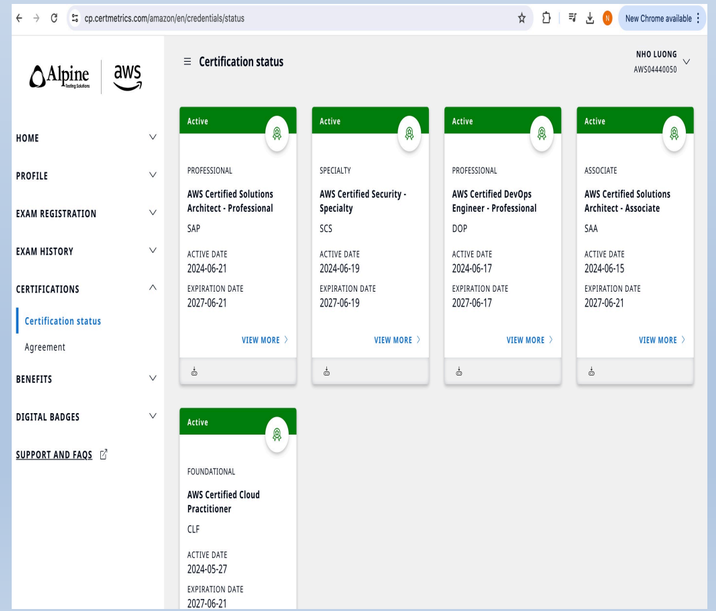
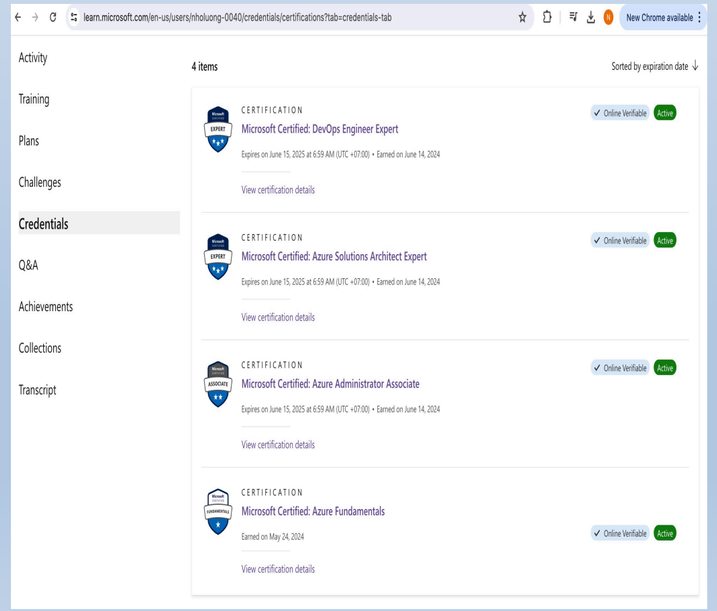
Securing Container Applications A Primer

Author: Nho Luong

Skill: DevOps Engineer Lead





Setting the Stage

Scope of Container Security

Application Security

Securing Your Container Image

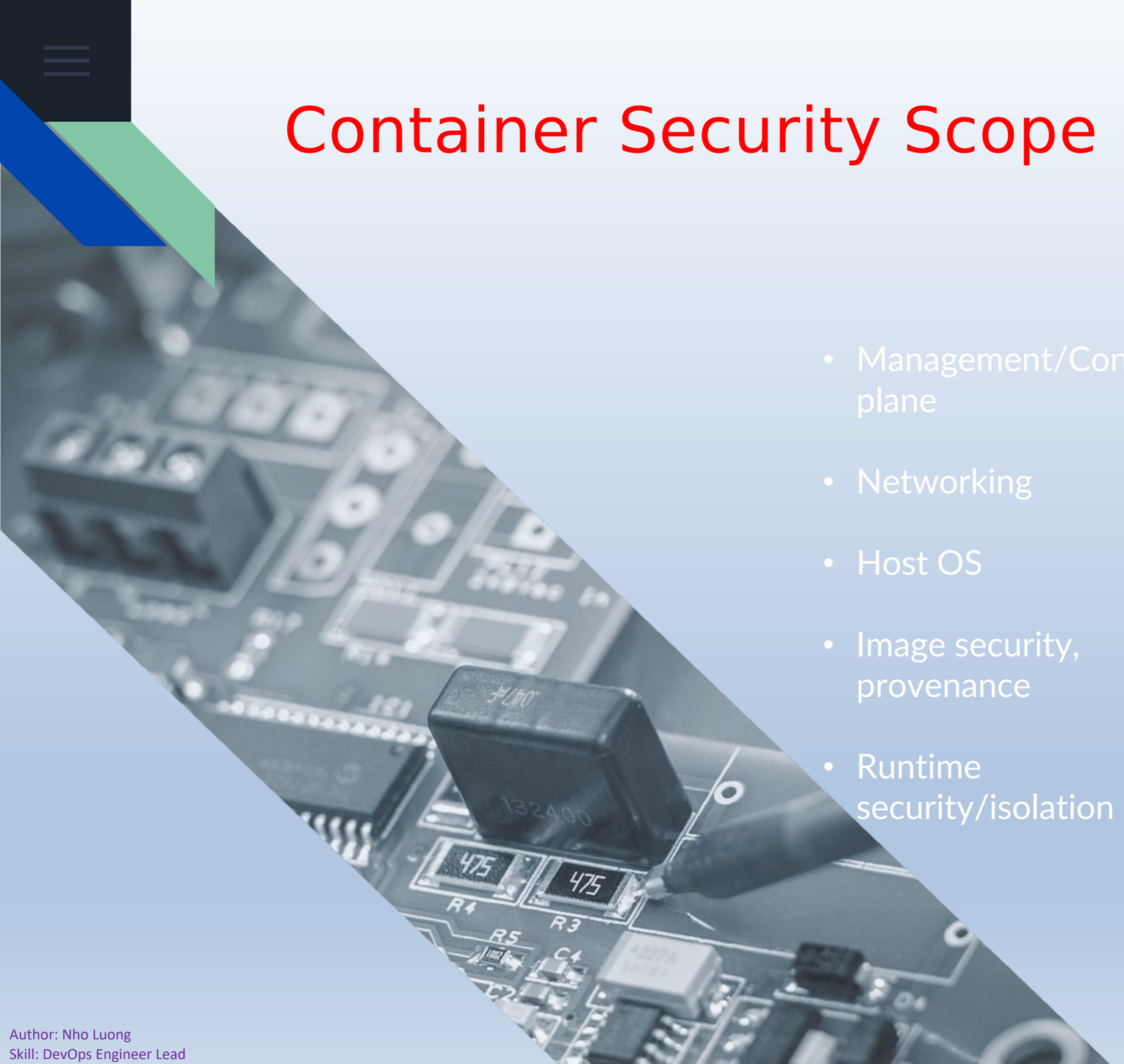
Runtime Security

Kubernetes Specifics

Summary

Author: Nho Luong
Skill: DevOps Engineer Lead

Container Security Scope (hint: it’s big)



- Management/Control plane
- Networking
- Host OS
- Image security, provenance
- Runtime security/isolation

The Ultimate Guide to Container Security - Container Journal
https://containerjournal.com › Topics › Container Security
Dec 11, 2018 - Here's a handy a primer on **security best practices** for every component involved in your **container** infrastructure.

8 Takeaways from NIST's Application Container Security Guide
https://www.synopsys.com › blogs › software-security › 8-takeaways-nists-...
Dec 13, 2017 - NIST published the “Application **Container Security Guide**” in September to address security risks associated with container adoption. Read 8 ...

Container Security Best Practices – BMC Blogs - BMC Software
https://www.bmc.com › blogs › container-security-best-practices
Apr 26, 2019 - In this Run and Reinvent podcast, I'm joined by Maya Kaczorowski, a product manager at Google, to discuss **container security**. Below is a ...

OWASP Container Security Verification Standard (CSVS ...
https://www.owasp.org › index.php › OWASP_Container_Security_Verific...
Jul 26, 2019 - The **Container Security** Verification Standard (CSVS) is a ... the CSVS with its strong focus on the proactive controls to tech about **best practices**.

Docker Container Security: Challenges and Best Practices
https://resources.whitesourcesoftware.com › blog-whitesource › docker-co...
Docker is a complicated beast, and there is no simple trick you can use to maintain **Docker container security**. We offer a set of **best practices** that should help ...

Container Security: A Developer Guide | Okta Developer
https://developer.okta.com › blog › 2019/07/18 › container-security-a-dev...
Jul 18, 2019 - A short **guide** which explains how to properly secure **containers** and things to keep in mind when using **containers**.

Docker Security Best-Practices - DEV Community - Dev.to
https://dev.to › petermbenjamin › docker-security-best-practices-45ih
Jul 19, 2018 - Exploring **security best practices** around **Docker**. ... Make sure you follow OS

Author: Nho Luong
Skill: DevOps Engineer Lead

What We **Won’t** Cover

Hardening your host OS

Hardening your cluster

Configuring container runtimes

Securing network traffic

Author: Nho Luong
Skill: DevOps Engineer Lead

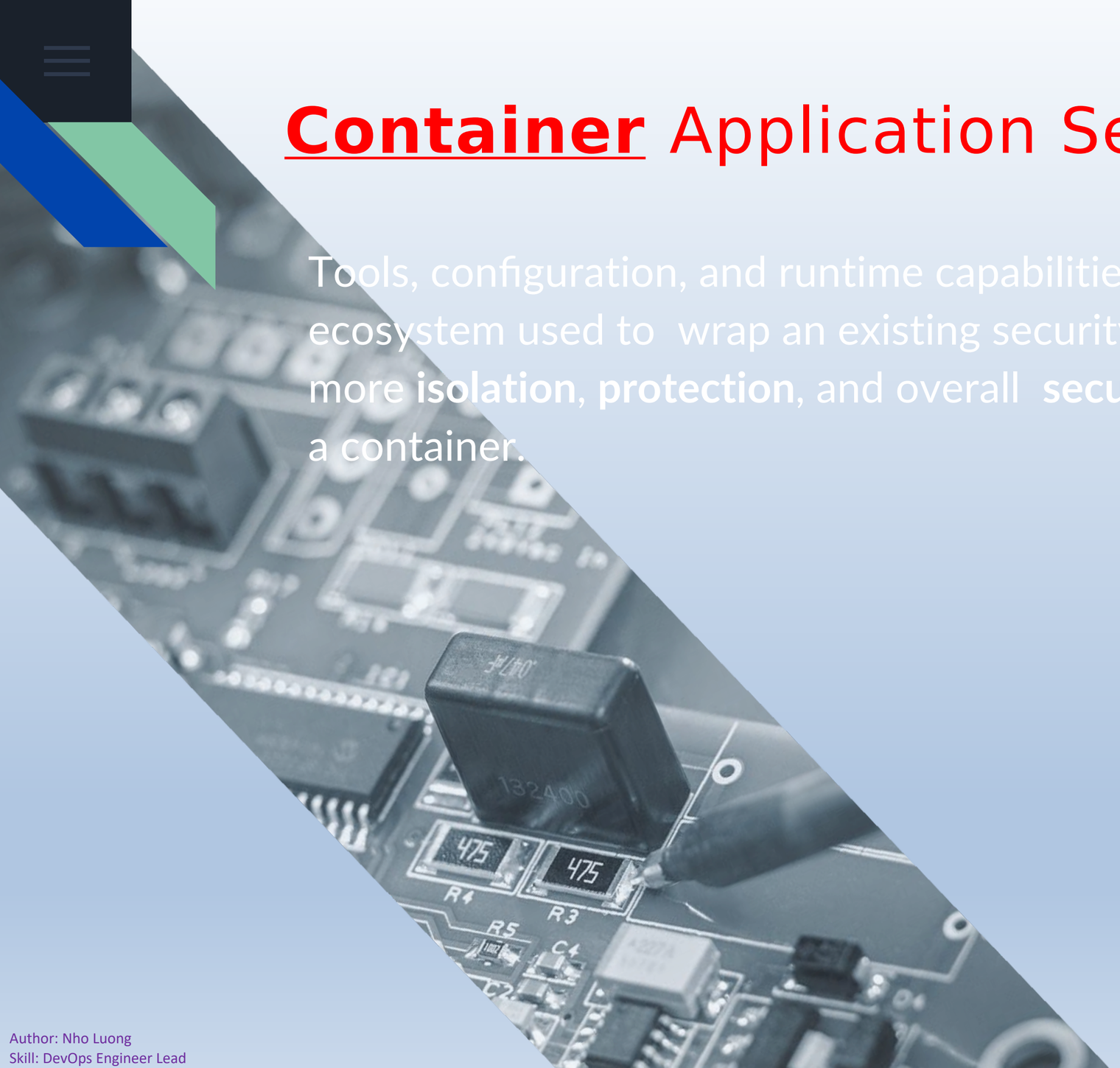
Application Security

I want my application to be secure from **potential exploits in my code or code I depend on**. I want my application to not misbehave in ways that **impact the host system I run on or other applications hosted in a shared environment**. I want my application to be **resilient to failure conditions** and have reasonable protection from **denial of service** outages.

Author: Nho Luong
Skill: DevOps Engineer Lead

Container Application Security

Tools, configuration, and runtime capabilities provided via the container ecosystem used to wrap an existing security-focused application to provide more **isolation**, **protection**, and overall **security** for your code running inside a container.



Author: Nho Luong
Skill: DevOps Engineer Lead

Image Security - Contents

- **Never include secrets** (passwords, API keys, tokens, private keys) in an image!
  - Container images are not encrypted; anyone with access to the registry or the local system where they are downloaded/unpacked will have access to these secrets
- **FROM what?**
  - Your starting point is very important; your application and the pieces of a Linux distribution (for Linux containers) are now your responsibility to maintain/manage.
  - Who maintains it? Who updates it for CVEs? How often do you rebuild on a new base?
- **Minimize content -** fewer packages means less maintenance/security issues
  - Use multi-stage builds to keep build-time dependencies out of your final image
  - Using DockerHub official images? Start with “slim” or “alpine” tagged variants
- **Use “FROM SCRATCH”**
  - Your files are added to an empty container filesystem
  - Requires ability to build a static binary with no dependencies
  - Requires some knowledge of how to assemble base filesystem (e.g. SSL root certificates)

Author: Nho Luong
Skill: DevOps Engineer Lead

Image Security - Build Concerns

- **Image signing**
  - TUF/Notary are CNCF projects; Docker-specific implementation as “Docker Content Trust”
  - Cons: Notary project health concerns/feature requests; some cloud providers looking at OCI for possible standardization (very early discussions)
  - Pros: can be used to enforce provenance of an image through a set of gates in your devops environment (e.g. test, scanning, promotion to prod)
- **Image scanning (larger topic: CI/CD pipelines)**
  - You need visibility into whether your image has unpatched contents when using a Linux distribution. Some vendor-specific scanners can look beyond CVE content at other potential runtime security issues.
  - A scanner won’t fix your out of date image!
- **Specify a non-root user**
  - “USER {some-unprivileged-user}” in Dockerfile
  - Rarely does your container need root access to perform its job; don’t run containers as root—it’s adding an unnecessary weakness in your layers of defense

Author: Nho Luong
Skill: DevOps Engineer Lead

Runtime Security - Containing your Workload

- **Resource Limits**
  - The essence of Linux containers are based on two key Linux features: namespaces + cgroups. Linux control groups (cgroups for short) are used to limit resources to any process in Linux
  - You can control memory, I/O bandwidth, and CPU usage with standard cgroups
- **The system call “attack surface”**
  - Your application, whether you know it or not, is using Linux system calls (syscalls)
  - If you know specifics about your application’s level of capabilities needed, you can drastically reduce the “attack surface” of the entire system call table (and related privileges)
- **Administrative privilege**
  - The “USER” specified in the Dockerfile can be overridden at runtime. It would be best to have runtime limitations to not allow images to run as root, or to whitelist some specific administrative tools/applications which cannot be changed
  - User namespaces are a larger hammer to prevent this, but have not arrived in Kubernetes yet
- **Read-only root filesystem**

Author: Nho Luong
Skill: DevOps Engineer Lead

Runtime Security Focus

01	RESOURCES	*As limited as is feasible.*
02	ATTACK SURFACE	*As small as is possible.*
03	PRIVILEGES	*The least amount necessary.*

Author: Nho Luong
Skill: DevOps Engineer Lead

Runtime Security - Resources

- **Limit memory & CPU**
  - Memory and CPU specifications are part of the OCI container runtime spec: all OCI-compliant runtimes implement these features
  - Kubernetes exposes these limits (in a less-granular form) via the Pod specification
- **Limit processes**
  - Pids limit is also part of the OCI spec; this can be important to prevent forkbombs or any other runaway process forking in your container
- **Limit disk consumption**
  - The OCI spec has I/O bandwidth limitations via cgroups, but not exposed in K8s
  - Kubernetes has ephemeral disk limitations
  - Advanced (ops) topic: use a quota-supporting FS
  - Be cautious with host-mounted filesystems


```
"hugepageLimits": [
    {
        "pageSize": "2MB",
        "limit": 9223372036854772000
    }
],
"memory": {
    "limit": 536870912,
    "reservation": 536870912,
    "swap": 536870912,
    "kernel": -1,
    "kernelTCP": -1,
    "swappiness": 0,
    "disableOOMKiller": false
},
"cpu": {
    "shares": 1024,
    "quota": 1000000,
    "period": 500000,
    "realtimeRuntime": 950000,
    "realtimePeriod": 1000000,
    "cpus": "2-3",
    "mems": "0-7"
},
```


Author: Nho Luong
Skill: DevOps Engineer Lead

Runtime Security - Attack Surface

- **Linux Capabilities**
  - Linux capabilities are collections of system calls with a similar purpose; they have names like CAP_NET_RAW or CAP_SYS_ADMIN. Some are fairly fine grained and some, like CAP_SYS_ADMIN, might as well be “the new root” as lwn.net called it!
  - Kubernetes provides a way to drop/add capabilities to a container via the Pod spec
- **LSMs: AppArmor/SELinux**
  - Linux Security Modules can be complicated to understand, but they effectively provide a “language” to describe a wide-ranging set of permission limits on processes: e.g. specific permissions (read/write) to filesystem paths, network socket access, among others
  - Docker (and other runtimes) have default profiles which restrict container permissions
  - Kubernetes provides support for named AppArmor profiles in the spec; however, it is an operational concern to install unique profiles onto your worker nodes
- **SECCOMP (“Secure Computing”)**
  - “Seccomp” support in container runtimes allows you to specify one-by-one which syscalls are allowed for a process. Docker’s default profile removes 44 from the list of 330+

Author: Nho Luong
Skill: DevOps Engineer Lead

Runtime Security - Privileges

- **Don’t run privileged containers!**
  - There really isn’t much else to say here: most of the controls discussed here are removed when you enable “privileged” for a container/pod.
  - Giving CAP_SYS_ADMIN to a container is effectively very similar to root/full privilege
- **Root/Administrative user**
  - Don’t run containers as root, and don’t give them ways to escalate privilege (various settings in

    PodSecurityPolicy can help here)
  - Make this enforceable via an admission controller which enforces baseline policies
  - User namespaces will come to Kubernetes at some point and offer a more nuanced way to have administrative privilege **only inside the container**
  - Remember that exposing the K8s or Docker (or other runtime) API—e.g. mounting the Docker socket— inside the container is most likely an escalation path to root on the host. There are solutions to most of the historic reasons applications have “reached down” into the container runtime!

Author: Nho Luong
Skill: DevOps Engineer Lead

Kubernetes: Controlling Resource Limits

- **Resource Limits**
  - Set per-container in the Pod yaml
  - Note that not all OCI spec memory/CPU options are exposed in the K8s API specification
- **Limit Processes**
  - Still alpha as of Kubernetes 1.16
  - Cluster operator must enable feature gate **SupportPodPidsLimit=true**, and then pass a --pod-max-pids integer to kubelet
  - Limit is fixed per-pod; no customization possible
- **I/O Bandwidth Limits**
  - The cgroups i/o settings are not exposed here to be set per container.
  - K8s does offer resource quotas, and QoS features—related but not the same features

```
apiVersion: v1  kind: Pod  metadata:
name: frontend  spec:
containers:
    •   name: db  image: mysql  resources:
    limits:
    memory: "128Mi"  cpu: "500m"
    •   name: wp
    image: wordpress  resources:
    limits:
    memory: "128Mi"  cpu: "500m"
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Kubernetes: Limiting Attack Surface

- **Capabilities**
  - Set per-container via **securityContext** ; can add/drop caps by name
- **AppArmor**
  - Annotations are used to identify AppArmor profiles in Kubernetes
  - Operator must install them on worker nodes; developing new profiles? Tools TBD
- **Seccomp**
  - Also set via annotation, but on **PodSecurityPolicy**; see upcoming example; **not default**

```
apiVersion: v1  kind: Pod  metadata:
name: hello-apparmor  annotations:
container.apparmor.security.beta.kubernetes.io/hello: localhost/deny-write  spec:
containers:
- name: hello
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Kubernetes: Reducing Privilege

- **Non-root user**
  - Use securityContext for containers and pod-level control
  - Use **PodSecurityPolicy** to enforce restrictions cluster-wide
- **Capabilities (privilege related)**
  - Also in securityContext; see example

```
apiVersion: v1  kind: Pod  metadata:
name: security-context-demo  spec:
securityContext:  runAsUser: 1000
runAsGroup: 3000
fsGroup: 2000  containers:
- name: sec-ctx-demo  image: busybox
command: [ "sh", "-c", "sleep 1h" ]
securityContext:  allowPrivilegeEscalation: false
runAsUser: 2000  capabilities:
add: ["NET_ADMIN", "SYS_TIME"]
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Kubernetes: Cluster Security Enforcement

```
apiVersion: policy/v1beta1  kind: PodSecurityPolicy  metadata:
name: restricted  annotations:
seccomp.security.alpha.kubernetes.io/allowedProfileNames:                    'runtime/default'
apparmor.security.beta.kubernetes.io/allowedProfileNames:                    'runtime/default'
seccomp.security.alpha.kubernetes.io/defaultProfileName:                     'runtime/default'
apparmor.security.beta.kubernetes.io/defaultProfileName: 'runtime/default'
spec:
privileged: false
allowPrivilegeEscalation: false
# This is redundant with non-root + disallow privilege escalation:
requiredDropCapabilities:
- ALL
hostNetwork: false  hostIPC: false  hostPID: false  runAsUser:
# Require the container to run without root privileges.
rule: 'MustRunAsNonRoot'
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Kubernetes: Applying Container Security

- **PodSecurityPolicy**: enforce many good practices cluster-wide! OpenShift is a good example of a Kubernetes distribution with strong defaults out of the box
- Use the **Kubernetes secrets** implementation to protect sensitive keys, tokens, materials. Vendor tools available as well (Hashicorp Vault), and potentially from your cloud provider
- **Don't circumvent security to make your code "easy"**: e.g. K8s API access with admin role; mounting container runtime (e.g. Docker) API with full privilege
- Have a unique workload requirement (multi-tenancy, untrusted code)? Take a look at **RuntimeClass features** in Kubernetes to allow custom isolators (gVisor, Kata, Firecracker, Nabla, etc.)
- Remember that you need **visibility** and not simply fire-and-forget security! **Logging**, **audit**, vendor tools/open source projects for runtime protection, **anomaly detection**, etc.

Author: Nho Luong
Skill: DevOps Engineer Lead

BUT...Container Security is Hard!!

- **Use a cloud provider**
  - Managed Kubernetes services many times can be created with a set of default tools and policies for strong controls pre-configured for you
  - Many managed services integrate with popular vendor tooling
    - e.g. Twistlock, Snyk, Aqua, Datadog, Sysdig, LogDNA and many others
- Use **recommended guides and profiles publicly available** (CIS, NIST, DockerBench, etc.)
- **Try out** emerging tooling
  - Generate seccomp profiles by running your application with BPF tracing: https://github.com/containers/oci-seccomp-bpf-hook

Author: Nho Luong
Skill: DevOps Engineer Lead

Resources

1. PodSecurityPolicy:

https://kubernetes.io/docs/concepts/policy/pod-security-policy/

2. Kubernetes Security Concepts:

https://kubernetes.io/docs/concepts/security/overview/

3. AppArmor documentation:

https://kubernetes.io/docs/tutorials/clusters/apparmor/

4. SELinux documentation:
https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#assign-selinux-labels-to-a-container

5. Resource controls:
https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

6. Complete list of Linux capabilities:
http://man7.org/linux/man-pages/man7/capabilities.7.html

Author: Nho Luong
Skill: DevOps Engineer Lead

Thank You

Author: Nho Luong
Skill: DevOps Engineer Lead