Using Kubernetes in the DAQ

Access control

Gordon Crone

University College London

CCM k8s chat 8th Dec 2021

Access Control with k8s

UCL

Controlling access to host resources from Pods

- Currently done via a PodSecurityPolicy (PSP) but
  PodSecurityPolicy is deprecated as of Kubernetes v1.21, and will be removed in v1.25.
- In the meantime recommended to keep to the Pod Security Standards (Restricted, Baseline, and Privileged)
- Only Privileged allows “HostNetwork: true” which we probably need
- Can define a custom PSP resource
- Associate PSPs with user accounts and service accounts by allowing accounts to ’use’ the resource

Access Control with k8s

UCL

Authentication within a Pod

- User has access to secrets belonging to ServiceAccount
- Can use bearer token

```
bash-4.2$ curl -k https://kubernetes/api/v1
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/api/v1\"",
  "reason": "Forbidden",
  "details": {

  },
  "code": 403
}
```

Access Control with k8s

UCL

Authentication within a Pod

- User has access to secrets belonging to ServiceAccount
- Can use bearer token

```
bash-4.2$ curl -k https://kubernetes/api/v1  --header \
"Authorization: Bearer $(cat /run/secrets/kubernetes.io/serviceaccount/token)"
{
  "kind": "APIResourceList",
  "groupVersion": "v1",
  "resources": [
    {
      "name": "bindings",
      "singularName": "",
      "namespaced": true,
      "kind": "Binding",
      "verbs": [
        "create"
      ]
...
```

Access Control with k8s

UCL

Authentication within a Pod

The "Official Python client library" handles this in `incluster_config.py`

incluster_config.py

```
SERVICE_TOKEN_FILENAME = "/var/run/secrets/kubernetes.io/serviceaccount/token"
SERVICE_CERT_FILENAME = "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt"
...
def load_incluster_config(client_configuration=None, try_refresh_token=True):
    """
    Use the service account kubernetes gives to pods to connect to kubernetes
    cluster. It's intended for clients that expect to be running inside a pod
    running on kubernetes. It will raise an exception if called from a process
    not running in a kubernetes environment."""
    InClusterConfigLoader(
        token_filename=SERVICE_TOKEN_FILENAME,
        cert_filename=SERVICE_CERT_FILENAME,
        try_refresh_token=try_refresh_token).load_and_set(client_configuration)
```

Access Control with k8s

UCL

Authentication from web application

- OpenID Connect Tokens looks like the way to go
- Well supported and documented by CERN Authorization Service (SSO)
  - https://auth.docs.cern.ch/user-documentation/oidc/oidc/
- Fermilab Federated Identity Project will use OIDC tokens
  - https://sciauth.org/workshop/2021/Federation_Status_WoTBAn_10_18_21.pdf

Backup Slides

Access Control with k8s

UCL

Controlling access to host resources from Pods

- From Pod Security Standards FAQ

  Why isn't there a profile between privileged and baseline?
  Privileges required above the baseline policy are typically very application specific, so we do not offer a standard profile in this niche. This is not to say that the privileged profile should always be used in this case, but that policies in this space need to be defined on a case-by-case basis.

Access Control with k8s

Resource allocation

Prevent multiple Pods (partitions...) from accessing FLX cards

Define resource

```
curl --header "Content-Type: application/json-patch+json" \
  --request PATCH \
  --data '[{"op": "add",
   "path": "/status/capacity/dune.daq~1flxcard-apa001","value": "1"}]' \
  http://localhost:8001/api/v1/nodes/daq6.hep.ucl.ac.uk/status
```

In Deployment

Allocate resource

```
spec:
  containers:
    resources:
      requests:
        dune.daq/flxcard-apa001: 1
      limits:
        dune.daq/flxcard-apa001: 1
```

- Request 1 unit of flxcard-apa001
- Pod will only be scheduled on a node with 1 available

'Proper' implementation of Kubernetes on multiple nodes

- Single control plane node and 2 'worker' nodes
- Using cri-o as the Container Runtime Interface
- Using Calico as the Pod network add-on
  - Nearly works but filed to get networking between Pods on different nodes to work
  - Eventually abandoned Calico
- Use Flannel as the Pod network add-on
  - Works out of the box (after cleaning up stuff left behind by Calico)

```
gjc> kubectl get nodes
NAME                  STATUS   ROLES                  AGE    VERSION
daq2.hep.ucl.ac.uk    Ready    <none>                 7d5h   v1.22.4
daq6.hep.ucl.ac.uk    Ready    <none>                 7d5h   v1.22.4
daq7.hep.ucl.ac.uk    Ready    control-plane,master   7d5h   v1.22.4
```

Services listed in the kube-dns

- Should make configuration simpler
- Applies to the control plane network but what about data network?
- Hang on, can't even see the data network in a pod

```
gjc> kubectl exec -i -t gordon-test -- ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.0.1.2  netmask 255.255.255.0  broadcast 10.0.1.255
...
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
```

Set '`hostNetwork: true`' in Pod's spec

- That works, can see all the host's network interfaces

```
gjc> kubectl exec -i -t iperf -- ifconfig
bond0: flags=5187<UP,BROADCAST,RUNNING,MASTER,MULTICAST>  mtu 1500
        inet 192.168.203.6  netmask 255.255.255.0  broadcast 192.168.203.255

cni0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.0.1.1  netmask 255.255.255.0  broadcast 10.0.1.255
```

Playing with k8s
Rabbit holes I've been down

UCL

Set 'hostNetwork: true' in Pod's spec

- But is this what we really want?
  - For example we lose the advantage of port numbering local to Pod

Try using Multus to add additional networks to Pod

- Works, but have to set up IP addresses for additional NICs
- Maybe complicates configuration too much
  - Better to go with `hostNetwork`

Using cvmfs config from Pocket

- Only works on single node (whichever it gets scheduled on)
- Change from Deployment to DaemonSet to make available everywhere