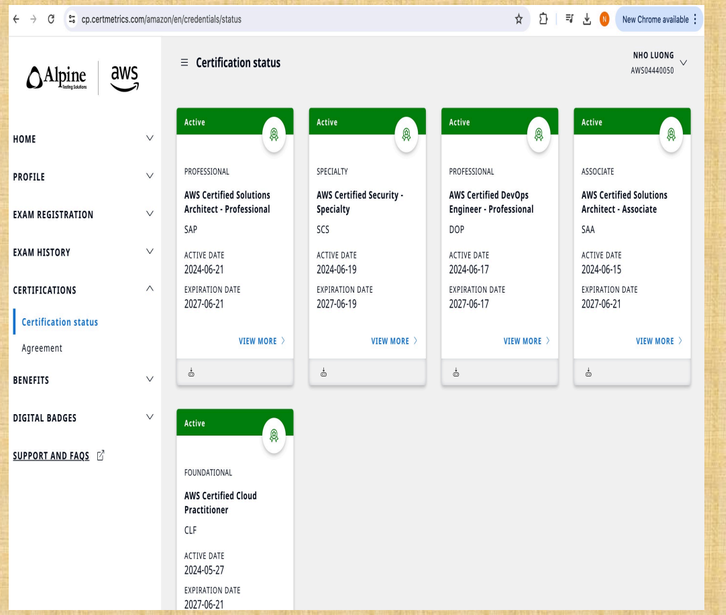
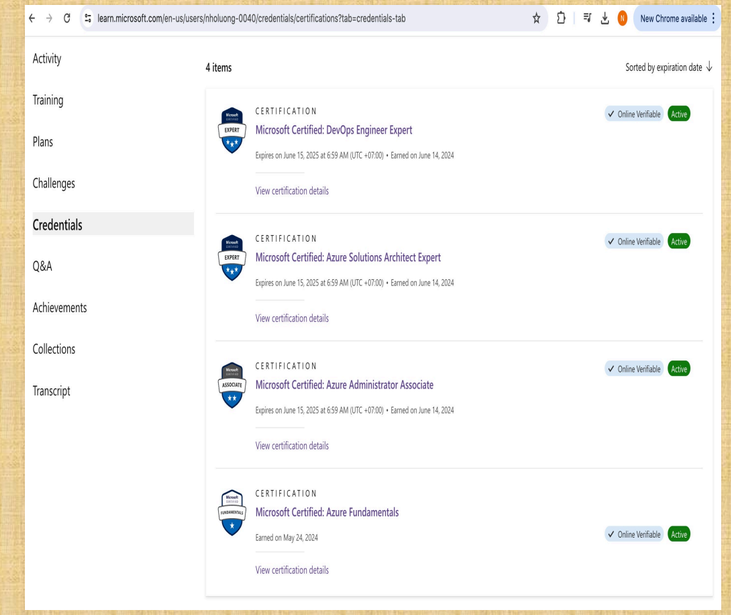
Kubernetes Ochestration

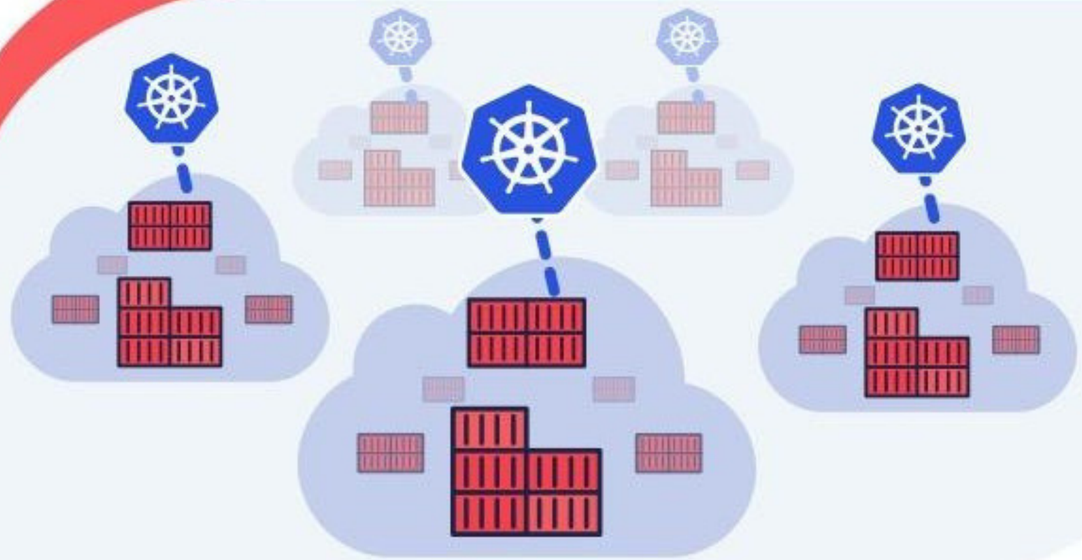
Author: Nho Luong

Skill: DevOps Engineer Lead





Securing Kubernetes Workloads



Best Practices for Securing Kubernetes Workload Configurations Across Clouds

Author: Nho Luong
Skill: DevOps Engineer Lead

Kubernetes Security Framework

	Build	Operate
Container Hosts:	• Minimal OS • OS Hardening	• CIS Benchmarks
Clusters:	• RBAC • Audit Policies and Logging • Certificate Management	• Identity and Access • Kubernetes upgrades • CIS Benchmarks
Applications:	• Image scanning	• Image Provenance • Secrets Management • Namespaces • Access Controls • Network Policies • Resource Quotas • Pod Security Policy

Author: Nho Luong
Skill: DevOps Engineer Lead

Kubernetes Security Framework

	Build	Operate
Container Hosts:	• Minimal OS • OS Hardening	• CIS Benchmarks
Clusters:	• RBAC • Audit Policies and Logging • Certificate Management	• Identity and Access • Kubernetes upgrades • CIS Benchmarks
Applications:	• Image scanning	• Image Provenance • Secrets Management • Namespaces • Access Controls • Network Policies • Resource Quotas • Pod Security Policy

Author: Nho Luong
Skill: DevOps Engineer Lead

Image Provenance

- Image scanning checks images for vulnerabilities
  - Ideally done when the image is built and before it is accepted into the image registry
- Image provenance
  1. Confirms that an image being deployed is from a trusted source
  2. Confirms that image has not been not tampered with

Image Provenance - Solutions

- **Kubernetes ImagePolicyWebhook**
  - Configured as an admission controller
  - Sends an ImageReview request
  - Expects an ImageReview response of accept or deny

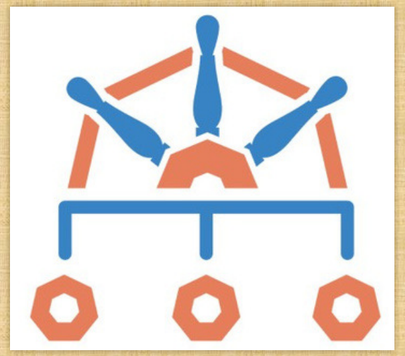
Image Provenance - Solutions

- **Portieris**
  - Also an admission controller
  - Integrates with Notary (a content trust store) – part of the The Update Framework (TUF)
  - Provides way to specify image security policies at a namespace and cluster level

Author: Nho Luong
Skill: DevOps Engineer Lead

Image Provenance – Partial Solutions

- **Kyverno**
  - Also an admission controller
  - Kubernetes Native Policy Engine
  - Policies are written as overlay rules



```yaml
kind: ClusterPolicy
metadata:
  name: validate-image-registry
spec:
  rules:
  - name: validate-image-registry
    match:
      resources:
        kinds:
        - Pod
    validate:
      message: "Image registry is not allowed"
      pattern:
        spec:
          containers:
          - name: "*"
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Image Provenance – Partial Solutions

- **OPA / Gatekeeper**
  - Also an admission controller
  - General Purpose Policy Engine
  - Policies are written in Rego

```
package kubernetes.admission

import data.kubernetes.namespaces

deny[msg] {
    input.request.kind.kind = "Deployment"
    input.request.operation = "CREATE"
    registry =
input.request.object.spec.template.spec.containers[_].image
    name = input.request.object.metadata.name
    namespace = input.request.object.metadata.namespace
    not reg_matches_any(registry,valid_deployment_registries)
    msg = sprintf("invalid deployment, namespace=%q, name=%q,
registry=%q", [namespace,name,registry])
}

valid_deployment_registries = {registry |
    whitelist = "<COMMA_SEPARATED_LIST_OF_ALLOWED_REGISTRIES>"
    registries = split(whitelist, ",")
    registry = registries[_]
}

reg_matches_any(str, patterns) {
    reg_matches(str, patterns[_])
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Secrets Management Anti-Patterns

(please try <u>not</u> to do this)

x Hard-coded

x Packaged with code

x Inserted via build tools x

Environment Variables

- Any sensitive data that an application needs
  - Passwords
  - Certificates
  - Keys
  - ...

Author: Nho Luong
Skill: DevOps Engineer Lead

What Kubernetes Provides

- API Object to define secrets
- Values are base 64 encoded (default)
- Secrets are namespaced
- Secrets can be mounted as volumes
- Secrets can be used as environment variables
- Encryption can be configured at the API Server

```
apiVersion: v1 kind:
Secret metadata: name:
mysecret type: Opaque
data:
username: YWRtaW4=
```

Author: Nho Luong
Skill: DevOps Engineer Lead

So, what's missing?

Kubernetes secrets are a step forward, but have a few limitations:

- Encryption requires configuring static keys or a KMS
- Shared (static) approach
- No leases, rotation, etc.

Author: Nho Luong
Skill: DevOps Engineer Lead

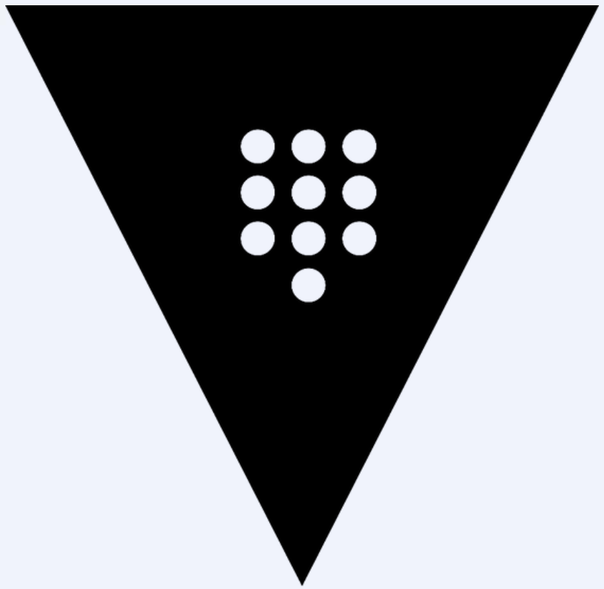
Secrets Management with Hashicorp Vault

- Helps automates security best practices for
  - Secrets Management
  - Auditing
  - Certificate Management
  - Encryption
- Dynamic Secrets

o Credentials (keys, passwords, certificates) are generated when a client requests them

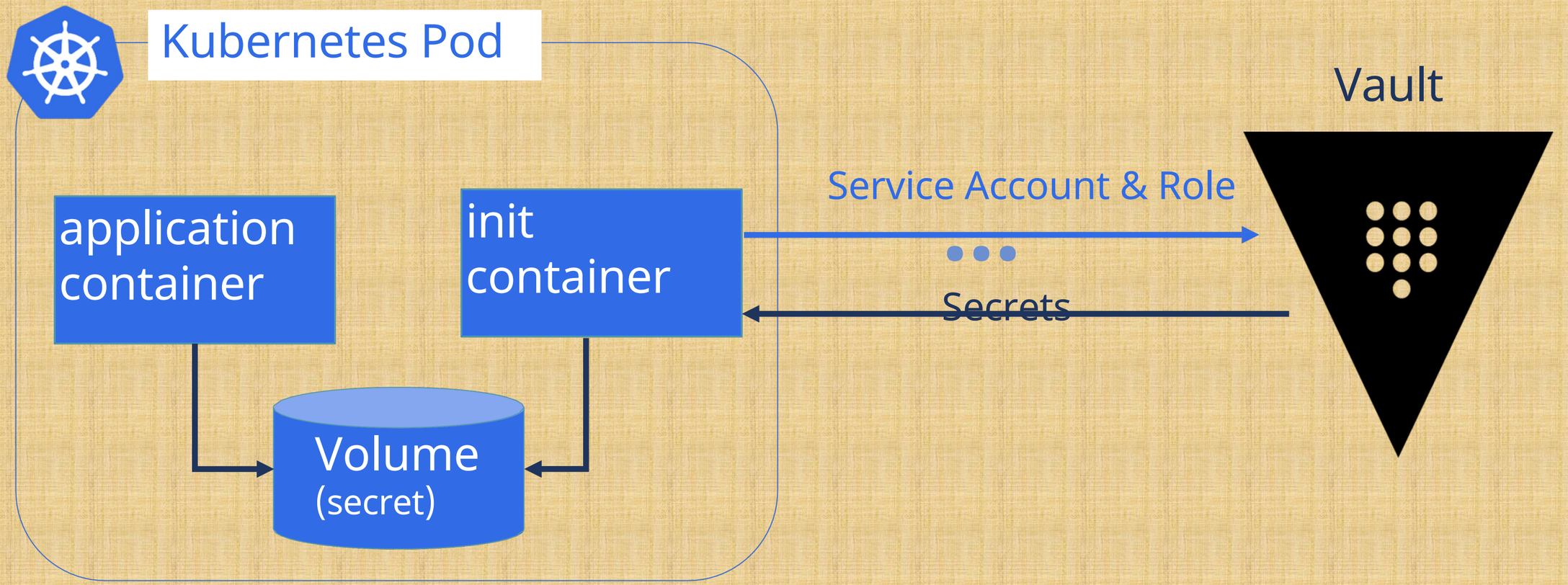
o Credentials are per client

o Credentials are automatically deleted if a lease expires



Author: Nho Luong
Skill: DevOps Engineer Lead

An init container to fetch secrets



Author: Nho Luong
Skill: DevOps Engineer Lead

Namespaces

- **Kubernetes Data Plane Virtualization**

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.

- Namespaces partition the Kubernetes object model so multiple objects with the same name can exist in the same cluster
- Namespaces are the foundation for applying other security constructs

Author: Nho Luong
Skill: DevOps Engineer Lead

Role-based access control (RBAC)

- Users are authenticated via OIDC, X.509 certificates, tokens, etc.
- The authentication result can provide user and group information.
- However, Users and User Groups are managed externally (e.g. in an LDAP / AD server).
- Kubernetes has a fine-grained permission model
  - Role (namespace) / ClusterRole
- Roles are mapped to users or groups via role bindings
  - RoleBinding (namespace) / ClusterRoleBinding

Author: Nho Luong
Skill: DevOps Engineer Lead

Service Accounts

- Service Accounts are meant for authenticating and authorizing processes
- Each namespace has a default service account
- Each Pod has a service account (default – if not specified)
- A best practice is to use a service account per app
- To prevent a service account token from being mounted in a Pod use "**automountServiceAccountToken: false**". This can be enforced via a policy.

Author: Nho Luong
Skill: DevOps Engineer Lead

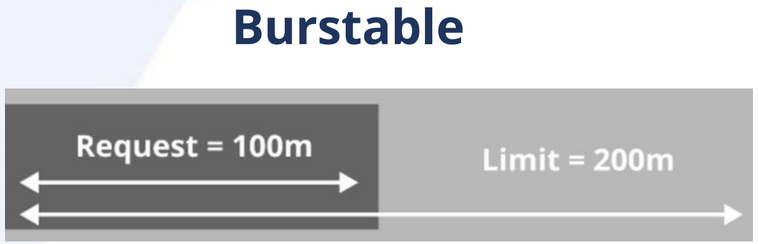
Network Segmentation via Network Policies

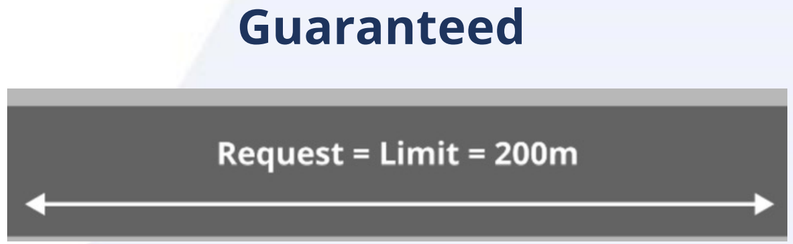
- By default, Kubernetes pods are “**non-isolated**”
  - They accept network connections from any source and can initiate connection requests to any destination
- Network Policies define traffic rules for Kubernetes pods
  - ingress (inbound traffic)
  - egress (outbound traffic)

Author: Nho Luong
Skill: DevOps Engineer Lead

Resource Management

- Pods can have resource requests and limits
- This allows three quality of service models





- A namespace can have limits and default allocations
- Quotas and limits ensure fairness and stability

Author: Nho Luong
Skill: DevOps Engineer Lead

Pod Security Policies

- Controls runtime security settings for pods
- Enabled at the API Controller
- Requires a role binding between pod Service Account and the PSP

Control Aspect	Field Names
Running of privileged containers	`privileged`
Usage of host namespaces	`hostPID`, `hostIPC`
Usage of host networking and ports	`hostNetwork`, `hostPorts`
Usage of volume types	`volumes`
Usage of the host filesystem	`allowedHostPaths`
White list of Flexvolume drivers	`allowedFlexVolumes`
Allocating an FSGroup that owns the pod's volumes	`fsGroup`
Requiring the use of a read only root file system	`readOnlyRootFilesystem`
The user and group IDs of the container	`runAsUser`, `runAsGroup`, `supplementalGroups`
Restricting escalation to root privileges	`allowPrivilegeEscalation`, `defaultAllowPrivilegeEscalation`
Linux capabilities	`defaultAddCapabilities`, `requiredDropCapabilities`, `allowedCapabilities`
The SELinux context of the container	`seLinux`
The Allowed Proc Mount types for the container	`allowedProcMountTypes`
The AppArmor profile used by containers	annotations
The seccomp profile used by containers	annotations
The sysctl profile used by containers	`forbiddenSysctls`, `allowedUnsafeSysctls`

Author: Nho Luong
Skill: DevOps Engineer Lead

Use a policy engine to audit and enforce

- Pod Security Policies are tricky to manage
  - Require a role binding to SA
  - Applied in alphabetical order
- Kyverno supports enforcement of the important PSP checks

```yaml
kind: ClusterPolicy
metadata:
  name: validate-deny-runasrootuser
spec:
  validationFailureAction: "audit"
  rules:
  - name: deny-runasrootuser
    exclude:
      resources:
        namespaces:
        - kube-system
    match:
      resources:
        kinds:
        - Pod
    validate:
      message: "Root user is not allowed. Set runAsNonRoot to true."
      anyPattern:
      - spec:
          securityContext:
            runAsNonRoot: true
      - spec:
          containers:
          - name: "*"
            securityContext:
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Thank You

Author: Nho Luong
Skill: DevOps Engineer Lead