

# Kubernetes Security

Attacking And Defending Modern Infrastructure

---

Lenin Alevski

## About Me

- Lenin Alevski 🇲🇽
- Security Engineer @ Google 🇺🇸
- Open Source Contributor
- Corporate & Startup world
- I'm obsessed with cybersecurity ❤️



ALEVSK

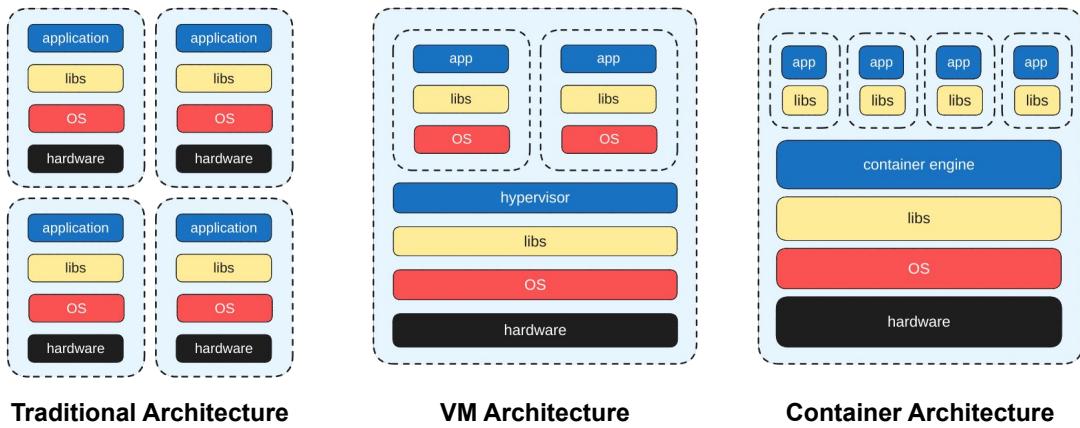
# Agenda

- Introduction to Containers
- Introduction to Kubernetes
- Kubernetes Threat Model
- Most common attack techniques in K8S
- Build-In Defenses
- Kubernetes Evolution



ALEVSK

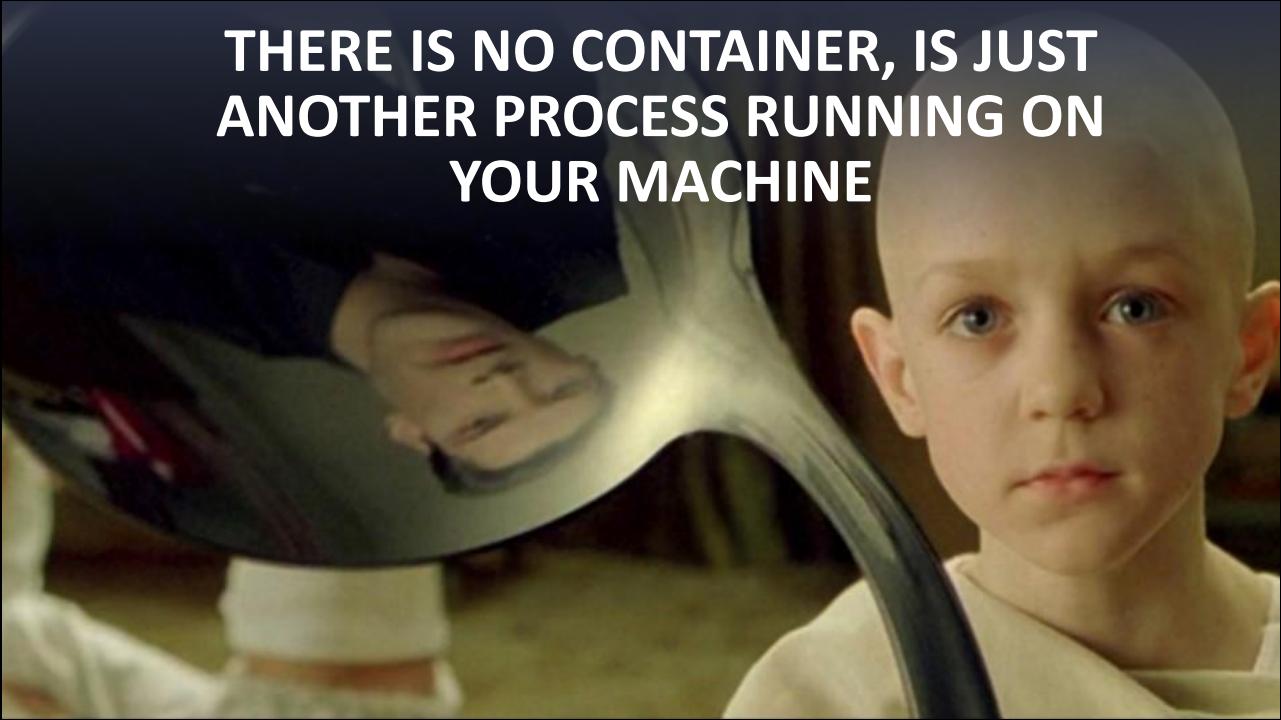
# Application Compartmentalization



ALEVSK

But first, let's talk about a little bit of the history of application compartmentalization.

- **Traditional Architecture:** In the traditional model if we want to have isolation between different applications and programs the solution was simple, put everything on separated machines, this was a straightforward solution however it come with a high cost, mostly on the hardware side. Migrating programs and regular maintenance was a challenge.
- **VM Architecture:** In the VM model a special type of software is introduced called the hypervisor, the hypervisor is capable of running multiple instances of an Operating System on top of the same hardware. This solution is more cost efficient compared to the previous one and offer a high level of Isolation between the different applications, however there's still some challenges when it come to the size of the virtual machines and having to manage multiple virtual operating systems.
- **Container Architecture:** Finally in the container model, applications packaged with all of its dependencies, libraries and configuration files together into a format called “a container”. These container are run using a container runtime. Containerized applications are in general faster and cheaper to run and have many advantages in comparison to the previous models (such as size) however using containers is not a silver bullet when it comes to process isolation.

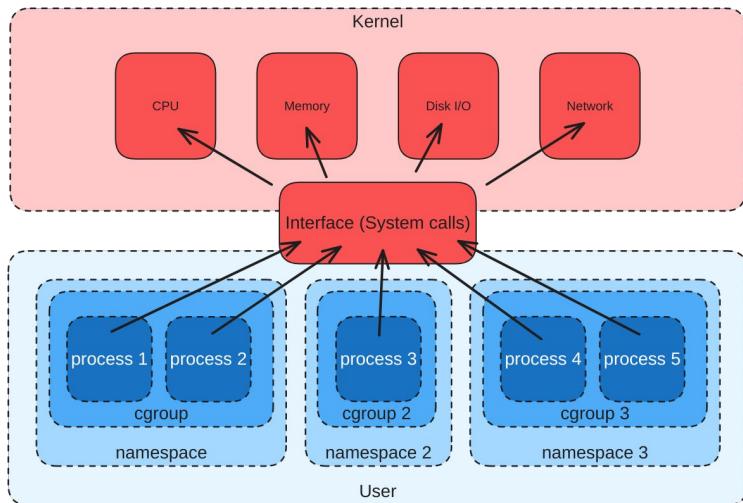


**THERE IS NO CONTAINER, IS JUST  
ANOTHER PROCESS RUNNING ON  
YOUR MACHINE**

I've been discussing about containers and some of their benefits and drawbacks, but what if I tell you there is no container, is just another process running on your machine

# So, What Containers Really Are?

- Namespaces
- Cgroups
- Capabilities



ALEVSK

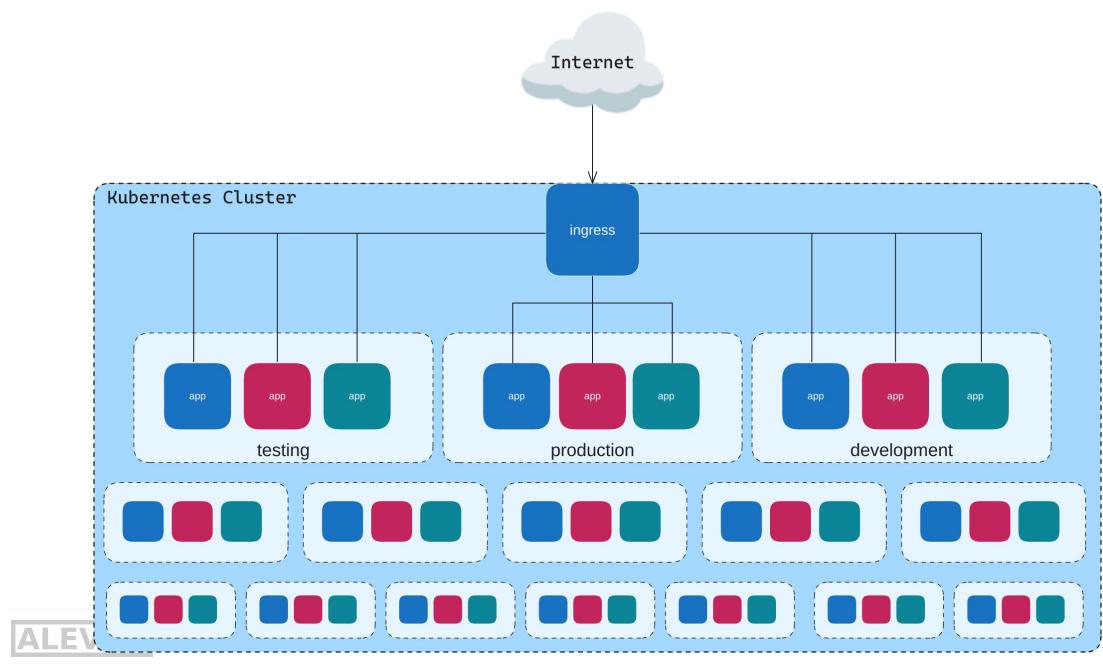
So, what containers really are? Containers are a very clever operating system level virtualization for running multiple “isolated” processes on your machine. You can think of a Sandbox for processes

This isolation is achieved using multiple features of the Linux kernel, ie:

- Namespaces: controls what the process can “see”
- cgroups (short for control groups) is a Linux kernel feature that limits and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.
- Capabilities: controls the different privileges a process can have, ie: CAP\_NET\_ADMIN (Allows various network-related operations), CAP\_SETUID (Allows changing the UID (User ID) and GID (Group ID))

# What Is Kubernetes Anyways?

ALEVSK



If Docker is a shipping container, Kubernetes is a fleet of cargo ships.

- Orchestration Tool
- Framework for building distributed systems
- Alternative way to deploy your Applications
- Operating system of the internet

As many containers as you want, at lightning speed!

## Kubernetes Primitives

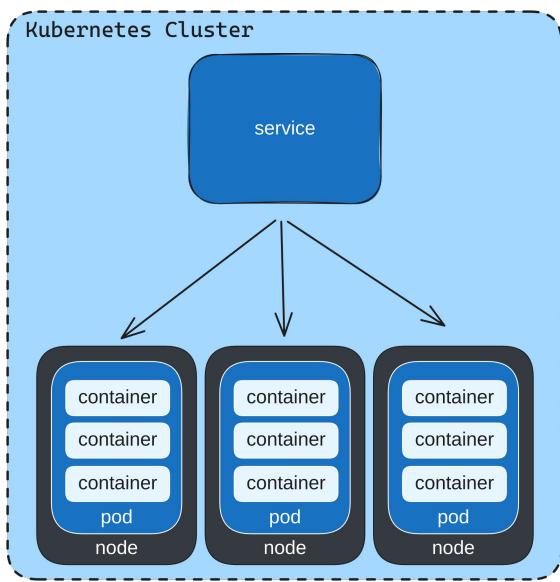


ALEVSK

Let's break it down

- Service: The way Kubernetes allows users to access your applications (the public IP address and port)
- Pod: Group of containers deployed together, they share a network namespace and can also share a file system namespace (webserver+backend+database+cache)
- Node: A virtual machine used to run pods (and by extension, containers)
- Cluster: A set of nodes that acts as the foundation of a Kubernetes application. Containers are grouped in pods which are allocated to and organized between nodes in a cluster.
- Deployment: Your cluster configuration, describes how your pods are allocated and managed

# Kubernetes Application



ALEVSK

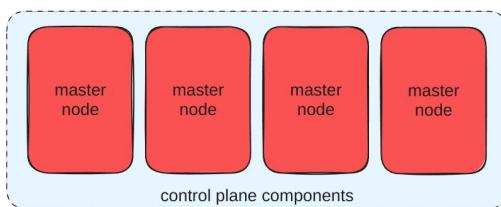
You have a single service, which all the user sees.

But behind that service is any number of nodes running any number of pods wrapping any number of containers each.

# Kubernetes Components

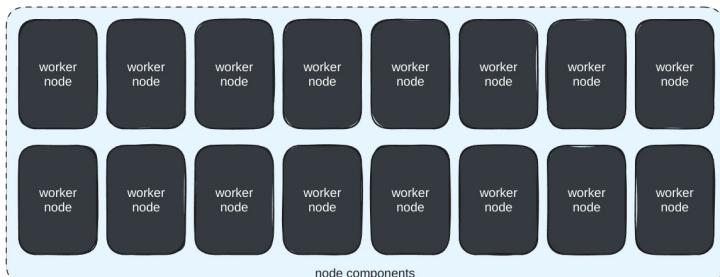
## Control Plane Components

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



## Node Components

- kube-proxy
- kubelet
- container runtime

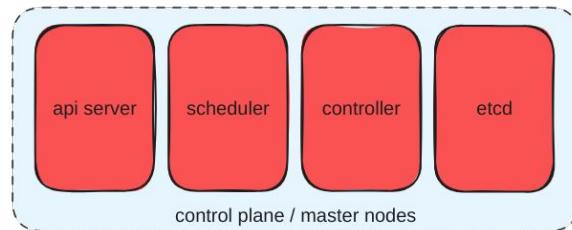


ALEVSK

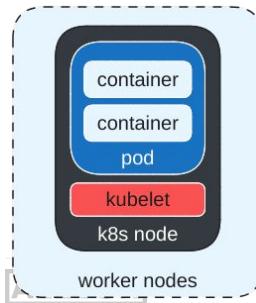
Jumping down a further layer of abstraction and we have the components, divided into two sections: control plane components (one per cluster) and node components (one per node)

- Kube-apiserver: The Kubernetes API server is the control interface. It validates and configures data for the api objects, including pods, services, replicationcontrollers, etc. The API Server exposes REST operations and provides the frontend to the cluster's shared state through which all other components interact.
- Etcd: data storage containing configmaps, secrets, states, etc
- Kube-scheduler: Responsible for scheduling newly created pods to the best available nodes in the cluster. It keeps the application workload evenly distributed among nodes depending on resource needs.
- Kube-controller: Ensures that the current state of running pods matches the desired state of deployments/statefulset/replica-set/etc.
- Kube-proxy: Proxies traffic between services and nodes.
- Kubelet: The primary "node agent", communicates with the API server and manages the node.
- Container runtime: Software that is responsible for running containers (Docker).

## Kubernetes Threat Model: If An Attacker Controls ...



**Control plane nodes:** Attacker controls your cluster. They can modify, access and destroy everything



**Pod / Container:** Attacker controls application and may be able to escape and attack the node

**Kubelet:** Attacker controls running pods

**Worker nodes:** Attacker controls running pods. They can attack master nodes

Components have a hierarchy, and therefore a varying attack surface.

Pod/Container -> Kubelet -> Worker Node -> Control Plane Node

## OWASP Kubernetes Risks Top (2022)

- K01: Insecure Workload Configurations
- K02: Supply Chain Vulnerabilities
- K03: Overly Permissive RBAC Configurations
- K04: Lack of Centralized Policy Enforcement
- K05: Inadequate Logging and Monitoring
- K06: Broken Authentication Mechanisms
- K07: Missing Network Segmentation Controls
- K08: Secrets Management Failures
- K09: Misconfigured Cluster Components
- K10: Outdated and Vulnerable Kubernetes Components

ALEVSK

With new technologies comes new attack surfaces, and with that comes new standards for defense.

# Kubernetes Most Common Attack Techniques

ALEVSK

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking	Denial of service
Application vulnerability	Application exploit (RCE)		Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files	
Exposed Dashboard	SSH server running inside container					Instance Metadata API	Writable volume mounts on the host	
							Access Kubernetes dashboard	
							Access tiller endpoint	

## Threat Matrix For Kubernetes (2020)

In 2020, security researchers at Microsoft published the first version of the “Threat Matrix For Kubernetes”, a MITRE ATT&CK-like matrix comprising the major techniques that are relevant to container orchestration, with focus on Kubernetes.

Although the attack techniques are different than those that target Linux or Windows systems, the tactics are actually similar, ie:

- “initial access to the computer” is similar to “initial access to the cluster”
- “malicious code on the computer” is similar to “malicious activity on the containers”
- “maintain access to the computer” is similar to “maintain access to the cluster”
- “gain higher privileges on the computer” similar to “gain higher privileges in the cluster”

<https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/>

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller			CoreDNS poisoning	
								ARP poisoning and IP spoofing	

## Threat Matrix For Kubernetes (2022)

ALEVSK

A second version of the matrix was published on 2022. The updated matrix add new techniques that were found by Microsoft researchers, as well as techniques that were suggested by the community. Also, several techniques that do not apply anymore were deprecated.

<https://www.microsoft.com/en-us/security/blog/2022/12/07/mitigate-threats-with-the-new-threat-matrix-for-kubernetes/#:~:text=The%20threat%20matrix%20for%20Kubernetes%20can%20help%20organizations%20to%20have,required%20to%20prevent%20these%20threats>

<https://microsoft.github.io/Threat-Matrix-for-Kubernetes/>

## Initial Access

- Using cloud credentials
- Compromised images and registry
- Kubeconfig file
- Application Vulnerability
- Exposed sensitive interfaces

ALEVSK



In Kubernetes environments, the initial access tactic consists of techniques that are used for enabling attackers their first access into the cluster. This access can be achieved directly via the cluster management layer or by gaining access to a malicious or vulnerable resource that is deployed on the cluster.

Examples of this techniques include phishing cloud credentials, supply chain attacks using compromised container registries and even exposed cluster credentials on public repositories.

## Using Cloud Credentials



Google Cloud



DigitalOcean

ALEVSK

When Kubernetes is deployed on the public cloud (AKS, EKS, or GCP) compromised cloud credentials can lead to cluster takeover via the management console

Attackers may try to gain access to those management consoles by using different phishing and social engineering techniques. Even in cases where the cloud accounts are protected with MFA attackers can still try to trick the users into giving away access using techniques like MFA bombing or MFA spamming (MFA fatigue) and get access to those privileged accounts.

## Compromised Registry And Images

- Supply Chain Attacks
- Vulnerable dependencies on images



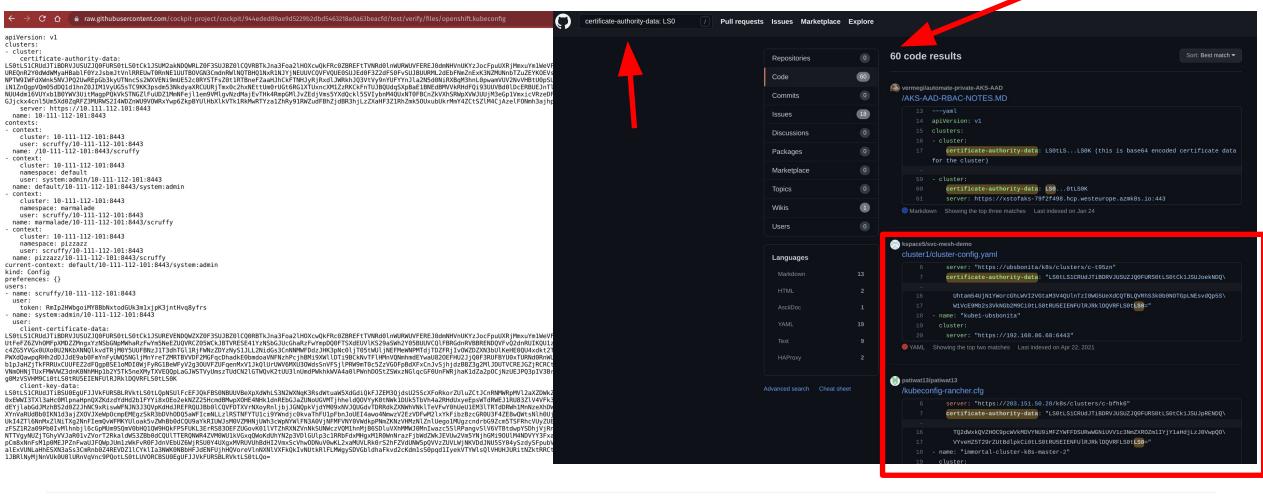
ALEVSK

Attackers who compromise a private registry can plant their own compromised images, running a compromised image could be the first step on the supply chain attack and could lead to compromise the entire cluster.

Examples of container registries:

- Docker hub
- jFrog artifactory
- Harbor

# Kubeconfig File - Hunting For .kube/config Files



ALEVSK

- If attackers get access to the `~/.kube/config` credentials, for instance via a compromised developer machine, they can use them to access clusters with the same level of privileges as the user to whom the credentials belong
  - These credentials are often exposed on public repositories like Github

## Application Vulnerabilities

- OWASP Top 10
- SQLi
- RCE
- Command injection
- Etc

A01:2021	Broken Access Control
A02:2021	Cryptographic Failures
A03:2021	Injection
A04:2021	Insecure Design
A05:2021	Security Misconfiguration
A06:2021	Vulnerable and Outdated Components
A07:2021	Identification and Authentication Failures
A08:2021	Software and Data Integrity Failures
A09:2021	Security Logging and Monitoring Failures
A10:2021	Server-Side Request Forgery

ALEVSK

At the end of the day we are running applications in our kubernetes clusters, running public-facing vulnerable applications (ie: OWASP Top 10) could be exploited by an attacker to gain initial access

## Exposed Sensitive Interfaces



VMware Tanzu



ALEVSK

Finally, exposing a sensitive interface to the internet poses a security risk. Examples of such interfaces that were seen exploited include Rancher, Kubeflow, Argo Workflows, and the Kubernetes API server.

## Execution

- Application exploit (RCE)
- Exec into container
- New container
- Sidecar Injection

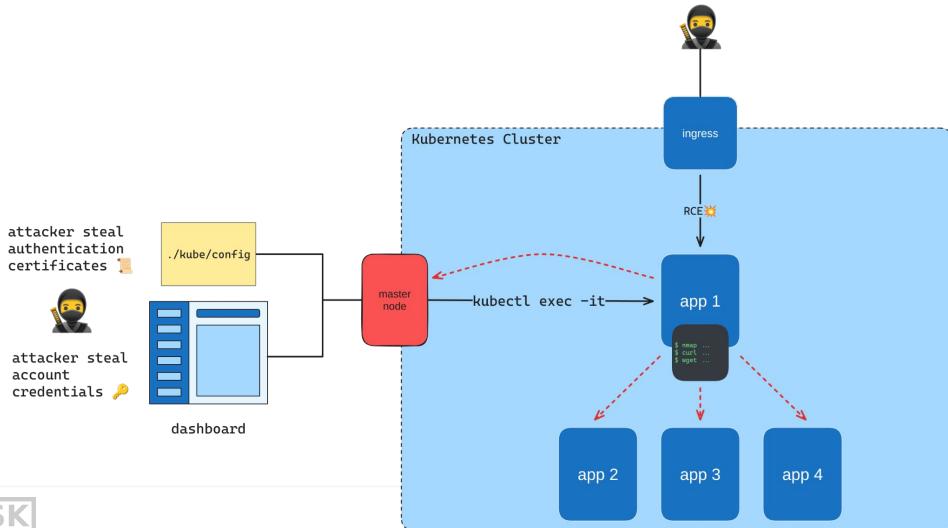
ALEVSK

```
dystatechange",H),e.attachE  
ng Function Array Date RegE  
){var t=_[e]={};return b.ea  
OnFalse){r=!1;break}n=!1,u&  
t,c(r))}return this},remove  
[],this},disable:function()  
rn p.fireWith(this,argument  
unction(){return n},always:  
).done(n.resolve).fail(n.re  
,t[1^e][2].disable,t[2][2].  
nts),r=n.length,i=1!==r||e&  
;t++);n[t]&&bisFunction(n[t  
a href='/a'>a</a><input typ  
0],r.style.cssText="top:1px  
e("style")),hrefNormalized:
```

The execution tactic consists of techniques that are used by attackers to run their code inside a cluster. It allows an attacker to gain a foothold in a cluster for further exploitation.

For example, a container may be running a vulnerable application that can be exploited remotely, allowing an attacker to take over said container.

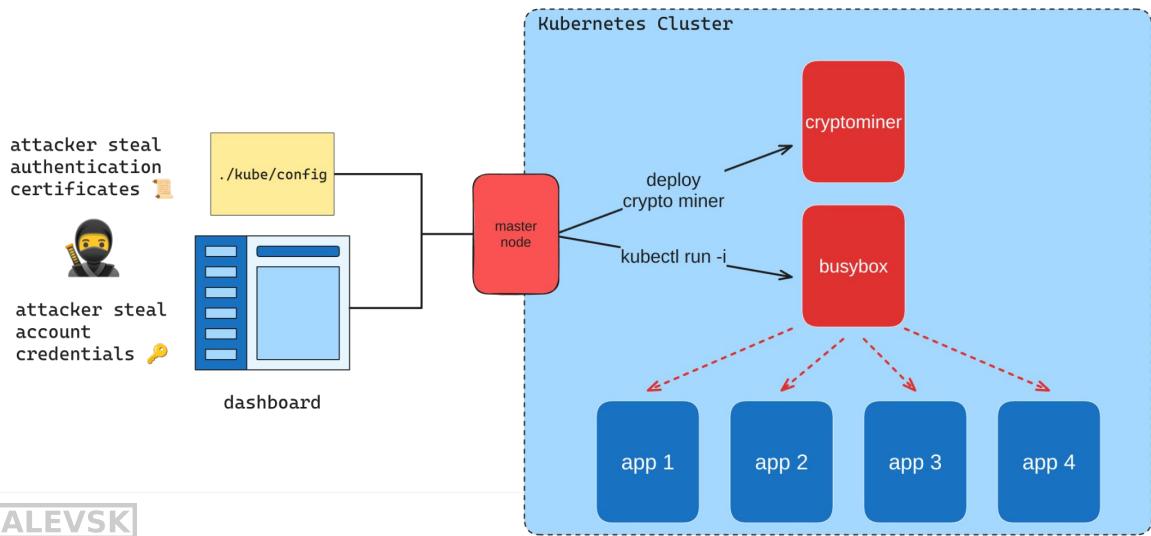
## Exec Into The Container



If an attacker obtains permissions to connect to a container through the API, they can run malicious commands remotely using “kubectl exec”. These commands give the attacker full control of said container, allowing them to execute arbitrary code and read secret keys.

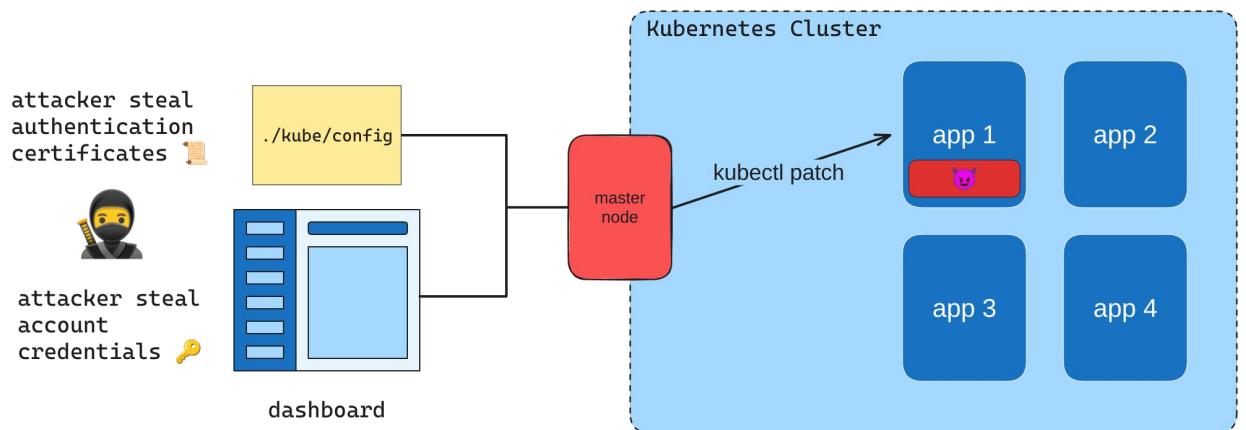
ALEVSK

# Deploying New Containers



Sometimes, an attacker may have access to the kubernetes API but cannot exec into existing containers, for whatever reason (ie: scratch containers). In these cases, the attacker can simply deploy their own container to execute code and interact with internal services.

## Sidecar Injection



ALEVSK

If an attacker wants to be especially sneaky, they can add a sidecar container to an existing pod in the cluster. Usually, pods have one main container, but they can also have many “sidecar” containers that run alongside it. These are typically harder to detect because they run in existing legitimate pod configurations.

## Persistence

- Backdoor container
- Kubernetes CronJob
- Writable hostPath mount
- Malicious admission controller

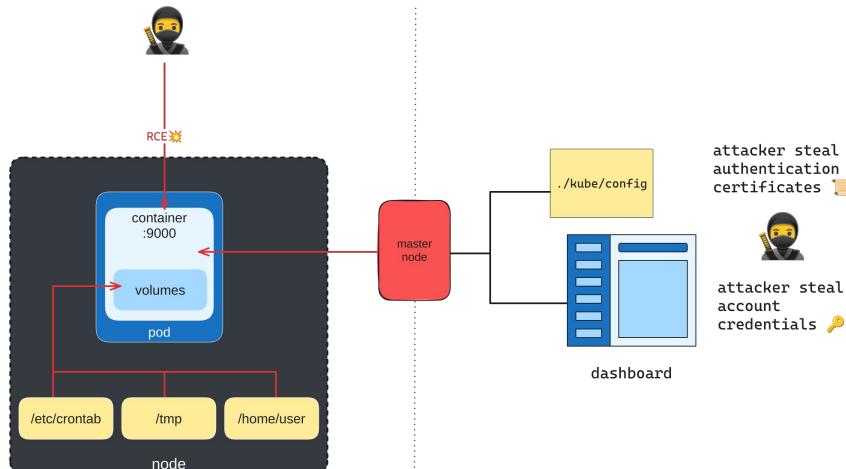
ALEVSK



The persistence tactic consists of techniques that are used by attackers to keep access to the cluster in case their initial foothold is lost.

- Backdoor container: Kubernetes is all about high availability, unfortunately that applies to attacker code as well. Attackers can use the Kubernetes controllers, such as DaemonSets or Deployments, to automatically deploy and maintain malicious containers running in the cluster.
- Kubernetes CronJob: Kubernetes Job is a controller that can make short-term, single run containers that perform specific jobs, and Kubernetes CronJob is used to schedule these jobs. Attackers can hijack CronJob and schedule their own malicious code to run in a container inside the cluster.

# Writable Hostpath Mount

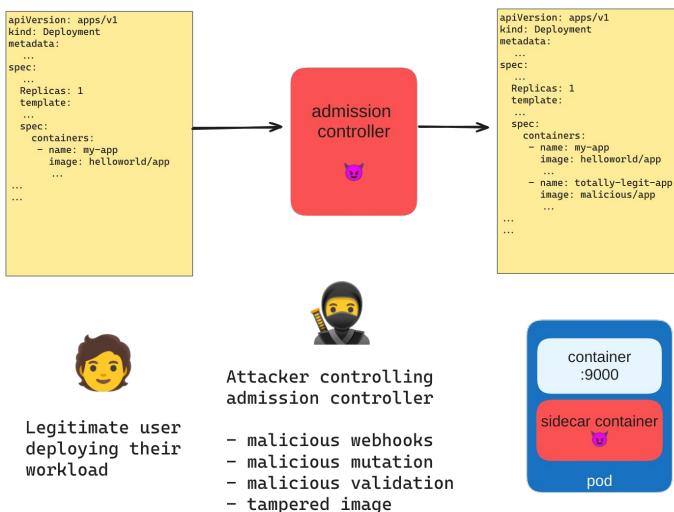


ALEVSK

A hostPath volume mounts a directory or a file from the host to the container. If an attacker can create a new container for code execution, they can potentially also create it with a writable hostPath volume. This gives the attacker access to host files and directories, which can provide persistence in a multitude of ways:

- A cronjob can be configured on the host to run malicious code
- Arbitrary commands can be added to the .bashrc file
- An SSH key can be added to the “authorized\_keys” file on the host

# Malicious Admission Controller

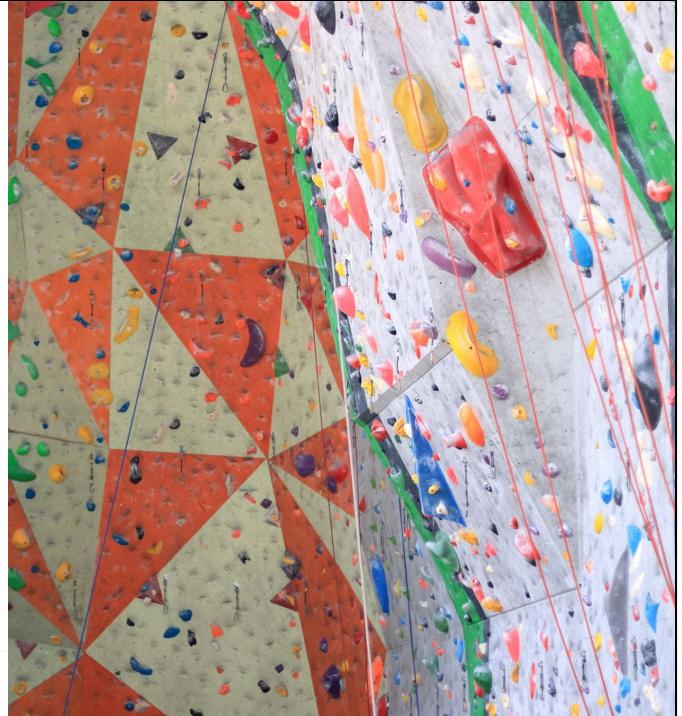


The admission controller is a Kubernetes component that intercepts, and possibly modifies, requests to the Kubernetes API server. Attackers can use such webhooks for gaining persistence in the cluster. For example, attackers can intercept and modify the pod creation operations in the cluster and add their malicious container to every created pod.

## Privilege Escalation

- HostPath mount
- Access cloud resources
- Privileged container
- Cluster-admin binding

ALEVSK



The privilege escalation tactic consists of techniques that are used by attackers to get higher privileges in the environment than those they currently have. In containerized environments, this can include getting access to the node from a container, gaining higher privileges in the cluster, and even getting access to the cloud resources

- HostPath mount: hostPath mount can be used by attackers to get access to the underlying host and thus break from the container to the host.
- Access cloud resources: depending on the cloud provider different services might be available from inside the cluster
- Privileged container: Attackers who gain access to a privileged container, or have permissions to create a new privileged container (by using the compromised pod's service account, for example), can get access to the host's resources.
- Cluster-admin binding: RBAC can restrict the allowed actions of the various identities in the cluster. Cluster-admin is a built-in high privileged role in Kubernetes

## Privileged Container



Duffie Cooley

@mauilion

...

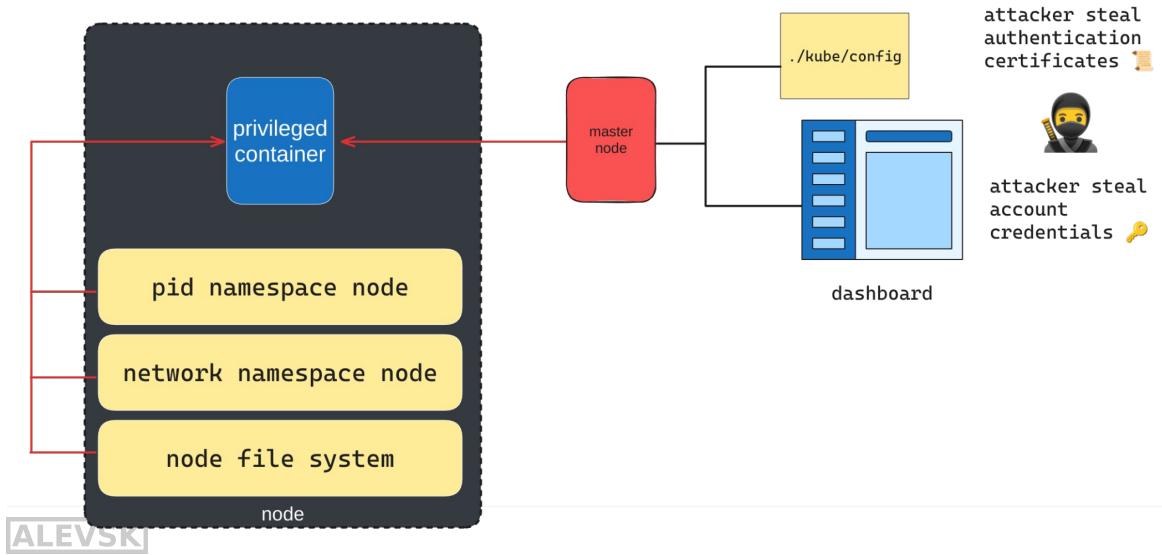
```
kubectl run r00t --restart=Never -ti --rm --image lol --overrides '{"spec":{"hostPID": true, "containers": [{"name":"1","image":"alpine","command":["nsenter","--mount=/proc/1/ns/mnt","--","/bin/bash"],"stdin":true,"tty":true,"securityContext":{"privileged":true}}]}{'
```

12:27 PM · May 17, 2019 · Twitter Web Client

ALEVSK

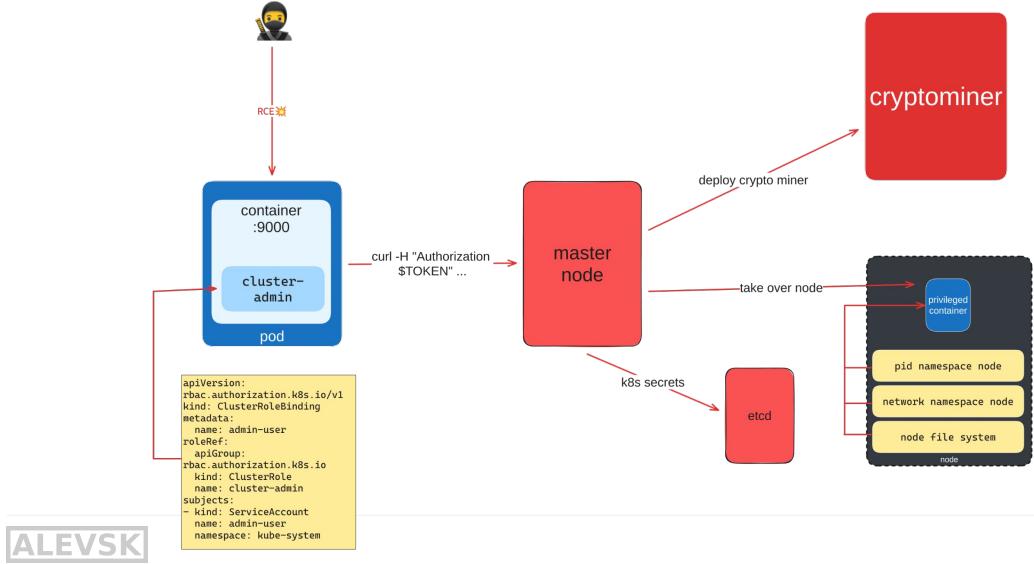
- The infamous 280 characters tweet that will allow you to hack Kubernetes clusters:
- ```
kubectl run r00t --restart=Never -ti --rm --image lol --overrides '{"spec":{"hostPID": true, "containers": [{"name":"1","image":"alpine","command":["nsenter","--mount=/proc/1/ns/mnt","--","/bin/bash"],"stdin":true,"tty":true,"securityContext":{"privileged":true}}]}{'
```
- Take a look at this command, take a picture and try to run it later on your kubernetes cluster if you want

# A Container That Doesn't Contain Anything



- So, how the privilege escalation command works?
- Basically an attacker with enough permissions could (via the management console or using kubectl directly) deploy a privileged container in the cluster
- A privileged container would have access to all the operating system processes and the node network namespace. Additionally, the node file system could be mounted directly into the privilege container giving the attacker access to every file on the system.
- This is essentially a container that doesn't contain anything
- From there an attacker could try to move laterally to other nodes and even try to reach the control plane

# Cluster-Admin Binding



- Sometimes privileged containers are already running inside the cluster. Containers running using a privilege role such as the “Cluster-admin” are high value targets for attackers
- The Cluster-admin role would be the equivalent to a “superuser” on a Linux system because it can perform any action on any resource in a cluster.
- In this example, an attacker that successfully steal the service account token associated to the cluster-admin role may be able to perform any task on the cluster, including, stealing secrets from etcd, deploying cryptominers, and take over the entire cluster.

## Defense Evasion

- Clear container logs
- Delete Kubernetes events
- Pod / Container name similarity
- Connect from Proxy server

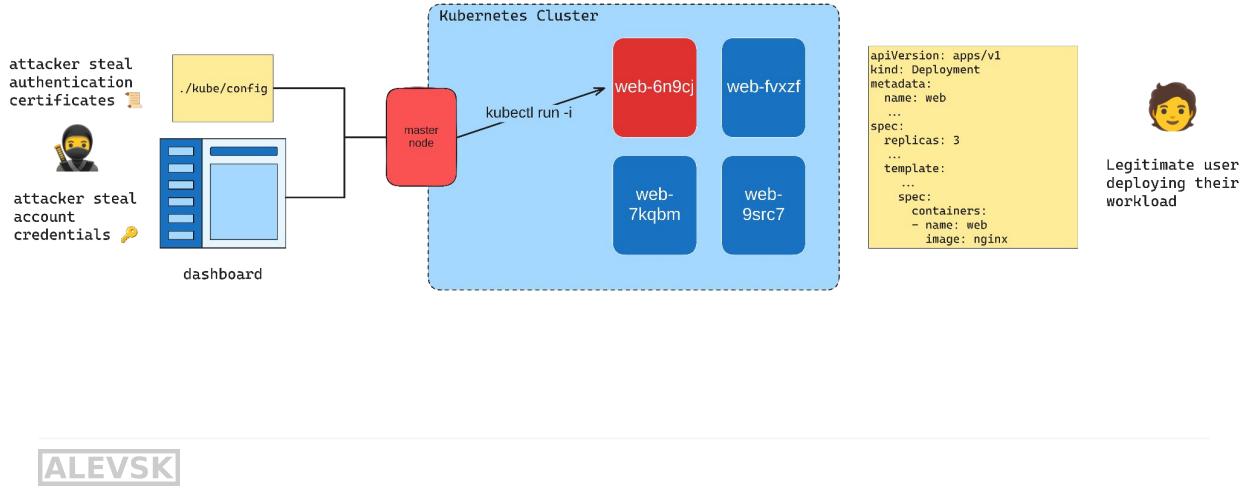
ALEVSK



The defense evasion tactic consists of techniques that are used by attackers to avoid detection and hide their activity.

- Clear container logs: Attackers can delete the application or OS logs on a compromised container, hiding their container activity.
- Delete Kubernetes events: Kubernetes events track changes that occur inside the cluster. Attackers can delete these events ("kubectl delete events--all") to hide their cluster activity.
- Pod / Container name similarity: Attackers can name their pods as if they were created by the existing controllers. For example, an attacker could create a malicious pod named coredns-{random suffix} which would look related to the CoreDNS Deployment.

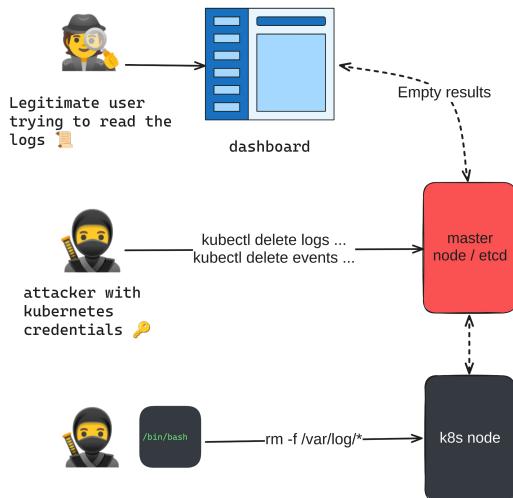
# Pod / Container Name Similarity



ALEVSK

By exploiting the naming convention of Kubernetes pods, such as the random suffix (e.g., `web-9src8`) added to each pod in a ReplicaSet, attackers may attempt to create malicious pods that mimic the names of legitimate applications running in the cluster. The goal is to deceive and confuse cluster administrators. This evasion technique is especially effective when the cluster lacks proper monitoring tools.

## Delete Container Logs & Kubernetes Events



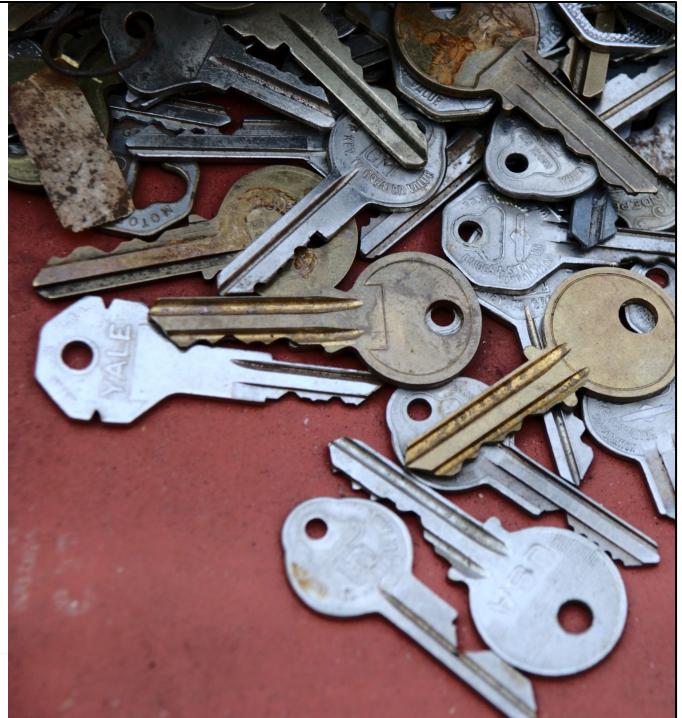
ALEVSK

An attacker with sufficient privileges in the cluster can delete container logs and Kubernetes events to cover their tracks, complicating the investigation for incident response teams. This is particularly problematic if the logs are not replicated outside the cluster, such as being sent to a SIEM or another log analysis solution.

## Credential Access

- List Kubernetes secrets
- Mount Service Principal
- Access container service account
- Applications credentials in configuration files
- Access managed identity credential
- Malicious admission controller

ALEVSK

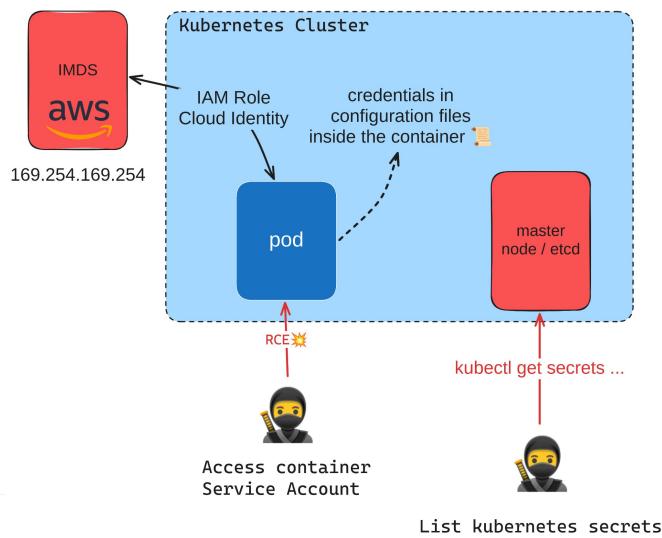


The credential access tactic consists of techniques that are used by attackers to steal credentials.

In kubernetes environments, this includes:

- Kubernetes secrets – objects that store sensitive information
- Service Principal credentials – cloud credentials. Attackers can sometimes leverage their access to a container to obtain these credentials. In AKS, for example, each node contains a service principal credential, so access to a node means access to this credential.
- Container service account credentials: Service Accounts represent applications in Kubernetes. By default, an SA is mounted to every created pod in the cluster.
- Applications credentials: Developers sometimes store secrets in the Kubernetes configuration files, such as environment variables in the pod configuration. This makes them very easy to steal, please don't do this!
- managed identity credentials – cloud credentials. Applications can obtain the token for a managed identity by accessing the Instance Metadata Service (IMDS). Attackers with access to a pod can query the IMDS to obtain a managed identity token and access cloud resources.
- Malicious admission controller: In addition to maintaining persistence, the admission controller can be used to read sensitive information in API calls.

## Access managed identity credential



Attackers may attempt to gather credentials from various sources. The most common method, if the attacker has sufficient Kubernetes permissions, is to list all secrets in the cluster using the `kubectl get secrets` command. Alternatively, if the attacker has arbitrary command execution on a running container, they will try to locate credentials mounted within the container, such as configuration files, database credentials, API keys, OAuth tokens, etc.

If Kubernetes is running on a managed service like EKS, GKE, or AKS, attackers may attempt to retrieve the cloud identity associated with the running pod by fetching identity credentials from the metadata service (e.g., IMDS in the case of EKS).

## Discovery

- Access the Kubernetes API server
- Access Kubelet API
- Network mapping
- Access Kubernetes dashboard
- Instance Metadata API

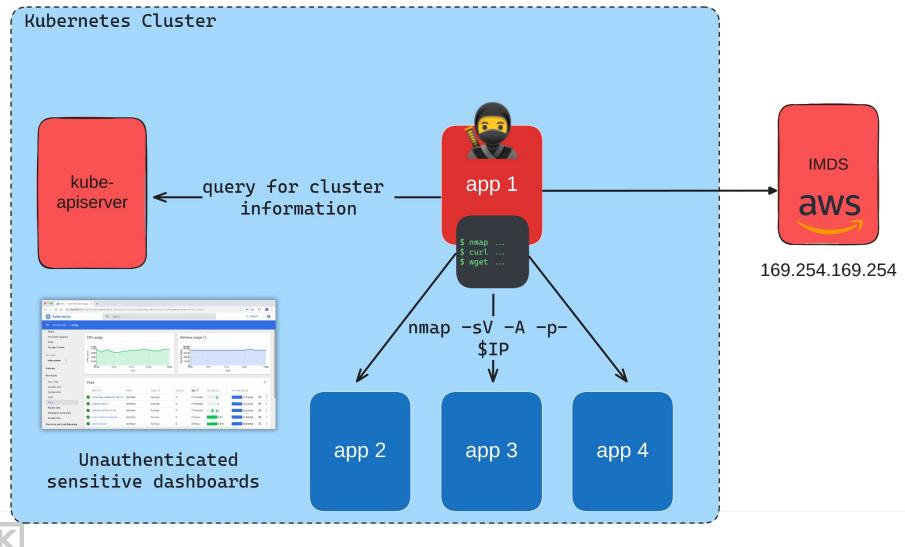


ALEVSK

The discovery tactic consists of techniques that are used by attackers to explore the environment to which they gained access. This exploration helps the attackers to perform lateral movement and gain access to additional resources.

- Access the Kubernetes API server: The Kubernetes API server is the gateway to the cluster. Attackers can send requests to the API server to probe the cluster and get information about pods, containers, secrets, and other resources.
- Access Kubelet API: the Kubelet exposes a read-only API service that does not require authentication (TCP port 10255). Attackers with network access to the host can send API requests to the Kubelet API and retrieve information about running pods and the node itself.
- Network mapping: By default, there is no restriction on pod communication in Kubernetes. Attackers can freely map the cluster network to get information on the running applications.
- Access Kubernetes dashboard: If an attacker steals service account credentials, they can use the Kubernetes dashboard (a Web-based UI) to perform actions in the cluster.
- Instance Metadata API: Cloud providers provide an instance metadata service for retrieving information about a virtual machine. Attackers who gain access to a container can query this service and obtain information about the underlying node the container is running on.

# Kubernetes Service



Once an attacker reaches the discovery phase and gains internal access to the cluster, they can use traditional network scanning methods to enumerate Kubernetes services running inside the cluster, hoping to find services with unauthenticated sensitive interfaces to escalate privileges. Additionally, if the attacker has credentials, they can query the kube-apiserver to gather information about workloads, such as container image versions, which can be used to identify known vulnerabilities. For Kubernetes clusters running on EKS, GKE, AKS, or other managed offerings, attackers may query the Instance Metadata Service (IMDS) to obtain information about the cloud identity of the current workload.

## Lateral Movement

- Access cloud resources
- Container service account
- Cluster internal networking
- Applications credentials in configuration files
- Writable volume mounts on the host
- CoreDNS poisoning

ALEVSK

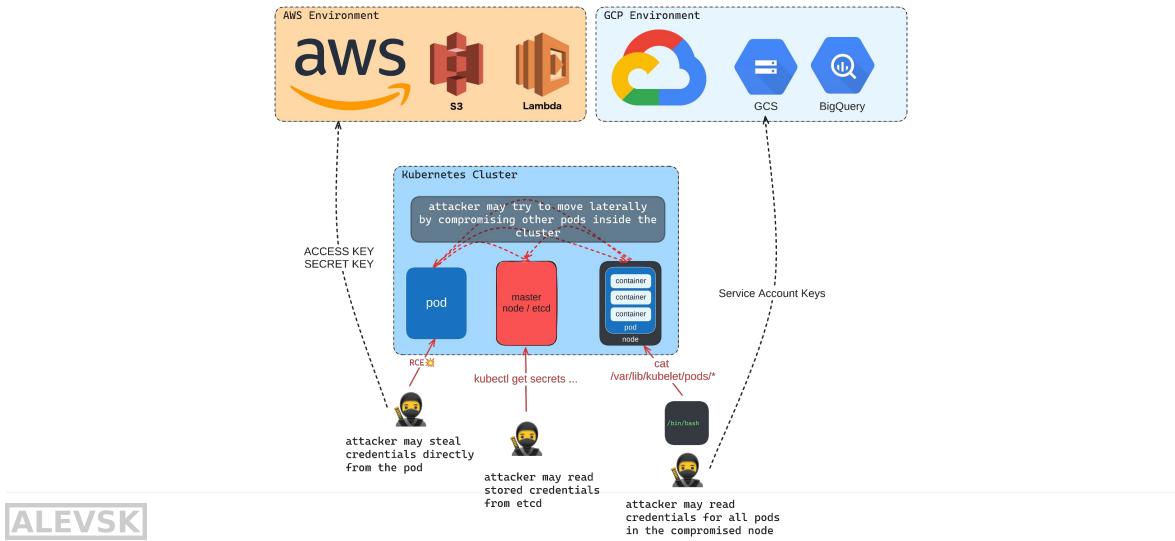


The lateral movement tactic consists of techniques that are used by attackers to move through the victim's environment. This includes gaining access to various resources in the cluster, gaining access to the underlying node from a container, or gaining access to the cloud environment itself.

- Access cloud resources: Attackers may move from a compromised container to the cloud environment.
- Container service account: Attackers who gain access to a container in the cluster may use the mounted service account token for sending requests to the API server, and gaining access to additional resources in the cluster.
- Cluster internal networking: Kubernetes networking behavior allows traffic between pods in the cluster as a default behavior. Attackers who gain access to a single container may use it for network reachability to another container in the cluster.
- Applications credentials in configuration files: Developers store secrets in the Kubernetes configuration files, for example, as environment variables in the pod configuration. Using those credentials attackers may gain access to additional resources inside and outside the cluster. (See “6: Application credentials in configuration files” for more details.)
- Writable volume mounts on the host: Attackers may attempt to gain access to the underlying host from a compromised container.
- CoreDNS poisoning: CoreDNS is the main DNS service used in Kubernetes. The configuration file (coorefile) is stored in a ConfigMap object, located at the kube-system namespace. If attackers have permissions to modify the

- take the network identity of other services.

# Lateral Movement and Credential Theft in Kubernetes



## Lateral Movement and Credential Theft in Kubernetes

- An attacker with arbitrary command execution on a container can steal credentials stored within the running pod.
- An attacker with sufficient privileges can retrieve credentials directly from etcd using the `kubectl get secrets` command.
- If an attacker compromises a worker node, they automatically gain access to all secrets associated with the pods running on that node.

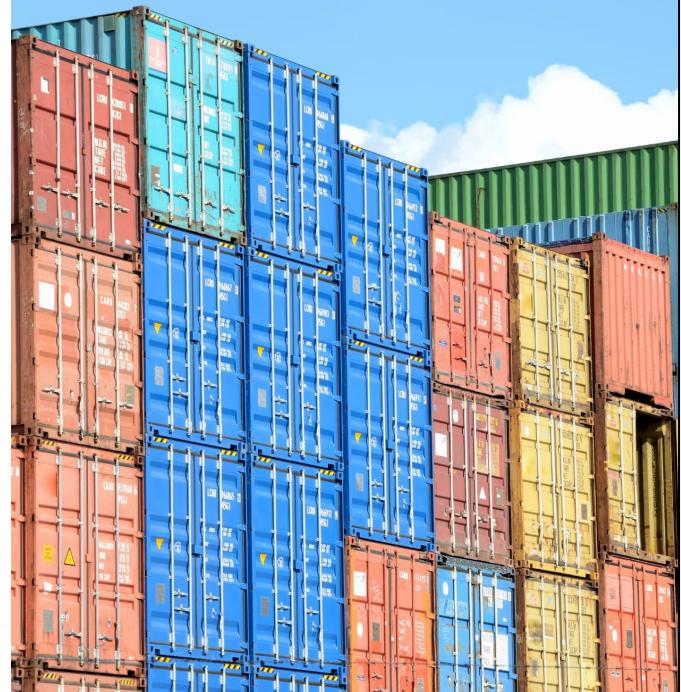
By obtaining additional credentials, an attacker can attempt lateral movement to compromise more workloads within the cluster or gain access to resources in other cloud environments.

## Collection

- Images from private registry



ALEVSK



In Kubernetes, collection consists of techniques that are used by attackers to collect data from the cluster or through using the cluster.

- Images from private registry: The images that are running in the cluster can be stored in a private registry. For pulling those images, the container runtime engine (such as Docker or containerd) needs to have valid credentials to those registries. If the registry is hosted by the cloud provider, in services like Azure Container Registry (ACR) or Amazon Elastic Container Registry (ECR), cloud credentials are used to authenticate to the registry. If attackers get access to the cluster, in some cases they can obtain access to the private registry and pull its images.

## Impact

- Data destruction
  - Deleting PV, logs, events, etc
- Resource Hijacking
  - Deploying cryptominers, torrents, etc
- Denial of service
  - Deleting pods, deployments, sts, services or ingress rules

ALEVSK



The Impact tactic consists of techniques that are used by attackers to destroy, abuse, or disrupt the normal behavior of the cluster.

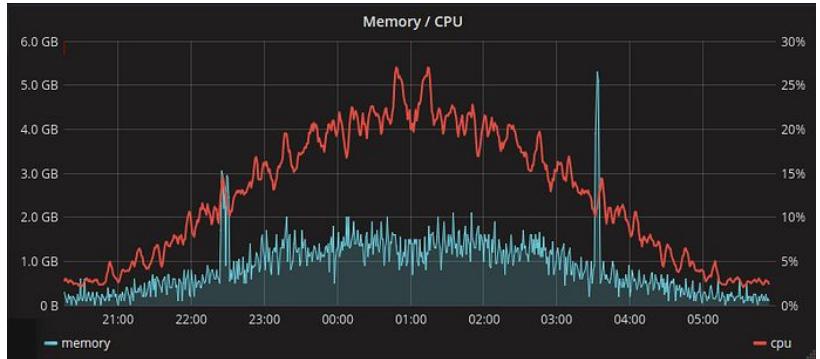
- Data destruction: Attackers may attempt to destroy data and resources in the cluster. This includes deleting deployments, configurations, storage, and compute resources.
- Resource Hijacking: Attackers may abuse a compromised resource for running tasks. A common abuse is to use compromised resources for running digital currency mining. Attackers who have access to a container in the cluster or have permissions to create new containers may use them for such activity.
- Denial of service: Attackers may attempt to perform a denial of service attack, which makes the service unavailable to the legitimate users. In container clusters, this include attempts to block the availability of the containers themselves, the underlying nodes, or the API server.

# Kubernetes Build-In Defenses

ALEVSK

# Resource Quotas and Limits

- Limits
  - CPU, Memory, etc.
- Requests
  - CPU, Memory, etc.



ALEVSK

When you configure a Pod, you can optionally specify how much of each resource each container needs. Most commonly, CPU and memory resources.

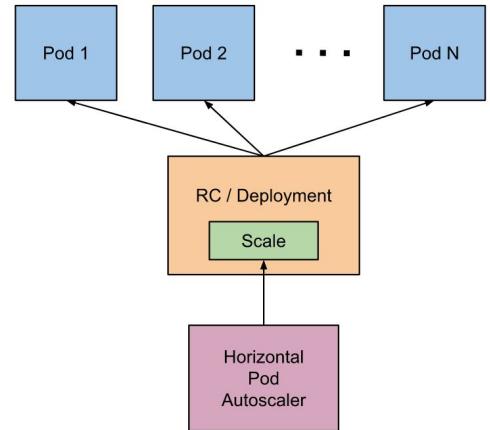
- The “resource request” configuration tells the kubelet how many resources to reserve specifically for a given container (lower limit).
- The “resource limit” configuration tells the kubelet that this container is not allowed to use any more of a resource than the limit you set (upper limit).
- The kubelet is in charge of enforcing these two values.
- This is a simply way to prevent malicious cryptominers.

<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

## Horizontal Pod Autoscaling

Horizontal Pod Autoscaling (HPA) in Kubernetes is a feature that automatically adjusts the number of pods in a deployment, replica set, or stateful set based on the current load or resource usage.

**Tip:** You can also use [PodDisruptionBudget](#) to avoid accidental outages



ALEVSK

Horizontal Pod Autoscaling (HPA) in Kubernetes is a feature that automatically adjusts the number of pods in a deployment, replica set, or stateful set based on the current load or resource usage.

# SecurityContext For Pods

- Privilege and Access Control
  - allowPrivilegeEscalation, privileged, runAsNonRoot, runAsUser, runAsGroup
- Capabilities Management
  - Capabilities: add, drop
- Filesystem and Mount Settings
  - readOnlyRootFilesystem
- Security Profiles
  - seLinuxOptions, seccompProfile, windowsOptions

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: security-context-demo
5 spec:
6   securityContext:
7     runAsUser: 1000
8     runAsGroup: 3000
9     fsGroup: 2000
10   volumes:
11     - name: sec-ctx-vol
12       emptyDir: {}
13   containers:
14     - name: sec-ctx-demo
15       image: busybox
16       command: [ "sh", "-c", "sleep 1h" ]
17       volumeMounts:
18         - name: sec-ctx-vol
19           mountPath: /data/demo
20           securityContext:
21             allowPrivilegeEscalation: false
```

ALEVSK

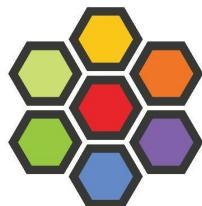
Security context defines privilege and access control settings for a Pod or Container. Security context settings include, but are not limited to:

- AllowPrivilegeEscalation: controls whether processes in a container can gain more privileges than their parent process
- privileged: processes in privileged containers essentially have root control over the host Node.
- runAsNonRoot, runAsUser, runAsGroup: enforce process owners.
- Capabilities: give a user a subset of root privileges
- Filesystem: control which filesystems and volumes are read-only
- Security Profiles: Apply SELinux, system call filtering with seccomp, etc.

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

# Network Policies

 flannel



 Istio

ALEVSK

Network Policies allow you to specify how a pod is allowed to communicate with various "endpoints" and "services" over the network. They control who can talk to who.

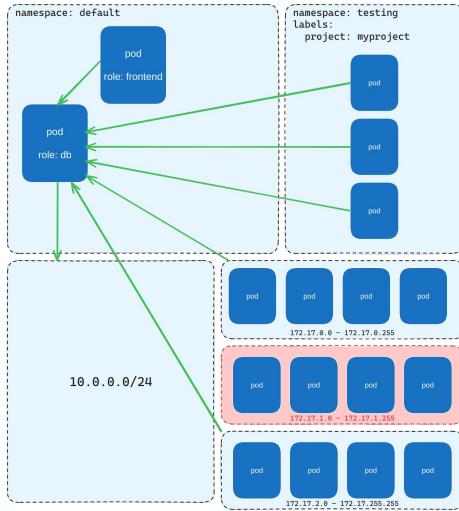
Network policies are implemented by the network plugin or CNI. To use network policies, that CNI must support network policies.

- kubenet is the default CNI plugin for kubernetes
- Popular CNIs: flannnel, calico, Cilium, istio CNI

<https://kubernetes.io/docs/concepts/services-networking/network-policies/>

# Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
        - namespaceSelector:
            matchLabels:
              project: myproject
            podSelector:
              matchLabels:
                role: frontend
    ports:
      - protocol: TCP
        port: 6379
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
    ports:
      - protocol: TCP
        port: 5978
```



ALEVSK

- By default, pods are non-isolated, they accept traffic from any source.
- Once there is any NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not explicitly allowed by the NetworkPolicy.
- Here we have a podSelector that selects all pods with the role “db”. We can enforce both inbound and outbound traffic in our policy.
- For ingress, we only accept connections from 3 places: the IP block 172.17.0.0/16 excluding 172.17.1.0/24, the kubernetes project “myproject”, and pods with the role “frontend”. We also only accept inbound connections to TCP port 6379.
- For egress, we only allow connections made to the IP block 10.0.0.0/24, and only coming from port 5978.

## Encryption In Transit

- Kubernetes Certificate Authority
  - certificates.k8s.io



- Cert-manager.io
- Service mesh



- Istio, Linkerd, Consul, etc

ALEVSK

- By default, traffic between Pods is unencrypted, meaning kubernetes traffic can be sniffed for secrets.
- Pods can be configured with TLS certificates, allowing their traffic to be encrypted. The CA for these certs is 'certificates.k8s.io'.
- These certs can also be generated and managed automatically by services like cert-manager.io.
- Service mesh?

<https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/>

## Encryption At Rest With EncryptionConfiguration

- Enable at-rest encryption for:
    - secrets, configmaps, events, \*.apps, \*\*, etc.
  - Supports:
    - Identity (plain text), AESCBC, AESGCM
  - External KMS

```
10 apiVersion: apiserver.config.k8s.io/v1
 9 kind: EncryptionConfiguration
 8 resources:
 7   - resources:
 6     - secrets
 5     providers:
 4     - deshbc:
 3       keys:
 2         - name: key1
 1           secret: 6YbxX+1+UK9ym6BP2tLV10JqYYZguVgzgMnrew7oK
 0   - identity: {}
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 678
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 999
 1000
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1298
 1299
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1389
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1398
 1399
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1489
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1498
 1499
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1518
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1529
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1569
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1579
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1589
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1598
 1599
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1609
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1618
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1629
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1638
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1659
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1669
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1679
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1689
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1698
 1699
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1709
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1718
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1729
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1738
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1769
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1779
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1789
 1789
 1790
 1791
 1792
 1793
 1794
 1795
 1796
 1797
 1798
 1798
 1799
 1799
 1800
 1801
 1802
 1803
 1804
 1805
 1806
 1807
 1808
 1809
 1809
 1810
 1811
 1812
 1813
 1814
 1815
 1816
 1817
 1818
 1818
 1819
 1820
 1821
 1822
 1823
 1824
 1825
 1826
 1827
 1828
 1829
 1829
 1830
 1831
 1832
 1833
 1834
 1835
 1836
 1837
 1838
 1838
 1839
 1840
 1841
 1842
 1843
 1844
 1845
 1846
 1847
 1848
 1848
 1849
 1850
 1851
 1852
 1853
 1854
 1855
 1856
 1857
 1858
 1859
 1859
 1860
 1861
 1862
 1863
 1864
 1865
 1866
 1867
 1868
 1869
 1869
 1870
 1871
 1872
 1873
 1874
 1875
 1876
 1877
 1878
 1879
 1879
 1880
 1881
 1882
 1883
 1884
 1885
 1886
 1887
 1888
 1889
 1889
 1890
 1891
 1892
 1893
 1894
 1895
 1896
 1897
 1898
 1898
 1899
 1899
 1900
 1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1909
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1918
 1918
 1919
 1920
 1921
 1922
 1923
 1924
 1925
 1926
 1927
 1928
 1929
 1929
 1930
 1931
 1932
 1933
 1934
 1935
 1936
 1937
 1938
 1938
 1939
 1940
 1941
 1942
 1943
 1944
 1945
 1946
 1947
 1948
 1948
 1949
 1950
 1951
 1952
 1953
 1954
 1955
 1956
 1957
 1958
 1959
 1959
 1960
 1961
 1962
 1963
 1964
 1965
 1966
 1967
 1968
 1969
 1969
 1970
 1971
 1972
 1973
 1974
 1975
 1976
 1977
 1978
 1979
 1979
 1980
 1981
 1982
 1983
 1984
 1985
 1986
 1987
 1988
 1989
 1989
 1990
 1991
 1992
 1993
 1994
 1995
 1996
 1997
 1998
 1998
 1999
 1999
 2000
 2001
 2002
 2003
 2004
 2005
 2006
 2007
 2008
 2009
 2009
 2010
 2011
 2012
 2013
 2014
 2015
 2016
 2017
 2018
 2019
 2019
 2020
 2021
 2022
 2023
 2024
 2025
 2026
 2027
 2028
 2029
 2029
 2030
 2031
 2032
 2033
 2034
 2035
 2036
 2037
 2038
 2038
 2039
 2040
 2041
 2042
 2043
 2044
 2045
 2046
 2047
 2048
 2048
 2049
 2050
 2051
 2052
 2053
 2054
 2055
 2056
 2057
 2058
 2059
 2059
 2060
 2061
 2062
 2063
 2064
 2065
 2066
 2067
 2068
 2069
 2069
 2070
 207
```

ALEVSK

`EncryptionConfiguration` covers at-rest encryption for resource data stored using the Kubernetes API.

You can also use an external KMS for secret management.

- Use a storage integration that provides encrypted volumes, or
  - Encrypt the data within your own application

**IMPORTANT:** This will NOT encrypt filesystems mounted to containers. To do that you need to:

<https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/>

## RBAC & Pod Security Policies (PSPs)

PSPs were deprecated in Kubernetes v1.21. Removed from Kubernetes in v1.25.



ALEVSK

When it comes to access control, PSP used to be a mechanisms to manage access to the different resources in the cluster

- PSPs objects are cluster-level resources that define the conditions pods must satisfy in order to be admitted into the cluster
- PSPs are enforced using an optional Kubernetes admission controller, if enabled, any attempts to create pods that do not satisfy authorized PSPs will be denied
  - PodSecurityPolicy: This was like a set of rules that determined what security conditions a pod must meet to run in your Kubernetes cluster
  - ClusterRole: Define a list of permissions that can be applied cluster-wide
  - ClusterRoleBinding: Ties the ClusterRole to users, groups, or service accounts, allowing them to use the permissions defined in the ClusterRole (doesn't limit to a single namespace)
  - RoleBinding: A RoleBinding connects roles to users, groups, or service accounts within a namespace.
  - Role: A Role is also a set of permissions but limited to a specific namespace within the cluster. It defines what actions a user, group, or service account can perform within that namespace

# Protect Cluster Against Privilege Pods

## PodSecurityPolicy

```
1 apiVersion: policy/v1beta1
2 kind: PodSecurityPolicy
3 metadata:
4   name: restrictive
5 spec:
6   privileged: false
7   hostNetwork: false
8   allowPrivilegeEscalation: false
9   defaultAllowPrivilegeEscalation: false
10  hostPID: false
11  hostIPC: false
12  runAsUser:
13    rule: RunAsAny
14  fsGroup:
15    rule: RunAsAny
16  seLinux:
17    rule: RunAsAny
18  supplementalGroups:
19    rule: RunAsAny
20  volumes:
21    - 'configMap'
22    - 'downwardAPI'
23    - 'emptyDir'
24    - 'persistentVolumeClaim'
25    - 'secret'
26    - 'projected'
```

## ClusterRole

```
1 kind: ClusterRole
2 apiVersion: rbac.authorization.k8s.io/v1
3 metadata:
4   name: psp-restrictive
5 rules:
6   - apiGroups:
7     - extensions
8     - resources:
9       - podsecuritypolicies
10      resourceNames:
11        - restrictive
12      verbs:
13        - use
```

## ClusterRoleBinding

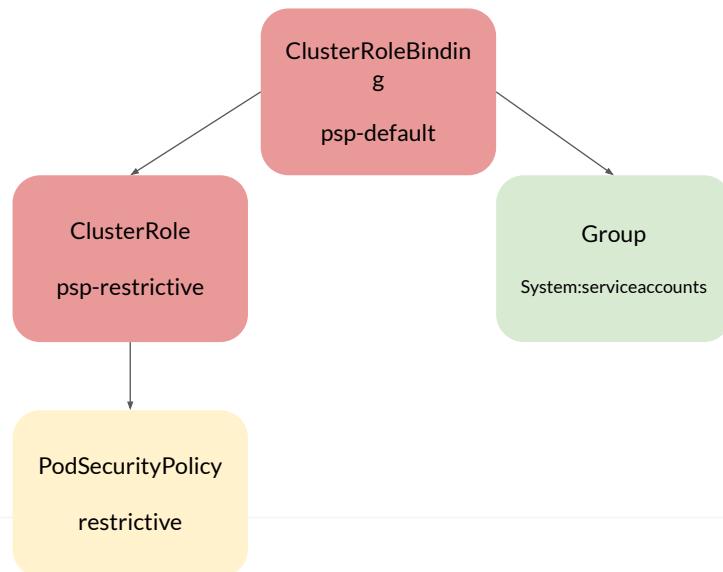
```
1 kind: ClusterRoleBinding
2 apiVersion: rbac.authorization.k8s.io/v1
3 metadata:
4   name: psp-default
5 subjects:
6   - kind: Group
7     name: system:serviceaccounts
8   namespace: kube-system
9 roleRef:
10  kind: ClusterRole
11  name: psp-restrictive
12  apiGroup: rbac.authorization.k8s.io
```

ALEVSK

Let say we want to prevent users to run privilege pods in your cluster.

- The first step would be to create a PSP that add all necessary rules to restrict our pod from using the host network and process namespace, among other things
- Second, we will have to create a ClusterRole to allow the use of the restrictive PSP
- Third, we create a ClusterRoleBinding to tie together the ClusterRole and a group, user or service account at the cluster level

# Protect Cluster Against Privilege Pods



ALEVSK

Visually the PSP configuration would look like the following tree structure

# Protect Cluster Against Privilege Pods

```
2 kind: Deployment
3 metadata:
4   name: nginx-hostnetwork-deployment
5   namespace: default
6   labels:
7     app: nginx
8 spec:
9   replicas: 1
0   selector:
1   matchLabels:
2     app: nginx
3   template:
4     metadata:
5       labels:
6         app: nginx
7     spec:
8       containers:
9         - name: nginx
0           image: nginx:1.15.4
1           hostNetwork: true
```

Controlled By: Deployment/nginx-hostnetwork-deployment  
Replicas: 0 current / 1 desired  
Pods Status: 0 Running / 0 Waiting / 0 Succeeded / 0 Failed  
Pod Template:  
 Labels: app=nginx  
 pod-template-hash=597c4cff45  
 Containers:  
 nginx:  
 Image: nginx:1.15.4  
 Port: <none>  
 Host Port: <none>  
 Environment: <none>  
 Mounts: <none>  
 Volumes: <none>  
 Conditions:  
 Type Status Reason  
 ----  
 ReplicaFailure True FailedCreate  
Events:  
 Type Reason Age From Message

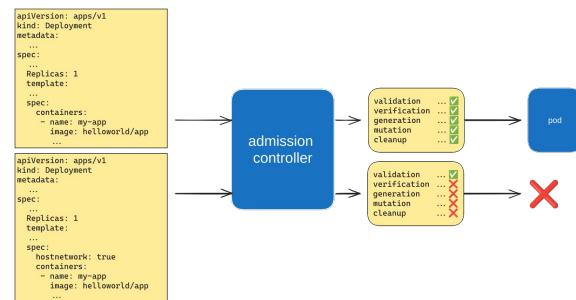
Warning FailedCreate 13s (x13 over 33s) replicaset-controller Error creating: pods "nginx-hostnetwork-deployment-597c4cff45-" is forbidden: unable to validate against any pod security policy: [spec.securityContext.hostNetwork: Invalid value: true: Host network is not allowed to be used]

ALEVSK

- In this example, If a malicious user try to run a container using the **hostNetwork: true** to share the host's network namespace, the restrictive PSP in place will prevent the pod from running
- **hostNetwork: true** provides a pod with greater network access at the cost of increasing security risk

# Pod Security Admission & Pod Security Standards

- Less flexible than PSP, but easier to use and implement
- Privileged, Baseline & Restricted
- Validation  
(Validating Admission Webhook)
- Mutation  
(Mutating Admission Webhook),
- Third-party: OPA GateKeeper, Kyverno, jsPolicy, etc



ALEVSK

A modern approach for access control would be to use Pod Security Admission

- Pod Security Admission is a build-in feature in Kubernetes that helps ensure your Pods meet certain Pod Security Standards (Privileged, Baseline & Restricted) before they're allowed to run. Think of it as a security checkpoint that a Pod has to pass through
- Imagine a gatekeeper for your Kubernetes cluster. Before any request (like creating a Pod, Service, etc.) is fully processed, it has to go through various checkpoints. Admission Controllers are these checkpoints, each with its own set of rules on what's allowed and what's not. They can modify requests or reject them if they don't meet certain criteria, helping to enforce security, governance, and operational policies in your cluster.

<https://kubernetes.io/docs/concepts/security/pod-security-admission/>  
<https://kubernetes.io/docs/concepts/security/pod-security-standards/>

# Audit Log

- When to Log:
  - RequestReceived, ResponseStarted, ResponseComplete, Panic
- What to Log:
  - None, Metadata, Request, RequestResponse
- Where to Log:
  - Filesystem, Webhook

```
15 apiVersion: audit.k8s.io/v1 # This is required.
14 kind: Policy
13 # Don't generate audit events for all requests in RequestR
12 omitStages:
11   "RequestReceived"
10 rules:
9   # Log pod changes at Request level
8   - level: Request
7   resources:
6   - group: ""
5   resources: ["pods"]
4   # Log pod changes at RequestResponse level
3   - level: RequestResponse
2   resources:
1   - group: ""
0   resources: ["pods"]
~
~
~
~
```

NORMAL audit-policy.yaml unix | utf-8 | yaml 100% | 16:2

ALEVSK

Next, Kubernetes has an audit logging feature that, when enabled, keeps a detailed record of all the actions that happen inside your cluster, such as who accessed it, what changes they made, and when these activities occurred. It's very customizable and it allows you to send these logs to other tools for further analysis, however there are a couple things to consider:

- It increases the memory footprint of the API server because additional resources are required when processing each new request.
- Audit Logs do not directly capture application-level events such as incoming HTTP requests
- These application-specific logs are managed separately and typically involve a different logging mechanism or setup.
- Tools like Fluentd, Elastic Stack (ELK), or Prometheus with Grafana are popular choices in the Kubernetes ecosystem for this purpose.

<https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/>

# Kubernetes Cluster Hardening

- Isolate Kubernetes Nodes
  - Don't expose nodes directly, use ingress controller, separate control and data flow
- Protect Kubelet
  - Disable anonymous authentication
  - Enable mTLS authentication on the kubelet's HTTPS endpoint
- Protect etcd with TLS, Firewall and Encryption
- Use Third-Party Authentication for API Server



ALEVSK

Finally, there's different methods, best practices and industry standards to follow when securing kubernetes, you can start by:

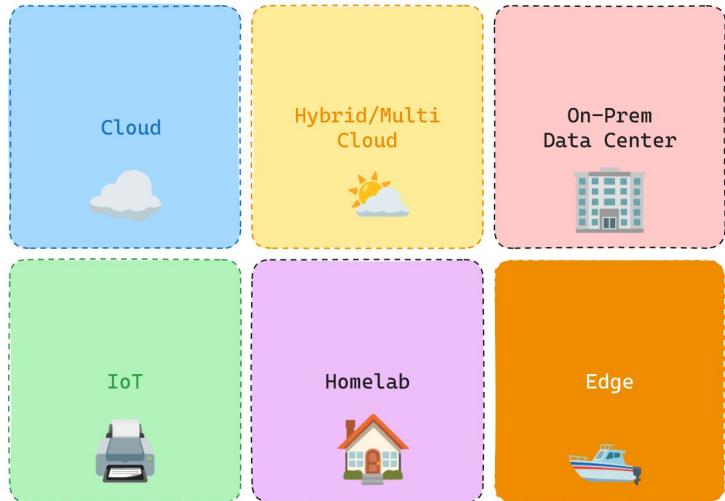
- Isolate kubernetes Nodes
  - Don't expose node directly to the Internet and instead use an ingress controller
  - If your environment supports it, configure separate network interfaces on your nodes for different types of traffic. For example:
    - One network interface handles public internet traffic (ingress traffic to ports 80 and 443).
    - Another interface handles internal cluster communications, including kube-api server traffic and node management traffic.
- Protect Kubelet
  - Disable anonymous authentication
  - Enable mTLS authentication on the kubelet's HTTPS endpoint
- Use Third-Party Authentication for API Server:
  - OIDC or implement a custom proxy for authentication

# What's Next?

ALEVSK

## Kubernetes Evolution (Edge & Hybrid Cloud)

- Kubeedge
- SuperEdge
- Akri
- OpenYurt
- K3s
- Microk8s
- Minikube
- ...



ALEVSK

- We are still facing many security challenges when it comes to defending Kubernetes, and it's becoming increasingly popular to deploy Kubernetes at the "edge" for use cases like hybrid environments, IoT and Edge computing.
- Tools like kubeedge, SuperEdge, K3s, MicroK8s and Minikube have a low CPU and memory footprint and make it very easy to deploy Kubernetes on low energy consumption devices, but at the cost of increasing our attack surface

## Next Time You Deploy Kubernetes – Part 1

1. Don't deploy images from untrusted registries in your cluster, regularly scan images for vulnerabilities and enable binary authorization
2. Don't store sensitive data on kubernetes secrets, instead use an [external KMS](#) to manage secrets or enable [EncryptionConfiguration](#)
3. Encryption in transit: generate TLS certificates for your workloads using the [certificates.k8s.io](#) api or use a third party solution like [cert-manager.io](#)

ALEVSK

## Next Time You Deploy Kubernetes – Part 2

4. Enforce **SecurityContext** and limit resource consumption for your workloads
5. Network Segmentation, choose **CNI plugin** that supports **network policies** to limit communication between your **services** and monitor network traffic
6. Configure Role-Based Access Control (**RBAC**) policies via an **Admission Controller** or **PSPs** always following the principle of least privilege. Never use **cluster-admin** in your workloads

ALEVSK

## Next Time You Deploy Kubernetes – Part 3

7. Turn on audit logging and monitor unusual or unwanted api calls, e.g. authentication failures.
8. Keep Kubernetes version up to date, update and patch regularly
9. Hardening control plane components and lock down kubelet
10. Adopt GitOps practices for managing your Kubernetes clusters to enhance security, auditability, and consistency through declarative configuration, automation, and tight integration with version control systems

ALEVSK

# Thanks



- 
- twitter [@alevsk](https://twitter.com/alevsk)
  - github <https://github.com/alevsk>
  - linkedin <https://www.linkedin.com/in/alevsk/>
  - website <https://www.alevsk.com>
-