



// SECURE BY DEFAULT? PRAGMATICALLY IMPROVE APP SECURITY USING K8S BUILT-INS

Johannes Schnatterer

Clouddogu GmbH

 @jschnatterer

Version: 202010201514-8c65eaa

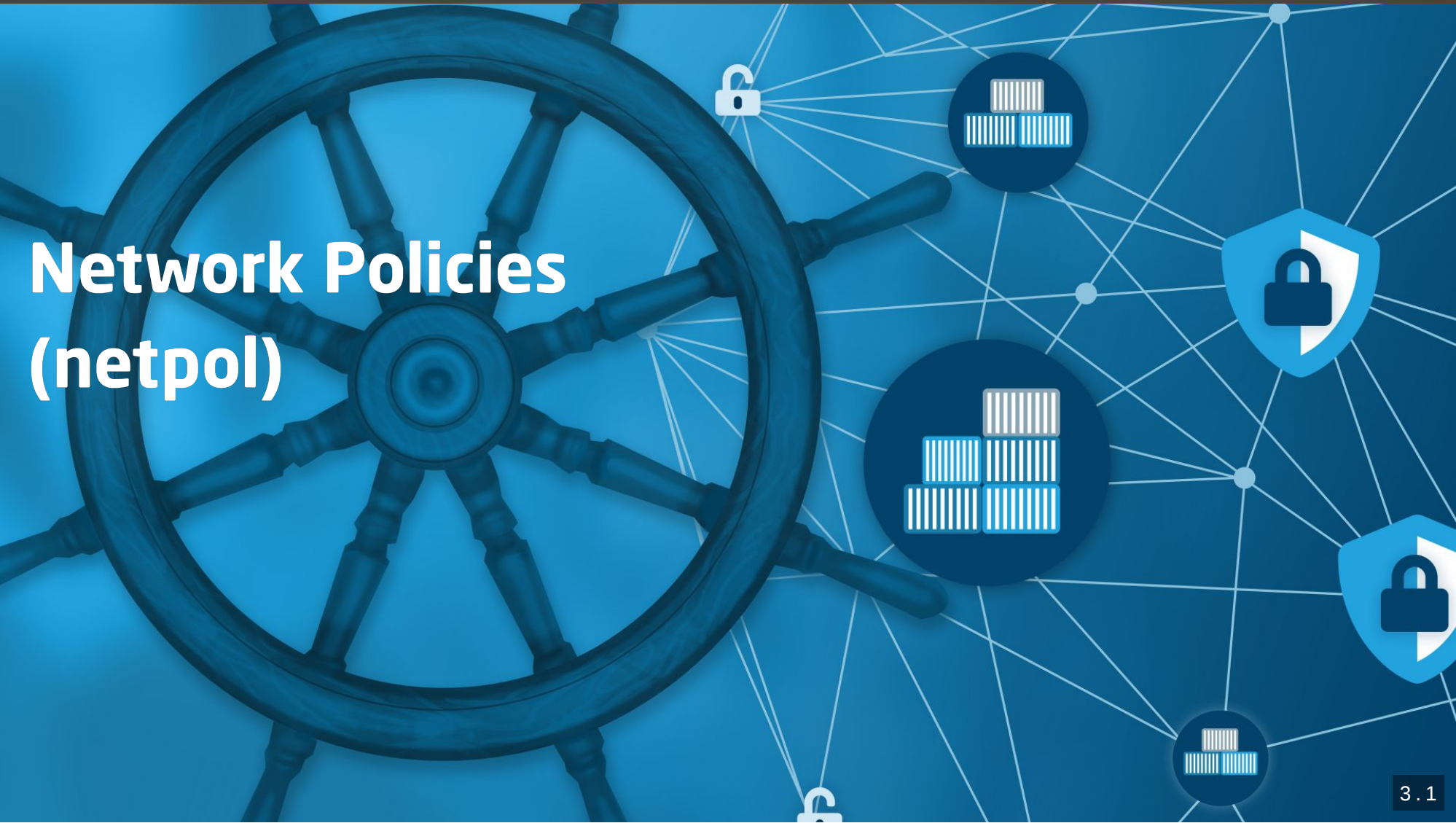
K8s built-in security mechanisms

- Network Policies
- Security Context
- Pod Security Policies


Plenty of Options

- Secure by default?
- How to improve pragmatically?

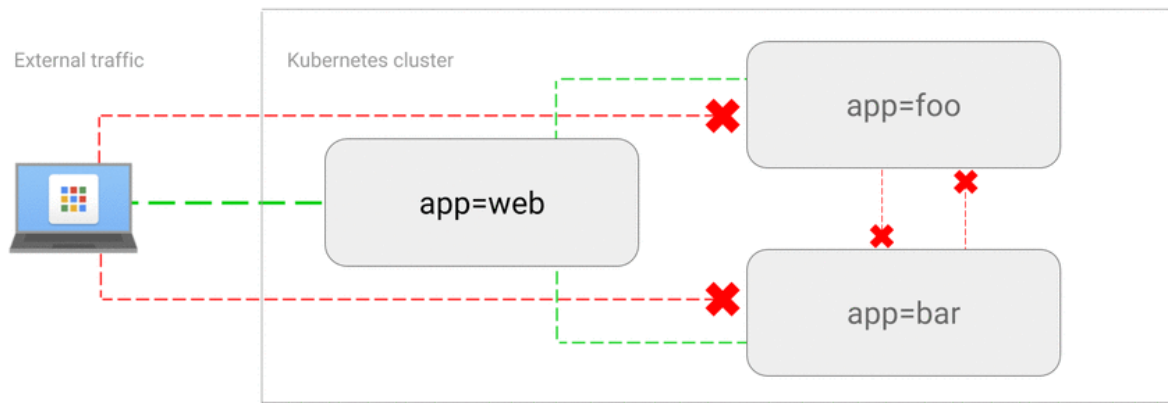
Network Policies (netpol)



A “firewall” for communication between pods.

- Applied to pods
 - within namespace
 - via labels
- Ingress / egress
 - to/from pods (in namespaces) or CIDRs (egress only)
 - for specific ports (optional)
- Enforced by the CNI Plugin (e.g. Calico)
-  No Network Policies: All traffic allowed

📌 Helpful to get started



-  <https://github.com/ahmetb/kubernetes-network-policy-recipes>
- Interactively describes what a netpol does:

```
kubectl describe netpol <name>
```

Recommendation: Allow selected ingress traffic only

In every namespace except kube-system:

- Deny ingress between pods,
- then allow specific routes only.

Advanced: ingress to kube-system

⚠ Might stop the apps in your cluster from working

Don't forget to:

- Allow external access to ingress controller
- Allow access to DNS from every namespace
- Allow DNS egress to the outside (if needed)

Advanced: egress

- Verbose solution:
 - Deny egress between pods,
 - then allow specific routes,
 - repeating all ingress rules. 🙄
- More pragmatic solution:
 - Allow only egress within the cluster,
 - then allow specific pods that need access to internet.

Net pol pitfalls

- Allow monitoring tools (e.g. Prometheus)
- Restart might be necessary (e.g. Prometheus)
- No labels on namespaces by default
- Allowing egress to API server difficult
- Policies might not be supported by CNI Plugin.

➡ Testing!

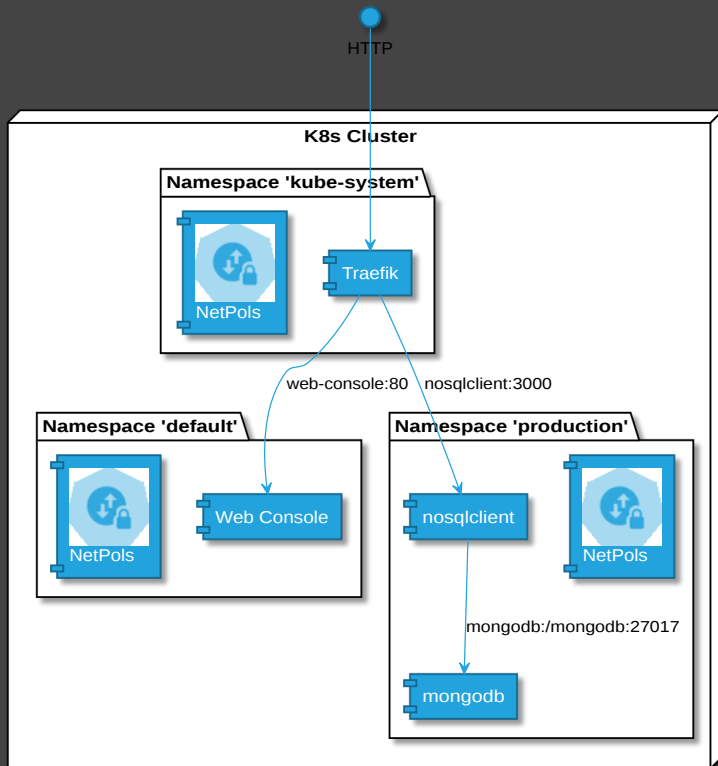
 <https://www.inovex.de/blog/test-kubernetes-network-policies/>

 <https://github.com/inovex/illumination>

More Features?

- Proprietary extensions of CNI Plugin (e.g. cilium or calico)
- Service Meshes: similar features, also work with multiple clusters
 - ➔ different strengths, support each other (ISO/OSI Layer 7 vs 3/4)
 - 🌐 <https://istio.io/blog/2017/0.1-using-network-policy/>

Demo



- nosqlclient
- web-console

Wrap-Up: Network Policies

My recommendations:

- In non-kube - system namespaces: allow selected ingress traffic only.
- Use with care
 - limiting ingress traffic in kube - system
 - egress limiting for cluster-external traffic

Security Context



- Security Context: Defines security parameters *per pod/container*
 - ➔ container runtime
- ↔ Cluster-wide security parameters: See Pod Security Policies

Recommendations per Container

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    seccomp.security.alpha.kubernetes.io/pod: runtime/default # k8s <= 1.18
spec:
  containers:
    - name: restricted
      securityContext:
        runAsNonRoot: true
        runAsUser: 100000
        runAsGroup: 100000
        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: true
        seccompProfile: # k8s >= 1.19
          type: RuntimeDefault
      capabilities:
        drop:
          - ALL
  enableServiceLinks: false
  automountServiceAccountToken: false # When not communicating with API Server
```


Recommendation per Container in Detail


Enable seccomp

- Enables e.g. docker's seccomp default profile that block 44/~300 Syscalls
- 🔥 Has mitigated Kernel vulns in past and might in future 🧪
🌐 <https://docs.docker.com/engine/security/non-events/>
- See also k8s security audit:
🌐 <https://www.cncf.io/blog/2019/08/06/open-sourcing-the-kubernetes-security-audit/>

Run as unprivileged user

- `runAsNonRoot: true`

Container is not started when the user is root

- `runAsUser` and `runAsGroup > 10000`
 - 🔥 Reduces risk to run as user existing on host
 - 🔥 In case of container escape UID/GID does not have privileges on host
- 🔥 E.g. mitigates vuln in runc (used by Docker among others)
 <https://kubernetes.io/blog/2019/02/11/runc-and-cve-2019-5736/>

No Privilege escalation

- Container can't increase privileges
- 🔥 E.g. sudo, setuid, Kernel vulnerabilities

Read-only root file system

- Starts container without read-write layer
- Writing only allowed in volumes
- 🔥 Config or code within the container cannot be manipulated

Drop Capabilities

- Drops even the default caps:



<https://github.com/moby/moby/blob/3152f94/oci/caps/defaults.go>

- 🔥 E.g. Mitigates CapNetRaw attack - DNS Spoofing on Kubernetes Clusters

 <https://blog.aquasec.com/dns-spoofing-kubernetes-clusters>

Bonus: No Services in Environment

- By default: Each K8s service written to each container's env vars
 - ➔ Docker Link legacy, no longer needed
- 🔥 But convenient info for attacker where to go next

Bonus: Disable access to K8s API

- SA Token in every pod for api-server authn

```
curl --cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt \  
-H "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \  
https://${KUBERNETES_SERVICE_HOST}/api/v1/
```

- If not needed, disable!
- No authentication possible
- 🔥 Lesser risk of security misconfig or vulns in authz

Security context pitfalls

Read-only root file system

Application might need temp folder to write to

- Run image locally using docker, access app
 - 📌 Run automated e2e/integration tests
- Review container's read-write layer via

```
docker diff <containerName>
```

- Mount folders as `emptyDir` volumes in pod

Drop Capabilities



Some images require capabilities

- Find out needed Caps locally:

```
docker run --rm --cap-drop ALL <image>
# Check error
docker run --rm --cap-drop ALL --cap-add CAP_CHOWN <image>
# Keep adding caps until no more error
```

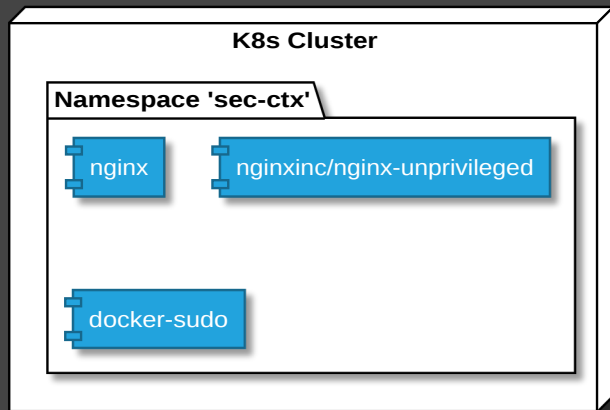
- Add necessary caps to k8s securityContext
- Alternative: Find image with same app that does not require caps, e.g. `nginxinc/nginx-unprivileged`

Run as unprivileged user

- Some official images run as root by default.
 - Find a **trusted** image that does not run as root
e.g. for mongo or postgres:
 <https://hub.docker.com/r/bitnami/>
- Create your own non-root image
(potentially basing on original image)
e.g. nginx:  <https://github.com/schnatterer/nginx-unpriv>

- UID 100000 might not have permissions. Solutions:
 - Init Container sets permissions for volume
 - Permissions in image ➡ `chmod/chown` in `Dockerfile`
 - Run in root Group - GID 0
- 🌐 https://docs.openshift.com/container-platform/4.3/openshift_images/create-images.html#images-create-guide-openshift_create-images

Demo



Wrap-Up: Security Context

My recommendations:

- Start with least privilege
- Only differ if there's absolutely no other way

Pod Security Policies (PSP)




- enforces security context settings cluster-wide
- additional options enforcing secure defaults
- more effort than security context and different syntax 🙄
- future from K8s 1.22 vague 😞

 <https://github.com/kubernetes/enhancements/issues/5>

➡ Still only built-in solution for cluster-wide security settings

Recommendation

 <https://github.com/cloudogu/k8s-security-demos/blob/master/4-pod-security-policies/demo/01-psp-restrictive.yaml>

Too much ground to cover for 45 min!



- 🎥 <https://youtu.be/YlvdFE1Rsml?t=3092> 🇩🇪 including Demo
- 🌐 <https://cloudogu.com/en/blog/k8s-app-ops-part-5-pod-security-policies-1> 🇬🇧 🇩🇪

Summary

- Don't allow arbitrary connections between pods, e.g. via NetPols
- Start with least privilege for your containers
 - using either Security Context or
 - PSP

Johannes Schnatterer

Cloudogu GmbH

 cloudogu.com/schulungen

K8s AppOps security series on JavaSPEKTRUM 05/2019+

See also  cloudogu.com/blog/tag/k8s-security

 [@cloudogu](https://twitter.com/cloudogu)

 [@jschnatterer](https://twitter.com/jschnatterer)

Demo Source:  github.com/cloudogu/k8s-security-demos

