// GOOD PRACTICES FOR SECURE KUBERNETES APPOPS

JOHANNES SCHNATTERER

CLOUDOGU GMBH

VERSION: 202003191825-5876AE2

Outline

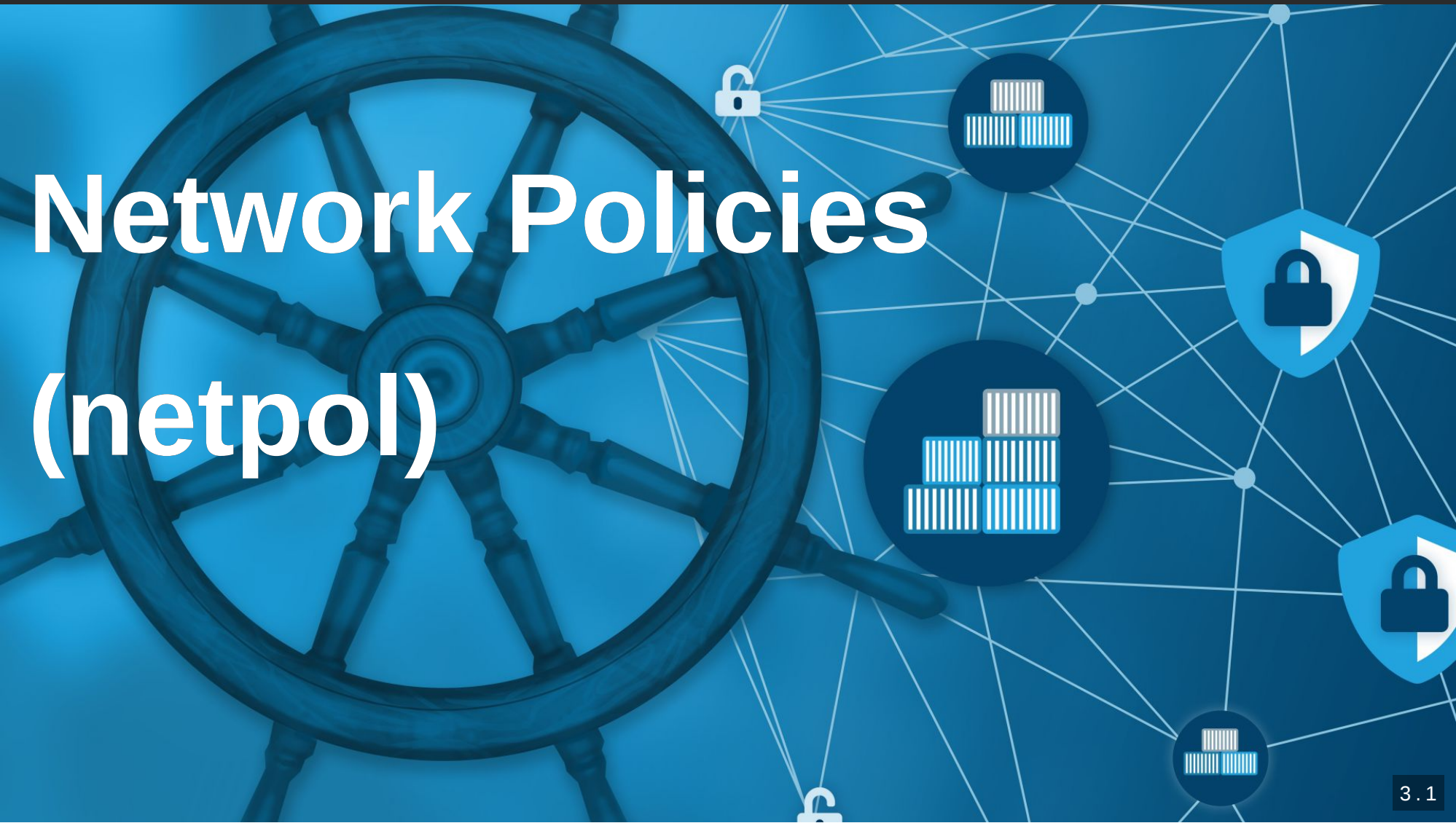
How to improve application security using Kubernetes security built-ins *pragmatically*

K8s built-in security mechanisms

- Network Policies
- Security Context
- Pod Security Policies

Plenty of Options

- Secure by default?
- How to improve pragmatically?

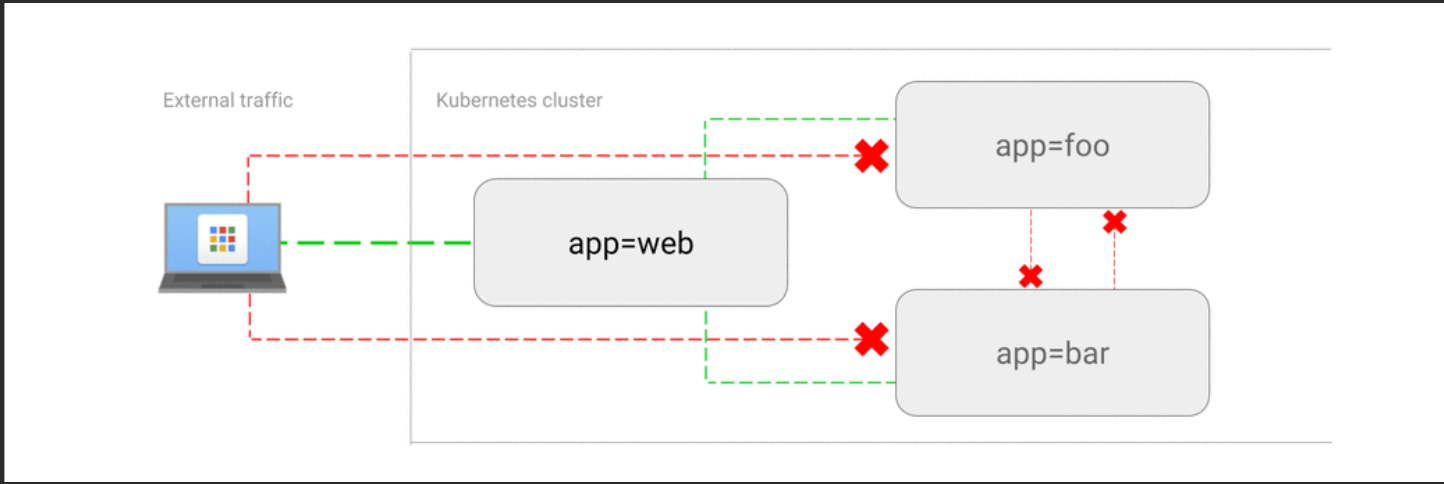


Network Policies (netpol)

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
- Securing Cluster Networking with Network Policies - Ahmet Balkan
  https://www.youtube.com/watch?v=3gGpMmYeEO8
- Interactively describes what a netpol does:

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

Recommendation: Whitelist ingress traffic

In every namespace except `kube-system`:

- Deny ingress between pods,
- then whitelist all allowed routes.

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 kube-dns/core-dns to every namespace

Advanced: egress

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

Net pol pitfalls

- Whitelisting monitoring tools (e.g. Prometheus)
- Restart might be necessary (e.g. Prometheus)
- No labels on namespaces by default
- `egress` more recent than `ingress` rules and less sophisticated
- Policies might not be supported by CNI Plugin.

  Testing!

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

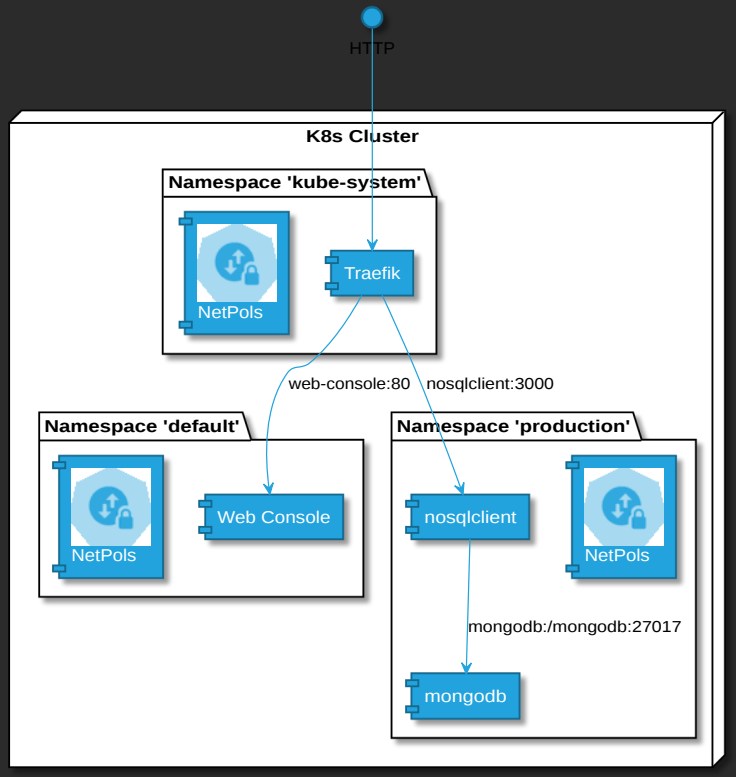
More Features?

- Proprietary extensions of CNI Plugin (e.g. cilium or calico)
- Service Meshes: similar features, also work with multiple clusters

  ➡️ different strengths, support each other

  🌐 https://istio.io/blog/2017/0.1-using-network-policy/

Demo



- nosqlclient
- web-console

🎁 Wrap-Up: Network Policies

My recommendations:

- Ingress whitelisting in non-`kube-system` namespaces
- Use with care
  - whitelisting in `kube-system`
  - `egress` whitelisting for cluster-external traffic


Security Context

- Security Context: Defines security parameters *per pod/container*
  ➡️ container runtime
- 📌 Secure Pods - Tim Allclair
  🎥 https://www.youtube.com/watch?v=GLwmJh-j3rs
- ↔️ Cluster-wide security parameters: See Pod Security Policies

Recommendations per Container

```yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    seccomp.security.alpha.kubernetes.io/pod: runtime/default
spec:
  containers:
  - name: restricted
    securityContext:
      runAsNonRoot: true
      runAsUser: 100000
      runAsGroup: 100000
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
      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 resource
- 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
- Non-root verification only supports numeric user. 🙄
  - `runAsUser: 100000` in `securityContext` of pod or
  - `USER 100000` in `Dockerfile` of image.

- UID 100000 might not have permissions. Solutions:
  - Init Container sets permissions for PVCs
  - 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

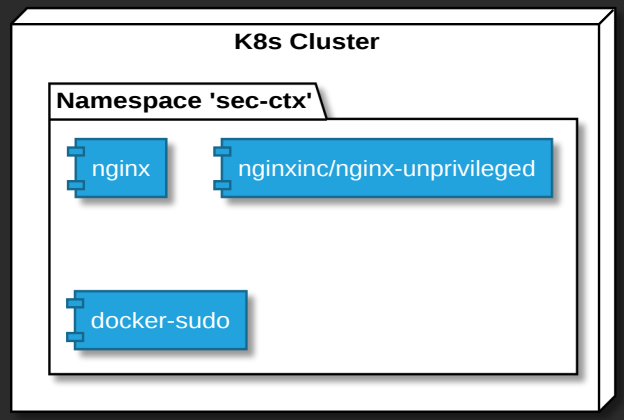
Tools

Find out if your cluster adheres to these and other good security practices:

- controlplaneio/kubesec - managable amount of checks
- Shopify/kubeaudit
  - a whole lot of checks,
  - even deny all ingress and egress NetPols and AppArmor Annotations

➡ Be prepared for a lot of findings
➡ Create your own good practices

Demo


K8s Cluster
Namespace 'sec-ctx'
nginx
nginxinc/nginx-unprivileged
docker-sudo

🎁 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 cluster-wide
- additional options enforcing secure defaults
- more effort than security context and different syntax 😕

➡️ Still highly recommended!

Recommendations

- Same as Security Context
- Plus: Enforce secure defaults.

  Block pods from
  - entering node's Linux namespaces, e.g. net, PID (includes binding ports to nodes directly),
  - mounting arbitrary host paths (from node) (includes docker socket),
  - starting `privileged` containers,
  - changing apparmor profile

Security Context Recommendations

PSP

```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  annotations:
    seccomp.security.alpha.kubernetes.io/defaultProfileName: runtime/default
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: runtime/default
spec:
  requiredDropCapabilities:
    - ALL
  allowedCapabilities: []
  defaultAllowPrivilegeEscalation: false
  allowPrivilegeEscalation: false
  readOnlyRootFilesystem: true
  runAsUser: # Same for runAsGroup, supplementalGroups, fsGroup
    rule: MustRunAs
    ranges:
      - min: 100000
        max: 999999
```

Additional Recommendations

```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  annotations:
    apparmor.security.beta.kubernetes.io/defaultProfileName: runtime/default
    apparmor.security.beta.kubernetes.io/allowedProfileNames: runtime/default
spec:
  hostIPC: false
  hostPID: false
  hostNetwork: false
  hostPorts: []
  privileged: false
  allowedHostPaths: []
  volumes:
    - configMap
    - emptyDir
    - projected
    - secret
    - downwardAPI
    - persistentVolumeClaim
```

Usage

1. Activate Admission controler via API-Server
   (also necessary for most managed k8s)
2. Define PSP (YAML)
3. Activate via RBAC

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

Activation via RBAC

🚧 PSP pitfalls

- Loose coupling in RBAC ➡️ fail late with typos
- ⚠️ *Admission*Controller
  - only evaluates Pods before starting
  - if not active ➡️ PSP are ignored
  - if active but no PSP defined ➡️ *no* pod can be started
- Different PSP API group in `(Cluster)Role`
  - < 1.16: `apiGroups [ extensions ]`
  - ≥ 1.16: `apiGroups [ policy ]`

📌 PSP Debugging Hints

```
# Query active PSP
kubectl get pod <POD> -o jsonpath='{.metadata.annotations.kubernetes\.io/psp}'
# Check authorization
kubectl auth can-i use psp/privileged --as=system:serviceaccount:<NS>:<SA>
# Show which SA's are authorized (kubectl plugin)
kubectl who-can use psp/<PSP>
# Show roles of a SA (kubectl plugin)
kubectl rbac-lookup <SA> # e.g. subject = sa name
```

PSP Limitations

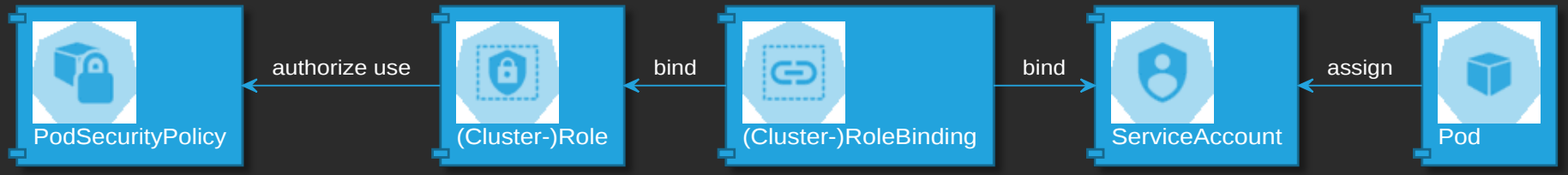
- Unavailable options in PSPs
  - `enableServiceLinks: false`
  - `automountServiceAccountToken: false`
- Future of PSPs uncertain

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

  ➡️ Still easiest way for cluster-wide least privilege

What if pod requires more privileges?

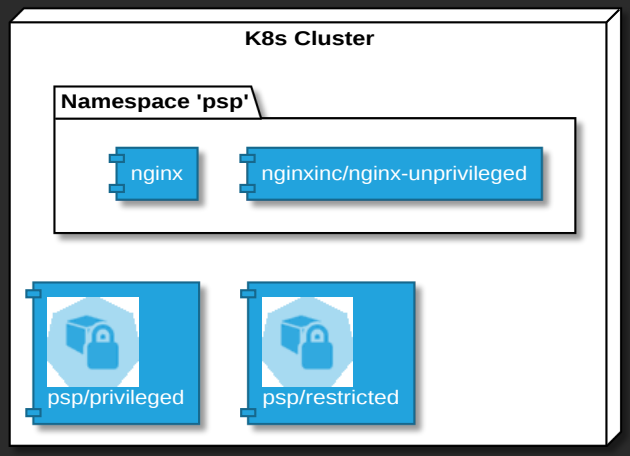
"Whitelisting" via RBAC.



1. Duplicate least privilege PSP
2. Grant required privilege in new PSP
3. Allow PSP via a Role (namespaced)
4. Create ServiceAccount
5. Create RoleBinding
6. Assign ServiceAccount to Pod

https://github.com/cloudogu/k8s-security-demos/blob/master/4-pod-security-policies/demo/02a-psp-whitelist.yaml

Demo


K8s Cluster
Namespace 'psp'
nginx
nginxinc/nginx-unprivileged
psp/privileged
psp/restricted

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

@jschnatterer

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