



cloudogu



# // 3 THINGS EVERY DEVELOPER SHOULD KNOW ABOUT K8S SECURITY

JOHANNES SCHNATTERER

*CLOUDOGU GMBH*

@JSCHNATTERER

VERSION: 201909181115-46DF801

# Requirements for dev and ops



🌐 <https://iso25000.com/images/figures/en/iso25010.png>

# End of Budget / Close to Deadline



# End of Budget / Close to Deadline

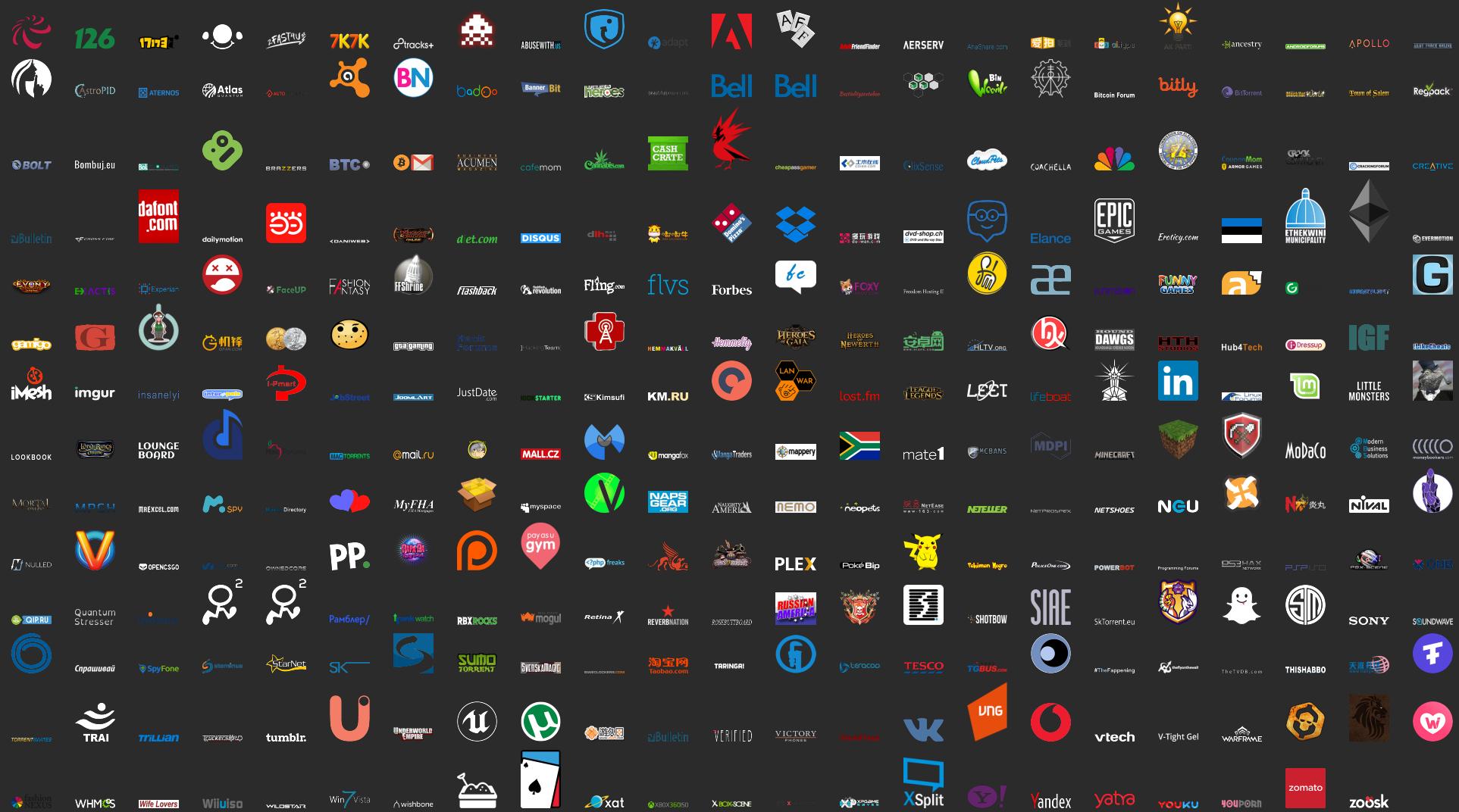


**Don't try this at home!**

# Otherwise

You might make the news...

Like so many others





Plenty of security options

securityContext`runAsNonRoot` `runAsUser` `privileged` `procMount`  
`allowPrivilegeEscalation` `readOnlyRootFilesystem` PodSecurityPolicy  
RBAC `seccomp` Linux Capabilities `AppArmor` SELinux `Falco` Open  
Policy Agent `NetworkPolicy` `gVisor` Kata Containers Nabla Containers Service  
Mesh mTLS `KubeSec` `KubeBench` kubetest Clair Vault Grafeas notary Bastion Host  
Certificate Rotation Threat detection SecOps







## **3 things every developer should know about K8s security**

- a very opinionated list of actions that make a huge difference with manageable effort
- distilled from the experience of the last years developing and operating apps on k8s

# 1. RBAC



🌐 <https://memegenerator.net/instance/83566913/homer-simpson-boring>

- RBAC active by default since K8s 1.6
- ... but not if you migrated!

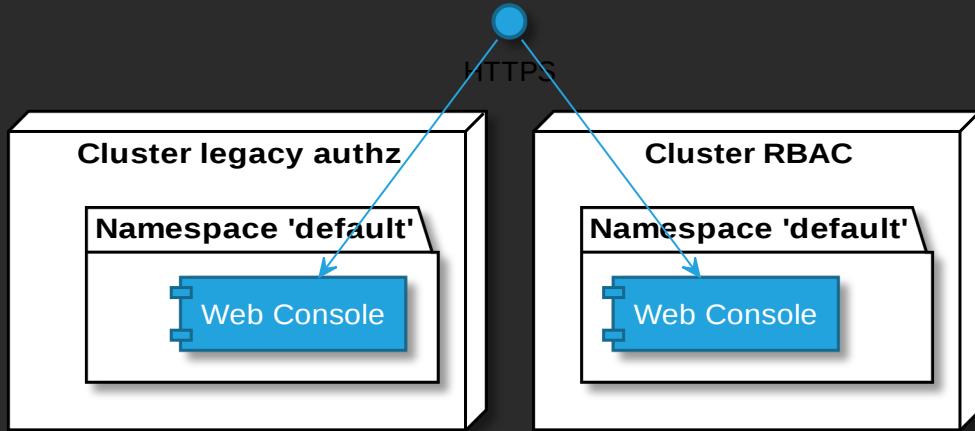
- Every Container is mounted the token of its service account at  
`/var/run/secrets/kubernetes.io/serviceaccount/token`
  - With RBAC the default service account is only authorized to read publicly accessible API info
  - ⚠️ With legacy authz the default service account is cluster admin
- You can test if your pod is authorized by executing the following in it:

```
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/secrets
```

- If a pod does not need access to K8s API, mounting the token can be disabled in the pod spec:  
`automountServiceAccountToken: false`



# Demo



- legacy-authz
- RBAC

## 2. Network Policies (netpol)

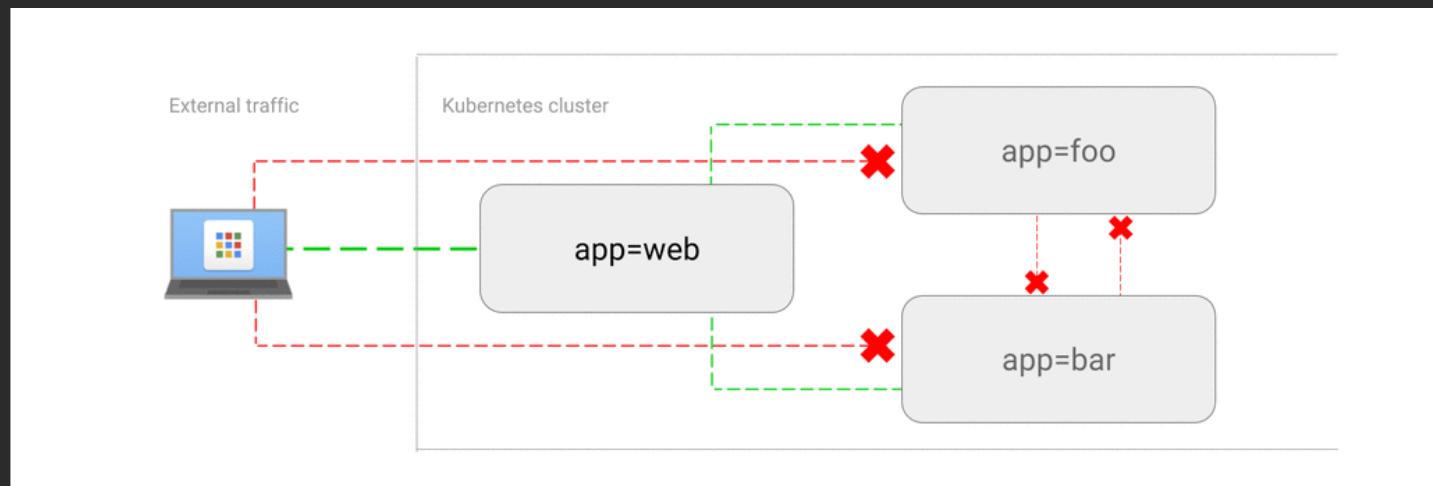
A kind of firewall for communication between pods.

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

# Recommendation: Whitelist ingress traffic

In every namespace except kube-system:

- Deny all ingress traffic between pods ...
- ... and then whitelist all allowed routes.



 <https://github.com/ahmetb/kubernetes-network-policy-recipes>

See also: Securing Cluster Networking with Network Policies - Ahmet Balkan

 <https://www.youtube.com/watch?v=3gGpMmYeEO8>

## Advanced: ingress to kube-system namespace

 You might stop the apps in your cluster from working

For example, don't forget to:

- Allow external access to ingress controller  
(otherwise no more external access on any cluster resource)
- Allow access to kube-dns/core-dns to every namespace  
(otherwise no more service discovery by name)

## Advanced: egress

- Verbose solution:
  - Deny all egress traffic between pods ...
  - ... and then whitelist all allowed routes...
  - ... repeating all ingress rules. 😕
- More pragmatic solution:
  - Allow only egress traffic within the cluster...
  - ... and then whitelist pods that need access to the internet.

## Net pol pitfalls

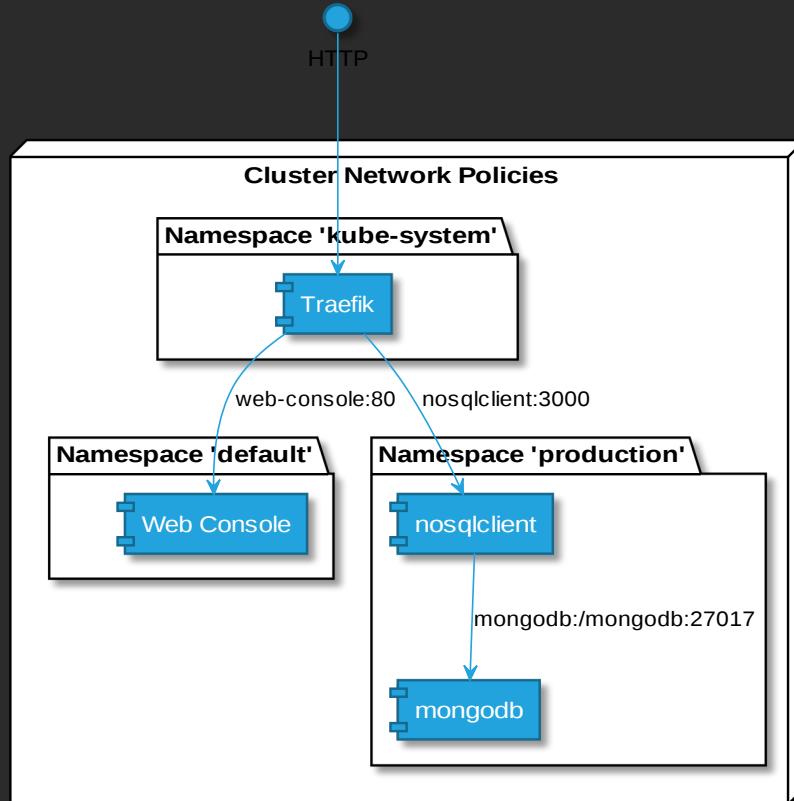
- Don't forget to whitelist your monitoring tools (e.g. Prometheus)
- A restart of the pods might be necessary for the netpol to become effective (e.g. Prometheus)
- In order to match namespaces, labels need to be added to the namespaces, e.g.

```
kubectl label namespace/kube-system namespace=kube-system
```

- Matching both pods and namespace is only possible from k8s 1.11+
- Restricting kube-system might be more of a challenge (DNS, ingress controller)
- egress rules are more recent feature than ingress rules and seem less sophisticated
- Policies might not be supported by CNI Plugin.  
Make sure to test them!  
 <https://www.inovex.de/blog/test-kubernetes-network-policies/>
- On GKE: "at least 2 nodes of type n1-standard-1" are required



# Demo



- nosqlclient
- web-console



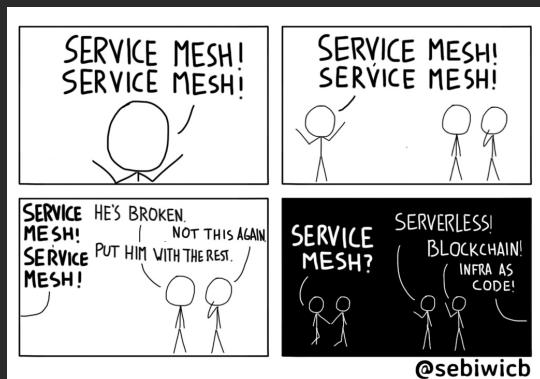
# Wrap-Up: Network Policies

- My recommendations
  - Definitely use DENY all ingress rule in non-kube-system namespaces
  - Use with care
    - rules in kube-system
    - egress rules
-  Helpful to get started - describe what a netpol does:  
`kubectl describe netpol <name>`
- Limitations:
  - netpols will not work in multi-cloud / cluster-federation scenarios
  - Service Meshes provides similar features and work also with multiple clusters



# 🎁 Wrap-Up: Network Policies

- My recommendations
  - Definitely use DENY all ingress rule in non-kube-system namespaces
  - Use with care
    - rules in kube-system
    - egress rules
- ⚡ Helpful to get started - describe what a netpol does:  
`kubectl describe netpol <name>`
- Limitations:
  - netpols will not work in multi-cloud / cluster-federation scenarios
  - Service Meshes provides similar features and work also with multiple clusters



🌐 <https://twitter.com/sebiwichb/status/962963928484630530>



### 3. Security Context

Defines privilege and access control settings for a Pod or Container

🌐 <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>

# Recommendation per Container

- `allowPrivilegeEscalation: false`
  - mitigates a process within the container from gaining higher privileges than its parent (the container process)
  - E.g. sudo, setuid, Kernel vulnerabilities
- `runAsNonRoot` - Container is not started when the user is root
- `readOnlyRootFilesystem: true`
  - Mounts the whole file system in the container read-only. Writing only allowed in volumes.
  - Makes sure that config or code within the container cannot be manipulated.
  - It's also more efficient (no CoW).

```
apiVersion: v1
kind: Pod
#...
spec:
  containers:
  - name: cntnr
    securityContext:
      runAsNonRoot: true
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
```

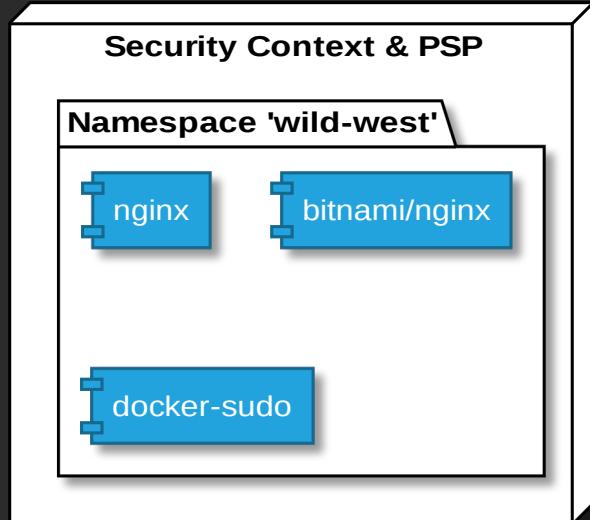
There is also a `securityContext` on pod level, but those settings cannot be applied there.

## Security context pitfalls

- `readOnlyRootFilesystem` - most applications need temp folders to write to
    - Run image locally using docker, access app ( run automated e2e/integration tests)
    - Then use `docker diff` to see a diff between container layer and image
    - and mount all folders listed there as `emptyDir` volumes in your pod
  - `runAsNonRoot`
    - Non-root verification only supports numeric user. 
    - Either set e.g.
      - `runAsUser: 1000` in `securityContext` of pod or
      - `USER 1000` in `Dockerfile` of image.
    - Some official images run as root by default.
- Possible solutions:
- Find a **trusted** image that does not run as root
    - e.g. for nginx, or postgres:  <https://hub.docker.com/r/bitnami/>
    - Derive from the original image and create your own non-root image
      - e.g. nginx:  <https://github.com/schnatterer/nginx-unpriv>



# Demo





## Wrap-Up: Security Context

My recommendations

- Security Context
  - Start with `readOnlyRootFilesystem`, `runAsNonRoot` and `allowPrivilegeEscalation: false`
  - Only differ if there's absolutely no other way
- BTW - you can enforce Security Context Settings by using Pod Security Policies. However, those cause a lot more effort to maintain.

# Summary

IMHO ever person working with k8s should *at least* adhere to the following:

- Enable RBAC!
- Don't allow arbitrary connection between pods.  
(e.g use Network Policies to whitelist ingresses)
- Don't allow privilege escalation via the security context of each container
- Try to run your containers
  - as non-root user and
  - with a read-only file system.



## Johannes Schnatterer

*Cloudogu GmbH*

🌐 <https://cloudogu.com/schulungen>

🌐 <https://cloudogu.com/blog>

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

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

Demo Source: 🐳 <https://github.com/cloudogu/k8s-security-demos>