

Kubernetes

12 March 2021 12:57

17) ===== YAML Configuration File

Each configuration file has

Deployment

- 1) metadata
- 2) specification

```
! nginx-deployment.yaml ×  
1  apiVersion: apps/v1  
2  kind: Deployment  
3  metadata:  
4    name: nginx-deployment  
5    labels: ...  
7  spec:  
8    replicas: 2  
9    selector: ...  
12   template: ...
```

Here total 3 parts are there. And 3rd Part is called Status:Status is automatically generated and added by Kubernetes.

```
status:  
  availableReplicas: 1
```

s 3 parts

Service

! nginx-service.yaml ✘

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: nginx-service
5 spec:
6   selector: ...
7   ports: ...
```

12

SUBSCRIBE



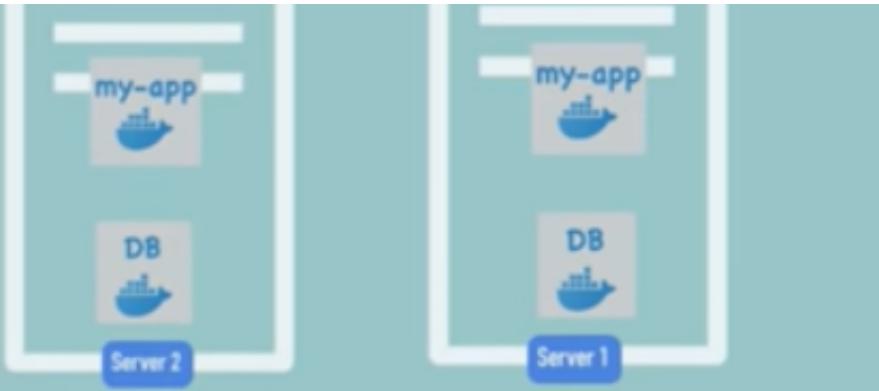
```
conditions:
- lastTransitionTime: "2020-01-24T10:54:59Z"
  lastUpdateTime: "2020-01-24T10:54:59Z"
  message: Deployment has minimum availability.
  reason: MinimumReplicasAvailable
  status: "True"
  type: Available
- lastTransitionTime: "2020-01-24T10:54:56Z"
  lastUpdateTime: "2020-01-24T10:54:59Z"
  message: ReplicaSet "nginx-deployment-7d64f4b"
  reason: NewReplicaSetAvailable
  status: "True"
  type: Progressing
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
  updatedReplicas: 1
```

ETCD holds the CURRENT STATUS of K8s.

18)

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  >  labels: ...
7  spec:
8    replicas: 2
9  >  selector: ...
12 template:
13   metadata:
```

- has it's own "apiVersion" and "kind"
- applies to pods



! nginx-deployment.yaml ✘

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  +  labels: ...
7  spec:
8    replicas: 2
9  +  selector: ...
12 +  template: ...
```

[SUBSCRIBE](#)

Template

own "metadata"
"spec" section

to Pod

```
14     labels:  
15         app: nginx  
16     spec:  
17         containers:  
18             - name: nginx  
19                 image: nginx:1.16  
20                 ports:  
21                     - containerPort: 8080
```

- blueprint

port?

ima

19)

Deployment

Connecting D

```
1   apiVersion: apps/v1  
2   kind: Deployment  
3   metadata:  
4       name: nginx-deployment  
5       labels:  
6           app: nginx  
7   spec:  
8       replicas: 2  
9       selector:  
10          matchLabels:  
11              app: nginx  
12          template:  
13              metadata:  
14                  labels:  
15                      app: nginx
```

- any key-value

labels:
| app: nginx

: for a Pod

name?

ge?

0%

Deployment to Pods

ue pair for component

nx

SUBSCRIBE

Connecting Deployment

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5    labels:
6      app: nginx
7  spec:
8    replicas: 2
9    selector:
10       matchLabels:
11         app: nginx
12    template:
13      metadata:
14        labels:
15          app: nginx
16    spec: ...
```

- Pods get the label through template blueprint

- This label is matched by selector

```
  selector:
    matchLabels:
      app: ng
```

Connecting Services to Deployment

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5    labels:
6      app: nginx
7  spec:
8    replicas: 2
9    selector:
10       matchLabels:
11         app: nginx
12    template:
```



Get to Pods

through the

Deployments

```
ls:  
inx
```

▶ SUBS

Deployments

Service

```
1  apiVersion: v1  
2  kind: Service  
3  metadata:  
4    name: nginx-service  
5  spec:  
6    selector:  
7      app: nginx  
8    ports: ...  
9  
10   ports:  
11     port: 80  
12       targetPort: 80  
13       protocol: TCP  
14   selector:  
15     app: nginx  
16  
17   type: ClusterIP
```

12

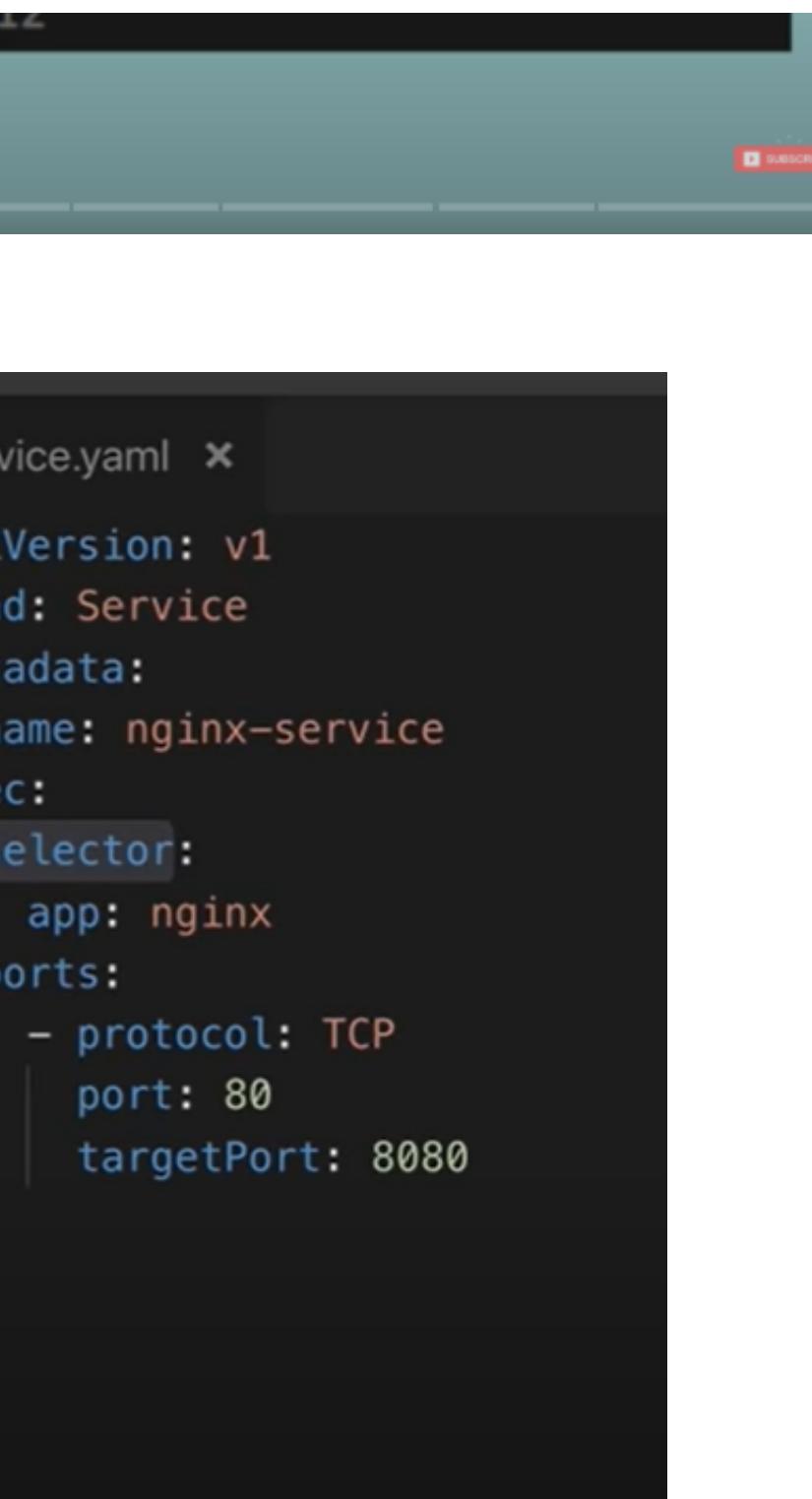
```
13     metadata:  
14       labels:  
15         app: nginx  
16   spec ...
```



20)

```
! nginx-deployment.yaml ✘          ! nginx-service.yaml ✘  
1   spec:  
2     replicas: 2  
3     selector:  
4       matchLabels:  
5         app: nginx  
6     template:  
7       metadata:  
8         labels:  
9           app: nginx  
10    spec:  
11      containers:  
12        - name: nginx  
13          image: nginx:1.16  
14          ports:  
15            - containerPort: 8080  
16  
17  
18  
19  
20  
21  
22
```

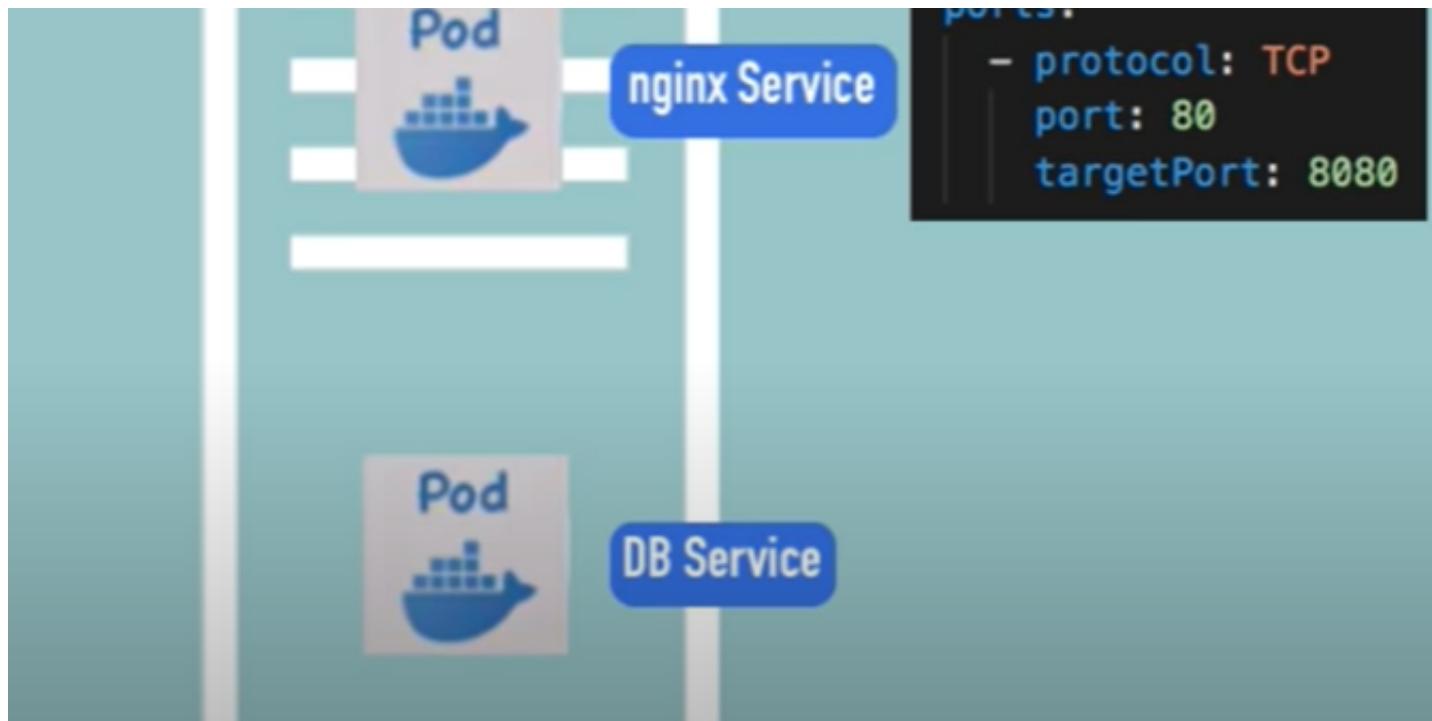
Ports in Service and Pod



A screenshot of a mobile device displaying a YAML configuration file. The file defines a service named 'nginx-service' with version v1. It includes metadata, an elector set to 'nginx', and a port mapping from port 80 to targetPort 8080.

```
service.yaml ✘
Version: v1
kind: Service
metadata:
  name: nginx-service
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```





Ports in Service and

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  >  labels: ...
6  spec:
7    replicas: 2
8  >  selector: ...
9  template:
10 >   metadata: ...
11   spec:
12     containers:
13       - name: nginx
14         image: nginx:1.16
15         ports:
16           - containerPort: 8080
```

```
ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

target





port: 80

nginx Service



Pod

Pod

DB Service



port: 80

nginx Service



Port: 8080

30

Port: 8080

Port: 8080

8080

21)

```
[Documents]$ kubectl get pod
NAME                               READY   STATUS
nginx-deployment-7d64f4b574-fklxj   1/1    Running
nginx-deployment-7d64f4b574-v7mwj   1/1    Running
[Documents]$ kubectl get service
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP
kubernetes  ClusterIP  10.96.0.1    <none>
nginx-service  ClusterIP  10.96.25.229  <none>
[Documents]$ kubectl describe service nginx-service
Name:           nginx-service
Namespace:      default
Labels:          <none>
Annotations:    kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"v1","kind":"Service","metadata":{"name":"nginx-service","namespace":"default"},"spec":{"ports":[{"port":80,"targetPort":8080}]}}, kubectl.kubernetes.io/pod-labels={"app": "nginx"}, kubectl.kubernetes.io/service-labels={"app": "nginx"}, kubectl.kubernetes.io/service-name="nginx-service"
Selector:        app=nginx
Type:            ClusterIP
IP:              10.96.25.229
Port:            <unset>  80/TCP
TargetPort:      8080/TCP
Endpoints:      172.17.0.6:8080,172.17.0.7:8080
Session Affinity: None
Events:          <none>
```

Events: > <none>

```
[Documents]$ kubectl get pod -o wide
```

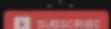
```
NAME          NOMINATED NODE  READINESS GATES
nginx-deployment-7d64f4b574-fklxj   1/1    Running
ikube         <none>       <none>
nginx-deployment-7d64f4b574-v7mwj   1/1    Running
ikube         <none>       <none>
```

Get Updated details of Deployment i.e. check how STATUS is getting saved in to .yaml file..

RESTARTS AGE
0 10s
0 10s

PORT(S) AGE
443/TCP 2d
80/TCP 19s

configuration:
e","metadata": {"annotations": {},"na
rts": [{"port": 80...



RESTARTS AGE
0 2m17s
0 2m17s

IP	NOD
172.17.0.7	min
172.17.0.6	min

```
kubectl get deployment nginx-deployment -o yaml > nginx-deployment.yaml
```

Deployment gets Deleted if we delete config file.....

```
[Documents]$ kubectl delete -f nginx-deployment.yaml  
deployment.apps "nginx-deployment" deleted  
[Documents]$ kubectl delete -f nginx-service.yaml  
service "nginx-service" deleted
```

22)

Real time Set up::

Overview of K8s Components

Internal Service

ConfigMap

DB Url

Deployment.yaml

Env

variables



MongoDB

Secret

DB User

DB Pwd

URL:
- IP Address of Node
- Port of external Service

x-deployment-result

nt.yaml

yaml

ents

2 Deployment / Pod
2 Service
1 ConfigMap
1 Secret

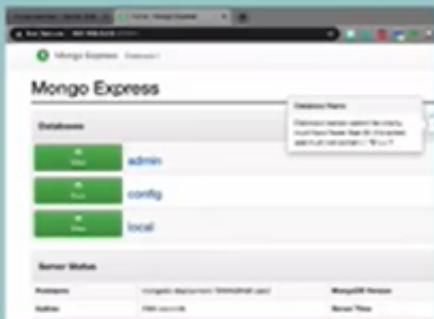


o Express

External Service



Browser Request Flow through the K8s components



Mongo Express
External Service



Mongo Express

Secret

DB User

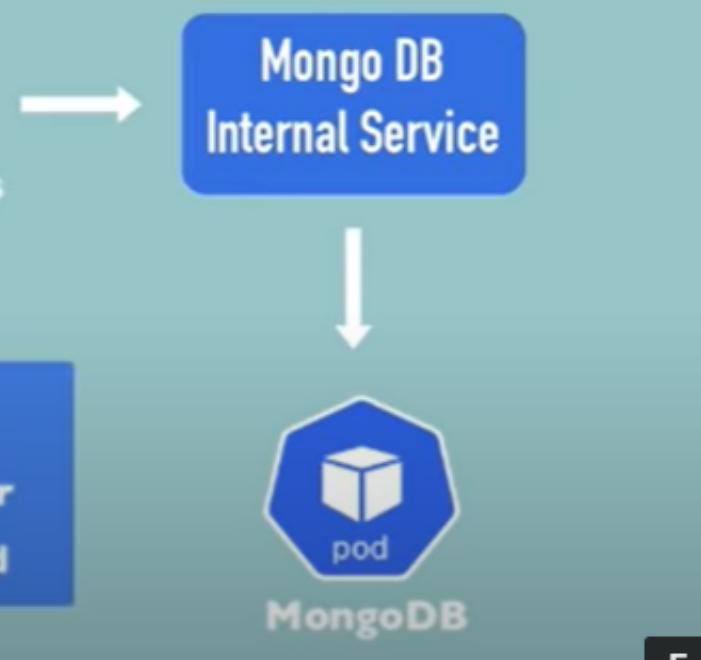
DB Pwd

Deployment Set up using config .yaml file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
  labels:
    app: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
```

Pod Blueprint:

Flow Components



```
spec:  
  containers:  
    - name: mongodb  
      image: mongo
```

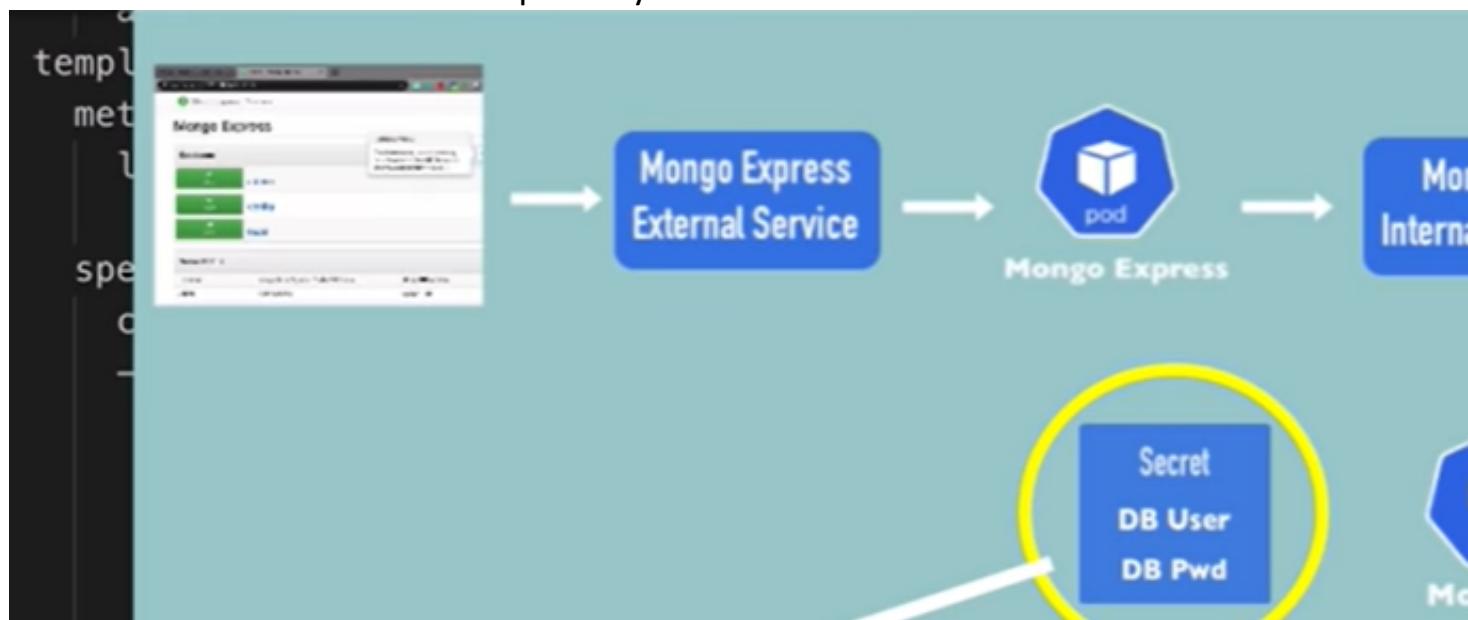
- name: mongodb
- image: mongo

```
template:  
  metadata:  
    labels:  
      app: mongodb  
  spec:  
    containers:  
      - name: mongodb  
        image: mongo  
    ports:  
      - containerPort: 27017  
    env:  
      - name: MONGO_INITDB_ROOT_USERNAME  
        value: |  
      - name: MONGO_INITDB_ROOT_PASSWORD  
        value:
```

Deployment Config checked into repos

Username and Password go here!

Secret Lives in K8s and not in Repository...



File is
itory.

should not



```
- name: MONGO_INITDB_ROOT_PASSWORD  
  value:
```

```
1 apiVersion: v1  
2 kind: Secret  
3 metadata:  
4   name: mongodb-secret  
5 type: Opaque  
6 data:  
7   username:  
8   password:
```

Secret Configuration

- kind:

- metadata / name:

- type:

- data:

Convert plain text into base 64 encrypted text so ideally we should convert username and password in to base64.

```
[~]$ echo -n 'username' | base64  
dXNlcjIyMjQ=
```

```
[~]$ echo -n 'password' | base64  
cGFzc3dvcmQ=
```

```
[~]$
```

! mongo.yaml

! mongo-secret.yaml x

guration File

"Secret"

a random name

"Opaque" - default
for arbitrary key-value
pairs

the actual contents -
in key-value pairs

```
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: mongodb-secret
5  type: Opaque
6  data:
7    mongo-root-username: dXNlcj5hbWU=
8    mongo-root-password: cGFzc3dvcmQ=
9
```

Apply secret:

```
[~]$ cd k8s-configuration/
[k8s-configuration]$ ls
mongo-secret.yaml          mongo.yaml
[k8s-configuration]$ kubectl apply -f mongo-secret/mongodb-secret
secret/mongodb-secret created
```

```
[k8s-configuration]$ kubectl get secret
NAME                      TYPE
default-token-4clzs        kubernetes.io/service-
mongodb-secret             Opaque
```

Deployment

```
env:
- name: MONGO_INITDB_ROOT_USERNAME
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: mongo-root-username
- name: MONGO_INITDB_ROOT_PASSWORD
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: mongo-root-password
```

```
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
data:
  mongo-root-username: dXNlcj5hbWU=
  mongo-root-password: cGFzc3dvcmQ=
```

```
go-secret.yaml
```

```
DATA  
-account-token    3  
                  2
```

```
Secret  
v1  
t  
mongodb-secret  
e  
root-username: dXNlcm5hbWU=  
root-password: cGFzc3dvcmQ=
```

```
26           name: mongodb-secret
27           key: mongo-root-username
28       - name: MONGO_INITDB_ROOT_PASSWORD
29           value:
30
```

23) Lets Create Internal Service related Config in same .yaml file :

--- ==> 3 dot is separator of another .yaml file in to same deployment.

```
mongo.yaml • mongo-secret.yaml
...
28   - name: MONGO_INITDB_
29     valueFrom:
30       secretKeyRef:
31         name: mongodb-s
32         key: mongo-root
33 ---
34 apiVersion: v1
35 kind: Service
36 metadata:
37   name: mongodb-service
38 spec:
39   selector:
40     app: mongodb
41   ports:
42     - protocol: TCP
43       port: 27017
44       targetPort: 27017
45
```

Service Configuration

- kind:

- metadata / name:

- selector:

- ports:

port:

targetPort:

```
[k8s-configuration]$ kubectl get service
NAME          TYPE        CLUSTER-IP      EXTERNA
kubernetes    ClusterIP   10.96.0.1      <none>
mongodb-service ClusterIP  10.96.86.105  <none>
[k8s-configuration]$
```

Configuration File

"Service"

a random name

to connect to Pod
through label

Service port

containerPort of
Deployment

EXTERNAL-IP	PORT(S)	AGE
one>	443/TCP	36m
one>	27017/TCP	51s

Get all components and filter them using Mongo..

- 24) Now create the mongo express server which can be accessed from external service...

External details we will put in config-map..

```
! mongo.yaml      ! Untitled-1 • ! m
16   spec:
17     containers:
18       - name: mongo-express
19         image: mongo-express
20         ports:
21           - containerPort: 8081
22         env:
23           - name: ME_CONFIG_MONGO
24             valueFrom:
25               secretKeyRef:
26                 name: mongodb-secret
27                 key: mongo-root-u
28           - name: ME_CONFIG_MONGO
29             valueFrom:
30               secretKeyRef:
31                 name: mongodb-secret
32                 key: mongo-root-p
33           - name: ME_CONFIG_MONGO
34             value: Y
```

- external config
- centralized
- other components



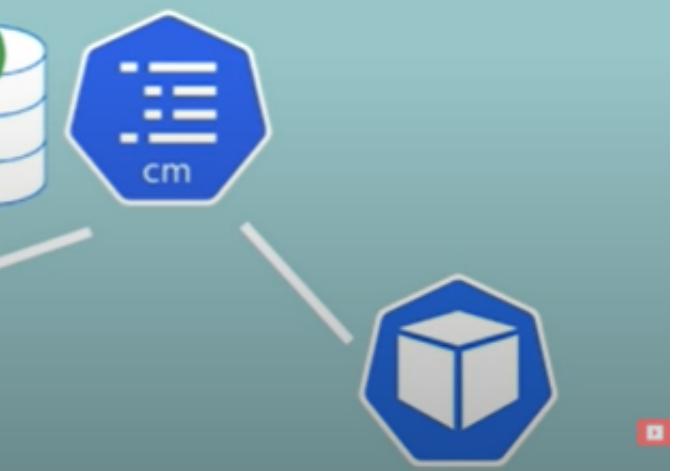
```
! mongo.yaml ✘ ! mongo-express.yaml
16   spec:
17     containers:
18       - name: mongo-express
19         image: mongo-express
20         ports:
21           - containerPort: 8081
22         env:
23           - name: MONGO_INITDB_ROOT_PASSWORD
24             valueFrom:
25               secretKeyRef:
26                 name: mongodb-secret
27                 key: mongo-root-password
```

ConfigMap

ConfigMap

Configuration

Components can use it



to Configuration File

```
33 ---  
34   apiVersion: v1  
35   kind: Service  
36   metadata:  
37     name: mongodb-service  
38   spec:  
39     selector:  
40       app: mongodb  
41     ports:  
42       - protocol: TCP  
43         port: 27017  
44         targetPort: 27017
```

- kind:

- metadata / na

- data:

```
ports:  
- containerPort: 8081  
env:  
- name: ME_CONFIG_MONGODB_ADMINUSERNAME  
  valueFrom:  
    secretKeyRef:  
      name: mongodb-secret  
      key: mongo-root-username  
- name: ME_CONFIG_MONGODB_ADMINPASSWORD  
  valueFrom:  
    secretKeyRef:  
      name: mongodb-secret  
      key: mongo-root-password  
- name: ME_CONFIG_MONGODB_SERVER  
  valueFrom:  
    configMapKeyRef:  
      name: mongodb-secret  
      key: mongo-root-password
```

configMapKeyRef ins
secretKeyRef

```
[k8s-configuration]$ kubectl apply -f mongo-confi  
configmap/mongodb-configmap created  
[k8s-configuration]$ kubectl apply -f mongo-expre  
deployment.apps/mongo-express created  
[k8s-configuration]$ kubectl get pod
```

"ConfigMap"

name: a random name

the actual contents -
in key-value pairs

instead of

configmap.yaml

i

configmap.yaml

```

NAME          READY   STATUS
E
mongo-express-797845bd97-p9grr      0/1     Con
mongodb-deployment-78444d94d6-zsrc1  1/1     Run
m
[k8s-configuration]$ kubectl get pod
NAME          READY   STATUS
E
mongo-express-797845bd97-p9grr      0/1     Con
s
mongodb-deployment-78444d94d6-zsrc1  1/1     Run
m
[k8s-configuration]$ █

```

- 25) Now Create external Service which could access Mongo Express.
 Node Port :: Used for external Service.

```

! mongo.yaml           ! mongo-express.yaml • !
31   name: mongodb-secret
32   key: mongo-root-password
33   - name: ME_CONFIG_MONGODB_SERV
34     valueFrom:
35       configMapKeyRef:
36         name: mongodb-configmap
37         key: database_url
38   ---
39   apiVersion: v1
40   kind: Service
41   metadata:
42     name: mongo-express-service
43   spec:
44     selector:
45       app: mongo-express
46       type: LoadBalancer
47

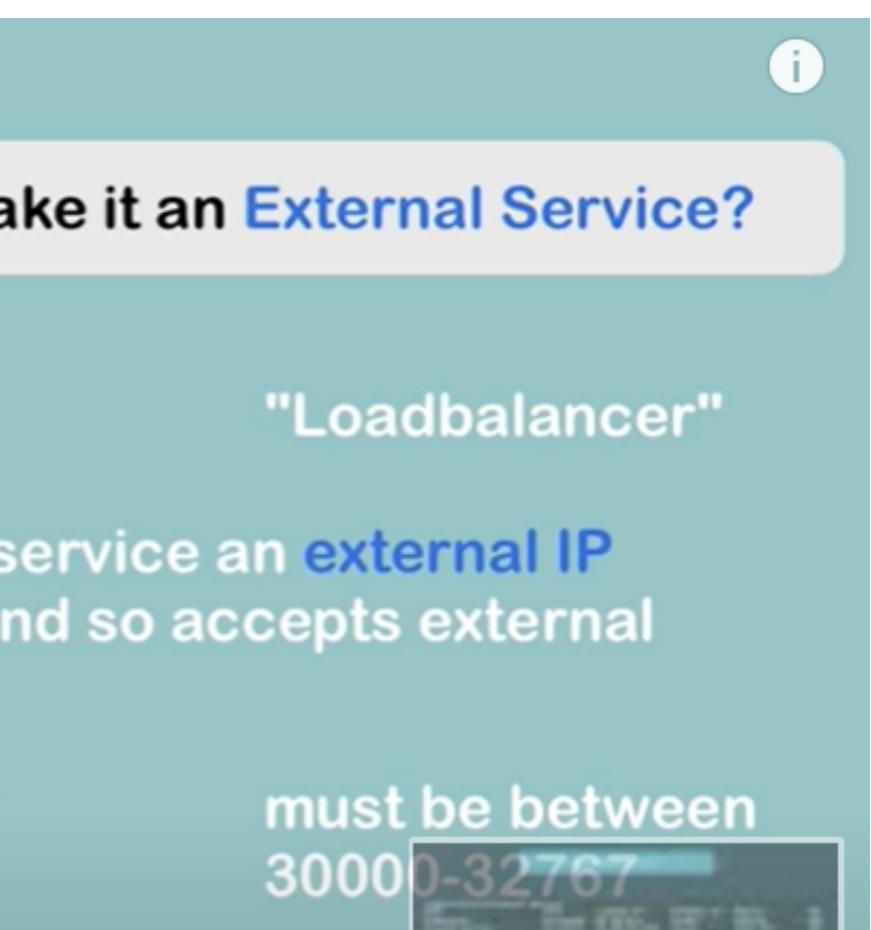
```

How to ma

- **type:**
 ..assigns s
 address a
 requests

- **nodePort:**

STATUS	RESTARTS	AG
ContainerCreating	0	5s
Running	0	23s



```
47     ports:
48       - protocol: TCP
49         port: 8081
50         targetPort: 8081
51         nodePort: 30
```

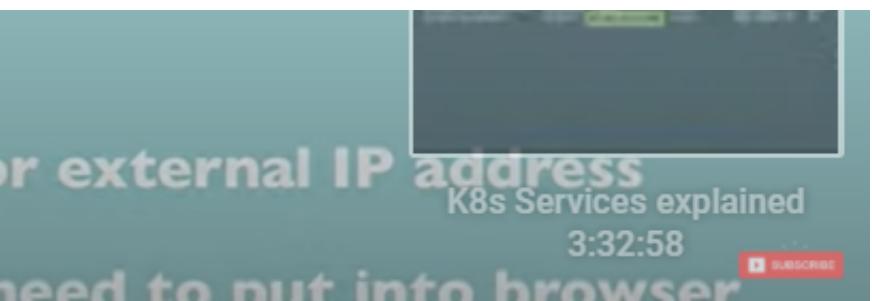
Port fo

```
---
apiVersion: v1
kind: Service
metadata:
  name: mongo-express-service
spec:
  selector:
    app: mongo-express
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      nodePort: 30000
```

Port you r

====> External Service Node Creation.

```
[k8s-configuration]$ kubectl apply -f mongo-ex
deployment.apps/mongo-express unchanged
service/mongo-express-service created
[k8s-configuration]$ kubectl get service
NAME          AGE   TYPE           CLUSTER-IP
kubernetes    62m   ClusterIP    10.96.0.1
mongo-express-service 6s   LoadBalancer  10.96.1.1
mongodb-service 26m  ClusterIP    10.96.8.1
[k8s-configuration]$
```

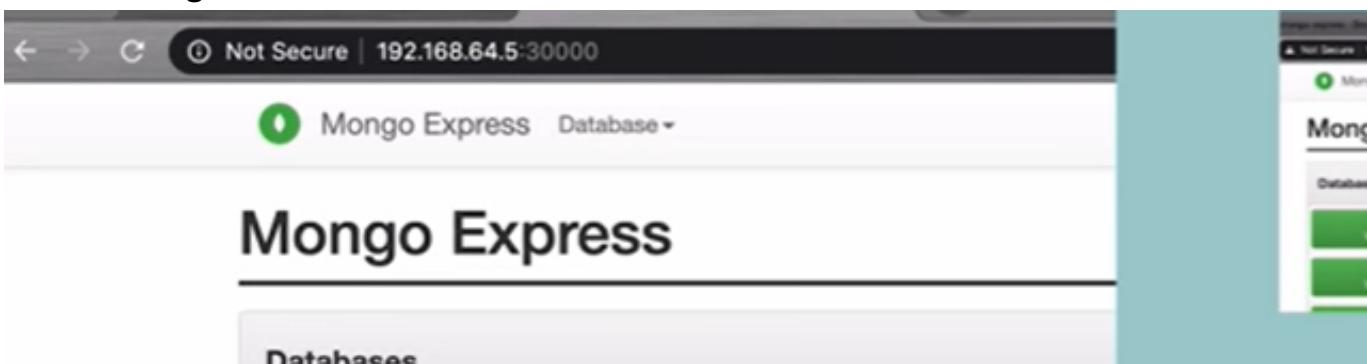


R-IP	EXTERNAL-IP	PORT(S)
0.1	<none>	443/TCP
178.16	<pending>	8081:30000/
36.105	<none>	27017/TCP

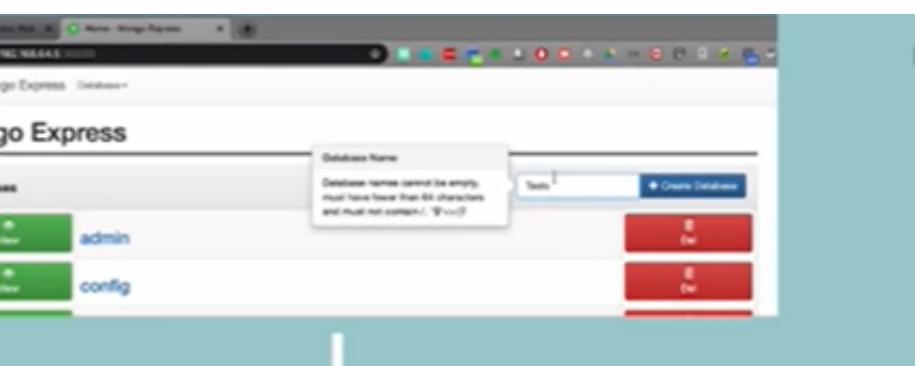
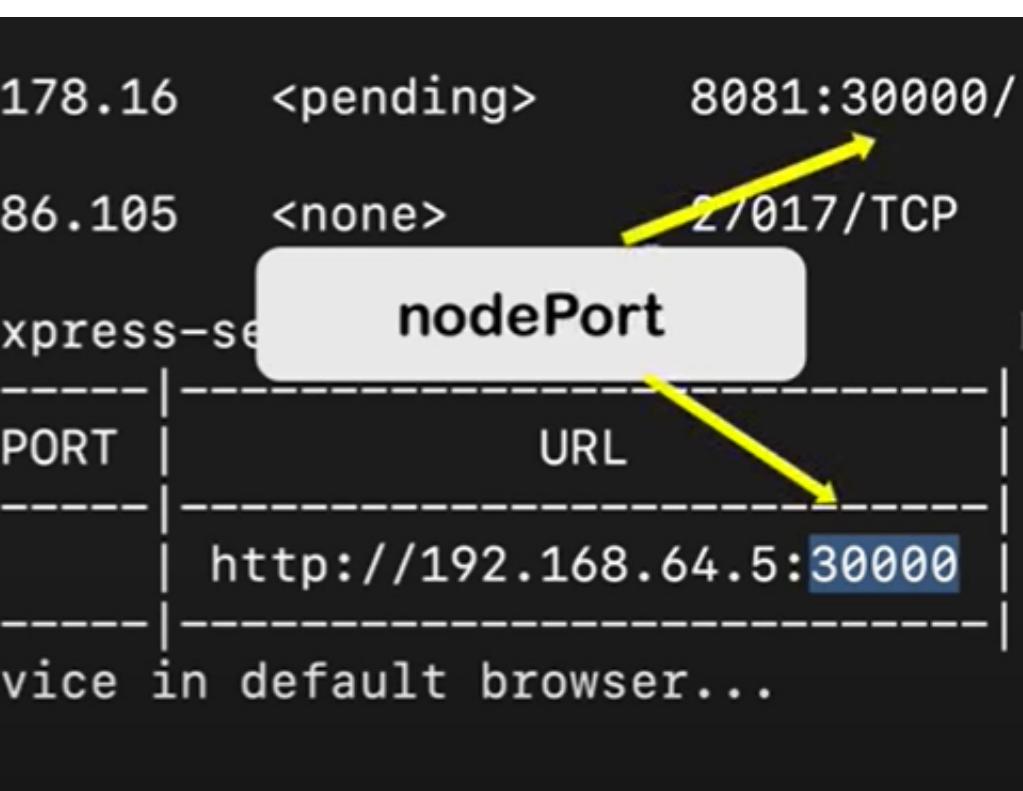
```
[k8s-configuration]$ kubectl get service
NAME                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes                ClusterIP   10.96.0.1     <none>        443/TCP         67m
mongo-express-service     LoadBalancer 10.96.1.100   192.168.64.5  30000/TCP       5m7s
mongodb-service           ClusterIP   10.96.1.101   <none>        27017/TCP       32m
[k8s-configuration]$ minikube service mongo-express
```

```
67m
mongo-express-service     LoadBalancer 10.96.1.100   192.168.64.5  30000/TCP       5m7s
mongodb-service           ClusterIP   10.96.1.101   <none>        27017/TCP       32m
[k8s-configuration]$ minikube service mongo-express
|-----|-----|-----|
| NAMESPACE | NAME | TARGET |
|-----|-----|-----|
| default   | mongo-express-service | <none> |
|-----|-----|-----|
💡 Opening service default/mongo-express-service
[k8s-configuration]$
```

Full Working:



R-IP	EXTERNAL-IP	PORT(S)
0.1	<none>	443/TCP
178.16	<pending>	8081:30000/
86.105	<none>	27017/TCP
xpress-service		



The screenshot shows a MongoDB interface. On the left, there's a sidebar with a search bar and a "Databases" section containing four items: "Test-db", "admin", "config", and "local", each with a "View" button. To the right is a main panel titled "Server Status" with two rows: "Hostname" (mongodb-deployment-78444d94d6-zsrc1) and "Uptime" (2344 seconds).

Server Status	
Hostname	mongodb-deployment-78444d94d6-zsrc1
Uptime	2344 seconds

26)

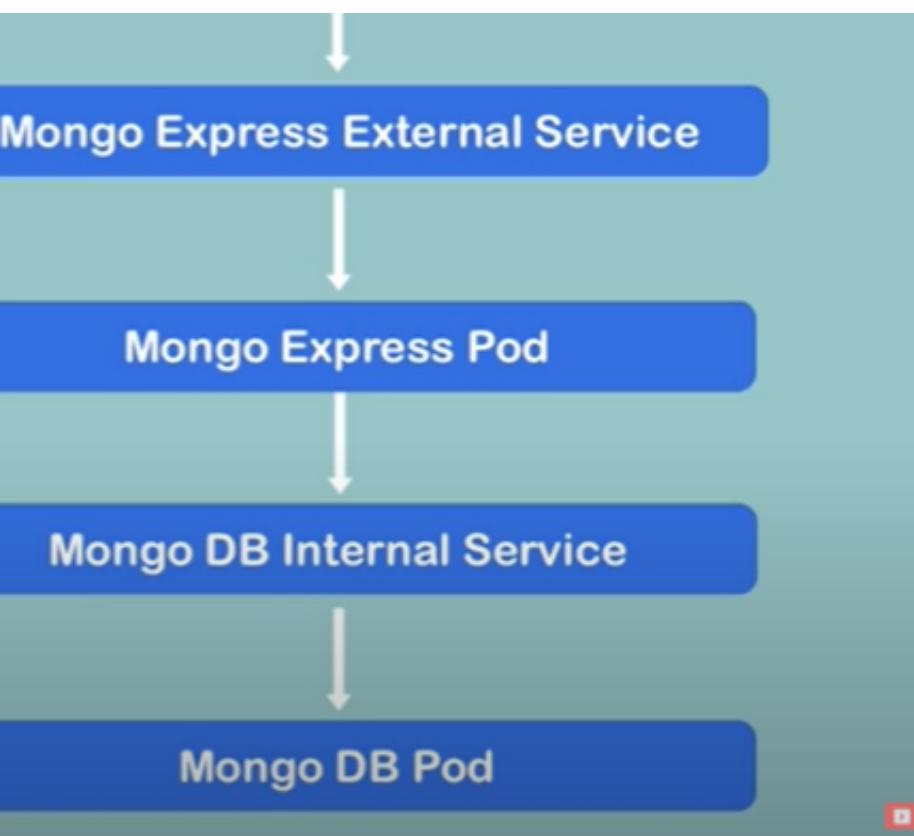
===== Istio and Service Mesh=====

What is Istio? What is Se

Istio is a Service Mesh

Service Mesh manages communication

Challenges of a microservice



Service Mesh?

between microservices

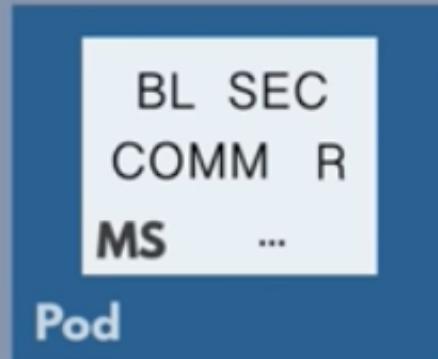
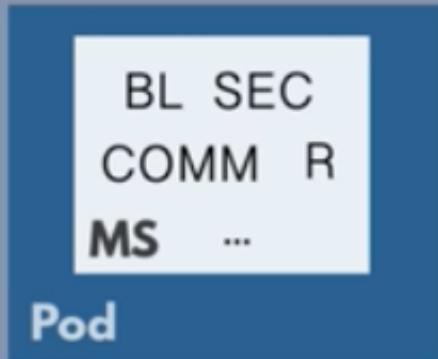
ce architecture

Security

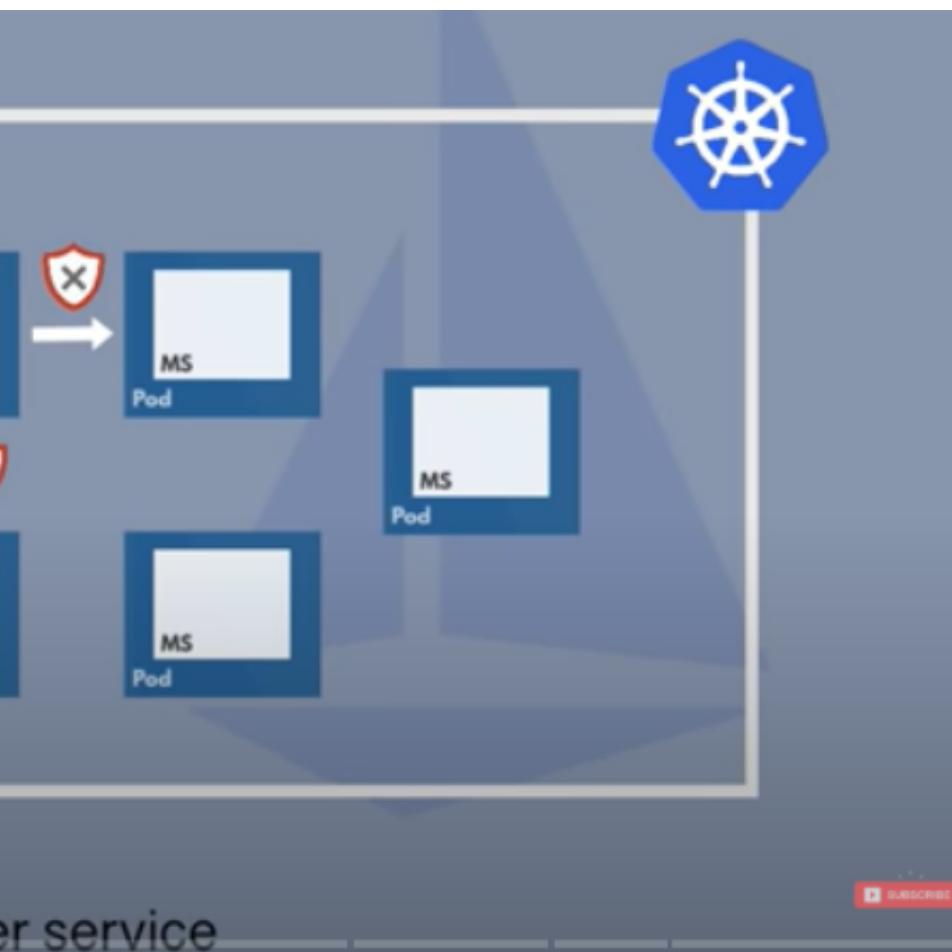


- ▶ communication inside cluster not secured
- ▶ ~~every service inside the cluster can talk to any other~~

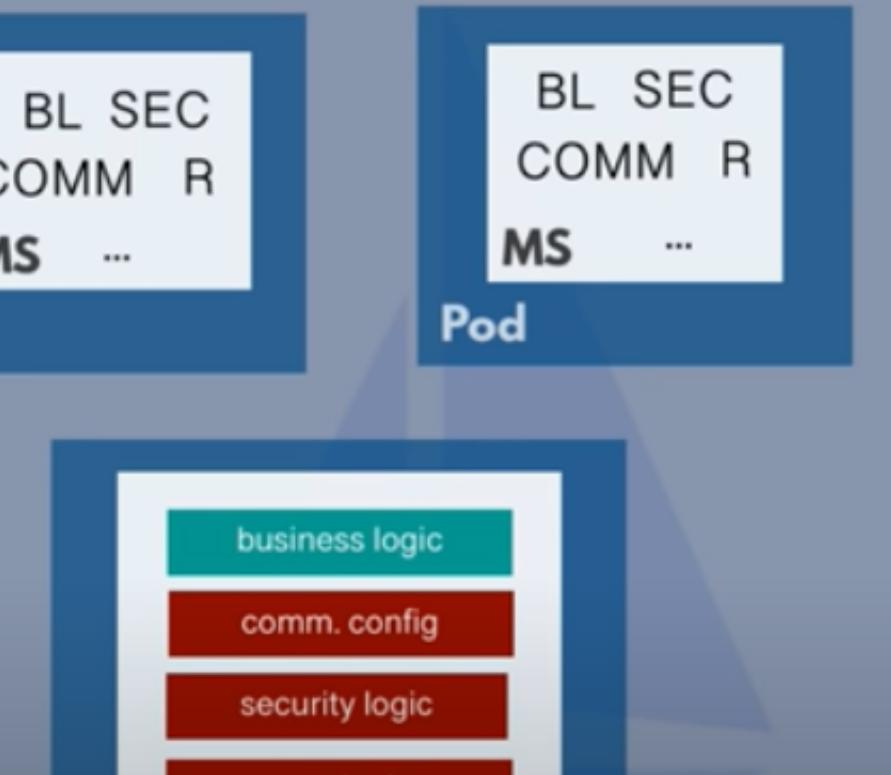
Challenges of a microservice



- ✓ business logic (BL)
- ✓ communication configurations (COMM)
- ✓ security logic (SEC)



Microservice architecture



- ✓ retry logic (R)

- ✗ Service Discovery (SD)

Challenges of a microservice



BL SEC
COMM R
MS ...

Pod



BL SEC
COMM R
MS ...

Pod



Pod

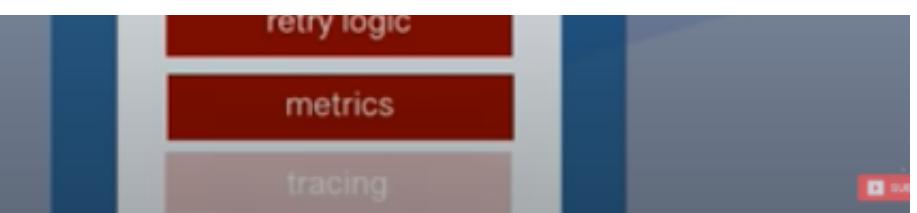
- ✗ These **NON business logic** must be added to **each application**
- ✗ Developers don't work on actual service/application



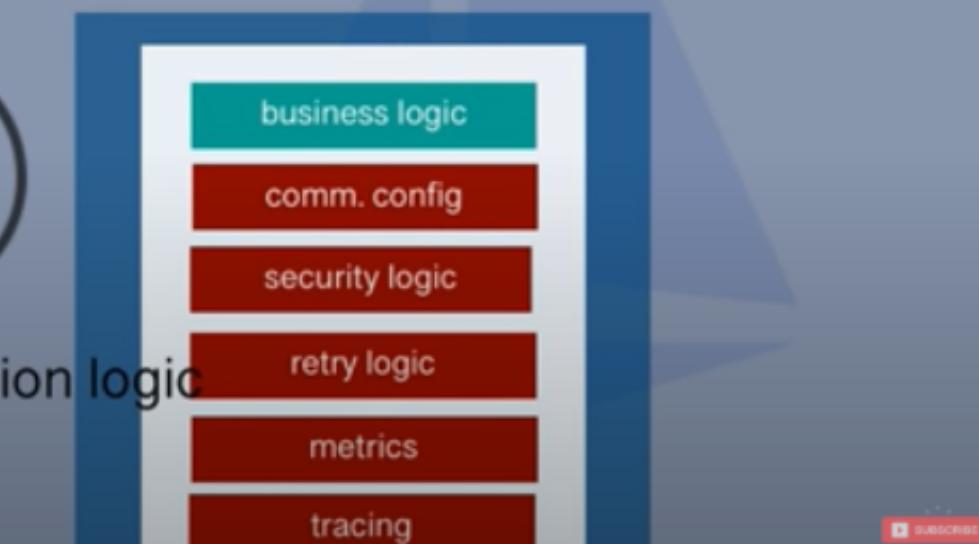
Solution: Service Mesh with Sidecar Proxy

Sidecar Proxy

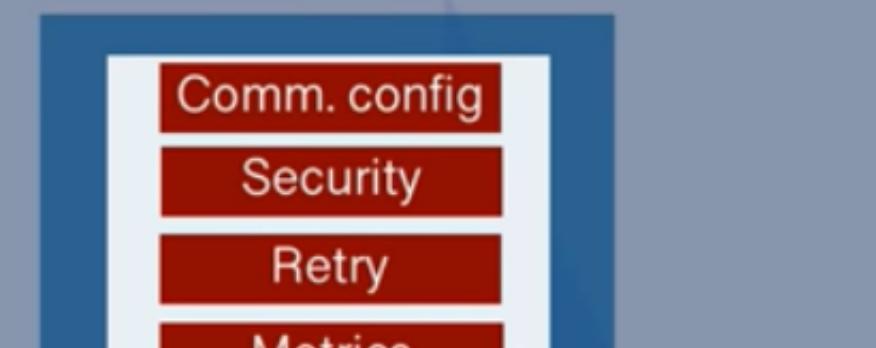
- ▶ handles these networking logic
- ▶ acts as a **Proxy**



Service architecture



With Sidecar Pattern

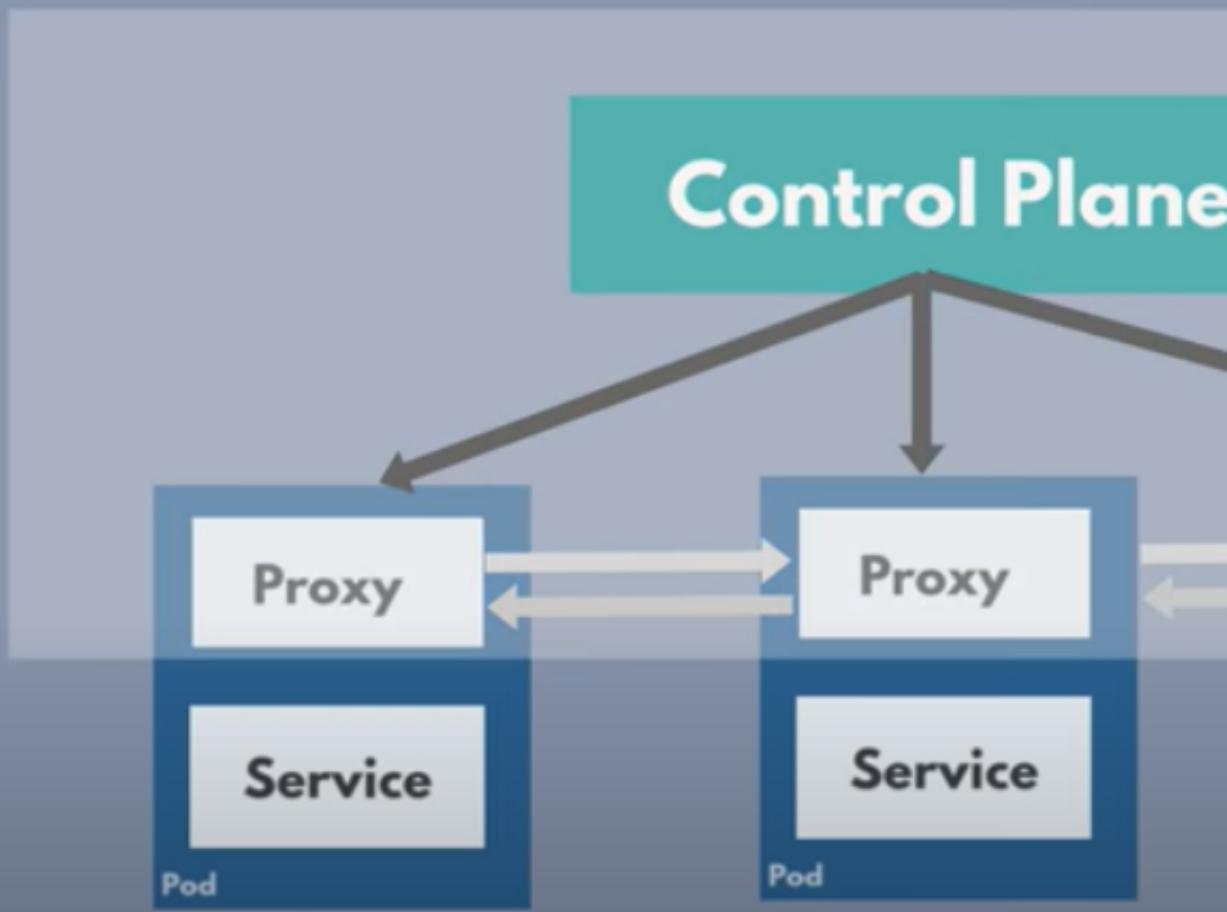


- ▶ third-party application
- ▶ cluster operators can configure it easily

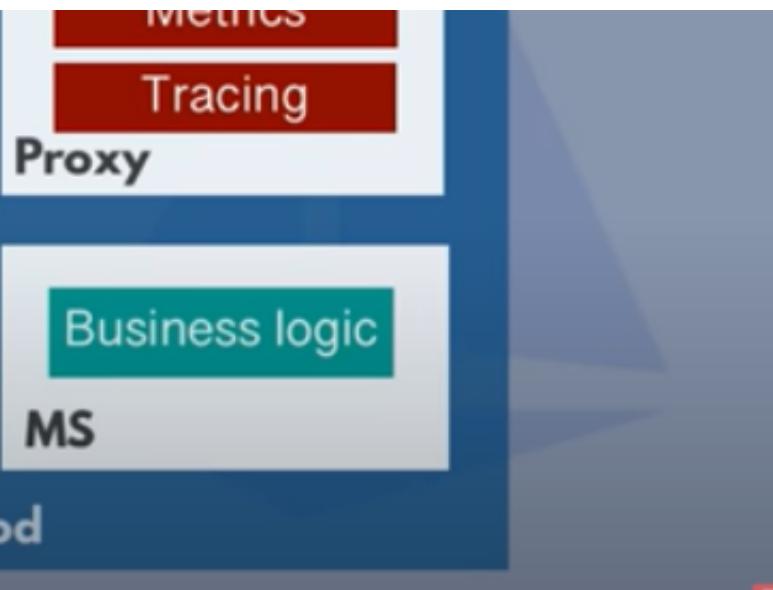
Control Plane

- ▶ Control Plane injects the Sidecar Proxy

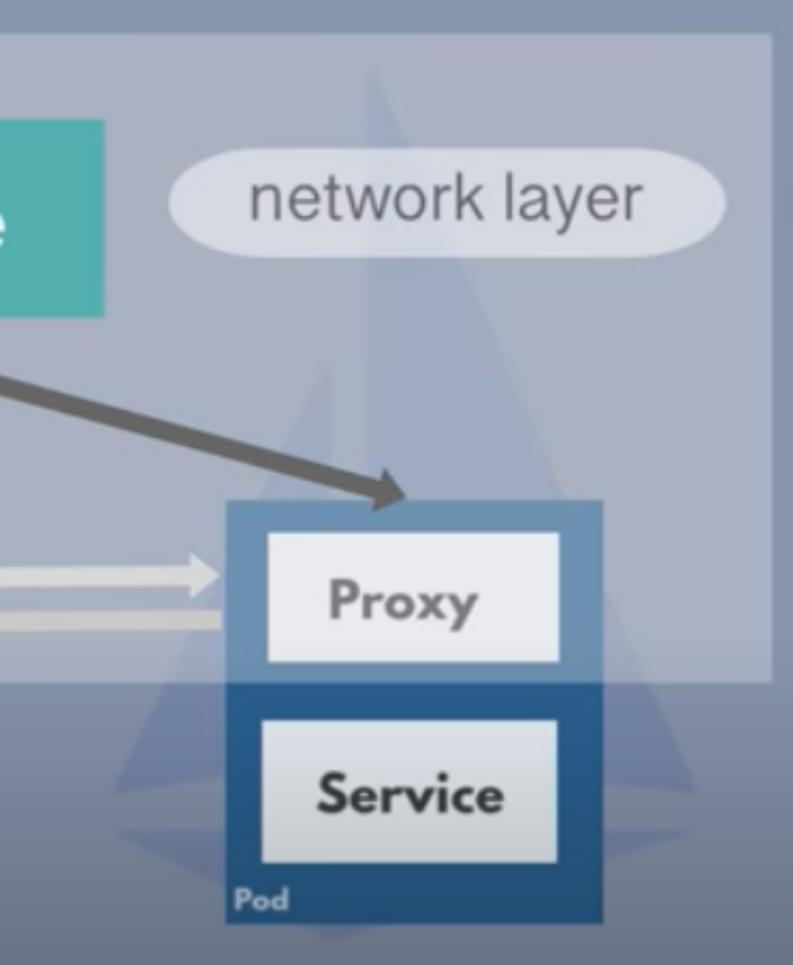
Solution: Service Mesh with Sidecar



Core feature: Traffic Splicing



Sidecar Pattern



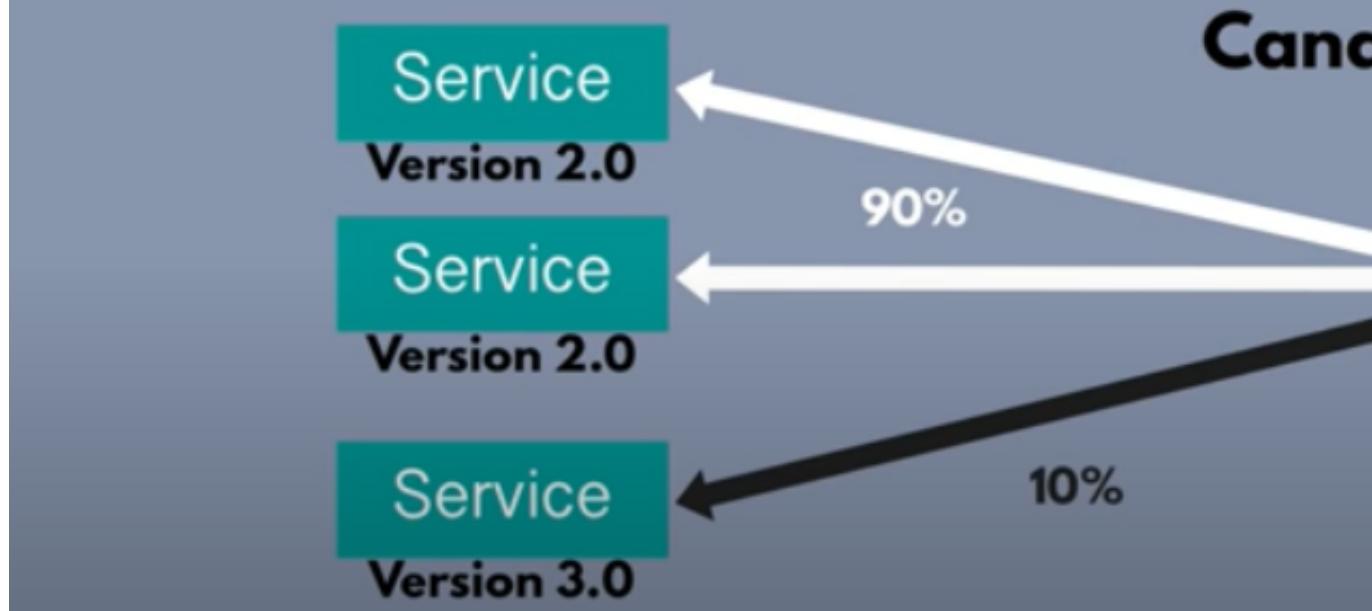
splitting



Payment Service



release new versions
breaking the application



- 26) ISTIO Architecture =====>

Istio Architecture

Service Mesh is a Pattern or Paradigm



Istio is an implementation

ISTIO Architecture

Control Plane



Istiod

Pilot

sions without worrying about
lication

ary Deployment

Webserver
Service

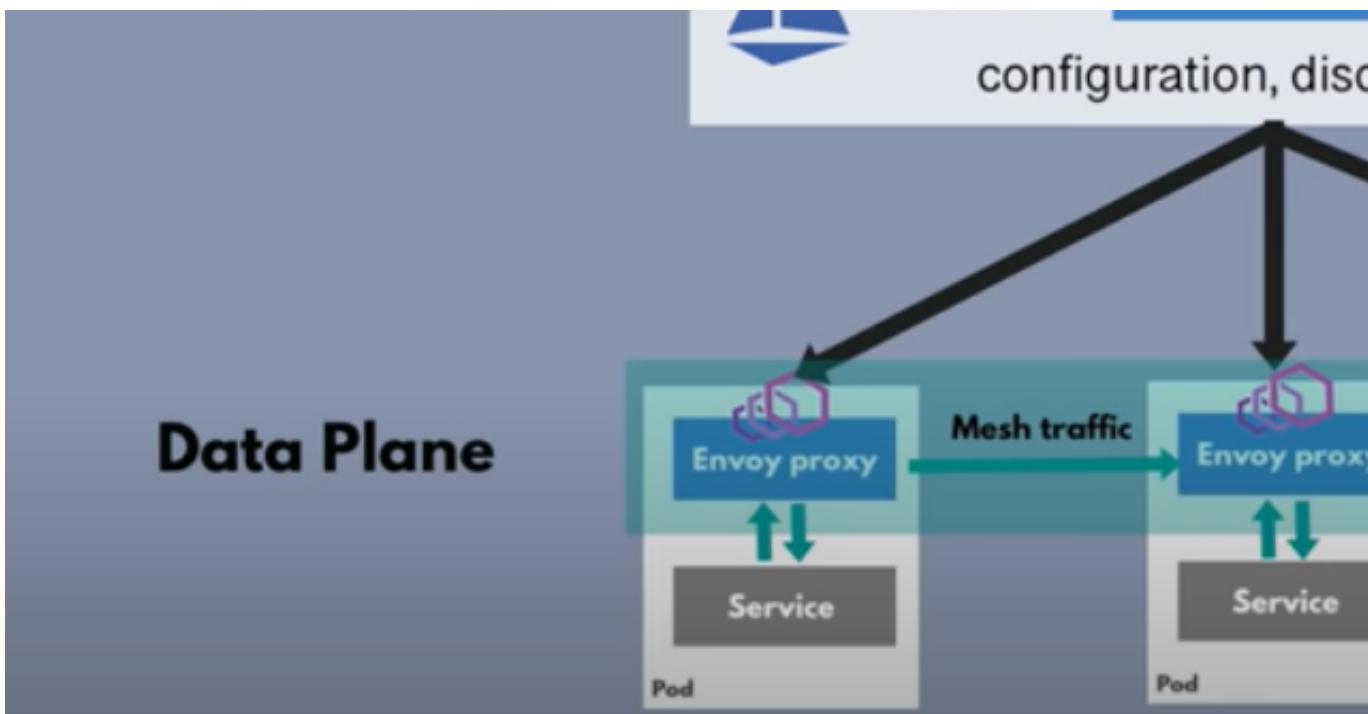


e

lementation

Citadel

Galley



How to configure Istio

How do we configure all these features for our microservice application?



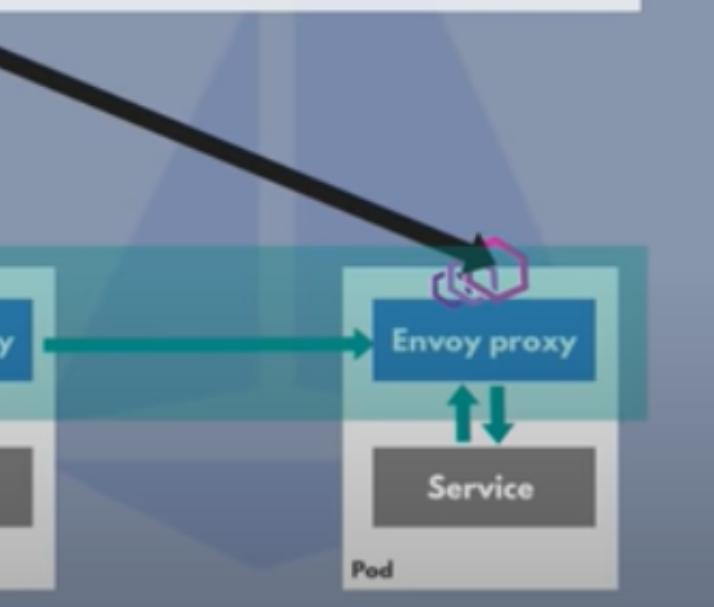
you don't have to adjust Deployment and Service



Istio configuration **separate** from application code

How to configure Istio

covery, certificates



stio?



vice K8s YAML files

configuration

stio?



Istio is configured with **Kubernetes YAML** files



no need to learn Istio specific language

CRD

- ▶ extending the Kubernetes API
- ▶ **custom** Kubernetes components
third-party technologies (like Istio)
- ▶ can be used like any other native

CRD's

traffic routing

which services can
communicate?

traffic split

- ▶ Istio configuration

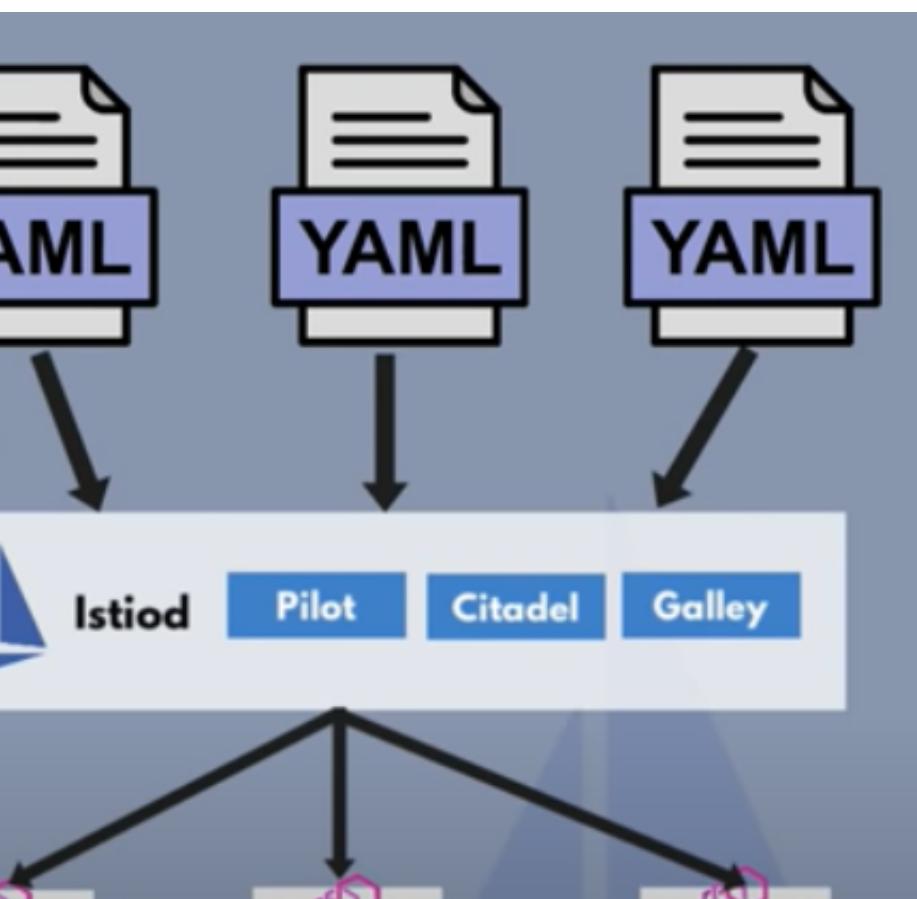
les



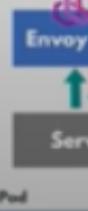
/object for e.g.
(Istio, Prometheus, etc.)

Create Kubernetes objects

SUBSCRIBE



retry rules



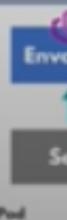
- ▶ We create CRD's

V
Se
Y

- ▶ Istiod converts these high level routing rules into Envoy-specific configurations



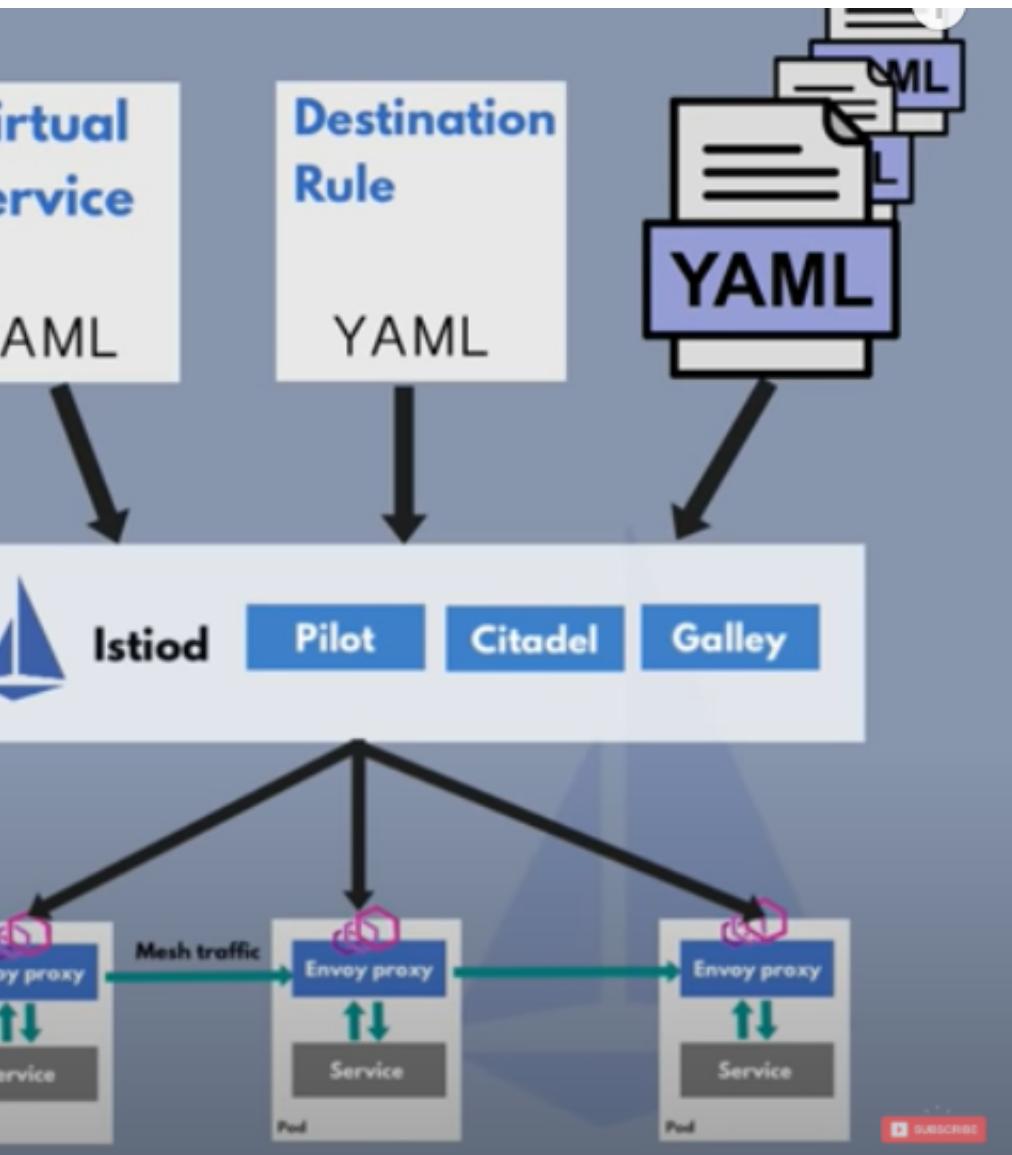
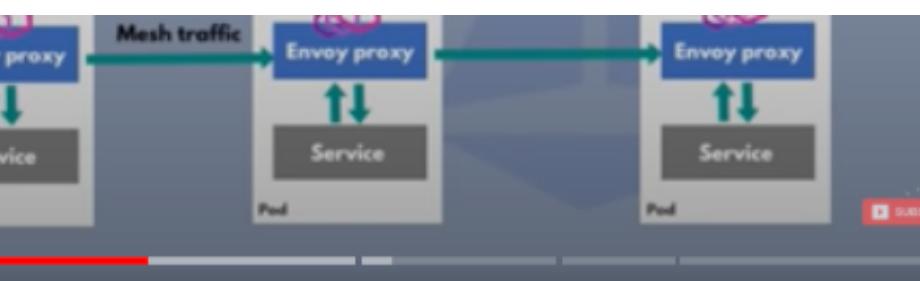
- ▶ Configuration is propagated into Proxy sidecars



We don't configure Proxies,
we configure Istiod

Virtu
Serv

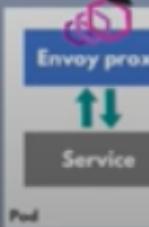
YAM



We configure Istiod



- ▶ Proxies can communicate without connecting to Istiod



Dynamic Service Discovery

Istio

- ✓ configuration
- ✓ service discovery



Pilot

configuration
discovery

Internal Registry for Services
& their endpoints

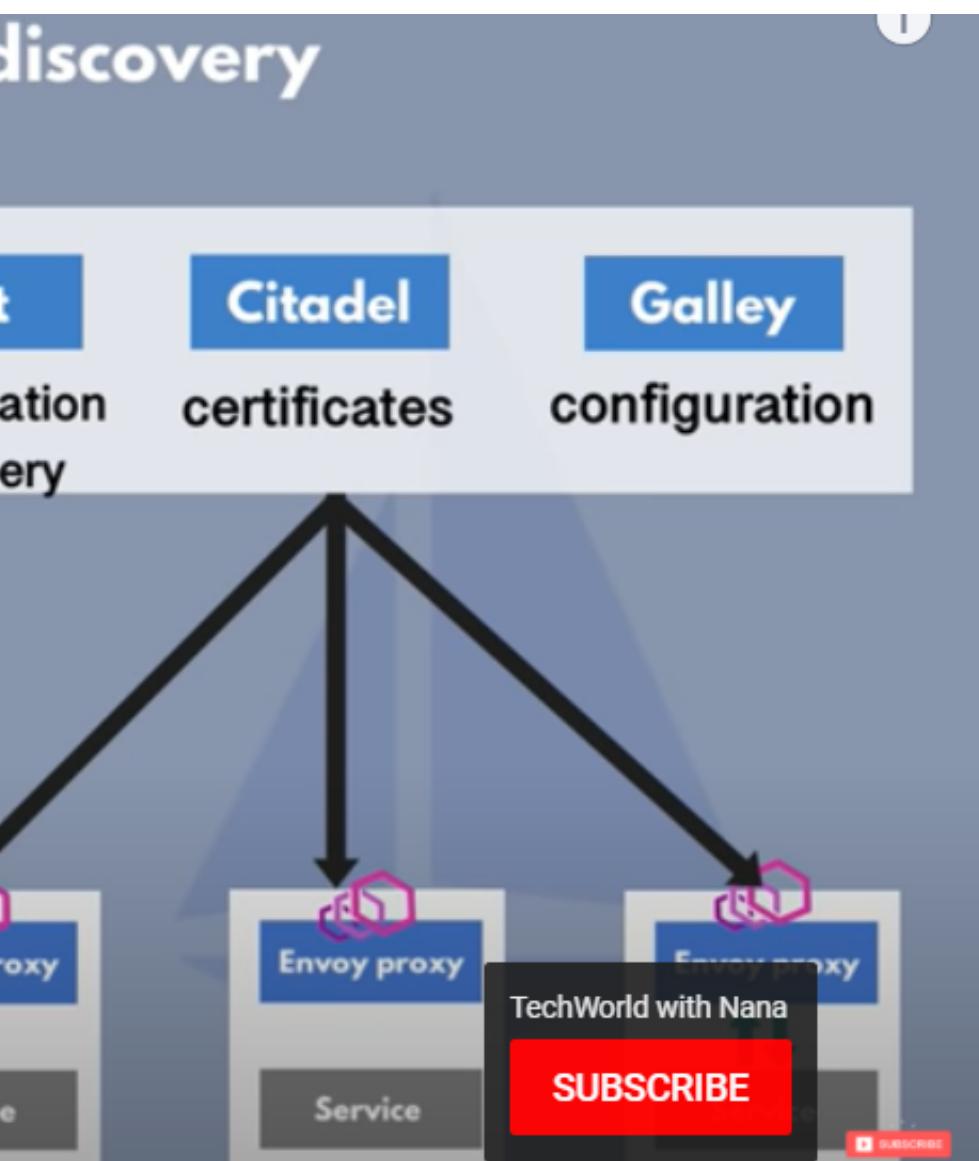
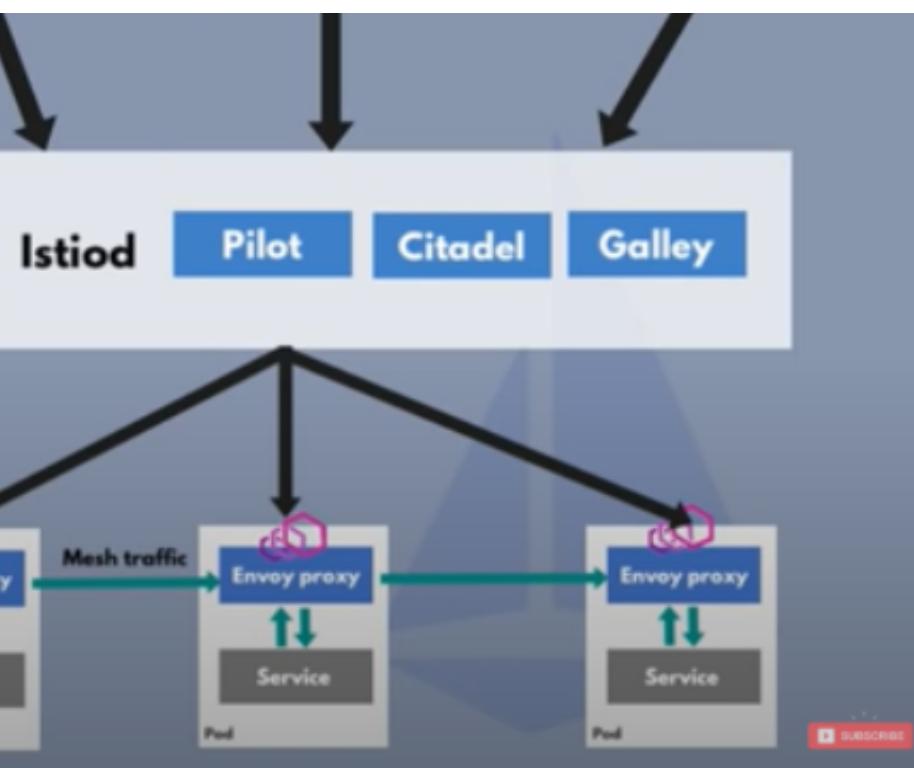


new microservice gets
registered automatically



Envoy proxy

Service



Security - Certificate Management

Istio

- configuration
- service discovery
- certificate management



Pilot

configuration
discovery

secure TLS communication between
microservices

Envoy

Service

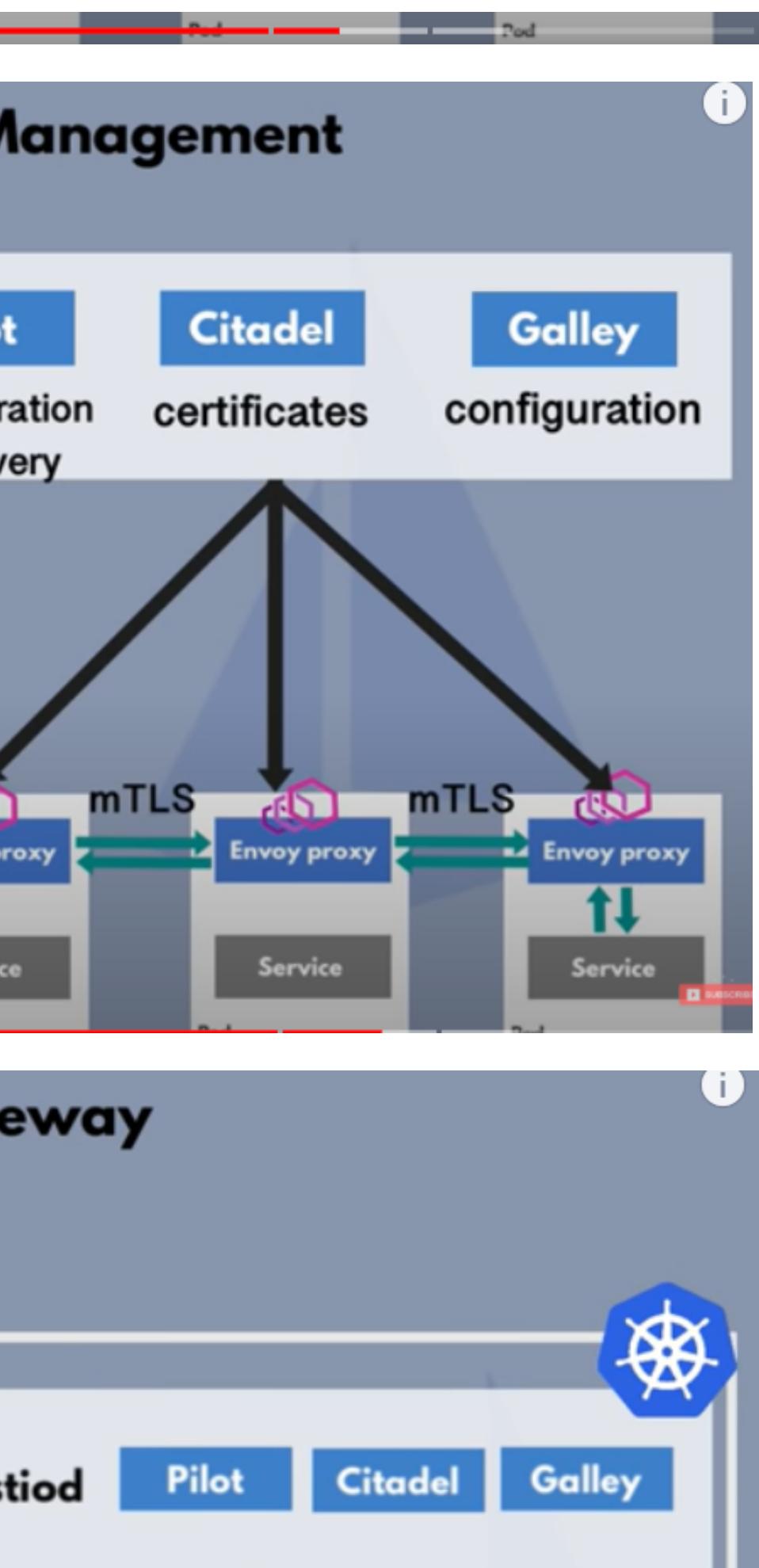
Istio Ingress Gateway

entrypoint to your cluster

alternative to nginx Ingress
Controller



Istio

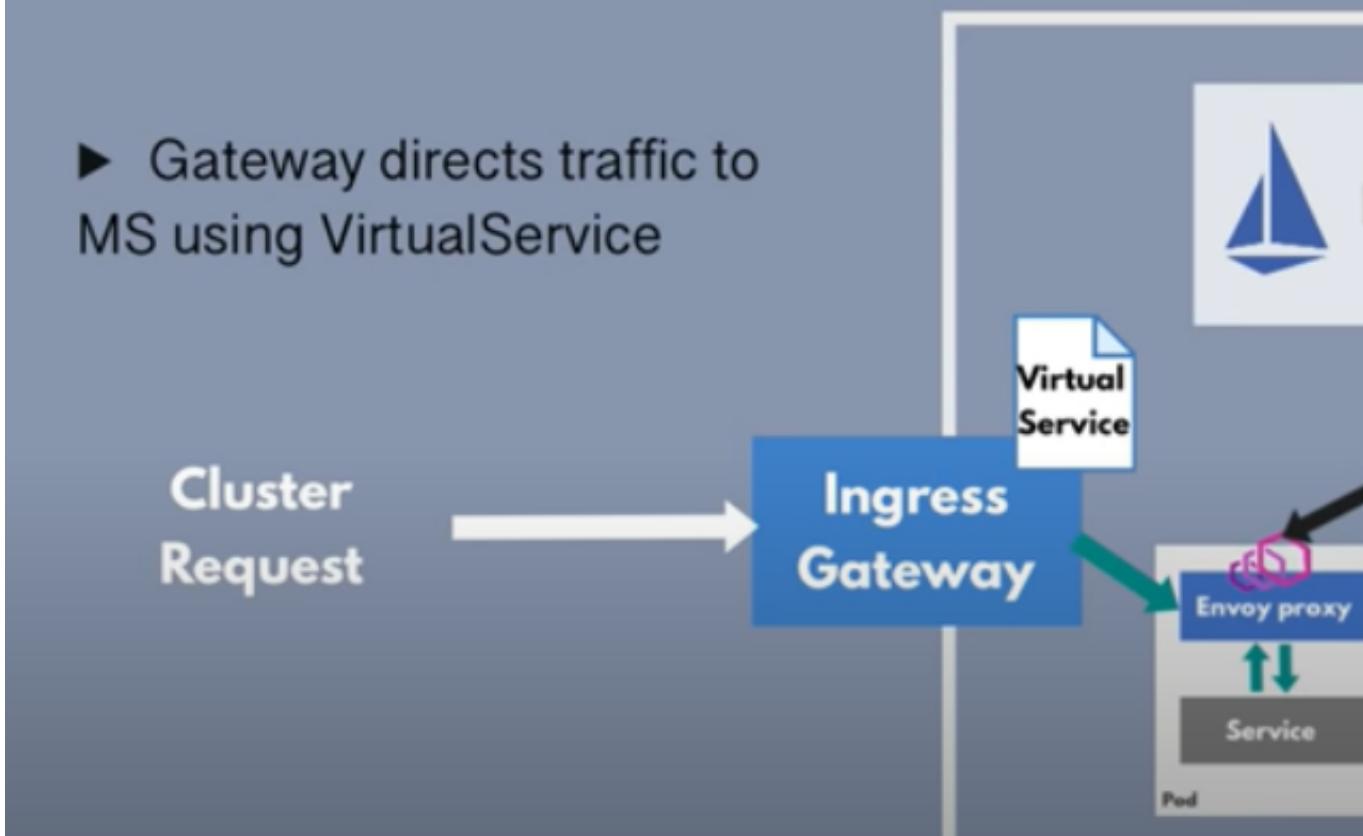




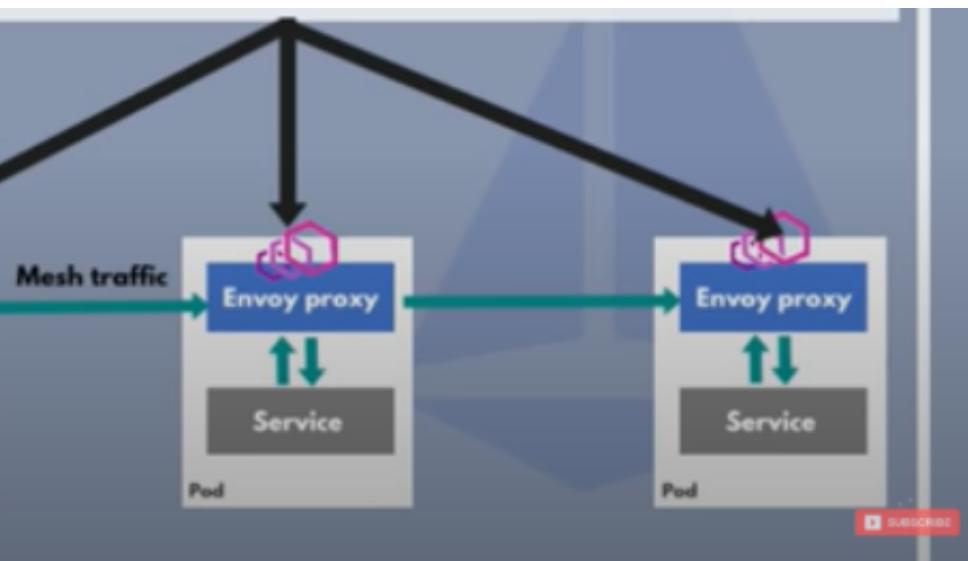
Istio Ingress Gate

entrypoint to your cluster

- ▶ Gateway directs traffic to MS using VirtualService



Istio: Traffic F



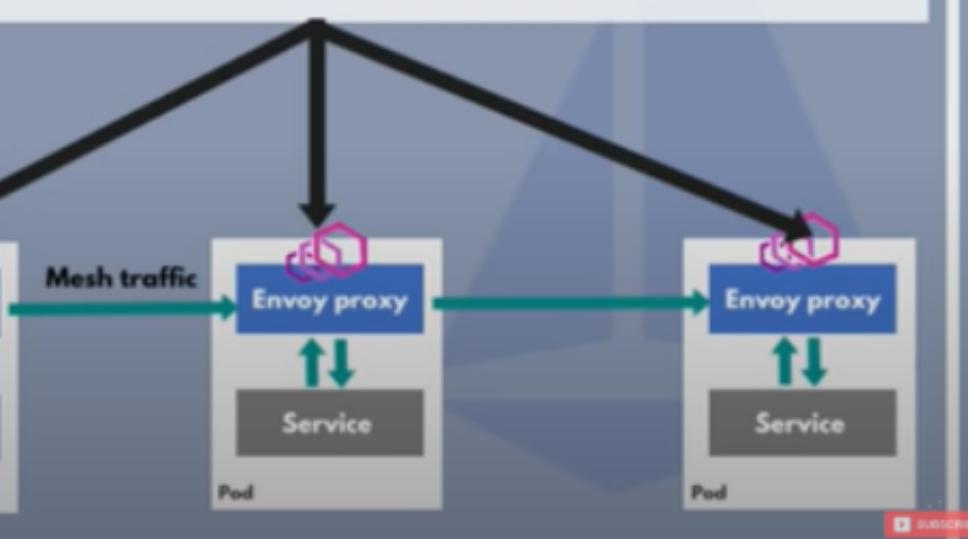
teway

Istiod

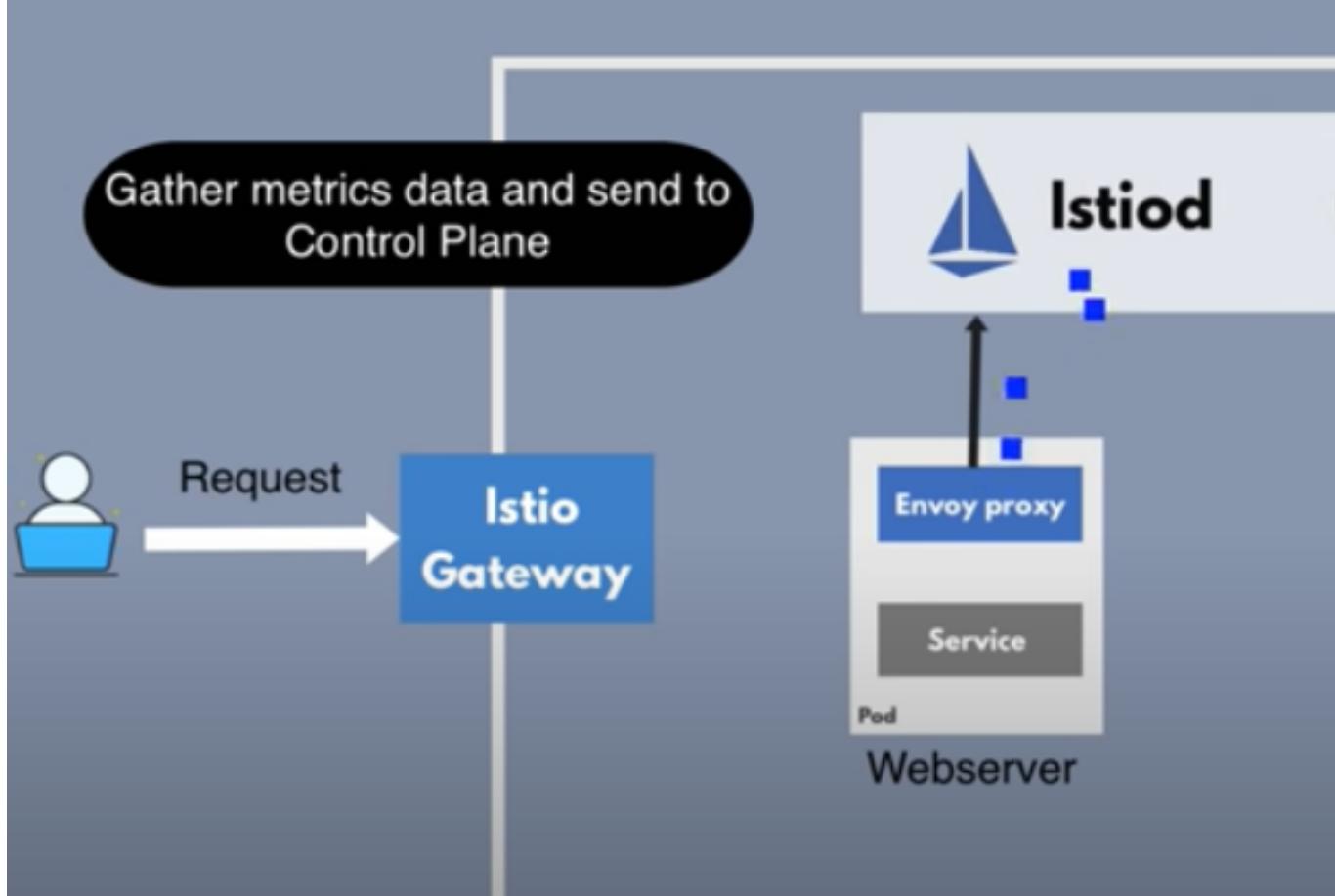
Pilot

Citadel

Galley



low



What is EKS?

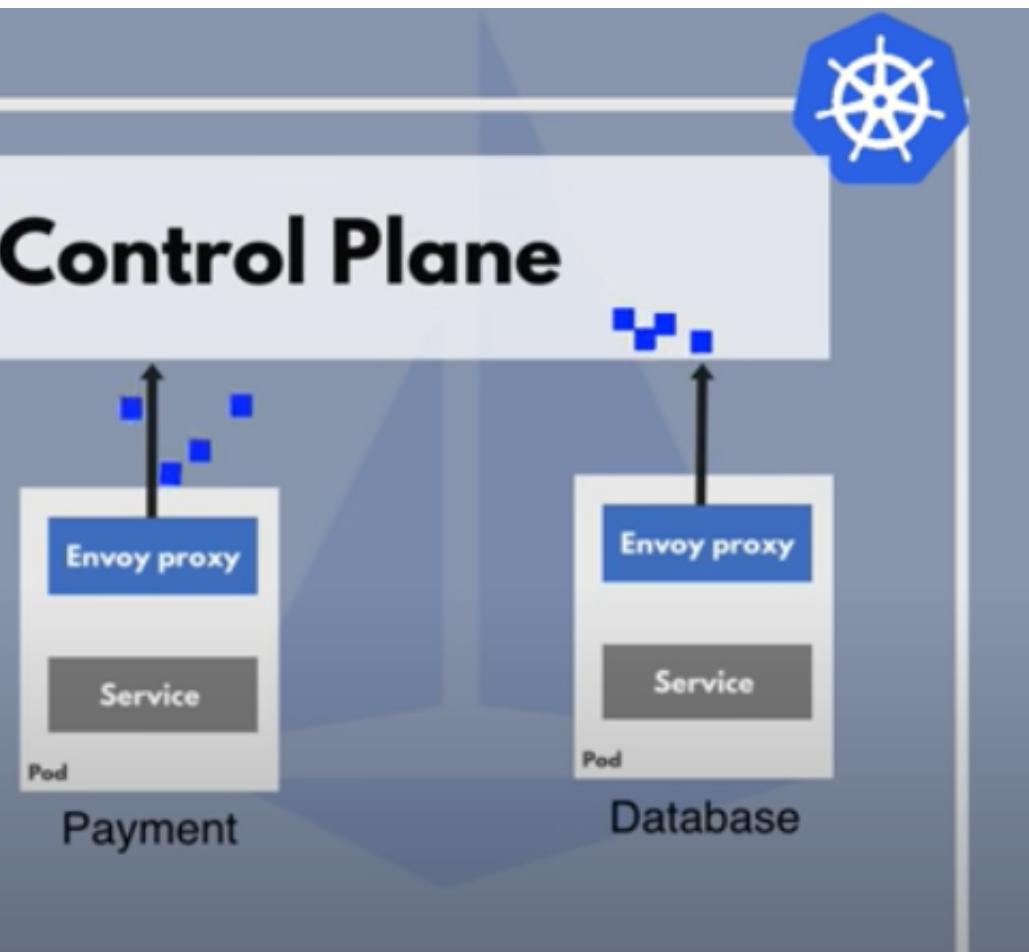
Managed Kubernetes Service



AWS manages Master Nodes



Control Plane



Kubernetes Cluster





Necessary apps pre-installed

Scaling and backups



You can focus on deploying your

Managed k8 Service :: No need to bother about Master Nodes.. Only our headache is to bother about Worker Nodes.

How to use EKS?

1 | Setup or preparation steps

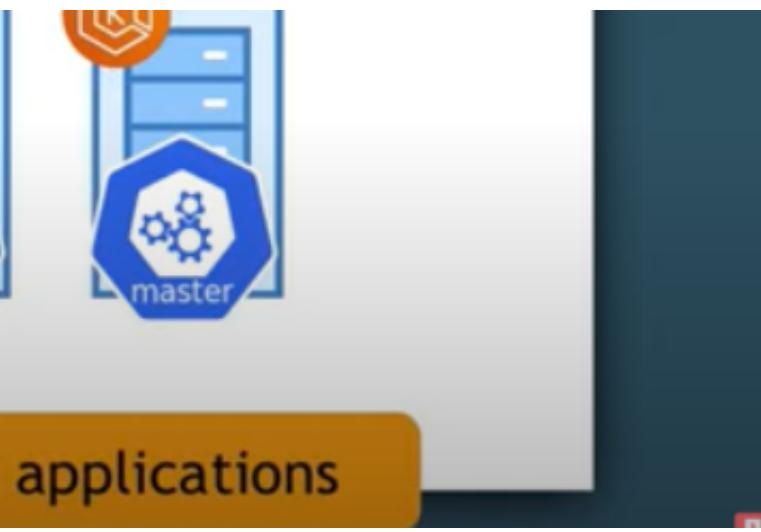
- ▶ Create AWS account (free tier)
- ▶ Create a VPC
- ▶ Create an IAM role with Security Group

IAM role

Se

...which means: Create AWS user with list





2 | Create Cluster Control Plane



- ▶ choose cluster name, k8s version
- ▶ choose region and VPC for your cluster
- ▶ set security for your cluster

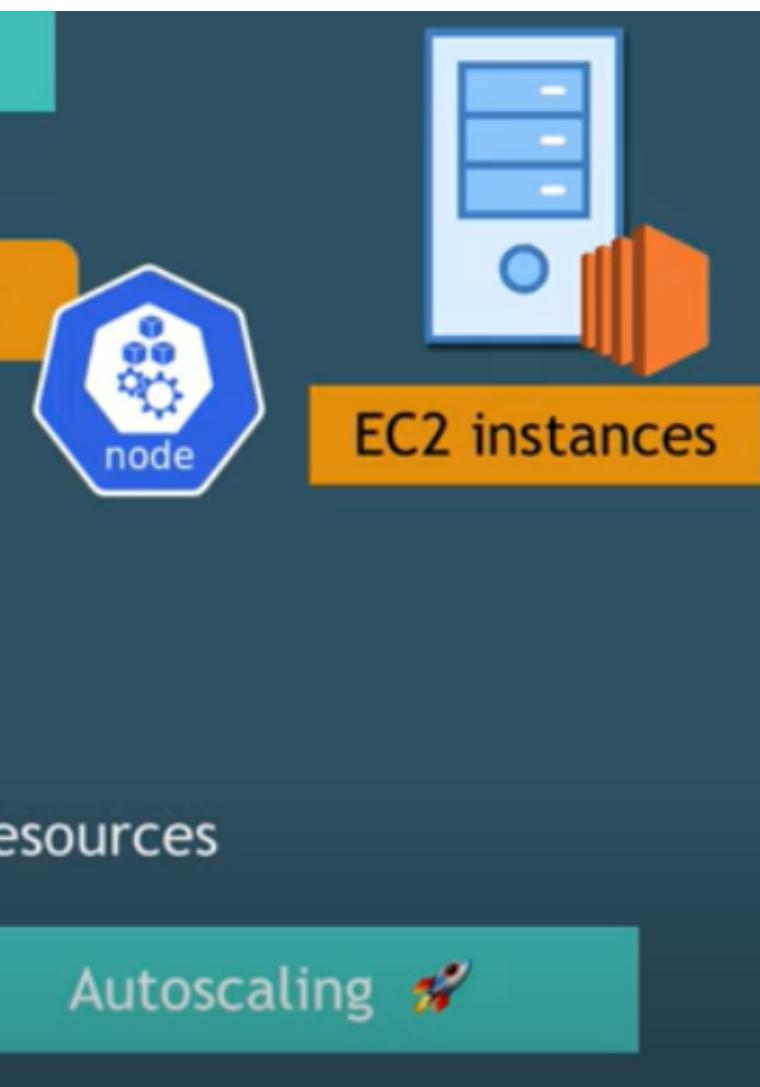
How to use EKS?

3 | Create Worker Nodes and connect to cluster

- ▶ Create as a Node Group (group of Nodes)
- ▶ Choose cluster it will attach to
- ▶ Define Security Group, select instance type, re
- ▶ Define max and min number of Nodes

Use Kubectl to connect your local to AWS or K8 Cluster.

...create with IAM role



3 | Create Worker Nodes and connect to cluster



kubectl

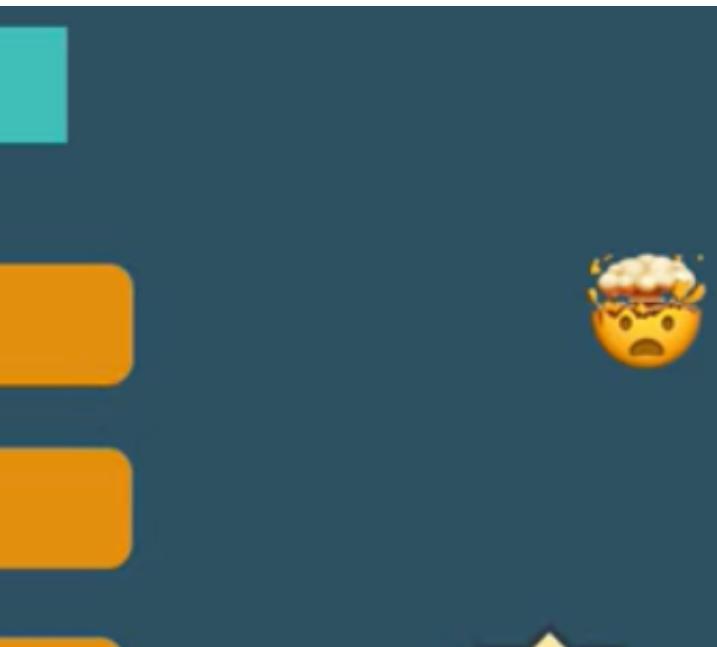


2)

Above steps Summarization:

1 | Setup or preparation steps

2 | Create Cluster Control Plane



3 | Create Worker Nodes and connect to cluster

- ▶ choose cluster name, k8s version
- ▶ choose region and VPC for your cluster
- ▶ set security for your cluster
- ▶ Create as a Node
- ▶ Choose cluster it will be part of
- ▶ Define Security Groups
- ▶ Define max and min worker nodes

3)

All above mentioned steps seems so much hectic as we need to handle both master and workers and so many other configuration things. But below tool is our savior.

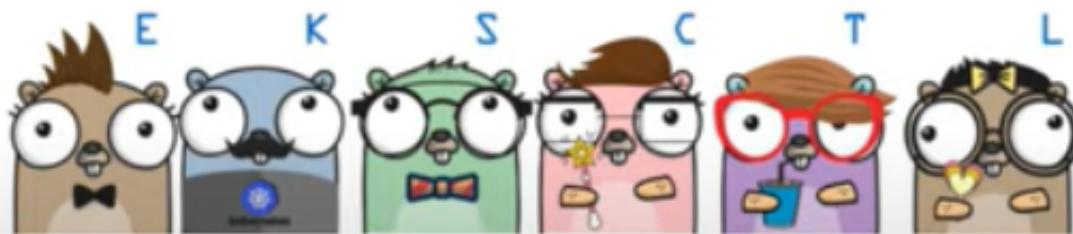
How to use EKS?

eksctl - The official CLI for Amazon EKS

[ci/validate](#) [passing](#) [coverage](#) [38%](#) [go report](#) [A+](#)

eksctl is a simple CLI tool for creating clusters on EKS - Amazon's new managed Kubernetes service for EC2. It is written in Go, and uses CloudFormation.

You can create a cluster in minutes with just one command - `eksctl create cluster`!



eksctl

How to use EKS?

1 | Setup or preparation steps



Group (group of Nodes)

will attach to

group, select instance type, resources

min number of Nodes

understanding the concepts



2 | Create Cluster Control Plane

3 | Create Worker Nodes and connect to cluster

- ▶ choose cluster name, k8s version
- ▶ Create as a Node Group (group of Nodes)
- ▶ choose region and VPC for your cluster
- ▶ Choose cluster it will attach to
- ▶ set security for your cluster
- ▶ Define Security Group, select instance type, res...
- ▶ Define max and min number of Nodes

3.1)

```
[~]$ brew tap weaveworks/tap
Updating Homebrew...
```

```
[~]$ brew install weaveworks/tap/eksctl
```

```
[~]$ eksctl version
0.26.0
```

! eksctl needs to authenticate with AWS

```
[~]$ eksctl create cluster \
> --name test-cluster \
> --version 1.17 \
> --region eu-central-1 \
> --nodegroup-name linux-nodes \
> --node-type t2.micro \
> --nodes 2
[!] eksctl version 0.26.0
```

```
eksctl create cluster
```



default values

override via parameters

```
[i] Deploying stack "eksctl-test-cluster-nodegroup" to "eu-central-1"
[i] waiting for the control plane availability...
[✓] saved kubeconfig as "/Users/nanajanashia/.kube/config"
[i] no tasks
[✓] all EKS cluster resources for "test-cluster" have been created

~]$ kubectl get nodes
NAME                               STATUS   ROLES
ip-192-168-12-111.eu-central-1.compute.internal   Ready   <none>
ip-192-168-33-61.eu-central-1.compute.internal   Ready   <none>

~]$ kubectl get pod
0 resources found in default namespace.

~]$ kubectl get ns
NAME          STATUS   AGE
default       Active   10m
kube-node-lease   Active   10m
kube-public    Active   10m
kube-system    Active   10m

~]$ eksctl delete cluster --name test-cluster
[i] eksctl version 0.26.0
[i] using region eu-central-1
[i] deleting EKS cluster "test-cluster"
[i] deleted 0 Fargate profile(s)
[✓] kubeconfig has been updated
[i] cleaning up AWS load balancers created by Kubernetes objects
```

1) =====ECS ===== Equivalent to K8s=====

What is Elastic Container Service

- ▶ Container Orchestration Service
- ▶ Manages the whole container lifecycle

start

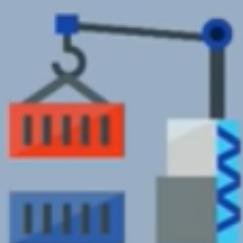
re-schedule

load

```
S     AGE      VERSION
e> 4m1s  v1.17.9-eks-4c6976
e> 4m1s  v1.17.9-eks-4c6976
```

ts of Kind Service or Ingress

(ECS)?



balance

How does ECS work?

Run containerized application cluster on AWS



Control Plane

Scheduling and
Orchestration



ECS Cluster



Man

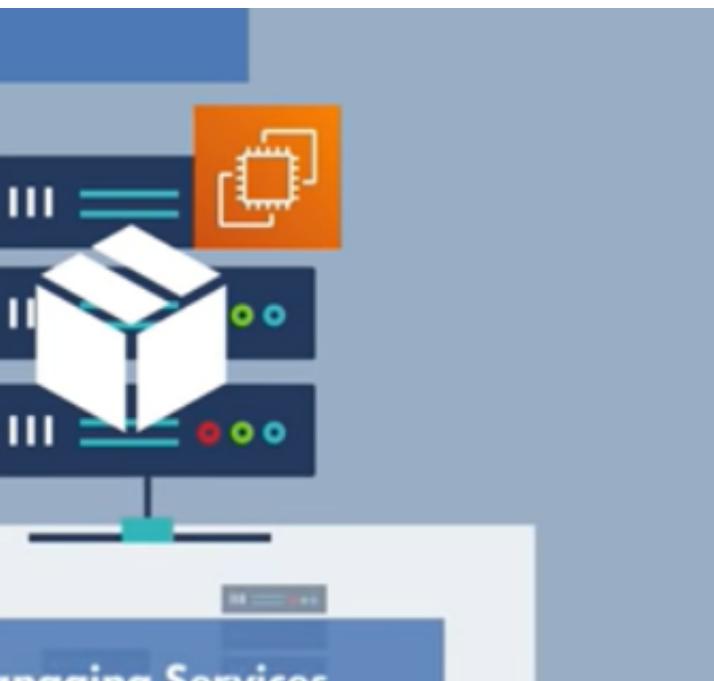
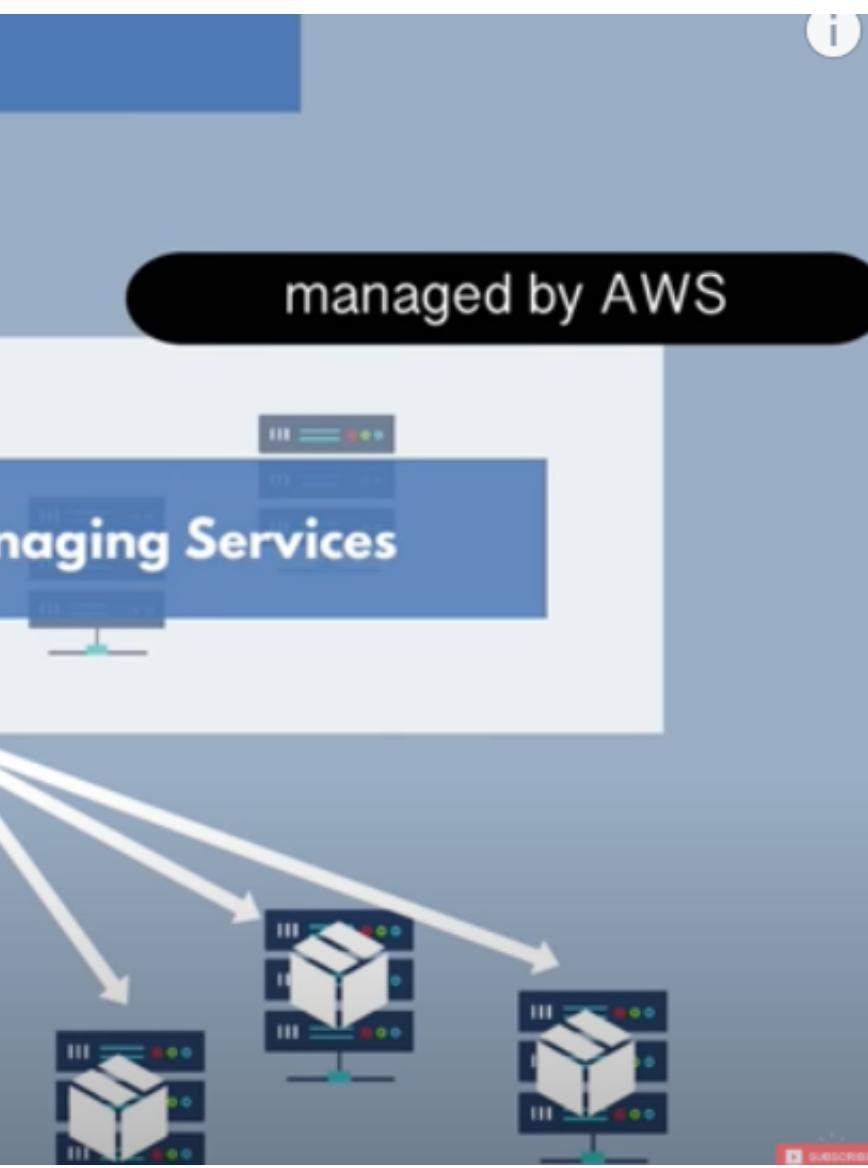


How does ECS work?

Where are these containers hosted?

Which virtual machines?





EC2 Instances

ECS Cluster



Which Services are running
on your EC2 Instance



- ▶ Container Runtime
- ▶ ECS Agent - for Control Plane Communication

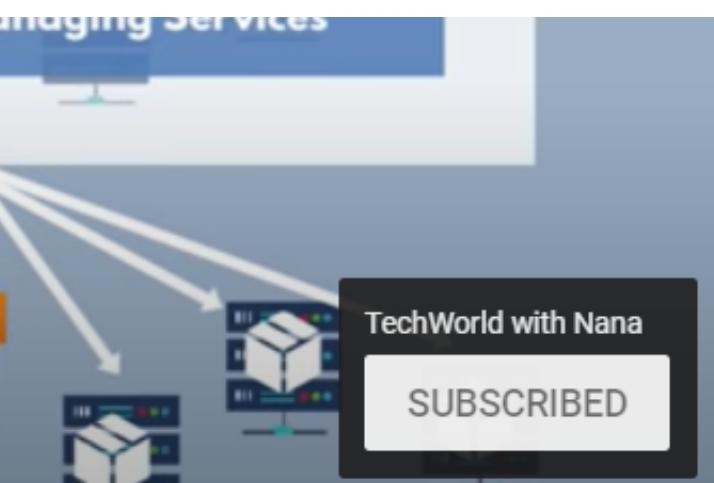
▶ ECS hosted on **EC2 Instances**



manages the **containers**



You still need to manage the Virtual



Machine



- ▶ Create EC2 instances
- ▶ Join to ECS cluster
- ▶ Check, whether enough resources
- ▶ Manage
- ▶ Docker



1) =====AWS with Fargate=====

ECS with AWS Fargate



Container Orchestration

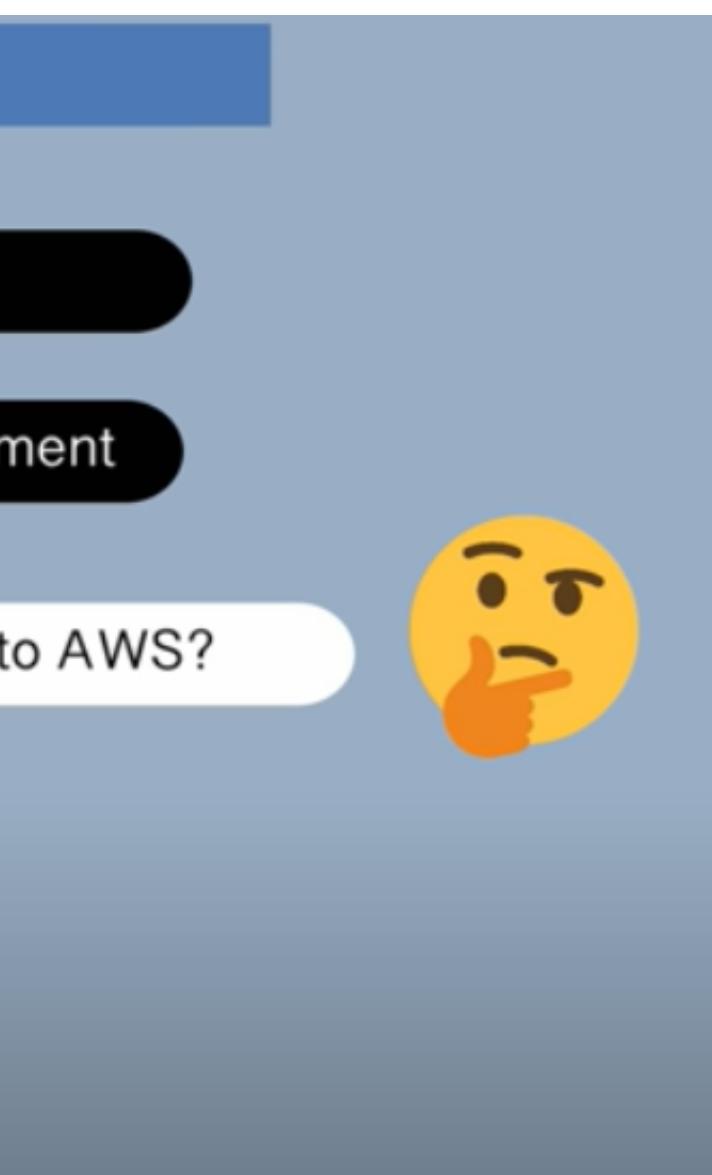
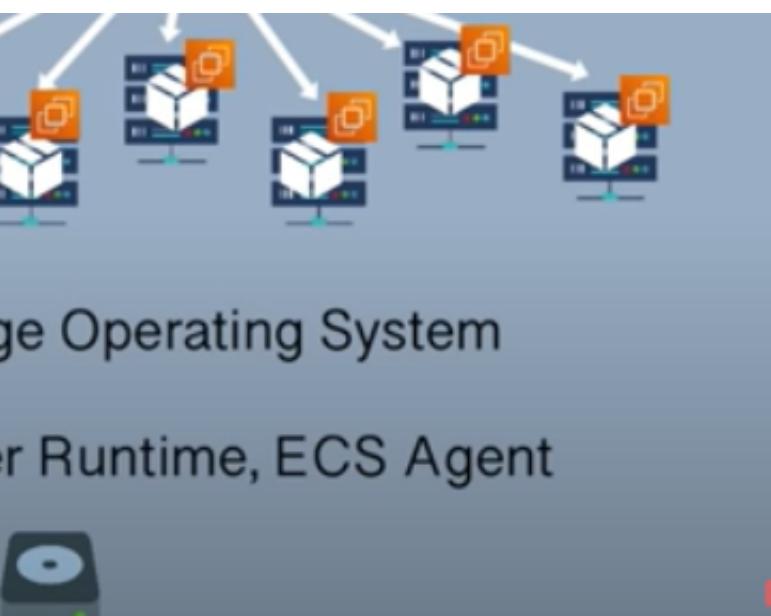


Hosting Infrastructure Manager

Delegate Infrastructure Management also to



AWS Fargate



ECS with AWS Fargate

- ▶ Alternative to EC2 Instances

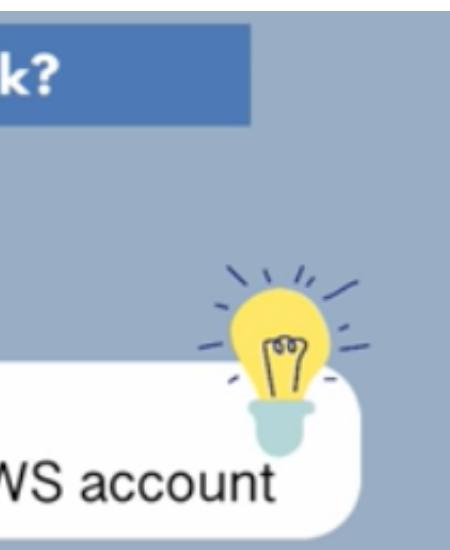
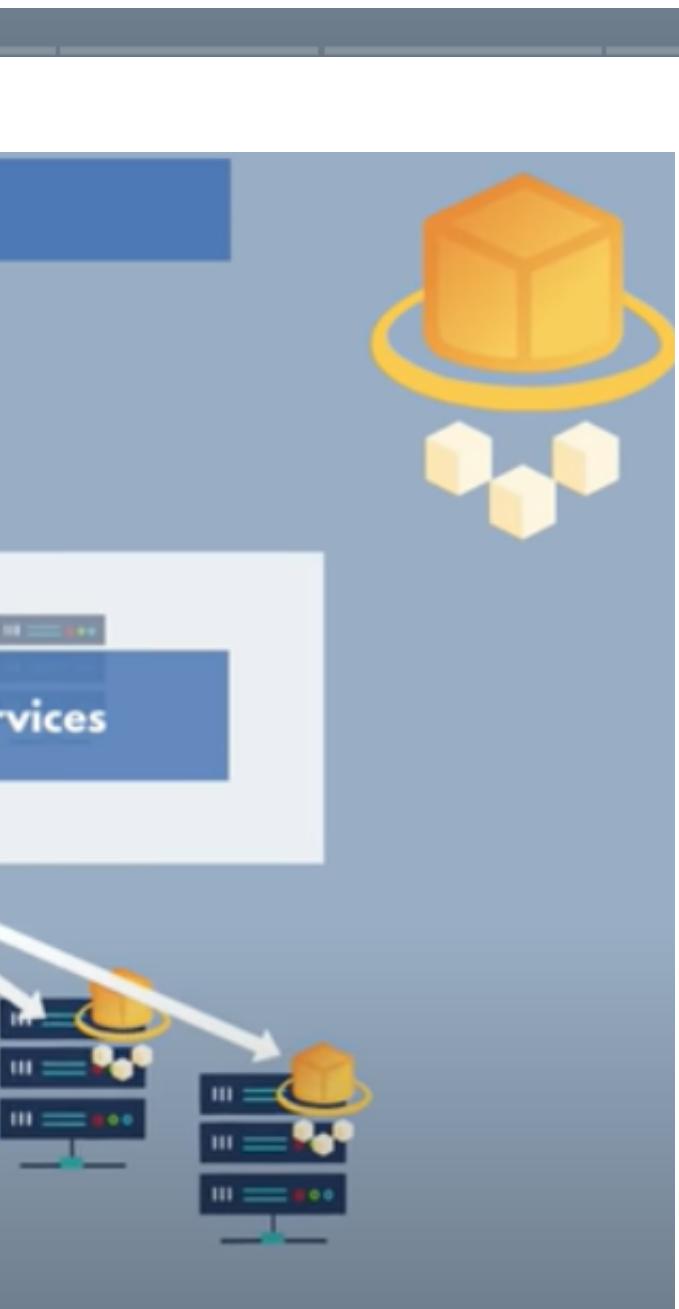


How does AWS Fargate work?

- ▶ Serverless way to launch containers

"Serverless" meaning:

No Server in your AWS account, Server in AW



How does AWS Fargate work?

- Serverless way to launch containers



Launch container with Fargate



ECS Cluster

No EC2 Instances provisioned

How does AWS Fargate work?

- Serverless way to launch containers



Launch container with Fargate



ECS Cluster

ork?

Managing Services

ance
l yet



ork?

Managing Services





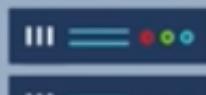
► provisions a server **on demand**

AWS Fargate advantages

► ECS with AWS Fargate

- No need to provision and manage servers
- on demand
- only the infrastructure resources needed to run your containers
- pay only for what you use** 
- easily scales up & down without fixed resources defined beforehand

AWS Fargate advantages



Infrastructure

The image shows a video player interface with a blue header bar. In the top left corner of the header, there is a small icon of a white server or cube with network cables. The main content area has a light blue background. At the top, the text "EC2 Instance Pricing" is displayed in bold black font. Below it, a white rounded rectangle contains the text "Pay for whole server". Further down, the text "AWS Fargate Pricing" is shown in bold black font. Below this, two white rounded rectangles contain the questions "How long?" and "How much capacity?". In the bottom right corner of the video player, there is a small red button with the word "SUBSCRIBE" and a play icon.

EC2 Instance Pricing

Pay for **whole server**

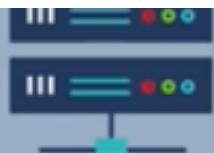
AWS Fargate Pricing

How long?

How much **capacity**?

SUBSCRIBE





managed by AWS

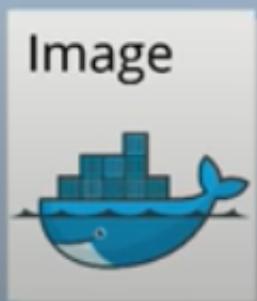
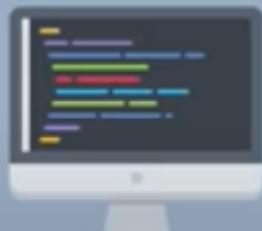


Containers
managed by AWS

AWS Fargate advantages



You only need to manage your application





ges

anage
on



aws



100%



AWS Fargate is great, but..

More control and flexibility
with EC2 Instances

Integration with other AWS Services



- ▶ AWS ecosystem available

CloudWatch for Monitoring

Elastic Load Balancer for Loadbalancing

flexibility
ances

es

IAM for
Users and Permissions

VPC for
Networking

...

- 1) ===== Alternative to ECS === EKS=====

Elastic Kubernetes Service



Amazon EKS

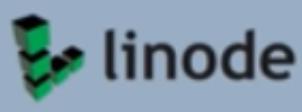
- ▶ Managing Kubernetes cluster on AWS
- ▶ alternative to ECS 

Elastic Kubernetes Service



Both managing the Control Plane

- 👍 you already use K8s (**same K8s API**)
- 👍 Kubernetes is open-source
- 👍 easier to migrate to another platform

 Google Cloud  Microsoft Azure  linode 

- ▶ migration can be difficult,
when using other AWS Services

Service (EKS)

infrastructure

Service (ECS)

Control Plane, but:



- ▶ specific to AWS
- ▶ migration difficult



less complex applications



ECS control plane is free



large community (Helm charts etc.)

- 2) ===== Control Plane ==> here we are configuring all MASTER NODES.

How does EKS work

Scheduling and
Orchestration



Control Plane



EKS Cluster



- ▶ EKS deploys and manages Kubernetes Master
- ▶ K8s Master Services already installed on them
- ▶ High Availability - Master Nodes replicated acr

How does EKS work

Scheduling and
Orchestration



Control Plane



EKS Cluster



k?



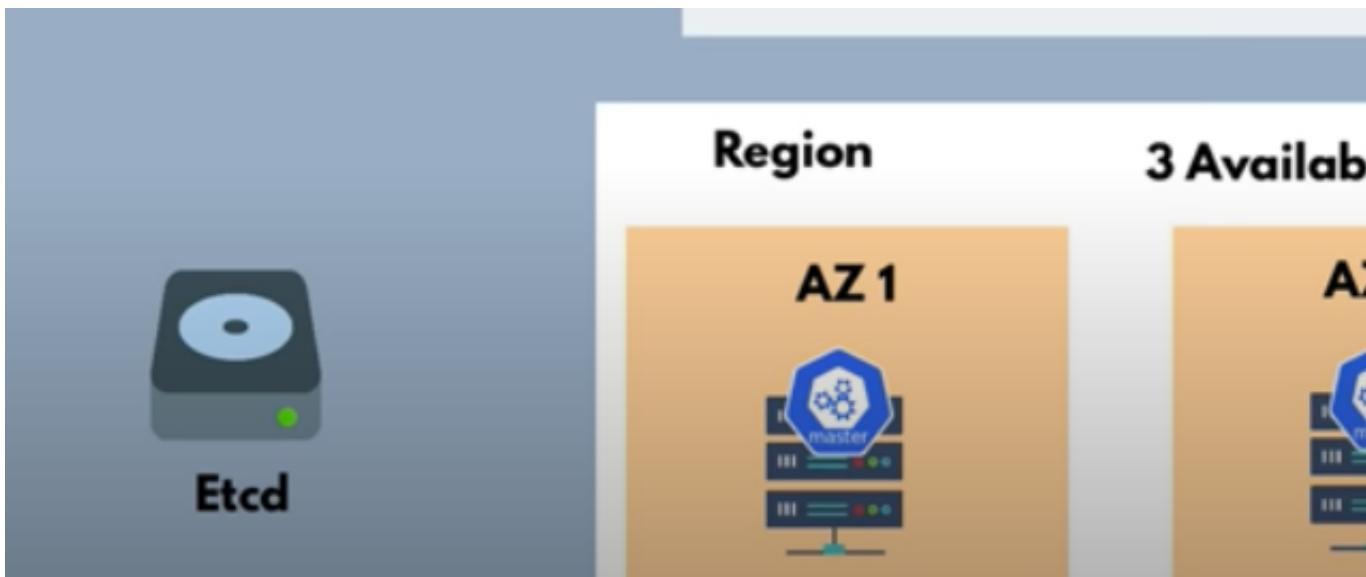
Nodes



ross Availability Zones

k?



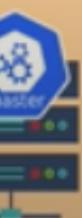


- 3) Worker Nodes Configuration:: Compute fleet is actually a EC2 Instance.



Availability Zones

AZ 2



AZ 3



work?



Availability Zones

AZ 2



AZ 3



SUBSCRIBE

work?

Scheduling and
Orchestration



Control Plane

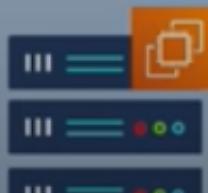


EKS Cluster



Worker Nodes

Compute Fleet



Scheduling and
Orchestration



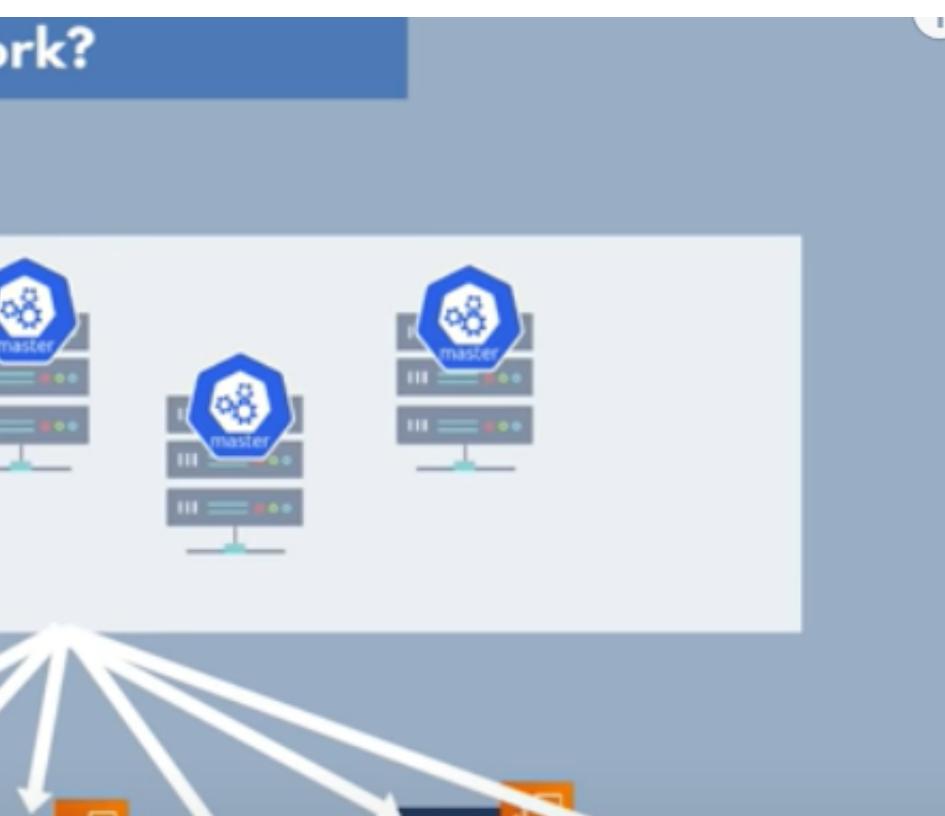
Control Plane

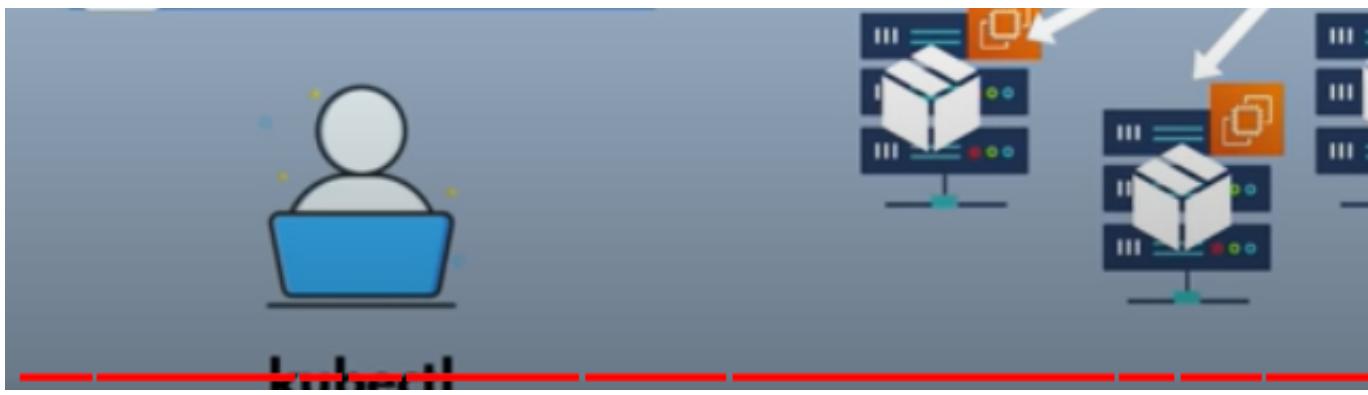


EKS Cluster



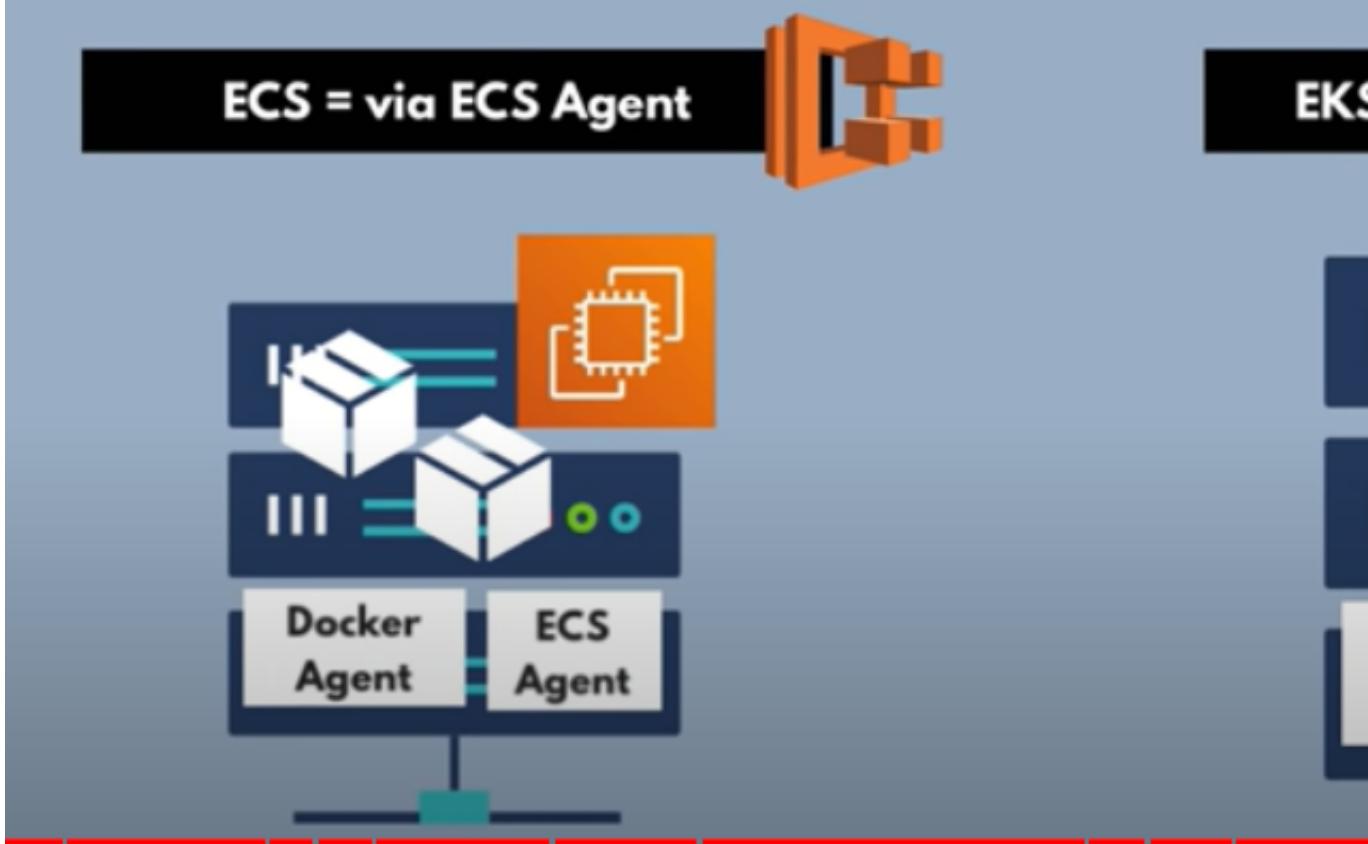
Worker Nodes





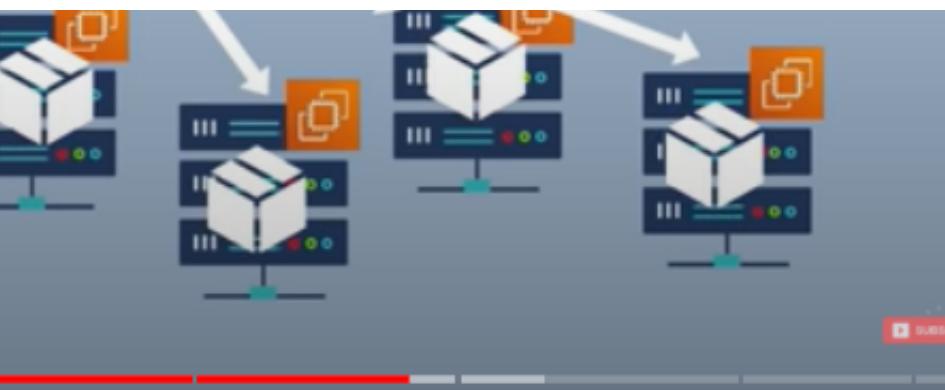
How does EKS work?

Communication between Control Plane (Master)

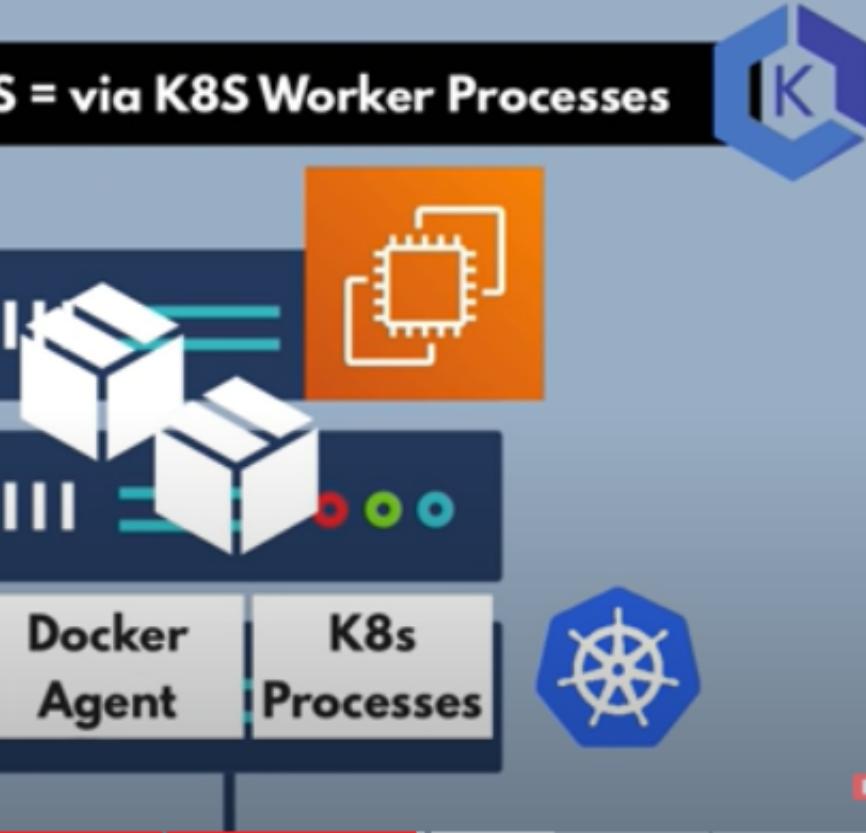


Worker Nodes

EC2 vs EC2 Node



Nodes) and Compute Fleet



vs Fargate

EKS with EC2 Instances

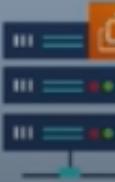
self-managed

- ▶ You need to manage the infrastructure for Worker Nodes

EKS with Nodegroup

semi-managed

- ▶ Creates, deletes EC2 Instances for you, but you need to configure it



20:55 / 25:09 • Worker Nodes options: EC2 vs Nodegroup vs Fargate



Worker Nodes

EC2 vs EC2 Nodegroups

EKS with EC2 Instances

self-managed

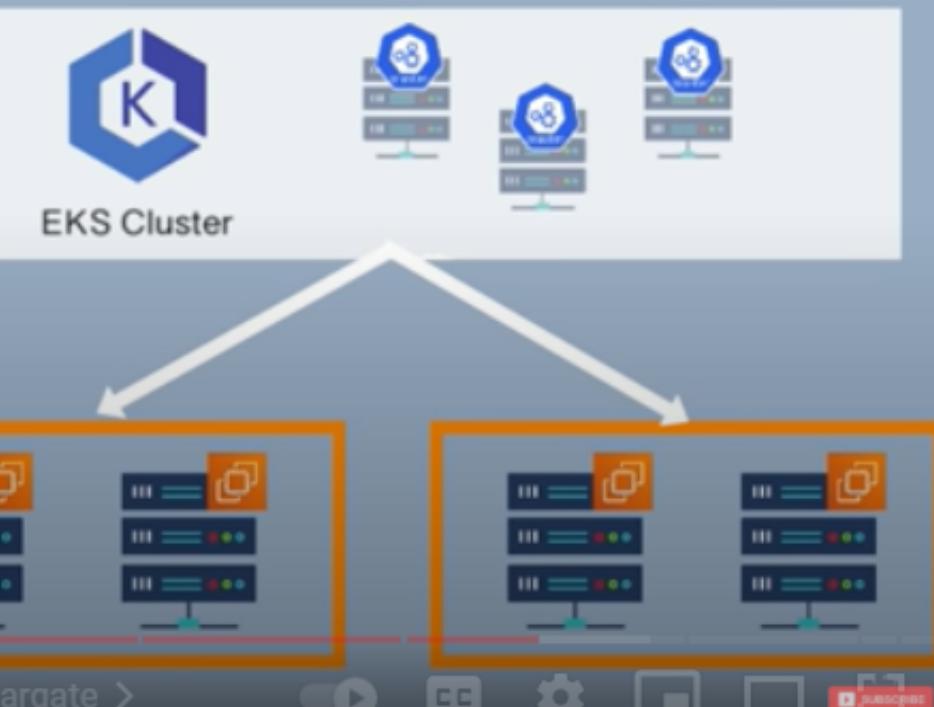
- ▶ You need to manage the infrastructure for Worker Nodes

EKS with Nodegroup

semi-managed

- ▶ Creates, deletes EC2 Instances for you, but you need to configure it

Worker Nodes

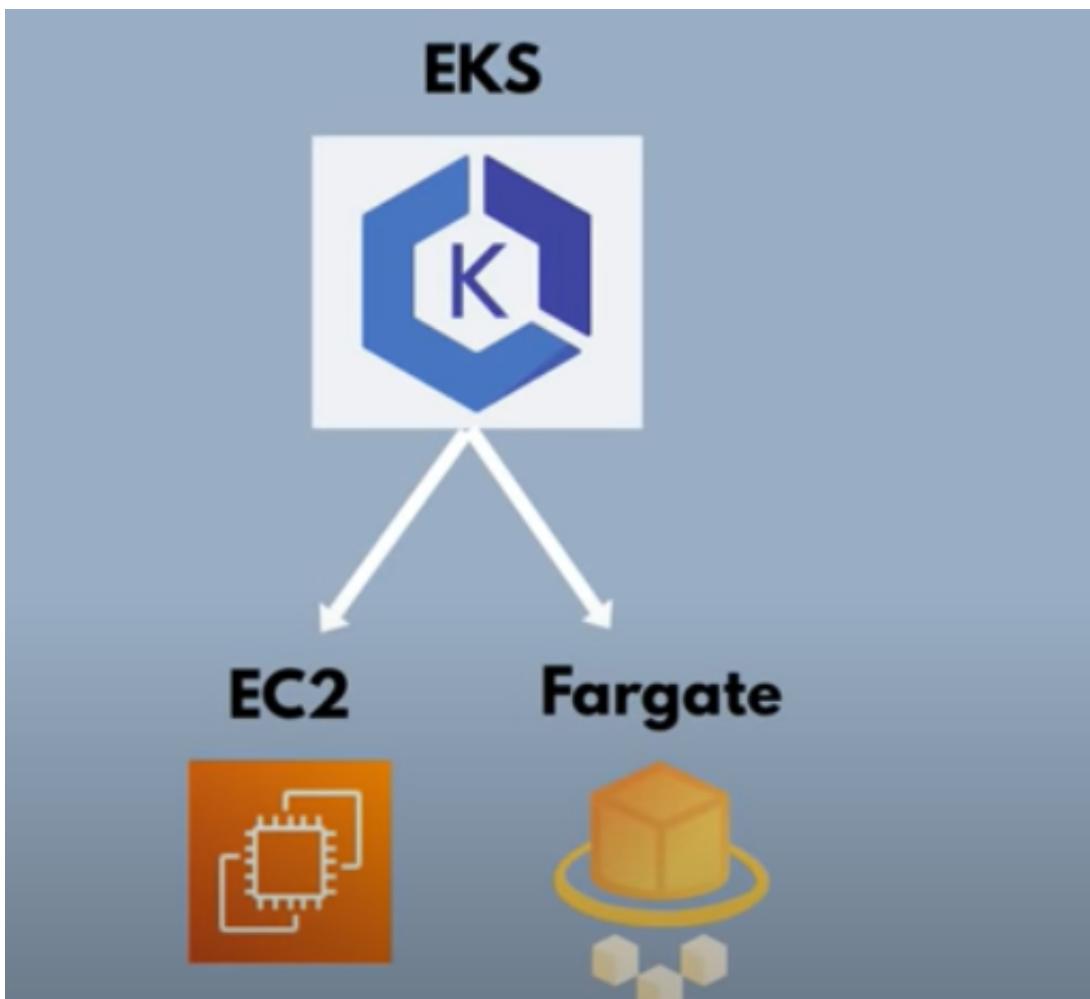


Taskgroup vs Fargate

Worker Nodes

EKS with Fargate

fully-managed



Combinations on AWS



ECS





9)

The slide is divided into two main sections. On the left, a large orange circle contains a white icon of a crane lifting a shipping container. A white callout bubble extends from the bottom of this circle towards the right, containing the text "Repository for Docker Images". On the right, a blue bar at the top contains the text "What is Amazon ECR". Below this, a blue button-like shape contains the "docker hub" logo, which includes a white ship icon and the text "docker hub".

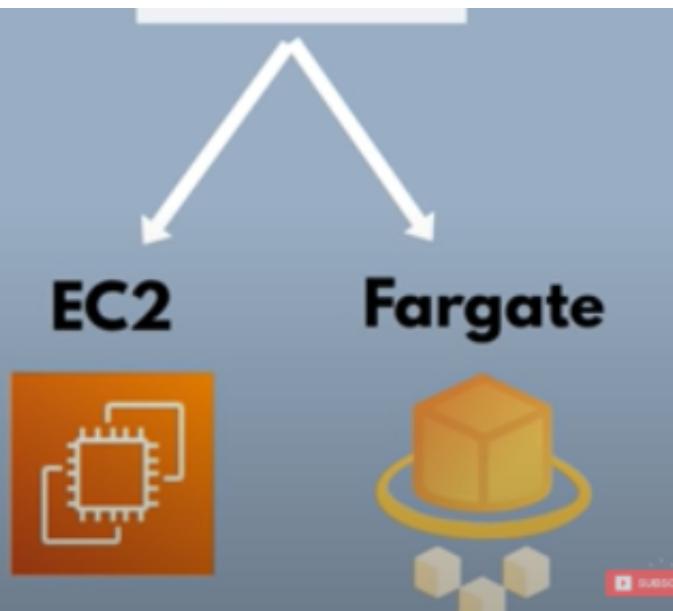
What is Amazon ECR

Repository for Docker Images

- ▶ store, manage and deploy Docker container images
- ▶ alternative to:

What is Amazon ECR





ECR?



 Nexus

CR?



integrates well with other AWS Services

notify when new image

pull images

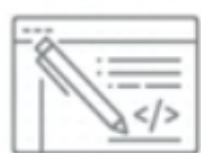


easy to connect and configure



your App Images
stored here

How it works



Write code

Write and package code
as a Docker image



Amazon ECR



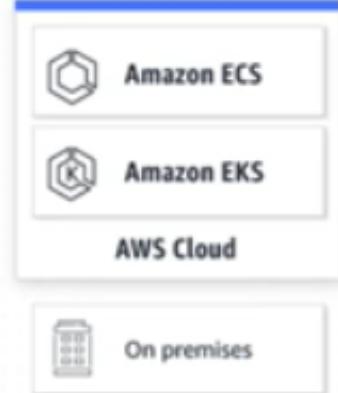
Compress, encrypt, and
control access to images

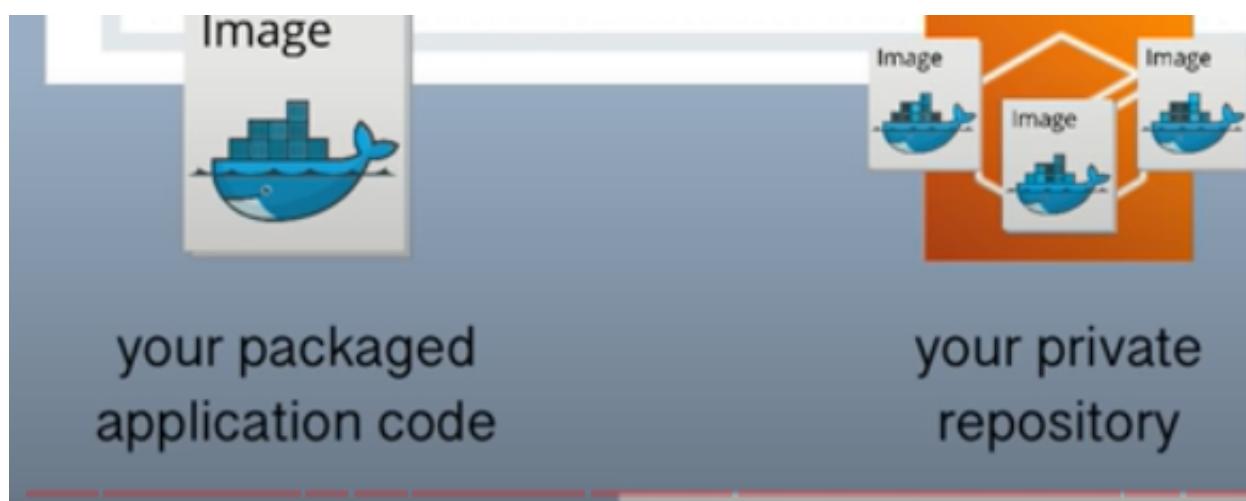


Version, tag, and manage
image lifecycles



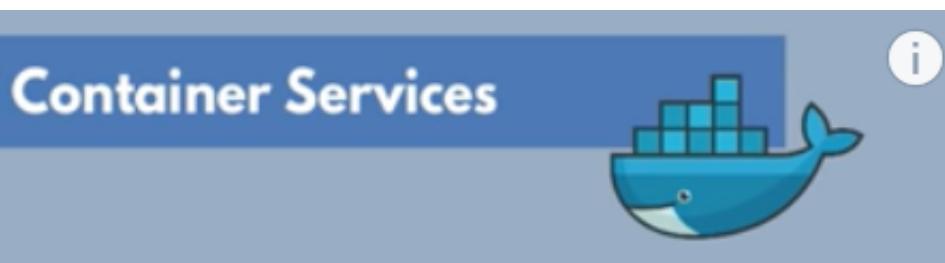
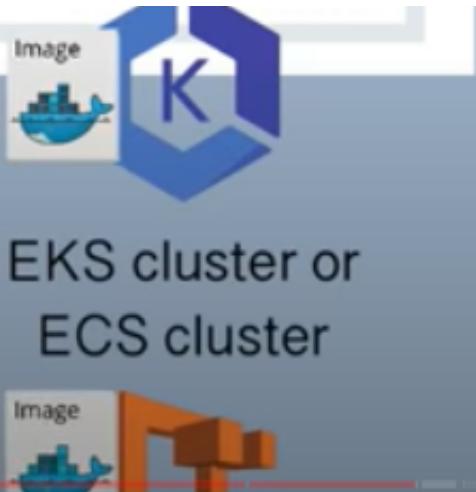
Run containers
Pull images and run containers anywhere





A screenshot of a Mac OS X desktop showing a web browser window for the AWS Management Console. The URL is `eu-west-3.console.aws.amazon.com/console/home?region=eu-west-3`. The browser title bar shows "Branches (4) [java-maven-app] Jenkinsfile · feature/pay". The AWS logo and "Services" dropdown are visible in the top-left corner of the browser window. The main content area displays the "All services" menu under the "Compute" category, listing EC2, Lightsail, Lambda, Batch, Elastic Beanstalk, Serverless Application Repository, AWS Outposts, and EC2 Image Builder. Below this, the "Containers" section lists ECR, Elastic Container Service, and Elastic Kubernetes Service. The "Storage" section is partially visible at the bottom.





Elastic Container Registry

Private Docker Repository



Elastic Container Service

Container Orchestration Service



Elastic Kubernetes Service

Container Orchestration Service

[SUBSCRIBE](#)

Managed Kubernetes Service

