KubeCon | CloudNativeCon

Europe 2023

Least Privilege Containers

Keeping a bad day from getting worse

Greg Castle
@mrgcastle
@gregcastle@infosec.exchange
GKE Security, Google Cloud

Vinayak Goyal
@vglovespizza
GKE Security, Google Cloud

Slides: sched.co/1HyX4

Don’t Run Containers As Root

We've been saying this for a long time....

Is it working?

@lizrice 2018 "don't run as root" keynote

No

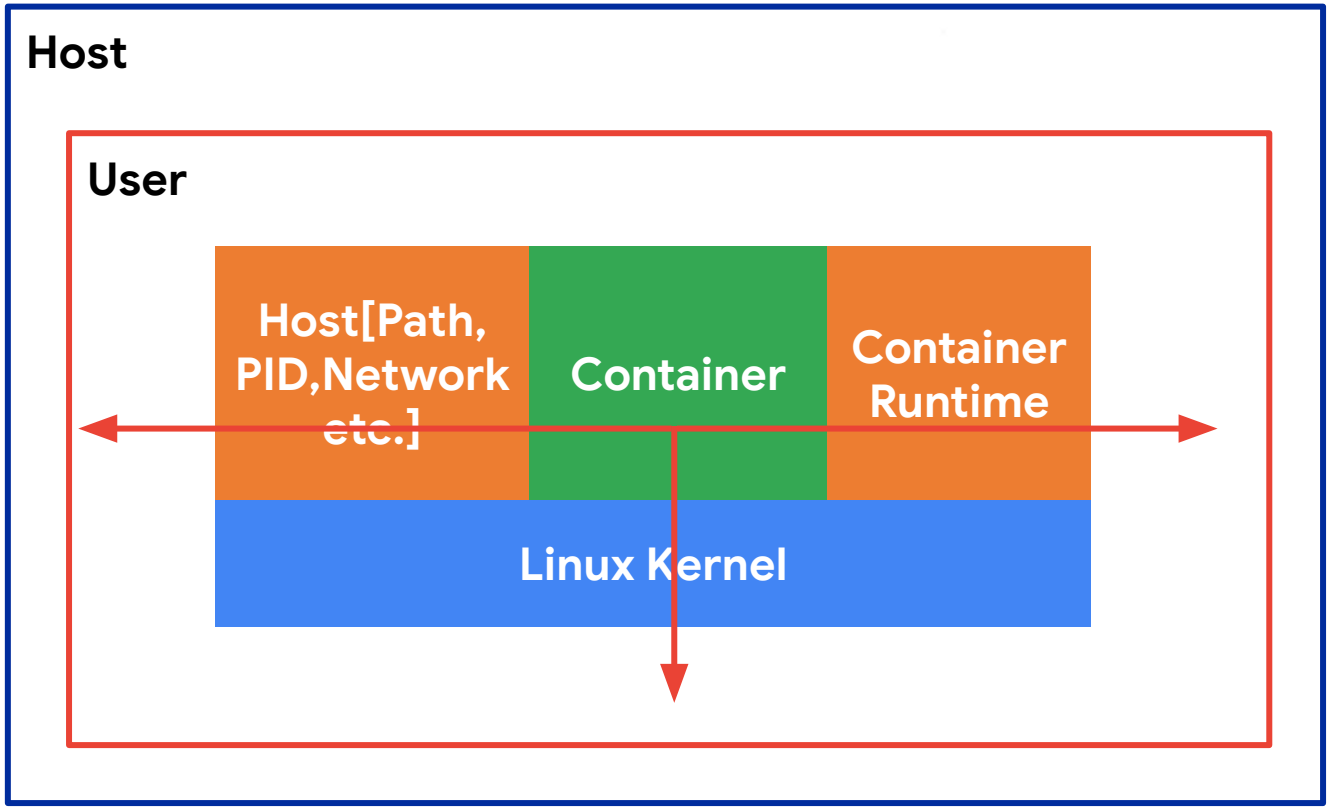
Sysdig 2022 Cloud-Native Security and Usage Report

GKE Non-Root Case Study

Migrated majority (83) of GKE platform containers to non-root

- Why bother?
- How: basic mechanics
- Strategy
- Design Choices
- Challenges and solutions
- The future: hostUsers feature

Non-Root Containers: Why



Fully/partially mitigate breakouts: [azurescape, and others]

Fully/partially mitigate escalation [through configured host access]

On Container Breakouts

~Every linux kernel has live breakout vulns *right now*

Our kCTF pays USD$20,000 to $91,337 for GKE exploits

Paid out $1,360,740 since May 2020 launch: **all linux kernel container breakouts**

Found and fixed 17 exploitable breakouts in 2022

Driving industry-changing kernel defense research: even more $$$!

Attack path very viable: need non-root protection

How to run as non-root

What does non-root mean?

😱 `docker run --privileged nginx`

YOLO: not even a container anymore

😿 `docker run nginx`

This talk: this transition

👍 `docker run --user nginx nginx`

Not covering: container *runtime* as non-root see rootlesscontaine.rs

Non-Root Containers: How

1. Modify **container** to **function** as non-root

2. Then modify **container/pod** to **run** as non-root

Demo

Platform-wide Migration Strategy

Platform-wide Migration Strategy

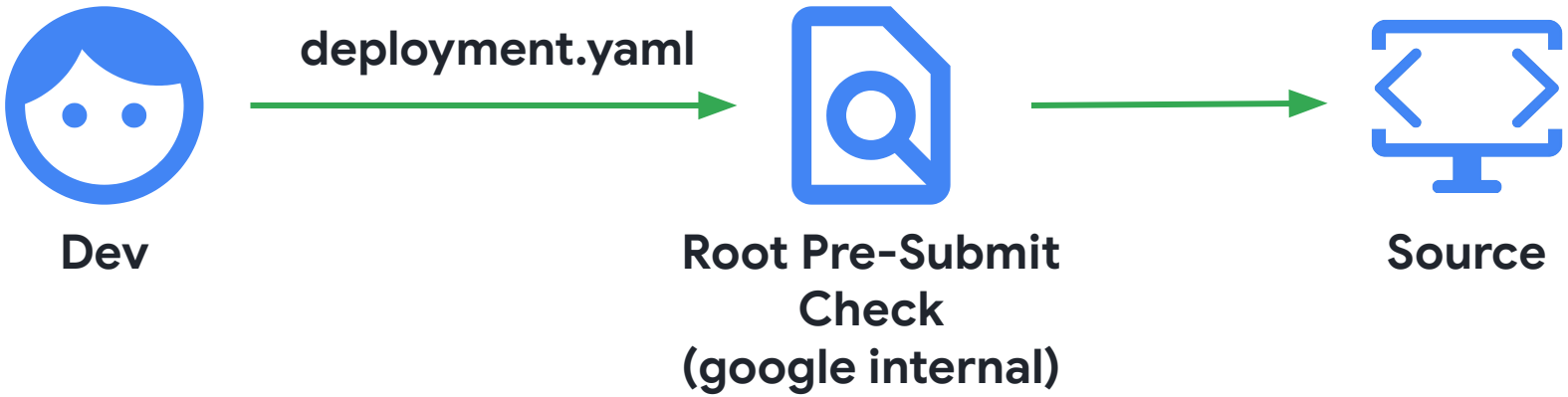
1. Block new root containers

2. Exempt and migrate existing root containers: fix upstream where practical

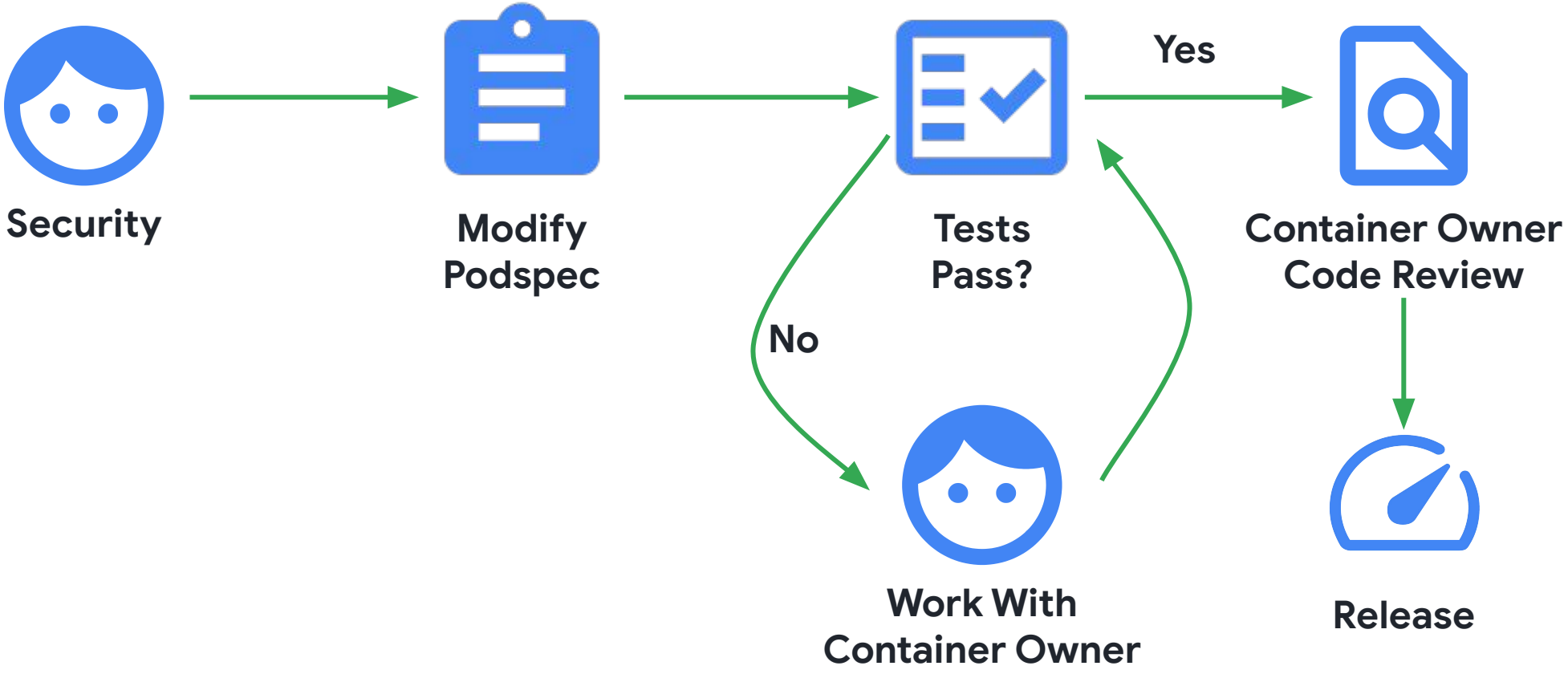
Upstream example changes:

- kube-apiserver
- kube-scheduler
- kube-controller-manager
- etcd

Strategy: Stop new root containers



Strategy: Migrate Existing



Design Choices

Design Choices

In-container vs. runtime configuration
Enforcement/Auditability
UID/GID management

GKE system containers = infrastructure-heavy

In-Container

You own the container

```
FROM gcr.io/distroless/static-debian11:latest

COPY server /server
ENTRYPOINT ["/server"]
USER 1001
```

Runtime Configuration

Our Approach

- No container rebuild
- Easy auditability
- Templatable
- Handle in central team

```yaml
name: fluentbit
securityContext:
  runAsUser: 2048
  runAsGroup: 2048
  privileged: false
  allowPrivilegeEscalation: false
  capabilities:
    drop:
    - all
```

All securityContext pieces required to run unprivileged

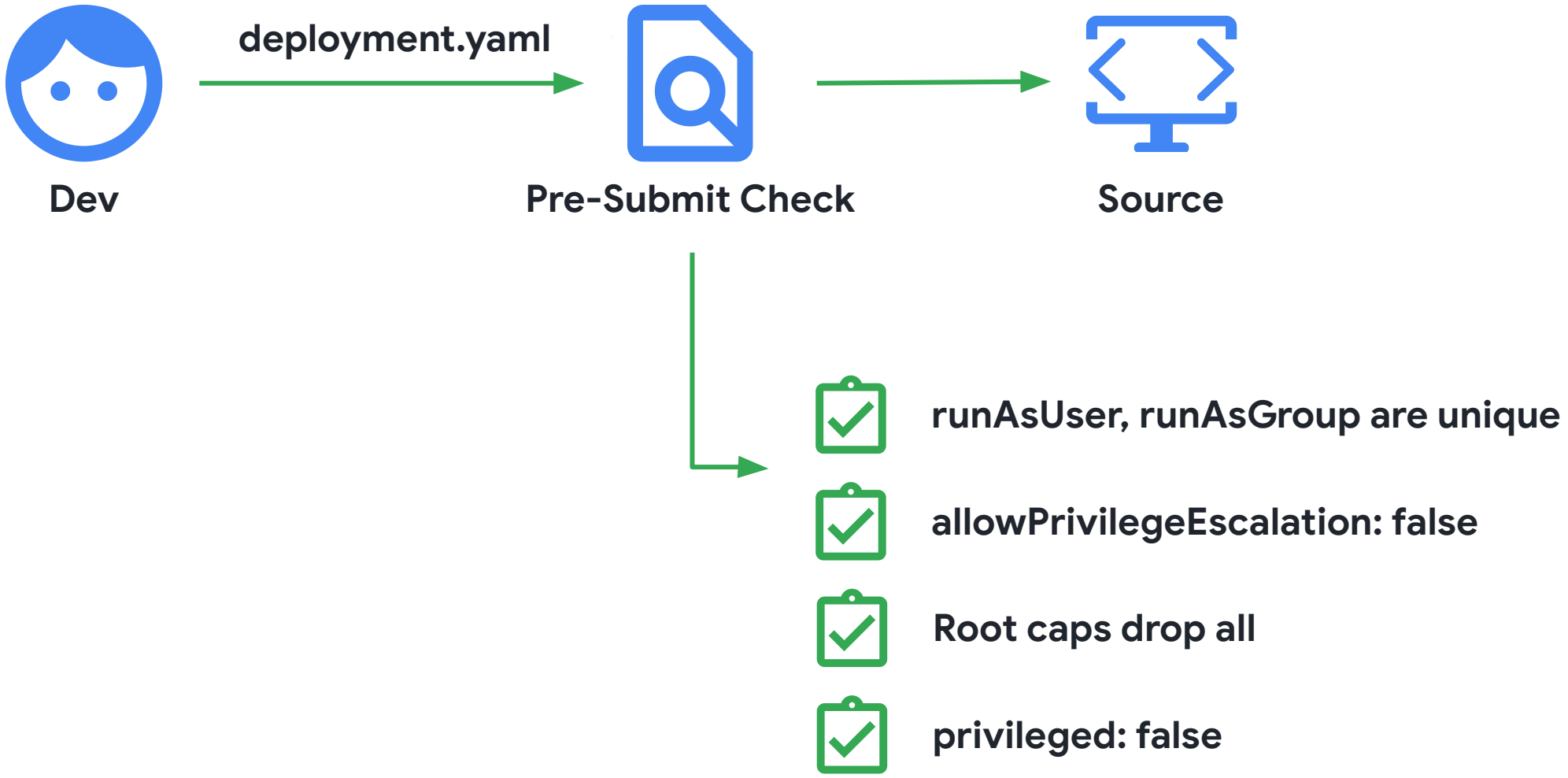
Design Choices

In-container vs. runtime configuration

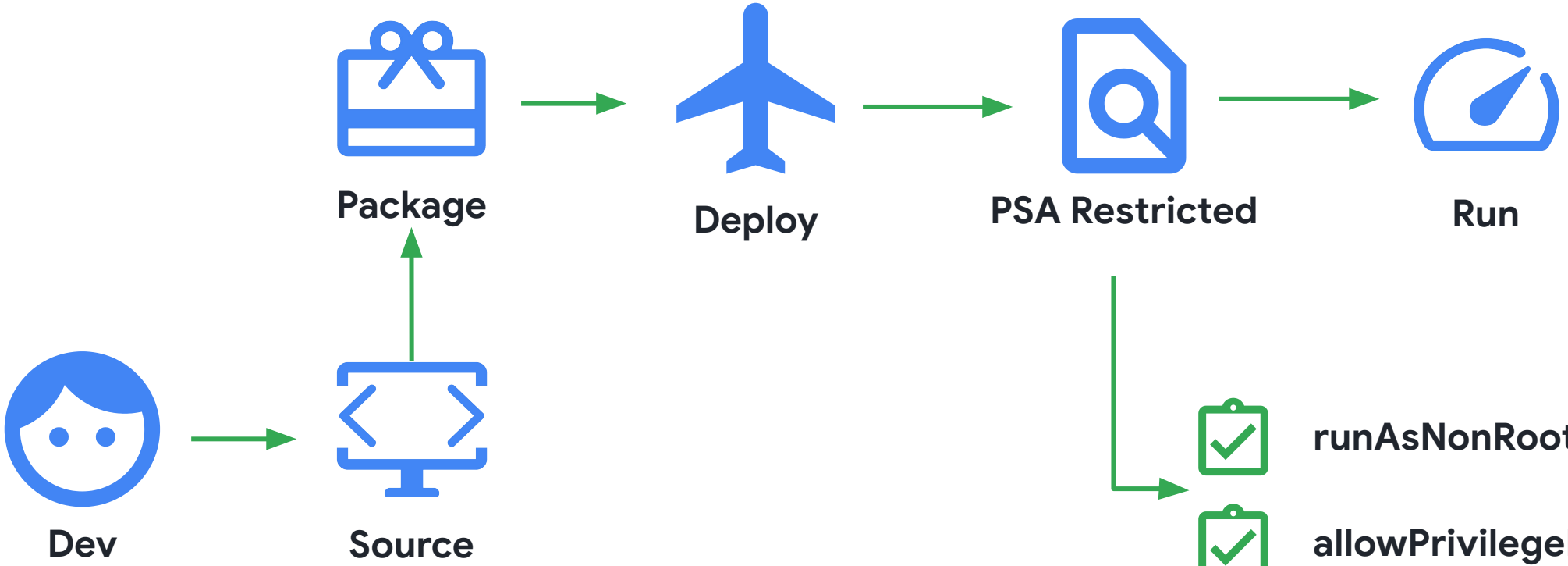
→ Enforcement/Auditability

UID/GID management

Enforcement: Our Solution



Alternative: Runtime Enforcement



- runAsNonRoot set
- allowPrivilegeEscalation: false
- Root caps drop all
- privileged: false

Gatekeeper: K8sPSPAllowedUsers, K8sPSPAllowPrivilegeEscalationContainer, K8sPSPCapabilities, K8sPSPPrivilegedContainer

Design Choices

In-container vs. runtime configuration
Enforcement/Auditability
UID/GID management

UID/GID Management

Goal: No collisions between containers or on-host

1. Decide on UID/GID range
2. Allocate *unique* UID and GID
3. *Enforce* uniqueness in pre-submit

Alternatives:

- Mutating admission that keeps track and sets
- Central coordination off-cluster

Challenges and Solutions

Migrating to Non-root: Challenges

- Access to host files/sockets
- Capability management

Host File Access

```yaml
containers:
- name: konnectivity-server-container
  securityContext:
    runAsUser: 2046
    runAsGroup: 2046
```

```yaml
volumes:
- name: konnectivity-uds
  hostPath:
    path: /etc/srv/kubernetes/konnectivity-server
    type: DirectoryOrCreate
```

fsGroup: not useful in this case

fsGroup *doesn't apply to hostPath mounts*

```
  securityContext:
    runAsUser: 2046
    runAsGroup: 2046
    fsGroup: 2046
```

OK for emptydir

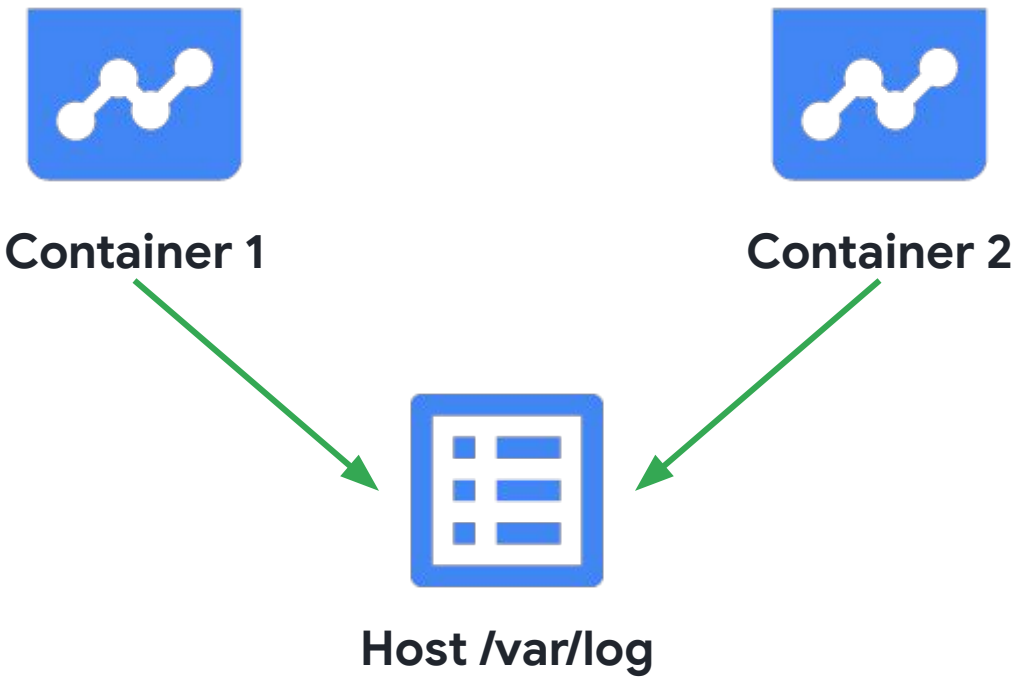
Workaround: initContainers

Run to completion before any of the long running containers

```yaml
initContainers:
- name: non-root-init
  image: ubuntu
  securityContext:
    runAsUser: 0
    runAsGroup: 0
  command:
  - "/bin/bash"
  - "-c"
  - "chown 2046 /etc/srv/kubernetes/konnectivity-server"
  volumeMounts:
  - mountPath: /etc/srv/kubernetes/konnectivity-server
    name: konnectivity-uds
```

Demo

Host File Access: Multi-user files



Host File Access: Multi-user files

```yaml
volumes:
- name: varlog
  hostPath:
    path: /var/log
    type: Directory
```

```
drwxr-sr-x 3 0 270 4096 Mar  3 00:56 /var/log/journal/
```

```yaml
spec:
  securityContext:
    supplementalGroups:
    # system-journal group on COS
    - 270
```

Migrating to Non-root: Challenges

Access to host files/sockets

→ Capability management

Capability Management

Need *part* of root powers (e.g., bind a low port)

```
name: "coredns"
securityContext:
  runAsUser: 2004
  runAsGroup: 2004
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["All"]
    add: ["NET_BIND_SERVICE"]
```

allowPrivilegeEscalation: not often needed

Surprise!

This works (root)

```
name: "coredns"
securityContext:
  runAsUser: 0
  runAsGroup: 0
  capabilities:
    drop: ["All"]
    add: ["NET_BIND_SERVICE"]
```

This *should* work, but doesn't

```
name: "coredns"
securityContext:
  runAsUser: 2004
  runAsGroup: 2004
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["All"]
    add: ["NET_BIND_SERVICE"]
```

Or: Just bind high!

Solution: originally Ambient capabilities KEP now hostUsers

Workaround: File Capabilities

```
RUN setcap cap_net_bind_service=+ep /coredns
```

Use setcap in Dockerfile

```yaml
name: "coredns"
securityContext:
  runAsUser: 2004
  runAsGroup: 2004
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["All"]
    add: ["NET_BIND_SERVICE"]
```

coredns OSS fix

Demo

This seems...harder than it should be

The Future

Foundation: linux user_namespaces

Allow **root inside** the container, but process is **unprivileged on the host**

Demo

In K8s: hostUsers

KEP-127: Kubernetes 1.25 Alpha

UserNamespacesSupport feature gate

containerd 1.7.0 (experimental support) and runc 1.1.4

Support only for stateless pods

No demo 😢

```
kind: Pod
spec:
  hostUsers: false
```

hostUsers FTW

Today's Challenges	hostUsers: false (new)
Modify to function as non-root	✅ Leave it using root
In-container vs. runtime	✅ Just do runtime
Enforcement/Auditability	✅ Add check for hostUsers: false
UID/GID management	✅ Kubelet handles allocation and uniqueness
Access to host files/sockets	🚫 Kubelet *might* handle with idmap mounts, maybe
Root capability management	✅ Separate from host and container!

Takeaways

Takeaways

- **Huge** security value in running non-root
- Convert containers you own!
- Org-wide conversion is do-able:
  - Stateless fairly easy
  - Volume mounts are harder
- Help is coming: hostUsers

Thanks!

KubeCon | CloudNativeCon
Europe 2023

- Slides and feedback: sched.co/1HyX4
- Demo code
- Liz Rice 2018 “don’t run as root” Keynote
- Sysdig 2022 Cloud-Native Security and Usage Report
- Privileged containers aren’t containers
- Privesc techniques with root containers
- kCTF improving kernel container breakout security
- Gatekeeper non-root policies: K8sPSPAllowedUsers, K8sPSPAllowPrivilegeEscalationContainer, K8sPSPCapabilities, K8sPSPPrivilegedContainer
- Broken K8s root capabilities
- allowPrivilegeEscalation use case explanations
- coredns OSS non-root fix
- Ambient capabilities KEP
- Linux user_namespaces
- KEP 127: Kubernetes user namespaces

Feedback

@mrgcastle
@gregcastle@infosec.exchange
@vglovespizza