RX-M Cloud Native Consulting

Advanced Kubernetes

==========

Lab 6 – 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

© Copyright 2017 RX-M LLC

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.

Step 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:

```
user@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.

Step 1 - Start etcd

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

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

Start etcd again:

© Copyright 2017 RX-M LLC

```
user@nodea:~$ etcd

2017-08-30 21:13:12.210432 I | etcdmain: etcd Version: 3.2.6
...
```

Step 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):

```
user@nodea:~$ export K8S_VERSION=v1.7.4

user@nodea:~$
```

```
user@nodea:~$ echo $K8S_VERSION

v1.7.4

user@nodea:~$
```

```
user@nodea:~$ export ARCH=amd64

user@nodea:~$
```

```
user@nodea:~$ echo $ARCH

amd64

user@nodea:~$
```


© Copyright 2017 RX-M LLC

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:

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

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

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

user@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:

```
user@nodea:~$ ip a s ens33

2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:32:15:a2 brd ff:ff:ff:ff:ff:ff
    inet 172.16.151.203/24 brd 172.16.151.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe32:15a2/64 scope link
       valid_lft forever preferred_lft forever

user@nodea:~$
```

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

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


© Copyright 2017 RX-M LLC

```
[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 = 172.16.151.203

user@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 ens33 on our lab system (**be sure to replace IP.2 with your lab system IP**).

Also be sure to replace the 172.16.151.203 example address with your actual VM address *throughout* this lab.

Now we can create the apiserver keypair:

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

Generating RSA private key, 2048 bit long modulus
.....................................................+++
.....................................................+++
e is 65537 (0x10001)

user@nodea:~$
```

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

user@nodea:~$
```


© Copyright 2017 RX-M LLC

```
user@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

user@nodea:~$
```

Step 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`:

```
user@nodea:~$ vi node-openssl.cnf
user@nodea:~$ cat node-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 = 172.16.151.203

user@nodea:~$
```

Now we can generate the worker node keypairs:

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


© Copyright 2017 RX-M LLC

```
Generating RSA private key, 2048 bit long modulus
.............................................+++
.....................................................................+++
e is 65537 (0x10001)

user@nodea:~$
```

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

user@nodea:~$
```

```
user@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 node-openssl.cnf

Signature ok
subject=/CN=nodea
Getting CA Private Key

user@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:

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

Generating RSA private key, 2048 bit long modulus
.............................+++
.....................................................................+++
e is 65537 (0x10001)

user@nodea:~$
```


© Copyright 2017 RX-M LLC

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

user@nodea:~$
```

```
user@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

user@nodea:~$
```

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

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

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

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

user@nodea:~$
```

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


© Copyright 2017 RX-M LLC

```
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

user@nodea:~$
```

```
user@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:

user@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:

```
user@nodea:~$ sudo su

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


© Copyright 2017 RX-M LLC

```
root@nodea:/home/user# mkdir -p /etc/kubernetes/ssl
root@nodea:/home/user#
```

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

Now restrict access to the private key to root only:

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

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

```
root@nodea:/home/user# exit
exit
user@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.

© Copyright 2017 RX-M LLC

```
user@nodea:~$ sudo ~user/k8s/_output/bin/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--runtime-config=rbac.authorization.k8s.io/v1beta1 \
--service-cluster-ip-range=10.0.0.0/16 \
--bind-address=0.0.0.0 \
--allow-privileged=true \
--secure-port=443 \
--advertise-address=172.16.151.203 \
--tls-cert-file=/etc/kubernetes/ssl/apiserver.pem \
--tls-private-key-file=/etc/kubernetes/ssl/apiserver-key.pem \
--client-ca-file=/etc/kubernetes/ssl/ca.pem \
--service-account-key-file=/etc/kubernetes/ssl/apiserver-key.pem

I0830 21:32:00.587129   16510 server.go:112] Version: v1.7.4+793658f2d7ca7
W0830 21:32:00.587280   16510 authentication.go:368] AnonymousAuth is not allowed with the AllowAll authorizer.
Resetting AnonymousAuth to false. You should use a different authorizer
W0830 21:32:01.112698   16510 genericapiserver.go:325] Skipping API autoscaling/v2alpha1 because it has no
resources.
W0830 21:32:01.114125   16510 genericapiserver.go:325] Skipping API batch/v2alpha1 because it has no resources.
[restful] 2017/08/30 21:32:01 log.go:33: [restful/swagger] listing is available at
https://172.16.151.203:443/swaggerapi
[restful] 2017/08/30 21:32:01 log.go:33: [restful/swagger] https://172.16.151.203:443/swaggerui/ is mapped to
folder /swagger-ui/
I0830 21:32:04.520695   16510 insecure_handler.go:118] Serving insecurely on 127.0.0.1:8080
I0830 21:32:04.521161   16510 serve.go:85] Serving securely on 0.0.0.0:443
...
```

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
- **--runtime-config=rbac.authorization.k8s.io/v1beta1** - enables the beta version RBAC api interface
- **--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
- **--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=172.16.151.203** - sets the address advertised by the server to the host's eth0 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

© Copyright 2017 RX-M LLC

- **--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:

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

default
kube-public
kube-system

user@nodea:~$
```

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

```
user@nodea:~$ curl https://172.16.151.203:443 \
--cacert ca.pem \
--key nodea-worker-key.pem \
--cert nodea-worker.pem && echo

{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/apis/apiextensions.k8s.io",
    "/apis/apiextensions.k8s.io/v1beta1",
    "/apis/apiregistration.k8s.io",
    "/apis/apiregistration.k8s.io/v1beta1",
    "/apis/apps",
    "/apis/apps/v1beta1",
    "/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/batch",
```


© Copyright 2017 RX-M LLC

```
    "/apis/batch/v1",
    "/apis/certificates.k8s.io",
    "/apis/certificates.k8s.io/v1beta1",
    "/apis/extensions",
    "/apis/extensions/v1beta1",
    "/apis/networking.k8s.io",
    "/apis/networking.k8s.io/v1",
    "/apis/policy",
    "/apis/policy/v1beta1",
    "/apis/rbac.authorization.k8s.io",
    "/apis/rbac.authorization.k8s.io/v1alpha1",
    "/apis/rbac.authorization.k8s.io/v1beta1",
    "/apis/settings.k8s.io",
    "/apis/settings.k8s.io/v1alpha1",
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1",
    "/apis/storage.k8s.io/v1beta1",
    "/healthz",
    "/healthz/autoregister-completion",
    "/healthz/ping",
    "/healthz/poststarthook/apiservice-registration-controller",
    "/healthz/poststarthook/apiservice-status-available-controller",
    "/healthz/poststarthook/bootstrap-controller",
    "/healthz/poststarthook/ca-registration",
    "/healthz/poststarthook/extensions/third-party-resources",
    "/healthz/poststarthook/generic-apiserver-start-informers",
    "/healthz/poststarthook/kube-apiserver-autoregistration",
    "/healthz/poststarthook/start-apiextensions-controllers",
    "/healthz/poststarthook/start-apiextensions-informers",
    "/healthz/poststarthook/start-kube-aggregator-informers",
    "/healthz/poststarthook/start-kube-apiserver-informers",
    "/logs",
    "/metrics",
    "/swagger-2.0.0.json",
    "/swagger-2.0.0.pb-v1",
    "/swagger-2.0.0.pb-v1.gz",
    "/swagger.json",
    "/swaggerapi",
    "/ui",
    "/ui/",
    "/version"
  ]
}
```


© Copyright 2017 RX-M LLC

```
user@nodea:~$
```

Finally try without credentials:

```
user@nodea:~$ curl -k https://172.16.151.203:443
Unauthorized
user@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:

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

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

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

© Copyright 2017 RX-M LLC

the kubelet's operating state. Create the following kubeconfig and copy it to the standard kubeconfig directory, `/var/lib/kubelet`:

```
user@nodea:~$ vi kubeconfig
user@nodea:~$ cat kubeconfig

apiVersion: v1
kind: Config
clusters:
- cluster:
    server: https://172.16.151.203:443
    certificate-authority: /etc/kubernetes/ssl/ca.pem
  name: local
users:
- name: kubelet
  user:
    client-certificate: /etc/kubernetes/ssl/nodea-worker.pem
    client-key: /etc/kubernetes/ssl/nodea-worker-key.pem
contexts:
- context:
    cluster: local
    user: kubelet
  name: kubelet-context
current-context: kubelet-context

user@nodea:~$
```

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

user@nodea:~$
```

```
user@nodea:~$ sudo cp kubeconfig /var/lib/kubelet/

user@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.

© Copyright 2017 RX-M LLC

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

```
user@nodea:~$ sudo ~user/k8s/_output/bin/kubelet \
--allow-privileged=true \
--kubeconfig=/var/lib/kubelet/kubeconfig \
--require-kubeconfig

I0830 21:44:44.244843   18576 feature_gate.go:144] feature gates: map[]
I0830 21:44:44.249680   18576 client.go:72] Connecting to docker on unix:///var/run/docker.sock
I0830 21:44:44.249907   18576 client.go:92] Start docker client with request timeout=2m0s
W0830 21:44:44.251016   18576 cni.go:189] Unable to update cni config: No networks found in /etc/cni/net.d
I0830 21:44:44.257198   18576 manager.go:143] cAdvisor running in container: "/user.slice"
W0830 21:44:44.269251   18576 manager.go:151] unable to connect to Rkt api service: rkt: cannot tcp Dial rkt api
service: dial tcp [::1]:15441: getsockopt: connection refused
I0830 21:44:44.279990   18576 fs.go:117] Filesystem partitions: map[/dev/sda1:{mountpoint:/var/lib/docker/aufs
major:8 minor:1 fsType:ext4 blockSize:0}]
I0830 21:44:44.281094   18576 manager.go:198] Machine: {NumCores:2 CpuFrequency:2711828 MemoryCapacity:4124880896
MachineID:6e883acc04fc7db3713776be57a3dac9 SystemUUID:F2564D56-3460-443D-57E3-836F703215A2 BootID:64a65210-18dc-
435a-a471-07aa96c9d5ee Filesystems:[{Device:/dev/sda1 DeviceMajor:8 DeviceMinor:1 Capacity:18889830400 Type:vfs
Inodes:1179648 HasInodes:true}] DiskMap:map[8:0:{Name:sda Major:8 Minor:0 Size:21474836480 Scheduler:deadline}]
NetworkDevices:[{Name:ens33 MacAddress:00:0c:29:32:15:a2 Speed:1000 Mtu:1500}] Topology:[{Id:0 Memory:4124880896
Cores:[{Id:0 Threads:[0] Caches:[{Size:32768 Type:Data Level:1} {Size:32768 Type:Instruction Level:1} {Size:262144
Type:Unified Level:2}]}] Caches:[{Size:8388608 Type:Unified Level:3}]} {Id:2 Memory:0 Cores:[{Id:0 Threads:[1]
Caches:[{Size:32768 Type:Data Level:1} {Size:32768 Type:Instruction Level:1} {Size:262144 Type:Unified Level:2}]}]
Caches:[{Size:8388608 Type:Unified Level:3}]}] CloudProvider:Unknown InstanceType:Unknown InstanceID:None}
I0830 21:44:44.282097   18576 manager.go:204] Version: {KernelVersion:4.4.0-93-generic ContainerOsVersion:Ubuntu
16.04.1 LTS DockerVersion:17.06.1-ce DockerAPIVersion:1.30 CadvisorVersion: CadvisorRevision:}
I0830 21:44:44.282703   18576 server.go:536] --cgroups-per-qos enabled, but --cgroup-root was not specified.
defaulting to /
W0830 21:44:44.284150   18576 container_manager_linux.go:216] Running with swap on is not supported, please
disable swap! This will be a fatal error by default starting in K8s v1.6! In the meantime, you can opt-in to
making this a fatal error by enabling --experimental-fail-swap-on.
I0830 21:44:44.284192   18576 container_manager_linux.go:246] container manager verified user specified cgroup-
root exists: /
I0830 21:44:44.284203   18576 container_manager_linux.go:251] Creating Container Manager object based on Node
Config: {RuntimeCgroupsName: SystemCgroupsName: KubeletCgroupsName: ContainerRuntime:docker CgroupsPerQOS:true
CgroupRoot:/ CgroupDriver:cgroupfs ProtectKernelDefaults:false NodeAllocatableConfig:{KubeReservedCgroupName:
SystemReservedCgroupName: EnforceNodeAllocatable:map[pods:{}] KubeReserved:map[] SystemReserved:map[]
HardEvictionThresholds:[{Signal:memory.available Operator:LessThan Value:{Quantity:100Mi Percentage:0}
GracePeriod:0s MinReclaim:<nil>} {Signal:nodefs.available Operator:LessThan Value:{Quantity:<nil> Percentage:0.1}
GracePeriod:0s MinReclaim:<nil>} {Signal:nodefs.inodesFree Operator:LessThan Value:{Quantity:<nil>
Percentage:0.05} GracePeriod:0s MinReclaim:<nil>}]} ExperimentalQOSReserved:map[]}
```


© Copyright 2017 RX-M LLC

```
I0830 21:44:44.284363   18576 kubelet.go:273] Watching apiserver
W0830 21:44:44.303332   18576 kubelet_network.go:70] Hairpin mode set to "promiscuous-bridge" but kubenet is not
enabled, falling back to "hairpin-veth"
I0830 21:44:44.303566   18576 kubelet.go:508] Hairpin mode set to "hairpin-veth"
W0830 21:44:44.305712   18576 cni.go:189] Unable to update cni config: No networks found in /etc/cni/net.d
I0830 21:44:44.312830   18576 docker_service.go:208] Docker cri networking managed by kubernetes.io/no-op
I0830 21:44:44.328714   18576 docker_service.go:225] Setting cgroupDriver to cgroupfs
I0830 21:44:44.347352   18576 remote_runtime.go:42] Connecting to runtime service unix:///var/run/dockershim.sock
I0830 21:44:44.349586   18576 kuberuntime_manager.go:163] Container runtime docker initialized, version: 17.06.1-
ce, apiVersion: 1.30.0
I0830 21:44:44.351050   18576 server.go:943] Started kubelet v1.7.4+793658f2d7ca7
E0830 21:44:44.351538   18576 kubelet.go:1229] Image garbage collection failed once. Stats initialization may not
have completed yet: unable to find data for container /
I0830 21:44:44.352687   18576 kubelet_node_status.go:247] Setting node annotation to enable volume controller
attach/detach
I0830 21:44:44.352962   18576 server.go:132] Starting to listen on 0.0.0.0:10250
I0830 21:44:44.361863   18576 server.go:310] Adding debug handlers to kubelet server.
E0830 21:44:44.365998   18576 kubelet.go:1729] Failed to check if disk space is available for the runtime: failed
to get fs info for "runtime": unable to find data for container /
E0830 21:44:44.369095   18576 kubelet.go:1737] Failed to check if disk space is available on the root partition:
failed to get fs info for "root": unable to find data for container /
I0830 21:44:44.369723   18576 fs_resource_analyzer.go:66] Starting FS ResourceAnalyzer
I0830 21:44:44.369747   18576 status_manager.go:140] Starting to sync pod status with apiserver
I0830 21:44:44.369758   18576 kubelet.go:1809] Starting kubelet main sync loop.
I0830 21:44:44.369795   18576 kubelet.go:1820] skipping pod synchronization - [container runtime is down PLEG is
not healthy: pleg was last seen active 2562047h47m16.854775807s ago; threshold is 3m0s]
W0830 21:44:44.370177   18576 container_manager_linux.go:747] CPUAccounting not enabled for pid: 18576
W0830 21:44:44.370188   18576 container_manager_linux.go:750] MemoryAccounting not enabled for pid: 18576
E0830 21:44:44.370218   18576 container_manager_linux.go:543] [ContainerManager]: Fail to get rootfs information
unable to find data for container /
I0830 21:44:44.370238   18576 volume_manager.go:245] Starting Kubelet Volume Manager
I0830 21:44:44.389824   18576 factory.go:351] Registering Docker factory
W0830 21:44:44.390064   18576 manager.go:247] Registration of the rkt container factory failed: unable to
communicate with Rkt api service: rkt: cannot tcp Dial rkt api service: dial tcp [::1]:15441: getsockopt:
connection refused
I0830 21:44:44.390267   18576 factory.go:54] Registering systemd factory
I0830 21:44:44.390959   18576 factory.go:86] Registering Raw factory
I0830 21:44:44.391352   18576 manager.go:1121] Started watching for new ooms in manager
I0830 21:44:44.391726   18576 oomparser.go:185] oomparser using systemd
I0830 21:44:44.392227   18576 manager.go:288] Starting recovery of all containers
I0830 21:44:44.470784   18576 kubelet_node_status.go:247] Setting node annotation to enable volume controller
attach/detach
I0830 21:44:44.475357   18576 manager.go:293] Recovery completed
```


© Copyright 2017 RX-M LLC

```
I0830 21:44:44.493900   18576 kubelet_node_status.go:82] Attempting to register node nodea
I0830 21:44:44.500959   18576 kubelet_node_status.go:85] Successfully registered node nodea
E0830 21:44:44.598241   18576 helpers.go:771] Could not find capacity information for resource
storage.kubernetes.io/scratch
W0830 21:44:44.598649   18576 helpers.go:782] eviction manager: no observation found for eviction signal
allocatableNodeFs.available
...
```

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:

```
user@nodea:~$ sudo ~/k8s/_output/local/bin/linux/amd64/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--runtime-config=rbac.authorization.k8s.io/v1beta1 \
--service-cluster-ip-range=10.0.0.0/16 \
--bind-address=0.0.0.0 \
--allow-privileged=true \
--secure-port=443 \
--advertise-address=172.16.151.203 \
--tls-cert-file=/etc/kubernetes/ssl/apiserver.pem \
--tls-private-key-file=/etc/kubernetes/ssl/apiserver-key.pem \
--client-ca-file=/etc/kubernetes/ssl/ca.pem \
--service-account-key-file=/etc/kubernetes/ssl/apiserver-key.pem \
--authorization-mode=RBAC

I0830 21:47:15.754606   18951 server.go:112] Version: v1.7.4+793658f2d7ca7
W0830 21:47:16.058372   18951 genericapiserver.go:325] Skipping API autoscaling/v2alpha1 because it has no
resources.
W0830 21:47:16.059203   18951 genericapiserver.go:325] Skipping API batch/v2alpha1 because it has no resources.
[restful] 2017/08/30 21:47:16 log.go:33: [restful/swagger] listing is available at
https://172.16.151.203:443/swaggerapi
[restful] 2017/08/30 21:47:16 log.go:33: [restful/swagger] https://172.16.151.203:443/swaggerui/ is mapped to
folder /swagger-ui/
I0830 21:47:19.350742   18951 insecure_handler.go:118] Serving insecurely on 127.0.0.1:8080
I0830 21:47:19.351459   18951 serve.go:85] Serving securely on 0.0.0.0:443
I0830 21:47:19.351681   18951 tprregistration_controller.go:144] Starting tpr-autoregister controller
```


© Copyright 2017 RX-M LLC

```
I0830 21:47:19.351840   18951 controller_utils.go:994] Waiting for caches to sync for tpr-autoregister controller
I0830 21:47:19.352161   18951 crd_finalizer.go:248] Starting CRDFinalizer
I0830 21:47:19.352658   18951 apiservice_controller.go:113] Starting APIServiceRegistrationController
I0830 21:47:19.352796   18951 cache.go:32] Waiting for caches to sync for APIServiceRegistrationController
controller
I0830 21:47:19.353060   18951 available_controller.go:201] Starting AvailableConditionController
I0830 21:47:19.353258   18951 cache.go:32] Waiting for caches to sync for AvailableConditionController controller
I0830 21:47:19.353612   18951 autoregister_controller.go:120] Starting autoregister controller
I0830 21:47:19.353756   18951 cache.go:32] Waiting for caches to sync for autoregister controller
I0830 21:47:19.353905   18951 customresource_discovery_controller.go:152] Starting DiscoveryController
I0830 21:47:19.354064   18951 naming_controller.go:284] Starting NamingConditionController
I0830 21:47:19.452917   18951 controller_utils.go:1001] Caches are synced for tpr-autoregister controller
I0830 21:47:19.453242   18951 cache.go:39] Caches are synced for APIServiceRegistrationController controller
I0830 21:47:19.453607   18951 cache.go:39] Caches are synced for AvailableConditionController controller
I0830 21:47:19.454289   18951 cache.go:39] Caches are synced for autoregister controller
I0830 21:47:20.360971   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/cluster-admin
I0830 21:47:20.363731   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:discovery
I0830 21:47:20.366429   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:basic-user
I0830 21:47:20.369197   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/admin
I0830 21:47:20.374749   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/edit
I0830 21:47:20.377307   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/view
I0830 21:47:20.379627   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:heapster
I0830 21:47:20.382426   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:node
I0830 21:47:20.384763   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:node-
problem-detector
I0830 21:47:20.387251   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:node-
proxier
I0830 21:47:20.389708   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:node-
bootstrapper
I0830 21:47:20.392398   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:auth-
delegator
I0830 21:47:20.395259   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:kube-
aggregator
I0830 21:47:20.398882   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:kube-
controller-manager
I0830 21:47:20.402300   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:kube-
scheduler
I0830 21:47:20.405421   18951 storage_rbac.go:178] created clusterrole.rbac.authorization.k8s.io/system:kube-dns
I0830 21:47:20.410372   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:persistent-volume-provisioner
I0830 21:47:20.413516   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:attachdetach-controller
I0830 21:47:20.416176   18951 storage_rbac.go:178] created
```


© Copyright 2017 RX-M LLC

```
clusterrole.rbac.authorization.k8s.io/system:controller:cronjob-controller
I0830 21:47:20.418703   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:daemon-set-controller
I0830 21:47:20.421519   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:deployment-controller
I0830 21:47:20.424499   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:disruption-controller
I0830 21:47:20.428458   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:endpoint-controller
I0830 21:47:20.432317   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:generic-garbage-collector
I0830 21:47:20.438027   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:horizontal-pod-autoscaler
I0830 21:47:20.440785   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:job-controller
I0830 21:47:20.445361   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:namespace-controller
I0830 21:47:20.448478   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:node-controller
I0830 21:47:20.452039   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:persistent-volume-binder
I0830 21:47:20.454969   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:pod-garbage-collector
I0830 21:47:20.460045   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:replicaset-controller
I0830 21:47:20.462815   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:replication-controller
I0830 21:47:20.465459   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:resourcequota-controller
I0830 21:47:20.468023   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:route-controller
I0830 21:47:20.471075   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:service-account-controller
I0830 21:47:20.474527   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:service-controller
I0830 21:47:20.478381   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:statefulset-controller
I0830 21:47:20.482292   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:ttl-controller
I0830 21:47:20.485926   18951 storage_rbac.go:178] created
clusterrole.rbac.authorization.k8s.io/system:controller:certificate-controller
I0830 21:47:20.489047   18951 storage_rbac.go:206] created clusterrolebinding.rbac.authorization.k8s.io/cluster-
admin
```


© Copyright 2017 RX-M LLC

```
I0830 21:47:20.491849   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:discovery
I0830 21:47:20.495183   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:basic-user
I0830 21:47:20.498164   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:node-proxier
I0830 21:47:20.500986   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:kube-controller-manager
I0830 21:47:20.503719   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:kube-dns
I0830 21:47:20.508952   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:kube-scheduler
I0830 21:47:20.511890   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:node
I0830 21:47:20.514525   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:attachdetach-controller
I0830 21:47:20.517341   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:cronjob-controller
I0830 21:47:20.519908   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:daemon-set-controller
I0830 21:47:20.536089   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:deployment-controller
I0830 21:47:20.541188   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:disruption-controller
I0830 21:47:20.546274   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:endpoint-controller
I0830 21:47:20.559423   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:generic-garbage-collector
I0830 21:47:20.598740   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:horizontal-pod-autoscaler
I0830 21:47:20.637490   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:job-controller
I0830 21:47:20.676729   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:namespace-controller
I0830 21:47:20.716896   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:node-controller
I0830 21:47:20.755736   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:persistent-volume-binder
I0830 21:47:20.797620   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:pod-garbage-collector
I0830 21:47:20.836477   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:replicaset-controller
I0830 21:47:20.877309   18951 storage_rbac.go:206] created
```


© Copyright 2017 RX-M LLC

```
clusterrolebinding.rbac.authorization.k8s.io/system:controller:replication-controller
I0830 21:47:20.917258   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:resourcequota-controller
I0830 21:47:20.956499   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:route-controller
I0830 21:47:20.998152   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:service-account-controller
I0830 21:47:21.037943   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:service-controller
I0830 21:47:21.076312   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:statefulset-controller
I0830 21:47:21.118543   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:ttl-controller
I0830 21:47:21.156920   18951 storage_rbac.go:206] created
clusterrolebinding.rbac.authorization.k8s.io/system:controller:certificate-controller
I0830 21:47:21.196736   18951 storage_rbac.go:237] created role.rbac.authorization.k8s.io/extension-apiserver-
authentication-reader in kube-system
I0830 21:47:21.236135   18951 storage_rbac.go:237] created
role.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-system
I0830 21:47:21.276356   18951 storage_rbac.go:237] created role.rbac.authorization.k8s.io/system:controller:cloud-
provider in kube-system
I0830 21:47:21.317528   18951 storage_rbac.go:237] created role.rbac.authorization.k8s.io/system:controller:token-
cleaner in kube-system
I0830 21:47:21.357986   18951 storage_rbac.go:237] created role.rbac.authorization.k8s.io/system::leader-locking-
kube-controller-manager in kube-system
I0830 21:47:21.397002   18951 storage_rbac.go:237] created role.rbac.authorization.k8s.io/system::leader-locking-
kube-scheduler in kube-system
I0830 21:47:21.437123   18951 storage_rbac.go:237] created
role.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-public
I0830 21:47:21.477507   18951 storage_rbac.go:267] created rolebinding.rbac.authorization.k8s.io/system::leader-
locking-kube-controller-manager in kube-system
I0830 21:47:21.518024   18951 storage_rbac.go:267] created rolebinding.rbac.authorization.k8s.io/system::leader-
locking-kube-scheduler in kube-system
I0830 21:47:21.556709   18951 storage_rbac.go:267] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-system
I0830 21:47:21.596359   18951 storage_rbac.go:267] created
rolebinding.rbac.authorization.k8s.io/system:controller:cloud-provider in kube-system
I0830 21:47:21.638493   18951 storage_rbac.go:267] created
rolebinding.rbac.authorization.k8s.io/system:controller:token-cleaner in kube-system
I0830 21:47:21.677353   18951 storage_rbac.go:267] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-public
...
```


© Copyright 2017 RX-M LLC

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

```
user@nodea:~$ curl -v https://172.16.151.203:443 \
--cacert ca.pem \
--key nodea-worker-key.pem \
--cert nodea-worker.pem && echo

* Rebuilt URL to: https://172.16.151.203:443/
*   Trying 172.16.151.203...
* Connected to 172.16.151.203 (172.16.151.203) port 443 (#0)
* found 1 certificates in ca.pem
* found 692 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: Thu, 31 Aug 2017 04:19:41 GMT
*        expire date: Fri, 31 Aug 2018 04:19:41 GMT
*        issuer: CN=kube-ca
*        compression: NULL
* ALPN, server accepted to use http/1.1
> GET / HTTP/1.1
> Host: 172.16.151.203
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Content-Type: text/plain
< X-Content-Type-Options: nosniff
< Date: Thu, 31 Aug 2017 04:49:16 GMT
< Content-Length: 33
<
* Connection #0 to host 172.16.151.203 left intact
User "nodea" cannot get path "/".

user@nodea:~$
```


© Copyright 2017 RX-M LLC

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. First we need to modify our existing config to use api server's insecure port for bootstrapping the first role.

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

The connection to the server 172.16.151.203:8080 was refused - did you specify the right host or port?

user@nodea:~$
```

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

user@nodea:~$
```

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

Switched to context "local".

user@nodea:~$
```

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

NAME         CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1       <none>        443/TCP   16h

user@nodea:~$
```


© Copyright 2017 RX-M LLC

Voila! 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:

```
user@nodea:~$ kubectl config set-cluster my-local \
--certificate-authority=ca.pem \
--server=https://172.16.151.203

Cluster "my-local" set.

user@nodea:~$
```

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

User "my-local-admin" set.

user@nodea:~$
```

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

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

Context "my-local-admin-ctx" created.

user@nodea:~$
```

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

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


© Copyright 2017 RX-M LLC

```
clusterrolebinding "kube-admin-binding" created

user@nodea:~$
```

Now we can activate the context:

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

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

user@nodea:~$
```

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

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

NAME      STATUS    AGE       VERSION
nodea     Ready     11m       v1.7.8+793658f2d7ca7

user@nodea:~$
```

Optional

If you really want to make sure what you did worked, you can reconfigure the `local` cluster to use your node's IP again, for example:

```
user@nodea:~$ kubectl config set-cluster local --server=http://172.16.151.203:8080

Cluster "local" set.

user@nodea:~$
```

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

Switched to context "local".
```


© Copyright 2017 RX-M LLC

```
user@nodea:~$
```

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

NAME      STATUS    AGE       VERSION
nodea     Ready     12m       v1.7.8+793658f2d7ca7

user@nodea:~$
```

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

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

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

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:

```
user@nodea:~$ vi svc-reader.yaml
user@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"]

user@nodea:~$
```


© Copyright 2017 RX-M LLC

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

role "svc-reader" created

user@nodea:~$
```

```
user@nodea:~$ kubectl get roles

NAME         AGE
svc-reader   8s

user@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:

```
user@nodea:~$ vi cluster-role.yaml
user@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"]

user@nodea:~$
```

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

clusterrole "pod-reader" created
```


© Copyright 2017 RX-M LLC

```
user@nodea:~$
```

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

NAME                                             AGE
admin                                            12m
cluster-admin                                    12m
edit                                             12m
pod-reader                                       5s
system:auth-delegator                            12m
system:basic-user                                12m
system:controller:attachdetach-controller        12m
system:controller:certificate-controller         12m
system:controller:cronjob-controller             12m
system:controller:daemon-set-controller          12m
system:controller:deployment-controller          12m
system:controller:disruption-controller          12m
system:controller:endpoint-controller            12m
system:controller:generic-garbage-collector      12m
system:controller:horizontal-pod-autoscaler      12m
system:controller:job-controller                 12m
system:controller:namespace-controller           12m
system:controller:node-controller                12m
system:controller:persistent-volume-binder       12m
system:controller:pod-garbage-collector          12m
system:controller:replicaset-controller          12m
system:controller:replication-controller         12m
system:controller:resourcequota-controller       12m
system:controller:route-controller               12m
system:controller:service-account-controller     12m
system:controller:service-controller             12m
system:controller:statefulset-controller         12m
system:controller:ttl-controller                 12m
system:discovery                                 12m
system:heapster                                  12m
system:kube-aggregator                           12m
system:kube-controller-manager                   12m
system:kube-dns                                  12m
system:kube-scheduler                            12m
system:node                                      12m
system:node-bootstrapper                         12m
```


© Copyright 2017 RX-M LLC

```
system:node-problem-detector                    12m
system:node-proxier                             12m
system:persistent-volume-provisioner            12m
view                                            12m

user@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:

```
user@nodea:~$ vi sbind.yaml
user@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

user@nodea:~$
```

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

rolebinding "read-svc" created

user@nodea:~$
```

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

NAME       AGE
read-svc   6s
```


© Copyright 2017 RX-M LLC

```
user@nodea:~$
```

Step 11 - Test your new role

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

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

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

user@nodea:~$
```

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

NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1     <none>        443/TCP   32m

user@nodea:~$
```

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

No resources found.

user@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:

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


© Copyright 2017 RX-M LLC

```
User "my-local-node" set.

user@nodea:~$
```

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

Context "my-local-node-ctx" created.

user@nodea:~$
```

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

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

user@nodea:~$
```

Now try to list services:

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

NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1     <none>        443/TCP   34m

user@nodea:~$
```

It works!

Now try to get rs-es:

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


© Copyright 2017 RX-M LLC

```
Error from server (Forbidden): User "nodea" cannot list replicasets.extensions in the namespace "default". (get
replicasets.extensions)

user@nodea:~$
```

Now try to get pods:

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

Error from server (Forbidden): User "nodea" cannot list pods in the namespace "default". (get pods)

user@nodea:~$
```

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

Add the pod reader cluster role to the nodea user:

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

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

user@nodea:~$
```

```
user@nodea:~$ vi cbind.yaml
user@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
```


© Copyright 2017 RX-M LLC

```
user@nodea:~$
```

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

clusterrolebinding "read-pods-cbind" created

user@nodea:~$
```

```
user@nodea:~$ kubectl get clusterrolebinding

NAME                                           AGE
cluster-admin                                  20m
kube-admin-binding                             12m
read-pods-cbind                                6s
system:basic-user                              20m
system:controller:attachdetach-controller      20m
system:controller:certificate-controller       20m
system:controller:cronjob-controller           20m
system:controller:daemon-set-controller        20m
system:controller:deployment-controller        20m
system:controller:disruption-controller        20m
system:controller:endpoint-controller          20m
system:controller:generic-garbage-collector    20m
system:controller:horizontal-pod-autoscaler    20m
system:controller:job-controller               20m
system:controller:namespace-controller         20m
system:controller:node-controller              20m
system:controller:persistent-volume-binder     20m
system:controller:pod-garbage-collector        20m
system:controller:replicaset-controller        20m
system:controller:replication-controller       20m
system:controller:resourcequota-controller     20m
system:controller:route-controller             20m
system:controller:service-account-controller   20m
system:controller:service-controller           20m
system:controller:statefulset-controller       20m
system:controller:ttl-controller               20m
system:discovery                               20m
system:kube-controller-manager                 20m
```


© Copyright 2017 RX-M LLC

```
system:kube-dns                                 20m
system:kube-scheduler                           20m
system:node                                     20m
system:node-proxier                             20m

user@nodea:~$
```

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

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

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

user@nodea:~$
```

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

No resources found.

user@nodea:~$
```

Now lets try a negative result:

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

Error from server (Forbidden): User "nodea" cannot list replicasets.extensions in the namespace "default". (get
replicasets.extensions)

user@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 TLS and Auth lab!!

© Copyright 2017 RX-M LLC

Copyright (c) 2013-2017 RX-M LLC, Cloud Native Consulting, all rights reserved

© Copyright 2017 RX-M LLC