

Certified Kubernetes Security Specialist (CKS)

Setting the Base

Kubernetes is one of the most popular container orchestration tools in the industry.

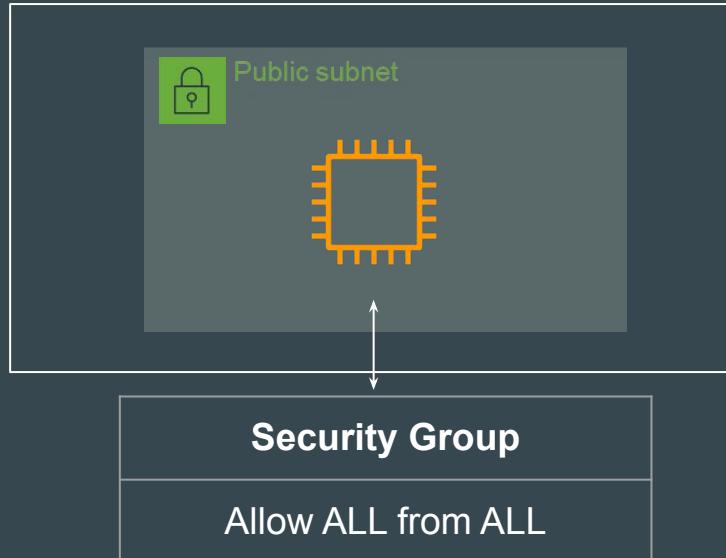
It is used extensively in many medium to large-scale organizations.



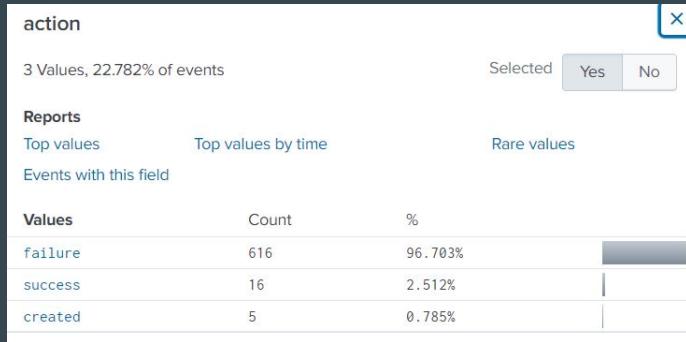
Major Challenge - Security is Neglected

Security is often neglected due to which there are so many breaches.

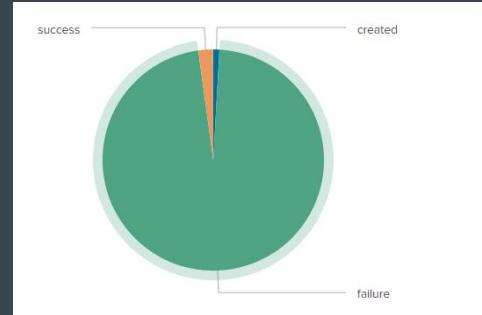
A simple EC2 with open security group can teach you many things.



Results



616 failed attempts



96% failure rate



Bruteforce from
across the world.

Kubernetes Security is Ignored

Most organizations deploying Kubernetes cluster ignore the security aspect of it.

There is big demand of individuals who know has expertise in K8s security.



Introducing CKS

The Certified Kubernetes Security Specialist (CKS) certification demonstrates competence on best practices for **securing Kubernetes platforms**.



What this Course is All About?

This is a certification-specific course aimed at individuals who intend to gain the **Certified Kubernetes Security Specialist** certification.

Thorough
Preparation



Certification is Beneficial

We will be covering all the domains for the Certified Kubernetes Security Specialist certification.

1. Cluster Setup
2. Cluster Hardening
3. System Hardening
4. Minimize Microservice Vulnerabilities
5. Supply Chain Security
6. Monitoring, Logging and Runtime Security

The Exciting Part

We have an **exam preparation section** to help you get prepared for the exam.

We also have practice tests available as part of the course.



Prerequisite for CKS

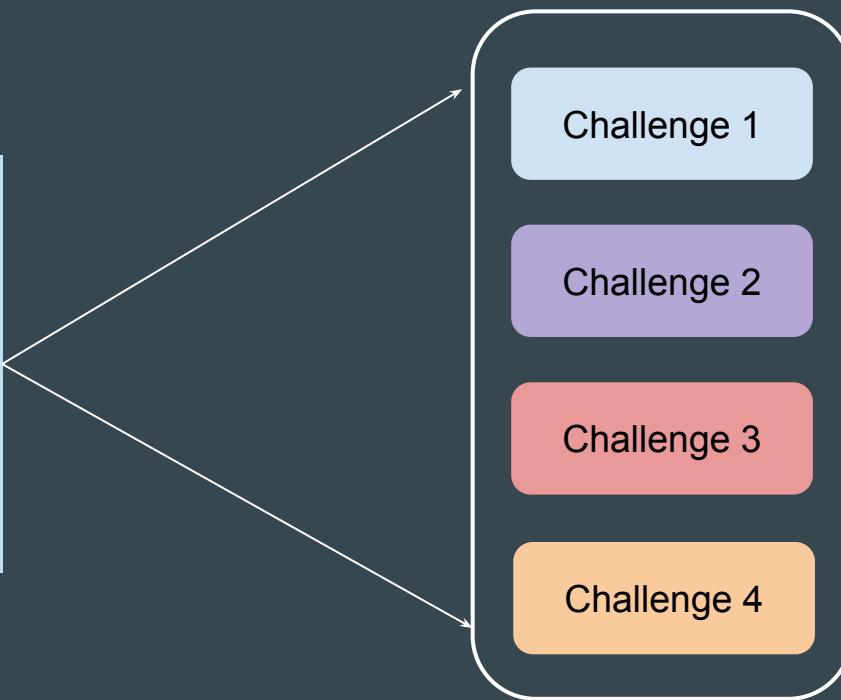
Having a CKA certification is a prerequisite for CKS certification.

CKA need not be active, even if it is expired, it will be acceptable.



Lab Based Exams

CKS exam is a **lab-based exam** where you will have to solve multiple scenarios presented to you.



About Me

- DevSecOps Engineer - Defensive Security.
- Teaching is one of my passions.
- I have total of 16 courses, and around 400,000+ students now.

Something about me :-



- Certified Kubernetes [Security Specialist, Administrator, Application Developer]
- HashiCorp Certified [Terraform Professional [Vault and Consul Associate]
- AWS Certified [Advanced Networking, Security Specialty, DevOps Pro, SA Pro, ...]
- RedHat Certified Architect (RHCA) + 13 more Certifications
- Part time Security Consultant

About the Course and Resources

1 - Aim of This Course

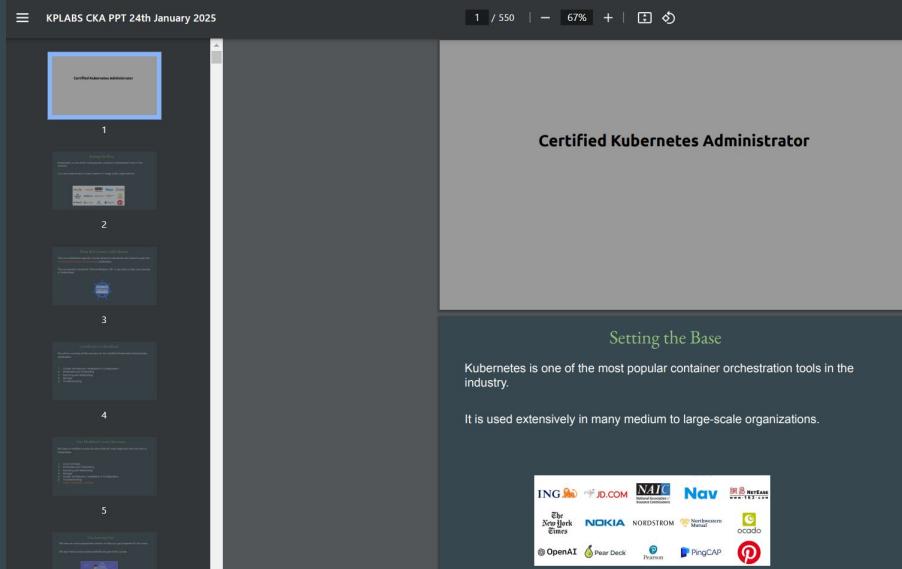
The **primary aim** of this course is to learn.



2 - PPT Slides PDF

ALL the slides that we use in this course is available to download as PDF.

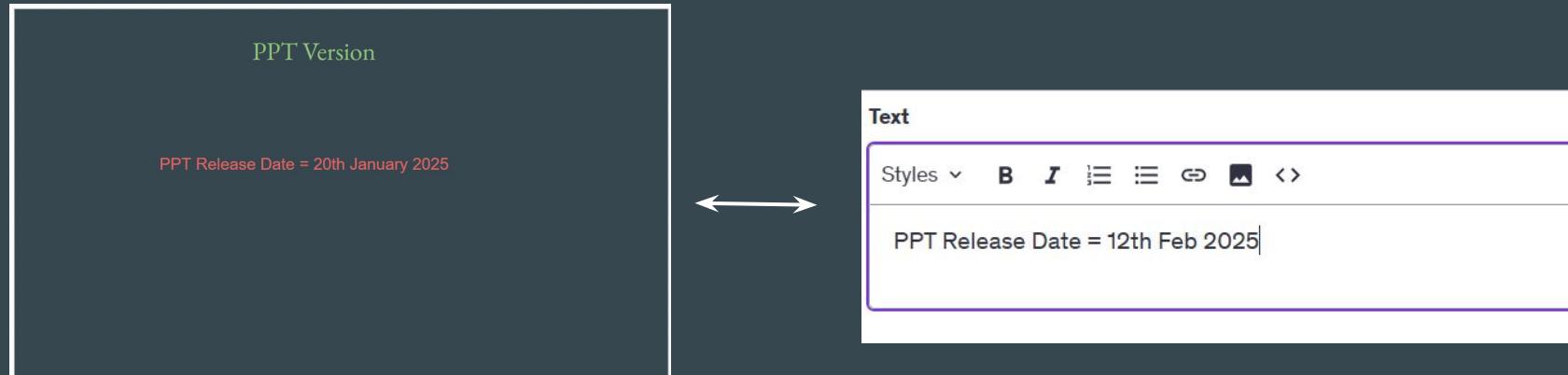
We have around 360+ slides that are available to download.



3 - PPT Version

The course is updated regularly and so are the PPTs.

We add the PPT release date in the PPT itself and the lecture from which you download the PPTs.

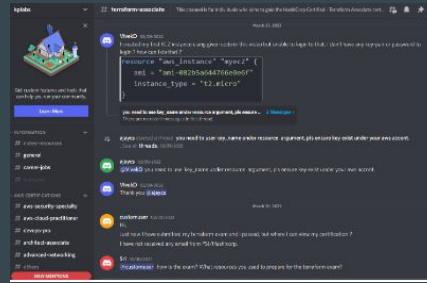


4 - Our Community (Optional)

You can **join our Discord community** for any queries / discussions. You can also connect with other students going through the same course in Discord (Optional)

Discord Link: <https://kplabs.in/chat>

Category: #cks



5 - Course Resource - GitHub

All the code that we use during practicals have been added to our GitHub page.

Section Name in the Course and GitHub are same for easy finding of code.



6 - Code Editor

Visual Studio Code is the default code editor used for this course.

```
! port-proto.yaml ●  
!  
apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: multi-port-egress  
  namespace: default  
spec:  
  podSelector:  
    matchLabels:  
      role: db  
  policyTypes:  
    - Egress  
  egress:  
    - to:  
      - ipBlock:  
          cidr: 10.0.0.0/24
```

Basics of CIS Benchmarks

Understanding with Analogy

Think about how we secure our homes - we lock doors, install security systems, and follow certain safety practices.



Setting the Base

In the digital world, organizations face a similar challenge but on a much larger scale. They have many computers, servers, and systems that need protection.



Understanding the Challenge

Different organizations were following different security practices, some good and some not so effective.

This inconsistency can create vulnerabilities.

Security Standard
Installed Anti-Virus
VPN Enabled

Organization 1

Security Standard
Install Anti-Virus
VPN Enabled
Server Hardening

Organization 2

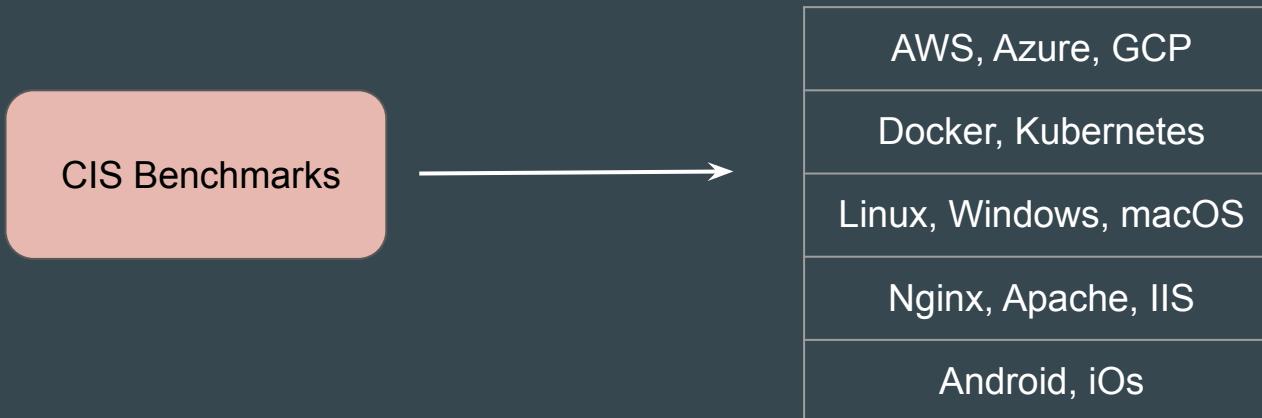
Security Standard
Install Anti-Virus
VPN Enabled
Server Hardening
Regular VA/PT
SIEM + 24/7 SOC

Organization 3

Introducing CIS Benchmarks

CIS Benchmarks are best-practice security guidelines developed by the Center for Internet Security (CIS).

They provide step-by-step instructions on how to secure systems



Tools that Assesses CIS Benchmarks

There are many tools available in the industry that can check whether an organization has implemented CIS Benchmark correctly.

The screenshot shows the AWS Foundations Benchmark tool interface. On the left, a sidebar menu includes options like SECURE, Overview, Image Scanning, Compliance, Policies, Events, Activity Audit, and Captures. Below these are Get Started and API links. The main content area displays the 'Tasks > AWS Foundations Benchmark' page. It shows account details (Account Id: 845151661675, Region: ca-central-1, Evaluation Date: March 16, 2021 7:55 PM), a 'Download CSV' button, and a summary bar indicating 89% of resources pass, 1992 resources passing, and 223 resources failing out of 2215 total resources. A detailed list of 20 tasks is provided, categorized by severity (Level 1 or Level 2). Task 5.2 is expanded to show its details: 'Ensure no security groups allow ingress from 0.0.0.0/0 to remote server administration ports'. It includes a note about security groups, how to address it, affected resources (GroupID: sg-07659df628f5d5070, GroupName: CloudVision-CloudBenchStack-1GV6XEDNJR-CloudBench), and a remediation procedure. The remediation procedure lists steps to perform:

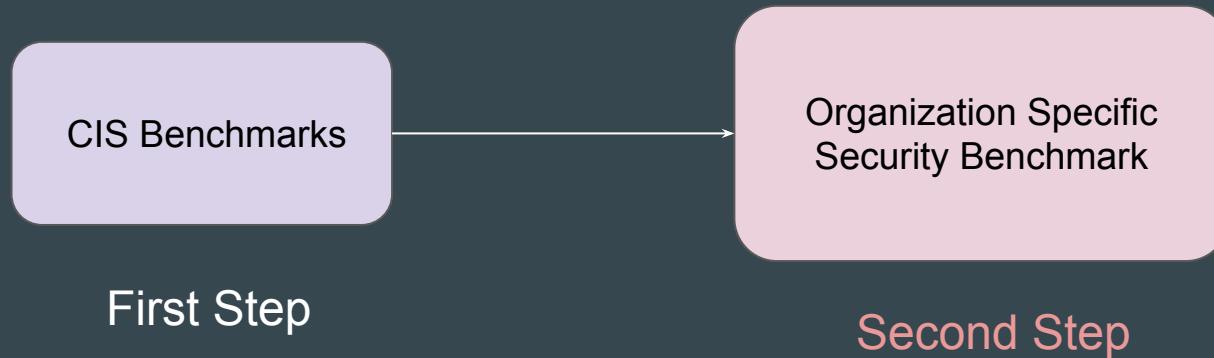
1. Login to the AWS Management Console at <https://console.aws.amazon.com/vpc/home>
2. In the left pane, click Security Groups
3. For each security group, perform the following:
 4. Select the security group
 5. Click the Inbound Rules tab
 6. Click the Edit inbound rules button
 7. Identify the rules to be edited or removed
 8. Either A) update the Source field to a range other than 0.0.0.0/0, or, B) Click Delete to remove the offending inbound rule
 9. Click Save rules

At the bottom, another task is partially visible: 'Ensure the default security group of every VPC restricts all traffic'.

Point to Note - Part 1

While valuable, implementing CIS Benchmarks, it might not cover 100% of what needs to be done.

CIS Benchmarks evolve with new technologies and new threats.



Point to Note - Part 2

Some recommendations might conflict with business needs and organization-specific security standards.

Example:

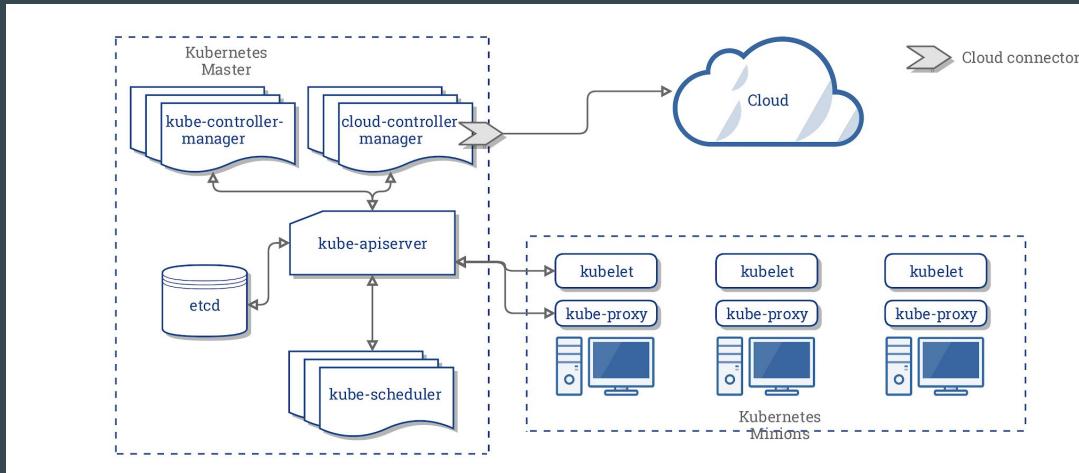
CIS Benchmark talks about password complexity for local Linux passwords.

Organization might use AD / IPA instead of local passwords

CIS Benchmarks for Kubernetes

Setting the Base

CIS Benchmarks give a specific set of guidelines suitable to Kubernetes and ensure the hardened Kubernetes environments.



Understanding the Structure

CIS Benchmark covers both the Control Plane Node + Worker Node components.

CIS Benchmarks

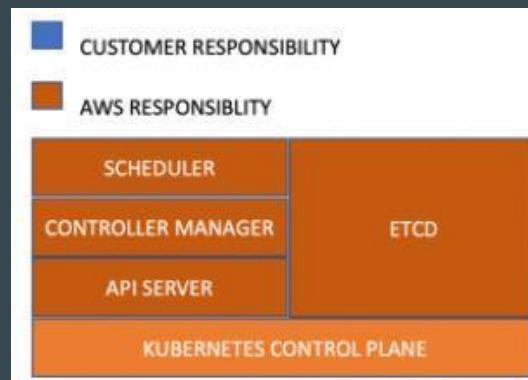
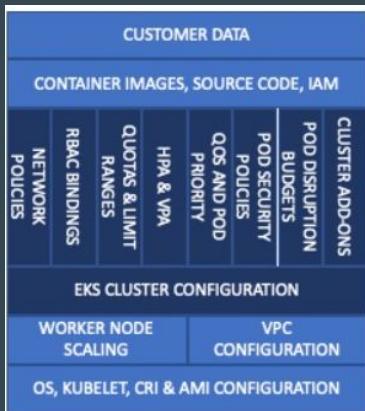


Areas Covered	Sub-Components
Control Plane Node	API, Controller Manager, Scheduler, ETCD
Worker Node	Kubelet, Kube-Proxy
Pod Security Standards	
CNI, Secrets, General	

Shared Responsibility Model

Lot of providers offer managed Kubernetes offerings.

Since customers do not have full control over the cluster, the security responsibility is divided.



Point to Note

If you are using a managed Kubernetes cluster, there are separate set of CIS Benchmark for Kubernetes available that takes into account the Customer responsibilities.

Kubernetes	Virtualization
CIS Alibaba Cloud Container Service For Kubernetes (ACK) Benchmark v1.0.0	
CIS Amazon Elastic Kubernetes Service (EKS) Benchmark v1.6.0	
CIS Azure Kubernetes Service (AKS) Benchmark v1.5.0	
CIS Google Kubernetes Engine (GKE) Autopilot Benchmark v1.0.0	
CIS Google Kubernetes Engine (GKE) Benchmark v1.6.1	
CIS Kubernetes Benchmark v1.9.0	

Tools to Analyze Kubernetes Benchmark

There are many tools like `kube-bench` that checks whether Kubernetes is deployed securely by running the checks documented in the CIS Kubernetes Benchmark.

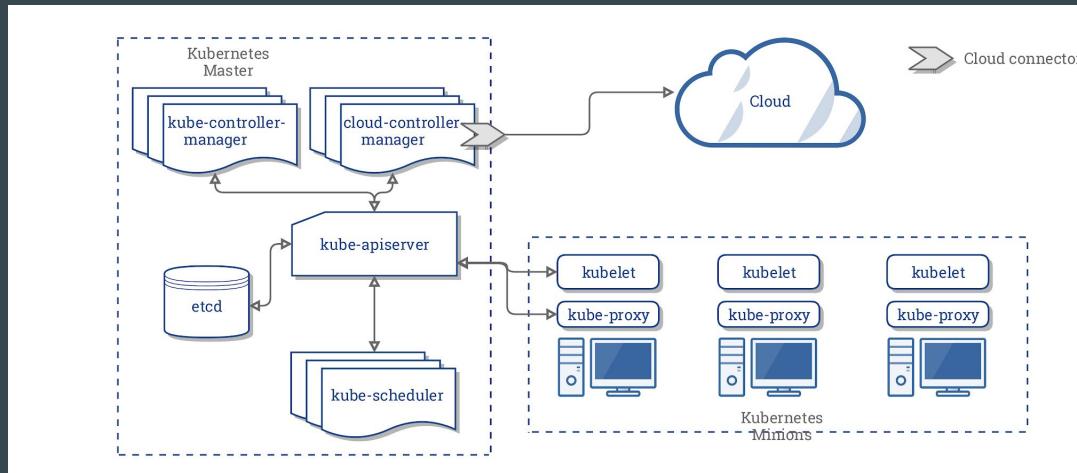
```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
[FAIL] 1.1.9 Ensure that the --profiling argument is set to false (Scored)
[FAIL] 1.1.10 Ensure that the --repair-malformed-updates argument is set to false (Scored)
[PASS] 1.1.11 Ensure that the admission control policy is not set to AlwaysAdmit (Scored)
[FAIL] 1.1.12 Ensure that the admission control policy is set to AlwaysPullImages (Scored)
[FAIL] 1.1.13 Ensure that the admission control policy is set to DenyEscalatingExec (Scored)
[FAIL] 1.1.14 Ensure that the admission control policy is set to SecurityContextDeny (Scored)
[PASS] 1.1.15 Ensure that the admission control policy is set to NamespaceLifecycle (Scored)
[FAIL] 1.1.16 Ensure that the --audit-log-path argument is set as appropriate (Scored)
[FAIL] 1.1.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate (Scored)
[FAIL] 1.1.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as appropriate (Scored)
[FAIL] 1.1.19 Ensure that the --audit-log-maxsize argument is set to 100 or as appropriate (Scored)
[PASS] 1.1.20 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 1.1.21 Ensure that the --token-auth-file parameter is not set (Scored)
[FAIL] 1.1.22 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored)
```

Our Lab Setup

Setting the Base

Throughout this course, we will be learning about security aspects of Kubernetes components in detail.

For this, we need a test environment to practice.



Lab Setup

For our labs, we will making use of following setup:

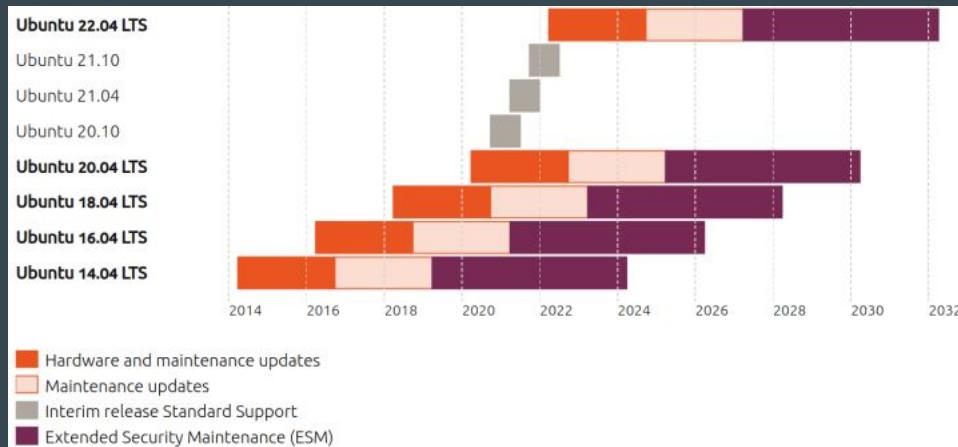
Operating System	Cloud Provider
Ubuntu 24 LTS	Digital Ocean

You can decide to use cloud provider of your choice.

Point to Note - OS

It is recommended to use the same OS version that we use in practicals.

We have intentionally used Ubuntu LTS (Long Term Support)



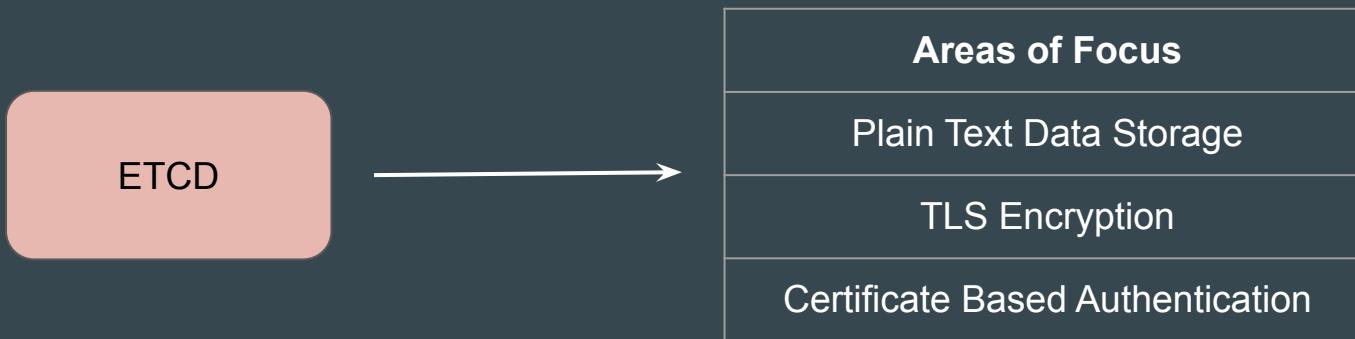
Why Digital Ocean?

1. Reasonably priced servers and pay per hour.
2. Good Amount of Credits for New Users (Referral) - \$100-\$200
3. Keep it simple approach.

Important Etcd Security Guidelines

Area of Focus

To secure etcd, we'll focus on three key areas.



1 - Plain Text Data Storage

By default, etcd stores data in plain text.

This means sensitive data (like Secrets) can be read directly from the disk.

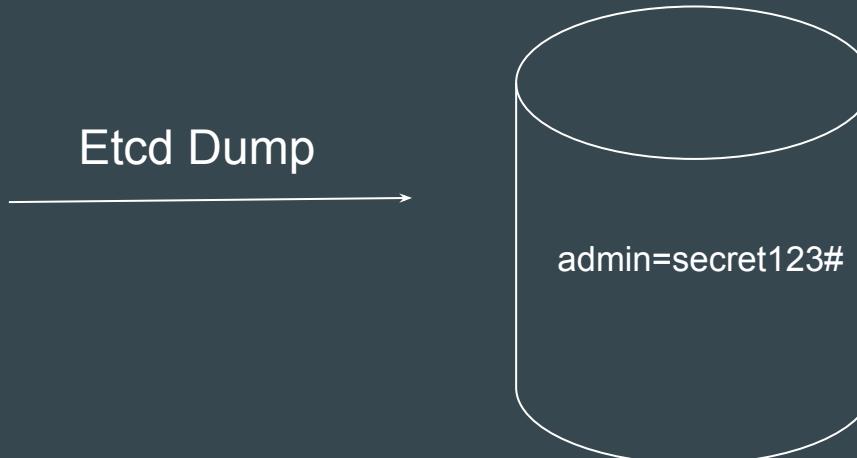


Hacker is Happy

If an attacker gets access to etcd, they can read secrets in plain text.



Attacker



2 - TLS Encryption

The data in-transit between API server and ETCD can also be intercepted.

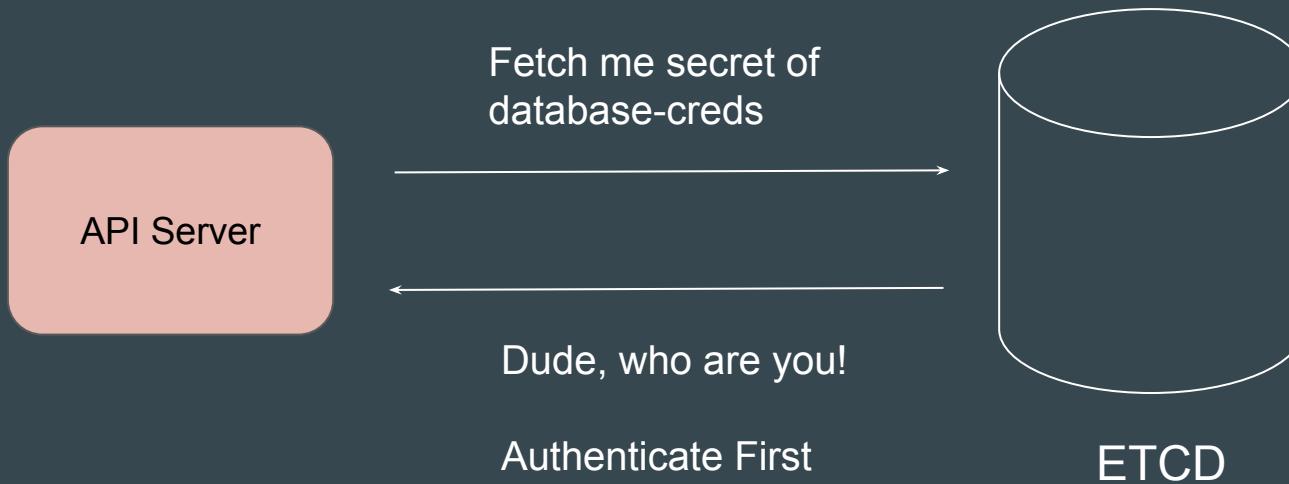
You have to ensure that this data is also protected.



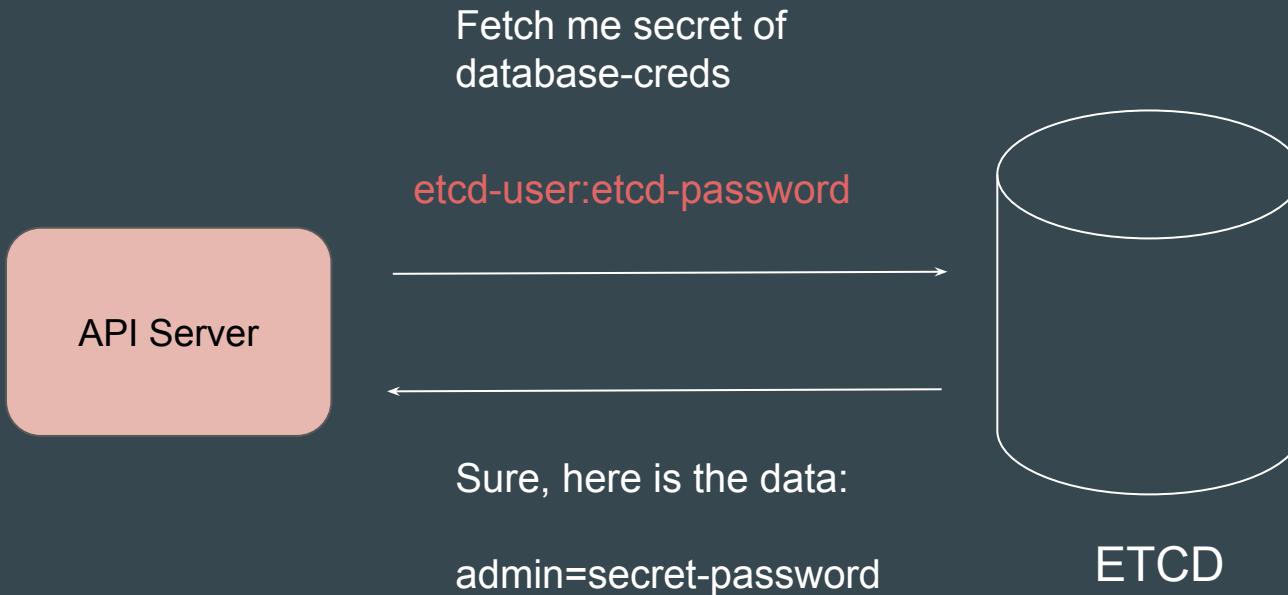
3 - Authentication

Without authentication, any client can connect to etcd and modify data.

It is important to have authentication in place for etcd.



Basic Authentication Workflow



Types of Authentication

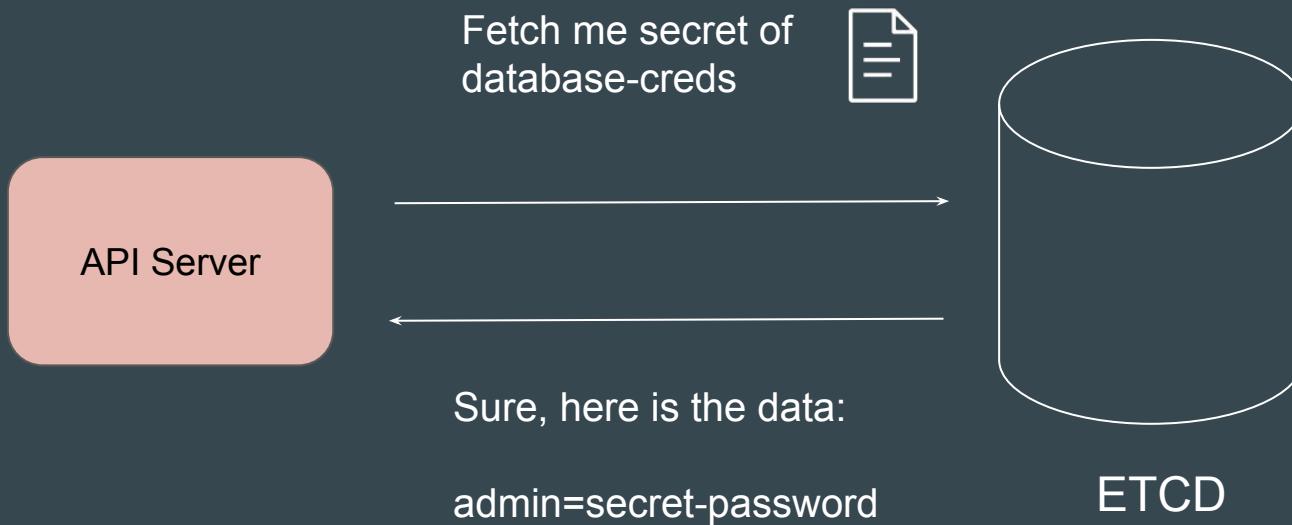
Etcd supports various approaches to authenticate.

We will take example of username/passwords and Certificates.

Feature	Username / Passwords	Certificates
Security Level	Lower	Higher (certificates are harder to forge)
Ease of Use	Easier to configure and manage	More complex to set up and maintain
Best Use Case	Simple setups, development environments	Production environments, high-security applications

4 - Certificate Based Authentication

In this approach, **certificates** are used to verify the identity.

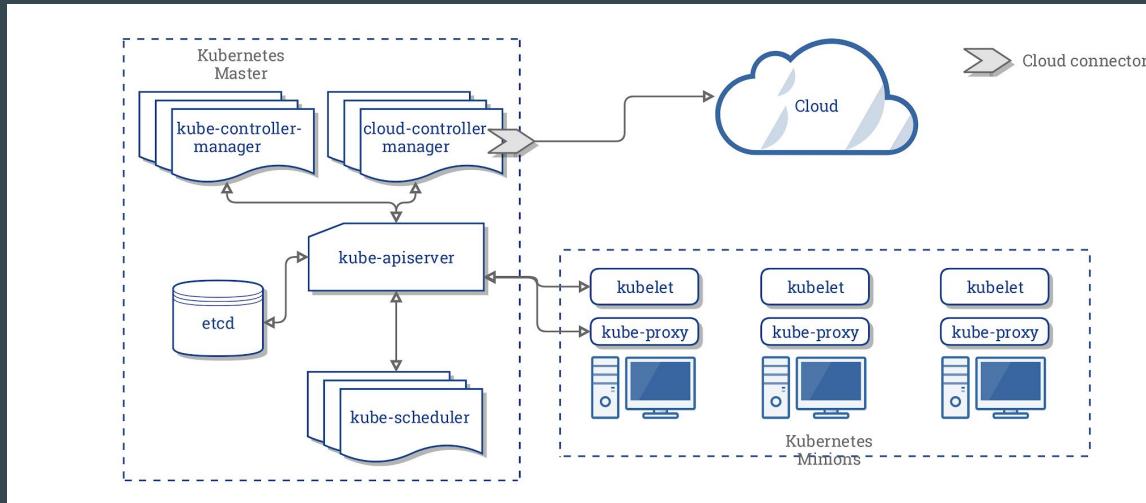


Certificate Authority

Setting the Base

Kubernetes components, such as the API server, kubelet, controller manager should communicate with each other over **secure channel**.

These components need a mechanism to **verify each other's identity**.



Certificate Authority

Certificate Authority is an entity which issues digital certificates.

Key part is that both the receiver and the sender trusts the CA.



Two Important Use-Cases

There are two important use-case where CA certificate will be used.

1. To generate TLS certificates for secure communication.

2. To generating certificates for client / component for authentication.

Workflow - Issuance of Signed Certificates

Certificate Authority

We are going to discuss workflow on how Certificate Authority issues signed certificates.



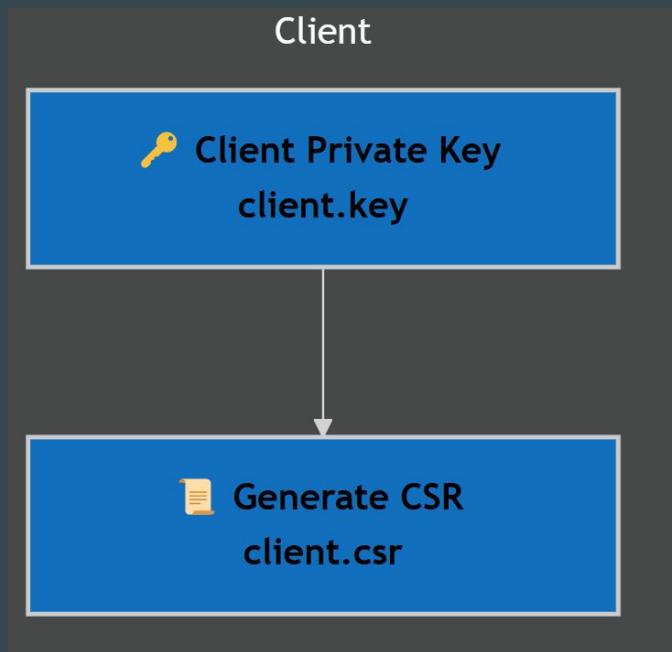
Understanding the Workflow

For the entire workflow to work, there are three steps:

1. Generate the CA Certificate and Key
2. Generate the Certificate Signing Request for Components and Clients.
3. Sign the CSR using CA creds to get the final certificate.

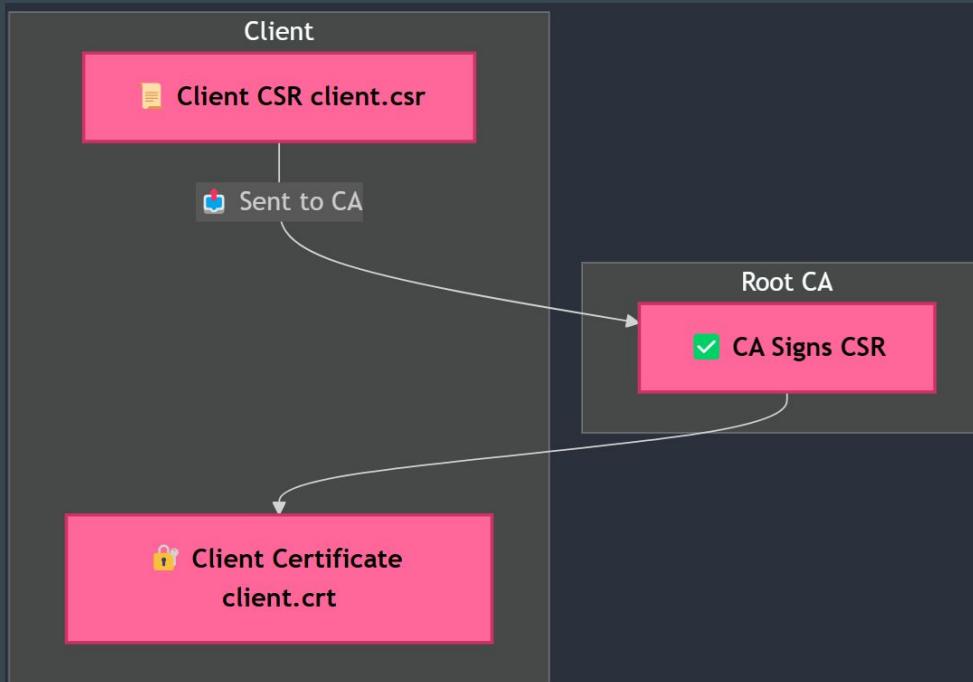
Part 1 - Generate Certificate Signing Request

In this step, we generate client key and certificate signing request (CSR).



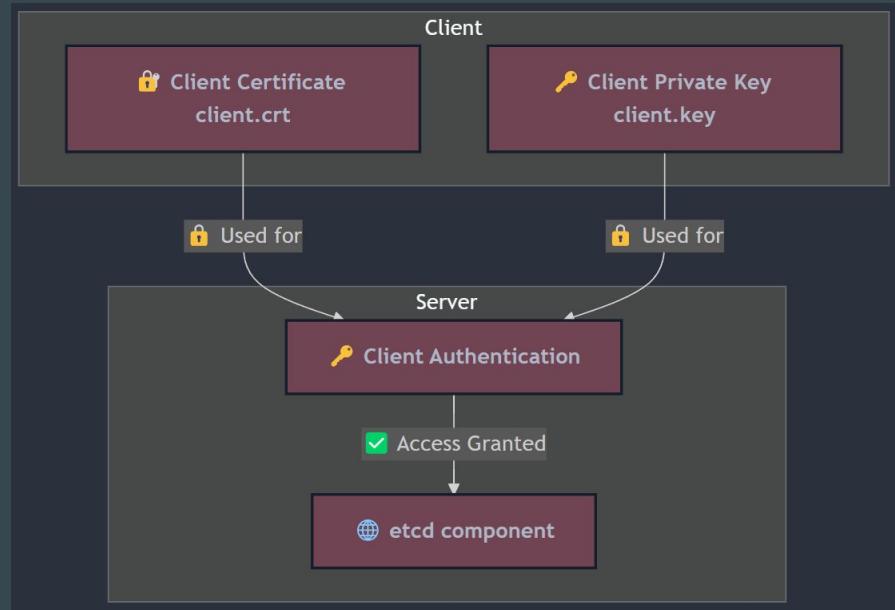
Part 2 - Sign CSR using CA Creds

In this step, we send CSR to the certificate authority. CA will verify and sign it to provide final client certificate.



Part 3 - Authenticate

The signed certificates can be used for client authentication as well as for secure communication over TLS.



Etcd - Transport Security with HTTPS

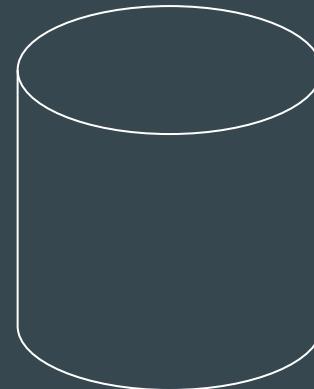
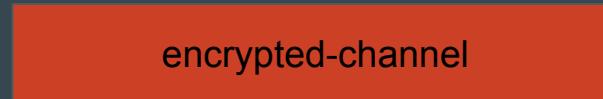
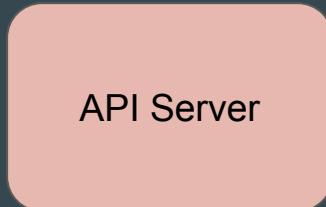
Understanding the Challenge

If the traffic between the API server and etcd is not encrypted, an attacker sniffing the network can easily fetch all of the data in plain text.



Setting the Base

You can configure etcd to listen on HTTPS so that the communication between a client and etcd is encrypted and secured.



API Server

encrypted-channel

ETCD

Step 1 - Generate Certificate for etcd

We have to generate a certificate for the etcd component.

This certificate will be used for HTTPS communication.

Step 2 - Start Etcd using HTTPS

Configure etcd to start listening on HTTPS and use the certificates.

```
root@demo:~/certificates# etcd --cert-file=/root/certificates/etcd.crt --key-file=/root/certificates/etcd.key --advertise-client-urls=https://127.0.0.1:2379 --listen-client-urls=https://127.0.0.1:2379
{"level":"warn","ts":"2025-02-07T18:10:30.155469Z","caller":"embed/config.go:689","msg":"Running http and grpc server on single port. This is not recommended for production."}
{"level":"info","ts":"2025-02-07T18:10:30.155753Z","caller":"etcdmain/etcd.go:73","msg":"Running: ","args":["etcd","--cert-file=/root/certificates/etcd.crt","--key-file=/root/certificates/etcd.key","--advertise-client-urls=https://127.0.0.1:2379","--listen-client-urls=https://127.0.0.1:2379"]}
 {"level":"warn","ts":"2025-02-07T18:10:30.155905Z","caller":"etcdmain/etcd.go:105","msg":"'data-dir' was empty; using default","data-dir":"default.etcd"}
 {"level":"info","ts":"2025-02-07T18:10:30.156098Z","caller":"etcdmain/etcd.go:116","msg":"server has been already initialized","data-dir":"default.etcd","dir-type":"member"}
 {"level":"warn","ts":"2025-02-07T18:10:30.156214Z","caller":"embed/config.go:689","msg":"Running http and grpc server on single port. This is not recommended for production."}
 {"level":"info","ts":"2025-02-07T18:10:30.156288Z","caller":"embed/etcd.go:140","msg":"configuring peer listeners","listen-peer-urls":["http://localhost:2380"]}
 {"level":"info","ts":"2025-02-07T18:10:30.157211Z","caller":"embed/etcd.go:148","msg":"configuring client listeners","listen-client-urls":["https://127.0.0.1:2379"]}
```

Important Flags

Flags	Description
--cert-file=<path>	Specifies the path to the server's SSL/TLS certificate file.
--key-file=<path>	Specifies the path to the private key file corresponding to the certificate.
--advertise-client-urls	Specifies the URLs that etcd should advertise to clients for client communication. Example: https://<IP>:2379
--listen-client-urls	Specifies the URLs on which etcd listens for client requests. Example: https://0.0.0.0:2379

Mutual TLS Authentication

Case Study of 3 Cats

My wife takes care of 1 street cat, and she asked me to feed the cat (brownish) while she is out of the city for the next 1 month.

I started to feed a cat regularly, but later realised it was an impersonator cat.



Original Cat

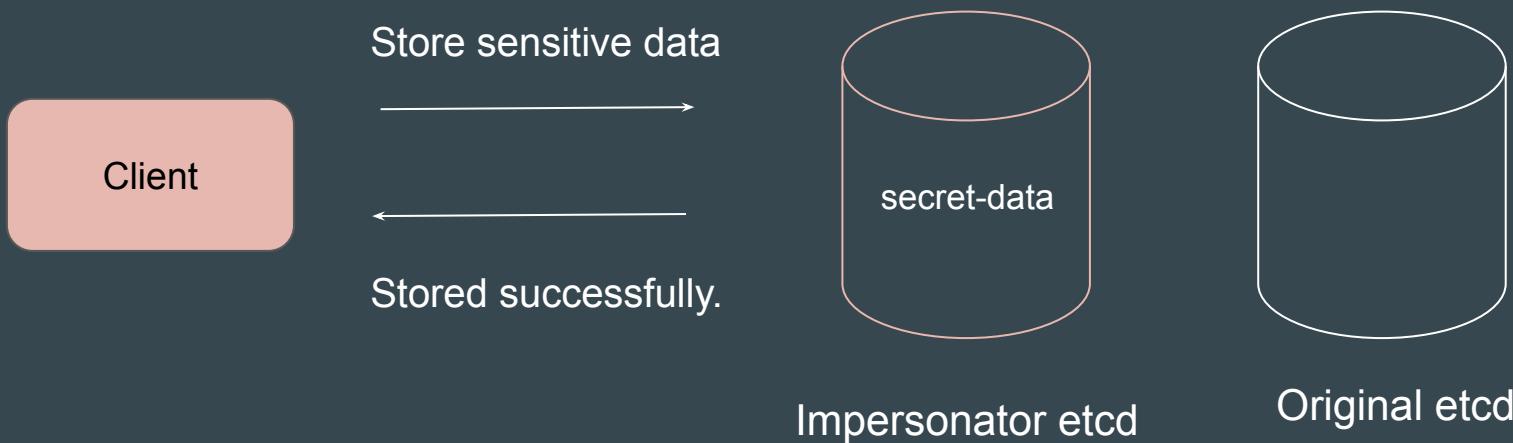


Impersonators

Setting the Base

Client wants to store super sensitive data in etcd.

An attacker can launch his own etcd that act as an impersonator to store sensitive data.



Solution - Mutual TLS

Mutual TLS (mTLS) is a security protocol that **ensures both the client and the server verify each other's identities** before establishing a connection.

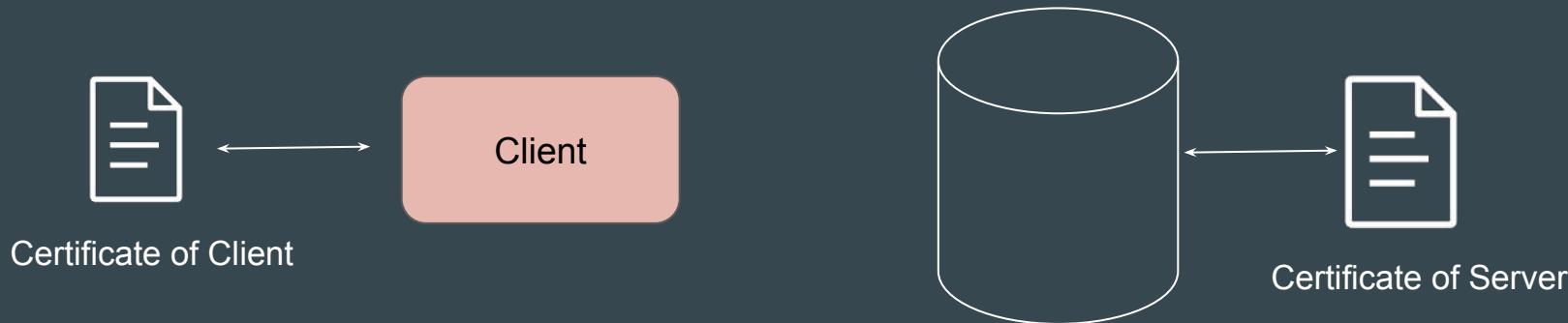
Both the client and the server have digital certificates issued by a trusted Certificate Authority (CA)



Requirement for mTLS to Work

Both the sender and receiver should have their certificates.

These certificates must be signed by trusted Certificate Authority that both sender and receiver trusts.



Handshake Process

1. The client connects to the server.
2. The server presents its certificate to the client.
3. The client verifies the server's certificate.
4. The client presents its certificate to the server.
5. The server verifies the client's certificate.
6. If both verifications pass, a secure, encrypted connection is established.

Mutual TLS Authentication - Practical

Workflow Steps

1. Certificate Authority.
2. Etcd certificate signed through the Certificate Authority.
3. Client certificate signed through the Certificate Authority.
4. Both etcd and client will trust the Certificate Authority (1)

Important Flags - etcd server

Flags	Description	Importance
--cert-file and --key-file	To start etcd with HTTPS	Secure communication
--client-cert-auth	Enables client certificate authentication.	Requires clients to provide a valid certificate signed by the trusted CA.
--trusted-ca-file	Path to Certificate Authority certificate file.	Etcd uses this file to verify the authenticity of client certificates.
--advertise-client-urls	Specifies the URLs that etcd should advertise to clients for client communication. Example: <code>https://<IP>:2379</code>	
--listen-client-urls	Specifies the URLs on which etcd listens for client requests. Example: <code>https://0.0.0.0:2379</code>	

Important Flags - etcdctl (client)

Flags	Description	Importance
--cacert	Path to Certificate Authority certificate file.	etcdctl uses this to verify the server's certificate. This ensures you're connecting to the correct etcd server and not a malicious one.
--cert-file and --key-file	To authenticate with etcd using client certificates.	

Manage etcd using Systemd

Understanding the Challenge

At this stage, we have been starting etcd manually from CLI

This is a good approach during test, but not ideal for Production.

```
root@k8s:~/certificates# etcd --cert-file=etcd.crt --key-file=etcd.key --advertise-client-urls=https://127.0.0.1:2379 --client-cert-auth --trusted-ca-file=ca.crt --listen-client-urls=https://127.0.0.1:2379
{"level":"warn","ts":"2025-02-08T16:48:41.953249Z","caller":"embed/config.go:689","msg":"Running http and grpc server on single port. This is not recommended for production."}
{"level":"info","ts":"2025-02-08T16:48:41.953545Z","caller":"etcdmain/etcd.go:73","msg":"Running: ","args":["etcd","--cert-file=etcd.crt","--key-file=etcd.key","--advertise-client-urls=https://127.0.0.1:2379","--client-cert-auth","--trusted-ca-file=ca.crt","--listen-client-urls=https://127.0.0.1:2379"]}
 {"level":"warn","ts":"2025-02-08T16:48:41.953703Z","caller":"etcdmain/etcd.go:105","msg":"'data-dir' was empty; using default ","data-dir":"default.etcd"}
 {"level":"warn","ts":"2025-02-08T16:48:41.953864Z","caller":"embed/config.go:689","msg":"Running http and grpc server on single port. This is not recommended for production."}
 {"level":"info","ts":"2025-02-08T16:48:41.953897Z","caller":"embed/etcd.go:140","msg":"configuring peer listeners","listen-peer-urls":["http://localhost:2380"]}
 {"level":"info","ts":"2025-02-08T16:48:41.954840Z","caller":"embed/etcd.go:148","msg":"configuring client listeners","listen-client-urls":["https://127.0.0.1:2379"]}
 {"level": "info", "ts": "2025-02-08T16:48:41.955030Z", "caller": "embed/etcd.go:323", "msg": "starting an etcd server", "etcd-version": "3.5.18", "git-sha": "5bca08e", "go-version": "go1.22.11", "go-os": "linux", "go-arch": "amd64", "max-cpu-set": 1, "max-cpu-available": 1, "member-initialized": false, "name": "default", "data-dir": "default.etcd", "wal-dir": "", "wal-dir-dedicated": "", "member-dir": "default.etcd/member", "force-new-cluster": false, "heartbeat-interval": "100ms", "election-timeout": "1s", "initial-election-tick-advantage": true, "snapshot-count": 10000, "max-wals": 5, "max-snapshots": 5, "snapshot-catchup-entries": 5000, "initial-advertise-peer-urls": ["http://localhost:2380"], "listen-peer-urls": ["http://localhost:2380"], "advertise-client-urls": ["https://127.0.0.1:2379"], "listen-client-urls": ["https://127.0.0.1:2379"], "listen-metrics-urls": [], "cors": ["*"], "host-whitelist": ["*"], "initial-cluster": "default=http://localhost:2380", "initial-cluster-state": "new", "initial-cluster-token": "etcd-cluster", "quota-backend-bytes": 2147483648, "max-request-bytes": 1572864, "max-concurrent-streams": 4294967295, "pre-vote": true, "initial-corrupt-check": false, "corrupt-check-time-interval": "0s", "compact-check-time-enabled": false, "compact-check-time-interval": "1m0s", "auto-compaction-mode": "periodic", "auto-compaction-retention": "0s", "auto-compaction-interval": "0s", "discovery-url": "", "discovery-proxy": "", "downgrade-check-interval": "5s"}
```

Systemd to Manage etcd

Systemd simplifies process management, handling tasks like starting, stopping, and restarting services, including automatically starting etcd on server reboot.

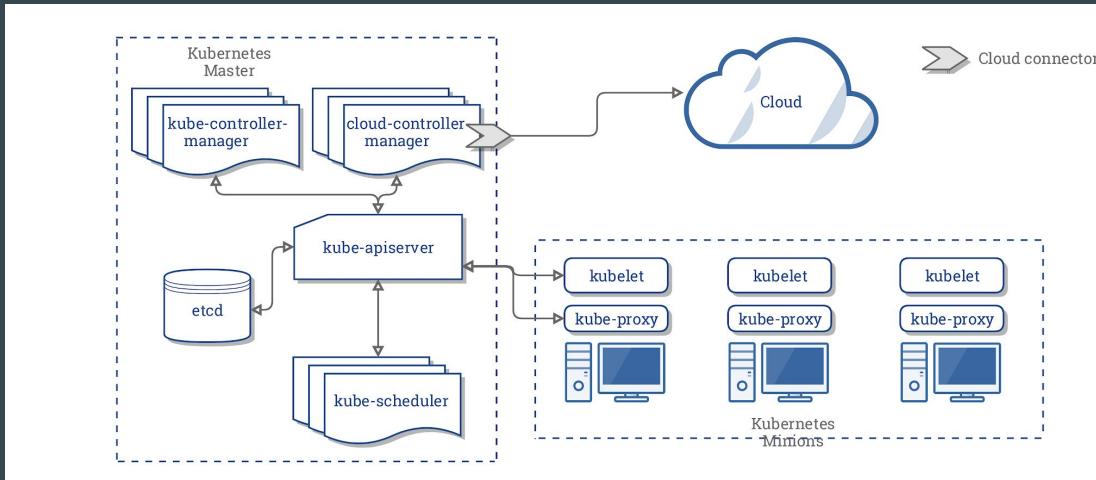
```
root@demo:~# systemctl status etcd
● etcd.service - etcd
    Loaded: loaded (/etc/systemd/system/etcd.service; disabled; preset: enabled)
    Active: active (running) since Sat 2025-02-08 17:00:42 UTC; 2min 30s ago
      Docs: https://github.com/coreos
   Main PID: 28229 (etcd)
     Tasks: 6 (limit: 1113)
    Memory: 6.9M (peak: 7.2M)
       CPU: 578ms
      CGroup: /system.slice/etcd.service
              └─28229 /usr/local/bin/etcd --cert-file=/root/certificates/etcd.crt
```

API Server Security

Setting the Base

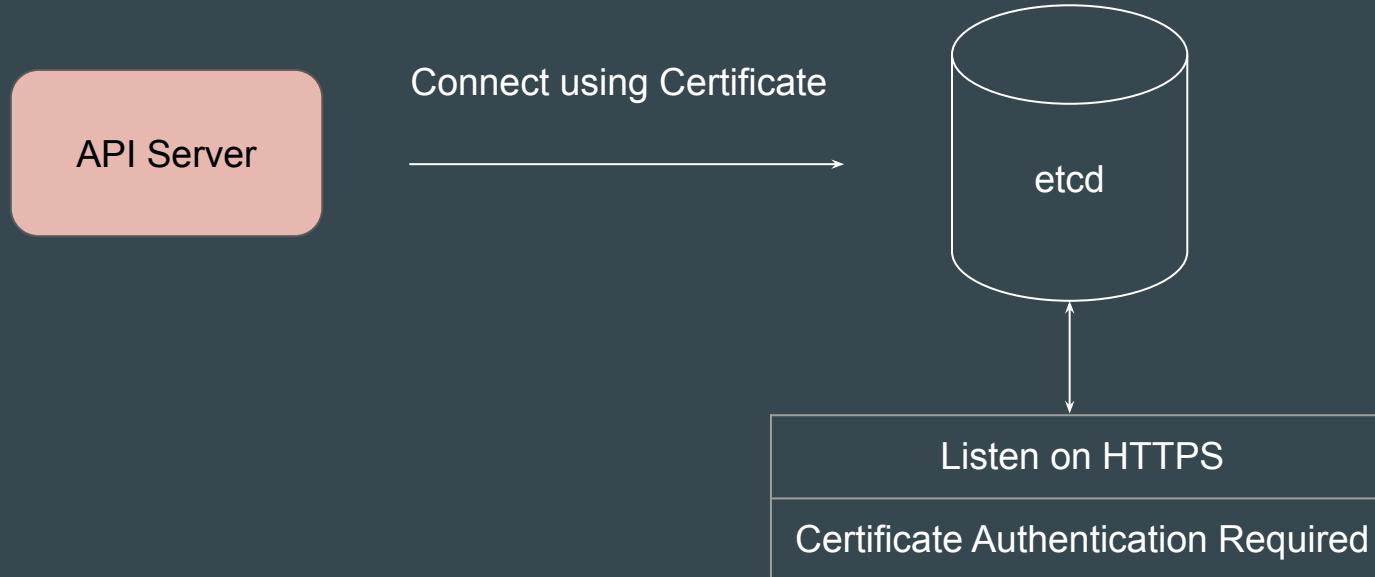
The Kubernetes API Server **acts as the gateway** for managing and interacting with a Kubernetes cluster.

When we interact with your Kubernetes cluster using the kubectl, the request goes to API server component.



Etcd Component Setup

Our etcd server is **configured to require certificate-based authentication** and operates over **HTTPS** for secure communication.



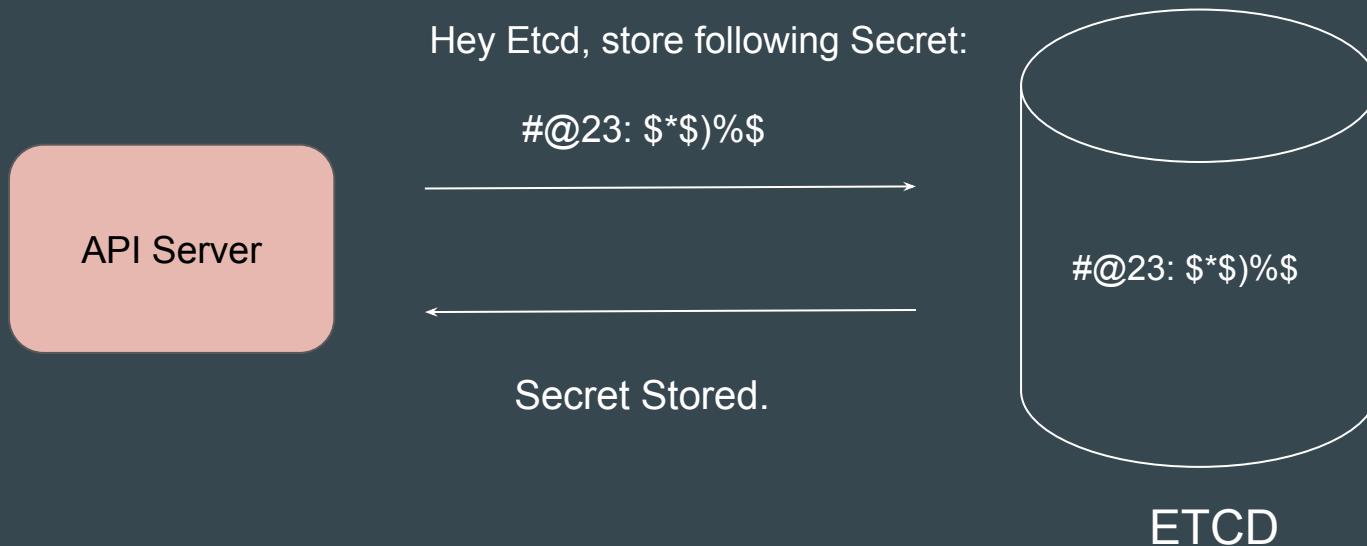
Connecting API Server to ETCD

To connect API Server with etcd component, there are two essential steps that need to be performed.

1. Generate certificates for API Server from trusted Certificate Authority.
2. API Server should connect to etcd over HTTPS endpoint.

Encryption Providers

The API Server should encrypt sensitive data, such as Kubernetes Secrets, before storing it in etcd.



TLS Encryption

Since users and other components may connect to the API Server over potentially insecure networks, it is critical to ensure that all traffic is encrypted.

API server should listen on HTTPS endpoint.



Client

encrypted-channel

API Server

Auditing

Auditing provides a security-relevant, chronological set of records documenting the sequence of actions in a cluster.

It is important to have an appropriate audit policy to capture relevant logs.

Other Security Configurations

Several additional security configurations should be implemented, which we will discuss in upcoming dedicated videos.

Some of these include:

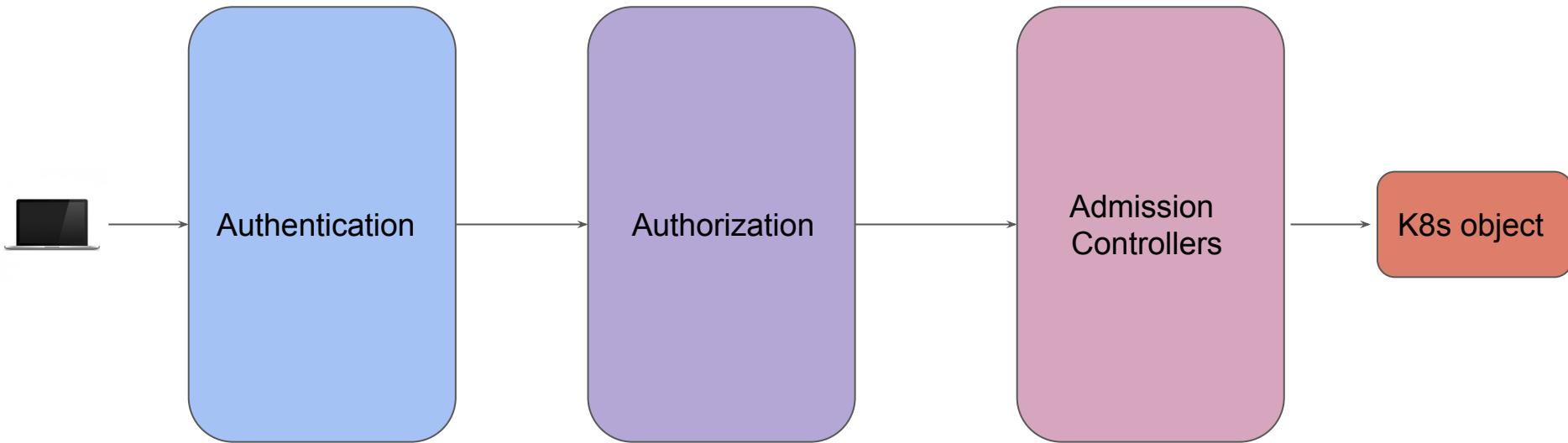
- Admission Control
- Authorization Mode
- Authentication

Access Control

K8s Security

Overview of Access Control

When a request reaches the API, it goes through several stages, illustrated in the following diagram:



Stage 1: Authentication

There are multiple ways in which we can authenticate. Some of these include:

Authentication Modes	Description
X509 Client Certificates	Valid client certificates signed by trusted CA.
Static Token File	Sets of bearer token mentioned in a file.

Stage 2: Authorization

After the request is authenticated as coming from a specific user, the request must be authorized.

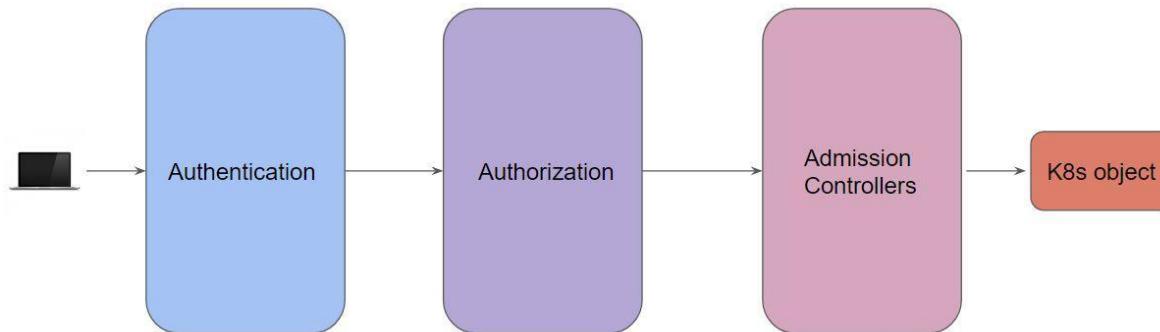
Multiple authorization modules are supported.

Authorization Modes	Description
AlwaysDeny	Blocks all requests (used in tests).
AlwaysAllow	Allows all requests; use if you don't need authorization.
RBAC	Allows you to create and store policies using the Kubernetes API.
Node	A special-purpose authorization mode that grants permissions to kubelets

Stage 3: Admission Controllers

An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized.

Controllers that can intercept Kubernetes API requests, and modify or reject them based on custom logic.

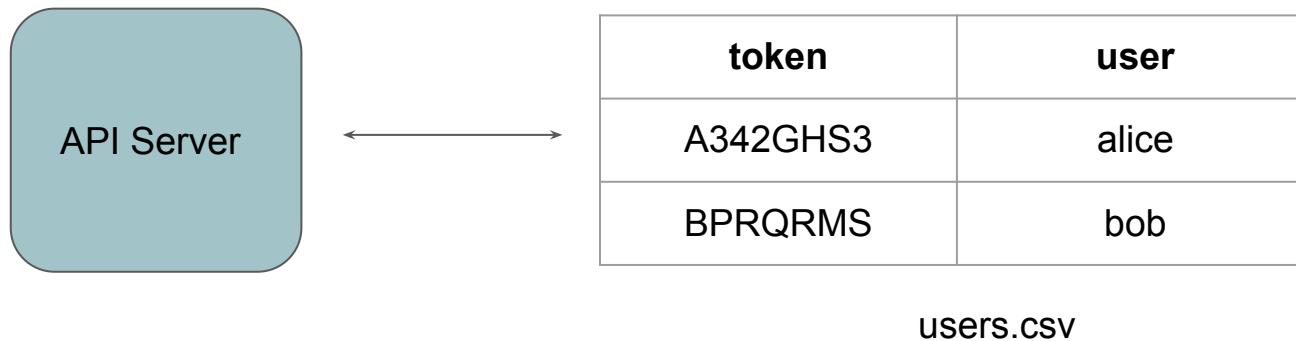


Static Token Authentication

Let's Authenticate

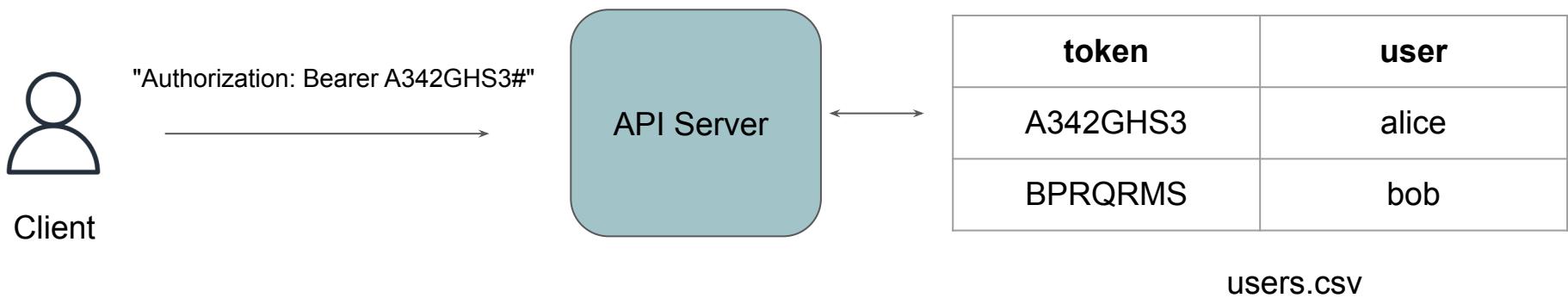
Overview of Static Token Authentication

The API server reads bearer tokens from a file when given the `--token-auth-file=SOMEFILE` option on the command line.



Connecting with API Server using Token

When using bearer token authentication from an http client, the API server expects an **Authorization** header with a value of **Bearer <token>**



Downsides of Token Authentication

Let's Authenticate Securely

Downside of Token Authentication

The tokens are stored in clear-text in a file on the apiserver

Tokens cannot be revoked or rotated without restarting the apiserver.

Hence, it is recommended to not use this type of authentication.

```
Dem0Passw0rd#, bob, 01, admins
```

X509 Client Certificates - Authentication

Let's Authenticate Securely

Authentication Types

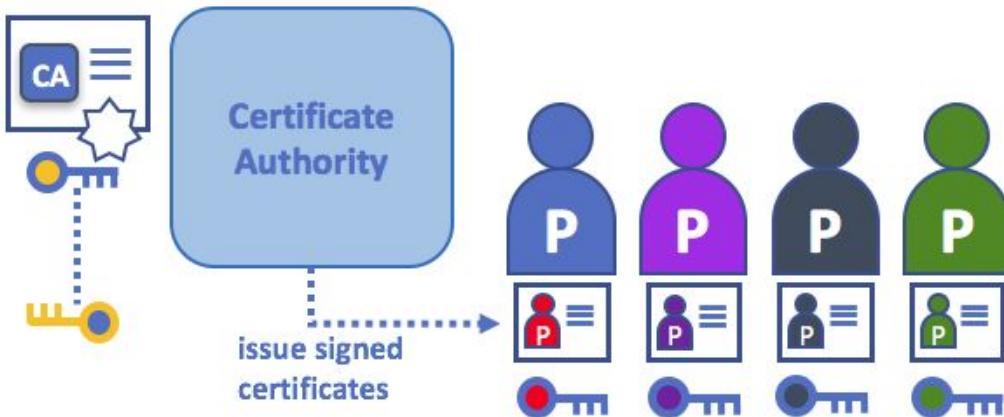
There are multiple ways in which we can authenticate. Some of these include:

Authentication Modes	Description
X509 Client Certificates	Valid client certificates signed by trusted CA.
Static Token File	Sets of bearer token mentioned in a file.

How Things Work

A request is authenticated if the client certificate is signed by one of the certificate authorities that is configured in the API server.





Downsides of X509 Client Certificates

Let's Authenticate Securely

Disadvantage of Certificate Based Authentication

The private key is stored on an insecure media (local disk storage).

Certificates are generally long-lived. Kubernetes does not support certificate revocation related area.

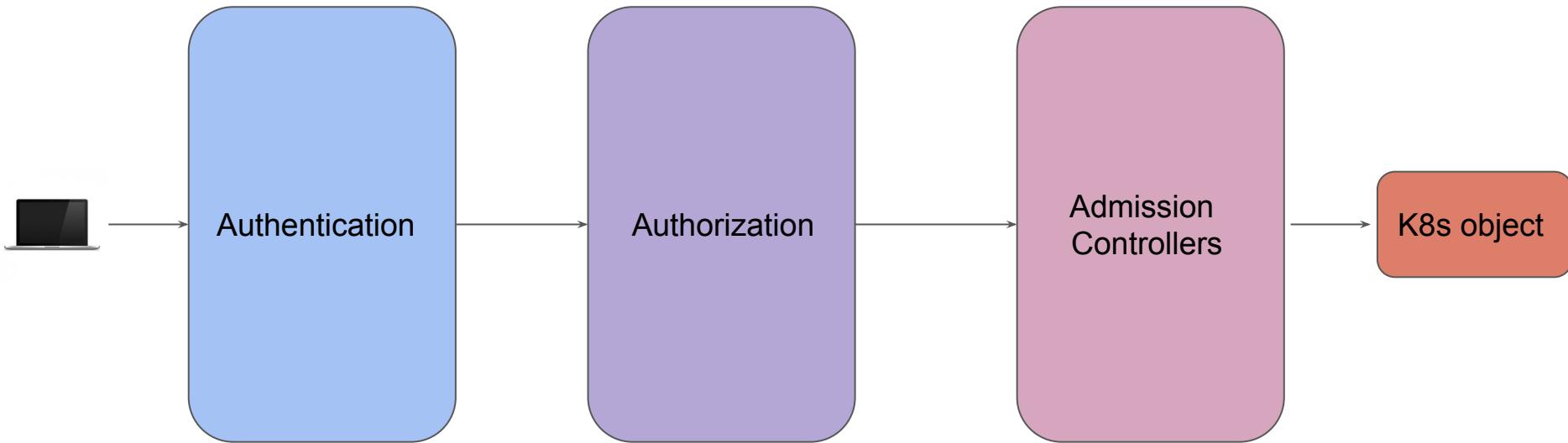
Groups are associated with Organization in certificate. If you want to change the group, you will have to issue a new certificate.

Authorization

K8s Security

Overview of Access Control

When a request reaches the API, it goes through several stages, illustrated in the following diagram:



Authorization

After the request is authenticated as coming from a specific user, the request must be authorized.

Multiple authorization modules are supported.

Authorization Modes	Description
AlwaysDeny	Blocks all requests (used in tests).
AlwaysAllow	Allows all requests; use if you don't need authorization.
RBAC	Allows you to create and store policies using the Kubernetes API.
Node	A special-purpose authorization mode that grants permissions to kubelets

Important Pointers - Certificates

Within a certificate, there are two important fields:

Common Name (CN) and **Organization (O)**

```
openssl req -new -key alice.key -subj "/CN=alice/O=admins" -out alice.csr
```

The above commands create CSR for the username alice belonging to admins group.

System Masters Group in Kubernetes

There is a group named `system:masters` and any user that are part of this group have an unrestricted access to the Kubernetes API server.

Even if every cluster role and role is deleted from the cluster, users who are members of this group retain full access to the cluster.



system:masters



normal-group

imgflip.com

Important Note

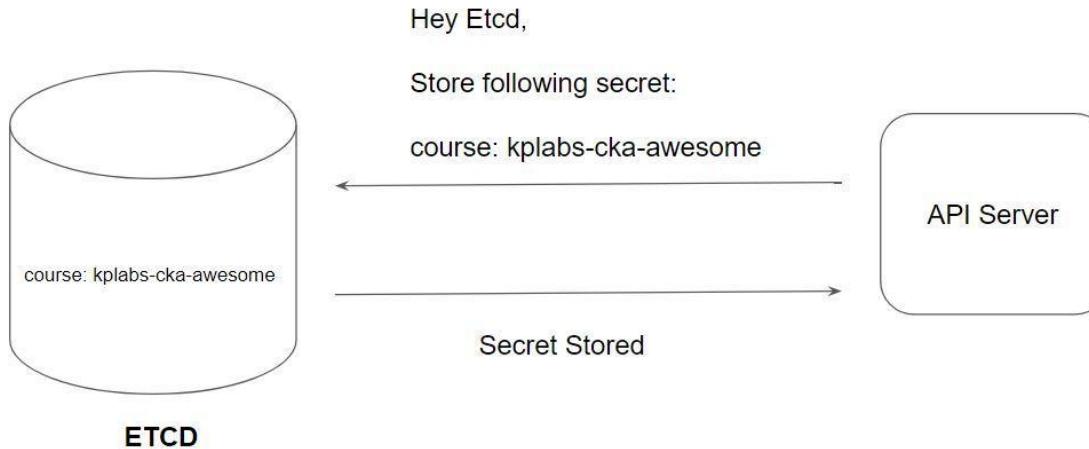
Membership of system:masters is particularly dangerous when combined with Kubernetes client certificate authentication model, as Kubernetes does not currently provide any mechanism for client certificates to be revoked.

Encryption Provider Config

Encrypting Data in ETCD

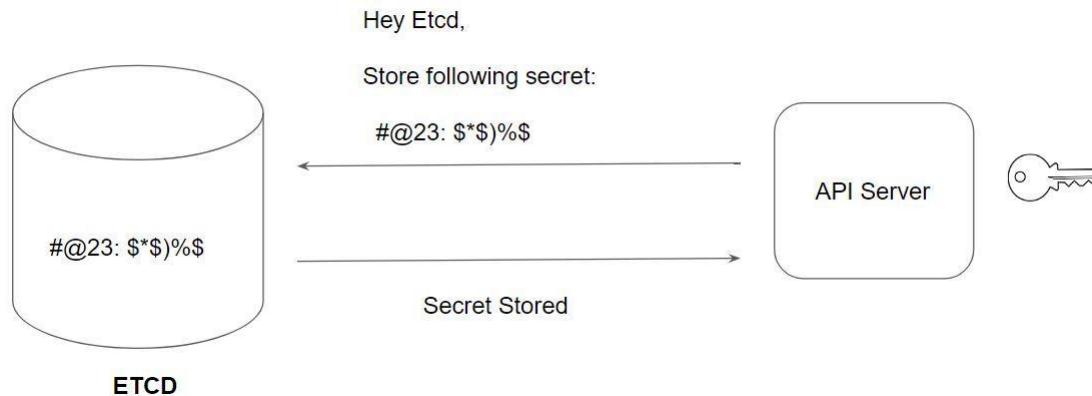
Challenge with Plain Text Storage in ETCD

Data like Kubernetes Secrets are stored in plain-text in ETCD.



Implementing Encryption

You can associate an encryption key at kube-apiserver level so that the data can be encrypted before being stored at the ETCD.



```
/registry/secret  
s/default/my-sec  
ret.k8s.....v1..  
Secret.....my  
-secret....defau  
lt".*$71bc0042-f  
04b-44d7-9488-2a  
56716307182.8.B.  
.....z...^..k  
ubectl..update..  
v1".....2.Fi  
eldsV1:3.1{"f:da  
ta":{".":{},"f:p  
assphrase":{}},"  
f:type":{}}}....p  
assphrase..topse  
cret..Opaque..".  
| .|
```

Before Encryption

```
/registry/secret  
s/default/new-se  
cret.k8s:enc:aes  
cbc:v1:key1:....  
.eu.P.....j.....  
Abbq^q.....A3..  
.!=....j.2.DP.(M  
.Z..?..;..$.5.E..  
.9....W.....IM  
#7.....=<.'t...  
.o.h.u.....>.  
{c0v..x"-%...e.  
....."....a[...;  
.J,.2G.=.l=....=h  
..(...../.....  
:.....e..c....D  
&...XqX..#....  
.g.[.r.RX=D....vc  
2.9hwh.BN....P...  
.hkE..Q.....|
```

After Encryption

Encryption Provider Configuration

The kube-apiserver process accepts an argument `--encryption-provider-config` that controls how API data is encrypted in etcd.

```
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets
providers:
  - aescbc:
    keys:
      - name: key1
        secret: ${ENCRYPTION_KEY}
    - identity: {}
```

Encryption providers

Encryption Providers	Encryption	Strength	Speed
Identity	None	N/A	N/A
aescbc	AES-CBC with PKCS#7 padding	Strongest	Fast
secretbox	XSalsa20 and Poly1305	Strong	Faster
kms	Uses envelope encryption scheme	Strongest	Fast

Important Pointers

By default, the identity provider is used to protect secrets in etcd, which provides no encryption.

You can make use of KMS provider for additional security.

The older secrets would still be in an unencrypted form.

Auditing

Let's Audit Securely

Overview of Auditing

Auditing provides a security-relevant, chronological set of records documenting the sequence of actions in a cluster.

The cluster audits the activities generated by users, by applications that use the Kubernetes API, and by the control plane itself.

- what happened?
- when did it happen?
- who initiated it?
- on what did it happen?
- from where was it initiated?
- to where was it going?

```
{  
    "kind": "Event",  
    "apiVersion": "audit.k8s.io/v1",  
    "level": "Metadata",  
    "auditID": "388ca4e1-c368-45b4-aca9-652e32baf8e1",  
    "stage": "RequestReceived",  
    "requestURI": "/api/v1/namespaces/default/secrets?limit=500",  
    "verb": "list",  
    "user": {  
        "username": "bob",  
        "groups": [  
            "system:masters",  
            "system:authenticated"  
        ]  
    },  
    "sourceIPs": [  
        "127.0.0.1"  
    ],  
    "userAgent": "kubectl/v1.19.0 (linux/amd64) kubernetes/e199641",  
    "objectRef": {  
        "resource": "secrets",  
        "namespace": "default",  
        "apiVersion": "v1"  
    },  
    "requestReceivedTimestamp": "2020-12-03T16:55:10.357789Z",  
    "stageTimestamp": "2020-12-03T16:55:10.357789Z"  
}
```

Audit Policy

Audit policy defines rules about what events should be recorded and what data they should include.

Audit Levels	Description
None	don't log events that match this rule.
Metadata	Log request metadata (requesting user, timestamp, resource, verb, etc.) but not request or response body.
Request	Log event metadata and request body but not response body.
RequestResponse	Log event metadata, request and response bodies.

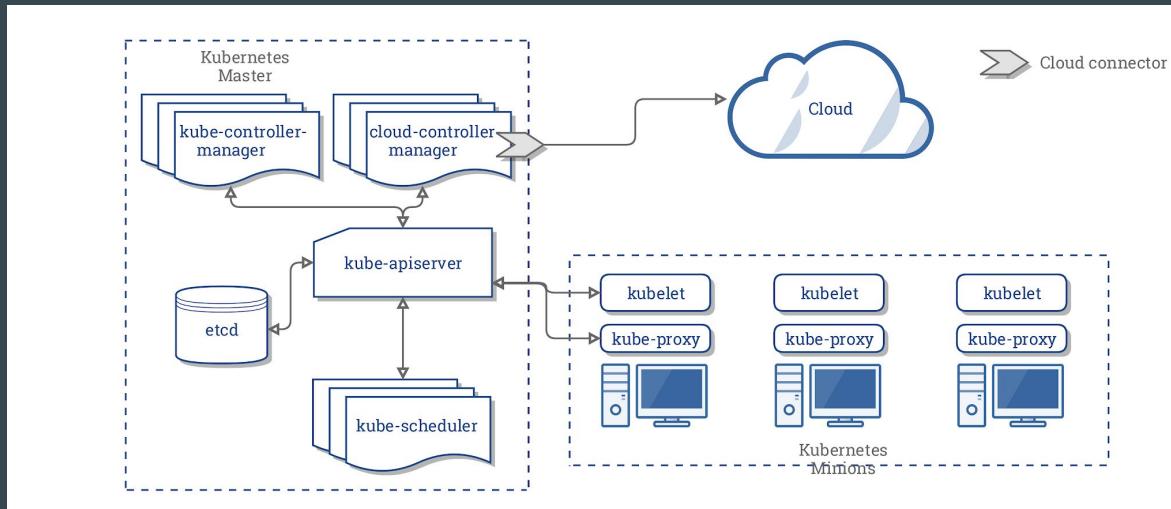
Important Flags

Audit Configuration	Description
-audit-policy-file	Path to the file that defines the audit policy configuration.
-audit-log-path	Specifies the log file path that log backend uses to write audit events.
--audit-log-maxage	Maximum number of days to retain old audit log files
--audit-log-maxbackup	Maximum number of audit log files to retain
--audit-log-maxsize	Maximum size in MB of the audit log file before it gets rotated

Setting Up kubeadm

Understanding the Need

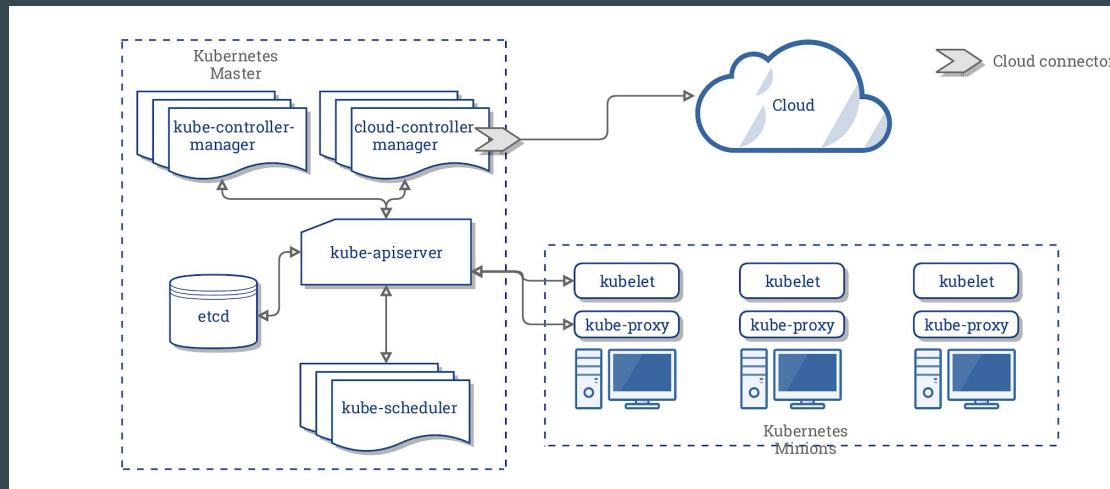
kubeadm allows us to provision a **secure** Kubernetes cluster quickly.



Two Part to Remember

First important component is Kubernetes **Master Node**

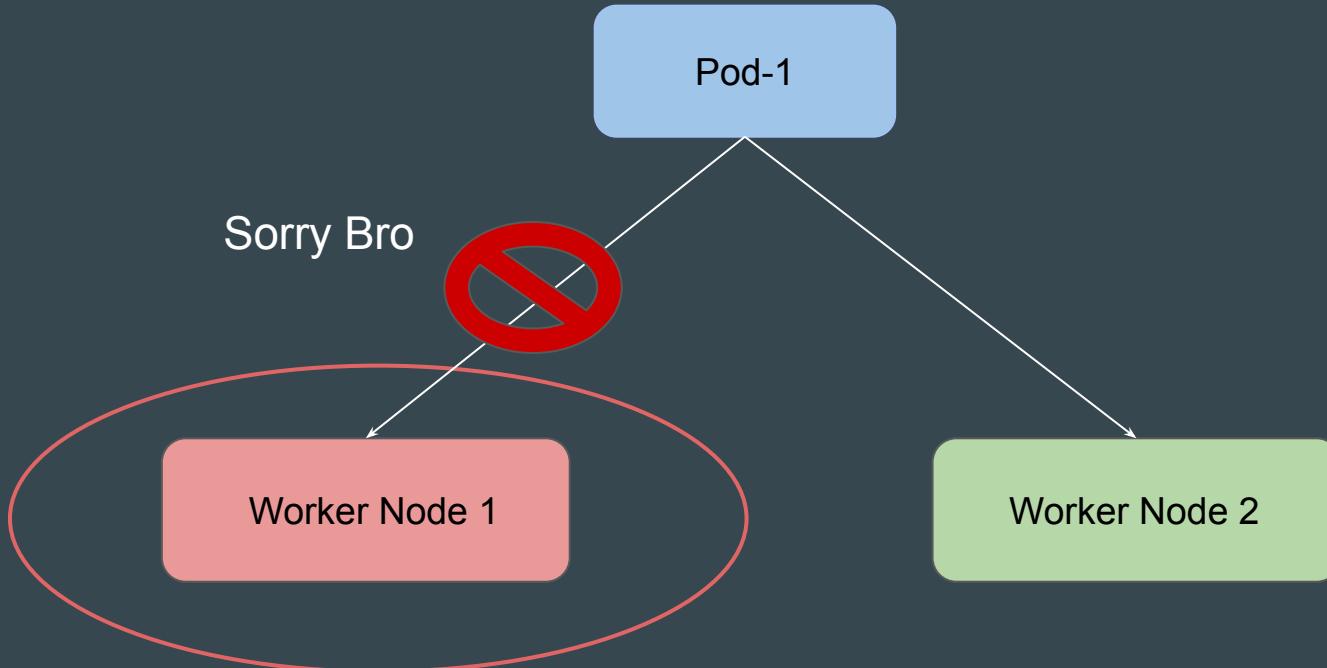
Second component is Kubernetes **Worker Node**



Taints and Tolerations

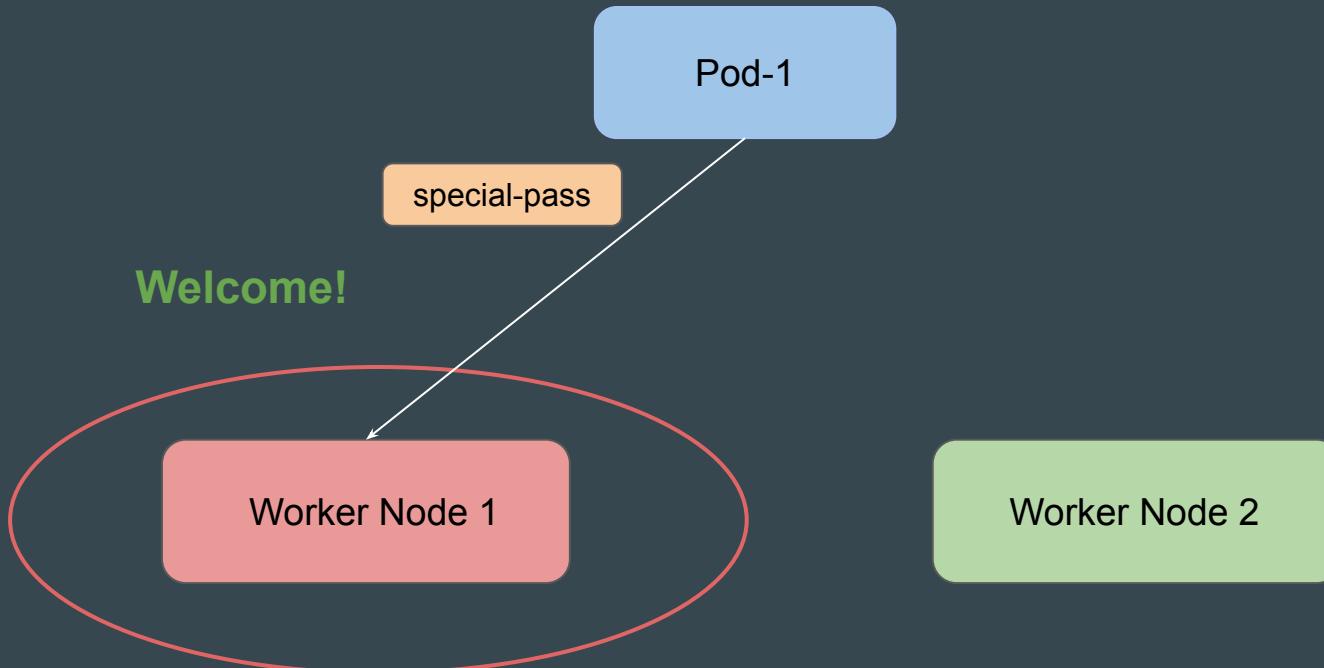
Understanding Taint

Taint is a property added to a node that repels certain pods



Understanding Tolerations

In order to schedule into the worker node with taint, you need a **special pass**. This special pass is called **Toleration**.



Defining a Taint

You can use the `kubectl taint node` command to add a taint to a node.

A taint consists of a key, value (optional), and effect.

```
root@kubeadm-2:~# kubectl taint node worker-01 key=value:NoSchedule  
node/worker-01 tainted
```

Effects in Taints

Effects	Description
NoSchedule	Prevents scheduling of new pods on the node unless they tolerate the taint.
PreferNoSchedule	Tries to avoid scheduling new pods on the node, but does not enforce it strictly.
NoExecute	Evicts existing pods and prevents new pods from being scheduled on the node.

Defining Tolerations - Sample Manifest

```
! pod-tolerate.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx
  tolerations:
  - key: "example-key"
    operator: "Exists"
    effect: "NoSchedule"
```

Defining Toleration - Structure

Component	Description	Important Pointers
key	The taint key that the toleration applies to. This is used to match a taint on a node.	
operator	Defines the relationship between the key and the value.	<ul style="list-style-type: none">- Equal: The toleration matches if the key and value are equal.- Exists: The toleration matches if the key exists, regardless of the value.
effect	Specifies the effect of the taint.	
value	The value associated with the key that the toleration applies to.	Any string value that matches the value of the taint

Revising the Concept

Taints repel pods, and tolerations allow exceptions.

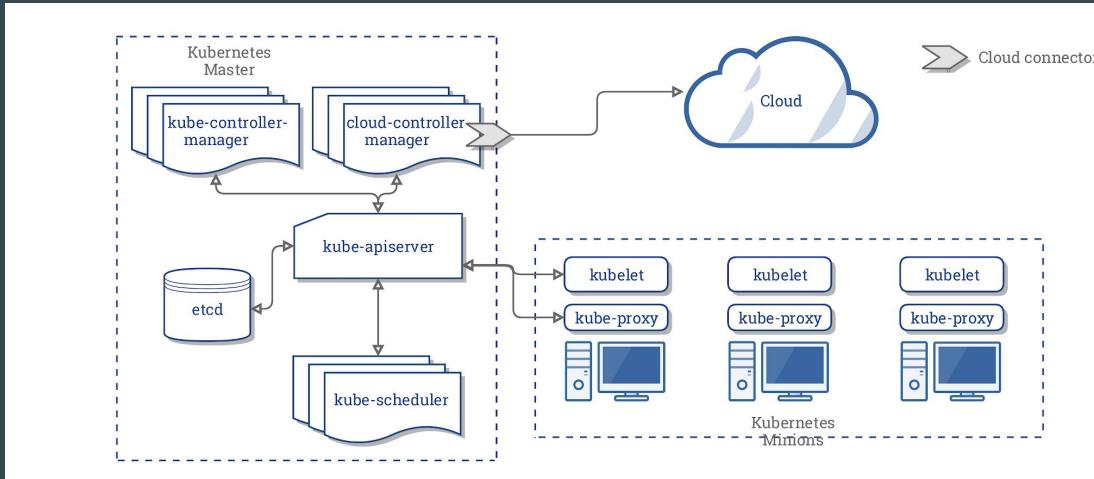
A pod without a toleration for a node's taint will not be scheduled on that node.

Taints are applied to nodes, while tolerations are applied to pods.

Kubelet Security

Basics of Kubelet API

The **Kubelet API** provides a set of endpoints that allow users to interact with the **Kubelet** to retrieve information about the node, running pods, and container statuses.



Accessing Kubelet API

The Kubelet API is available on each node at:

<https://<node-ip>:10250/>

```
root@worker:~# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp      0      0 127.0.0.1:9099           0.0.0.0:*            LISTEN     4148/calico-node
tcp      0      0 0.0.0.0:179             0.0.0.0:*            LISTEN     4361/bird
tcp      0      0 127.0.0.53:53            0.0.0.0:*            LISTEN     677/systemd-resolve
tcp      0      0 127.0.0.54:53            0.0.0.0:*            LISTEN     677/systemd-resolve
tcp      0      0 127.0.0.1:41117           0.0.0.0:*            LISTEN     2170/containerd
tcp      0      0 127.0.0.1:10248           0.0.0.0:*            LISTEN     3232/kubelet
tcp      0      0 127.0.0.1:10249           0.0.0.0:*            LISTEN     3452/kube-proxy
tcp6     0      0 ::::22                  ::::*                 LISTEN     1/init
tcp6     0      0 ::::10256               ::::*                 LISTEN     3452/kube-proxy
tcp6     0      0 ::::10250               ::::*                 LISTEN     3232/kubelet
tcp6     0      0 ::::32647               ::::*                 LISTEN     3452/kube-proxy
```

Understanding the Challenge

If kubelet is misconfigured, the Kubelet API can be accessible to everyone over internet without authentication.

```
C:\>curl -k https://143.244.140.236:10250/pods
{"kind":"PodList","apiVersion":"v1","metadata":{},"items":[{"metadata":{"name":"csi-node-driver-sb2c4","de-driver-","namespace":"calico-system","uid":"1521e4e6-9270-43d4-930e-cf838fdee7f3","resourceVersion":1,"creationTimestamp":"2025-02-10T08:25:28Z","labels":{"app.kubernetes.io/name":"csi-node-driver","controller-revision-hash":"k8s-app-csi-node-driver","name":"csi-node-driver","pod-template-generation":"1"},"annotations":{"cni.projectcalico.org/podIP":172.17.0.1,"cni.projectcalico.org/podIPs":172.17.0.1/32,"kubernetes.io/config.seen":2025-02-10T08:35:33.093Z,"kubernetes.io/config.source":"api"},"ownerReferences":[{"apiVersion":"apps/v1","kind":"DaemonSet","name":"csi-node-driver-56-7957-4e0f-9fd9-5d7d70730b73","controller":true,"blockOwnerDeletion":true}],"managedFields":[{"manager":"Kubelet API","operation":"Update","apiVersion":"v1","time":"2025-02-10T08:25:28Z","fieldsType":"FieldsV1","fieldsV1":{"f:generateName":{},"f:labels":{},"f:app.kubernetes.io/name":{},"f:controller-revision-hash":{},"f:nodeSelector":{},"f:pod-template-generation":{},"f:ownerReferences":{},"k:{\\"uid\\":\"dc88f756-7957-4e0f-9fd9-5d7d70730b73\"}":{},"f:spec":{"f:affinity":{},"f:nodeAffinity":{},"f:requiredDuringSchedulingIgnoredDuringExecution":{}}}}]}
```

Anonymous Authentication

Anonymous authentication in Kubernetes allows unauthenticated requests to the Kubelet API.

It is primarily used as a fallback mechanism when no other authentication method is provided.

```
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
```

Authorization Mode

Kubelet supports different authorization modes to control which requests are allowed.

Feature	AlwaysAllow	WebHook
Security Level	Low (No authorization)	High (Centralized authorization)
Use Case	Development, testing	Production, fine-grained access control
Authorization Mechanism	Allows all requests	Uses an external webhook to decide
Recommended for Production?	No	Yes

Client Certificates

When a client (such as kube-apiserver or other components) connects to the kubelet API, it must present a TLS certificate.

The kubelet verifies the presented client certificate against the CA certificate stored defined through clientCAfile option.

```
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

HTTPS for Kubelet

The `--tls-cert-file` and `--tls-private-key-file` allow you to specify path of certificate and key used for serving HTTPS request at kubelet.

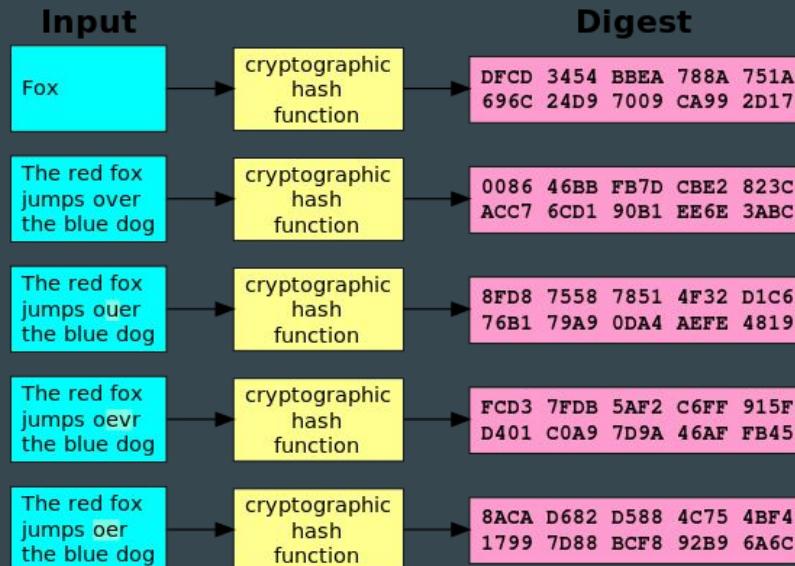
If `--tls-cert-file` and `--tls-private-key-file` are not provided, a self-signed certificate and key are generated.

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 9067462728377425652 (0x7dd6194f5b07def4)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN = worker-ca@1739175928
    Validity
      Not Before: Feb 10 07:25:27 2025 GMT
      Not After : Feb 10 07:25:27 2026 GMT
    Subject: CN = worker@1739175928
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
```

Verify Platform Binaries

Basic of Hashing

Hashing is a one-way function that maps data of arbitrary size (often called the "message") to a bit array of a fixed size (message digest)



Verify Platform Binaries

To verify the integrity of the archive, you can take a hash of the archive file and compare it with the hash value posted in the official website.



sha512sum kubernetes.tar.gz

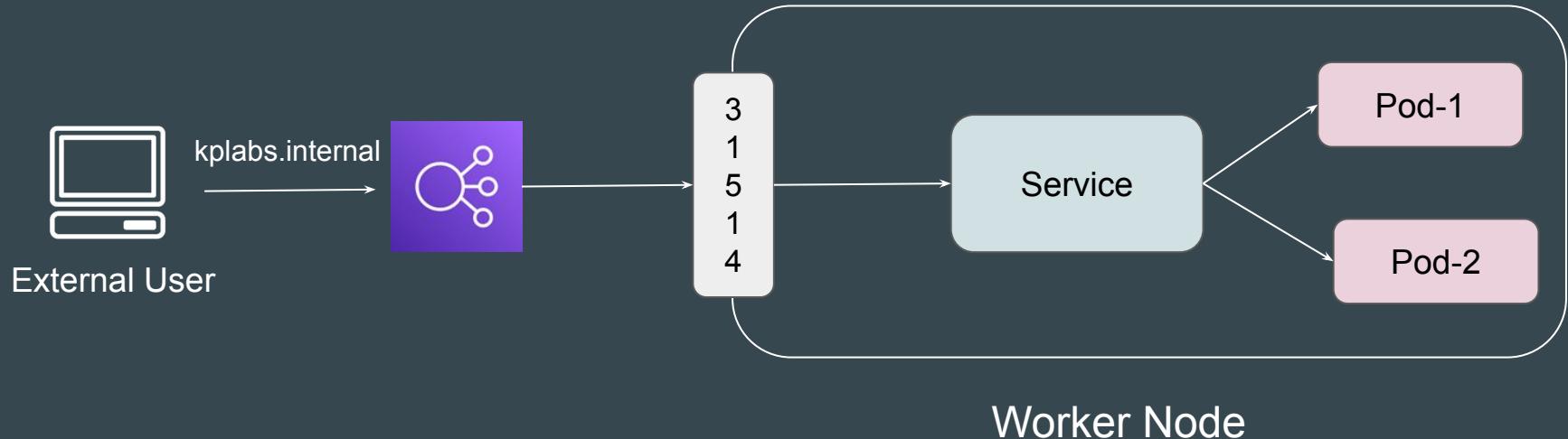


kubernetes.tar.gz

Ingress

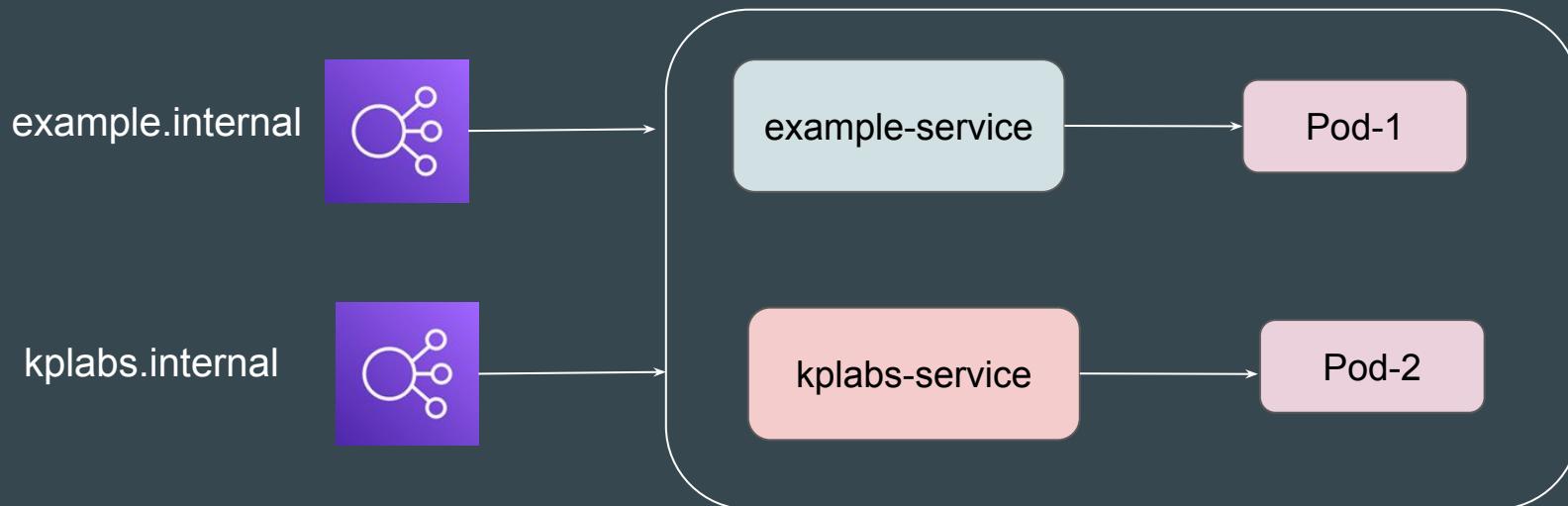
Challenge with Basic Configuration

When we use a LoadBalancer Service Type, the Load balancer forwards traffic to a NodePort associated with a single service.



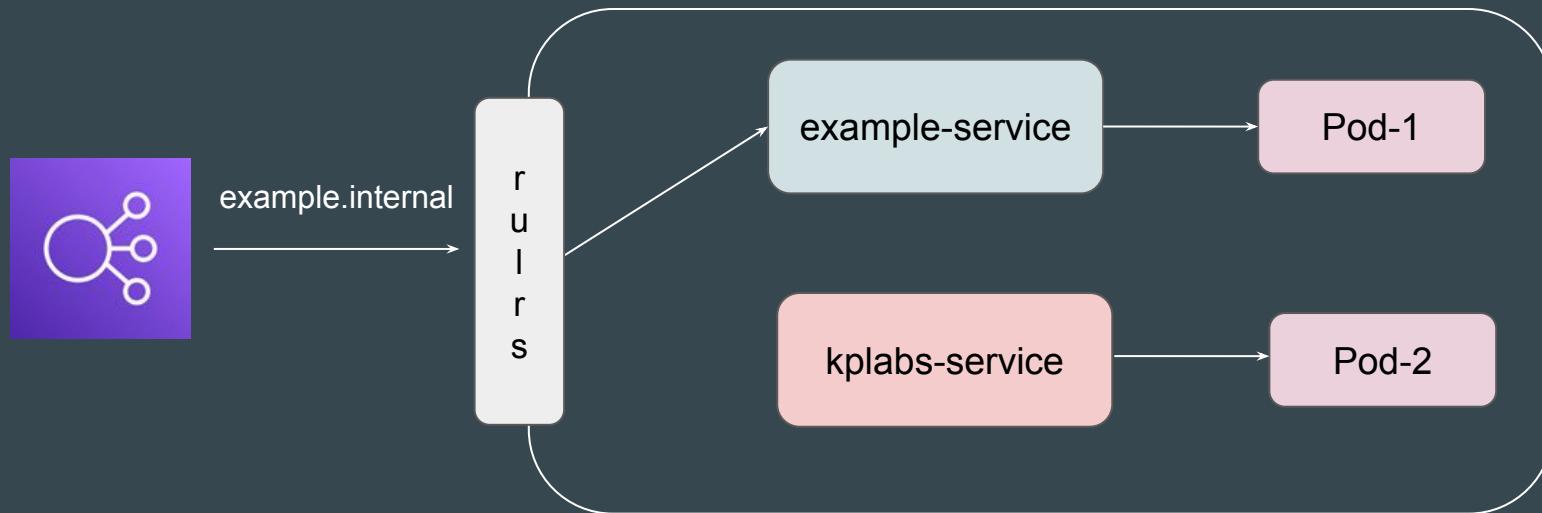
Multiple Service Scenario

In a scenario where you have multiple services for different websites, you might have to create multiple sets of load balancers for each service. This is expensive.



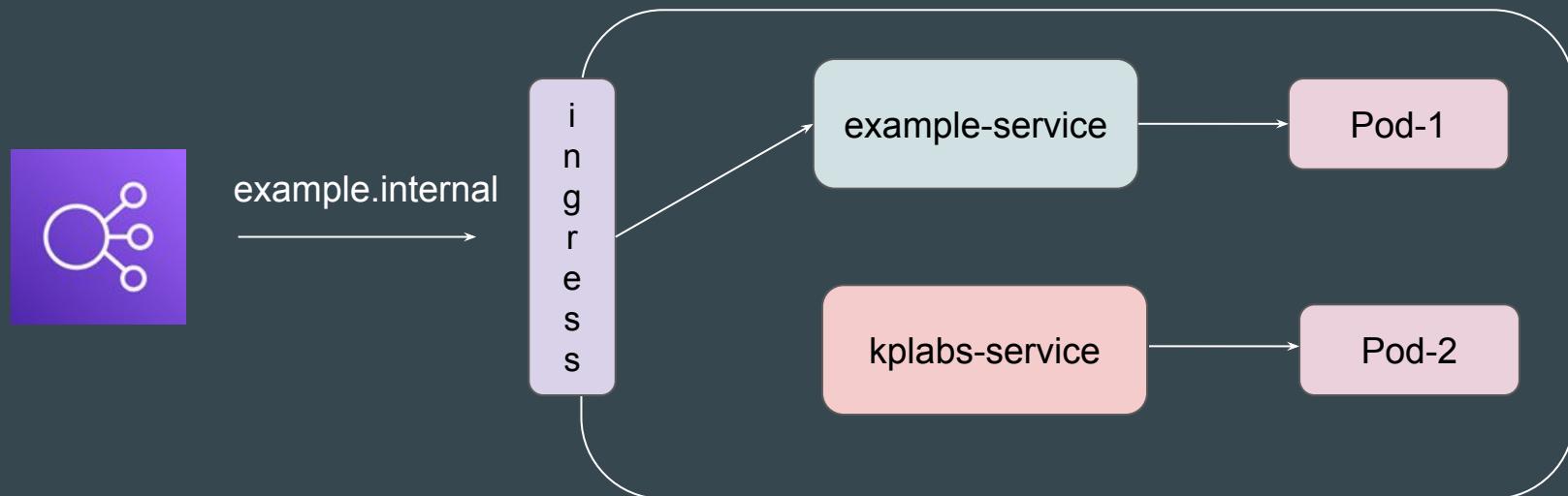
Ideal Approach

In an ideal approach, you want a single load balancer to handle requests for multiple services and a logic that can route traffic accordingly.



Introducing Ingress

Ingress acts as an entry point that routes traffic to specific services based on rules you define.



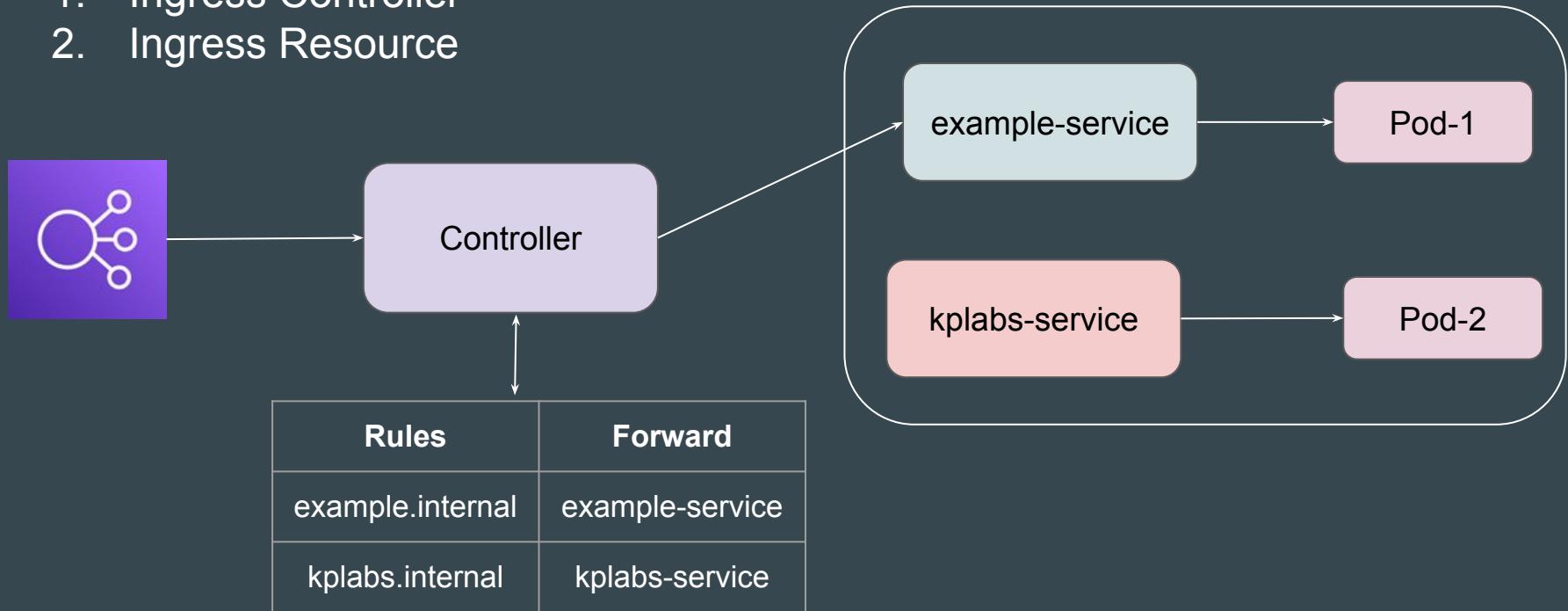
Reference Diagram



Components of Ingress

There are two sub-components of Ingress:

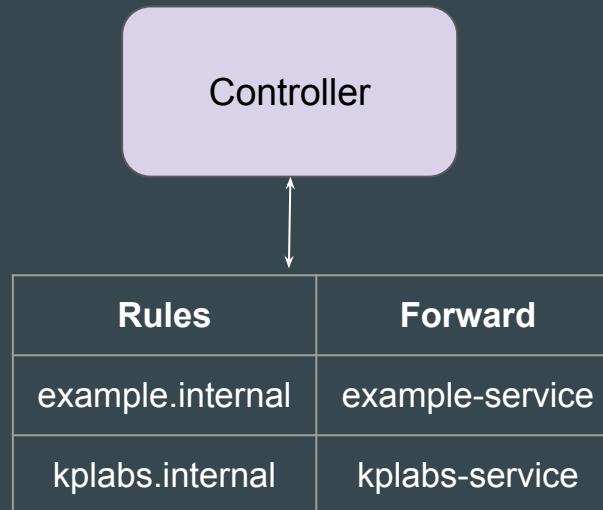
1. Ingress Controller
2. Ingress Resource



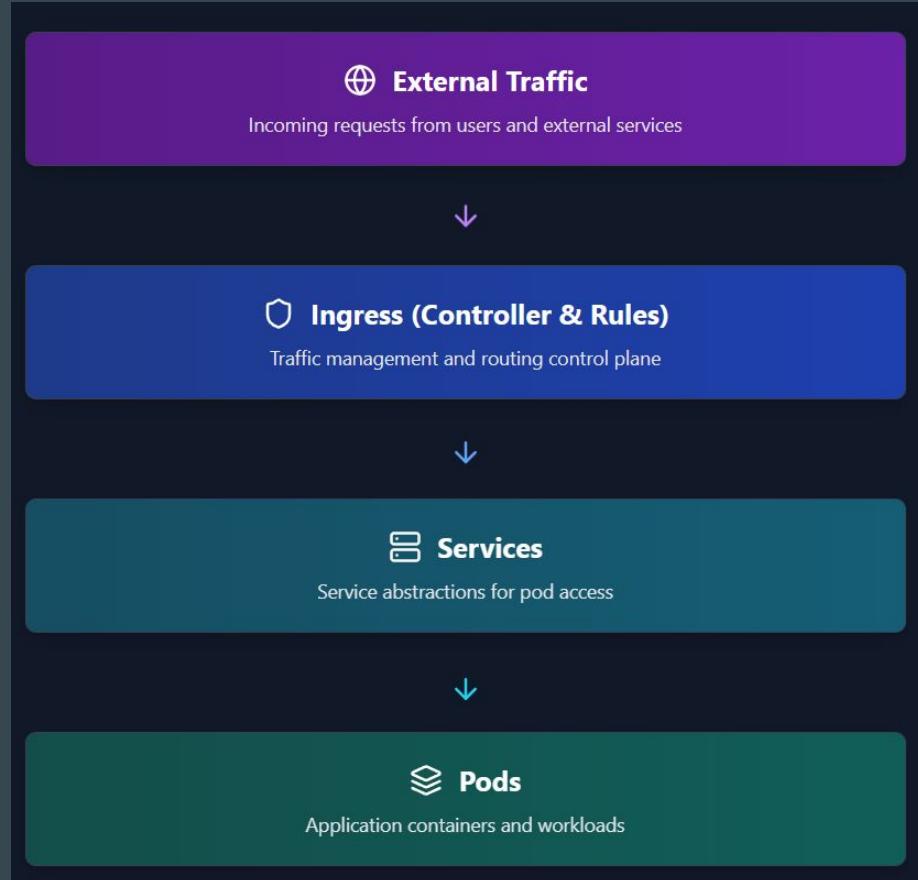
Components of Ingress

An **Ingress Controller** is a component that implements the rules defined in Ingress resources.

Ingress Controller is a running application within your cluster.



Reference Workflow Diagram



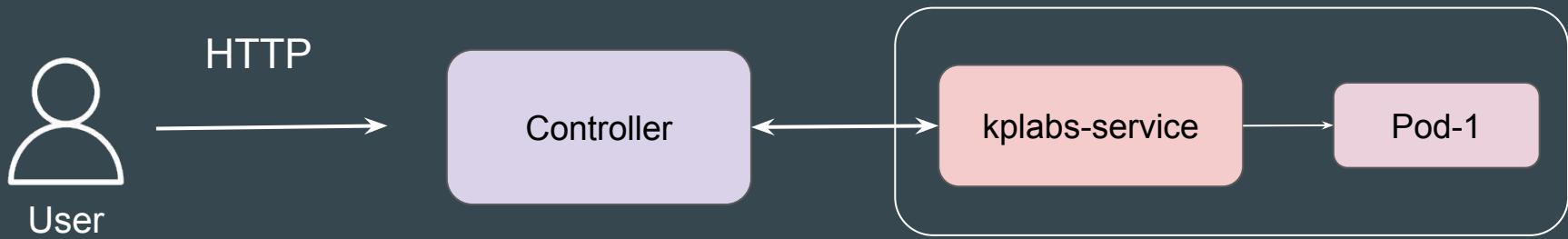
Key Difference Summarized

Ingress	Ingress Controllers
API object (rules, configuration)	Application (implements the rules)
Defines routing rules	Enforces routing rules, manages traffic flow

Ingress with TLS

Understanding the Challenge

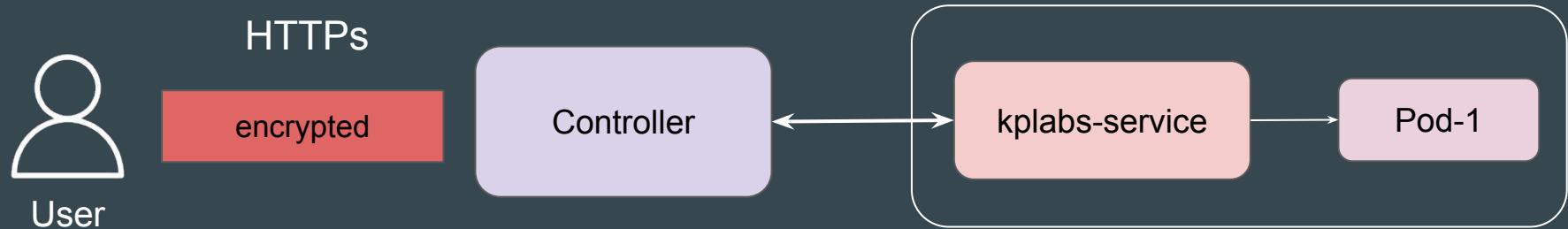
A HTTP based connection to the ingress controller is not a secure.



Ingress with TLS

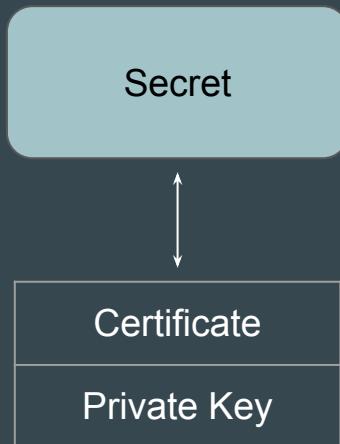
TLS ensures **secure communication** between the client and the server

You can secure the connection by setting up TLS at Ingress level.



Point to Note

The certificates are stored in Kubernetes as **Secrets**, and the Ingress resource is configured to use these secrets for HTTPS traffic.



```
root@kubeadm:~# kubectl describe ingress tls-ingress
Name:          tls-ingress
Labels:        <none>
Namespace:    default
Address:
Ingress Class:  nginx
Default backend: <default>
TLS:
  tls-cert terminates demo.kplabs.in
Rules:
  Host            Path  Backends
  ----           ----  -----
  demo.kplabs.in      /   example-service:80 (192.168.45.195:80)
```

Reference Screenshot

```
root@kubeadm:~# kubectl describe secret tls-cert
Name:         tls-cert
Namespace:    default
Labels:       <none>
Annotations: <none>
Type:        kubernetes.io/tls

Data
=====
tls.crt:  2839 bytes
tls.key:  241 bytes
```

```
root@kubeadm:~# kubectl describe ingress tls-ingress
Name:           tls-ingress
Labels:         <none>
Namespace:      default
Address:
Ingress Class: nginx
Default backend: <default>
TLS:
→   tls-cert terminates demo.kplabs.in
Rules:
  Host          Path  Backends
  ----          ---   -----
  demo.kplabs.in
                                /  example-service:80 (192.168.45.195:80)
```

Nginx Ingress - SSL Redirect Annotation

Setting the Base

By default the controller redirects HTTP clients to the HTTPS port 443 if TLS is enabled for that Ingress.

NGINX assumes that once TLS is defined in the Ingress, all traffic should be secure.

```
root@kubeadm:~# curl -I http://example.internal:30239
HTTP/1.1 308 Permanent Redirect
Date: Thu, 06 Mar 2025 17:14:05 GMT
Content-Type: text/html
Content-Length: 164
Connection: keep-alive
Location: https://example.internal
```

Change the Setting

To modify this behavior, you can add following **annotation** to ingress resource.

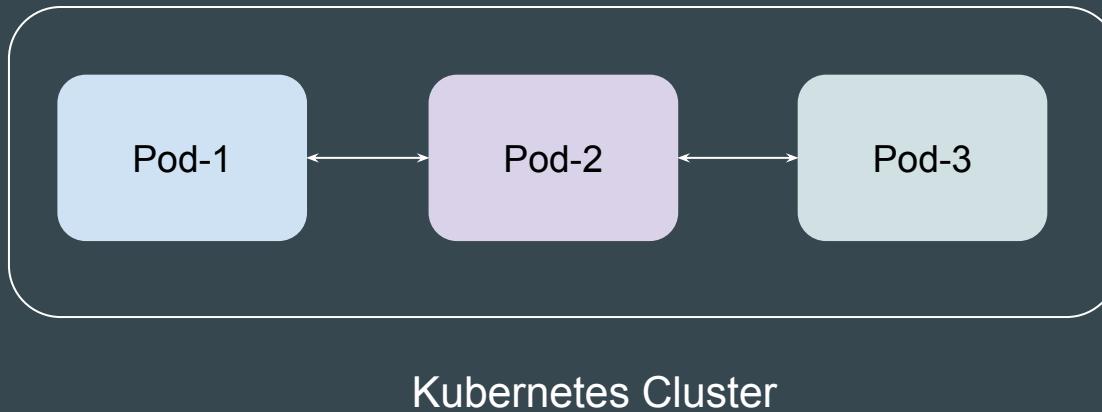
[nginx.ingress.kubernetes.io/ssl-redirect](#): true

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
  creationTimestamp: "2025-03-06T16:59:09Z"
  generation: 1
  name: demo-ingress
  namespace: default
  resourceVersion: "2998"
```

Overview of Network Policies

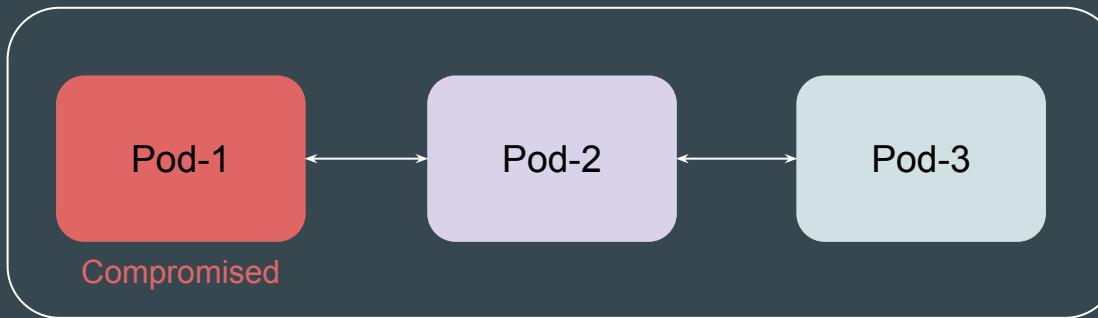
Understanding the Basics

By default, Kubernetes **allows all traffic between pods within a cluster**. Network Policies help you lock down this open communication.



Understanding the Challenge

If a application inside any Pod gets compromised, attacker can essentially communicate with all other Pods easily over the network.



Kubernetes Cluster

Ideal Scenario

You only want Pods that have genuine requirement to connect to other pods to be able to communicate.



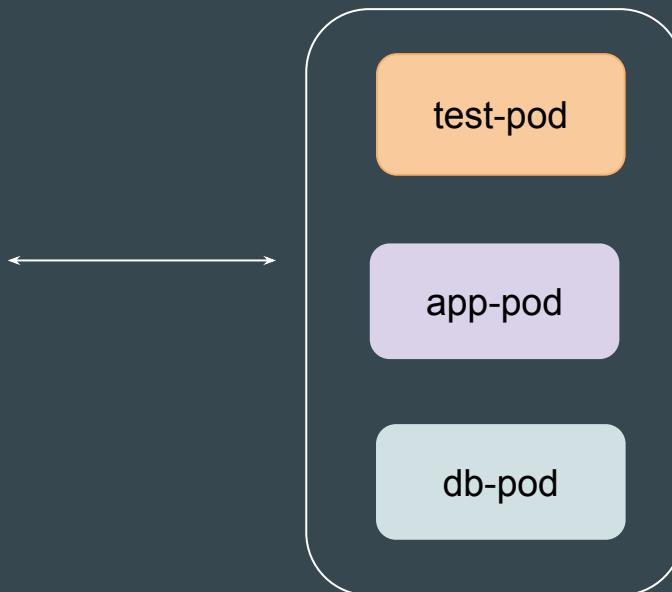
Kubernetes Cluster

Introducing Network Policies

Network Policies are a **mechanism for controlling network traffic flow** in Kubernetes clusters.

Source	Destination	Effect
app-pod	db-pod	Allow
test-pod	ALL	Deny

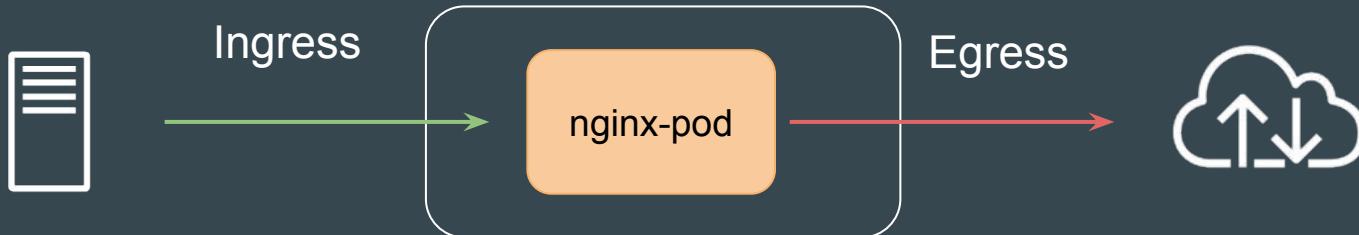
Network Policies



Types of Rules

There are two types of rules supported as part of Network policies:

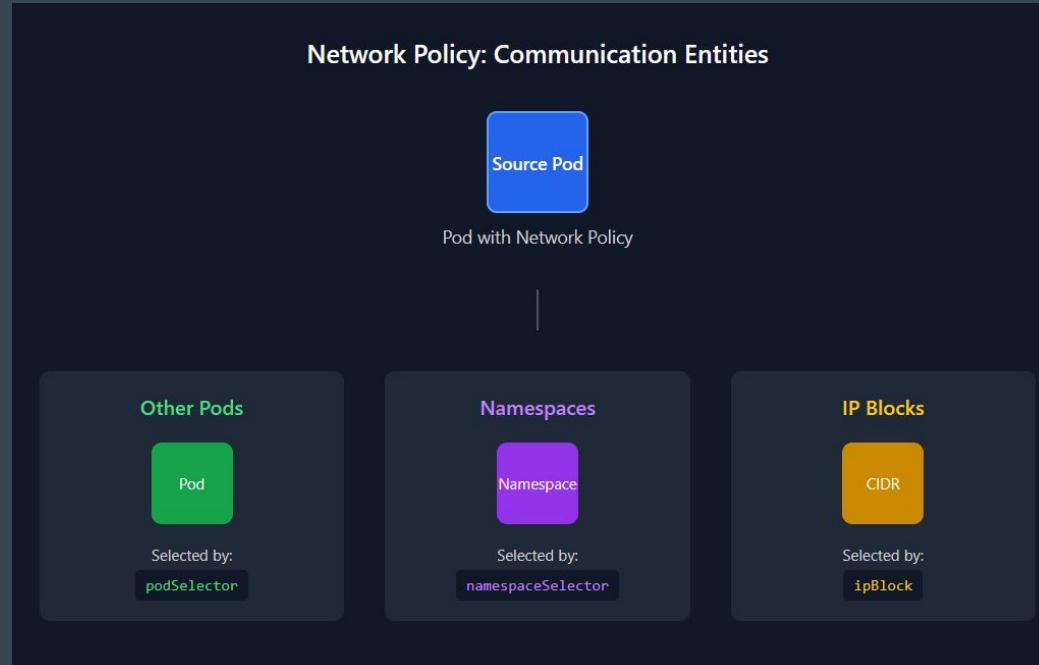
1. Ingress Rules (Inbound Rule)
2. Egress Rules (Outbound Rule)



Supported Filtering Entities

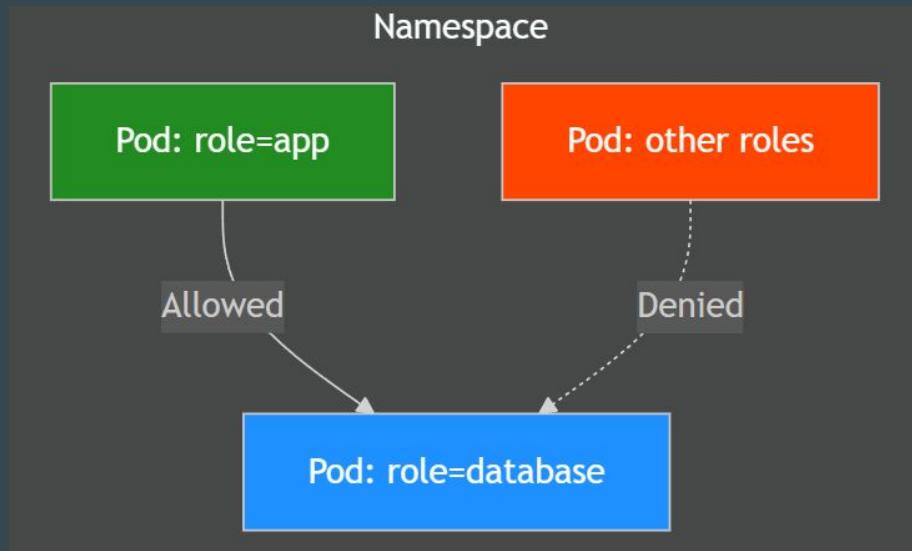
The entities that a Pod can communicate with are identified through a combination of the following three identifiers:

1. Other pods
2. Namespaces
3. IP Blocks



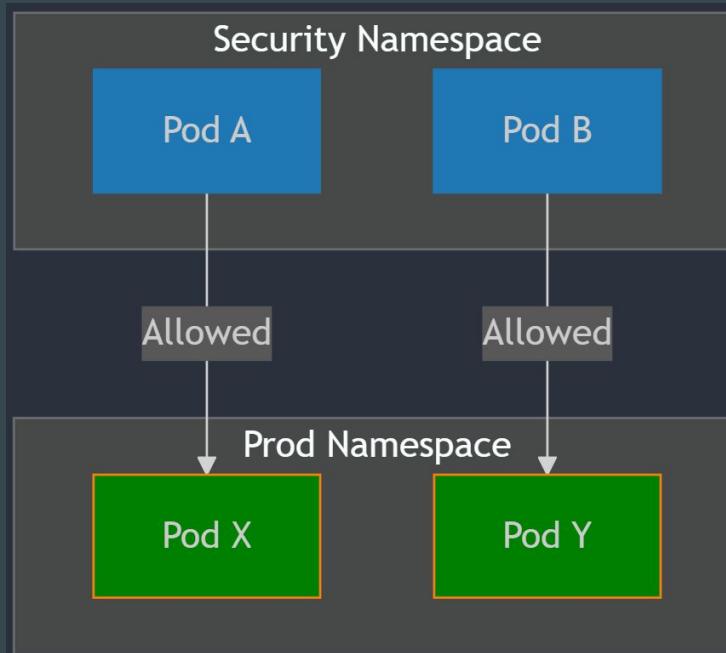
Example 1 - Pod Selector

Allow pods with label of role=app to connect to pods with labels of role=database



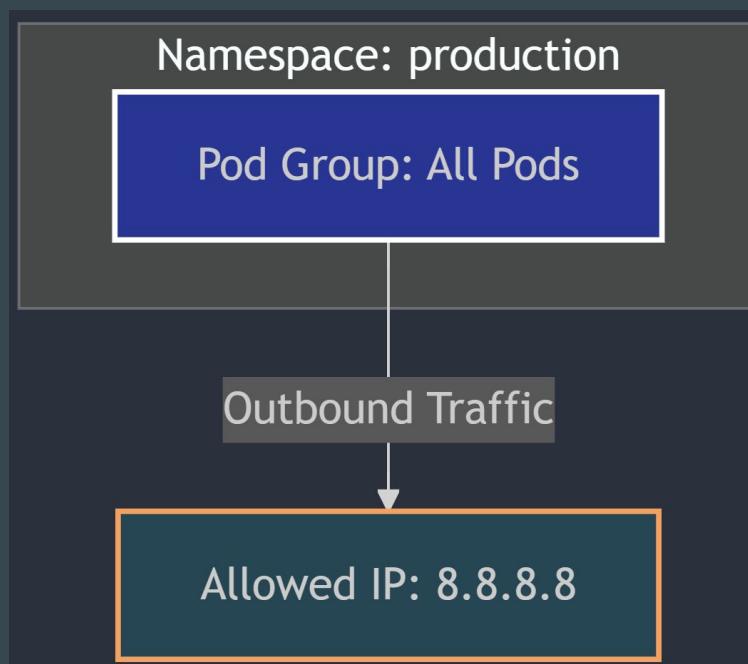
Example 2 - NameSpace Selector

Allow Pods from Security namespace to connect to Pods in Prod namespace.



Example 3 - ipBlock

Allow Pods from production namespace to connect to only 8.8.8.8 IP address outbound.



Support for Network Policy

Not all Kubernetes network plugins (CNI) support NetworkPolicy.

The ability to enforce NetworkPolicies is a feature that must be implemented by the CNI plugin

Some Network Plugins like Calico, Cilium, etc supports Network policy.

Some Network plugins like kubenet, Flannel does NOT support network Policy

Network Policy and Managed K8s Cluster

Most managed Kubernetes services (like AKS, EKS, GKE) come with a CNI that supports NetworkPolicy by default.

However, it's always a good idea to check the documentation for your specific service to confirm.

Structure of Network Policy

Sample Network Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              env: security
  egress:
    - to:
        - ipBlock:
            cidr: 8.8.8.8/32
```

Basic Mandatory Fields

As with all other Kubernetes config, a NetworkPolicy needs the following fields:

- apiVersion
- kind
- metadata

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
```

Contents of Spec

NetworkPolicy spec has all the information needed to define a particular network policy in the given namespace.

- podSelector
- policyTypes
- Ingress
- egress

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              env: security
  egress:
    - to:
        - ipBlock:
            cidr: 8.8.8.8/32
```

1 - Pod Selector

Each NetworkPolicy includes a podSelector which selects the grouping of pods to which the policy applies.

The example network policy applies to all pods that has label of env=production

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
```

Point to Note

An empty podSelector selects all pods in the namespace.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector: {}
```

2 - Policy Types

Each NetworkPolicy includes a policyTypes list which may include either Ingress, Egress, or both.

The policyTypes field indicates whether or not the given policy applies to inbound traffic to selected pod, outbound traffic from selected pods, or both

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
```

3 - Ingress

Inside ingress, you can use various combinations of podSelector, nameSpace selector etc to define the rules.

Inbound traffic for Pods with label of env=production will be allowed from pods with label env=security.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            env: security
```

4 - Egress

Inside Egress rule, you can use various combinations of podSelector, namespaceselector, ipBlock etc to define the rules.

Allow Outbound Traffic only to IP address of 8.8.8.8 for Pods having label of env=production



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: demo-network-policy
spec:
  podSelector:
    matchLabels:
      env: production
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              env: security
  egress:
    - to:
        - ipBlock:
            cidr: 8.8.8.8/32
```

Role of from and to

from	<p>Used in an Ingress rule.</p> <p>Specifies the sources of incoming traffic allowed to the selected pods.</p> <p>Sources can be other pods, namespaces, or IP blocks.</p>
to	<p>Used in an Egress rule.</p> <p>Specifies the destinations of outgoing traffic allowed from the selected pods.</p> <p>Destinations can be other pods, namespaces, or IP blocks</p>

Practical - Network Policies

Example 1 - Block All Ingress and Egress

Since no specific rules are defined for ingress or egress, Kubernetes denies all traffic by default

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: production
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
```



Points to Note - podSelector

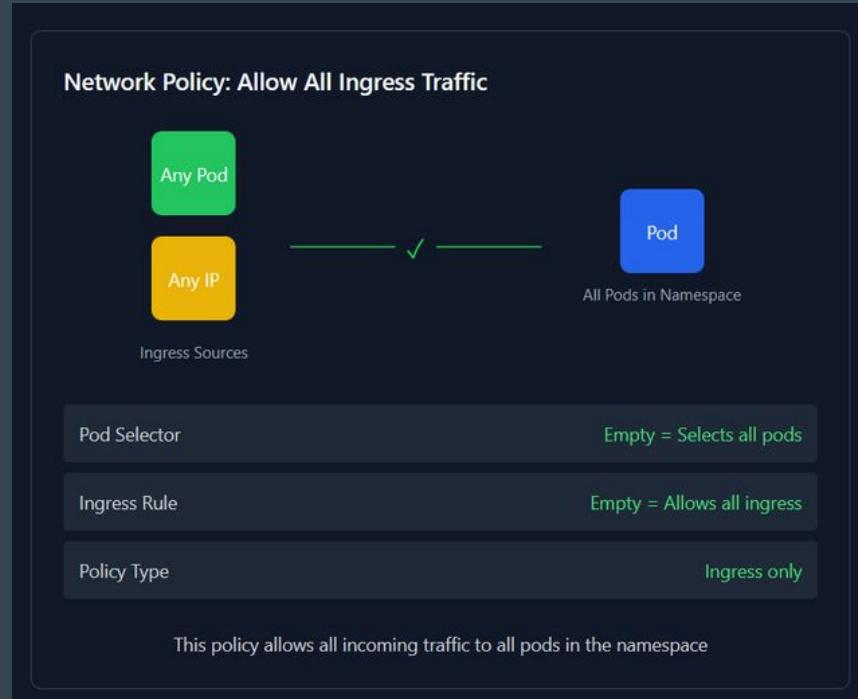
podSelector { }

This means the policy applies to all pods in the namespace because the selector is empty (matches all pods).

Example 2 - Allow Ingress Traffic

This policy allows all incoming traffic (ingress) to the selected pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```



Points to Note

ingress:

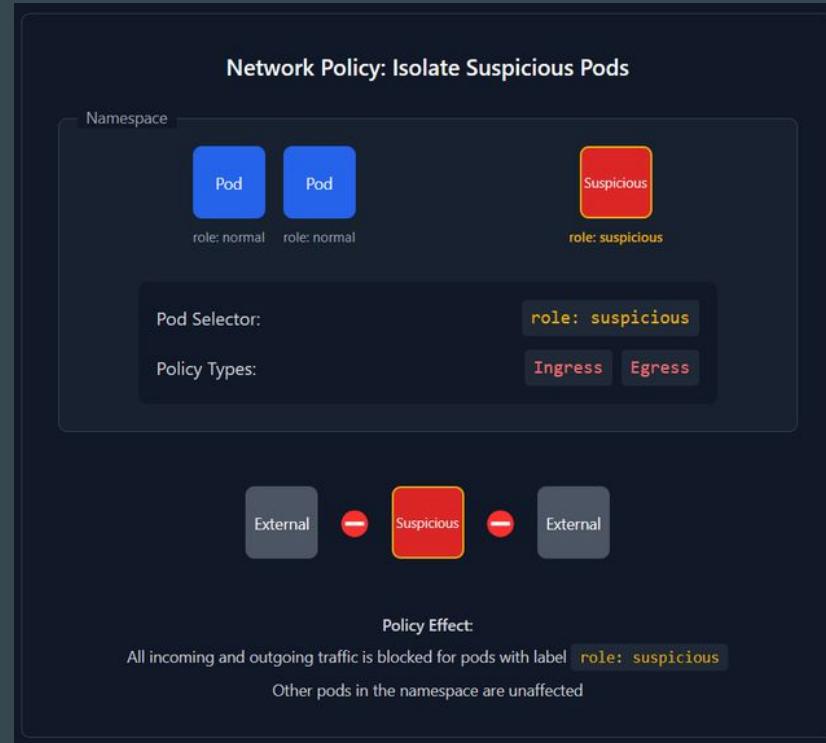
- { }

The empty {} means that there are no restrictions on the source of the traffic (any source is allowed).

Example 3 - Isolate Suspicious Pod

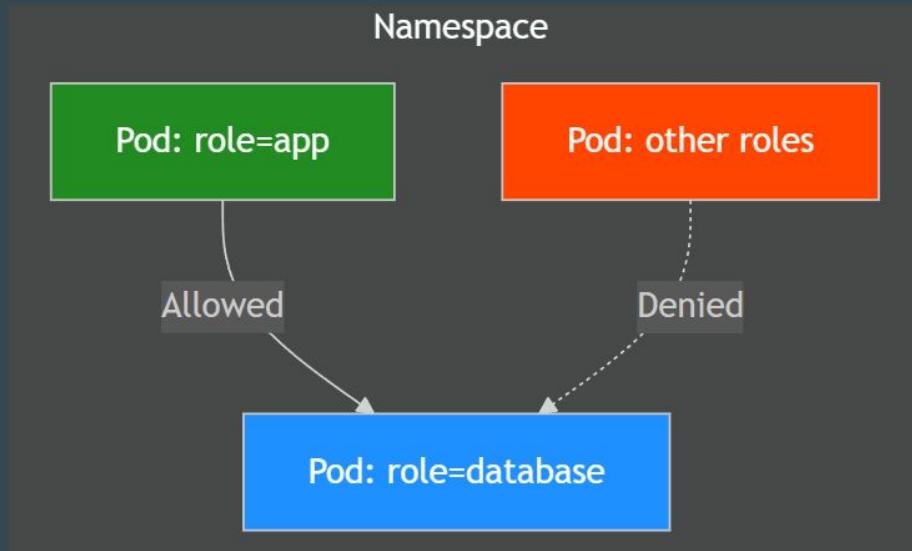
This policy blocks all ingress and egress access to pod with role=suspicious.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: suspicious-pod
spec:
  podSelector:
    matchLabels:
      role: suspicious
  policyTypes:
    - Ingress
    - Egress
```



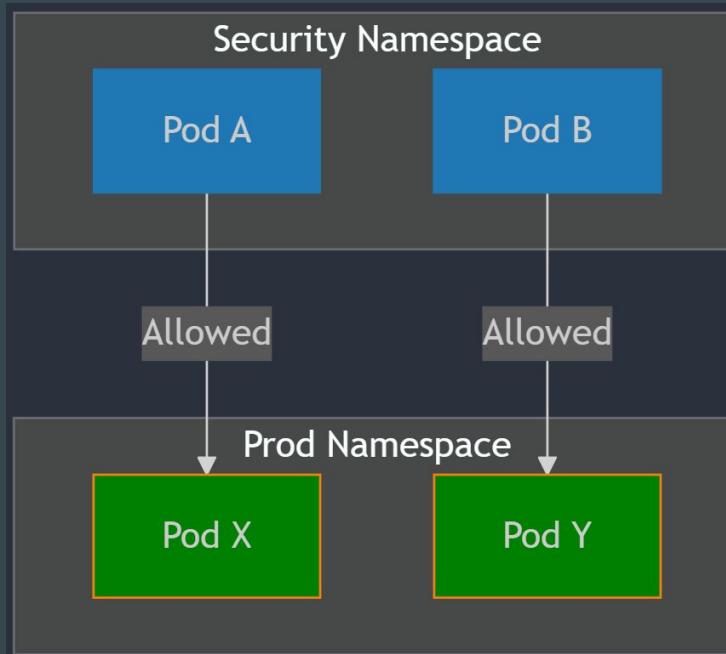
Example 4 - PodSelector

Allow pods with label of role=app to connect to pods with labels of role=database



Example 5 - Namespace Selector

Allow Pods from Security namespace to connect to Pods in Prod namespace.

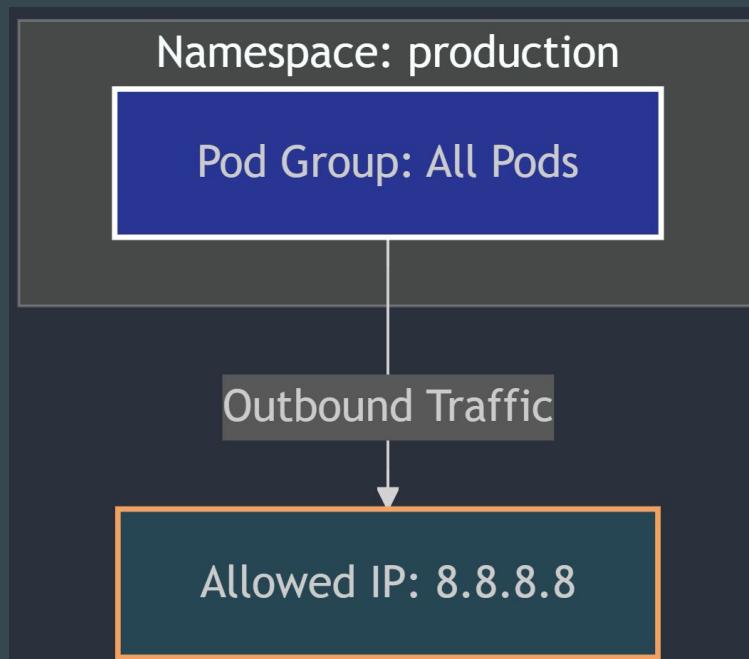


Example 5 - Reference Code

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: namespace-eselector
  namespace: production
spec:
  podSelector: {}
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: security
  policyTypes:
  - Ingress
```

Example 6 - ipBlock

Allow Pods from production namespace to connect to only 8.8.8.8 IP address outbound.



Example 6 - Reference Code

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: outbound-8888
spec:
  podSelector: {}
  egress:
  - to:
    - ipBlock:
        cidr: 8.8.8.8/32
  policyTypes:
  - Egress
```

Network Policies - Except, Port and Protocol

Except

The `except` field in a Kubernetes NetworkPolicy allows you to define exceptions to a broader rule.

Following policy allows ingress for cidr range of 172.17.0.0/16 except the range of 172.17.1.0/24

```
ingress:  
- from:  
  - ipBlock:  
    cidr: 172.17.0.0/16  
  except:  
  - 172.17.1.0/24
```

Reference Screenshot - Entire Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: except-ingress
spec:
  podSelector:
    matchLabels:
      role: database
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
  policyTypes:
    - Ingress
```

Ports and Protocol

When writing a NetworkPolicy, you can target a port or range of ports.

```
egress:  
  - to:  
    - ipBlock:  
        cidr: 10.0.0.0/24  
  ports:  
    - protocol: TCP  
      port: 32000  
      endPort: 32768
```

Reference Screenshot - Entire Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: multi-port-egress
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Egress
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 32000
      endPort: 32768
```

Kubeadm - Structure

Basic Structure

The `/etc/kubernetes` directory is critical, as it holds all the essential manifest files and certificates needed for the functioning of your Kubernetes cluster's components.

```
root@kubeadm:~# ls -l /etc/kubernetes/
total 44
-rw----- 1 root root 5658 Mar  7 17:11 admin.conf
-rw----- 1 root root 5682 Mar  7 17:11 controller-manager.conf
-rw----- 1 root root 1974 Mar  7 17:12 kubelet.conf
drwxrwxr-x 2 root root 4096 Mar 10 02:52 manifests
drwxr-xr-x 3 root root 4096 Mar  7 17:11 pki
-rw----- 1 root root 5630 Mar  7 17:11 scheduler.conf
-rw----- 1 root root 5682 Mar  7 17:11 super-admin.conf
```

Certificates

kubeadm generates certificate and private key pairs for different purposes.
Certificates are stored by default in [/etc/kubernetes/pki](#).

```
root@kubeadm:~# ls -l /etc/kubernetes/pki/
total 60
-rw-r--r-- 1 root root 1123 Mar  7 17:11 apiserver-etcd-client.crt
-rw----- 1 root root 1679 Mar  7 17:11 apiserver-etcd-client.key
-rw-r--r-- 1 root root 1176 Mar  7 17:11 apiserver-kubelet-client.crt
-rw----- 1 root root 1679 Mar  7 17:11 apiserver-kubelet-client.key
-rw-r--r-- 1 root root 1281 Mar  7 17:11 apiserver.crt
-rw----- 1 root root 1679 Mar  7 17:11 apiserver.key
-rw-r--r-- 1 root root 1107 Mar  7 17:11 ca.crt
-rw----- 1 root root 1675 Mar  7 17:11 ca.key
drwxr-xr-x 2 root root 4096 Mar  7 17:11 etcd
-rw-r--r-- 1 root root 1123 Mar  7 17:11 front-proxy-ca.crt
-rw----- 1 root root 1679 Mar  7 17:11 front-proxy-ca.key
-rw-r--r-- 1 root root 1119 Mar  7 17:11 front-proxy-client.crt
-rw----- 1 root root 1679 Mar  7 17:11 front-proxy-client.key
-rw----- 1 root root 1679 Mar  7 17:11 sa.key
-rw----- 1 root root  451 Mar  7 17:11 sa.pub
```

Certificates Mounted in Static Pods

The generated certificates are mounted in appropriate static pods using volume mounts.

```
volumes:
- hostPath:
    path: /etc/ssl/certs
    type: DirectoryOrCreate
    name: ca-certs
- hostPath:
    path: /etc/ca-certificates
    type: DirectoryOrCreate
    name: etc-ca-certificates
- hostPath:
    path: /etc/kubernetes/pki
    type: DirectoryOrCreate
    name: k8s-certs
```

Kubeconfig file - Admin

A kubeconfig file for kubeadm to use itself and the admin,
`/etc/kubernetes/admin.conf`

With this file, the admin has full control (root) over the cluster.

```
root@kubeadm:~# cat /etc/kubernetes/admin.conf
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tC
mNOQVFTEJRQXdGVEVUTUJFR0ExVUUKQXHNS2EzVmIaWEp1WlhSbGN6QWVGdzB5tLRBek
d0VkJBTVRDbQxWl1WeWjTvjBaWE13Z2dFaU1BMEDDU3FHU01iM0RRRUJBUVVbQTRJQkR
xd1zaZE82Y1R1emdpuTdvkTosXUxb0hzM1zSNXgKZGI0bFZsb1RieVhnakl4bCtkNwtF
Qgp4UkZaN15QX1NU3RNNUhPMzgrQ0RHWoL1dPeFlONFBCz52YkQ3RzQ4M3hIWi92V
H1DK2xNQTRRNX1sR3NjM0VqS1d0c3p0RnVtMm9WQ1RVb3BV0WMKME14aFQ4Q1k5NW54anl
dPM1djcuPhwQpyTVJBMG91czhuc1JuQnI4TkJxSfc2ME9RZy9uQwdNQkfBR2pXVEJYTUE
vTUIwR0ExVWREZ1FXQkJTR3plbjJ1R2grTkorRWdWUGxSN1oxNS9ZZjZUQVYKQmd0VkhSI
QTRJQkFRRFF0MStqMmRoWgpsWEdNWvhXOURMZTZRMjFxT1NNbkFtaSs5btJ2a2o3STZDV
HFUo3dxZE1ZS291WTBOSDRQN3czS0Qzc3QzdFNV3ps0DBPbDhPdEk0NElvS3ZUM2g3c3p
EwZjJSTTNMWjJvdGUyOXE2a2NzRk5teApOV1NsahYYzlyc1BGUXFLUWMrbj1jSDRPbEt
TRHRMUTVGdE15YVdJSUtHTnZCeXp1ZmtDNzNER2xkZDdpZ0c2NE53ZU5ydjZlcnnFFcTlql
LS0tLS0K
    server: https://167.71.227.219:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
    name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
```

Kubeconfig file - Kubelet

A **kubeconfig** file for **kubelet** to use, /etc/kubernetes/kubelet.conf

This certificate have the following:

- CN system:node:<hostname-lowercased>
- Be in the system:nodes organization

```
server: https://167.71.227.219:6443
name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: system:node:kubeadm
    name: system:node:kubeadm@kubernetes
current-context: system:node:kubeadm@kubernetes
kind: Config
preferences: {}
users:
- name: system:node:kubeadm
  user:
    client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem
    client-key: /var/lib/kubelet/pki/kubelet-client-current.pem
```

Kubeconfig file - Controller Manager

A **kubeconfig** file for controller-manager is /etc/kubernetes/controller-manager.conf

This file should have the CN system:kube-controller-manager

```
server: https://167.71.227.219:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: system:kube-controller-manager
    name: system:kube-controller-manager@kubernetes
current-context: system:kube-controller-manager@kubernetes
kind: Config
preferences: {}
users:
- name: system:kube-controller-manager
  user:
```

Kubeconfig file - Scheduler

A **kubeconfig** file for scheduler, /etc/kubernetes/scheduler.conf

This file should have the CN system:kube-scheduler

```
server: https://167.71.227.219:6443
name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: system:kube-scheduler
    name: system:kube-scheduler@kubernetes
current-context: system:kube-scheduler@kubernetes
kind: Config
preferences: {}
users:
- name: system:kube-scheduler
```

Static Pod manifests for Control Plane

Static pod manifest files for control-plane components are primarily defined in path of /etc/kubernetes/manifests

```
root@kubeadm:~# ls -l /etc/kubernetes/manifests/
total 16
-rw----- 1 root root 2549 Mar  7 17:11 etcd.yaml
-rw----- 1 root root 3892 Mar  7 17:11 kube-apiserver.yaml
-rw----- 1 root root 3394 Mar  7 17:11 kube-controller-manager.yaml
-rw----- 1 root root 1656 Mar  7 17:11 kube-scheduler.yaml
```

Properties for Control Plane Components - 1

All **static** Pods are deployed on **kube-system** namespace.

root@kubeadm:~# kubectl get pods -n kube-system				
NAME	READY	STATUS	RESTARTS	AGE
coredns-668d6bf9bc-cjhls	1/1	Running	0	2d10h
coredns-668d6bf9bc-xnsqn	1/1	Running	0	2d10h
etcd-kubeadm	1/1	Running	0	2d10h
kube-apiserver-kubeadm	1/1	Running	0	2d10h
kube-controller-manager-kubeadm	1/1	Running	0	2d10h
kube-proxy-whxsh	1/1	Running	0	2d10h
kube-scheduler-kubeadm	1/1	Running	0	2d10h

Properties for Control Plane Components - 2

All static Pods get tier:control-plane and component:{component-name} labels.

NAME	READY	STATUS	RESTARTS	AGE	LABELS
coredns-668d6bf9bc-cjhls	1/1	Running	0	2d10h	k8s-app=kube-dns,pod-template-hash=668d6bf9bc
coredns-668d6bf9bc-xnsqn	1/1	Running	0	2d10h	k8s-app=kube-dns,pod-template-hash=668d6bf9bc
etcd-kubeadm	1/1	Running	0	2d10h	component=etcd,tier=control-plane
kube-apiserver-kubeadm	1/1	Running	0	2d10h	component=kube-apiserver,tier=control-plane
kube-controller-manager-kubeadm	1/1	Running	0	2d10h	component=kube-controller-manager,tier=control-plane
kube-proxy-whxsh ,pod-template-generation=1	1/1	Running	0	2d10h	controller-revision-hash=7bb84c4984,k8s-app=kube-proxy
kube-scheduler-kubeadm	1/1	Running	0	2d10h	component=kube-scheduler,tier=control-plane

Kubelet Configuration

Kubelet is configured on the host system and is managed using systemd.

Path to kubelet config file: /var/lib/kubelet

```
root@kubeadm:~# ls -l /var/lib/kubelet/
total 48
drwx----- 2 root root 4096 Mar  7 17:11 checkpoints
-rw-r--r--  1 root root 1123 Mar  7 17:11 config.yaml
-rw-r--r--  1 root root 1123 Mar 10 12:51 config.yaml.bak
-rw-------  1 root root   62 Mar  7 17:11 cpu_manager_state
drwxr-xr-x  2 root root 4096 Mar  7 17:28 device-plugins
-rw-r--r--  1 root root  150 Mar  7 17:11 kubeadm-flags.env
-rw-------  1 root root   61 Mar  7 17:11 memory_manager_state
drwxr-xr-x  2 root root 4096 Mar  7 17:11 pki
drwxr-x---  3 root root 4096 Mar  7 17:12 plugins
drwxr-x---  2 root root 4096 Mar  7 17:13 plugins_registry
drwxr-x---  2 root root 4096 Mar  7 17:28 pod-resources
drwxr-x--- 18 root root 4096 Mar 10 12:55 pods
```

Mark Control Plane Node

As soon as the control plane is available, kubeadm executes following actions:

1. Label the control-plane node with `node-role.kubernetes.io/control-plane=""`
2. Taints the node with `node-role.kubernetes.io/control-plane:NoSchedule`

Kubeadm - Troubleshooting

Setting the Base

You should be familiar with **troubleshooting kubeadm** based clusters.



Kubelet Logs

You can check the **kubelet logs** using journalctl

```
root@kubeadm:~# journalctl -u kubelet -f
Mar 10 04:09:40 kubeadm kubelet[13096]: E0310 04:09:40.876232    13096 controller.go:145] "Failed to ensure
  retry" err="Get \"https://167.71.227.219:6443/apis/coordination.k8s.io/v1/namespaces/kube-node-lease/leas
10s\": dial tcp 167.71.227.219:6443: connect: connection refused" interval="7s"
Mar 10 04:09:42 kubeadm kubelet[13096]: I0310 04:09:42.533857    13096 scope.go:117] "RemoveContainer" cont
02be267f0c03fb9c2d4e17815f7e4d7ad15ce117c060483d2a854f"
Mar 10 04:09:42 kubeadm kubelet[13096]: E0310 04:09:42.534063    13096 pod_workers.go:1301] "Error syncing
  \"failed to \\"StartContainer\\" for \\"tigera-operator\\" with CrashLoopBackOff: \\"back-off 5m0s restarting fa
ra-operator pod=tigera-operator-7d68577dc5-p76xl_tigera-operator(a95fcf02-eb0c-4de7-a56a-3560a148faae)\\""
r/tigera-operator-7d68577dc5-p76xl" podUID="a95fcf02-eb0c-4de7-a56a-3560a148faae"
Mar 10 04:09:42 kubeadm kubelet[13096]: I0310 04:09:42.534406    13096 status_manager.go:890] "Failed to ge
odUID="ec9ae59b2a9e7b371cacc628b70bd013" pod="kube-system/kube-controller-manager-kubeadm" err="Get \"http
6443/api/v1/namespaces/kube-system/pods/kube-controller-manager-kubeadm\": dial tcp 167.71.227.219:6443: c
refused"
```

Pod Specific Logs

You can check Pod specific logs under `/var/log` directory.

```
root@kubeadm:~# ls /var/log/pods
calico-apiserver_calico-apiserver-68f8f9d9b8-ttq9z_1633276f-11ae-4aeb-bf9f-1e731b516b68
calico-apiserver_calico-apiserver-68f8f9d9b8-zc7kn_c9580ad9-9cb3-4251-9ff6-3deb2f6fc897
calico-system_calico-kube-controllers-8d756595-zxzxw_cf52a707-183c-484c-80ac-bde09be27901
calico-system_calico-node-89h8v_d0defed4-f210-4311-af49-98ed933f7087
calico-system_calico-typha-787b9c4566-pfc5v_53ee8670-b69b-4b8e-820c-129371d78690
calico-system_csi-node-driver-zr542_56c77d71-62fe-4881-9282-5cd7042cac55
default_nginx-pod_06e8ed14-b7a3-4dde-88f4-0aa4e2218189
default_privileged-pod_2bb337ea-56a9-48b2-98b9-4955de35e9cf
kube-system_coredns-668d6bf9bc-cjhls_a4723c92-5696-490c-81fb-29c5a5cc8c89
kube-system_coredns-668d6bf9bc-xnsqn_2783dd9f-30f9-441a-860a-445e20519eae
kube-system_etcd-kubeadm_ed465a6538f60c0c332edbc0fa5764e4
kube-system_kube-apiserver-kubeadm_9ced367dd009364d5f7175f54768a567
kube-system_kube-controller-manager-kubeadm_ec9ae59b2a9e7b371cacc628b70bd013
kube-system_kube-proxy-whxsh_3722719e-90f0-48f0-8bb9-57084d8ed27c
kube-system_kube-scheduler-kubeadm_2e41779e83593bc084841f6def231e10
tigera-operator_tigera-operator-7d68577dc5-p76x1_a95fcf02-eb0c-4de7-a56a-3560a148faae
```

Pod Specific Logs - Better Way

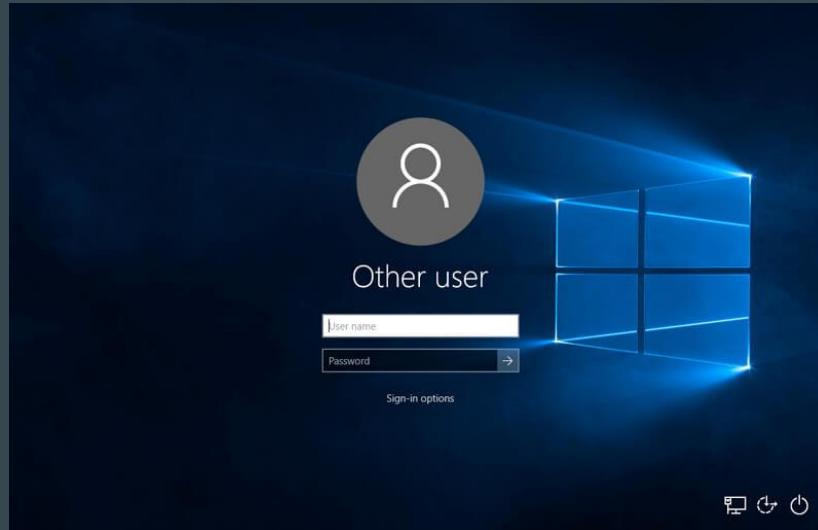
You can directly check the `/var/log/containers` that has symlink to the latest log file for Pods.

```
root@kubeadm:~# ls -l /var/log/containers/
total 88
lrwxrwxrwx 1 root root 124 Mar  7 17:13 calico-apiserver-68f8f9d9b8-ttq9z_calico-apiserver_calico-apiserver-24d7d49b54a12f298adbe95f2b100fb4f4a9c0f2d2f5d44.log -> /var/log/pods/calico-apiserver_calico-apiserver-68f8-76f-11ae-4aeb-bf9f-1e731b516b68/calico-apiserver/0.log
lrwxrwxrwx 1 root root 124 Mar  7 17:13 calico-apiserver-68f8f9d9b8-zc7kn_calico-apiserver_calico-apiserver-affd0ae69afdf3b79082f6dfb91a217654564d44e50cd73b.log -> /var/log/pods/calico-apiserver_calico-apiserver-68f8-ad9-9cb3-4251-9ff6-3deb2f6fc897/calico-apiserver/0.log
lrwxrwxrwx 1 root root 133 Mar 10 04:15 calico-kube-controllers-8d756595-zxzxw_calico-system_calico-kube-controllers-756462232e5a675c390177d599e4f18845b4de70a8250172768f07ac.log -> /var/log/pods/calico-system_calico-kube-controllers-zxzxw_cf52a707-183c-484c-80ac-bde09be27901/calico-kube-controllers/4.log
lrwxrwxrwx 1 root root 133 Mar 10 04:17 calico-kube-controllers-8d756595-zxzxw_calico-system_calico-kube-controllers-8b97741f74b7c6569466af5a3d723b04c5a5bc0deca104b4b1549bd3.log -> /var/log/pods/calico-system_calico-kube-controllers-zxzxw_cf52a707-183c-484c-80ac-bde09be27901/calico-kube-controllers/5.log
lrwxrwxrwx 1 root root 100 Mar  7 17:13 calico-node-89h8v_calico-system_calico-node-10bc1a54124853a76c2ea1f2cea4352c4d8737e1f3f0212.log -> /var/log/pods/calico-system_calico-node-89h8v_d0defed4-f210-4311-af49-98ed931/0.log
lrwxrwxrwx 1 root root 103 Mar  7 17:12 calico-node-89h8v_calico-system_flexvol-driver-f6ecb3d4b0bec5064177a454343f4d0926e3574da1ac90.log -> /var/log/pods/calico-system_calico-node-89h8v_d0defed4-f210-4311-af49-98ed931/driver/0.log
lrwxrwxrwx 1 root root 100 Mar  7 17:12 calico-node-89h8v_calico-system_install-cni-e6794f04433ec68777322f0:d83982669d9c014fb98d265.log -> /var/log/pods/calico-system_calico-node-89h8v_d0defed4-f210-4311-af49-98ed931/0.log
```

Authentication in Kubernetes

Basics of Authentication

Authentication is the process of verifying a user's identity before granting them access to a system or resource



Accessing Resources in Kubernetes

To access resources in Kubernetes cluster, we have to authenticate first.



Analogy of AWS

In AWS, you can authenticate using multiple set of methods.

1. Username and Passwords.
2. Access Key and Secret Keys

```
C:\>aws ec2 describe-security-groups
{
    "SecurityGroups": [
        {
            "Description": "default VPC security group",
            "GroupName": "default",
            "IpPermissions": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "UserIdGroupPairs": [
                        {
                            "GroupId": "sg-01aa5110c343f107d",
                            "UserId": "430118823531"
                        }
                    ]
                },
            ]
        }
    ]
}
```



Point to Note - Kubernetes

Kubernetes **does not manage the user accounts natively**.

Normal users cannot be added to a cluster through an API call



`kubectl create user alice`



Authentication in Kubernetes

Kubernetes supports several authentication methods such as:

Client Certificates, Static Token Authentication, Service Account Tokens etc



Example 1 - Static Token File

The API server reads bearer tokens from a file provided.

The token file is a csv file with a minimum of 3 columns: token, user name, user uid

```
root@control-plane:~# cat /root/token.csv  
Dem0Passw0rd#,bob,01,admins
```



```
[Service]  
ExecStart=/usr/local/bin/kube-apiserver --advertise-address=165.22.212.16 --etcd-cafile=/root/certificates/ca.crt --etcd-cert  
file=/root/certificates/etcd.crt --etcd-keyfile=/root/certificates/etcd.key --etcd-servers=https://127.0.0.1:2379 --service-a  
ccount-key-file=/root/certificates/service-account.crt --service-cluster-ip-range=10.0.0.0/24 --service-account-signing-key-f  
ile=/root/certificates/service-account.key --service-account-issuer=https://127.0.0.1:6443 --token-auth-file /root/token.csv
```

Example 2 - X509 Certificates

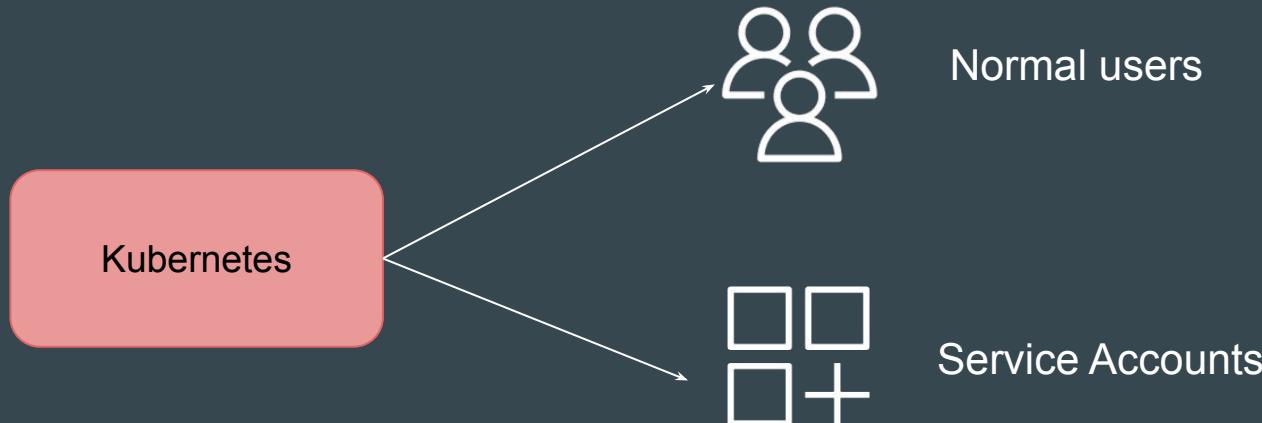
Uses the client certificates for authentication.

```
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR
    QVFTEJRQXdGVEVUTUJFR0ExVUUJKQXhNS2EzVm1aWEp1WlhSbGN6QVGdzB5TlRBeE1qVxdN
    kJBb1RGbXQxWW1WaFpHMDZZMngxYzNSbGNpMWhaRzFwYm5NeEdUQvHcZ05WQkFNVEVhdDFZb
    FRVUFBNelCRHdBd2dnRUtBb01CQVFER01aWjkKNnQ0ZC90NGhUNWpxb2p6SjRBT2JObnRTQe
    rTGtXaXoveCszTwdyREJwNGNheDjqS0ZTU0dNbU5udUZnT1NMR21GaS9yK3IyR2MyUUJaN3N
    RGtOQWVzd1BIVUVQcWc1RFQ5MU50eXpiUhdjN0UwdkEwODgKQuDyV3FKMWhTN291VmNhTmE0
    2srLzFydgpubGIwM1lrT1EwWUsvMU9jSEI3UEZQZ21Wb1AvWVeRk2xqNEgyWWpzUkE4UmFTT
    FHalZqQlVNQTRHQTfVZER3RUIvd1FFQXdJRm9EQVRCZ05WSFNRUREQUsKQmdnckJnRUZCUw
    O1NMQkNtVktP0xadwpGWUdtWxc1aGRRWkxNQTBHQ1NxR1NJYjNEUUVQCQ3dVQUE0SUJBUE
    TDIxdXN0UWjtZ3pubUN6cndyQXpwdHZwLzFORUY0MkpTVjBpem8veW1JWFZEVmJJMW8KSVI0
    nVabEdYZDYxbUNZTkwyckdpE9BZgp0L0R3OUZVcVdtcnVsaUp1cEJOMHNBeVZ4dUUxSDNYL
    1xMDNIUjdTUUY2NGV5SHB4SUt4QnoyNWJ3cVhETytEdnJjR3piUE5EcW9WUGFidWJZQzBKdU
    vNktzd15zOULqMEDBcFFER1YzUGdoejU5Ci0tLS0tRU5EIENFU1RJRk1DQVRFLS0tLS0K
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkfFURSLRVktLS0tLQpNSU1Fb2d
    aW0rdnF3RXpvVS9Sa1J5TFNwCnJZUVVicCs1cCtJdk1Qb21RZDRQTXBDNUZvcy84ZnR6SUt3
```

Categories of Users

Kubernetes Clusters have **two categories of users**:

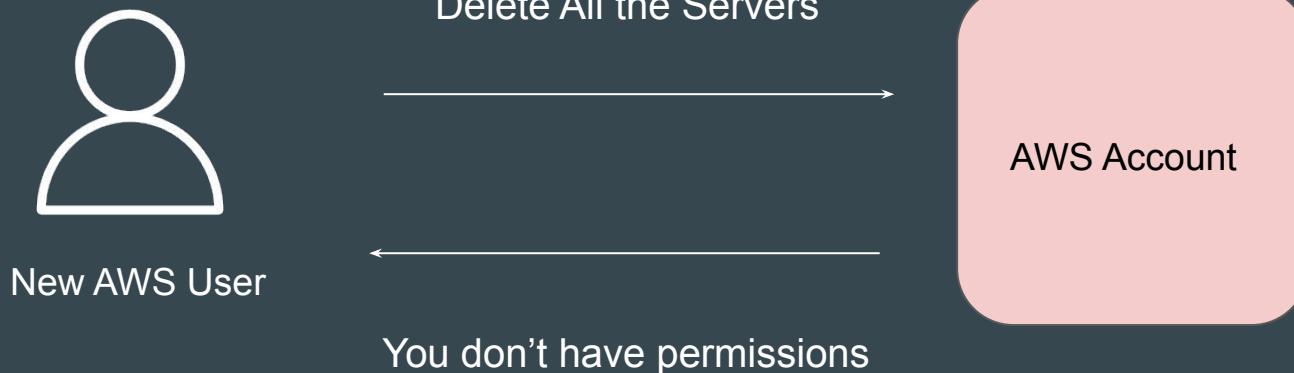
1. Normal Users (for humans)
2. Service Accounts (for apps)



Authorization

Basics of Authorization

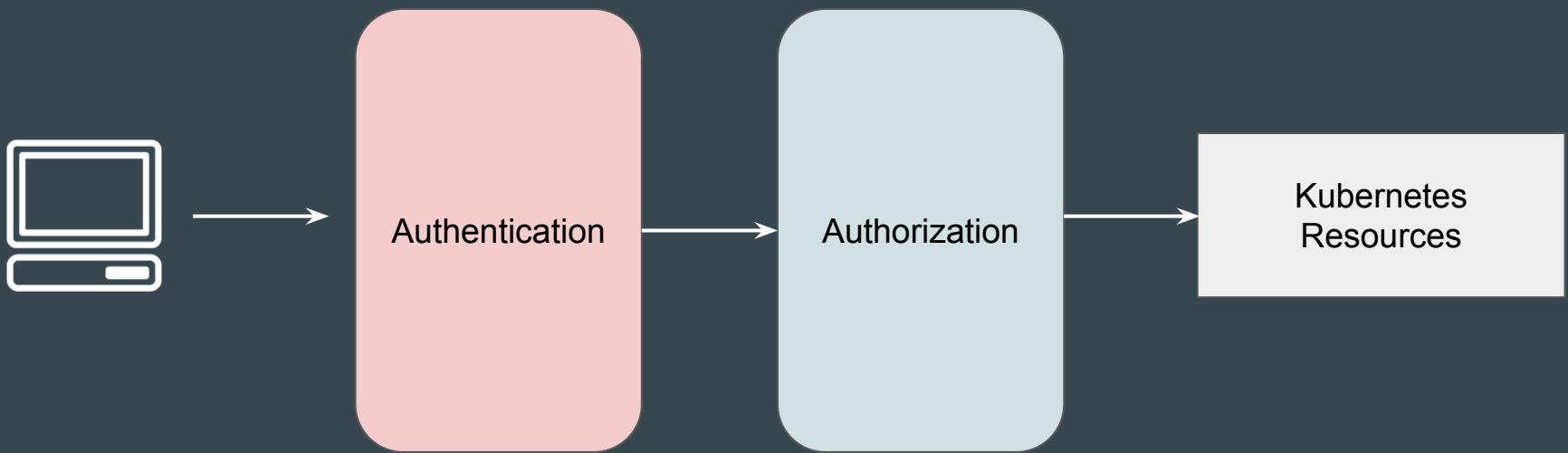
Authorization is the process of determining what an authenticated user or entity is allowed to do



Authorization in Kubernetes

Kubernetes authorization takes place following authentication.

Usually, a client making a request must be authenticated (logged in) before its request can be allowed.



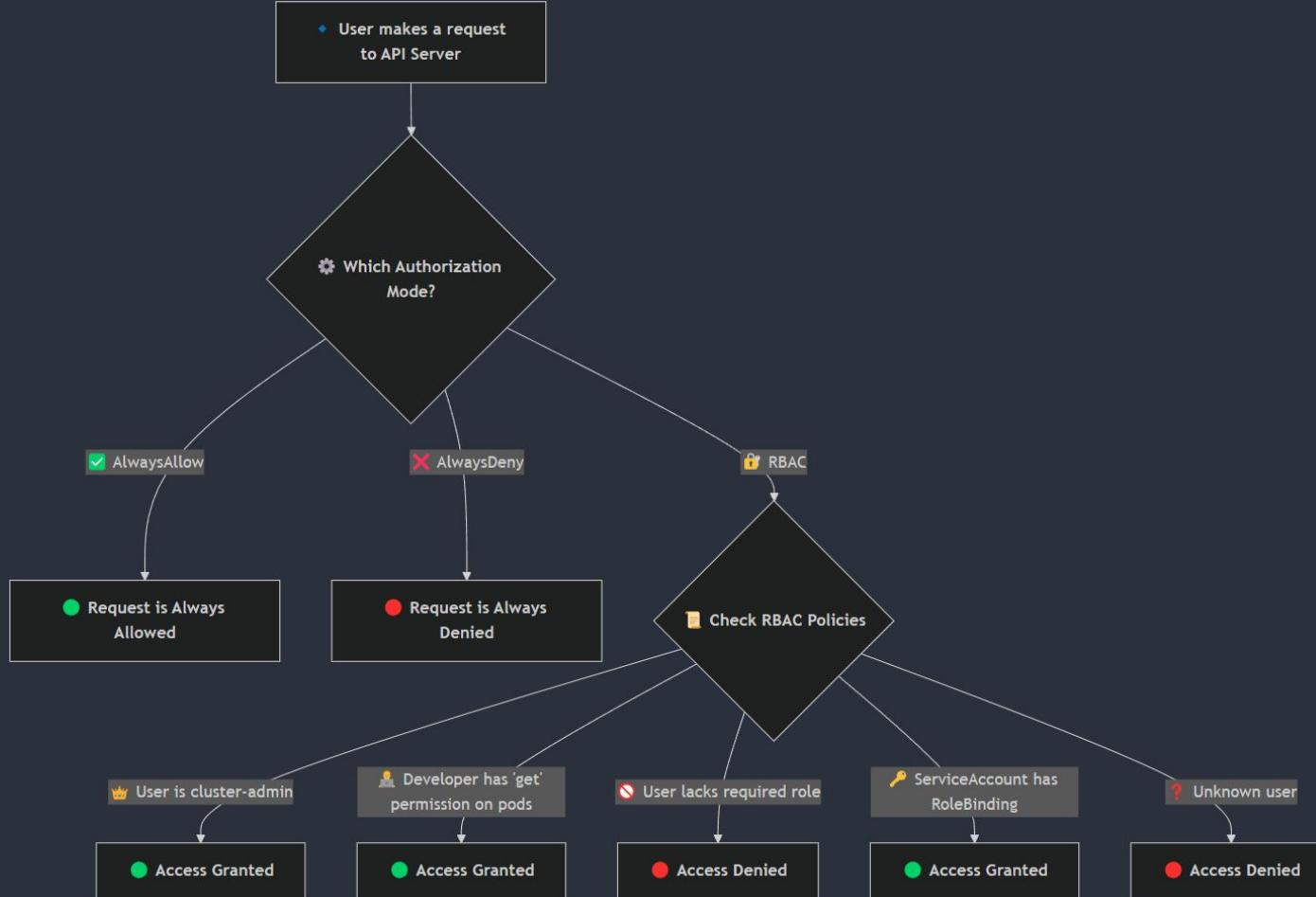
Authorization Modes

The Kubernetes API server may authorize a request using one of several authorization modes. Some of these include:

Authorization Mode	Description
AlwaysAllow	<p>This mode allows all requests, which brings security risks.</p> <p>Use this authorization mode only for testing.</p>
AlwaysDeny	<p>This mode blocks all requests.</p> <p>Use this authorization mode only for testing.</p>
RBAC	<p>Defines set of permissions based on which access is granted.</p> <p>Recommended for Production.</p>

Point to Note

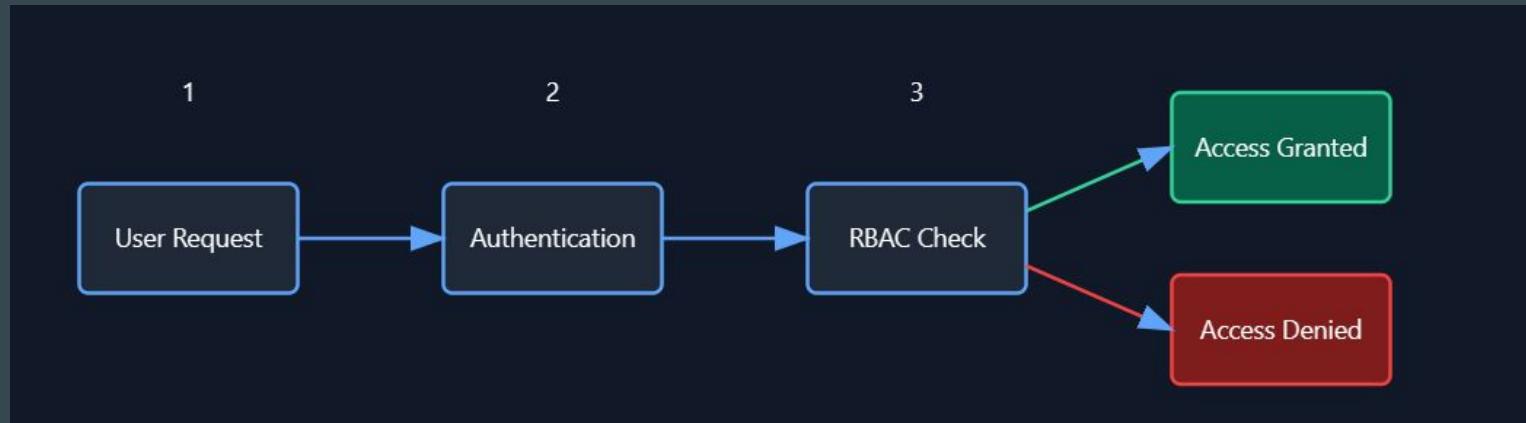
In Kubernetes, if the authorization mode is not explicitly defined in the API server configuration, the default mode used is AlwaysAllow.



Role-Based Access Control (RBAC)

Setting the Base

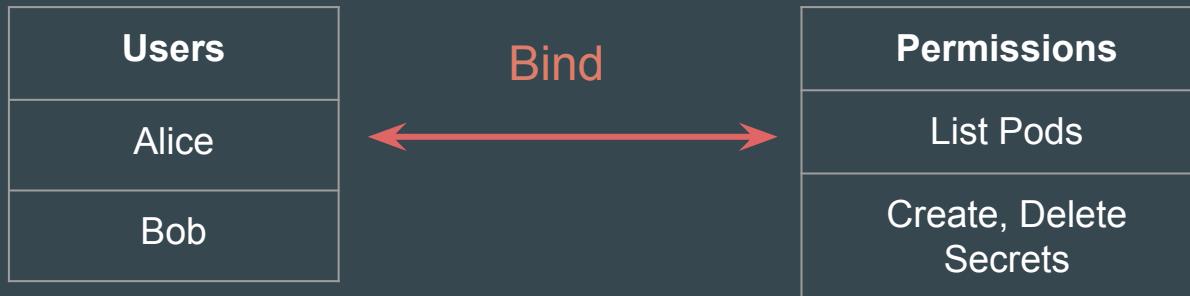
RBAC allows us to control what actions users and service accounts can perform on resources within your cluster.



Basic Workflow

In the below diagram, we have a list of users in Table 1 and list of permissions in Table 2.

We have to bind these together for users to get the defined permissions.

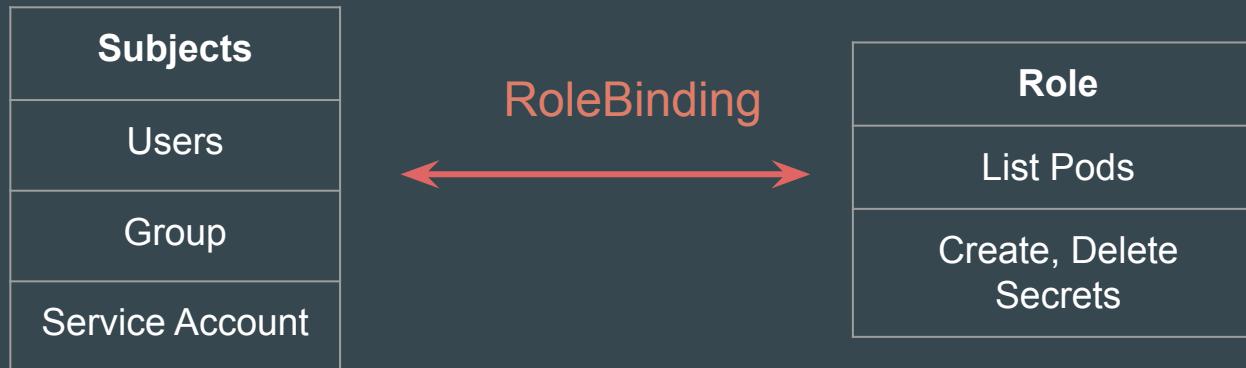


3 Important Concepts

Role defines a set of permissions.

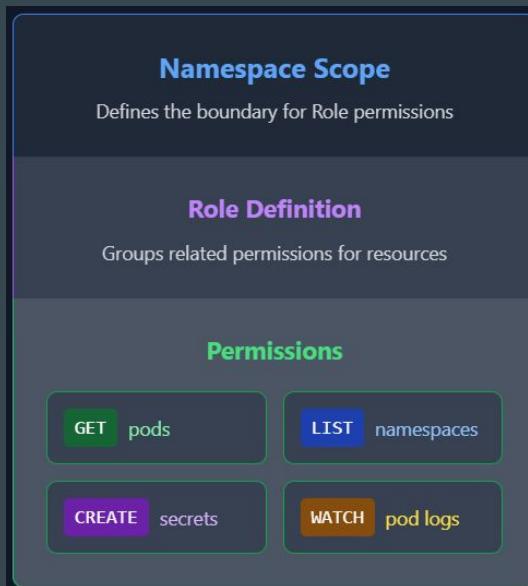
Subjects can be user, groups, service account.

RoleBinding ties the permission defined in the role to subjects like Users.



Introducing Roles

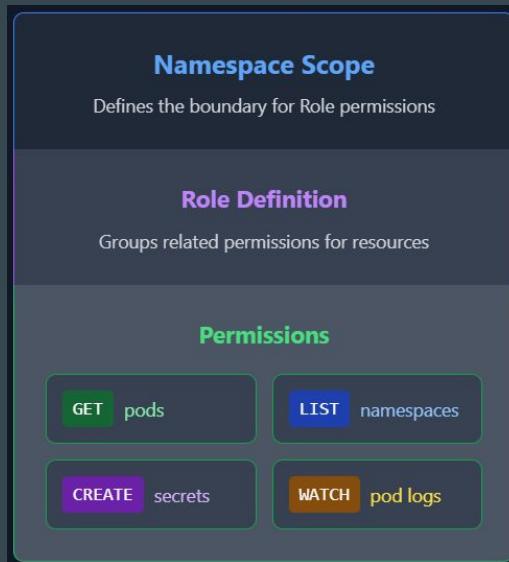
A Role always sets permissions within a particular namespace.



Introducing RoleBinding

RoleBinding associates a Role with a user, group, or service account within a specific namespace.

It grants the defined permissions to the subjects in that namespace.



RoleBinding



ClusterRole and ClusterRoleBinding

Similar to Role and RoleBinding, but the main difference is that the permissions granted by a ClusterRole apply across all namespaces in the cluster. ClusterRoleBinding connects ClusterRole to Subjects.



ClusterRoleBinding



Practical - Role and RoleBinding

Basic Structure of Role Manifest

The following image represents the basic structure of the first part of a Role manifest file.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-read-only
  namespace: default
```

Defining Rules in Role Manifest

The **rules** field is a list of policies that define the permissions granted by the Role.

Each rule specifies which actions (verbs) are allowed on which resources (API objects).

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-read-only
  namespace: default
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["list"]
```

1 - API Groups

apiGroups specify which API group the rule applies to.

Kubernetes APIs are categorized into different API groups.

API Groups	Description
"" (empty string)	Refers to the core API group (e.g., pods, services, configmaps etc).
apps	Refers to the apps API group (e.g., deployments, daemonsets,replicasets)
batch	Includes Jobs, CronJobs.
networking.k8s.io	Handles Ingress and Network Policies.

2 - Resources

This field specifies which Kubernetes resources the rule applies to.

These resources belong to the specified API group.

C:\>kubectl api-resources --api-group="apps"				
NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
controllerrevisions		apps/v1	true	ControllerRevision
daemonsets	ds	apps/v1	true	DaemonSet
deployments	deploy	apps/v1	true	Deployment
replicasets	rs	apps/v1	true	ReplicaSet
statefulsets	sts	apps/v1	true	StatefulSet

3 - Verbs

Verb specifies what actions (operations) are allowed on the specified resources.

Common Verbs	Description
get	Read a specific resource.
list	List all resources of that type.
create	Create a new resource.
delete	Modify an existing resource.
update	Remove a resource.
watch	Observe changes to a resource.

Structure - RoleBinding

While defining RoleBinding, we have to define subjects and Role Reference.



Generate Role Manifest File

```
C:\>kubectl create role pod-reader --verb=list --resource=pods --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: pod-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
```

Generate Role Binding Manifest File

```
C:\>kubectl create rolebinding pod-reader --role=pod-reader --user=bob --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: pod-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-reader
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob
```

Practical - ClusterRole and ClusterRoleBinding

Structure of ClusterRole Manifest

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-read-only
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["list"]
```

Structure of ClusterRoleBinding Manifest

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pod-rolebinding
  namespace: default
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-read-only
  apiGroup: rbac.authorization.k8s.io
```

Generate ClusterRole Manifest File

```
C:\>kubectl create clusterrole pod-read-only --verb=list --resource=pods --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: pod-read-only
rules:
- apiGroups:
  - ""
    resources:
    - pods
    verbs:
    - list
```

Generate ClusterRoleBinding Manifest File

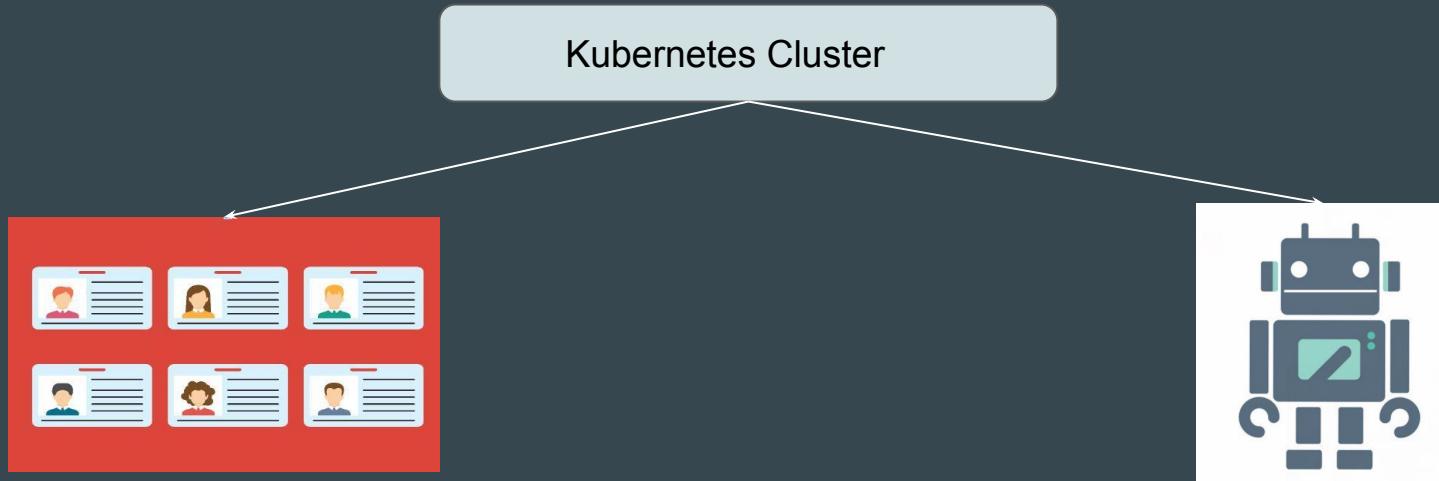
```
C:\>kubectl create clusterrolebinding pod-read --clusterrole=pod-read-only --user=bob --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: pod-read
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pod-read-only
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob
```

Service Accounts

Understanding the basics

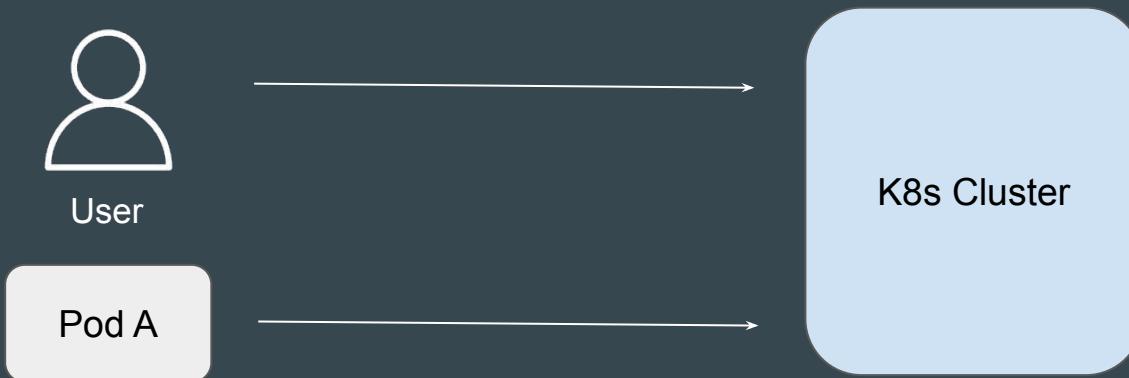
Kubernetes Clusters have two categories of accounts:

- User Accounts (For Humans).
- Service Accounts (For Applications)



Importance of Credentials

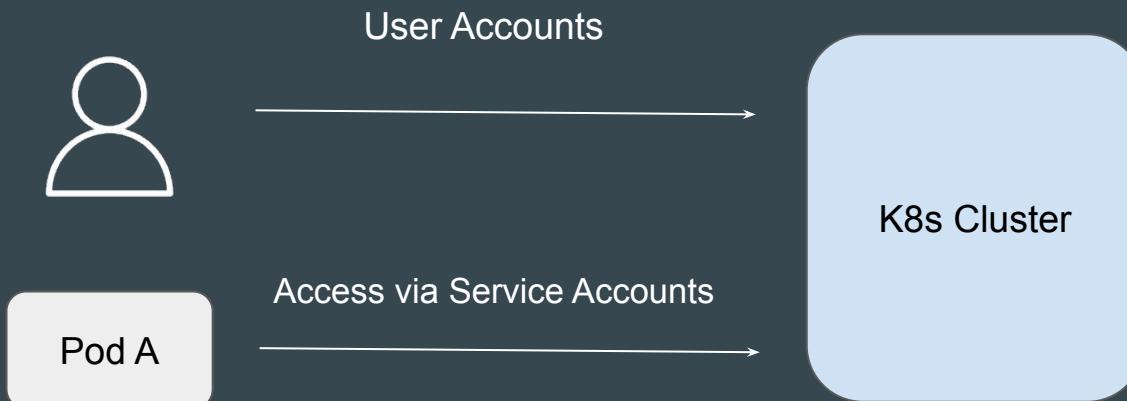
To connect to Kubernetes cluster, an entity needs to have some kind of authentication credentials.



Different Type of Credentials

Humans will use **User Accounts** to connect to Cluster.

Pods / Applications will use **Service Accounts** to connect to Cluster.



Service Accounts in K8s Cluster

Various different components of Kubernetes uses service accounts to communicate with the cluster

```
C:\Users\zealv>kubectl get serviceaccounts --all-namespaces
NAMESPACE      NAME          SECRETS   AGE
default        default       0         5m57s
kube-node-lease default       0         5m57s
kube-public    default       0         5m57s
kube-system   attachdetach-controller 0         5m57s
kube-system   certificate-controller 0         6m1s
kube-system   cilium        0         4m22s
kube-system   cilium-operator 0         4m22s
kube-system   cloud-controller-manager 0         5m54s
kube-system   cluster-autoscaler 0         5m10s
kube-system   clusterrole-aggregation-controller 0         5m57s
kube-system   coredns        0         102s
```

Service Accounts and Pods

Let's assume that a Service Account is associated with Pod A.

Pod A can use the token associated with the service account to perform some actions on Kubernetes cluster.



Service Accounts - Points to Note

Default Service Account

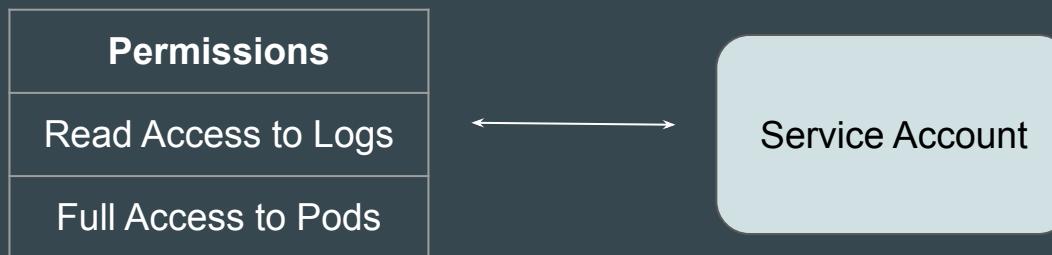
When you create a cluster, Kubernetes automatically creates a ServiceAccount object named **default** for every namespace in your cluster.

```
C:\Users\zealv>kubectl get serviceaccount
NAME      SECRETS   AGE
default    0          23m
```

Service Account and Permissions

Each Service Account in Kubernetes can be associated with certain permissions.

When Pod uses the service account, it can inherit the permissions.



Point to Note

The default service accounts in each namespace get no permissions by default other than the default API discovery permissions that Kubernetes grants to all authenticated principals if role-based access control (RBAC) is enabled

Assigning Pods to Service Accounts

If you deploy a Pod in a namespace, and you don't manually assign a ServiceAccount to the Pod, Kubernetes **assigns the default ServiceAccount** for that namespace to the Pod



Accessing Service Account Token

Service Account Token gets mounted inside the Pod inside the /var/run directory and can easily be accessed using cat command.

```
root@nginx:/# cat /var/run/secrets/kubernetes.io/serviceaccount/token
eyJhbGciOiJSUzIiNlIsImtpZCI6IiJLZWlNejJxdTh3NlY2SjhRZHvkSEZfMWxKazFoZmttb3JLNnNTNzTdxMifQ.eyJhdWQiolsic3lzdGVt0mtvbm5lY3Rpdm
l0eS1zZXJ2ZXIiXSwiZXhwIjoxNzI3NzU50TY2LCJpYXQiOjE2OTYyMjM5NjYsImlzcyI6Imh0dHBz0i8va3ViZXJuZXRlcyc5kZWZhdWx0LnN2Yy5jbHVzdGVyLmx
vY2FsIiwiia3ViZXJuZXRlcyc5pbI6eyJuYW1lc3BhY2Ui0iJkZWZhdWx0IiwickG9kIjp7Im5hbWUiOjJuZ2lueCIsInVpZCI6ImU3ZjUyYWY4LWM5MGUtNDY5Zi1i
MTg3LTc3MjliMmNmE5MyJ9LCJzZXJ2aWN1YWNjb3VudCI6eyJuYW1lIjoIZGVmYXVsdcIsInVpZCI6ImZlOTUzMTQ0LTThjNDYtNDF1My1iODFjLTMsMThkNTNhM
zAyOCJ9LCJ3YXJuYWZ0ZXIIoje2OTYyMjc1NzN9LCJuYmYiOjE2OTYyMjM5NjYsInN1Yi6In5c3R1bTpzZXJ2aWN1YWNjb3VudDpkZWZhdWx00mR1ZmF1bHQifQ
.i5jS2uEbNgj_1GA6LQh2YxQKEJsNjZbvoJh-Qjf-vH_f10w0KTvhmLPCL1UxCLfoI1NejT07Agckj3SFXYjEqmlvcbEYpLXt8eUjrHC8s6Ra105XGnpbt5uUy3GU
BYP4HnRE970zzU3HyU2gM14Kd8d8a9iiHb7yov82ph6moOPuuxCxBowNo5edWMWnNDWLKLNOFX168oxAmQo8ZQI5k37INIE1oN5N2bzp7Y40Gv3CURK1X904B0YuT
UN8gSYZsHaQaVq8FT-Q_xGfGQbvjqUzi0rRaKNc9QJ0BiIE3CKQN7PL6C0Lxyt_9LieJV438xPgwer2V_aNeHBWgU99oAroot@nginx:/#
```

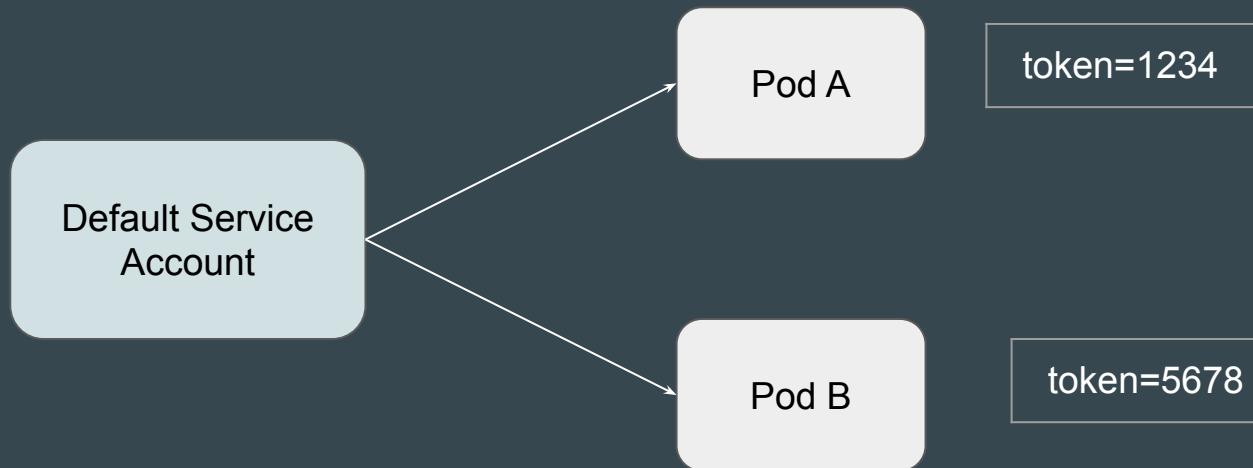
Connecting to K8s using Token

Using the Service Account Token, you can connect to the Kubernetes Cluster to perform operations.

```
root@nginx:/var/run/secrets/kubernetes.io/serviceaccount# curl -k -H "Authorization: Bearer $t" https://0f3570d8-03b7-45af-a  
f4-4c1b90504be3.k8s.ondigitalocean.com/api/v1  
{  
    "kind": "APIResourceList",  
    "groupVersion": "v1",  
    "resources": [  
        {  
            "name": "bindings",  
            "singularName": "binding",  
            "namespaced": true,  
            "kind": "Binding",  
            "verbs": [  
                "create"  
            ]  
        },  
        {  
            "name": "namespaces",  
            "singularName": "namespace",  
            "namespaced": true,  
            "kind": "Namespace",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "pods",  
            "singularName": "pod",  
            "namespaced": true,  
            "kind": "Pod",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "services",  
            "singularName": "service",  
            "namespaced": true,  
            "kind": "Service",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "endpoints",  
            "singularName": "endpoint",  
            "namespaced": true,  
            "kind": "Endpoints",  
            "verbs": [  
                "get",  
                "list"  
            ]  
        },  
        {  
            "name": "events",  
            "singularName": "event",  
            "namespaced": true,  
            "kind": "Event",  
            "verbs": [  
                "get",  
                "list"  
            ]  
        },  
        {  
            "name": "leases",  
            "singularName": "lease",  
            "namespaced": true,  
            "kind": "Lease",  
            "verbs": [  
                "get",  
                "list"  
            ]  
        },  
        {  
            "name": "configmaps",  
            "singularName": "configmap",  
            "namespaced": true,  
            "kind": "ConfigMap",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "secrets",  
            "singularName": "secret",  
            "namespaced": true,  
            "kind": "Secret",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "servicesaccounts",  
            "singularName": "serviceaccount",  
            "namespaced": true,  
            "kind": "ServiceAccount",  
            "verbs": [  
                "create",  
                "get",  
                "list",  
                "patch",  
                "update"  
            ]  
        },  
        {  
            "name": "nodes",  
            "singularName": "node",  
            "namespaced": false,  
            "kind": "Node",  
            "verbs": [  
                "get",  
                "list"  
            ]  
        },  
        {  
            "name": "events",  
            "singularName": "event",  
            "namespaced": false,  
            "kind": "Event",  
            "verbs": [  
                "get",  
                "list"  
            ]  
        }  
    ]  
}
```

Points to Note

Even though 2 Pods use same service account, each Pod will receive different set of tokens.

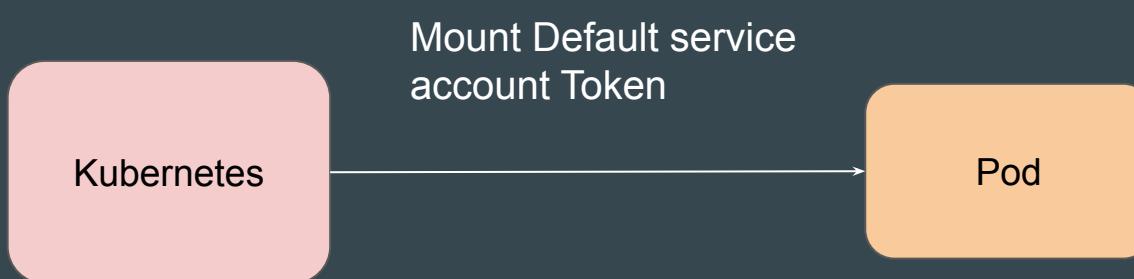


Service Account Security

Understanding the Challenge

When you create a pod without specifying a service account, the Pod is automatically assigned the default service account in the same namespace.

If the default service account is granted **excessive permissions**, all pods using it will inherit those privileges, potentially **leading to security risks**.

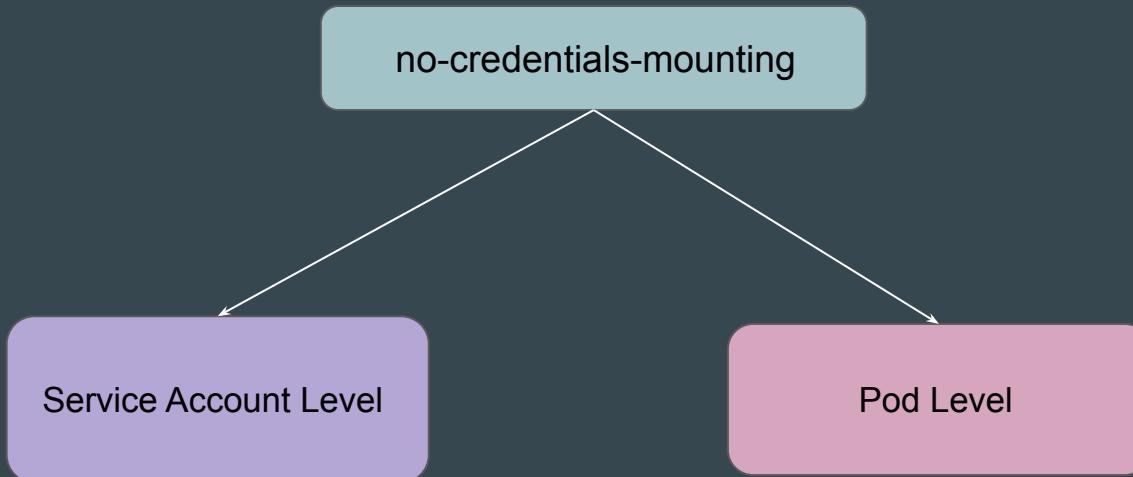


Reference Screenshot

```
Containers:
  test-pod:
    Container ID:  containerd://406a23e1970c3c14aa3f16a0d1b8432b35c3049b40fb0b46f511b06f8c590bbd
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:91734281c0ebfc6f1aea979cffeed5079cfe786228a71c
    Port:           <none>
    Host Port:      <none>
    State:          Running
    Started:        Tue, 18 Feb 2025 00:45:10 +0530
    Ready:          True
    Restart Count:  0
    Environment:   <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-96x4x (ro)
```

Opt Out of Auto-Mounting Credentials

We can **opt out of automounting the credentials** inside the Pods in two different approaches.



Disabling Auto-Mount at the Service Account Level

We can opt out of automounting API credentials for a service account by setting `automountServiceAccountToken: false` on the service account:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
automountServiceAccountToken: false
```

Disabling Auto-Mount at the Pod Level

This can be achieved by adding `automountServiceAccountToken: false` in Pod Spec.

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-pod
spec:
  automountServiceAccountToken: false
  containers:
  - image: nginx
    name: demo
```

Precedence of Auto-Mounting Settings

What if:

1. Auto Mounting is set to False at Service Account Level
2. Auto Mounting is set to True at POD Level



Answer - Clash Situation

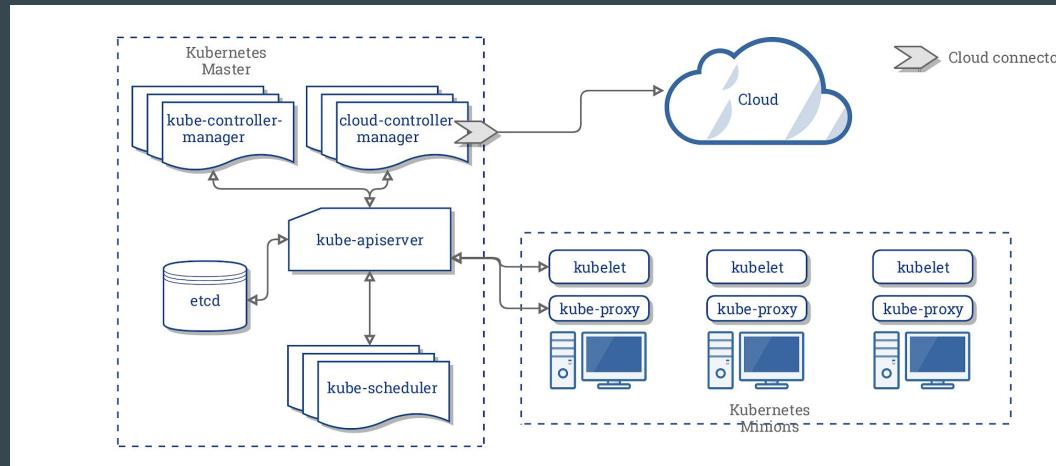
If both the pod specification and the service account define `automountServiceAccountToken`, the pod-level setting takes precedence.

Version Skew Policy

Setting the Base

In Kubernetes, different components (such as the API server, kubelet, etc) interact with each other across various versions.

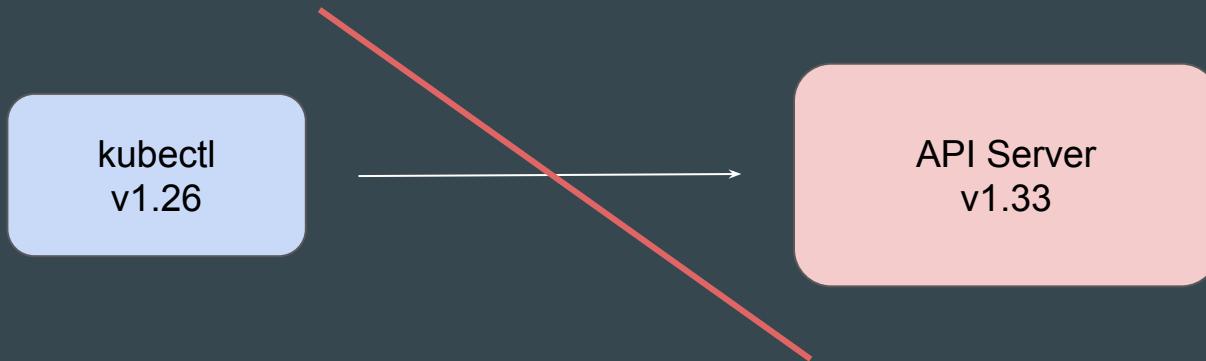
To ensure compatibility and stability, Kubernetes follows a **Version Skew Policy** that defines how versions of these components can differ while still maintaining a functional cluster.



Understanding the Challenge

Version skew happens whenever two components of a software system communicate, but they aren't running at exactly the same version

While some level of skew is allowed, excessive differences between versions can lead to compatibility issues, failures, or unexpected behaviors.



Kubernetes Versioning Basics

Kubernetes versions are expressed as **x.y.z**, where x is the major version, y is the minor version, and z is the patch version.

Format: <MAJOR>.<MINOR>.<PATCH> (e.g., v1.32.2)

Major Version	Minor Version	Patch Version
1	32	2

kube-apiserver and kubelet

Component	Version Skew
kube-apiserver	<p>In highly-available (HA) clusters, the newest and oldest kube-apiserver instances must be within one minor version.</p> <p>If newest kube-apiserver is at 1.32 other kube-apiserver instances are supported at 1.32 and 1.31</p>
kubelet	<p>kubelet must not be newer than kube-apiserver.</p> <p>kubelet may be up to three minor versions older than kube-apiserver</p> <p>Example:</p> <p>kube-apiserver is at 1.32 kubelet is supported at 1.32, 1.31, 1.30, and 1.29</p>

kube-proxy

Component	Version Skew
kube-proxy	<p>kube-proxy must not be newer than kube-apiserver.</p> <p>kube-proxy may be up to three minor versions older than kube-apiserver</p> <p>Example:</p> <p>kube-apiserver is at 1.32</p> <p>kube-proxy is supported at 1.32, 1.31, 1.30, and 1.29</p>

Controller Manager, Scheduler, Cloud Controller Manager

Component	Version Skew
Controller Manager	<p>Must not be newer than the kube-apiserver instances they communicate with.</p> <p>They are expected to match the kube-apiserver minor version, but may be up to one minor version older (to allow live upgrades).</p>
Scheduler	
Cloud Controller Manager	<p>Example:</p> <p>kube-apiserver is at 1.32</p> <p>kube-controller-manager, kube-scheduler, and cloud-controller-manager are supported at 1.32 and 1.31</p>

kubectl

Component	Version Skew
kubectl	<p>kubectl is supported within one minor version (older or newer) of kube-apiserver.</p> <p>Example:</p> <p>kube-apiserver is at 1.32</p> <p>kubectl is supported at 1.33, 1.32, and 1.31</p>

Overview - Upgrading kubeadm Clusters

Setting the Base

It is important to upgrade minor versions sequentially (1.31 -> 1.32 -> 1.33 etc.)



Approach for Upgradation

If you plan to upgrade the Kubernetes version, you have to do it for both the Control Plane Node and Worker Nodes.



Determine Version to Upgrade to

Find the latest patch release for Kubernetes using the OS package manager:

```
root@kubeadm-upgrade:~# apt-cache madison kubeadm
  kubeadm | 1.32.2-1.1 | https://pkgs.k8s.io/core:/stable:/v1.32/deb Packages
  kubeadm | 1.32.1-1.1 | https://pkgs.k8s.io/core:/stable:/v1.32/deb Packages
  kubeadm | 1.32.0-1.1 | https://pkgs.k8s.io/core:/stable:/v1.32/deb Packages
```

kubeadm upgrade [plan, apply] - Control Plane

kubeadm upgrade plan check which versions are available to upgrade to and validate whether your current cluster is upgradeable

Run the **kubeadm upgrade apply** to upgrade the version.

```
Upgrade to the latest stable version:
```

COMPONENT	NODE	CURRENT	TARGET
kube-apiserver	kubeadm-upgrade	v1.31.6	v1.32.2
kube-controller-manager	kubeadm-upgrade	v1.31.6	v1.32.2
kube-scheduler	kubeadm-upgrade	v1.31.6	v1.32.2
kube-proxy		1.31.6	v1.32.2
CoreDNS		v1.11.3	v1.11.3
etcd	kubeadm-upgrade	3.5.15-0	3.5.16-0

```
You can now apply the upgrade by executing the following command:
```

```
kubeadm upgrade apply v1.32.2
```

Point to Note

The kubelet component is **not upgraded** during the kubeadm upgrade apply operation.

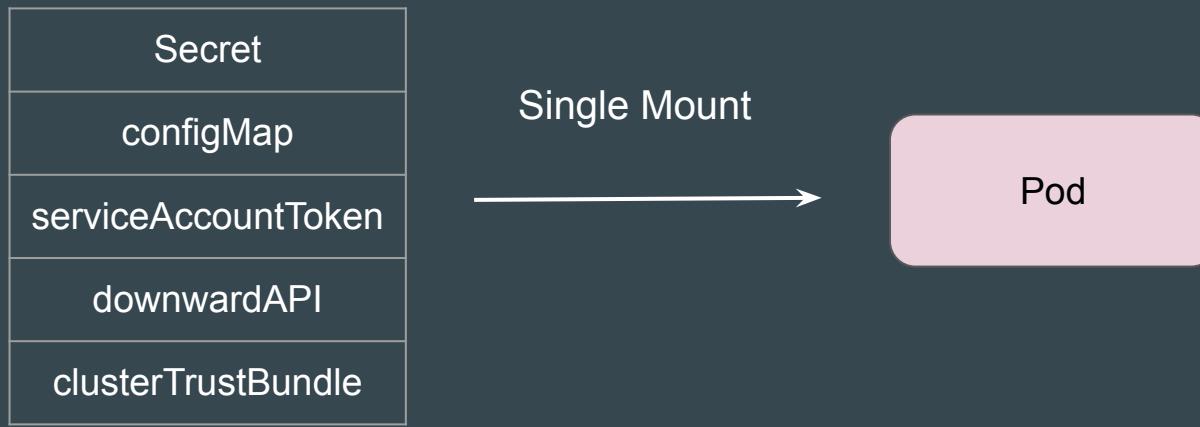
You have to **manually** upgrade kubelet.

```
Components that must be upgraded manually after you have upgraded the control plane with 'kubeadm upgrade apply':  
COMPONENT      NODE          CURRENT    TARGET  
kubelet        kubeadm-upgrade  v1.31.6   v1.32.2  
kubelet        kubeadm-worker-upgrade  v1.31.6   v1.32.2
```

Projected Volumes

Setting the Base

Projected volumes lets you combine several different volume sources into a single volume mount in your pod.



Volume Sources

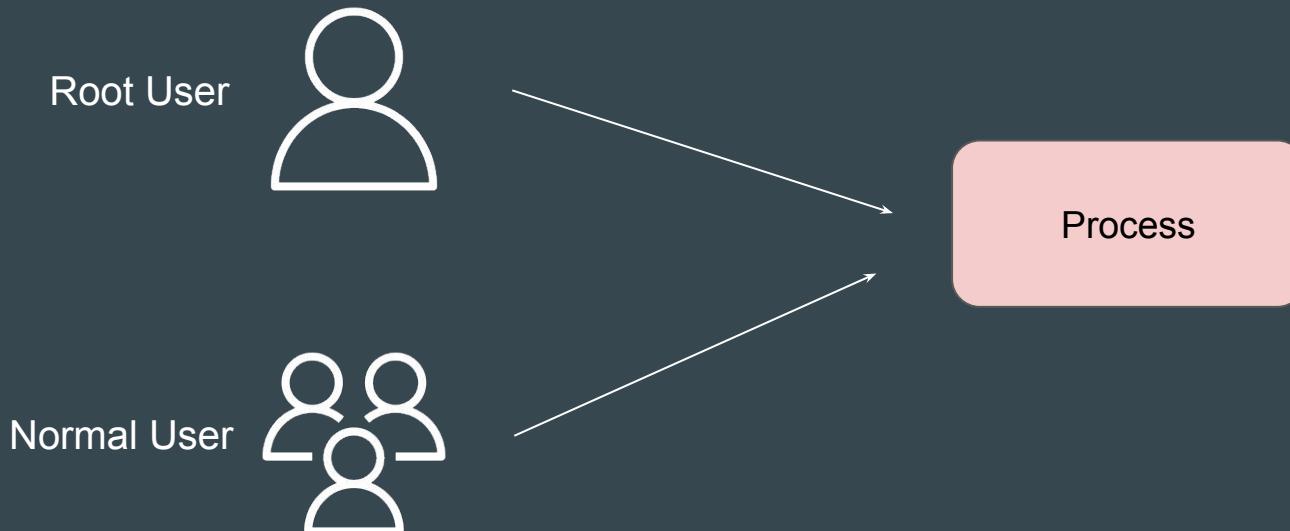
Reference Screenshot

```
volumeMounts:
- name: all-in-one
  mountPath: "/projected-volume"
  readOnly: true
volumes:
- name: all-in-one
  projected:
    sources:
    - secret:
        name: mysecret
        items:
        - key: username
          path: my-group/my-username
    - configMap:
        name: myconfigmap
        items:
        - key: config
          path: my-group/my-config
```

Linux Capabilities

Setting the Base

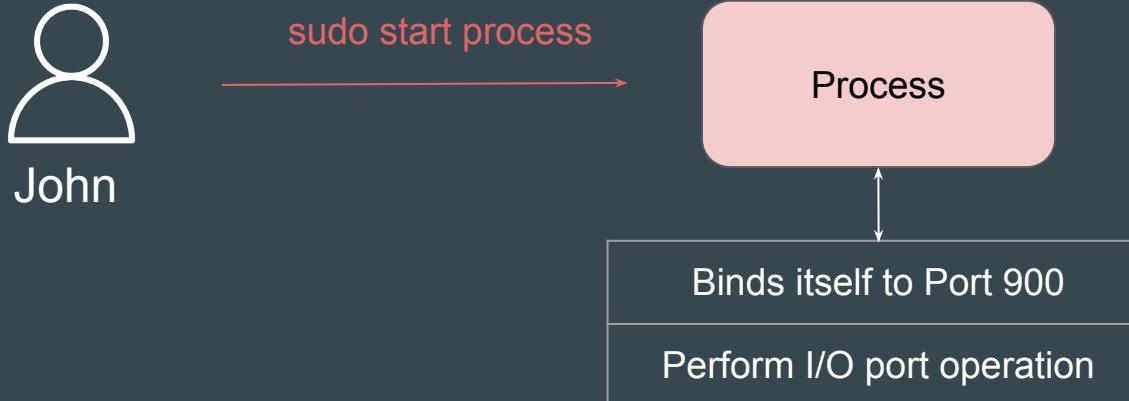
Processes on a Unix-like system run primarily with the permissions of either a user account, or with root permissions.



Understanding the Challenge

This approach of **root user having unlimited privilege** and a **normal user having limited privilege** is not a good enough model.

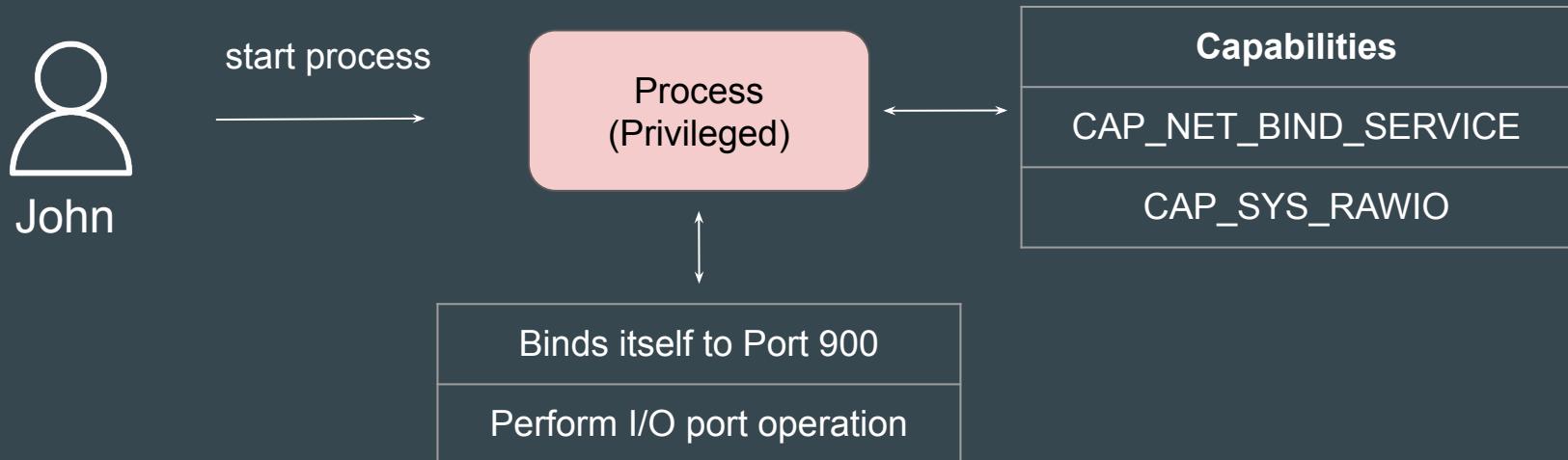
User John wants to start the process that requires certain privileged access.
Most organizations will grant John full access using sudo.



Introducing Linux Capabilities

Linux Capabilities are used to allow binaries (executed by non-root users) to perform privileged operations without providing them all root permissions.

It also allows process started with root to have limited privilege.

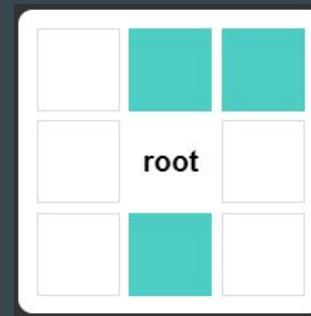
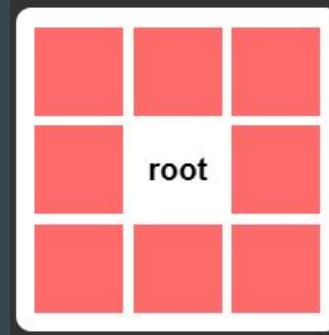


Reference Workflow

The first square represents root without capabilities before Linux kernel 2.2.

The second square represents root with full capabilities.

The third square represents root with only a few capabilities enabled.



Capabilities Available

There are wide range of capabilities available.

Capabilities	Description
CAP_NET_BIND_SERVICE	Bind to ports <1024
CAP_NET_RAW	Use raw sockets
CAP_SYS_TIME	Modify system clock
CAP_SYS_ADMIN	Perform various administrative tasks
CAP_DAC_OVERRIDE	Bypass file permissions

Example - Ping

The `ping` command **uses raw sockets** to send and receive ICMP packets.

Allowing any user to create arbitrary network packets could be a security risk

Instead of giving `ping` full root access, `ping` is given only the `cap_net_raw` capability, which is the minimum required to send and receive ICMP packets.

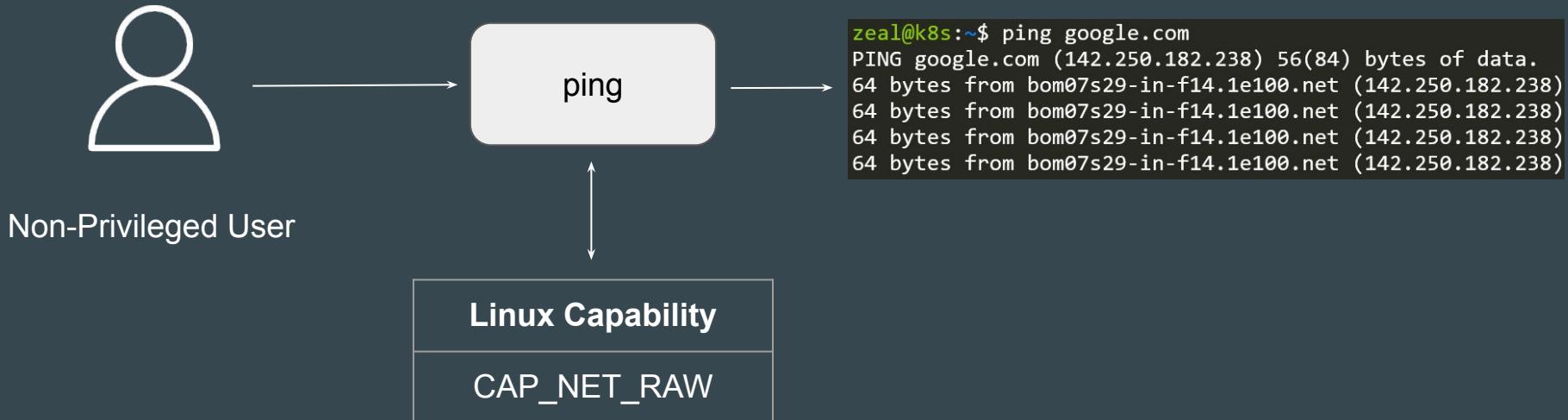
ping



```
root@k8s:~# getcap /bin/ping  
/bin/ping cap_net_raw=ep
```

Example - Ping

Since **CAP_NET_RAW** capability is added to ping, even a non-privileged user will be able to run it without any admin privileges.



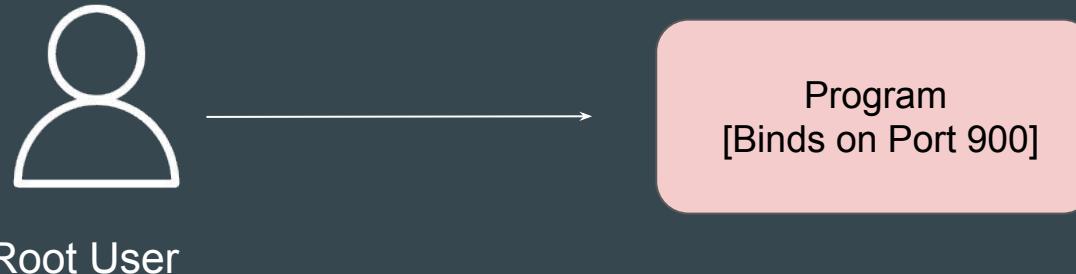
Practical - Linux Capabilities

Simple Use-Case

We have a program that binds to Port 900.

In Linux, the **Port 1-1023** require root privileges

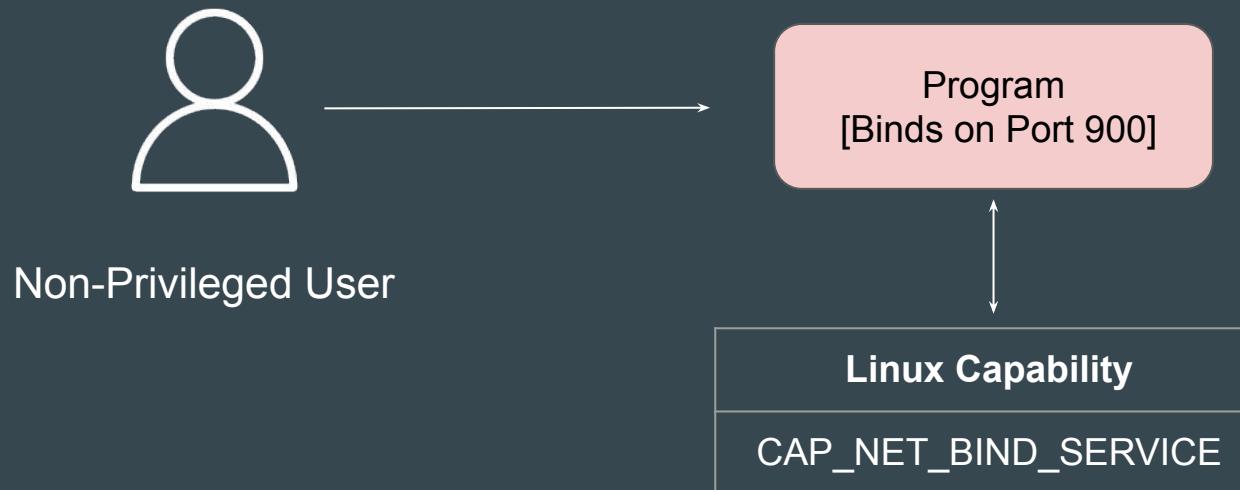
Aim is to allow non-privileged user to also run the program.



Using Right Capability

The `CAP_NET_BIND_SERVICE` capability in Linux allows a process to bind to privileged network ports (those below 1024) without requiring root privileges.

We will associate this capability with the program.



Reference Screenshot - Without Capability

The binary of `bind_port_900` does NOT have any capabilities.

If a non-privileged user tries to run it, it will result in permission denied..

```
zeal@k8s:/tmp$ getcap bind_port_900  
zeal@k8s:/tmp$ |
```



```
zeal@k8s:/tmp$ ./bind_port_900  
Bind failed: Permission denied
```

Reference Screenshot - With Capability

After the `cap_net_bind_service` capability is added, a non-privileged user will be able to start the process that binds to Port 900.

```
zeal@k8s:/tmp$ getcap bind_port_900  
bind_port_900 cap_net_bind_service=ep
```



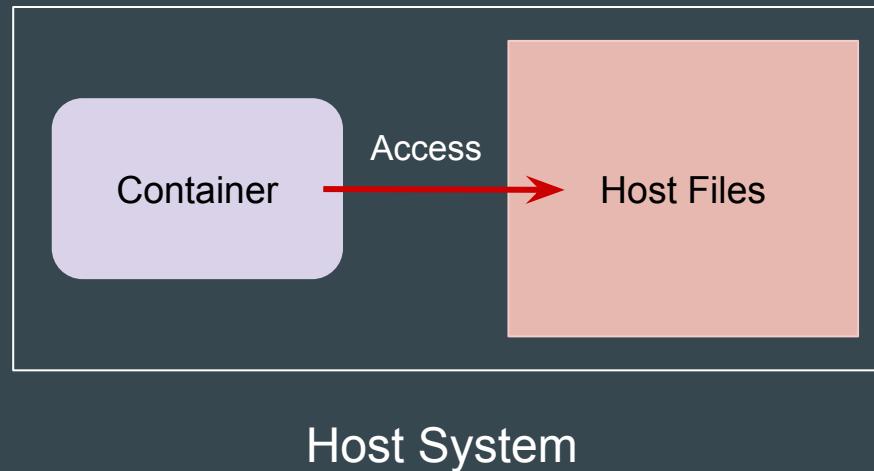
```
zeal@k8s:/tmp$ ./bind_port_900  
Successfully bound to port 900
```

Security Context

Understanding the Challenge

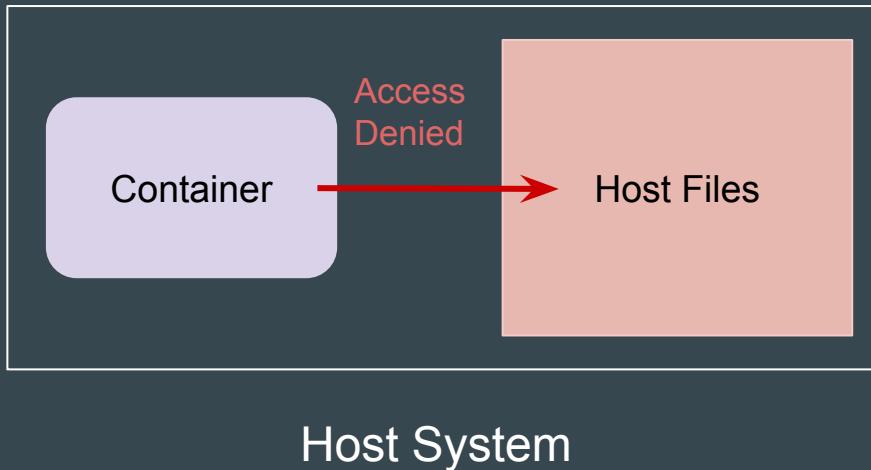
Many times, the containers run with **root user privileges**.

In case of **container breakouts**, an attacker can get full access to the host system.



Running Container with Non Root User

If the container runs with non-root privileges, it will be unable to modify the critical host files and will have limited access to the host system.



Introduction to Security Context

A security context defines privilege and access control settings for a Pod or Container.

Run as non-privileged user

```
apiVersion: v1
kind: Pod
metadata:
  name: better-pod
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 1000
  containers:
  - name: better-container
    image: busybox
    command: ["sleep", "36000"]
```

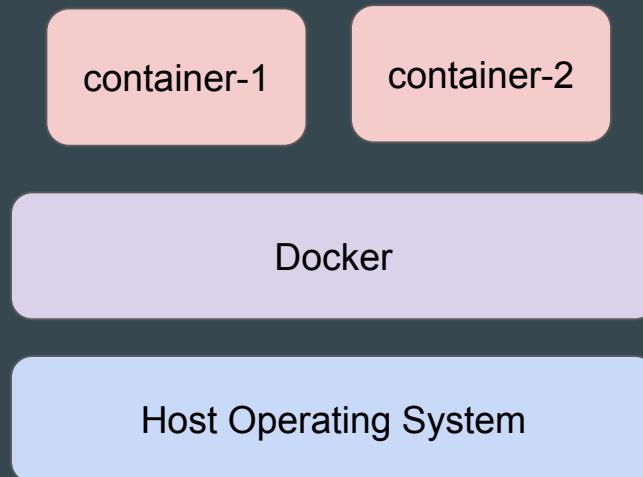
Comparison Table

Field	Description	Use-Case
runAsUser	Specifies the user ID (UID) a container's process runs as.	Use when you want the container to run as a specific user rather than the default (commonly root).
runAsGroup	Specifies the primary group ID (GID) a container's process runs as.	Use when you want the container's primary group to be a specific GID.
fsGroup	Specifies a group ID (GID) for volume-mounted files. Files created in mounted volumes will be owned by this GID.	Use when you need to control file permissions for a shared volume (e.g., for multiple containers in a Pod).

Privileged Pods

Setting the Base

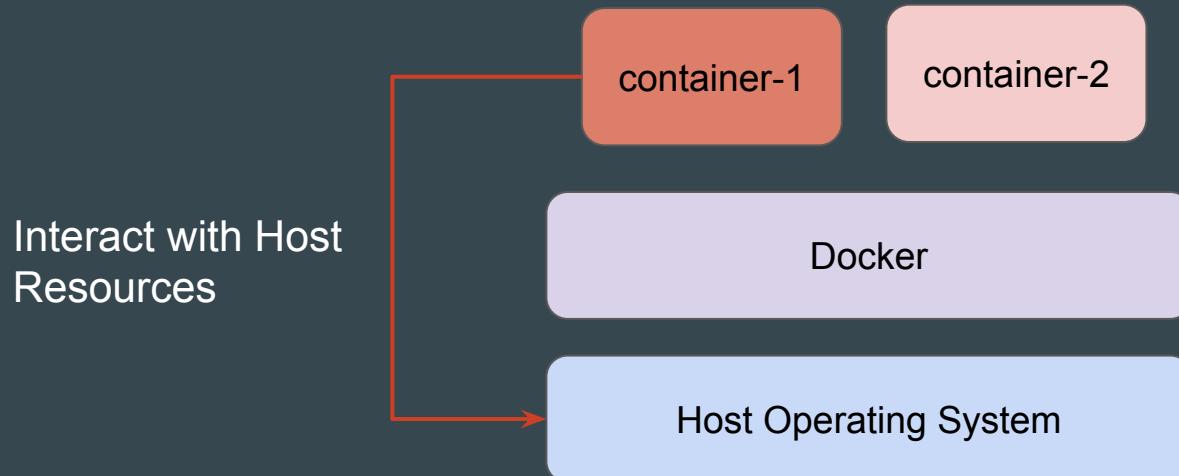
Kubernetes, by default, **enforces strong security boundaries to isolate containers from the host system and from each other.**



Introducing Privileged Pods

Certain workloads **require elevated privileges** to interact directly with the host system's resources or kernel capabilities.

This is where "Privileged Pods" come into play.



Why use Privileged Pods?

Tasks that require direct hardware access, such as loading kernel modules, manipulating network devices (e.g., creating custom network interfaces), or accessing specific device files.

Some networking tools that need deep system access to manage network interfaces, routing tables, or firewalls.

Example - Access to Host Device

```
/ # ls /dev
core          mqueue      pts          stderr      termination-log zero
fd            null        random      stdin       tty
full          ptmx        shm         stdout      urandom
```

Non-Privileged Pod (Top) vs Privileged Pods (Bottom)

```
/ # ls /dev
autofs        loop7       tty          tty28       tty48       ttyS1       vcsu1
btrfs-control mapper     tty0         tty29       tty49       ttyS2       vcsu2
bus           mem        tty1         tty3        tty5        ttyS3       vcsu3
core          mqueue     tty10        tty10      tty30       tty50       uhid
cpu_dma_latency net        tty11        tty11      tty31       tty51       uinput
cuse          null       tty12        tty12      tty32       tty52       urandom
fd            nvram     tty13         tty13      tty33       tty53       userfaultfd
full          port       tty14        tty14      tty34       tty54       vcs
fuse          ppp        tty15        tty15      tty35       tty55       vcs1
hpet          psaux     tty16        tty16      tty36       tty56       vcs2
hwrng         ptmx      tty17        tty17      tty37       tty57       vcs3
input          pts        tty18        tty18      tty38       tty58       vcs4
kmsg          random     tty19        tty19      tty39       tty59       vcs5
kvm           rfckill   tty2         tty2       tty4        tty6        vcs6
loop-control  rtc0      tty20        tty20      tty40       tty60       vcsa
                                         vhost-net
```

Example - dmesg

```
/ # dmesg  
dmesg: klogctl: Operation not permitted
```

Non-Privileged Pod (Top) vs Privileged Pods (Bottom)

```
[ 0.000000] BIOS-provided physical RAM map:  
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable  
[ 0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved  
[ 0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved  
[ 0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000007ffdafff] usable  
[ 0.000000] BIOS-e820: [mem 0x00000000007ffdb000-0x00000000007ffffffff] reserved  
[ 0.000000] BIOS-e820: [mem 0x000000000feffc000-0x000000000fefffff] reserved  
[ 0.000000] BIOS-e820: [mem 0x000000000ffffc0000-0x000000000ffffffff] reserved  
[ 0.000000] printk: bootconsole [earlyser0] enabled  
[ 0.000000] NX (Execute Disable) protection: active  
[ 0.000000] SMBIOS 2.8 present.  
[ 0.000000] DMI: DigitalOcean Droplet/Droplet, BIOS 20171212 12/12/2017  
[ 0.000000] Hypervisor detected: KVM  
[ 0.000000] kvm-clock: Using msrs 4b564d01 and 4b564d00  
[ 0.000005] kvm-clock: using sched offset of 3783394224 cycles  
[ 0.001065] clocksource: kvm-clock: mask: 0xfffffffffffffff max_cycles: 0x1cd42e4dfffb,  
[ 0.003421] tsc: Detected 1999.999 MHz processor
```

Deploying a Privileged Pod

You can configure a Pod to be privileged by setting `privileged: true` in `securityContext`.

```
apiVersion: v1
kind: Pod
metadata:
  name: privileged-pod
spec:
  containers:
  - image: nginx
    name: privileged
    securityContext:
      privileged: true
```

Point to Note

Privileged containers are given all Linux capabilities, including capabilities that they don't require.

In most cases, you should avoid using privileged containers, and instead grant the specific capabilities required by your container using the capabilities field in the securityContext field



Set Capabilities for a Container

Setting the Base

By default, Kubernetes runs containers with a restricted set of capabilities.

If your application requires additional privileges (e.g., binding to privileged ports, modifying network settings), you must explicitly grant them.

```
/ # cat /proc/1/status | grep Cap
CapInh: 0000000000000000
CapPrm: 0000000a80425fb
CapEff: 0000000a80425fb
CapBnd: 0000000a80425fb
CapAmb: 0000000000000000
```

Setting the Base

In Kubernetes, Linux capabilities are managed under the `securityContext` in `pod.spec`

```
apiVersion: v1
kind: Pod
metadata:
  name: capabilities-demo
spec:
  containers:
    - name: sec-ctx-4
      image: gcr.io/google-samples/hello-app:2.0
      securityContext:
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
```

Add and Drop Section

The **add** section grants specific capabilities

The **drop** section removes capabilities to minimize security risks.

```
apiVersion: v1
kind: Pod
metadata:
  name: capabilities-pod-2
spec:
  containers:
    - name: demo-2
      image: busybox
      command: ["sleep", "36000"]
      securityContext:
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
          drop:
            - ALL
```

Points to Note

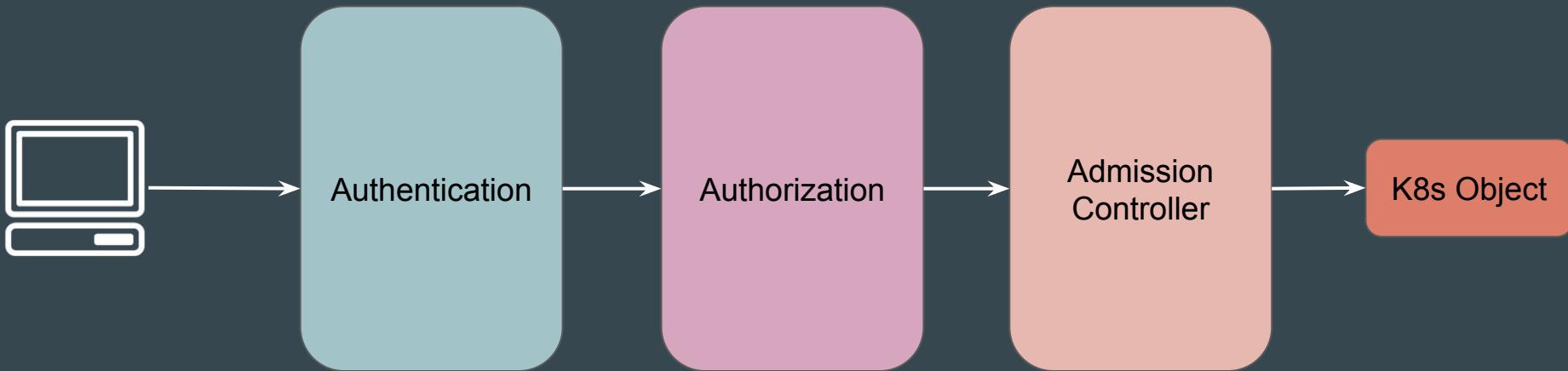
Only add the capabilities your application actually needs.

Use drop: ["ALL"] First: Then explicitly add only required capabilities.

Admission Controllers

Setting the Base

Admission controllers that allow you to intercept, validate, and potentially modify requests to the Kubernetes API server before they are persisted as objects in etcd



Types of Admission Controllers

Admission Controller Type	Description
Validating	These can only allow or deny requests based on custom rules.
Mutating	These can modify requests before they are processed, in addition to allowing or denying them.

Example 1 - NamespaceAutoProvision

By default, if you attempt to create resources in a nonexistent namespace, you will immediately encounter an error.

The **NameSpaceAutoProvision admission controller** inspects all incoming requests for namespaced resources and checks whether the referenced namespace exists.

If the namespace does not exist, the controller automatically creates it.

```
C:\>kubectl run nginx --image=nginx -n unknown-namespace  
Error from server (NotFound): namespaces "unknown-namespace" not found
```

Example 2 - PodSecurity

The **PodSecurity Admission controller** enforces Pod Security Standards.

By enforcing Pod Security Standards, it ensures that pods deployed in your cluster comply with defined security best practices.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: nginx
      securityContext:
        privileged: true
```

Is this good?



PodSecurity Admission
Controller

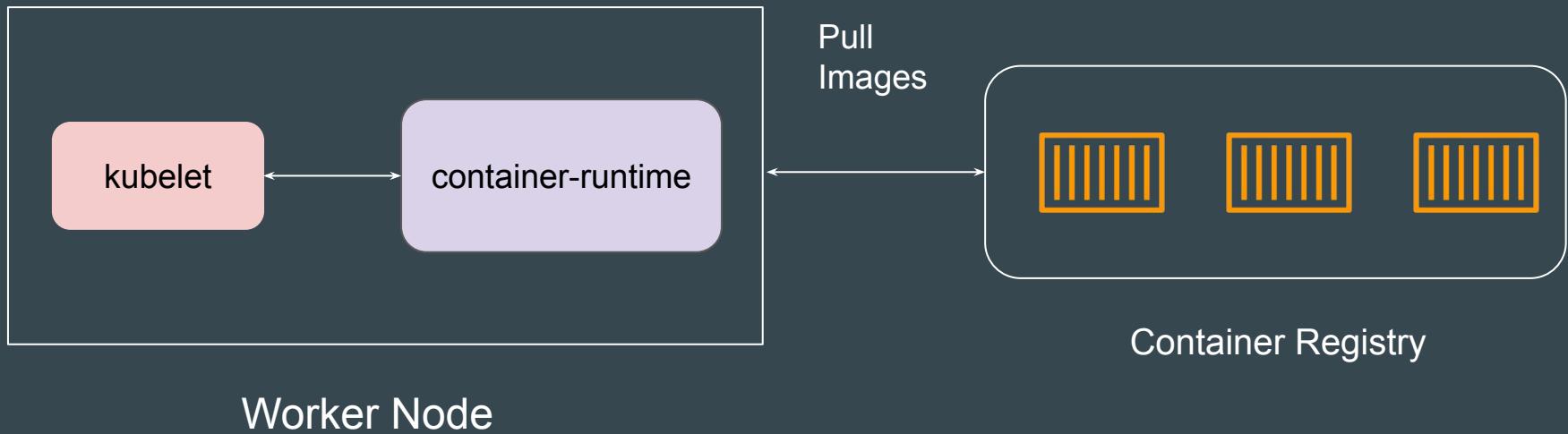
Hell Naw. No Privileged
containers



ImagePullPolicy

Setting the Base

Whenever you create a Pod in kubernetes, the kubelet component interacts with the runtime (docker, containerd) to pull the container image from registry.



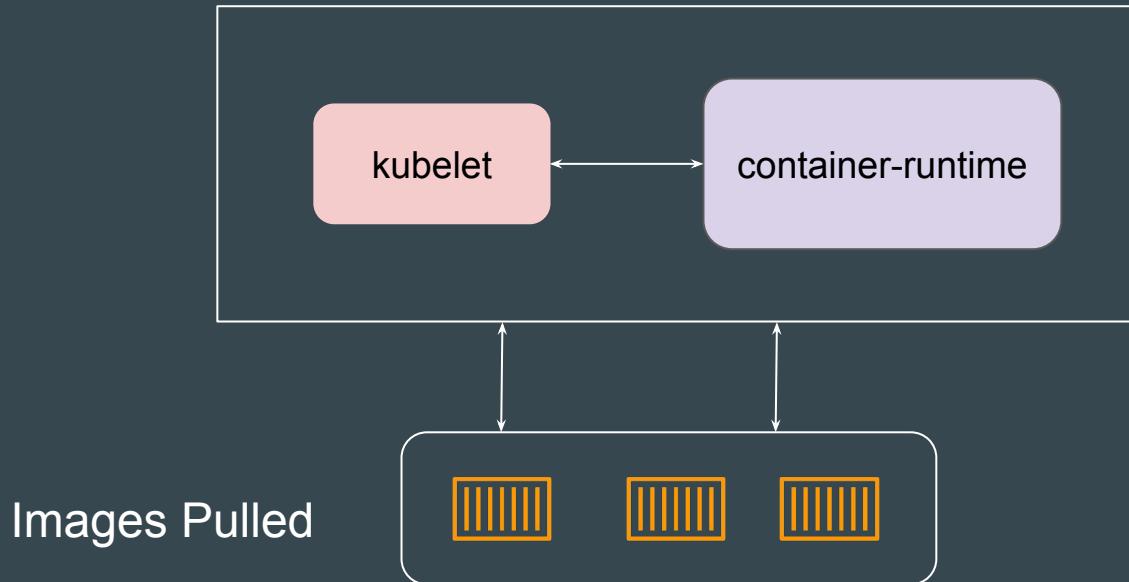
```
root@kubeadm:~# crictl images
```

IMAGE	TAG
docker.io/calico/apiserver	v3.29.1
docker.io/calico/cni	v3.29.1
docker.io/calico/csi	v3.29.1
docker.io/calico/kube-controllers	v3.29.1
docker.io/calico/node-driver-registrar	v3.29.1
docker.io/calico/node	v3.29.1
docker.io/calico/pod2daemon-flexvol	v3.29.1
docker.io/calico/typha	v3.29.1
docker.io/library/httpd	latest
docker.io/library/nginx	<none>
docker.io/library/nginx	latest
quay.io/tigera/operator	v1.36.2

IMAGE ID	SIZE
421726ace5ed1	43.5MB
7dd6ea186aba0	97.6MB
bda8c42e04758	9.4MB
6331715a2ae96	35.6MB
8b7d18f262d5c	12MB
feb26d4585d68	143MB
2b7452b763ec8	6.86MB
4cb3738506f5a	31.3MB
f7d8bafbd9a9f	58.5MB
9bea9f2796e23	72.1MB
97662d24417b3	72.2MB
3045aa4a360d4	21.8MB

Introducing ImagePullPolicy

The **ImagePullPolicy** tells Kubernetes when to pull an image from a registry.



Available ImagePullPolicy Settings

Values	Description
Always	Pulls the latest image from container registry.
IfNotPresent	Pulls the image only if it isn't already present on the node.
Never	Never pull the image. Instead, it assumes the image is already available on the node.

Setting ImagePullPolicy

We can explicitly specify the `imagePullPolicy` for the Pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: nginx
      imagePullPolicy: Always
```

Clarification Point - AlwaysPullImages

Every time the kubelet launches a container, the kubelet queries the container image registry to resolve the name to an image digest.

If the kubelet has a **container image with that exact digest cached locally**, the **kubelet uses its cached image**; otherwise, the kubelet pulls the image with the resolved digest, and uses that image to launch the container.

Default Image Pull Policy

If you omit the `imagePullPolicy` field, and you don't specify the tag for the container image, `imagePullPolicy` is automatically set to `Always`

If you omit the `imagePullPolicy` field, and you specify the tag for the container image that isn't latest, the `imagePullPolicy` is automatically set to `IfNotPresent`

Point to Note

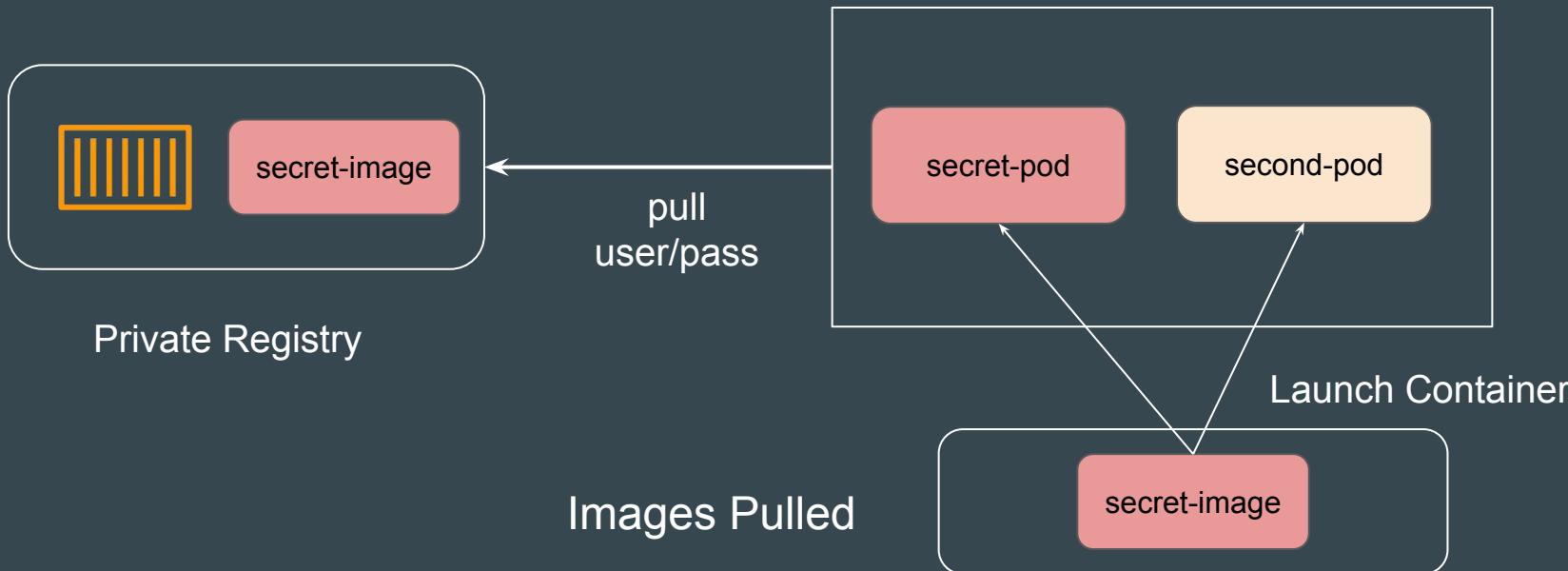
You should **avoid using the :latest tag when deploying containers in production** as it is harder to track which version of the image is running and more difficult to roll back properly.

Instead, specify a meaningful tag such as v1.42.0 and/or a digest.

Admission Controller - AlwaysPullImages

Understanding the Challenge

If a **sensitive container image** is downloaded to a worker node using valid credentials, an unauthorized person can later create a new Pod using that same image with `imagePullPolicy` set to 'Never', **bypassing the need for authentication credentials**.



```
root@kubeadm:~# crictl images
```

IMAGE	TAG
docker.io/calico/apiserver	v3.29.1
docker.io/calico/cni	v3.29.1
docker.io/calico/csi	v3.29.1
docker.io/calico/kube-controllers	v3.29.1
docker.io/calico/node-driver-registrar	v3.29.1
docker.io/calico/node	v3.29.1
docker.io/calico/pod2daemon-flexvol	v3.29.1
docker.io/calico/typha	v3.29.1
docker.io/library/httpd	latest
docker.io/library/nginx	<none>
docker.io/library/nginx	latest
quay.io/tigera/operator	v1.36.2

	IMAGE ID	SIZE
421726ace5ed1	43.5MB	
7dd6ea186aba0	97.6MB	
bda8c42e04758	9.4MB	
6331715a2ae96	35.6MB	
8b7d18f262d5c	12MB	
feb26d4585d68	143MB	
2b7452b763ec8	6.86MB	
4cb3738506f5a	31.3MB	
f7d8bafbd9a9f	58.5MB	
9bea9f2796e23	72.1MB	
97662d24417b3	72.2MB	
3045aa4a360d4	21.8MB	

Introducing AlwaysPullImages

This admission controller modifies every new Pod to force the image pull policy to Always

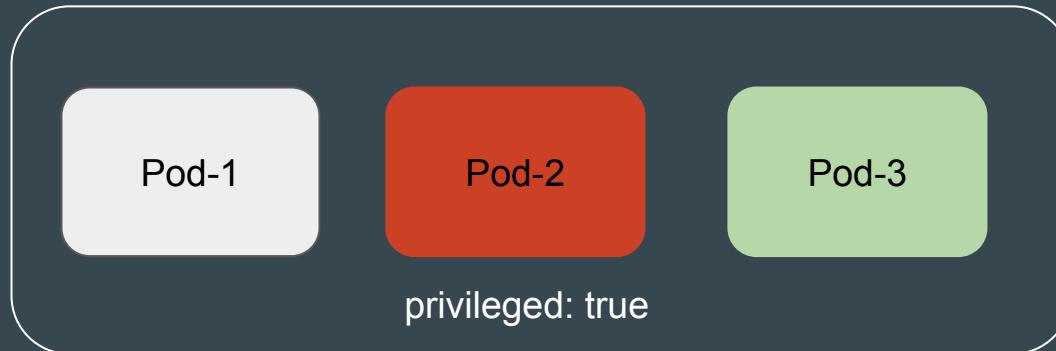
This is useful in a multitenant cluster so that users can be assured that their private images can only be used by those who have the credentials to pull them.

Without this admission controller, once an image has been pulled to a node, any pod from any user can use it by knowing the image's name

Pod Security Standard

Simple Example

It is often seen that users launch privileged pods in production namespaces even when they are not required.



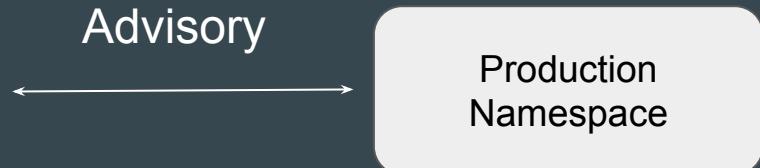
Production Namespace

Example - Security Standard Set

The security team has defined the following standards for the production namespaces.

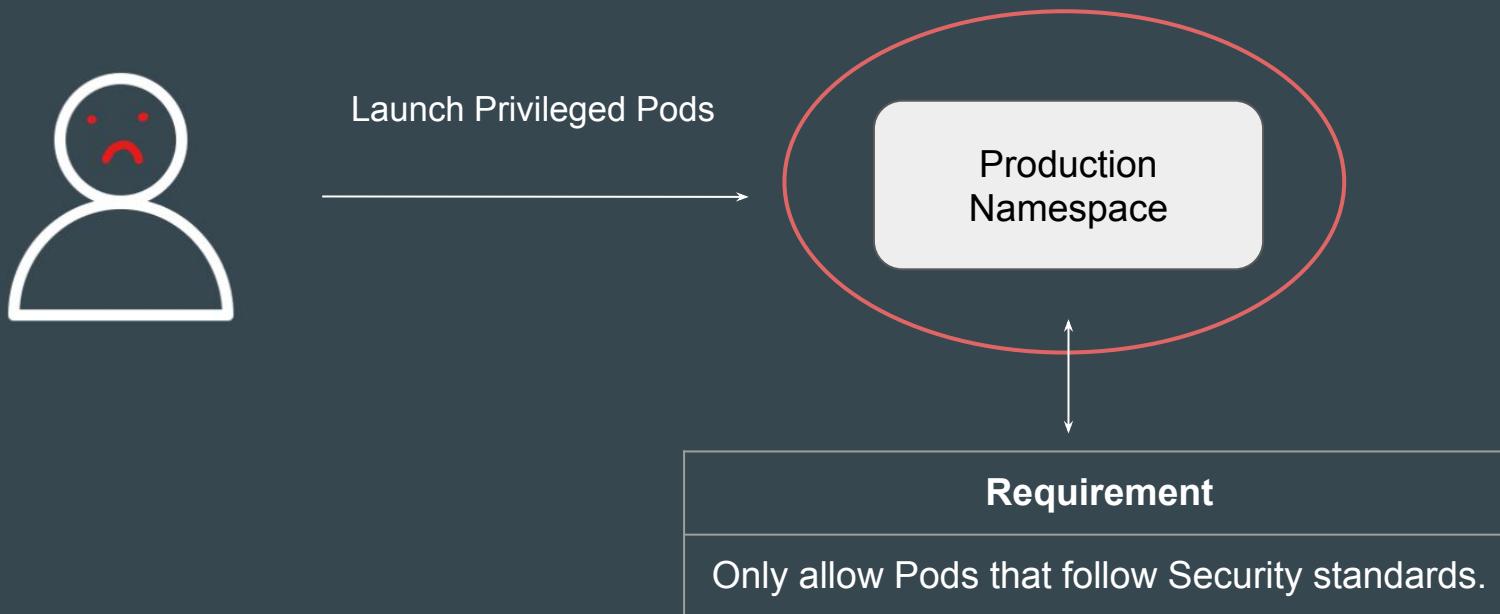
This is more of an advisory, and enforcement is not 100% achieved.

Security Standards
Containers must run as non-root users.
HostPath volumes must be forbidden.
Seccomp profile must not be set to Unconfined.
AppArmor profile is applied by default.
No Privileged Containers



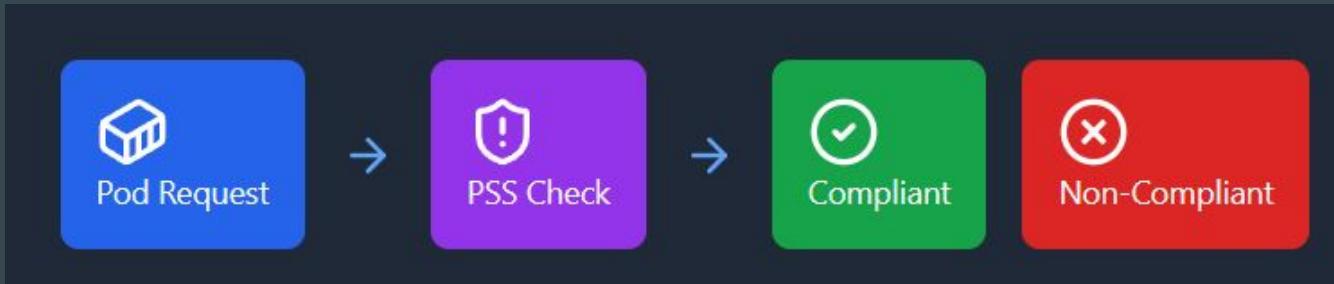
Ideal Approach

In an ideal approach, you want any attempt to circumvent the security standards set by the security team to be blocked automatically.



Introducing Pod Security Standards

Pod Security Standards are a set of guidelines established by Kubernetes to ensure that Pods running in a namespace meet specific security requirements.



Policies in Pod Security Standard

The Pod Security Standards **define three different policies**.

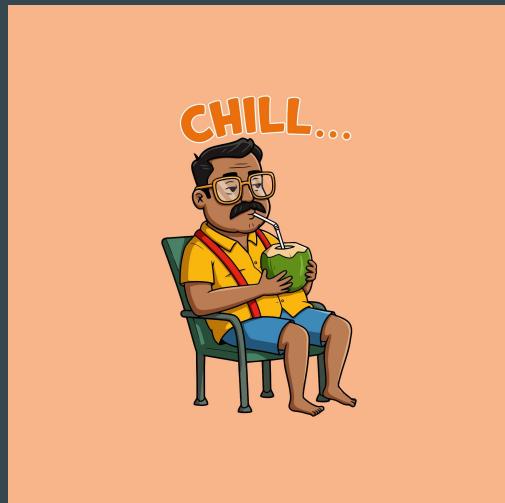
These policies range from highly-permissive to highly-restrictive.

Policies	Description
Privileged	Unrestricted policy, providing the widest possible level of permissions. Allows privilege escalations
Baseline	Minimally restrictive policy which prevents known privilege escalations.
Restricted	Heavily restricted policy, following current Pod hardening best practices.

1 - Privileged Policy

The Privileged policy is **purposely-open**, and entirely unrestricted.

The Privileged policy is defined by an absence of restrictions



2 - Baseline Policy

The Baseline policy is aimed at **ease of adoption** for common containerized workloads while preventing known privilege escalations.

Targeted at application operators and developers of non-critical applications

Operations Not Allowed (Reference purpose)

Sharing the host namespaces.

Privileged Pod

HostPath volumes and HostPorts

3 - Restricted Policy

The Restricted policy is **aimed at enforcing current Pod hardening best practices**, at the expense of some compatibility.

Targeted at operators and developers of **security-critical applications**, as well as lower-trust users.

Examples

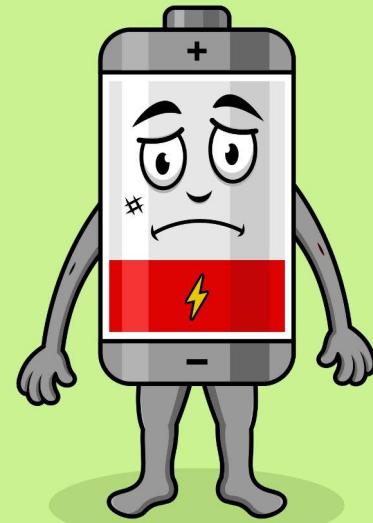
Everything from Baseline Policy +
Containers must be required to run as non-root users.

Containers must not set runAsUser to 0

Seccomp profile must be explicitly set to one of the allowed values.



Pods with Privileged Policy

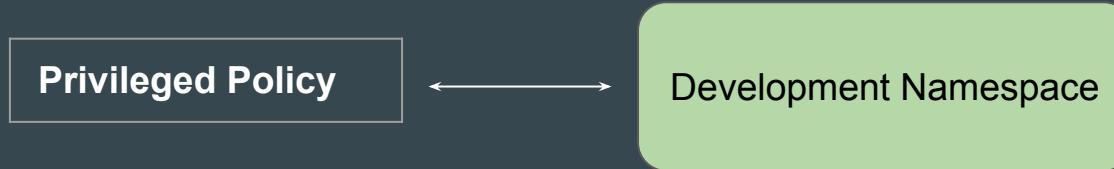
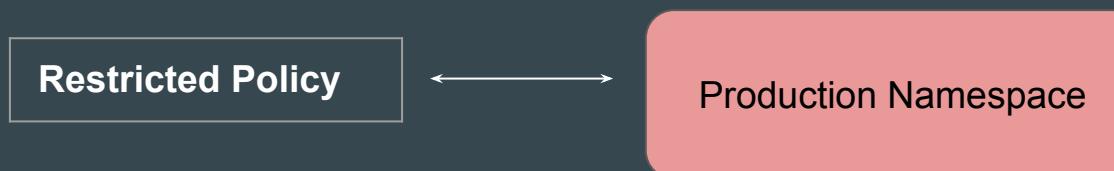


Pods with Restricted Policy

Where to Define Policy

The policies can be defined at a namespace level.

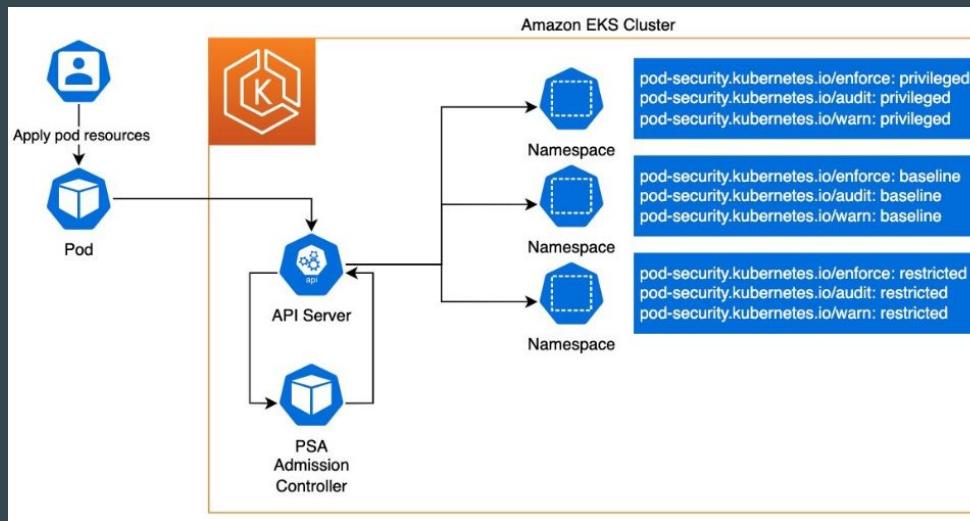
You can also apply it at a cluster level.



Pod Security Admission

The **Pod Security Admission** (PSA) controller is a built-in admission controller in Kubernetes that enforces the Pod Security Standards (PSS).

When a pod is created, PSA checks if it complies with the security policies set at the namespace level.

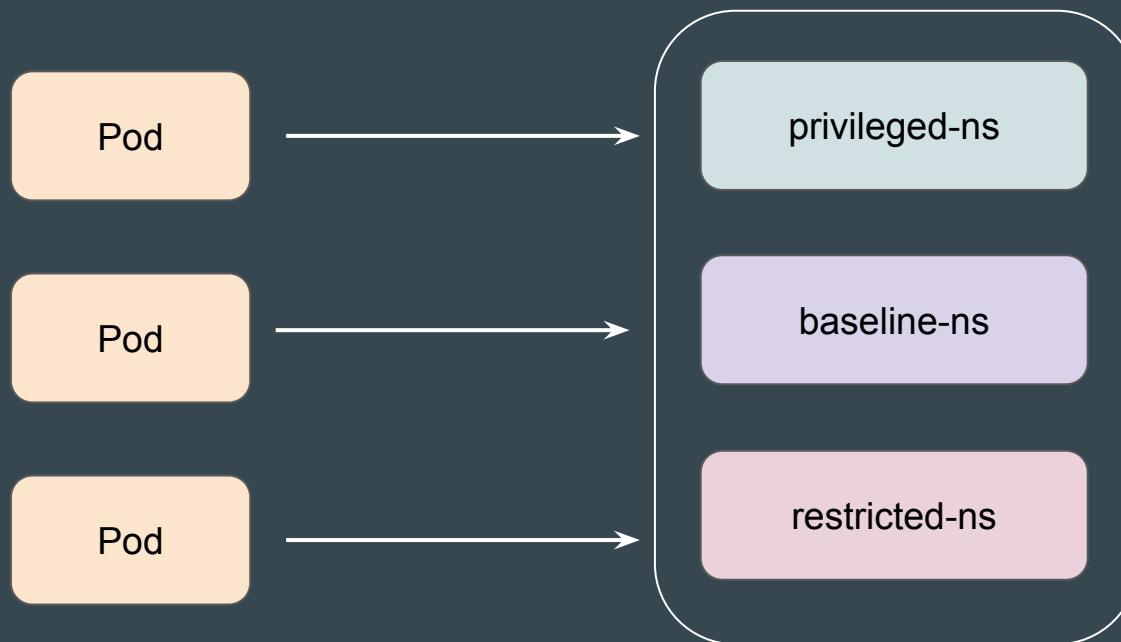


Practical - Pod Security Standards

Overall Workflow

We will create 3 namespaces for each policy level as part of PSP.

We will launch ideal pod that fits PSS restrictions.



Modes - Pod Security Admission

Revising the Basics

We usually add a label on a namespace to define the appropriate pod security standard profile.

Label Format:

pod-security.kubernetes.io/<MODE>: <profile>

```
root@kubeadm:~# kubectl label namespace secured-ns pod-security.kubernetes.io/enforce=restricted  
namespace/secured-ns labeled
```

Understanding the Modes

Modes	Description
Enforce	Rejects Pods with policy violations.
Audit	Allows Pods with policy violations but includes an audit annotation in the audit log event record.
Warn	Allows Pods with policy violations but warns users.

Multiple Modes can be used

A namespace can configure any or all modes, or even set a different level for different modes.

```
apiVersion: v1
kind: Namespace
metadata:
  name: secured-ns
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/warn: restricted
```

Reference Screenshot

The following image depicts the user trying to deploy the nginx pod in the test-namespace that has two labels attached.

- pod-security.kubernetes.io/enforce: privileged
- pod-security.kubernetes.io/warn: restricted

```
root@kubeadm:~/test# kubectl run nginx --image=nginx -n test-namespace
Warning: would violate PodSecurity "restricted:latest": allowPrivilegeEscalation != false (container "nginx" must set securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container "nginx" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "nginx" must set securityContext.runAsNonRoot=true), seccompProfile (pod or container "nginx" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")
pod/nginx created
```

Points to Note - PSA and PSS

Mode Version

For each mode, there are two labels that determine the policy used.

Kubernetes introduces different versions of security policies (e.g v1.24, v1.25, etc.), and the mode version indicates which set of security rules are applied.

```
apiVersion: v1
kind: Namespace
metadata:
  name: secured-ns
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/enforce-version: v1.32
```

Point to note - Version not Defined

If you do not define the version in the Pod Security Standard label, Kubernetes will use the default version of the Pod Security Admission (PSA) policy that is supported by the cluster.

The default version is typically the latest stable version supported by the Kubernetes API in that release.

If later, when Kubernetes is upgraded, the Pod Security Standards may change in newer versions. This could lead to unexpected policy enforcement changes that might break workloads.

Workload resources and Pod templates

The **enforce mode does not apply to workload objects** like Deployments etc. Instead, enforcement happens only when the actual Pods are created.

Example:

A workload object (like a Deployment) can be created even if its Pod template violates security policies.

But when Kubernetes tries to create Pods from that Deployment, those Pods will be blocked if they violate the enforced security policies.

Labels to Existing Namespace

When an **enforce policy label** is added or changed, the admission plugin will test each pod in the namespace against the new policy. Violations are returned to the user as warnings.

Existing running pods are not affected.

```
root@kubeadm:~/test# kubectl label namespace default pod-security.kubernetes.io/enforce=restricted
Warning: existing pods in namespace "default" violate the new PodSecurity enforce level "restricted:latest"
Warning: capabilities-pod-1 (and 3 other pods): allowPrivilegeEscalation != false, unrestricted capabilities,
true, seccompProfile
namespace/default labeled
root@kubeadm:~/test# kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
capabilities-pod-1  1/1     Running   5 (7h30m ago)   2d9h
capabilities-pod-2  1/1     Running   5 (7h26m ago)   2d9h
normal-pod       1/1     Running   5 (7h33m ago)   2d9h
test-pod         1/1     Running   0            63s
```

Adding Labels with Dry Run

It is helpful to apply the `--dry-run` flag when initially evaluating security profile changes for namespaces.

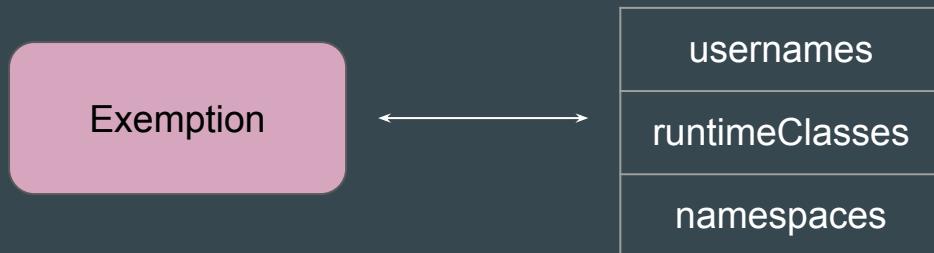
The Pod Security Standard checks will still be run in dry run mode, giving you information about how the new policy would treat existing pods, without actually updating a policy.

```
root@kubeadm:~/test# kubectl label --dry-run=server ns default pod-security.kubernetes.io/enforce=restricted
Warning: existing pods in namespace "default" violate the new PodSecurity enforce level "restricted:latest"
Warning: capabilities-pod-1 (and 3 other pods): allowPrivilegeEscalation != false, unrestricted capabilities,
true, seccompProfile
namespace/default labeled (server dry run)
```

Exemptions

You can **define exemptions from pod security enforcement** in order to allow the creation of pods that would have otherwise been prohibited due to the policy associated with a given namespace.

Exemptions can be statically configured in the Admission Controller configuration via the `--admission-control-config-file` to kube-apiserver.

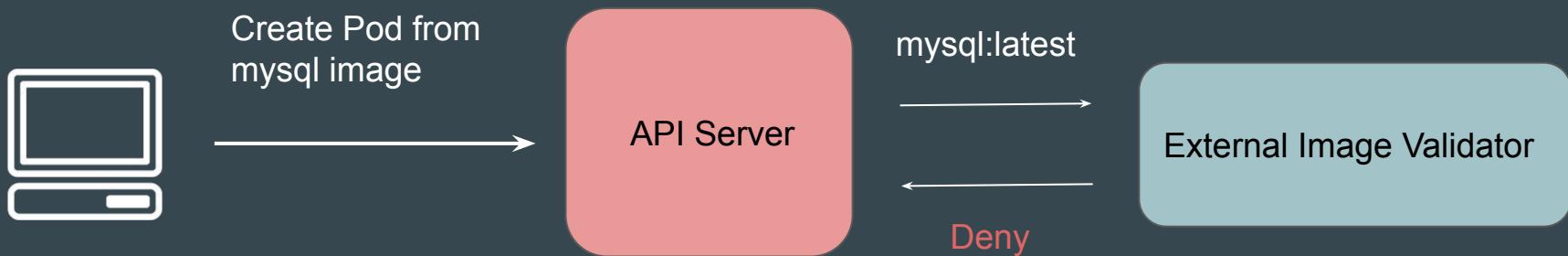


```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: PodSecurity
  configuration:
    apiVersion: pod-security.admission.config.k8s.io/v1
    kind: PodSecurityConfiguration
    defaults:
      enforce: "privileged"
      enforce-version: "latest"
      audit: "privileged"
      audit-version: "latest"
      warn: "privileged"
      warn-version: "latest"
    exemptions:
      # Array of authenticated usernames to exempt.
      usernames: []
      # Array of runtime class names to exempt.
      runtimeClasses: []
      # Array of namespaces to exempt.
      namespaces: []
```

Admission Controller - ImagePolicyWebHook

Understanding the Basics

The **ImagePolicyWebhook** admission controller allows Kubernetes to check with an external service before allowing pods to run based on their container images.



Configuration File

ImagePolicyWebhook uses a configuration file to set options for the behavior of the backend.

ImagePolicyWebHook

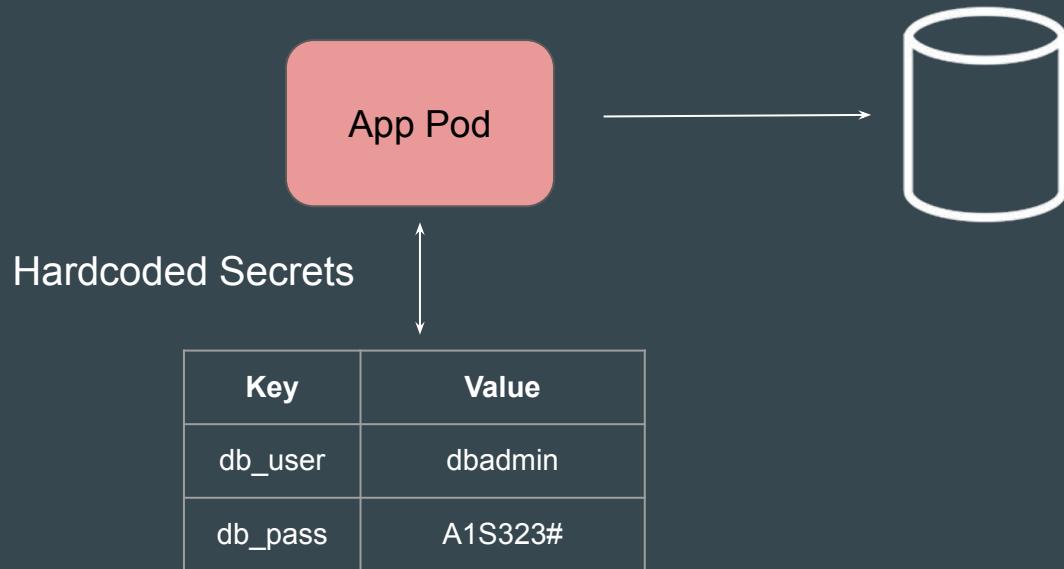


```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: ImagePolicyWebhook
    configuration:
      imagePolicy:
        kubeConfigFile: "/etc/kubernetes/pki/webhook-kubeconfig"
        allowTTL: 50
        denyTTL: 50
        retryBackoff: 500
        defaultAllow: false
```

Kubernetes Secrets

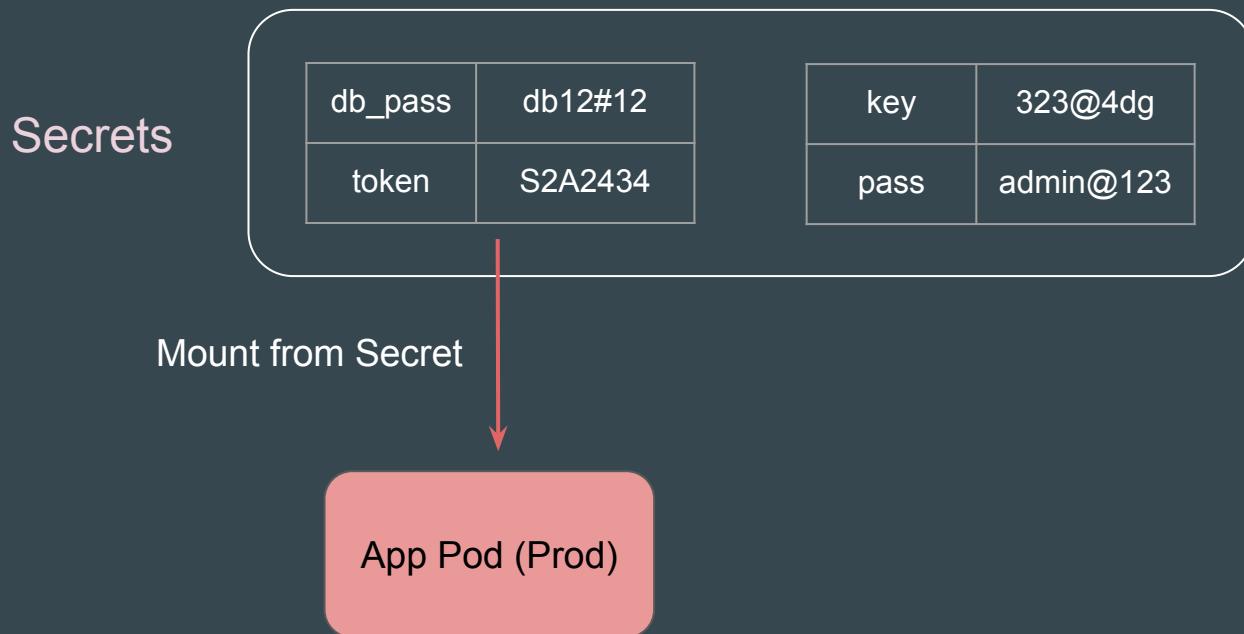
HardCoding Secrets Should be Avoided

It is frequently observed that sensitive data like passwords, tokens, etc., are hard-coded as part of the container image.



Introducing Secret

Kubernetes Secrets is a feature that allows us to store these sensitive data.



Reference Screenshot

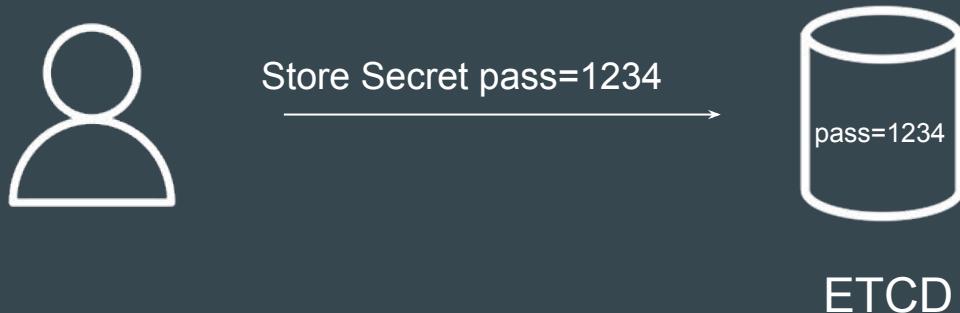
```
C:\>kubectl get secret
NAME          TYPE        DATA   AGE
my-secret     Opaque      1      22m
```

```
C:\>kubectl get secret my-secret -o yaml
apiVersion: v1
data:
  db_pass: QTIjMTI1U0A=
kind: Secret
metadata:
  creationTimestamp: "2025-01-16T02:34:21Z"
  name: my-secret
  namespace: default
  resourceVersion: "5877928"
  uid: d3c383cb-c2e3-4f9b-95ef-d1f0ec6a04be
type: Opaque
```

Point to Note - Part 1

By default, Secrets are not very secure as they are not stored in encrypted format in the data store (ETCD). You can setup this configuration manually.

You can also additionally protect access to secrets using RBAC for access control.



Point to Note - Part 2

When you view a secret, Kubectl will print the Secret in **base64** encoded format.

You'll have to use an external base64 decoder to decode the Secret fully

```
C:\>kubectl get secret my-secret -o yaml
apiVersion: v1
data:
  db_pass: QTIjMTI1U0A= ←
kind: Secret
metadata:
  creationTimestamp: "2025-01-16T02:34:21Z"
  name: my-secret
  namespace: default
  resourceVersion: "5877928"
  uid: d3c383cb-c2e3-4f9b-95ef-d1f0ec6a04be
type: Opaque
```

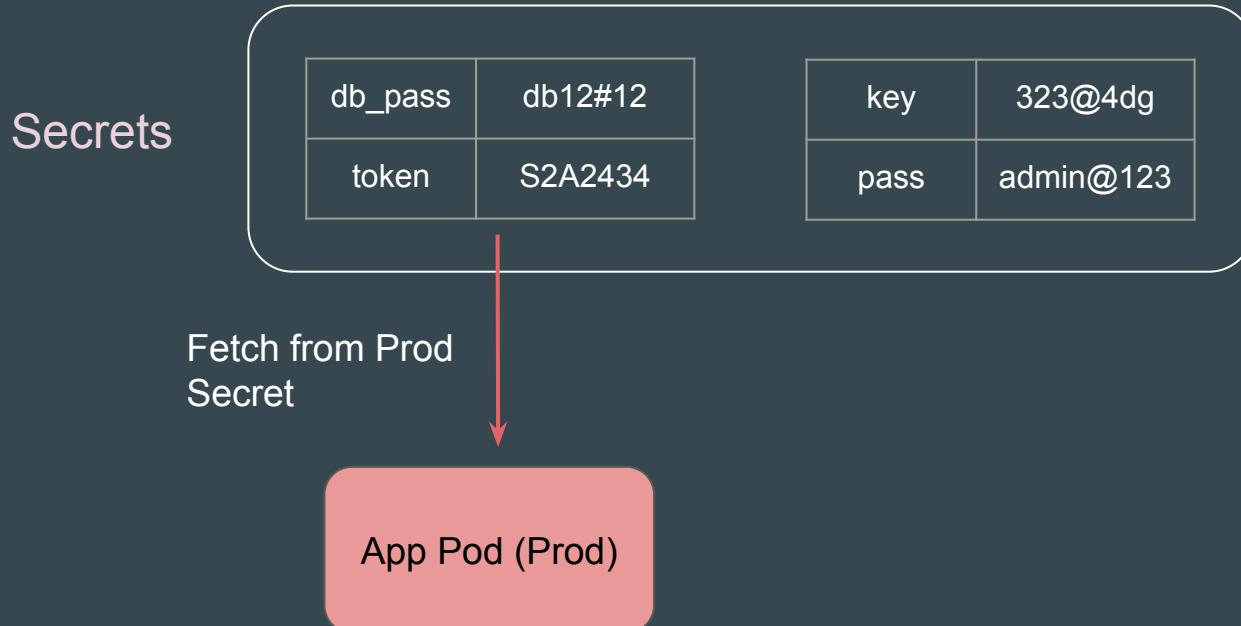
base64 encoded

Kubernetes Secrets - Practical

Two Parts of this Practical

First Part: Create Kubernetes Secret

Second Part: Mount the Secret inside the Pod.



Part 1 - Create Secret

Use the `kubectl create secret` command to create secret in Kubernetes.

Examples:

```
# Create a new secret named my-secret with keys for each file in folder bar
kubectl create secret generic my-secret --from-file=path/to/bar

# Create a new secret named my-secret with specified keys instead of names on disk
kubectl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa
--from-file=ssh-publickey=path/to/id_rsa.pub

# Create a new secret named my-secret with key1=supersecret and key2=topsecret
kubectl create secret generic my-secret --from-literal=key1=supersecret --from-literal=key2=topsecret

# Create a new secret named my-secret using a combination of a file and a literal
kubectl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-literal=passphrase=topsecret

# Create a new secret named my-secret from env files
kubectl create secret generic my-secret --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

Different Approaches for Reference

A Pod can reference the Secret in a variety of ways, such as in a volume mount or as an environment variable.



Part 1 - Mount Secret Inside Pod (Volume)

Using Volume Mounts, you can mount a specific secret inside a Pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
    - name: secretmount
      image: nginx
      volumeMounts:
        - name: secret-volume
          mountPath: "/etc/secrets"
          readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: firstsecret
```

Part 2 - Mount Secret Inside Pod (Env)

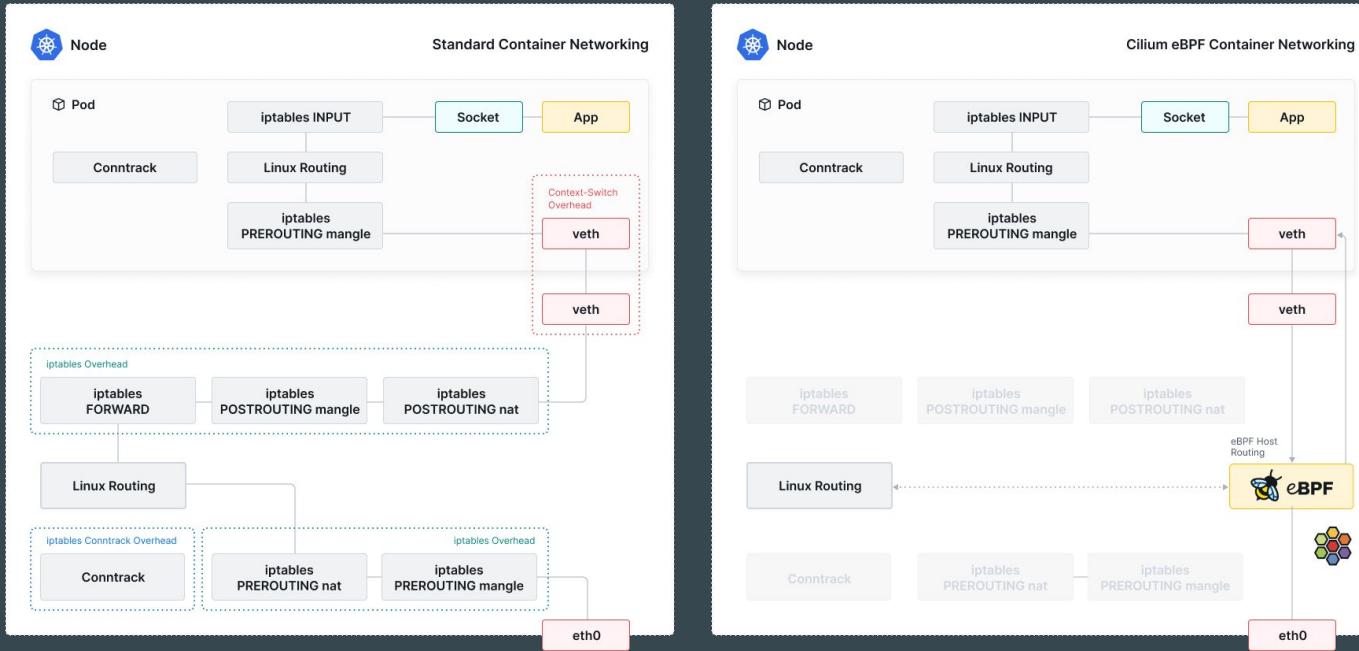
In this method, the values in the Secrets are exposed as **environment variables** to the container.

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env
spec:
  containers:
  - name: secret-env
    image: nginx
    env:
      - name: SECRET_USERNAME
        valueFrom:
          secretKeyRef:
            name: firstsecret
            key: dbpass
```

Overview of Cillium

Setting the Base

Unlike traditional CNI plugins which primarily rely on iptables and IP routing, **Cilium uses eBPF to achieve efficient packet processing**, reducing complexity and improving performance.



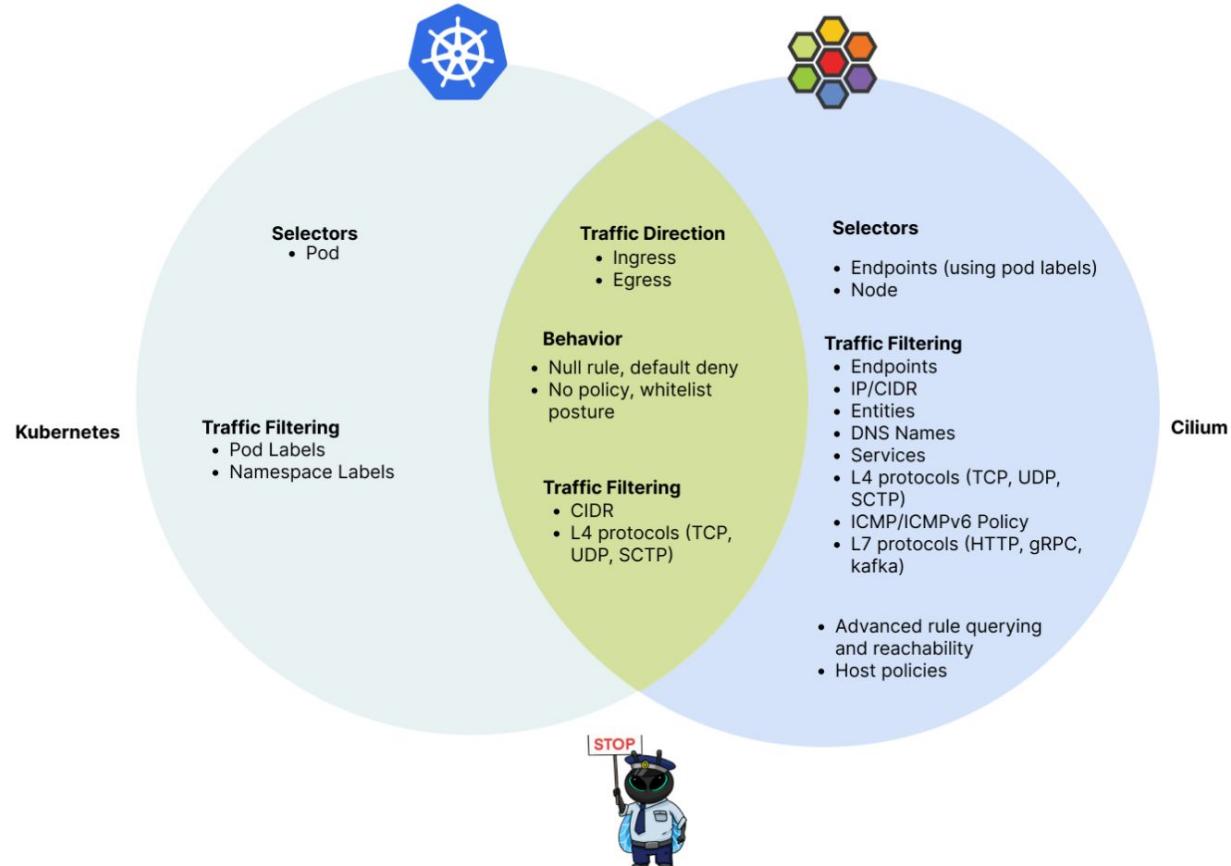
Cilium Network Policies

Cilium network policy provides more granularity, flexibility, and advanced features than the standard Kubernetes network policy.

Cilium supports defining granular rulesets at Layers 3, 4, and 7 of the OSI model

Feature	K8s Network Policy	Cilium Network Policy
Basic L3/L4 layer isolations	Yes	Yes
L7 (HTTP,DNS, Kafka)	No	Yes
Better observability	No	Yes (Hubble)

Kubernetes Network Policy vs Cilium Network Policy

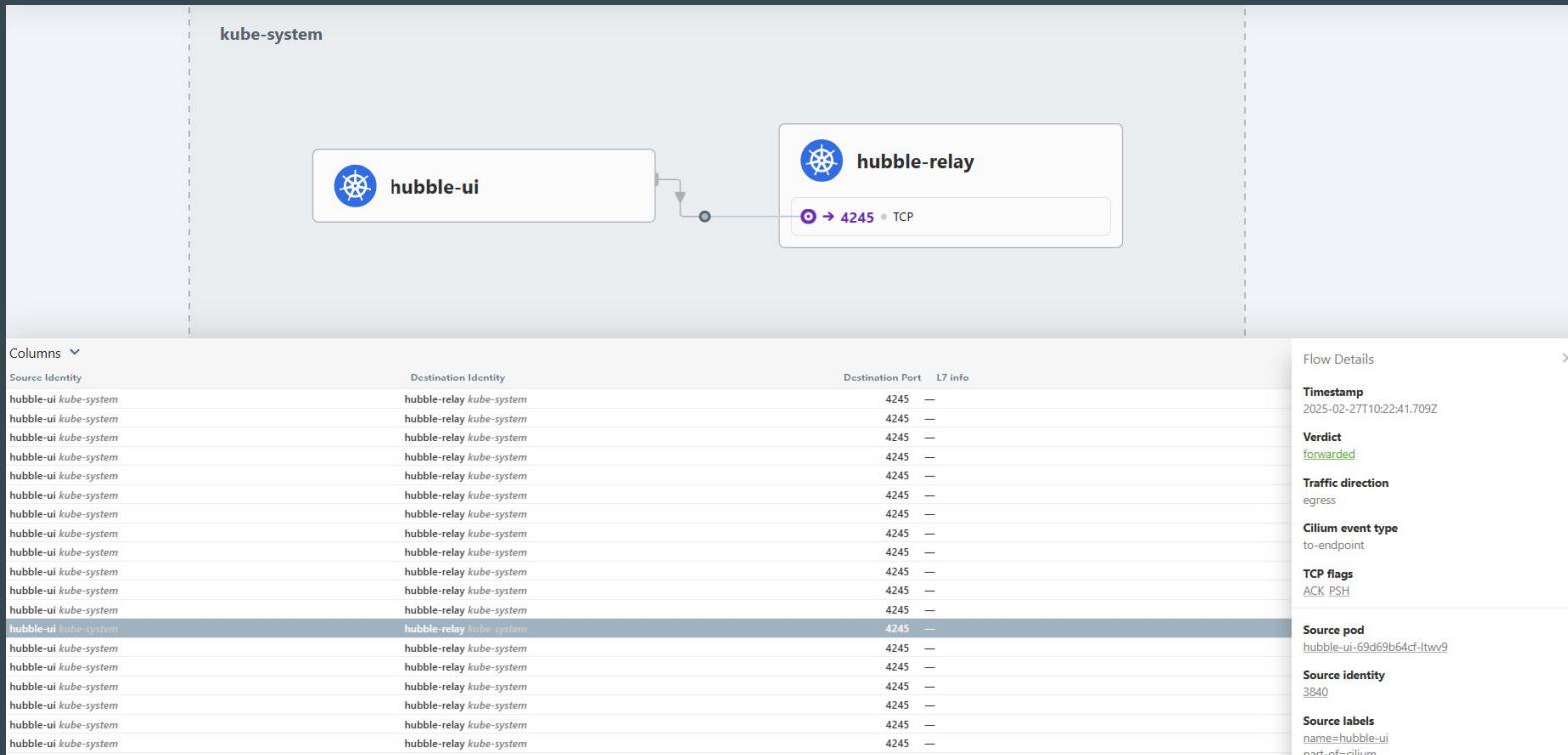


Hubble in Cilium

Hubble is Cilium's observability layer, offering deep insights into your Kubernetes cluster's network.

It's like a powerful microscope for your cluster's network traffic, allowing you to see and understand what's happening at a granular level.

```
root@ubuntu:~# hubble observe --pod busybox -f
Feb 27 10:04:28.687: default/busybox:54712 (ID:10040) -> kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
Feb 27 10:04:28.688: default/busybox:54712 (ID:10040) <- kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
Feb 27 10:04:28.688: default/busybox:35437 (ID:10040) -> kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
Feb 27 10:04:28.689: default/busybox:35437 (ID:10040) <- kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
Feb 27 10:04:28.689: default/busybox:46665 (ID:10040) -> kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
Feb 27 10:04:28.689: default/busybox:46665 (ID:10040) <- kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
Feb 27 10:04:28.690: default/busybox:51442 (ID:10040) -> kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
Feb 27 10:04:28.710: default/busybox:51442 (ID:10040) <- kube-system/coredns-668d6bf9bc-mv57p:53 (ID:36861) to-endpoint FORWA
RDED (UDP)
```



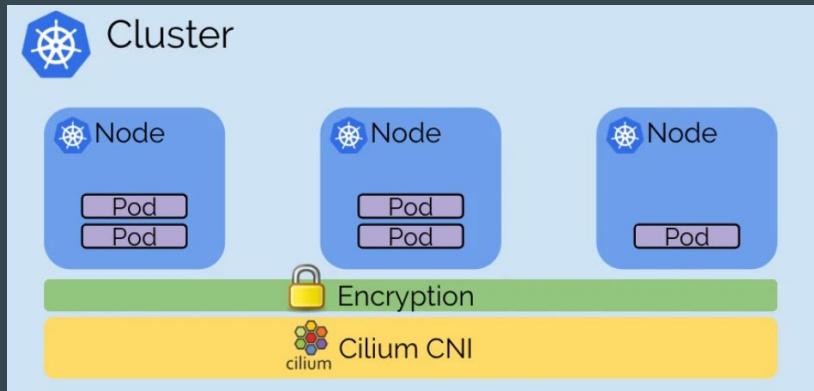
Point to Note - Network Policies

Cilium implements the standard Kubernetes network policy spec. Your Kubernetes network policies work out of the box with Cilium without any additional changes.

Transparent Encryption

Cilium provides encryption using IPsec or WireGuard to secure communication between workloads in a Kubernetes cluster.

Encryption ensures that traffic between pods or nodes remains confidential and protected from interception.



Structure of Cilium Network Policies

Cilium Network Policies

Cilium network policy provides more granularity, flexibility, and advanced features than the standard Kubernetes network policy.

Cilium supports defining granular rulesets at Layers 3, 4, and 7 of the OSI model

Feature	K8s Network Policy	Cilium Network Policy
Basic L3/L4 layer isolations	Yes	Yes
L7 (HTTP,DNS, Kafka)	No	Yes
Better observability	No	Yes (Hubble)

Base Structure

The base structure of CiliumNetworkPolicy is similar to a traditional Network policies in Kubernetes.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-policy
```

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: cilium-network-policy
```

Simple Default Deny Policy

The following policy selects all the pods in the default namespace and denies all inbound and outbound traffic.

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: default-deny
spec:
  # empty select matches all pods in the namespace
  endpointSelector: {}

  # no rules on ingress, deny all incoming traffic
  ingress:
  - {}

  # no rules on egress, deny all outgoing traffic
  egress:
  - {}
```

Match Labels

The `matchLabels` field within an `endpointSelector` is used to select which endpoints (pods) the policy applies to, based on their Kubernetes labels

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector:
    matchLabels:
      run: nginx
```

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: cilium-network-policy
spec:
  endpointSelector:
    matchLabels:
      run: nginx
```

Ingress

The ingress field **defines** rules for incoming traffic.

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: allow-from-curl
spec:
  endpointSelector:
    matchLabels:
      app: nginx
  ingress:
    - fromEndpoints:
        - matchLabels:
            app: curl
```

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: database-policy
  namespace: default
spec:
  endpointSelector:
    matchLabels:
      tier: database
  ingress:
    - fromCIDRSet:
        - cidr: 102.213.50.174/32
  toPorts:
    - ports:
        - port: "3306"
          protocol: TCP
```

Egress

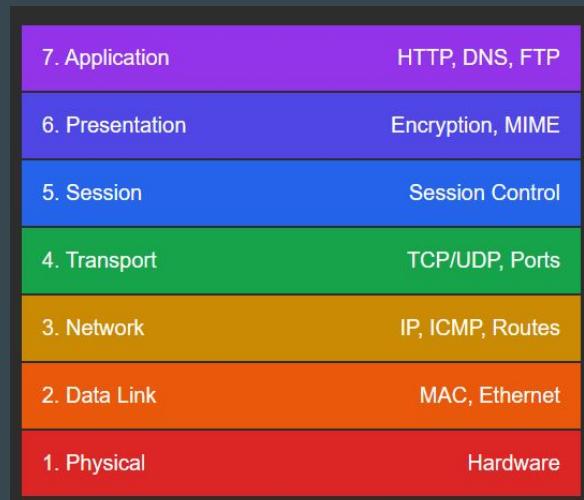
The egress field defines rules for outgoing traffic.

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: allow-egress-to-payments-app
spec:
  endpointSelector:
    matchLabels:
      run: database
  egress:
    - toEndpoints:
        - matchLabels:
            app: payment-app
```

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "egress-cidr-rule"
spec:
  endpointSelector:
    matchLabels:
      app: prod
  egress:
    - toCIDR:
        - 206.189.132.19/32
```

Setting the Base

We can create Cilium Network Policies to control traffic at Layer 3, Layer 4, and Layer 7 of the OSI Model.

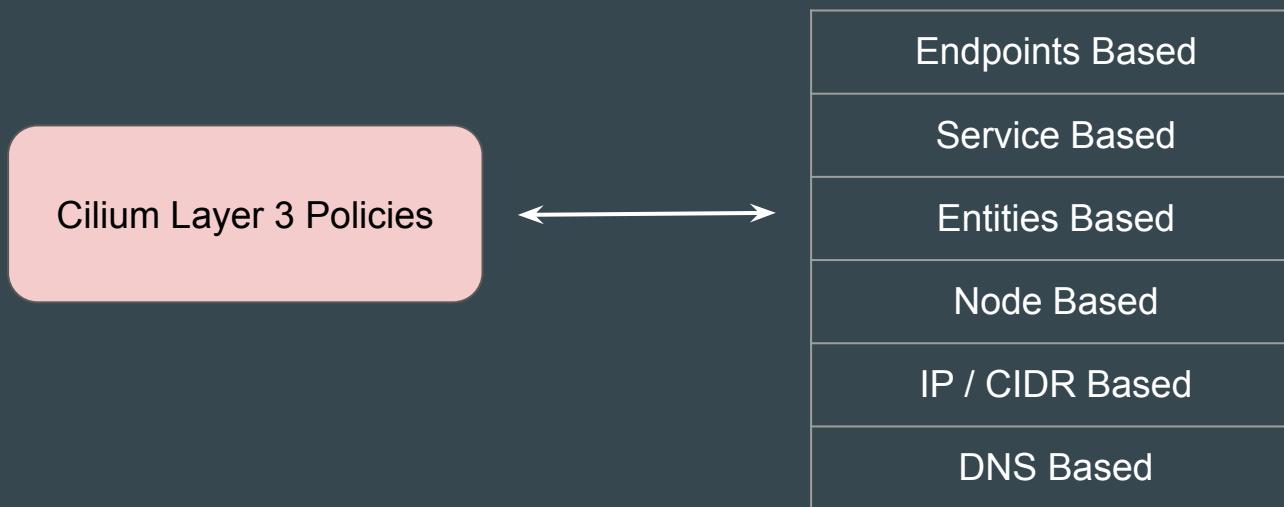


Cilium Network Policies - Layer 3 Rules

Setting the Base

The layer 3 policy establishes the base connectivity rules regarding which endpoints can talk to each other.

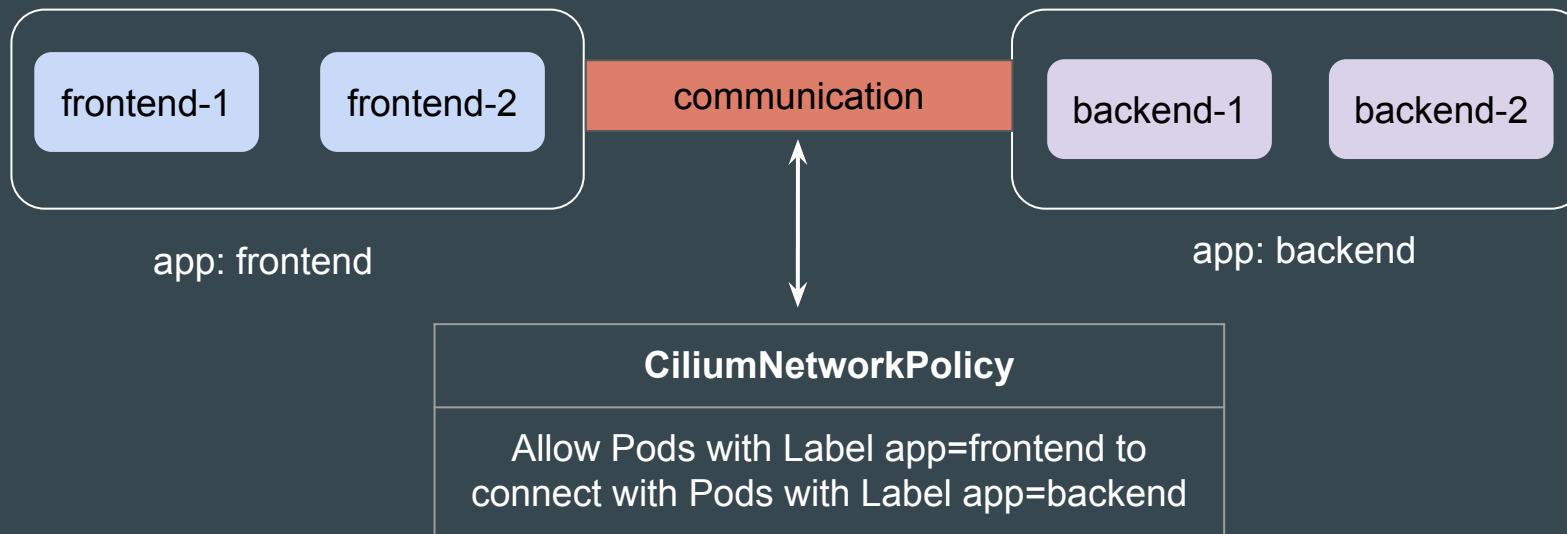
Layer 3 policies can be specified using the following methods:



Types	Description
Endpoints Based	Based on Kubernetes pod labels, allowing or denying traffic between specific pods.
Services based	Policies are applied based on Kubernetes services, controlling traffic based on service names rather than individual pods.
Entities Based	Policies targeting predefined entity groups like "cluster", "host", "world", or "all". Simplifies policy creation for common traffic patterns.
Node based	Policies define traffic rules based on the nodes in the cluster
IP/CIDR based	Policies allow or deny traffic based on specific IP addresses or CIDR blocks

Endpoint Based Policies

These policies are based on Kubernetes pod labels, allowing or restricting traffic between specific pods within a cluster.



Endpoint Based - Example

This policy allows inbound traffic from pods with label of role=frontend to connect with Pods with label of role=backend.

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "allow-from-frontend"
spec:
  endpointSelector:
    matchLabels:
      role: backend
  ingress:
  - fromEndpoints:
    - matchLabels:
        role: frontend
```

Endpoint Based - Example 2

An empty Endpoint Selector will select all endpoints, thus writing a rule that will allow all ingress traffic to an endpoint may be done as follows:

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "allow-all-to-victim"
spec:
  endpointSelector:
    matchLabels:
      role: victim
  ingress:
  - fromEndpoints:
    - {}
```

Cilium Network Policies - Entities Based

Entities Based Policy

Cilium provides predefined entities like world, host, cluster, and remote-node to define network policies

Entities	Description
world	Represents any external (non-cluster) traffic, including the internet.
host	Represents the local Kubernetes node (host network)
remote-node	Represents other Kubernetes nodes in the cluster that are not the local node.
cluster	Represents all workload endpoints within the Kubernetes cluster. Includes all pods across all namespaces
all	Represents all possible endpoints both inside and outside the cluster.

Entities Based Policy - Example 1

Pods can communicate with other pods and services within the cluster.
Pods cannot access external IPs or the internet

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "restrict-egress-to-cluster"
spec:
  endpointSelector: {}  # Apply to all pods
  egress:
    - toEntities:
        - "cluster"
```

Entities Based Policy - Example 2

The following policy will allow traffic from all pods to connect to destination outside of the cluster.

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "restrict-egress-to-cluster"
spec:
  endpointSelector: {}
  egress:
    - toEntities:
        - "world"
```

Cilium Network Policies - Layer 4 Rules

Setting the Base

Layer 4 policy can be specified in addition to layer 3 policies or independently.

It restricts the ability of an endpoint to emit and/or receive packets on a **particular port using a particular protocol**.



Point to Note

If no layer 4 policy is specified for an endpoint, the endpoint is allowed to send and receive on all layer 4 ports and protocols including ICMP.

Example Policy - Port

This policy allow curl-pod to connect outbound on Port 80 for the protocol of TCP.

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: allow-curl-to-nginx-service
spec:
  endpointSelector:
    matchLabels:
      run: curl-pod
  egress:
    - toPorts:
        - ports:
            - port: "80"
              protocol: TCP
```

Cilium Network Policies - DNS Rules

Example Policy - DNS

The following policy allows DNS resolution for domain of kplabs.in and other domain resolutions will be blocked.

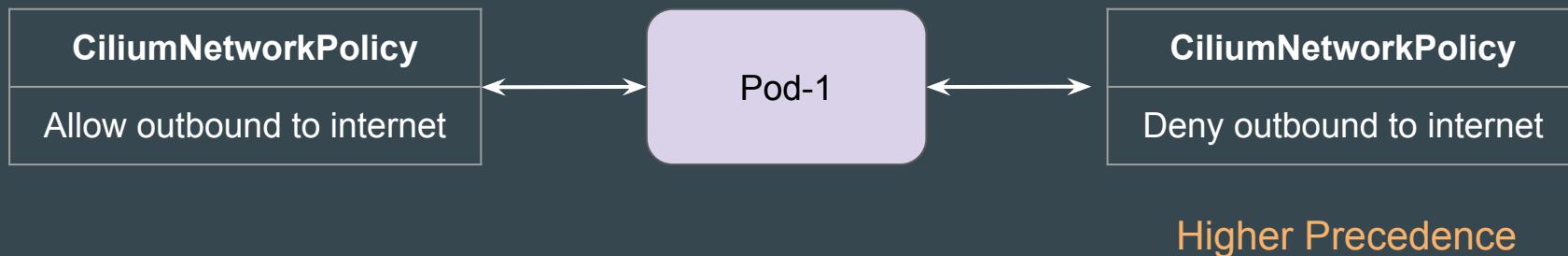
```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "allow-dns-kplabs"
spec:
  endpointSelector: {}
  egress:
    - toPorts:
        - ports:
            - port: "53"
  rules:
    dns:
      - matchName: "kplabs.in"
```

Cilium - Deny Policies

Setting the Base

Cilium's Deny Policies allow you to explicitly block certain network traffic between pods in a Kubernetes cluster.

Deny policies take precedence over allow policies, meaning that if both an allow and deny policy exist, the deny policy will win.



Deny Policies

ingressDeny and egressDeny are features in Cilium Network Policies that **allow you to explicitly deny specific traffic patterns**

ingressDeny	Blocks specific incoming traffic, even if other policies would allow it
egressDeny	Blocks specific outgoing traffic, even if other policies would allow it

Example 1 - ingressDeny

The following policy allows all the entities to connect to the pod with label of app=server **except** the pod with label app=random-pod

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "deny-ingress"
spec:
  endpointSelector:
    matchLabels:
      app: server
  ingress:
    - fromEntities:
        - all
  ingressDeny:
    - fromEndpoints:
        - matchLabels:
            app: random-pod
```

Example 2 - egressDeny

The following CNP blocks connection for pod with label of app=random-pod towards endpoint with label of app=server

All other egress is allowed.

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "deny-egress"
spec:
  endpointSelector:
    matchLabels:
      app: random-pod
  egress:
    - toEntities:
        - all
  egressDeny:
    - toEndpoints:
        - matchLabels:
            app: server
```

Cilium - Transparent Encryption

Understanding the Challenge

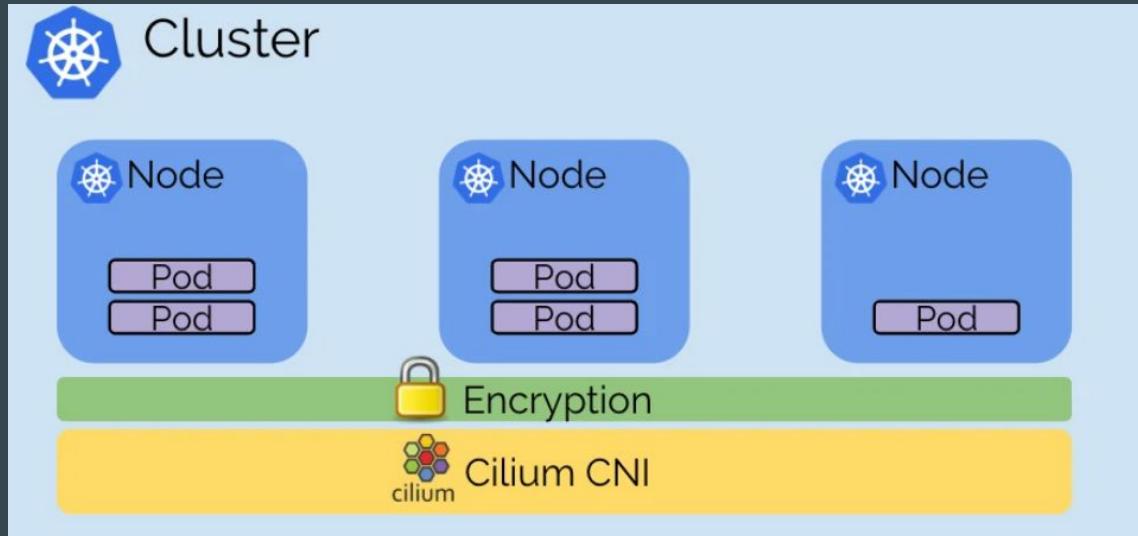
Kubernetes **does not natively support pod-to-pod encryption** for network traffic.

By default, communication between pods in a Kubernetes cluster happens in plaintext unless additional security measures are implemented.



Setting the Base

Cilium supports the **transparent encryption** of Cilium-managed host traffic and traffic between Cilium-managed endpoints either using **IPsec or WireGuard**



Verifying the Results

After transparent encryption is enabled, you can capture the tcpdump traffic to verify the results.

```
root@kind-control-plane:/home/cilium# tcpdump -n -i cilium_vxlan esp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on cilium_vxlan, link-type EN10MB (Ethernet), snapshot length 262144 bytes
11:56:14.499534 IP 10.244.1.139 > 10.244.0.107: ESP(spi=0x00000003,seq=0x1), length 108
11:56:14.500089 IP 10.244.1.139 > 10.244.0.107: ESP(spi=0x00000003,seq=0x2), length 108
11:56:14.521524 IP 10.244.0.107 > 10.244.1.139: ESP(spi=0x00000003,seq=0x1), length 204
11:56:14.522048 IP 10.244.0.107 > 10.244.1.139: ESP(spi=0x00000003,seq=0x2), length 204
```

Security Context - readOnlyRootFilesystem

Setting the Base

`readOnlyRootFilesystem` mounts the container's root filesystem as read-only.

This can mitigate many common attack vectors by preventing unauthorized changes to critical files within the container.

```
apiVersion: v1
kind: Pod
metadata:
  name: readonly-pod
spec:
  containers:
    - name: demo
      image: busybox:1.37
      command: [ "sleep", "1h" ]
      securityContext:
        readOnlyRootFilesystem: true
```

Adding Exception

For temporary file storage within your application, an `emptyDir` volume mounted to a location like `/tmp` provides can be a suitable solution

```
apiVersion: v1
kind: Pod
metadata:
  name: readonly-pod-emptydir
spec:
  containers:
    - name: my-container
      image: busybox:1.37
      command: [ "sleep", "1h" ]
      securityContext:
        readOnlyRootFilesystem: true
      volumeMounts:
        - name: tmp-storage
          mountPath: /tmp
  volumes:
    - name: tmp-storage
      emptyDir: {}
```

When to Use It

It is ideal for containers where the application does not need to modify the root filesystem at runtime.

For example, applications that rely on external volumes for persistent or temporary data storage.

If your application requires writable areas (like /tmp for temporary data), you can explicitly mount these volumes with write permissions while keeping the rest of the filesystem read-only.

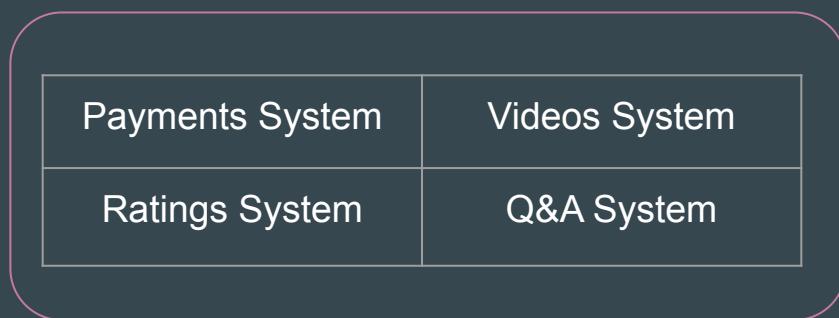
When Not to Use It

Some applications are designed to write logs, cache data, or manage runtime configurations on the root filesystem. In such cases, forcing the root filesystem to be read-only may break functionality.

Monolithic vs Microservices

Setting the Base

A **monolithic architecture** is a traditional software development model that uses one code base to perform multiple business functions.



Single Udemy Application

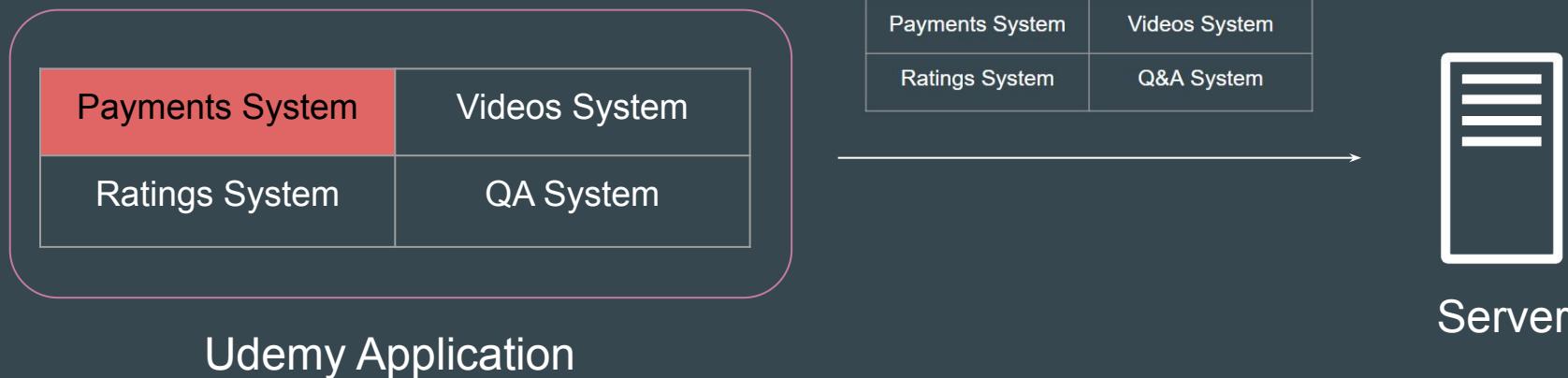


Server

Challenges with Monolithic Approach

If there is a **bug** in any one of the sub-components, we have to redeploy the entire application.

We can't just fix and deploy the 'Payments System' module independently."

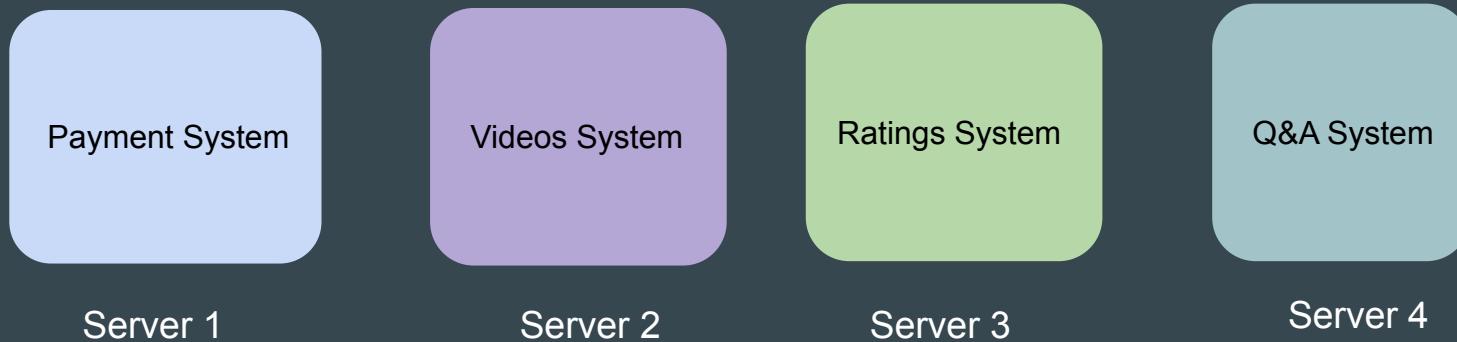


Disadvantages of Monolithic Application

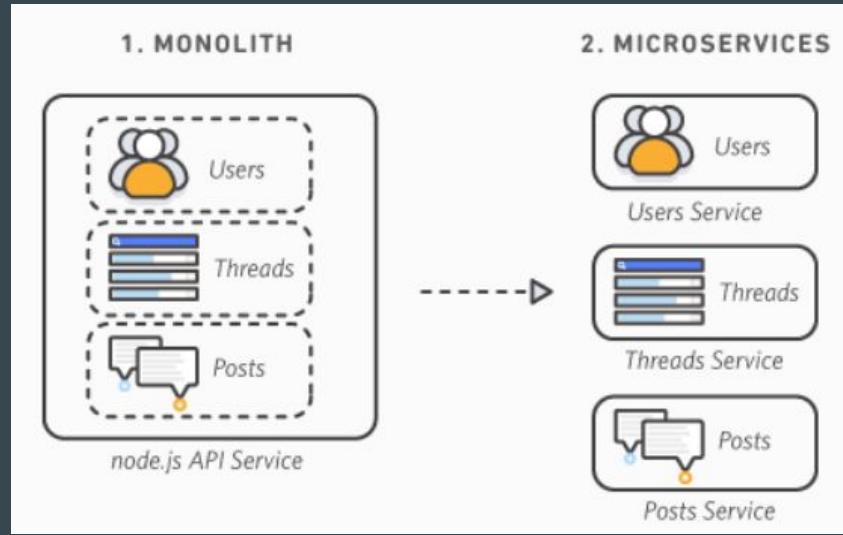
1. Any change requires deploying the entire application, increasing risk and potential downtime.
2. Everything is intertwined, so **scaling** one part (e.g., the payments system module) means scaling the entire app. This leads to inefficiency—wasting resources on underused components.
3. One language/framework for the whole app limits experimentation.
4. Large codebase increases coordination overhead and conflict.

Introducing Microservices Based Architecture

In this architecture, the complete application is divided into a set of individual services, each of which can be deployed and scaled independently.



Reference Screenshot



Advantages of Microservices based architecture

1. Scale only what needs it. If your payments system service is experiencing high load during peak hours, scale just that—without touching the rest.
2. Each service is a small, focused codebase, so teams can work independently. Updates are quicker, and bugs are contained.
3. Deploy changes to a single service without affecting others.
4. Flexibility in Technology and Teams.

Quick Revision Slide

Disadvantages of Monolithic Applications

Monolith: All in One

Tightly Coupled

- Scalability Challenges: Scale everything or nothing.
- Maintenance Complexity: Large codebase leads to bugs.
- Deployment Risks: Whole app downtime on updates.
- Technology Lock-in: Stuck with one stack.
- Fault Isolation Issues: One failure crashes all.
- Team Bottlenecks: Collaboration is hard.

Quick Revision Slide

Slide 2: Advantages of Microservices Architecture

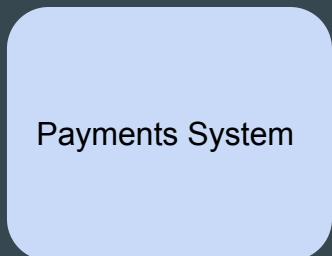


- Improved Scalability: Scale individual services.
- Easier Maintenance: Small, focused codebases.
- Independent Deployments: Zero-downtime updates.
- Better Fault Isolation: Failures are contained.
- Technology Flexibility: Mix languages and tools.
- Enhanced Agility: Faster development and teams.

Challenges with Microservices Architecture

Challenges with Microservices Architecture

Adopting a microservices architecture solves many problems of monolithic systems, but it introduces its own unique set of challenges.



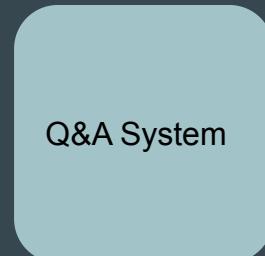
Server 1



Server 2



Server 3

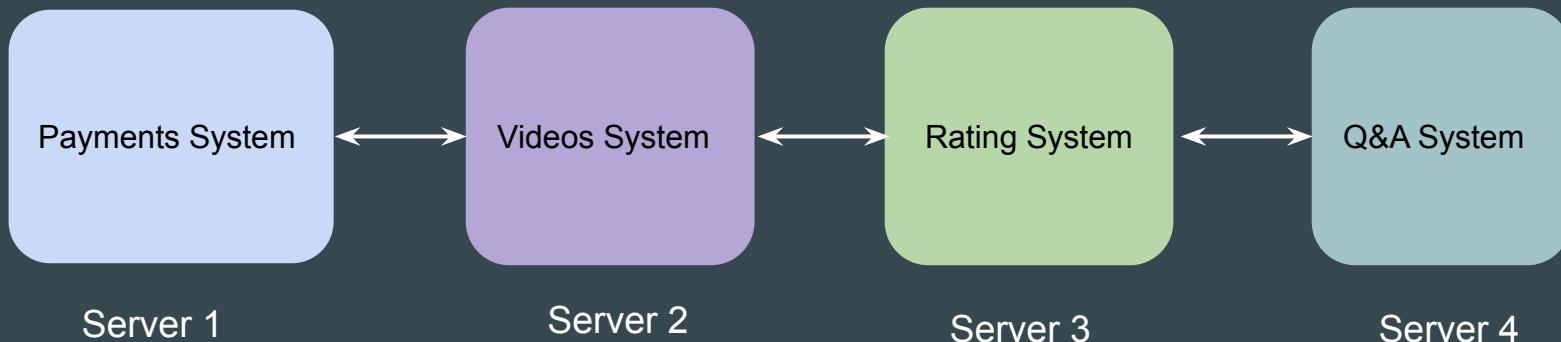


Server 4

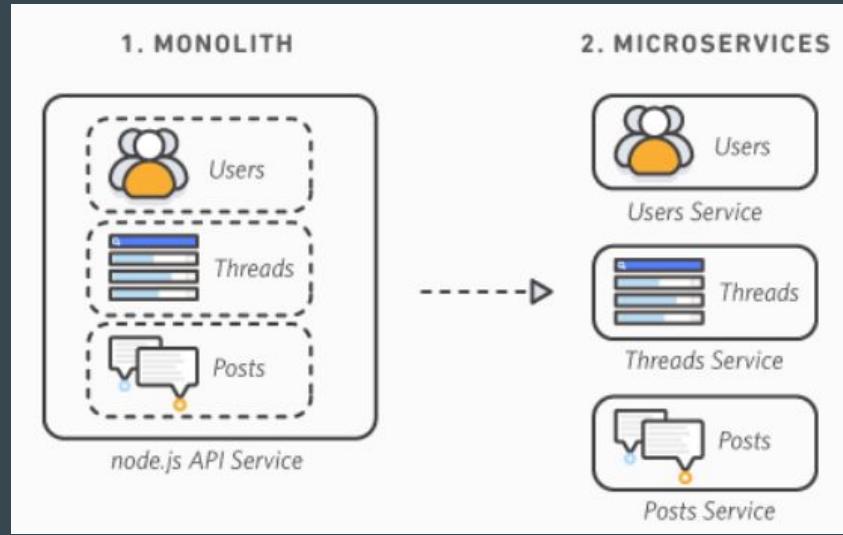
Challenge 1 - Network Connectivity

In a microservices architecture, services are distributed across multiple servers or containers.

A fundamental requirement is **establishing reliable network connectivity** between these services to enable communication.

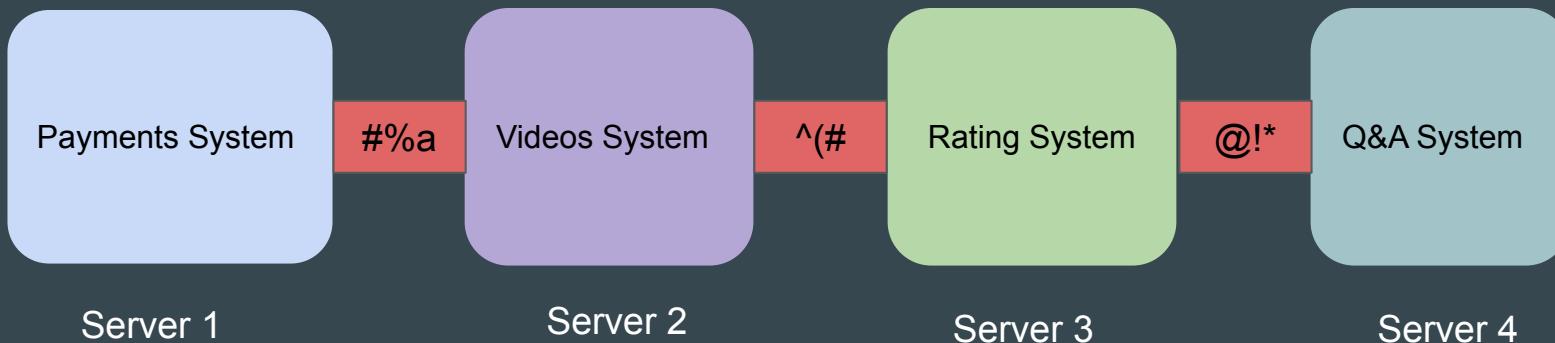


Reference Screenshot



Challenge 2 - Secure Communication

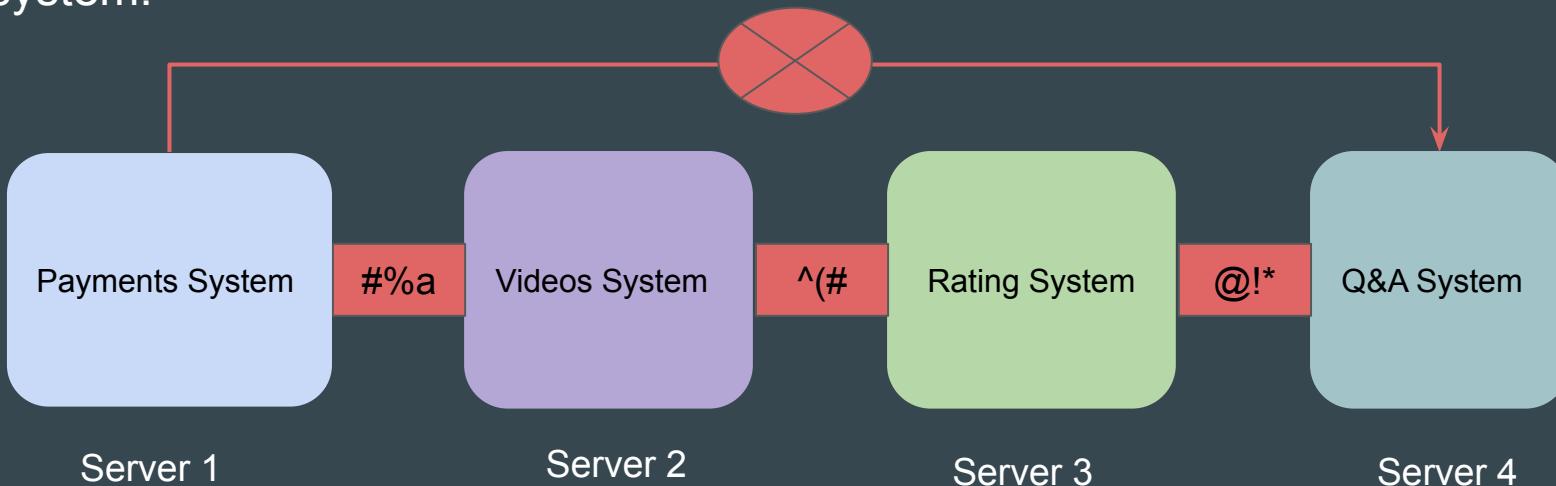
The communication between these services should be encrypted and secured.



Challenge 3 - Network Isolation

Only services that explicitly need to communicate with each other should be permitted to do so.

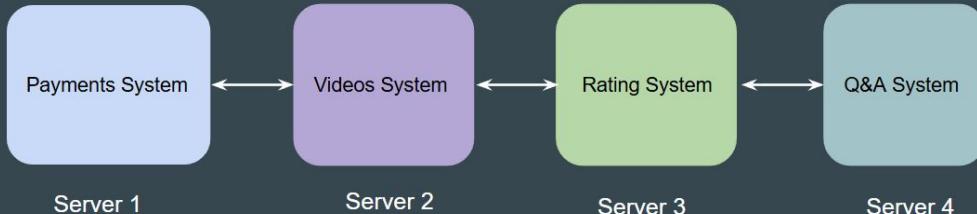
Example: Payments system should not be allowed communication with Q&A system.



Challenge 4 - Service Discovery

Services are ephemeral and dynamic; they can be scaled up or down, and their IP addresses can change frequently.

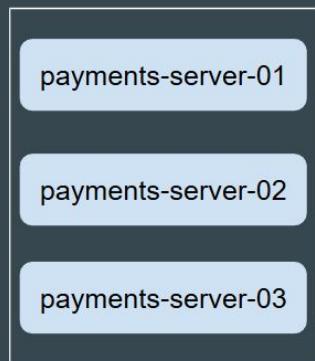
Hardcoding IP addresses is not a viable solution.



Services	IP Address
Payments	172.31.0.4
Videos	172.31.0.5
Ratings	172.31.0.6
Q&A	172.31.0.7

Example - Service Discovery Challenge

Example: Due to high load, the payments system has scaled from 1 server to 3 servers.



Server 4



Server 5



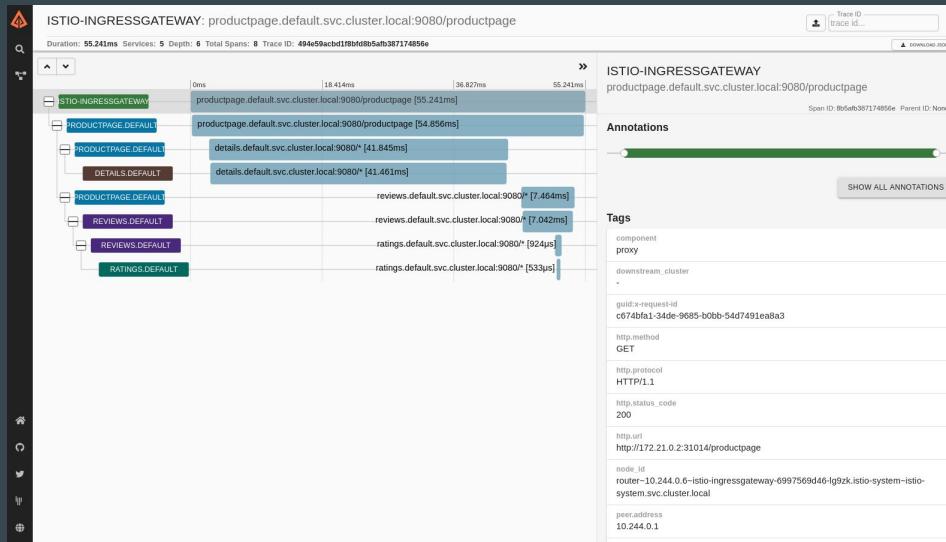
Server 6

Services	IP Address
Payments	172.31.0.4, 172.31.0.8, 172.31.0.10
Videos	172.31.0.5
Ratings	172.31.0.6
Q&A	172.31.0.7

Challenge 5 - Monitoring

With traffic flowing between dozens or hundreds of services, it becomes critical to monitor the health and performance of the entire system.

This requires real-time visibility into inter-service communication to quickly diagnose failures and performance bottlenecks.



Introduction to Service Mesh

Challenges with Microservices Architecture

When your application is broken down into dozens or hundreds of services, managing the communication between them becomes a major hurdle.

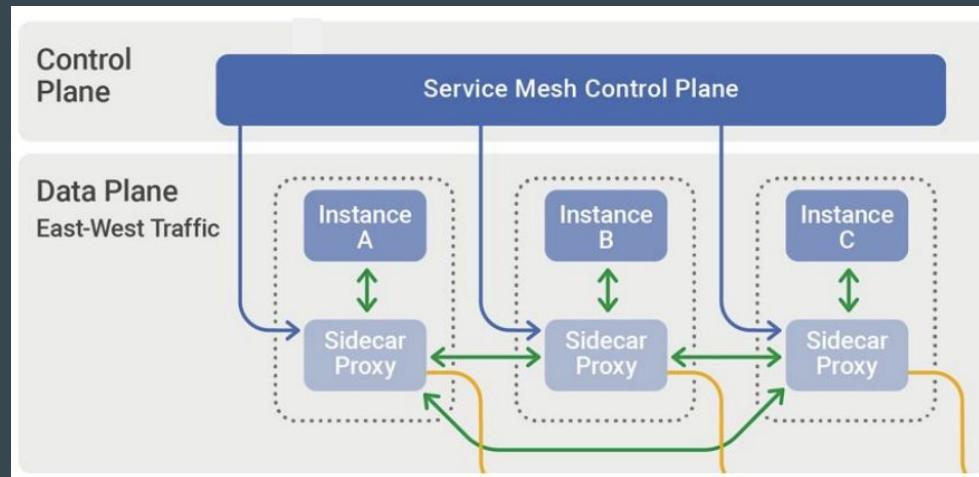
Key challenges include:

Secure Communication	How can we ensure that all data exchanged between services is encrypted and that services can verify each other's identity to prevent man-in-the-middle attacks?
Network Isolation	How to allow only whitelisted service to communicate with other service.
Service Discovery	How does a service find the network location (IP address) of another service it needs to talk to, especially in a dynamic environment where containers are constantly changing?
Monitoring	How do we get a unified view of the health, performance, and dependencies across the entire system to quickly diagnose issues?

Introducing Service Mesh

Service mesh provides traffic management, observability, reliability, and security for microservices.

Services **communicate and handle requests via a proxy**, which is dynamically injected into each pod.

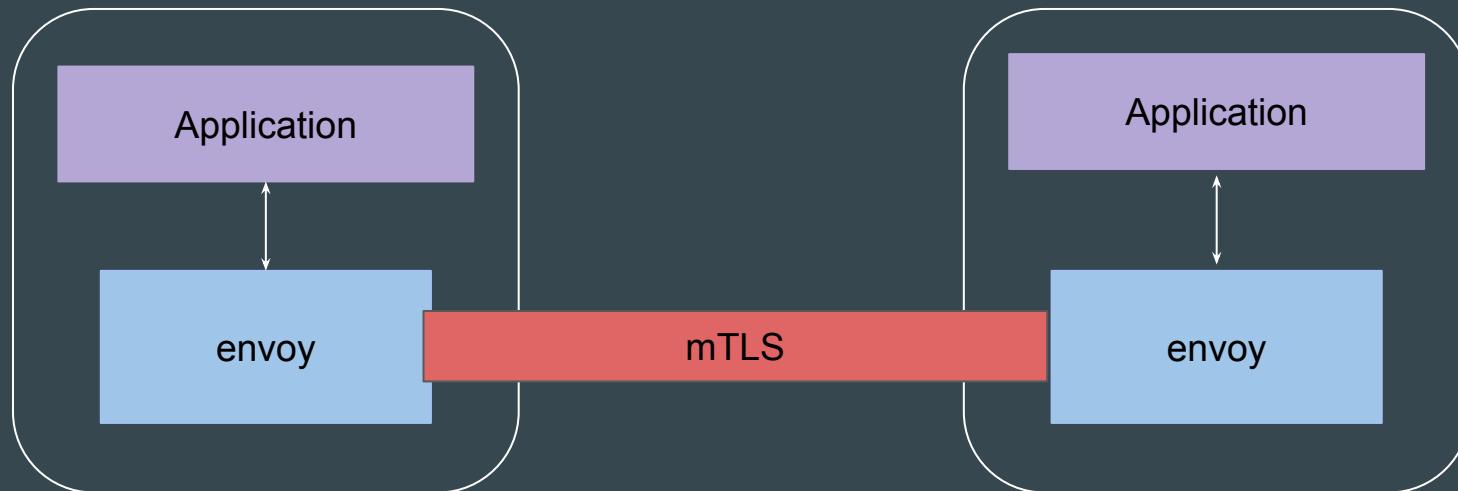


Point to Note

1. The core of a service mesh is the **sidecar proxy**.
2. A lightweight network proxy (like Envoy) is deployed alongside each instance of your application service (e.g., in the same Kubernetes Pod).
3. All network traffic to and from your application is intercepted and routed through this proxy. This allows the service mesh's control plane to manage and observe all traffic without the application needing to be aware of it.

Secure Network Connectivity

A service mesh automates the security of inter-service communication. By intercepting all traffic, the sidecar proxies can enforce security policies centrally.



Service Discovery

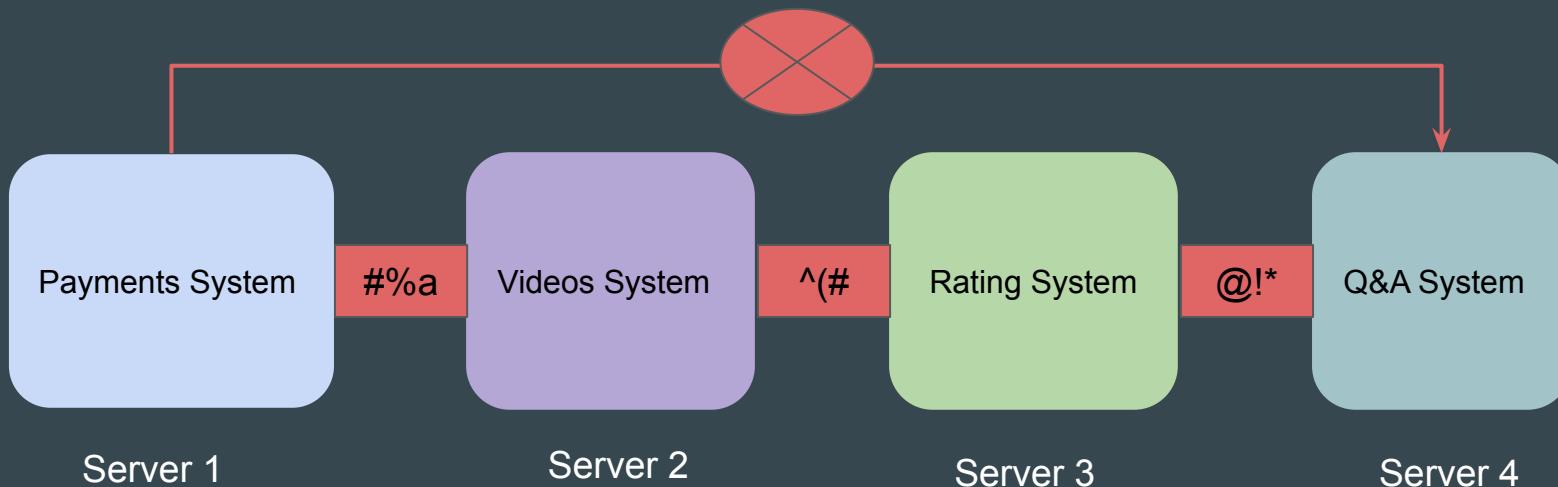
Many tools like Consul provides a dedicated **service discovery feature** that maintains registry with list of IP addresses of all the services.

If a service wants to connect to a messaging service, it can query the registry and find the latest set of IP addresses for the messaging service.



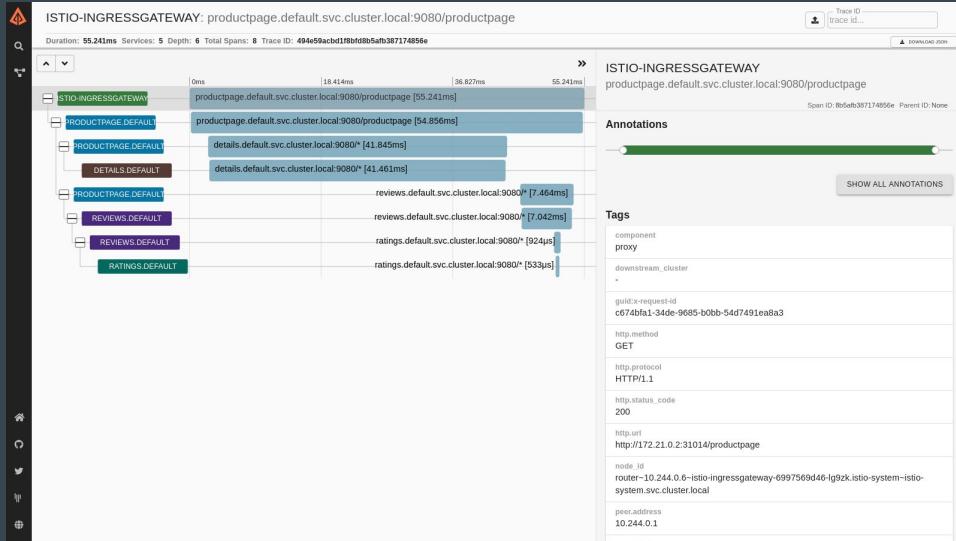
Network Isolation

A service mesh can enable a **zero-trust network model** by allowing you to define fine-grained access policies based on service identity, not just network location.



Monitoring

Because all service-to-service traffic flows through the sidecar proxies, the service mesh is in a unique position to generate detailed telemetry data for the entire application.



Popular Service Mesh Solutions

There are many popular service mesh solutions available in the market.

Some of these include: Istio, Consul, Linkerd

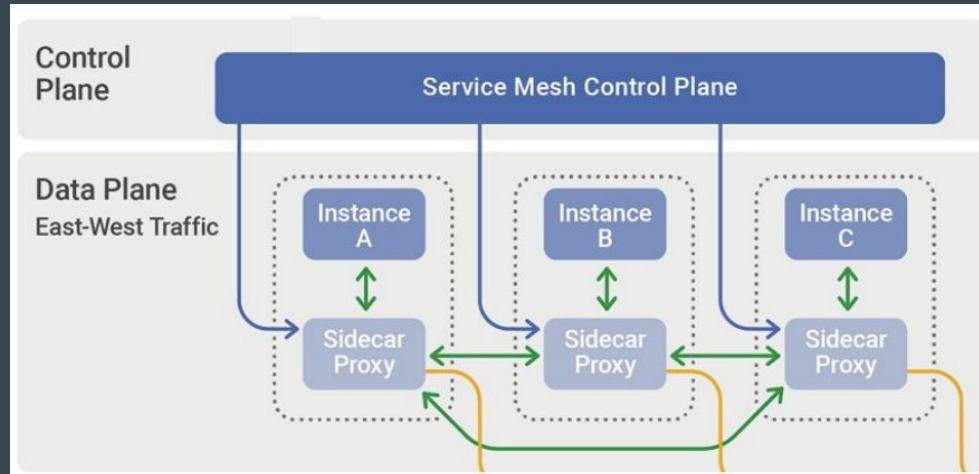


Introduction to Istio

Introducing to Istio

Istio is an **open-source service mesh platform** designed to manage microservices in cloud-native environments, particularly Kubernetes.

Istio uses a proxy to intercept all your network traffic, allowing a broad set of application-aware features.



Point to Note

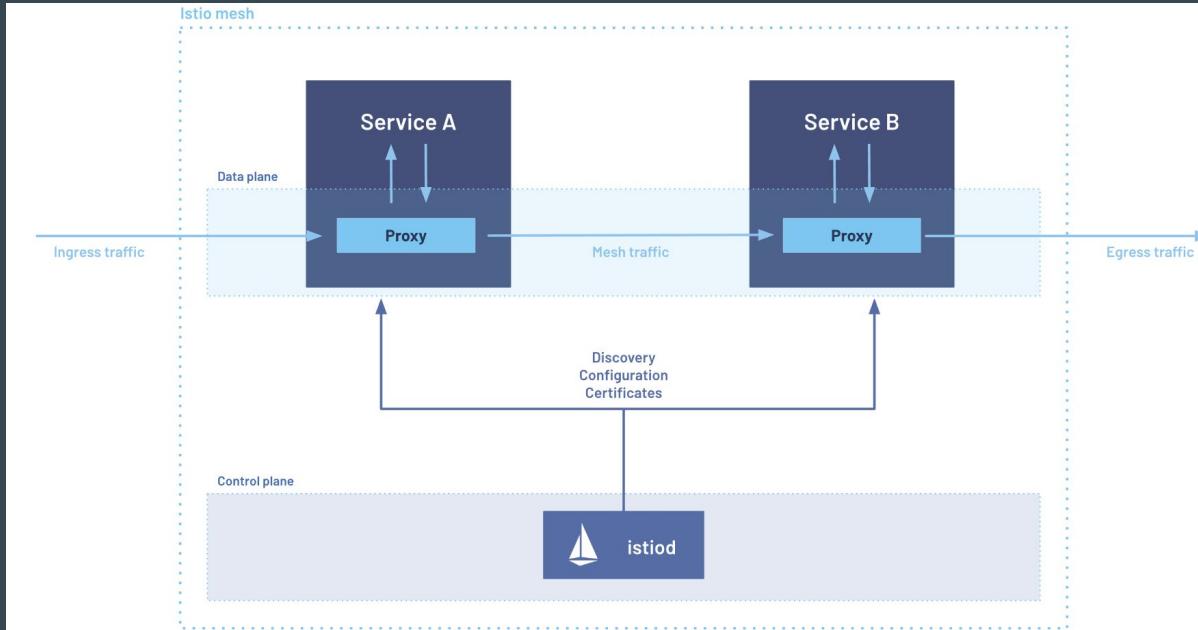
Founded by Google, IBM and Lyft in 2016, Istio is a graduated project in the Cloud Native Computing Foundation alongside projects like Kubernetes and Prometheus.

Features of Istio

Important Features	Description
Secure by default	Istio provides a market-leading zero-trust solution based on workload identity, mutual TLS, and strong policy controls.
Increase observability	Istio generates telemetry within the service mesh, enabling observability on service behavior. It integrates with APM systems including Grafana and Prometheus to deliver insightful metrics for operators to troubleshoot, maintain, and optimize applications.
Manage traffic	Istio simplifies traffic routing and service-level configuration, allowing easy control over flow between services and setup of tasks like A/B testing, canary deployments, and staged rollouts with percentage-based traffic splits.

High-Level Architecture

An Istio service mesh is logically split into a data plane and a control plane.



Control Plane and Data Plane

Data Plane	<p>The data plane is composed of a set of intelligent proxies (Envoy) deployed as sidecars.</p> <p>These proxies mediate and control all network communication between microservices.</p> <p>They also collect and report telemetry on all mesh traffic.</p>
Control Plane	<p>The control plane manages and configures the proxies to route traffic.</p>

Envoy

Istio uses an **extended version of the Envoy proxy**.

Envoy is a high-performance proxy developed in C++ to mediate all inbound and outbound traffic for all services in the service mesh.

Many built-in features

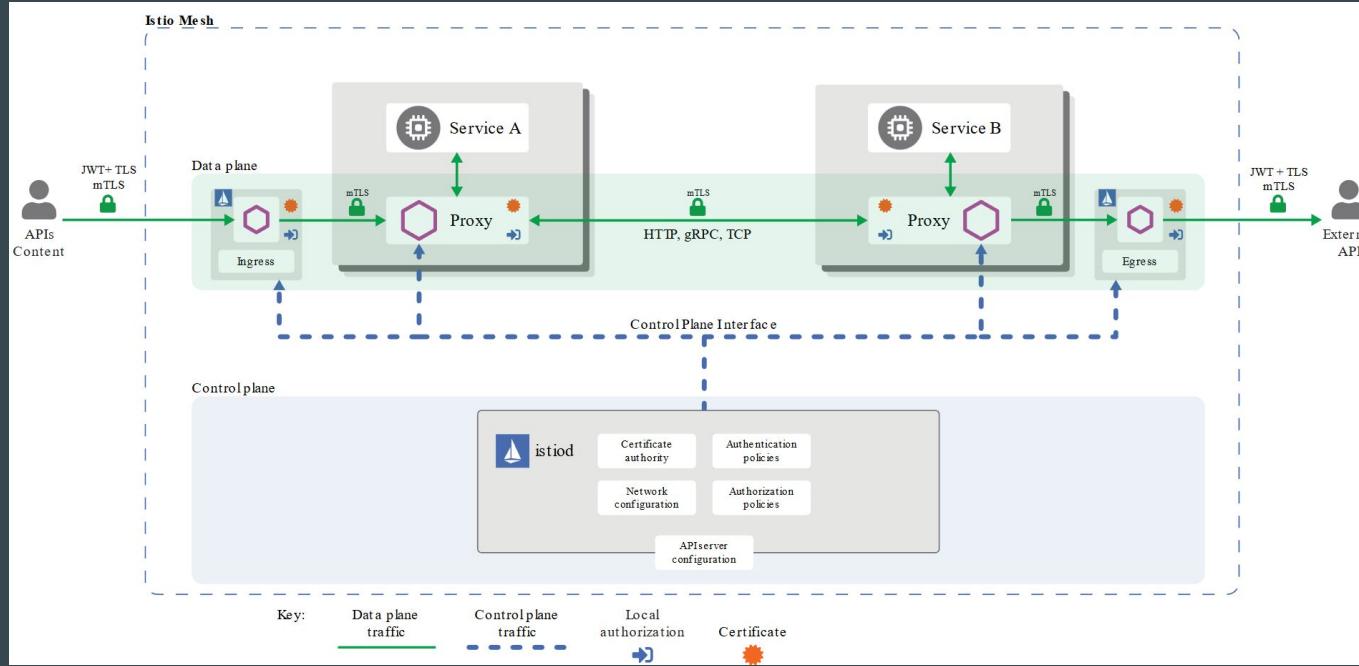
Dynamic service discovery	Load balancing	TLS termination
Health checks	Fault injection	Rich metrics

Point to Note

The sidecar proxy model also allows you to add Istio capabilities to an existing deployment without requiring you to rearchitect or rewrite code.

Security

Istiod acts as a Certificate Authority (CA) and generates certificates to allow secure mTLS communication in the data plane.



Monitoring

With Istio, you gain monitoring of the traffic between microservices by default.

You can use the Istio Dashboard for monitoring your microservices in real time.

The screenshot shows the Istio Dashboard interface. At the top, there are tabs: Overview (selected), Traffic, Logs, Inbound Metrics, Outbound Metrics, and Envoy. The main area has several sections:

- Service Details:** Shows a service named "bookinfo-gateway-istio" with labels: gateway.istio.io/managed=istio.io-gateway-controller, gateway.networking.k8s.io/gateway-name=bookinfo-gateway-istio, and More labels... (with a help icon).
- Annotations:** Shows annotations A and S for the service.
- Pods:** A table showing a single pod named "bookinfo-gateway-istio-6cb97cb689-n8b49" in the default revision.
- Traffic Flow:** A diagram showing a flow from a blue circle (labeled "bookinfo-gateway-istio") to a green triangle (labeled "productpage") with metrics: 0.01rps and 42ms.

Istio - Installing the Sidecar

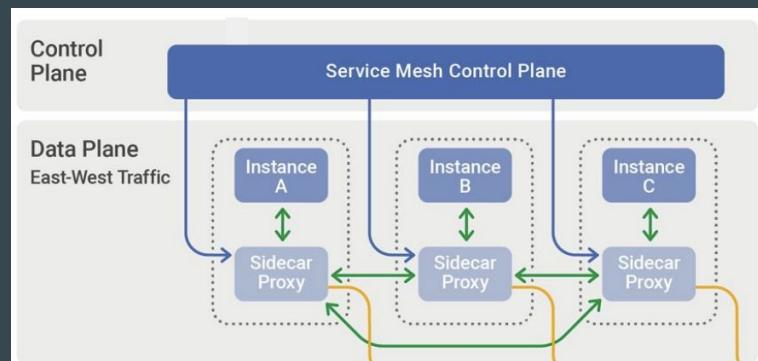
Setting the Base

In order to take advantage of all of Istio's features, pods in the mesh **must be running** an Istio sidecar proxy.

Two primary ways of injecting the Istio sidecar into a pod

Enable automatic Istio sidecar injection in the pod's namespace

Manually using the `istioctl` command



Point to Note

If you are not sure which one to use, automatic injection is recommended.

Enabling Automatic Injection

When you set the `istio-injection=enabled` label on a namespace and the injection webhook is enabled, any new pods that are created in that namespace will automatically have a sidecar added to them.

Automatic injection occurs at the pod-level.

Mutual TLS with Istio

Revising Basics of Mutual TLS

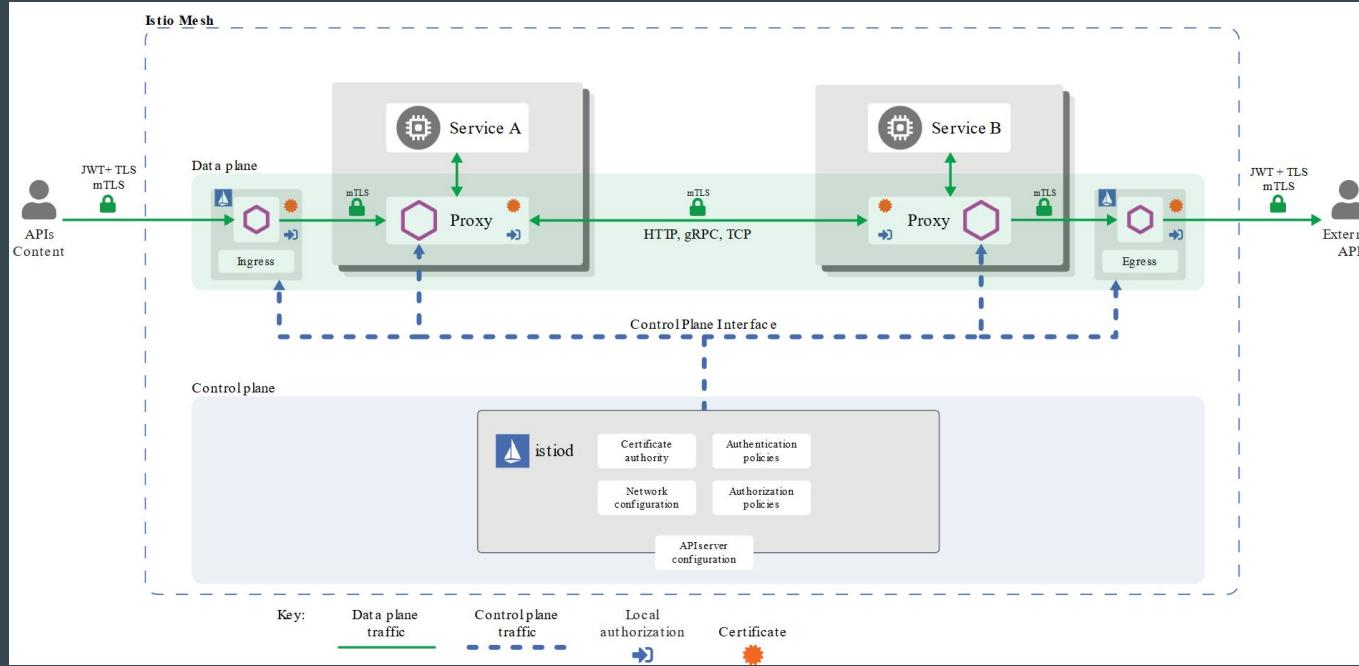
Mutual TLS (mTLS) is a security protocol that ensures both the client and the server verify each other's identities before establishing a connection.

Both the client and the server have digital certificates issued by a trusted Certificate Authority (CA)



Istio and Certificate Authority

Istiod can act as a **Certificate Authority** (CA) and generates certificates to allow secure mTLS communication in the data plane.



Reference Screenshot

```
root@k8s:~# istioctl proxy-config secret curl-pod-02
RESOURCE NAME      TYPE          STATUS    VALID CERT    SERIAL NUMBER
BEFORE
default           Cert Chain   ACTIVE    true        e0c1e1aef10ac39bf423ab64be5fb0a4
-11-05T06:10:08Z
ROOTCA            CA           ACTIVE    true        98d7dc7acf0732a1601358ea49a1ccdd
-11-04T16:35:11Z
```

Peer Authentication - Namespace Level

In sidcar mode, PeerAuthentication determines whether or not mTLS is allowed or required for connections to an Envoy proxy sidecar.

Policy to require mTLS traffic for all workloads under namespace prod

```
apiVersion: security.istio.io/v1
kind: PeerAuthentication
metadata:
  name: default
  namespace: prod
spec:
  mtls:
    mode: STRICT
```

Peer Authentication - Mesh Level

For mesh level, put the PeerAuthentication related policy in root-namespace according to your Istio installation.

```
apiVersion: security.istio.io/v1
kind: PeerAuthentication
metadata:
  name: default
  namespace: istio-system
spec:
  mtls:
    mode: STRICT
```

Overview of AppArmor

Revising DAC

Discretionary access control (DAC) allows restricting access to objects based on the identity of subjects and/or groups to which they belong.

```
root@kubeadm:~# ls -l /etc/kubernetes/
total 44
-rw----- 1 root root 5658 Mar  7 17:11 admin.conf
-rw----- 1 root root 5682 Mar  7 17:11 controller-manager.conf
-rw----- 1 root root 1974 Mar  7 17:12 kubelet.conf
drwxrwxr-x 2 root root 4096 Mar 10 04:06 manifests
drwxr-xr-x 3 root root 4096 Mar  7 17:11 pki
-rw----- 1 root root 5630 Mar  7 17:11 scheduler.conf
-rw----- 1 root root 5682 Mar  7 17:11 super-admin.conf
```

Challenges with DAC

DAC allows programs to inherit the full permissions of the user running them. If a user can access sensitive files, any program they run (including malware) can access those same files.

Sample Use-Case

You have a binary file that performs some basic operation on server like deleting old log files to cleanup resources.

Suddenly you have seen that binary file is connecting to internet and sending network traffic.



Mandatory Access Controls

Mandatory Access Control (MAC) is a security model in which access to resources is strictly regulated by a central authority based on predefined security policies.

Two important concept: Confined (Restricted) and Unconfined (Not Restricted)

Process X

Confined

Process Z

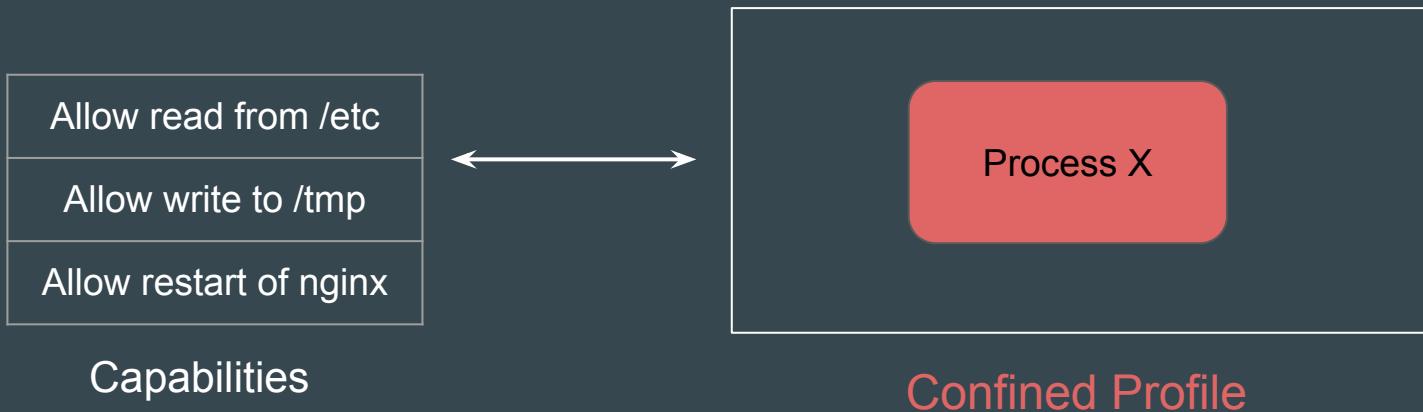
Not Confined

Confined Process

Confined Processes are restricted.

Everything that process intends to do must be listed in a profile.

If that capability is not listed in the profile, the process will not be allowed to run that.



Primary Modes of AppArmor

Modes	Description
Enforce	Actively enforces the defined AppArmor security profile.
Complain	Violations are logged, but the application runs normally without restrictions.
Unconfined	The application runs without AppArmor restrictions.

AppArmor and Kubernetes

Setting the Base

Kubernetes allows you to apply AppArmor profiles to Pods and containers

```
securityContext:  
  appArmorProfile:  
    type: <profile_type>
```

Profile Types Available

Profile Type	Description
RuntimeDefault	To use the runtime's default profile
LocalHost	Uses a custom security profile stored on the node's filesystem
Unconfined	To run without AppArmor

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-apparmor
spec:
  securityContext:
    appArmorProfile:
      type: Localhost
      localhostProfile: k8s-apparmor-example-deny-write
  containers:
  - name: hello
    image: busybox
    command: [ "sh", "-c", "echo 'Hello AppArmor!' && sleep 1h" ]
```

Open Container Initiative

Let's Standardize

Importance of Standardization

In an organization, if image standardization is not set, different developers will use different set of images.

This leads to challenges in troubleshooting, as well as security.



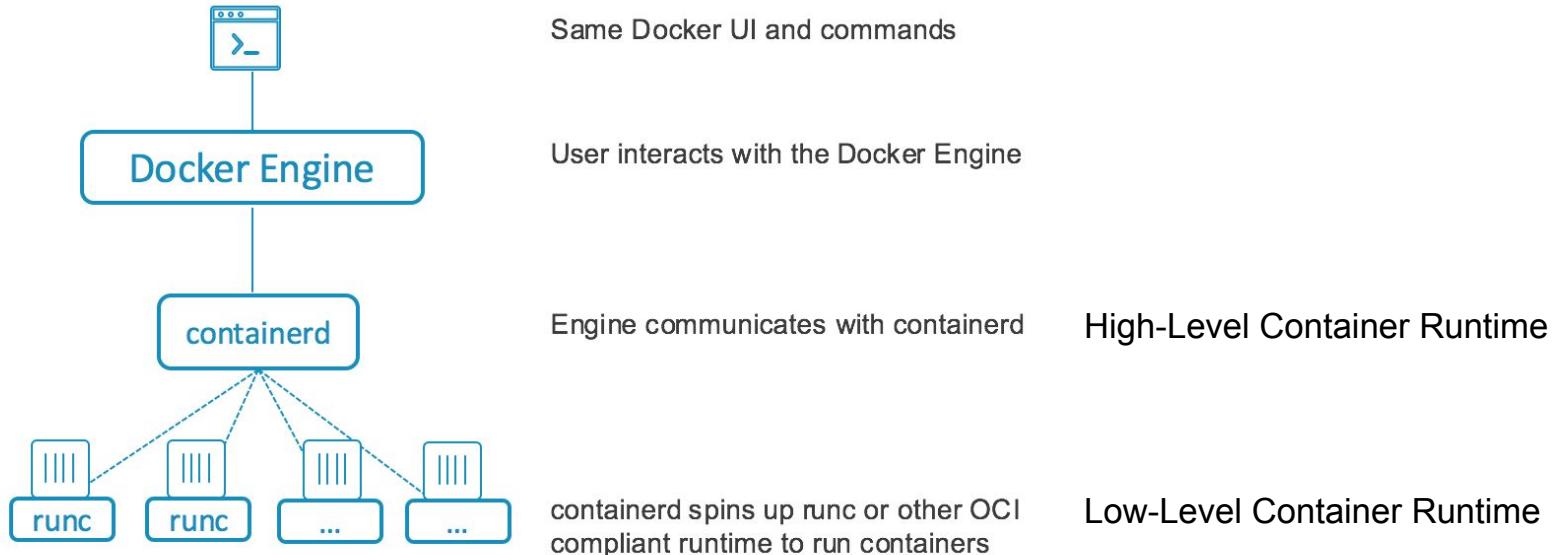
Open Container Initiative

The Open Container Initiative (OCI) is a Linux Foundation project to design open standards for containers.

There are two important specifications

Specification	Description
Image Specification	Defines how to create an OCI Image, which includes an image manifest, a filesystem (layer) serialization, and an image configuration.
Runtime Specification	defines how to run the OCI image bundle as a container.

Docker Workflow



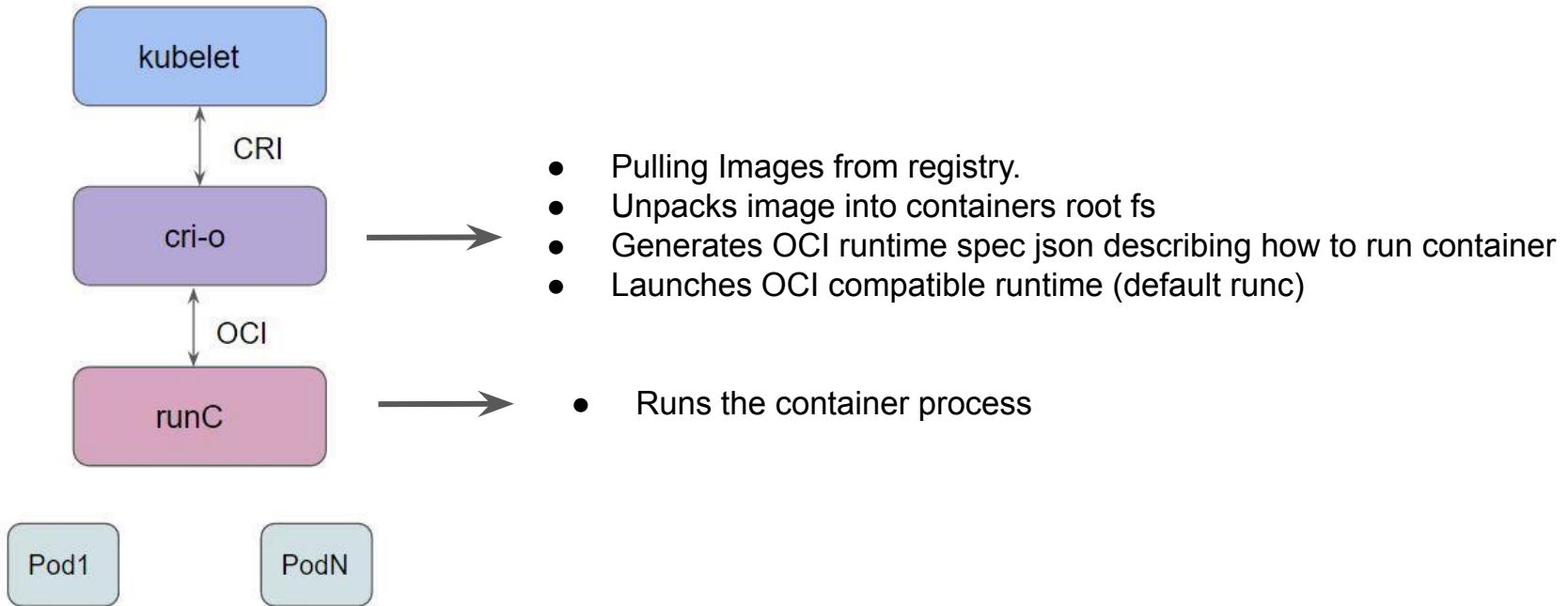
Container Runtimes

A container runtime is software that executes containers and manages container images on a node

There are multiple container runtimes available. Some of these include:

- Docker
- containerd
- Cri-o
- Podman

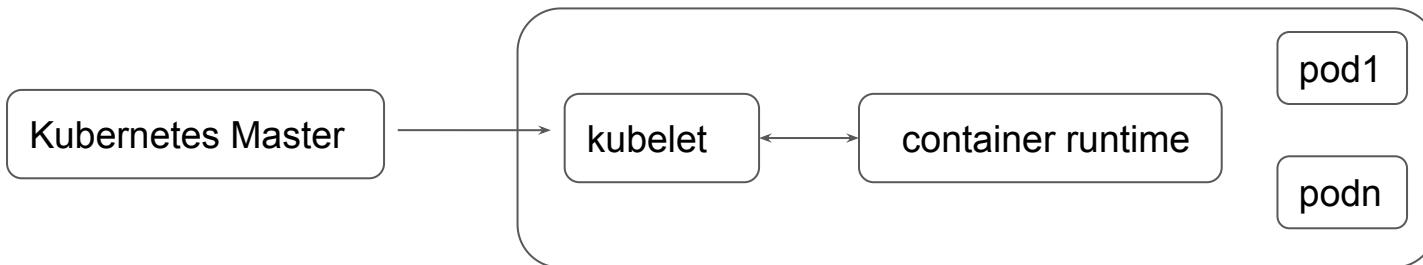
High-Level and Low Level Runtimes



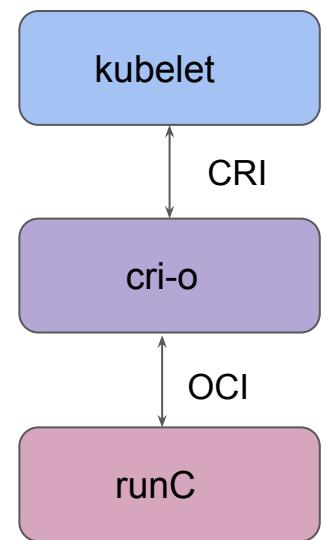
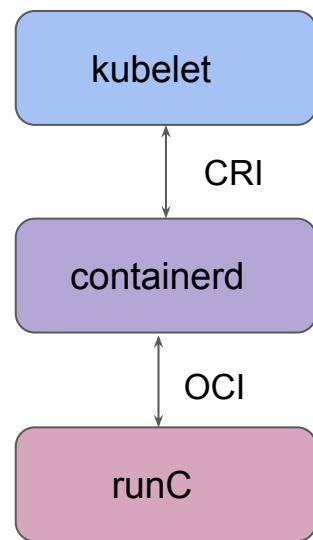
Container Runtime Interface

Each container runtime has its own strengths.

K8s uses Container Runtime Interface which is a plugin interface which enables kubelet to use a wide variety of container runtimes without the need to recompile.



Flexibility for Container Runtimes



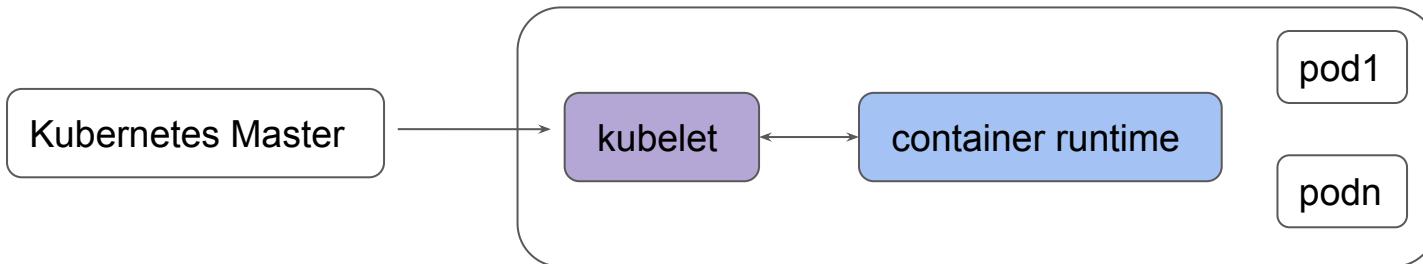
Container Runtime Interface

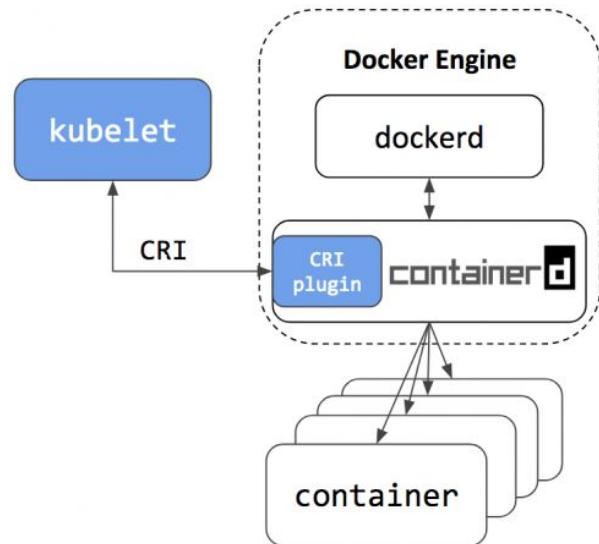
CRI

Container Runtime Interface

Each container runtime has its own strengths.

K8s uses Container Runtime Interface which is a plugin interface that enables kubelet to use a wide variety of container runtimes without the need to recompile.



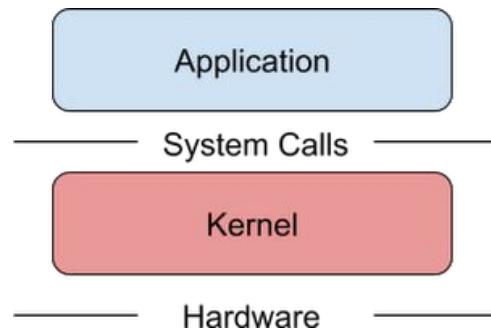


Container Sandbox

Sandboxing

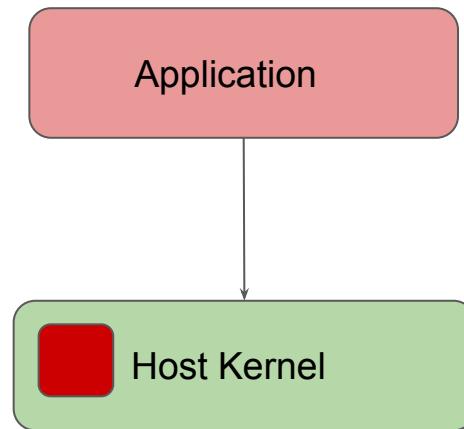
Basic Architecture

Applications that run in traditional Linux containers access system resources in the same way that regular (non-containerized) applications do: by making system calls directly to the host kernel.



Use Cases - Bugs

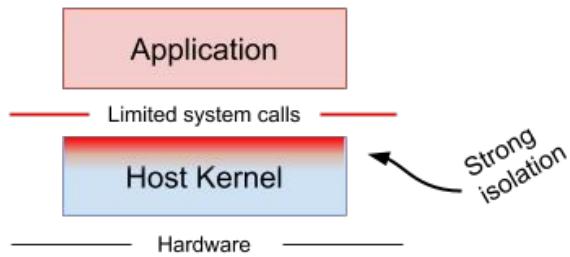
If there are certain bugs at the Kernel level, the application can take advantage of it to achieve various use-cases like privilege escalation, and others.



Seccomp

Kernel features like seccomp filters can provide better isolation between the application and host kernel, but they require the user to create a predefined whitelist of system calls.

In practice, it's often difficult to know which system calls will be required by an application beforehand.



Container Sandbox

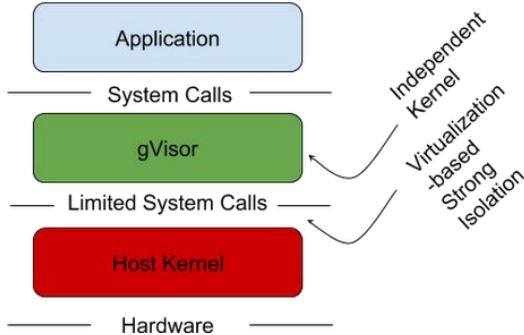
Sandboxing is a approach that enforces a level of isolation between the software running on the machine and the underlying operating system.

In practice, it's often difficult to know which system calls will be required by an application beforehand.

Overview of gVisor

The core of gVisor is a kernel that runs as a normal, unprivileged process that supports most Linux system calls.

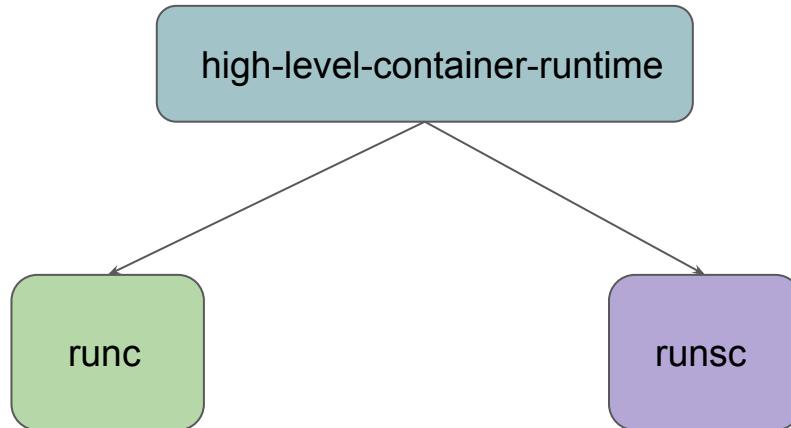
gVisor intercepts all system calls made by the application, and does the necessary work to service them thus providing a strong isolation boundary.



Exploring gVisor

It primarily replaces runc (default runtime) which had few serious vulnerabilities

It comes with an OCI runtime named runsc and hence can act as a drop-in replacement to the runc.



Exploring dmesg

dmesg (diagnostic message) is a command on most Unix-like operating systems that prints the message buffer of the kernel.

```
root@nginx:/# dmesg
[ 0.000000] Starting qVisor...
[ 0.485125] Moving files to filing cabinet...
[ 0.886586] Feeding the init monster...
[ 1.125605] Daemonizing children...
[ 1.327612] Verifying that no non-zero bytes made
[ 1.395538] Searching for socket adapter...
[ 1.737309] Synthesizing system calls...
[ 2.108302] Waiting for children...
[ 2.290812] Adversarially training Redcode AI...
[ 2.402811] Creating process schedule...
[ 2.773946] Creating cloned children...
[ 3.250078] Ready!
```

gVisor based POD

```
root@nginx2:/# dmesg
[ 0.000000] Linux version 5.4.0-1029-aws (buildd@lcy01-amd64-04)) #30-Ubuntu SMP Tue Oct 20 10:06:38 UTC 2020 (Ubuntu 5.4.0-1029-aws)
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.4.0-1029-aws root=UUID=6a2a2a2a-2a2a-4a2a-a2a2-a2a2a2a2a2a ro=quiet
[ 0.000000] Kernel supported cpus:
[ 0.000000]   Intel GenuineIntel
[ 0.000000]   AMD AuthenticAMD
[ 0.000000]   Hygon HygonGenuine
[ 0.000000]   Centaur CentaurHauls
[ 0.000000]   zhaoxin Shanghai
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating-point'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[ 0.000000] x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]: 144
[ 0.000000] x86/fpu: Enabled xstate features 0x7, context size 144
[ 0.000000] BIOS-provided physical RAM map:
```

Default runtime class pod

Challenges

Generally organization makes use of sandboxes like gVisor for the applications that are not entirely trusted (cloning repo from GitHub and running that application)

It can lead to certain performance degradation.

RunTimeClass

RuntimeClass is a feature for selecting the container runtime configuration.

You can set a different RuntimeClass between different Pods to provide a balance of performance versus security.



default, runc

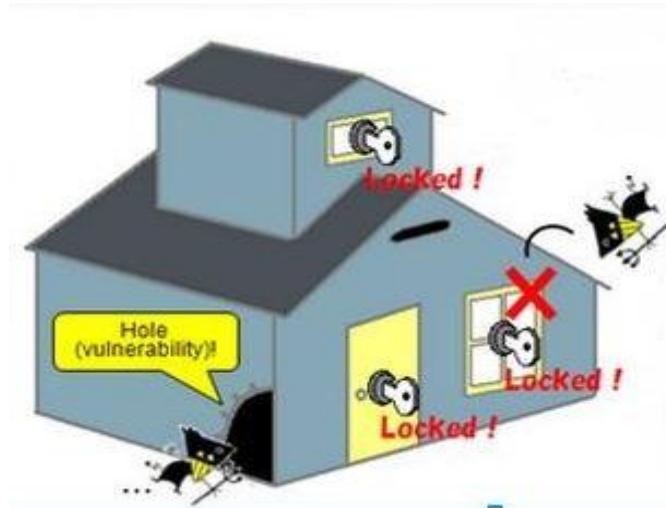


gvisor,runsc

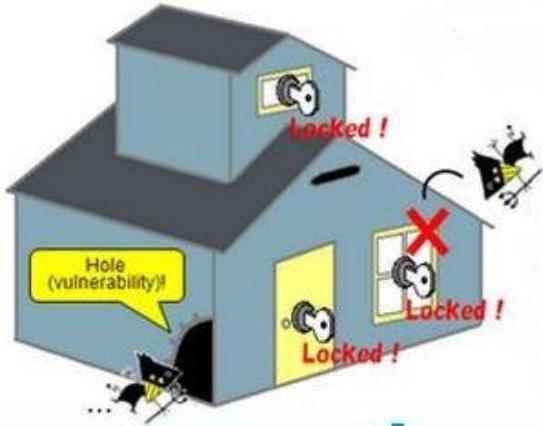
Vulnerability, Exploit, Payload

Ethical Hacking Terminology

The simple house terminology



The Answers



Vulnerability :- Hole on the Side of the House

Exploit :- The Robber

Payload :- What Robber does inside the house

Security Terminology



Vulnerability :- Bad Software Code

Exploit :- Program that exploits code to get inside.

Payload :- Stealing Data, Ransomwares etc.

Scan Result of Vulnerability Scanners

Internal Scan
CURRENT RESULTS: TODAY AT 9:55 PM

Configure Audit Trail Launch Export

Hosts > 127.0.0.1 > Vulnerabilities 164

<input type="checkbox"/>	Severity ▲	Plugin Name	Plugin Family	Count
<input type="checkbox"/>	Critical	Ubuntu 12.04 LTS / 14.04 LTS / 16.04 LTS / 16.10 : firefox regression (USN-3216-2)	Ubuntu Local Security Checks	1
<input type="checkbox"/>	Critical	Ubuntu 12.04 LTS / 14.04 LTS / 16.04 LTS / 16.10 : icu vulnerabilities (USN-3227-1)	Ubuntu Local Security Checks	1
<input type="checkbox"/>	Critical	Ubuntu 12.04 LTS / 14.04 LTS / 16.04 LTS / 16.10 : libxml2 vulnerabilities (USN-3235-1)	Ubuntu Local Security Checks	1
<input type="checkbox"/>	Critical	Ubuntu 12.04 LTS / 14.04 LTS / 16.04 LTS : python2.7, python3.2, python3.4, python3.5 vulnerabilities (USN-3134-1) (httpoxy)	Ubuntu Local Security Checks	1
<input type="checkbox"/>	High	PostgreSQL Default Unpassworded Account	Databases	1
<input type="checkbox"/>	High	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 / 16.04 LTS : nspr vulnerability (USN-3028-1)	Ubuntu Local Security Checks	1
<input type="checkbox"/>	High	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 / 16.04 LTS : nss vulnerability (USN-3029-1)	Ubuntu Local Security Checks	1
<input type="checkbox"/>	High	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 / 16.04 LTS : thunderbird vulnerabilities (USN-3023-1)	Ubuntu Local Security Checks	1
<input type="checkbox"/>	High	Ubuntu 12.04 LTS / 14.04 LTS / 15.10 : pidgin vulnerabilities (USN-3031-1)	Ubuntu Local Security Checks	1

Container Security Scanning

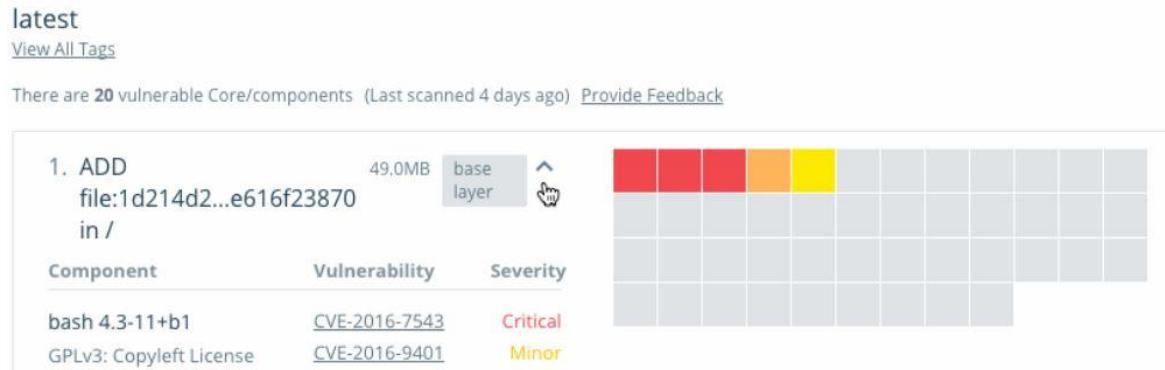
Container Security



Getting Started

Docker Containers can have security vulnerabilities.

If blindly pulled and if containers are running in production, it can result in breach.



Overview of Security Scanning in DTR

DTR allows us to perform security scan for the containers.

These scan can perform “On Push” or even manually.

admin / webserver								
	Info	Tags	Webhooks	Promotions	Pruning	Mirrors	Settings	Activity
	Image	Os/Arch	Image ID	Size (Compressed)	Signed	Last Pushed	Vulnerabilities	
<input type="checkbox"/>	ubuntu	linux / amd64	9361ce633ff1	43.56 MB		12 minutes ago by admin	5 Critical 31 Major 18 Minor	View details
<input type="checkbox"/>	v1	linux / amd64	d8233ab899d4	755.9 kB		29 minutes ago by admin	Clean	View details

Trivy

Let's Scan with Trivy

Overview of Trivy

Trivy is a open-source based simple and comprehensive vulnerability Scanner for containers

```
bash-3.2$ trivy knqyf263/test-image:1.2.3
2019-05-13T15:19:03.912+0000  INFO  Updating vulnerability database...
2019-05-13T15:19:05.983+0000  INFO  Detecting Alpine vulnerabilities...
2019-05-13T15:19:06.000+0000  INFO  Updating npm Security DB...
2019-05-13T15:19:07.048+0000  INFO  Detecting nodejs vulnerabilities...
2019-05-13T15:19:07.048+0000  INFO  Updating pipenv Security DB...
2019-05-13T15:19:08.507+0000  INFO  Detecting pipenv vulnerabilities...
2019-05-13T15:19:08.508+0000  INFO  Updating python Security DB...
2019-05-13T15:19:09.500+0000  INFO  Detecting builder vulnerabilities...
2019-05-13T15:19:09.575+0000  INFO  Updating cargo Security DB...
2019-05-13T15:19:10.441+0000  INFO  Detecting cargo vulnerabilities...
2019-05-13T15:19:10.441+0000  INFO  Updating composer Security DB...
2019-05-13T15:19:11.649+0000  INFO  Detecting composer vulnerabilities...

knqyf263/test-image:1.2.3 (alpine 3.7.1)
=====
Total: 26 (UNKNOWN: 0, LOW: 3, MEDIUM: 16, HIGH: 5, CRITICAL: 2)

+-----+-----+-----+-----+-----+
| LIBRARY | VULNERABILITY ID | SEVERITY | INSTALLED VERSION | FIXED VERSION | TITLE |
+-----+-----+-----+-----+-----+
| curl   | CVE-2018-14618 | CRITICAL | 7.61.0-r0    | 7.61.1-r0    | curl: NTLM password overflow via integer overflow |
| curl   | CVE-2018-16839 | HIGH     | 7.61.1-r1    | 7.61.1-r1    | curl: Integer overflow leading to heap-based buffer overflow in Curl_sasl_createnew_message() |
| curl   | CVE-2019-3822  |          | 7.61.1-r2    | 7.61.1-r2    | curl: NTLMv2 type-3 header stack buffer overflow |
| curl   | CVE-2018-16840 |          | 7.61.1-r1    | 7.61.1-r1    | curl: Use-after-free when closing "easy" handle in Curl_close() |
| curl   | CVE-2018-16890 | MEDIUM   | 7.61.1-r2    | 7.61.1-r2    | curl: NTLM type-2 header out-of-bounds read |
| curl   | CVE-2019-3823  |          | 7.61.1-r1    | 7.61.1-r1    | curl: SMTP end-of-response out-of-bounds read |
| curl   | CVE-2018-16842 |          | 7.61.1-r1    | 7.61.1-r1    | curl: Heap-based buffer over-read in the curl tool warning formatting |
| git    | CVE-2018-19486 | HIGH     | 2.15.2-r0    | 2.15.3-r0    | git: Improper handling of PATH allows for commands to be executed from...
```

kube-bench

Security Monitoring

Overview of kube-bench

kube-bench is a Go application that checks whether Kubernetes is deployed securely by running the checks documented in the CIS Kubernetes Benchmark.

```
root@ip-172-26-4-221:~# kube-bench
[INFO] 2 Worker Node Security Configuration
[INFO] 2.1 Kubelet
[FAIL] 2.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[PASS] 2.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 2.1.3 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 2.1.4 Ensure that the --client-ca-file argument is set as appropriate (Scored)
[FAIL] 2.1.5 Ensure that the --read-only-port argument is set to 0 (Scored)
[PASS] 2.1.6 Ensure that the --streaming-connection-idle-timeout argument is not set to 0 (Scored)
[FAIL] 2.1.7 Ensure that the --protect-kernel-defaults argument is set to true (Scored)
[PASS] 2.1.8 Ensure that the --make-iptables-util-chains argument is set to true (Scored)
[PASS] 2.1.9 Ensure that the --hostname-override argument is not set (Scored)
[FAIL] 2.1.10 Ensure that the --event-qps _argument is _set to 0 (Scored)
```

Securing Docker Daemon

Setting the Base

In most organizations, Docker daemon runs with root privileges, which means any user with access to the daemon can potentially gain elevated privileges on the host system.

```
root@docker:~# ps aux | grep docker
root      4000  0.6  7.8 1914260 77388 ?        Ssl  04:14   0:05 /usr/bin/dockerd -H unix:///var/run/docker.sock
27.0.0.1:2375 --containerd=/run/containerd/containerd.sock
```

1 - Removing Users from Docker Group

Users in the Docker group effectively have root privileges on the host system, as they can create containers that mount sensitive host directories.

```
root@docker:~# cat /etc/group | grep docker
docker:x:988:test-user
```

2 - Deny Traffic to Docker Daemon

If the **daemon** is exposed over TCP (`tcp://0.0.0.0:2375` or `tcp://0.0.0.0:2376`), it becomes a prime target for remote attacks

```
root@docker:~# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp      0      0 127.0.0.54:53           0.0.0.0:*            LISTEN    676/systemd-resolve
tcp      0      0 127.0.0.53:53           0.0.0.0:*            LISTEN    676/systemd-resolve
tcp      0      0 127.0.0.1:2375          0.0.0.0:*            LISTEN    4000/dockerd
tcp6     0      0 ::1:22                 ::*                  LISTEN    1/init
```

Reference Screenshot of Request to Remote Access

```
root@docker:~# curl http://127.0.0.1:2375/containers/json | jq
% Total    % Received % Xferd  Average Speed   Time      Time      Time  Current
                                         Dload  Upload   Total   Spent   Left  Speed
100  1778  100  1778     0      0  603k      0  --:--:--  --:--:--  --:--:--  868k
[
{
  "Id": "c86e7d79dfd876152bbd30035ac6bc537a38fabeb3890a96f0cae974ba45854",
  "Names": [
    "/romantic_mclean"
  ],
  "Image": "alpine",
  "ImageID": "sha256:aded1e1a5b3705116fa0a92ba074a5e0b0031647d9c315983ccba2ee5428ec8b",
  "Command": "tail -f /dev/null",
  "Created": 1740975652,
  "Ports": [],
  "Labels": {},
  "State": "running",
  "Status": "Up 17 minutes",
  "HostConfig": {
    "NetworkMode": "bridge"
```

Dockerfile - Security Best Practices

Setting the Base

A Dockerfile designed with security in mind avoids common security issues like privileged access, open ports, redundant software, and credential leaks

```
FROM ubuntu:18.04
USER root
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get install -y wget
RUN apt-get install -y nano
EXPOSE 80
WORKDIR /usr/src/app
COPY app.sh .
RUN chmod -R 777 /usr/src/app/app.sh
CMD ["./app.sh"]
```

1 - Use Updated Base Image

The original Dockerfile uses `ubuntu:18.04`, which is outdated.

Switching to a more recent version like `ubuntu:24.10` ensures better security, package support, and optimizations.



2 - Prefer Minimal Image

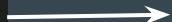
If the application does not require Ubuntu, a more minimal image like alpine can further reduce image size.



3 - Reduce Number of Layers

Instead of multiple RUN statement, combine them into a single RUN command

```
RUN apt-get update  
RUN apt-get install -y curl  
RUN apt-get install -y wget  
RUN apt-get install -y nano
```



```
RUN apt-get update && \  
    apt-get install -y curl && \  
    apt-get install -y wget && \  
    apt-get install -y nano
```

4 - Avoid Running as Root

The original Dockerfile uses ROOT user. This gives full read, write, and execute permissions to everyone, which is a security risk.

Instead use other user with limited privilege.

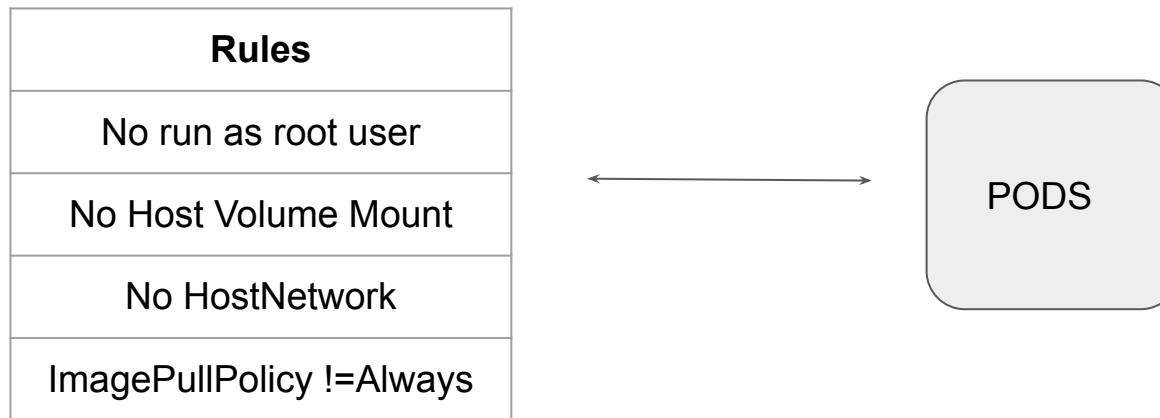


Static Analysis

Let's Secure

Overview of Static Analysis

Static code analysis is a method of debugging by examining source code before a program is run.



Tools for Static Analysis

There are various tools like Checkov that can perform static analysis.

```
Check: CKV_K8S_23: "Minimize the admission of root containers"
      FAILED for resource: Pod.privileged.default
      File: pod-priv.yaml:1-10
      Guide: https://docs.bridgecrew.io/docs/bc\_k8s\_22

      1 | apiVersion: v1
      2 | kind: Pod
      3 | metadata:
      4 |   name: privileged
      5 | spec:
      6 |   containers:
      7 |     - image: nginx
      8 |       name: demo-pod
      9 |       securityContext:
     10 |         privileged: true

Check: CKV_K8S_28: "Minimize the admission of containers with the NET_RAW capability"
      FAILED for resource: Pod.privileged.default (container 0) - demo-pod
      File: pod-priv.yaml:7-10
      Guide: https://docs.bridgecrew.io/docs/bc\_k8s\_27

      7 |     - image: nginx
      8 |       name: demo-pod
      9 |       securityContext:
     10 |         privileged: true
```

Dockerfile is Important

It is also important to go through Dockerfile for potential misconfiguration (security side)

```
FROM ubntu:latest
RUN apt-get update && apt-get install nano
USER ROOT
CMD[ "sleep 3600" ]
```

Securing Docker Daemon

Setting the Base

In most organizations, Docker daemon runs with root privileges, which means any user with access to the daemon can potentially gain elevated privileges on the host system.

```
root@docker:~# ps aux | grep docker
root      4000  0.6  7.8 1914260 77388 ?        Ssl  04:14   0:05 /usr/bin/dockerd -H unix:///var/run/docker.sock
27.0.0.1:2375 --containerd=/run/containerd/containerd.sock
```

1 - Removing Users from Docker Group

Users in the Docker group effectively have root privileges on the host system, as they can create containers that mount sensitive host directories.

```
root@docker:~# cat /etc/group | grep docker  
docker:x:988:test-user
```

2 - Deny Traffic to Docker Daemon

If the **daemon** is exposed over TCP (`tcp://0.0.0.0:2375` or `tcp://0.0.0.0:2376`), it becomes a prime target for remote attacks

Reference Screenshot of Request to TCP Port

```
root@docker:~# curl http://127.0.0.1:2375/containers/json | jq
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
                                         Dload  Upload   Total   Spent    Left  Speed
100  1778  100  1778     0      0   603k      0 --:--:-- --:--:-- --:--:--  868k
[
{
  "Id": "c86e7d79dfd876152bbd30035ac6bc537a38fabebb3890a96f0cae974ba45854",
  "Names": [
    "/romantic_mclean"
  ],
  "Image": "alpine",
  "ImageID": "sha256:aded1e1a5b3705116fa0a92ba074a5e0b0031647d9c315983ccba2ee5428ec8b",
  "Command": "tail -f /dev/null",
  "Created": 1740975652,
  "Ports": [],
  "Labels": {},
  "State": "running",
  "Status": "Up 17 minutes",
  "HostConfig": {
    "NetworkMode": "bridge"
  }
}
```

Dockerfile - Security Best Practices

Setting the Base

A Dockerfile designed with security in mind avoids common security issues like privileged access, open ports, redundant software, and credential leaks

```
FROM ubuntu:18.04
USER root
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get install -y wget
RUN apt-get install -y nano
EXPOSE 80
WORKDIR /usr/src/app
COPY app.sh .
RUN chmod -R 777 /usr/src/app/app.sh
CMD ["./app.sh"]
```

1 - Use Updated Base Image

The original Dockerfile uses `ubuntu:18.04`, which is outdated.

Switching to a more recent version like `ubuntu:24.10` ensures better security, package support, and optimizations.



2 - Prefer Minimal Image

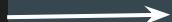
If the application does not require Ubuntu, a more minimal image like alpine can further reduce image size.



3 - Reduce Number of Layers

Instead of multiple RUN statement, combine them into a single RUN command

```
RUN apt-get update  
RUN apt-get install -y curl  
RUN apt-get install -y wget  
RUN apt-get install -y nano
```



```
RUN apt-get update && \  
    apt-get install -y curl && \  
    apt-get install -y wget && \  
    apt-get install -y nano
```

4 - Avoid Running as Root

The original Dockerfile uses ROOT user. This gives full read, write, and execute permissions to everyone, which is a security risk.

Instead use other user with limited privilege.



Docker Daemon Configuration

Setting the Base

There are **two ways** to configure the Docker daemon:

1. Use a JSON configuration file (preferred)
2. Use flags when starting dockerd

```
root@ubuntu:/etc/docker# cat /etc/docker/daemon.json
{
  "tls": true,
  "tlsverify": true,
  "tlscacert": "/etc/docker/certs/ca.pem",
  "tlscert": "/etc/docker/certs/server-cert.pem",
  "tlskey": "/etc/docker/certs/server-key.pem",
  "hosts": ["tcp://0.0.0.0:2376", "unix:///var/run/docker.sock"]
}
```

```
root@ubuntu:/etc/docker# dockerd --storage-driver overlay2
INFO[2025-03-05T13:56:03.735061939Z] Starting up
INFO[2025-03-05T13:56:03.736616384Z] OTEL tracing is not configured, using
INFO[2025-03-05T13:56:03.736814871Z] detected 127.0.0.53 nameserver, assum
stemd/resolve/resolv.conf
INFO[2025-03-05T13:56:03.759355871Z] [graphdriver] trying configured drive
INFO[2025-03-05T13:56:03.786205136Z] Loading containers: start.
INFO[2025-03-05T13:56:04.257306179Z] Loading containers: done.
INFO[2025-03-05T13:56:04.277348692Z] Docker daemon
e storage-driver=overlay2 version=28.0.1
INFO[2025-03-05T13:56:04.277821374Z] Initializing buildkit
INFO[2025-03-05T13:56:04.312898054Z] Completed buildkit initialization
INFO[2025-03-05T13:56:04.320436449Z] Daemon has completed initialization
INFO[2025-03-05T13:56:04.320508048Z] API listen on /var/run/docker.sock
```

JSON Configuration File Location

The following table shows the location where the Docker daemon expects to find the configuration file by default

OS and Configuration	Description
Linux, regular setup	/etc/docker/daemon.json
Windows	C:\ProgramData\docker\config\daemon.json

Protecting Docker Daemon Socket

Setting the Base

If you need Docker to be reachable through HTTP, you can enable TLS (HTTPS) and allow trusted connections through certificate based authentication



Authentication via Certificates

docker

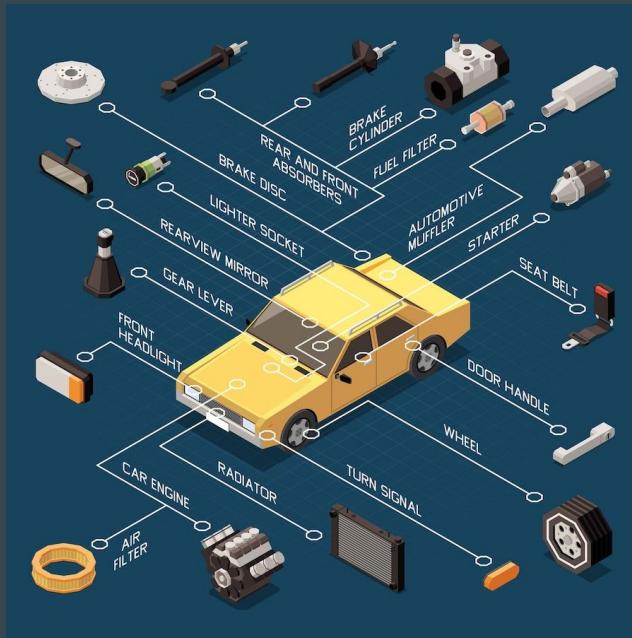
```
root@ubuntu:/etc/docker# curl --cert /etc/docker/certs/client-cert.pem --key /etc/docker/certs/client-key.pem --cacert /etc/docker/certs/ca.pem https://ubuntu:2376/version | jq
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
                                         Dload  Upload   Total   Spent   Left  Speed
100    820  100    820     0      0  36525       0  --::--  --::--  --::-- 37272
{
  "Platform": {
    "Name": "Docker Engine - Community"
  },
  "Components": [
    {
      "Name": "Engine",
      "Version": "28.0.1",
      "Details": {
        "ApiVersion": "1.48",
        "Arch": "amd64",
        "BuildTime": "2025-02-26T10:41:12.000000000+00:00",
        "Experimental": "false",
        "GitCommit": "bbd0a17",
        "GoVersion": "go1.23.6",
        "KernelVersion": "6.8.0-51-generic",
        "MinAPIVersion": "1.24",
        "Os": "linux"
      }
    }
  ]
}
```

BOM and SBOM

Basics of Bill of Materials

A Bill of Materials (BOM) is like a **recipe or blueprint for building a product.**

It lists all the components, quantity, materials required to manufacture an item.



INTEGRATIONS HELP + Oleg Demo

All ITEMS - 9559

ITEMS 9559 **ALL BOMs** 663 **TOP LEVEL BOMs** 243 **ALL CATALOGs** 124

My BOMs 647 **PPP-420240** Type: ELECTRIC Constr... Description: Cost: \$4.59

My Catalogs 117 **Flashlight-TLA** Type: Assembly Description: Cost: \$ 50.15

BOMs shared with me 10 **Base_&** Type: Subassembly Description: Cost: \$ 27.40

Catalogs shared with me 7 **TOP-59359** Type: Description: Cost:

Recent BOMs 18 **assy-plate100** Type: Description: Cost:

Deleted BOMs 11 **plate0100** Type: Description: Cost:

Deleted Catalogs 25 **cyl02** Type: Description: Cost:

Folders

- Alex
- Demo1
- Demo2
- Onshape Podcast
- + ADD
- Test 1

CREATE NEW FOLDER

BOM_Shell test assy-Main-Shell A... **Display Rim** **234234-1** **magic1**

Item Information

Part Number: Base_& **Catalog:** Demo SW cat.

BOM properties **Catalog properties**

Property	Value
Part Number	Base_&
Thumbnail image	
Description	Subassembly
Quantity On Hand	
Cost	\$ 27.40
Revision	cost adjustment
Revision Number	4
Author	
Category	MOLDED
Comment	
Configuration Name	Default
Date created	5/26/2011 9:30:51 AM
Date saved	5/26/2022 3:32:11 PM
File Name	Base_&.SLDASM
Keywords	

Support

© 2023 - Newman Cloud Inc. All Right Reserved. [Terms & Privacy](#). Version: master.NET

Software Bill of Materials

A Software Bill of Materials (SBOM) is a **detailed list of all the components** that make up a software application.





📁 SPDX Document SBOM-SPDX-779140af-12b0-410a-831d-a6f9986f7543

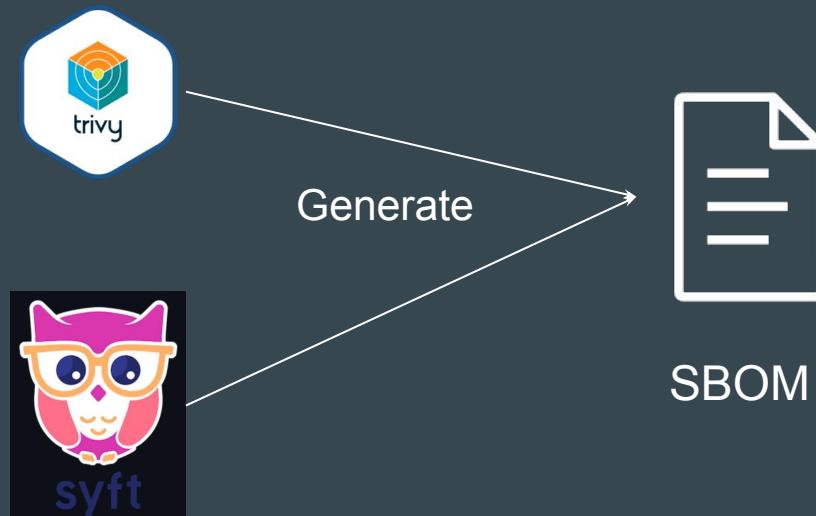
📦 DESCRIBES 1 Packages

- sha256:28edb1806e63847a8d6f77a7c312045e1bd91d5e3c944c8a0012f0b14c830c44
 - 🔗 7 Relationships
 - CONTAINS PACKAGE sha256:7cf63256a31a4cc44f6defe8e1af95363aee5fa75f30a248d95cae684f87c53c
 - CONTAINS PACKAGE sha256:bf9acace214a6c23630803d90911f1fd7d1ba06a3083f0a62fd036a6d1d8e274
 - 🔗 149 Relationships
 - CONTAINS PACKAGE adduser@3.134
 - CONTAINS PACKAGE apt@2.6.1
 - CONTAINS PACKAGE base-files@12.4+deb12u9
 - CONTAINS PACKAGE base-passwd@3.6.1
 - CONTAINS PACKAGE bash@5.2.15-2+b7
 - CONTAINS PACKAGE bsdutils@1:2.38.1-5+deb12u3
 - CONTAINS PACKAGE ca-certificates@20230311
 - CONTAINS PACKAGE coreutils@9.1-1
 - CONTAINS PACKAGE curl@7.88.1-10+deb12u8
 - CONTAINS PACKAGE dash@0.5.12-2
 - CONTAINS PACKAGE debconf@1.5.82
 - CONTAINS PACKAGE debian-archive-keyring@2023.3+deb12u1

Generating SBOM

Generating a Software Bill of Materials (SBOM) involves using specialized tools that analyze a software application, its dependencies, and components.

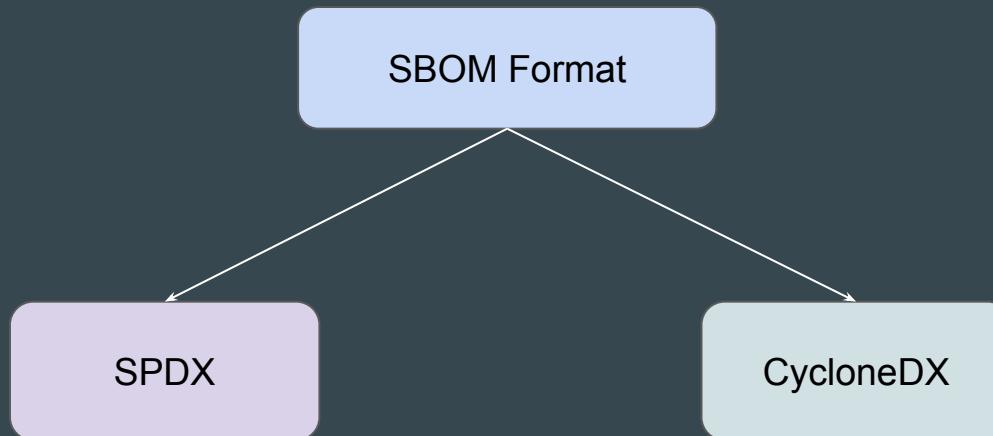
Various tools Trivy, Syft, Bom can generate SBOM.



SBOM Formats

The two most widely used SBOM formats are:

1. SPDX (Software Package Data Exchange)
2. CycloneDX



Understanding the Formats

Features	SPDX	CycloneDX
Developed By	Linux Foundation	OWASP
Focus Area	License compliance, intellectual property, security	Security, software supply chain risk management
Primary Use Case	Used widely in open source compliance and legal audits	Used mainly for security, vulnerability management, and risk assessment
Complexity	More complex, detailed metadata about licenses and compliance	Simplified, security-focused, lightweight

Overview of Falco

Detection and Prevention

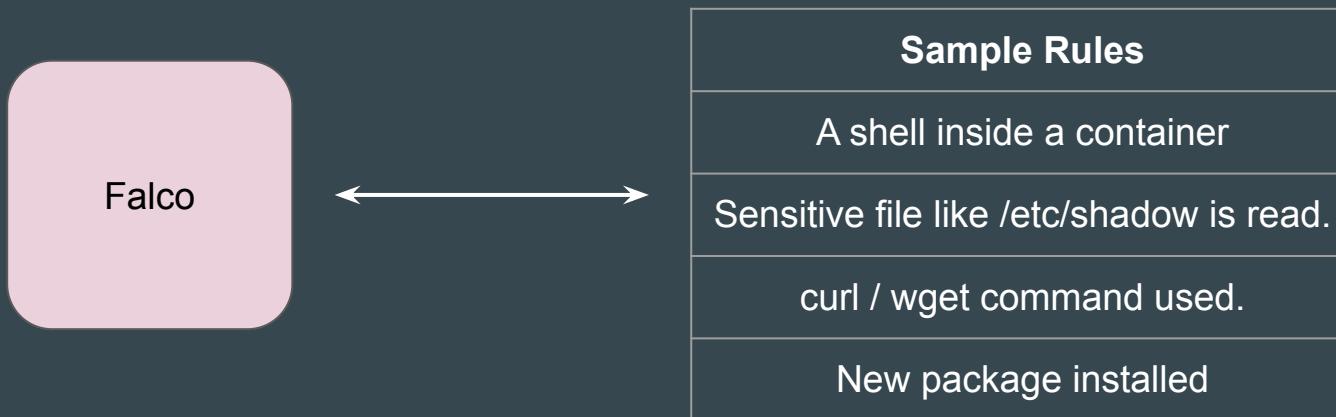
In Kubernetes, features like Network Policies and RBAC are primarily used for prevention related capabilities.

It is equally important to implement detection measures that provide visibility into the environment, allowing us to monitor and understand ongoing activities.



Introduction to Falco

Falco is an open-source **runtime security tool** that allows users to define a set of rules that will trigger an alert whenever the conditions are met.



Host Based Events Captured

```
=====Mar 07 07:16:35 k8s falco[134162]: 07:16:35.714043982: Warning Sensitive file opened for reading by non-trusted program (file=/etc/shadow gparent=sshd gparent=sshd gparent=systemd evt_type=openat user=root user_uid=0 user_loginuid=0 process =cat proc_exepath=/usr/bin/cat parent=bash command=cat /etc/shadow terminal=34817 container_id=host container_name=host)
```

Container Based Events Captured

```
Mar 07 07:28:05 k8s falco[134162]: 07:28:05.304940082: Notice A shell was spawned in a container with an attached terminal (evt_type=execve user=root user_uid=0 user_loginuid=-1 process=bash proc_exepath=/usr/bin/bash parent=containerd-shim command=bash terminal=34816 exe_flags=EXE_WRITABLE|EXE_LOWER_LAYER container_id=b30aaaf02ca11 container_name=<NA>)
```

Writing Custom Falco Rules

Basic Elements of Falco Rule

```
- rule: shell_in_container
  desc: notice shell activity within a container
  condition: >
    evt.type = execve and
    evt.dir = < and
    (proc.name = bash or
     proc.name = ksh)
  output: >
    shell in a container
    (user=%user.name container_id=%container.id container_name=%container.name
     shell=%proc.name parent=%proc.pname cmdline=%proc.cmdline)
  priority: WARNING
```

Basic Rule Format

Field	Description
rules	The name of the rule. It should be unique and descriptive of what the rule detects.
desc	A human-readable description of what the rule does, explaining the security threat being detected.
condition	A logical expression that defines when the rule should trigger an alert.
output	The alert message that is generated when the rule condition is met
priority	The severity level of the rule (e.g., EMERGENCY, ALERT, CRITICAL, ERROR, WARNING etc) . Higher severity indicates more critical security events.

Sample Rule

Below is a sample **Falco rule** that detects the execution of the `curl` command inside a Kubernetes pod.

```
- rule: Detect curl Execution in Kubernetes Pod
  desc: Detects when the curl utility is executed within a Kubernetes pod.
  condition: >
    spawned_process and container and
    proc.name = "curl"
  output: >
    Suspicious process detected (curl) inside a Kubernetes pod.
  priority: WARNING
```

Macros

Macros are essentially predefined rule conditions. They allows you to avoid repeatedly writing the same complex expressions.

```
- macro: sensitive_files
  condition: fd.name in (/tmp/sensitive.txt)

- rule: Access to Sensitive Files
  desc: Detect any process attempting to read or write sensitive files
  condition: sensitive_files
  output: "Sensitive file access detected (user=%user.name process=%proc.name file=%fd.name)"
  priority: WARNING
```

Falco Rule for /dev/mem Access

Setting the Base

`/dev/mem` is a special device file in Linux that provides access to the system's physical memory.

```
root@kubeadm:~# ls -l /dev/mem
crw-r----- 1 root kmem 1, 1 Mar  7 17:06 /dev/mem
```

Important to Monitor /dev/mem

Since `/dev/mem` grants access to critical system memory, unauthorized access can lead to Privilege Escalation, Kernel Exploits etc.

Containers are meant to be isolated and should not interact directly with system memory.



Container and /dev/mem Access

Not all Kubernetes pods have access to /dev/mem by default.

Access to /dev/mem is restricted unless a pod is privileged or explicitly granted special permissions.

```
root@nginx-pod:~# ls -l /dev
total 0
lrwxrwxrwx 1 root root   11 Mar  7 17:39 core -> /proc/kcore
lrwxrwxrwx 1 root root   13 Mar  7 17:39 fd -> /proc/self/fd
crw-rw-rw- 1 root root 1, 7 Mar  7 17:39 full
drwxrwxrwt 2 root root   40 Mar  7 17:38 mqueue
crw-rw-rw- 1 root root 1, 3 Mar  7 17:39 null
lrwxrwxrwx 1 root root    8 Mar  7 17:39 ptmx -> pts/ptmx
drwxr-xr-x 2 root root    0 Mar  7 17:39 pts
crw-rw-rw- 1 root root 1, 8 Mar  7 17:39 random
drwxrwxrwt 2 root root   40 Mar  7 17:38 shm
lrwxrwxrwx 1 root root  15 Mar  7 17:39 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root  15 Mar  7 17:39 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root  15 Mar  7 17:39 stdout -> /proc/self/fd/1
-rw-rw-rw- 1 root root    0 Mar  7 17:39 termination-log
crw-rw-rw- 1 root root 5, 0 Mar  7 17:39 tty
crw-rw-rw- 1 root root 1, 9 Mar  7 17:39 urandom
crw-rw-rw- 1 root root 1, 5 Mar  7 17:39 zero
```

Falco Configuration File

Setting the Base

Falco's configuration file is a **YAML file** containing a collection of key: value or key: [value list] pairs.

Configuration file is available at /etc/falco.yaml

```
root@kubeadm:~# ls -l /etc/falco
total 196
drwxr-xr-x 2 root root 4096 Mar  7 17:29 config.d
-rw-r--r-- 1 root root 58628 Jan 28 09:06 falco.yaml
-rw-r--r-- 1 root root 58628 Mar  7 18:55 falco.yaml.bak
-rw-r--r-- 1 root root   328 Mar  7 18:46 falco_rules.local.yaml
-rw-r--r-- 1 root root 63723 Jan  1 1970 falco_rules.yaml
drwxr-xr-x 2 root root 4096 Jan 28 09:34 rules.d
```

```
# When appending Falco alerts to a file, each new
# line. It's important to note that Falco does no
# file. If the `keep_alive` option is set to `true`
# and continuously written to, else the file will
# message. Furthermore, the file will be closed a
# the SIGUSR1 signal.
file_output:
  enabled: false
  keep_alive: false
  filename: ./events.txt
```

Sysdig

Monitoring System calls

Overview of Sysdig

In a normal scenario of troubleshooting and performance monitoring, we make use of the following tools

Sysdig offers the functionality of these tools along with a lot more.

strace	Discovering system calls
tcpdump	Network traffic monitoring
lsof	Files are opened by which process.
netstat	Network Connection monitoring
htop	Process Monitoring
iftop	Network Bandwidth monitoring



Interactive Options

Sysdig Utility comes with a command line option (sysdig) as well as interface UI (csysdig)

Viewing: Processes For: whole machine Source: Live System Filter: evt.type!=switch								
PID	CPU	USER	TH	VIRT	RES	FILE	NET	Command
123827	6.50	root	6	577M	159M	0	0.00	csysdig
102296	6.00	root	9	1G	334M	0	15.22K	kube-apiserver --advertise-address=172.31.59.221
471	3.00	root	18	2G	107M	0	3.39K	/usr/bin/kubelet --bootstrap-kubeconfig=/etc/kube
102255	2.50	root	7	793M	100M	0	17.89K	kube-controller-manager --allocate-node-cidrs=true
663	1.50	root	27	1G	114M	0	0.00	/usr/bin/dockerd -H fd:// --containerd=/run/conta
1966	1.50	root	14	2G	73M	70K	2.98K	etcd --advertise-client-urls=https://172.31.59.22
4714	0.50	root	9	730M	38M	0	893.50	/coredns -conf /etc/coredns/corefile
491	0.50	root	33	1G	49M	0	0.00	/usr/bin/containerd
3550	0.00	root	9	1G	36M	0	0.00	/opt/bin/flanneld --ip-masq --kube-subnet-mgr
123819	0.00	ubuntu	1	6M	4M	0	0.00	/usr/lib/openssh/sftp-server
1689	0.00	root	10	106M	6M	0	0.00	containerd-shim -namespace moby -workdir /var/lib
488	0.00	root	1	17M	8M	0	0.00	/lib/systemd/systemd-logind
4368	0.00	root	10	106M	5M	0	0.00	containerd-shim -namespace moby -workdir /var/lib
4353	0.00	root	10	106M	5M	0	0.00	containerd-shim -namespace moby -workdir /var/lib
102256	0.00	root	9	729M	46M	0	1.51K	kube-scheduler --authentication-kubeconfig=/etc/k
534	0.00	www-data	1	57M	5M	0	0.00	nginx: worker process
3265	0.00	root	10	106M	6M	0	0.00	containerd-shim -namespace moby -workdir /var/lib
512	0.00	root	1	6M	2M	0	0.00	/sbin/agetty -o -p -- \u --noclear tt1 linux
1641	0.00	root	11	853M	28M	0	0.00	/snap/amazon-ssm-agent/2996/ssm-agent-worker
3123	0.00	root	1	964K	4K	0	0.00	/pause
458	0.00	root	1	8M	3M	0	0.00	/usr/sbin/cron -f

Running sysdig

In its simplest form, when you run sysdig, you will see all the system calls that are happening within the system.

```
52 07:50:42.829676193 1 <NA> (0) > switch next=1157782 pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_swap=0
54 07:50:42.829686696 0 sysdig (1171310) > switch next=1155766 pgft_maj=0 pgft_min=2183 vm_size=447872 vm_rss=25612
      vm_swap=0
55 07:50:42.829687819 1 <NA> (1157782) > switch next=1170533(sshd) pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_swap=0
56 07:50:42.829690356 0 <NA> (1155766) > switch next=1171310(sysdig) pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_swap=0
57 07:50:42.829696854 1 sshd (1170533) < select res=1
58 07:50:42.829697168 0 sysdig (1171310) > switch next=1155766 pgft_maj=0 pgft_min=2183 vm_size=447872 vm_rss=25612
      vm_swap=0
59 07:50:42.829699893 0 <NA> (1155766) > switch next=1171310(sysdig) pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_swap=0
60 07:50:42.829702032 1 sshd (1170533) > rt_sigprocmask
61 07:50:42.829703224 1 sshd (1170533) < rt_sigprocmask
62 07:50:42.829704021 1 sshd (1170533) > rt_sigprocmask
63 07:50:42.829704472 1 sshd (1170533) < rt_sigprocmask
64 07:50:42.829707634 1 sshd (1170533) > clock_gettime
65 07:50:42.829708525 1 sshd (1170533) < clock_gettime
68 07:50:42.829710931 1 sshd (1170533) > read fd=15(<f>/dev/ptmx) size=16384
69 07:50:42.829714826 1 sshd (1170533) < read res=1201 data=1 07:50:42.736703000 0 container:0d65633084bf (-1) > container_json={"container": "0d65633084bf", "pid": 1170533, "uid": 0, "start_time": "2023-07-05T07:50:42.736703000Z", "processes": [{"name": "sshd", "pid": 1170533}], "files": [{"fd": 15, "path": "/dev/ptmx"}]}
```

Filters

Since you will get a huge amount of data when monitoring system calls, you can use sysdig with filters to make the output more fine grained.

```
root@ip-172-31-59-221:~# sysdig proc.name=nano
310161 07:55:39.213391058 0 nano (1173625) < execve res=0 exe=nano args=test.txt. tid=1173625(nano) pid=1173625(nano) ptid=1173599(bash) cwd= fdlimit=1024 pgft_maj=1 pgft_min=24 vm_size=652 vm_rss=4 vm_swap=0 comm=nano cgroups=cpu set=/user.slice.cpuacct=/user.slice.io=/user.slice.memory=/user.slice... env=SHELL=/bin/bash.SUDO_GID=1000.SUDO_COMMAND=/usr/bin/su.SUDO_USER=ubuntu.PWD=/... tty=34822 pgid=1173625(nano) loginuid=1000
310162 07:55:39.213423440 0 nano (1173625) > brk addr=0
310163 07:55:39.213424690 0 nano (1173625) < brk res=563F7341E000 vm_size=652 vm_rss=4 vm_swap=0
310164 07:55:39.213434121 0 nano (1173625) > arch_prctl
310165 07:55:39.213434779 0 nano (1173625) < arch_prctl
310166 07:55:39.213459275 0 nano (1173625) > access mode=4(R_OK)
310167 07:55:39.213464446 0 nano (1173625) < access res=-2(ENOENT) name=/etc/ld.so.preload
310168 07:55:39.213469242 0 nano (1173625) > openat
310169 07:55:39.213476122 0 nano (1173625) > fstat fd=3
310170 07:55:39.213477566 0 nano (1173625) < fstat res=0
```

Sysdig Chisels

Sysdig's chisels are little scripts that analyze the sysdig event stream to perform useful actions

```
root@ip-172-31-59-221:~# sysdig -cl
Category: Application
-----
httplog      HTTP requests log
httptop      Top HTTP requests
memcachedlog memcached requests log

Category: CPU Usage
-----
spectrogram   Visualize OS latency in real time.
subsecoffset  Visualize subsecond offset execution time.
topcontainers_cpu
              Top containers by CPU usage
topprocs_cpu  Top processes by CPU usage

Category: Errors
-----
topcontainers_error
                  Top containers by number of errors
topfiles_errors Top files by number of errors
topprocs_errors top processes by number of errors
```

Audit Logging

Designing Right Logging Rules

Revising Auditing

Auditing provides a security-relevant, chronological set of records documenting the sequence of actions in a cluster.

- what happened?
- when did it happen?
- who initiated it?
- on what did it happen?
- from where was it initiated?
- to where was it going?

```
{  
    "kind": "Event",  
    "apiVersion": "audit.k8s.io/v1",  
    "level": "Metadata",  
    "auditID": "388ca4e1-c368-45b4-aca9-652e32baf8e1",  
    "stage": "RequestReceived",  
    "requestURI": "/api/v1/namespaces/default/secrets?limit=500",  
    "verb": "list",  
    "user": {  
        "username": "bob",  
        "groups": [  
            "system:masters",  
            "system:authenticated"  
        ]  
    },  
    "sourceIPs": [  
        "127.0.0.1"  
    ],  
    "userAgent": "kubectl/v1.19.0 (linux/amd64) kubernetes/e199641",  
    "objectRef": {  
        "resource": "secrets",  
        "namespace": "default",  
        "apiVersion": "v1"  
    },  
    "requestReceivedTimestamp": "2020-12-03T16:55:10.357789Z",  
    "stageTimestamp": "2020-12-03T16:55:10.357789Z"  
}
```

Audit Policy Levels

Audit policy defines rules about what events should be recorded and what data they should include.

Audit Levels	Description
None	don't log events that match this rule.
Metadata	Log request metadata (requesting user, timestamp, resource, verb, etc.) but not request or response body.
Request	Log event metadata and request body but not response body.
RequestResponse	Log event metadata, request and response bodies.

Stages

The kube-apiserver processes request in stages and each stage generates an audit event.

Stage	Description
RequestReceived	The stage for events generated as soon as the audit handler receives the request, and before it is delegated down the handler chain.
ResponseStarted	Once the response headers are sent, but before the response body is sent. This stage is only generated for long-running requests (e.g. watch).
ResponseComplete	The response body has been completed and no more bytes will be sent.
Panic	Events generated when a panic occurred.

Important Pointers for Exams

Setting the Base

The 'Important Pointers for Exams' video is **not a substitute** for the full course.

We highly recommend completing all the videos that are part of the course.



Keep Close Look on External Documentation

The specific external documentation permitted in Linux Foundation exams provides insight into the key topics that might be assessed.

- **Kubernetes Documentation:**

- <https://kubernetes.io/docs/>
- <https://kubernetes.io/blog/>

This includes all available language translations (e.g. <https://kubernetes.io/zh/docs/>)

- **Tools:**

- Falco documentation <https://falco.org/docs/>
- Bom documentation <https://kubernetes-sigs.github.io/bom/cli-reference/>
- etcd documentation <https://etcd.io/docs/>

This includes all available language translations (e.g. <https://falco.org/zh/docs/>)

- **NGINX Ingress Controller**

- Documentation <https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/>

- **Cilium:**

- Documentation <https://docs.cilium.io/en/stable>

CIS Benchmarks

It is important to know about **configuring Kubernetes components** (Control Plane + Worker Node) based on various CIS Benchmark related configuration.

Be very familiar with kubeadm structure and troubleshooting pointers.

Set AuthorizationMode for API Server to RBAC,WebHook

Disable Anonymous Authentication in Kubelet

Disable --auto-tls in etcd

ImagePolicyWebHook

Know the end to end steps to create and enable ImagePolicyWebhook.

Step 1: Create a Configuration File.

Step 2: Create a KubeConfig file.

Step 3: Mount Volumes

Step 4: Enable Admission Controller.

```
imagePolicy:  
  kubeConfigFile: /path/to/kubeconfig/for/backend  
  # time in s to cache approval  
  allowTTL: 50  
  # time in s to cache denial  
  denyTTL: 50  
  # time in ms to wait between retries  
  retryBackoff: 500  
  # determines behavior if the webhook backend fails  
  defaultAllow: true
```

Know about the **defaultAllow** parameter in the configuration file.

Auditing

You should be able to enable Auditing based on the requirements.

Field	Description
--audit-log-path	Specifies the file path where the audit log is written.
--audit-log-maxage	Defines the maximum number of days to retain old audit log files before deletion.
--audit-log-maxbackup	Sets the maximum number of old audit log files to retain.
--audit-log-maxsize	Specifies the maximum size (in megabytes) of the audit log file before it gets rotated.

Example Question - Auditing

1. Logs should be stored at /var/log/demo-audit.logs
2. Logs should be retained for the next 30 days.
3. Maximum size before rotation should be 500 MB.
4. Maximum number of 10 audit log files should be made available.

Audit Policy

Be familiar setting the Audit Policy based on requirements.

```
apiVersion: audit.k8s.io/v1
kind: Policy
omitStages:
  - "RequestReceived"
rules:
  - level: None
    resources:
      - group: ""
        resources: ["secrets"]
        namespaces: ["kube-system"]
  - level: None
    users: ["system:kube-controller-manager"]
    resources:
      - group: ""
        resources: ["secrets"]
```

Docker Security

You need to be aware of Docker Daemon Security + Dockerfile security best practices.

Example Scenarios:

1. Analyze the Dockerfile and fix 5 security issues.
2. Disable Docker Daemon to listen on 2375
3. Make Docker Daemon Secure
4. Remove user from docker group.

Static Analysis on Kubernetes Manifest

You should be able to read a given Kubernetes manifest file and fix any security related issues.

```
kind: Deployment
metadata:
  name: insecure-deployment
  labels:
    app: insecure-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: insecure-app
  template:
    metadata:
      labels:
        app: insecure-app
    spec:
      containers:
        - name: insecure-container
          image: nginx:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
      env:
        - name: DB_PASSWORD
          value: "supersecretpassword"
      securityContext:
        privileged: true
        allowPrivilegeEscalation: true
        runAsUser: 0
        capabilities:
          add:
            - SYS_ADMIN
```

Network Policies + Cilium Network Policies

Be comfortable writing Network Policies + Cilium Network Policy.

For Cilium Network Policy:

- Be aware of ingressDeny and egressDeny block.
- Be aware of the Entities in Cilium Network Policies.

Pod Security Standards

You need to have clear understanding of pod security standards, including how to implement and adjust PSS configurations for pods and deployments

Policies	Description
Privileged	Unrestricted policy, providing the widest possible level of permissions. Allows privilege escalations
Baseline	Minimally restrictive policy which prevents known privilege escalations.
Restricted	Heavily restricted policy, following current Pod hardening best practices.

Security Context

Be familiar with Security Context.

Privileged Pods, Capabilities, readOnlyRootFilesystem (immutability)

```
apiVersion: v1
kind: Pod
metadata:
  name: privileged-pod
spec:
  containers:
    - image: nginx
      name: privileged
      securityContext:
        privileged: true
```

```
apiVersion: v1
kind: Pod
metadata:
  name: better-pod
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 1000
  containers:
    - name: better-container
      image: busybox
      command: ["sleep", "36000"]
```

```
apiVersion: v1
kind: Pod
metadata:
  name: capabilities-demo
spec:
  containers:
    - name: sec-ctx-4
      image: gcr.io/google-samples/hello-app:2.0
      securityContext:
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
```

Kubernetes Secrets

Know the basics of creating Secrets and mounting them to Pods.

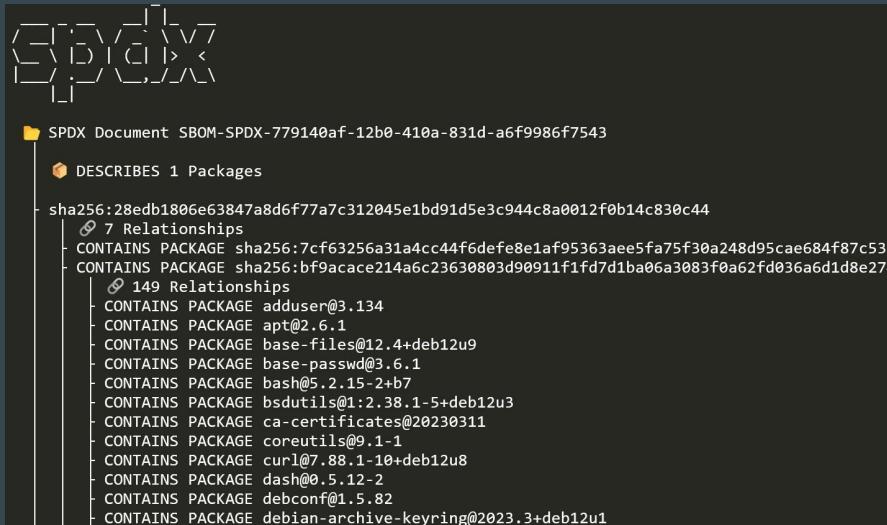
Be familiar with various type of secrets

1. Opaque Secrets.
2. TLS Secrets
3. Docker config Secrets

BOM and SBOM

You should know on how to create SBOM using **bom** tool based on requirements.

Example: Identify Image that has xyz 1.3.2 package and create SBOM for it.



```
SPDX Document SBOM-SPDX-779140af-12b0-410a-831d-a6f9986f7543

📦 DESCRIBES 1 Packages

- sha256:28edb1806e63847a8d6f77a7c312045e1bd91d5e3c944c8a0012f0b14c830c44
  ⚡ 7 Relationships
    - CONTAINS PACKAGE sha256:7cf63256a31a4cc44f6defe8e1af95363aee5fa75f30a248d95cae684f87c53c
    - CONTAINS PACKAGE sha256:bf9acace214a6c23630803d90911f1fd7d1ba06a3083f0a62fd036a6d1d8e274
      ⚡ 149 Relationships
        - CONTAINS PACKAGE adduser@3.134
        - CONTAINS PACKAGE apt@2.6.1
        - CONTAINS PACKAGE base-files@12.4+deb12u9
        - CONTAINS PACKAGE base-passwd@3.6.1
        - CONTAINS PACKAGE bash@5.2.15-2+b7
        - CONTAINS PACKAGE bsduutils@1:2.38.1-5+deb12u3
        - CONTAINS PACKAGE ca-certificates@20230311
        - CONTAINS PACKAGE coreutils@9.1-1
        - CONTAINS PACKAGE curl@7.88.1-10+deb12u8
        - CONTAINS PACKAGE dash@0.5.12-2
        - CONTAINS PACKAGE debconf@1.5.82
        - CONTAINS PACKAGE debian-archive-keyring@2023.3+deb12u1
```

Kubernetes Cluster Upgrade

Learn to upgrade both control plane and worker nodes using kubeadm

Don't mix the steps. The steps for upgrading the kubeadm worker node are different from control plane node.

Upgrade Control
Plane Node

Upgrade Worker
Node

Ingress with TLS

Be familiar with the steps required to set up Ingress with TLS.

Be familiar with **ssl-redirect** annotation for HTTP to HTTPS

```
root@kubeadm:~# kubectl describe ingress tls-ingress
Name:           tls-ingress
Labels:         <none>
Namespace:      default
Address:        192.168.45.195
Ingress Class:  nginx
Default backend: <default>
TLS:
  tls-cert terminates demo.kplabs.in
Rules:
  Host            Path  Backends
  ----          -----
  demo.kplabs.in   /    example-service:80 (192.168.45.195:80)
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
  creationTimestamp: "2025-03-06T16:59:09Z"
  generation: 1
  name: demo-ingress
  namespace: default
  resourceVersion: "2998"
```

Service Account + Projected Volumes

Know how to create service accounts with auto mounting as disabled.

Be familiar with mounting volume sources like SA using Projected Volumes.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
automountServiceAccountToken: false
```

```
containers:
- name: container-test
  image: busybox:1.37
  command: ["sleep", "3600"]
  volumeMounts:
    - name: token-vol
      mountPath: "/service-account"
      readOnly: true
  volumes:
    - name: token-vol
      projected:
        sources:
          - serviceAccountToken:
              audience: api
              expirationSeconds: 3600
              path: token
```

Falco (Keep it for Last)

Be prepared to develop a Falco rule according to a given specification.

If you encounter issues with Falco log generation, verify that syslog is enabled with debug priority. Alternatively, run Falco directly from the command line, bypassing systemd.

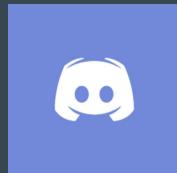
```
- rule: Detect curl Execution in Kubernetes Pod
  desc: Detects when the curl utility is executed within a Kubernetes pod.
  condition: >
    spawned_process and container and
    proc.name = "curl"
  output: >
    Suspicious process detected (curl) inside a Kubernetes pod.
  priority: WARNING
```

Be Familiar with Deployment Manifest

Exams love Deployment manifests more than Pod manifests.

Join us in our Adventure

Be Awesome



kplabs.in/chat



kplabs.in/linkedin