k8s_internal_local_delegation_chain_0000000002
I0331 05:19:20.753757   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:disruption-controller
I0331 05:19:20.793867   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:endpoint-controller
I0331 05:19:20.833737   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:expand-controller
I0331 05:19:20.873733   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:generic-garbage-collector
I0331 05:19:20.913692   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:horizontal-pod-autoscaler
I0331 05:19:20.953719   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:job-controller
I0331 05:19:20.993958   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:namespace-controller
I0331 05:19:21.034052   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:node-controller
I0331 05:19:21.073790   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:persistent-volume-binder
I0331 05:19:21.113694   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:pod-garbage-collector
I0331 05:19:21.153743   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:replicaset-controller
I0331 05:19:21.194049   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:replication-controller
I0331 05:19:21.233705   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:resourcequota-controller
I0331 05:19:21.286720   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:route-controller
I0331 05:19:21.313783   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:service-account-controller
E0331 05:11:34.330644   10104 kubelet.go:2244] node "nodea" not found
I0331 05:11:34.334768   10104 kubelet_node_status.go:72] Attempting to register node nodea
I0331 05:11:34.338509   10104 kubelet_node_status.go:75] Successfully registered node nodea
I0331 05:11:34.377863   10104 cpu_manager.go:155] [cpumanager] starting with none policy
I0331 05:11:34.377886   10104 cpu_manager.go:156] [cpumanager] reconciling every 10s
I0331 05:11:34.377896   10104 policy_none.go:42] [cpumanager] none policy: Start
W0331 05:11:34.378657   10104 manager.go:538] Failed to retrieve checkpoint for "kubelet_internal_checkpoint":
checkpoint is not found
W0331 05:11:34.379820   10104 container_manager_linux.go:818] CPUAccounting not enabled for pid: 10104
W0331 05:11:34.379919   10104 container_manager_linux.go:821] MemoryAccounting not enabled for pid: 10104
I0331 05:11:34.546456   10104 reconciler.go:154] Reconciler: start to sync statI0331 05:19:21.353640   10628
storage_rbac.go:223] created clusterrolebinding.rbac.authorization.k8s.io/system:controller:service-controller
```

```
I0331 05:19:21.393943   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:statefulset-controller
I0331 05:19:21.433977   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:ttl-controller
I0331 05:19:21.474190   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:certificate-controller
I0331 05:19:21.513873   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:pvc-protection-controller
I0331 05:19:21.554428   10628 storage_rbac.go:223] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:pv-protection-controller
E0331 05:11:34.330644   10104 kubelet.go:2244] node "nodea" not found
I0331 05:11:34.334768   10104 kubelet_node_status.go:72] Attempting to register node nodea
I0331 05:11:34.338509   10104 kubelet_node_status.go:75] Successfully registered node nodea
I0331 05:11:34.377863   10104 cpu_manager.go:155] [cpumanager] starting with none policy
I0331 05:11:34.377886   10104 cpu_manager.go:156] [cpumanager] reconciling every 10s
I0331 05:11:34.377896   10104 policy_none.go:42] [cpumanager] none policy: Start
W0331 05:11:34.378657   10104 manager.go:538] Failed to retrieve checkpoint for "kubelet_internal_checkpoint":
checkpoint is not found
W0331 05:11:34.379820   10104 container_manager_linux.go:818] CPUAccounting not enabled for pid: 10104
W0331 05:11:34.379919   10104 container_manager_linux.go:821] MemoryAccounting not enabled for pid: 10104
I0331 05:11:34.546456   10104 reconciler.go:154] Reconciler: start to sync statI0331 05:19:21.592817   10628
controller.go:606] quota admission added evaluator for: roles.rbac.authorization.k8s.io
I0331 05:19:21.595309   10628 storage_rbac.go:254] created role.rbac.authorization.k8s.io/extension-apiserver-
authentication-reader in kube-system
I0331 05:19:21.633951   10628 storage_rbac.go:254] created
role.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-system
I0331 05:19:21.674251   10628 storage_rbac.go:254] created role.rbac.authorization.k8s.io/system:controller:cloud-
provider in kube-system
I0331 05:19:21.713933   10628 storage_rbac.go:254] created role.rbac.authorization.k8s.io/system:controller:token-
cleaner in kube-system
I0331 05:19:21.726587   10628 controller.go:102] OpenAPI AggregationController: Processing item
k8s_internal_local_delegation_chain_0000000001
I0331 05:19:21.726661   10628 controller.go:102] OpenAPI AggregationController: Processing item
k8s_internal_local_delegation_chain_0000000002
I0331 05:19:21.753989   10628 storage_rbac.go:254] created role.rbac.authorization.k8s.io/system::leader-locking-
kube-controller-manager in kube-system
I0331 05:19:21.795386   10628 storage_rbac.go:254] created role.rbac.authorization.k8s.io/system::leader-locking-
kube-scheduler in kube-system
I0331 05:19:21.834563   10628 storage_rbac.go:254] created
role.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-public
I0331 05:19:21.872787   10628 controller.go:606] quota admission added evaluator for:
rolebindings.rbac.authorization.k8s.io
I0331 05:19:21.873889   10628 storage_rbac.go:284] created
```

```
rolebinding.rbac.authorization.k8s.io/system::extension-apiserver-authentication-reader in kube-system
I0331 05:19:21.913605   10628 storage_rbac.go:284] created rolebinding.rbac.authorization.k8s.io/system::leader-
locking-kube-controller-manager in kube-system
I0331 05:19:21.954011   10628 storage_rbac.go:284] created rolebinding.rbac.authorization.k8s.io/system::leader-
locking-kube-scheduler in kube-system
I0331 05:19:21.993739   10628 storage_rbac.go:284] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-system
I0331 05:19:22.033802   10628 storage_rbac.go:284] created
rolebinding.rbac.authorization.k8s.io/system:controller:cloud-provider in kube-system
I0331 05:19:22.073765   10628 storage_rbac.go:284] created
rolebinding.rbac.authorization.k8s.io/system:controller:token-cleaner in kube-system
I0331 05:19:22.114127   10628 storage_rbac.go:284] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-public

...
```

Now rerun the previous curl command with the -v switch to get verbose output:

```
ubuntu@nodea:~$ curl -v https://$IP:443 \
--cacert ca.pem \
--key nodea-worker-key.pem \
--cert nodea-worker.pem && echo

* Rebuilt URL to: https://172.31.7.235:443/
*   Trying 172.31.7.235...
* Connected to 172.31.7.235 (172.31.7.235) port 443 (#0)
* found 1 certificates in ca.pem
* found 592 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* SSL connection using TLS1.2 / ECDHE_RSA_AES_128_GCM_SHA256
*        server certificate verification OK
*        server certificate status verification SKIPPED
*        common name: kube-apiserver (matched)
*        server certificate expiration date OK
*        server certificate activation date OK
*        certificate public key: RSA
*        certificate version: #3
*        subject: CN=kube-apiserver
*        start date: Sun, 31 Mar 2019 04:37:26 GMT
*        expire date: Mon, 30 Mar 2020 04:37:26 GMT
*        issuer: CN=kube-ca
*        compression: NULL
* ALPN, server accepted to use http/1.1
> GET / HTTP/1.1
> Host: 172.31.7.235
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Content-Type: application/json
< X-Content-Type-Options: nosniff
< Date: Sun, 31 Mar 2019 05:23:07 GMT
< Content-Length: 222
<
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"nodea\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
* Connection #0 to host 172.31.7.235 left intact
}

ubuntu@nodea:~$
```

As you can see, we have a secure TLS session (Privacy), kubernetes knows we are nodea (Authentication), however we are not allowed to look at the root IRI (Authorization). Thus the server responds with the 403 forbidden message. We will need to add roles and access permissions before we can use our new RBAC apiserver, which we'll do shortly.

Step 9 - Setup kubectl for cluster-admin access

To simplify creating RBAC object we can setup kubectl for cluster-admin access.

If you receive the message below, modify your existing config to use api server's insecure port for bootstrapping the first role:

```
ubuntu@nodea:~$ kubectl get svc

The connection to the server 172.31.7.235:8080 was refused - did you specify the right host or port?
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl config set-cluster local --server=http://localhost:8080

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl config use-context local

Switched to context "local".
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get service

NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP   26m
ubuntu@nodea:~$
```

We can temporarily make API calls via the API server insecure port, which does not enforce authentication or authorization. Now we can create a kubectl config for the cluster and cluster-admin credentials:

```
ubuntu@nodea:~$ kubectl config set-cluster my-local \
--certificate-authority=ca.pem \
--server=https://$IP

Cluster "my-local" set.
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl config set-credentials my-local-admin \
--client-certificate=admin.pem \
--client-key=admin-key.pem

User "my-local-admin" set.
ubuntu@nodea:~$
```

Now create a context that uses the cluster and the credentials, but do not activate the context:

```
ubuntu@nodea:~$ kubectl config set-context my-local-admin-ctx \
--cluster=my-local \
--user=my-local-admin

Context "my-local-admin-ctx" created.
ubuntu@nodea:~$
```

Next grant the cluster-admin ClusterRole to our kube-admin user:

```
ubuntu@nodea:~$ kubectl create clusterrolebinding kube-admin-binding \
--clusterrole=cluster-admin \
--user=kube-admin

clusterrolebinding.rbac.authorization.k8s.io "kube-admin-binding" created
ubuntu@nodea:~$
```

Now we can activate the context:

```
ubuntu@nodea:~$ kubectl config use-context my-local-admin-ctx

switched to context "my-local-admin-ctx".
ubuntu@nodea:~$
```

Verify that the my-local-admin context can access the cluster unfettered:

```
ubuntu@nodea:~$ kubectl get nodes

NAME    STATUS   ROLES    AGE     VERSION
nodea   Ready    <none>   15m     v1.14.0
ubuntu@nodea:~$
```

Optional

If you *really* want to make sure what you did worked, you can reconfigure api-server to stop listening on the insecure port (`--insecure-port=0`) and set the `local` cluster to use your node's IP:

```
ubuntu@nodea:~$ sudo $HOME/k8s/_output/bin/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--service-cluster-ip-range=10.0.0.0/16 \
--bind-address=0.0.0.0 \
--allow-privileged=true \
--secure-port=443 \
--advertise-address=$IP \
--tls-cert-file=/etc/kubernetes/pki/apiserver.pem \
--tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem \
--client-ca-file=/etc/kubernetes/pki/ca.pem \
--service-account-key-file=/etc/kubernetes/pki/apiserver-key.pem \
--authorization-mode=RBAC \
--insecure-port=0

Flag --insecure-port has been deprecated, This flag will be removed in a future version.
I0714 16:05:03.355946   73071 server.go:703] external host was not specified, using 172.31.7.235
I0714 16:05:03.356197   73071 server.go:145] Version: v1.14.0

...
```

```
ubuntu@nodea:~$ kubectl config set-cluster local --server=http://$IP:8080

Cluster "local" set.
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl config use-context local

Switched to context "local".
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get nodes

The connection to the server 172.31.7.235:8080 was refused - did you specify the right host or port?
ubuntu@nodea:~$
```

Just make sure you switch back to the my-local-admin context before moving on!

```
ubuntu@nodea:~$ kubectl config use-context my-local-admin-ctx

Switched to context "my-local-admin-ctx".

ubuntu@nodea:~$ kubectl get nodes

NAME     STATUS   ROLES    AGE   VERSION
nodea    Ready    <none>   17m   v1.14.0
user@nodea:~$
```

Step 10 - Creating RBAC objects

Now that we have our security all buttoned up we can create our first role to enable read access to the services API route:

```
ubuntu@nodea:~$ vi svc-reader.yaml
ubuntu@nodea:~$ cat svc-reader.yaml

kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: svc-reader
rules:
  - apiGroups: ["*"] # The API group "" indicates the default API Group.
    resources: ["services"]
    verbs: ["get", "watch", "list"]
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl create -f svc-reader.yaml

role.rbac.authorization.k8s.io/svc-reader created
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get role

NAME         AGE
svc-reader   8s
ubuntu@nodea:~$
```

We now have a role that allows members to get a list of services.

ClusterRoles like the one we gave our admin user, hold the same information as a Role but can apply to any namespace as well as non-namespaced resources (such as Nodes, PersistentVolumes, etc.). Create a ClusterRole to grant permissions to read pods in any namespace:

```
ubuntu@nodea:~$ vi cluster-role.yaml
ubuntu@nodea:~$ cat cluster-role.yaml

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced.
  name: pod-reader
rules:
  - apiGroups: ["*"]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl create -f cluster-role.yaml
```

```
clusterrole.rbac.authorization.k8s.io/pod-reader created
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get clusterrole

NAME                                                                   AGE
admin                                                                  9m
cluster-admin                                                          9m
edit                                                                   9m
pod-reader                                                             14s
system:aggregate-to-admin                                              9m
system:aggregate-to-edit                                               9m
system:aggregate-to-view                                               9m
system:auth-delegator                                                  9m
system:aws-cloud-provider                                              9m

...

ubuntu@nodea:~$
```

RoleBindings perform the task of granting the permission to a user or set of users. They hold a list of subjects which they apply to, and a reference to the Role being assigned. Create a RoleBinding assigns the "svc-reader" role to the user “nodea” within the “default” namespace:

```
ubuntu@nodea:~$ vim sbind.yaml
ubuntu@nodea:~$ cat sbind.yaml

kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-svc
  namespace: default
subjects:
  - kind: User # May be "User", "Group" or "ServiceAccount"
    name: nodea
roleRef:
  kind: Role
  name: svc-reader
  apiGroup: rbac.authorization.k8s.io
```

```
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl create -f sbind.yaml

rolebinding.rbac.authorization.k8s.io/read-svc created
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get rolebinding

NAME       AGE
read-svc   6s
ubuntu@nodea:~$
```

Step 11 - Test your new role

To test our new role we will try to get the service list as the admin user:

```
ubuntu@nodea:~$ kubectl config use-context my-local-admin-ctx

Switched to context "my-local-admin-ctx".
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get services

NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1     <none>        443/TCP   32m
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get rs

No resources found.
ubuntu@nodea:~$
```

Of course this works, we're cluster-admin! We can get services, rs, and anything else.

Now let's try using the nodea identity. First create credentials and a context that uses them on the local cluster:

```
ubuntu@nodea:~$ kubectl config set-credentials my-local-node \
--client-certificate=nodea-worker.pem \
--client-key=nodea-worker-key.pem

User "my-local-node" set.
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl config set-context my-local-node-ctx \
--cluster=my-local \
--user=my-local-node

Context "my-local-node-ctx" created.
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl config use-context my-local-node-ctx

Switched to context "my-local-node-ctx".
ubuntu@nodea:~$
```

Now try to list services:

```
ubuntu@nodea:~$ kubectl get services

NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP   36m
ubuntu@nodea:~$
```

It works!

Now try to get rs-es:

```
ubuntu@nodea:~$ kubectl get rs

No resources found.
Error from server (Forbidden): replicasets.extensions is forbidden: User "nodea" cannot list resource
"replicasets" in API group "extensions" in the namespace "default"
ubuntu@nodea:~$
```

Now try to get pods:

```
ubuntu@nodea:~$ kubectl get pods

Error from server (Forbidden): pods is forbidden: User "nodea" cannot list resource "pods" in API group "" in the
namespace "default"
ubuntu@nodea:~$
```

These fail because we do not have permissions on the resource types.

Add the pod reader cluster role to the nodea user:

```
ubuntu@nodea:~$ kubectl config use-context my-local-admin-ctx

Switched to context "my-local-admin-ctx".
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ vi cbind.yaml
ubuntu@nodea:~$ cat cbind.yaml

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-pods-cbind
subjects:
  - kind: User # May be "User", "Group" or "ServiceAccount"
    name: nodea
```

```
roleRef:
  kind: ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl create -f cbind.yaml

clusterrolebinding.rbac.authorization.k8s.io/read-pods-cbind created
ubuntu@nodea:~$
```

Now switch back to the nodea user and test your cluster binding:

```
ubuntu@nodea:~$ kubectl config use-context my-local-node-ctx

Switched to context "my-local-node-ctx".
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ kubectl get pods

No resources found.
ubuntu@nodea:~$
```

Now lets try a negative result:

```
ubuntu@nodea:~$ kubectl get rs

Error from server (Forbidden): replicasets.extensions is forbidden: User "nodea" cannot list resource
"replicasets" in API group "extensions" in the namespace "default"
ubuntu@nodea:~$
```

To explore RBAC further refer to the Kubernetes specs for RBAC objects: http://kubernetes.io/docs/api-reference/v1/definitions/

Congratulations, you have completed the lab!