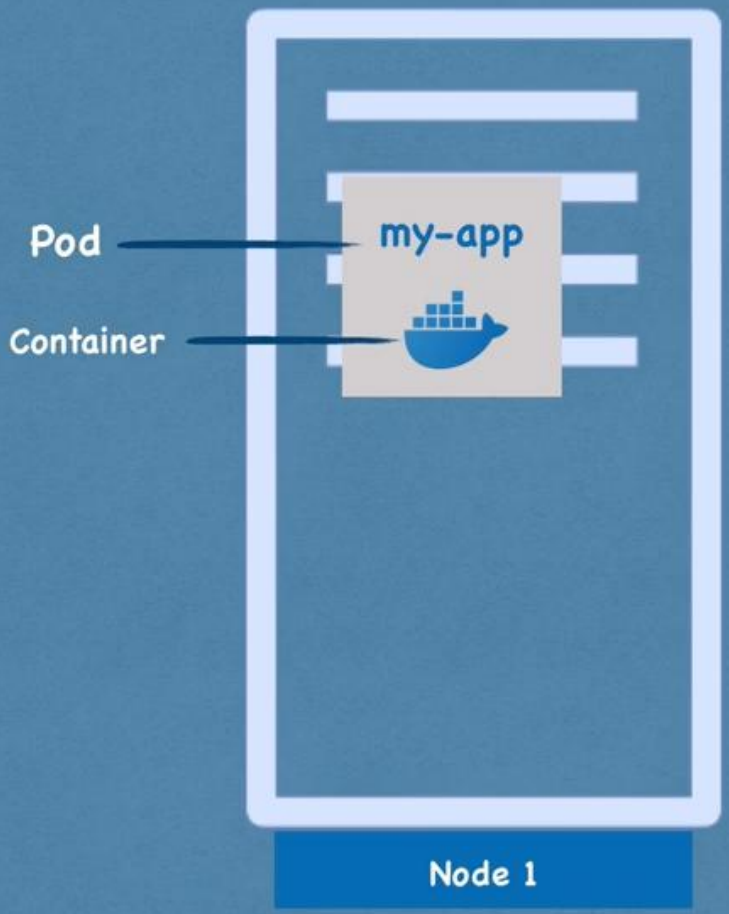
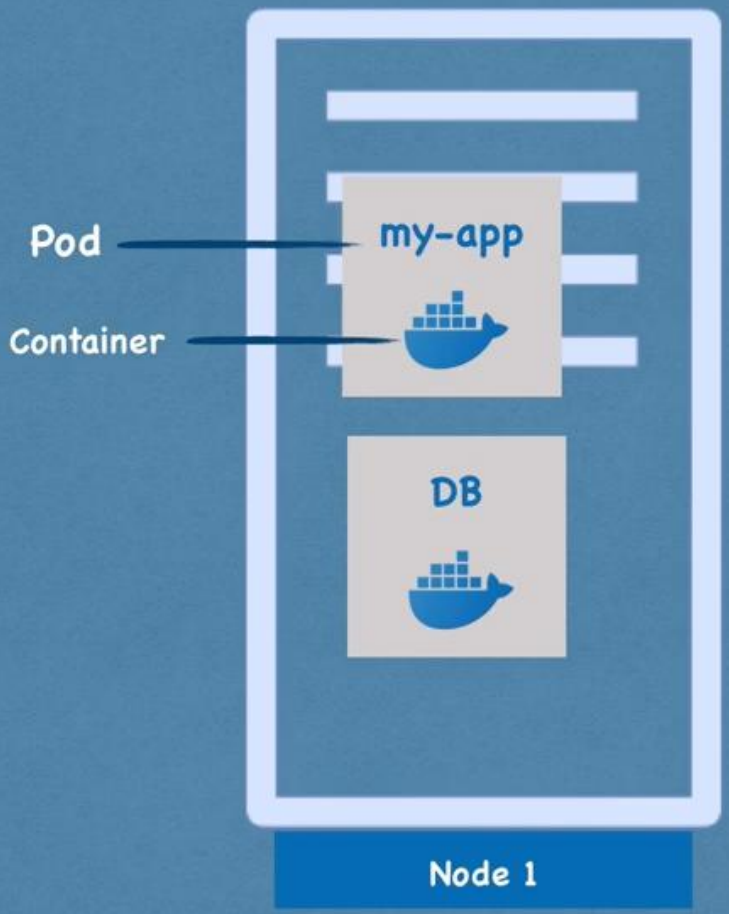
Node and Pod



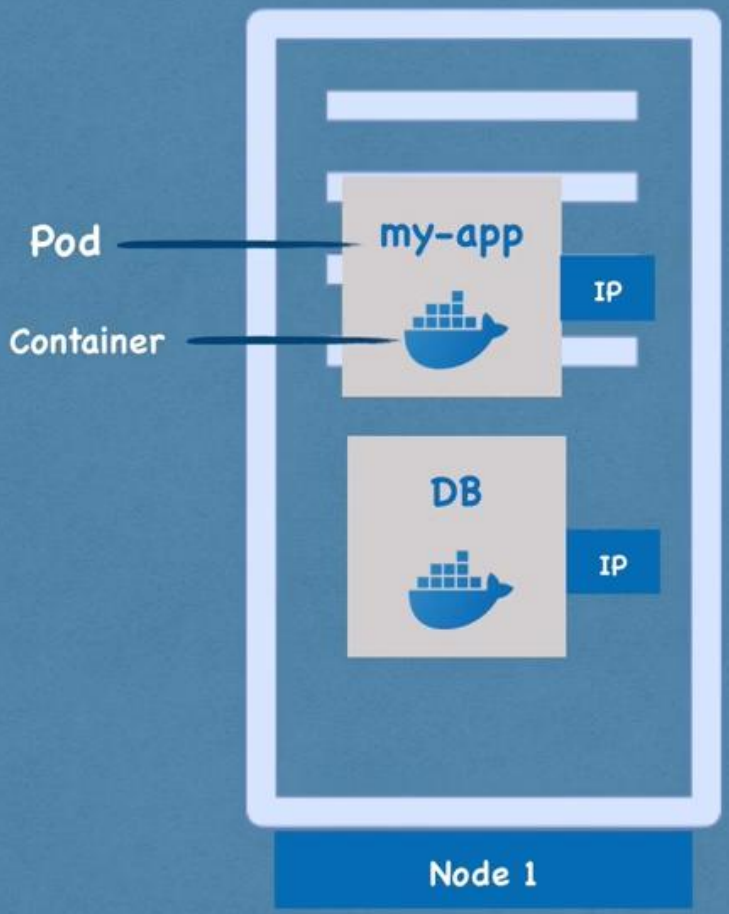
Pod:

- Smallest unit of K8s
- Abstraction over container



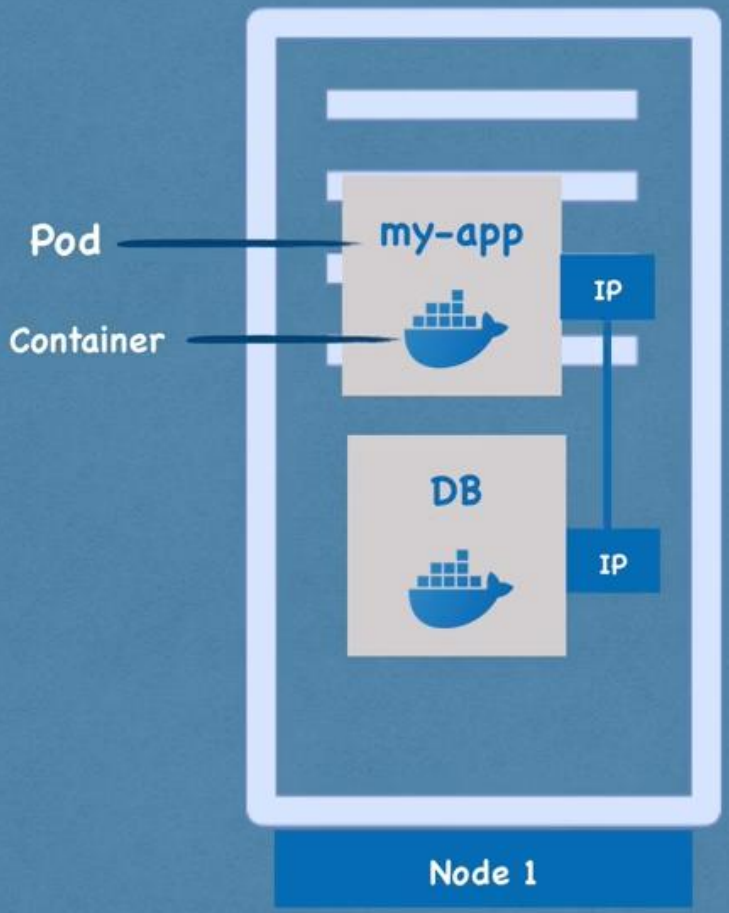
Pod:

- Smallest unit of K8s
- Abstraction over container
- Usually 1 application per Pod



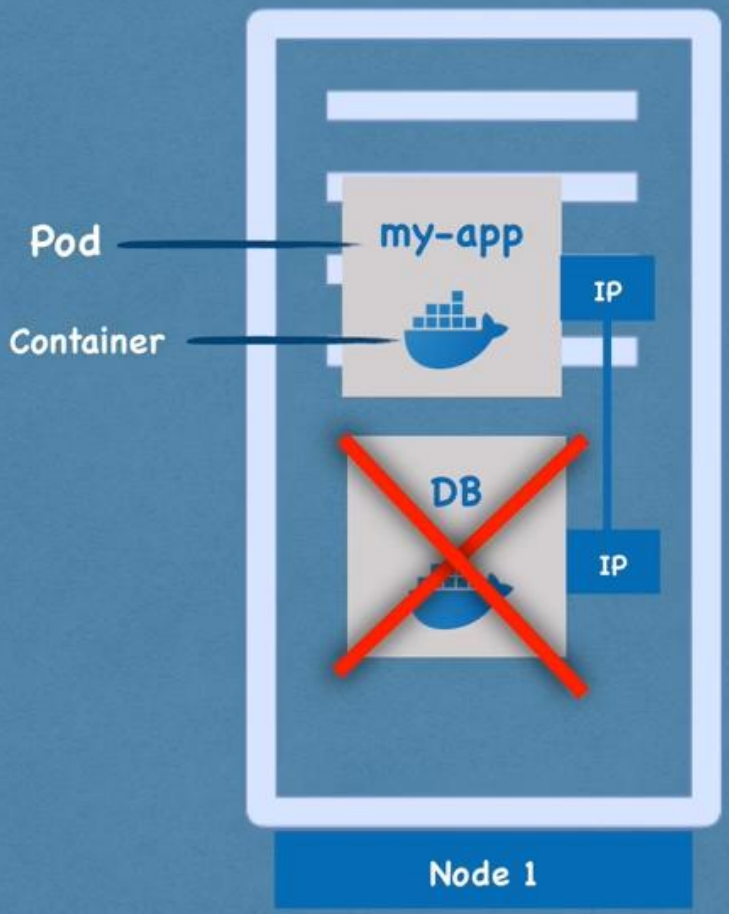
Pod:

- Smallest unit of K8s
- Abstraction over container
- Usually 1 application per Pod
- Each Pod gets its own IP address



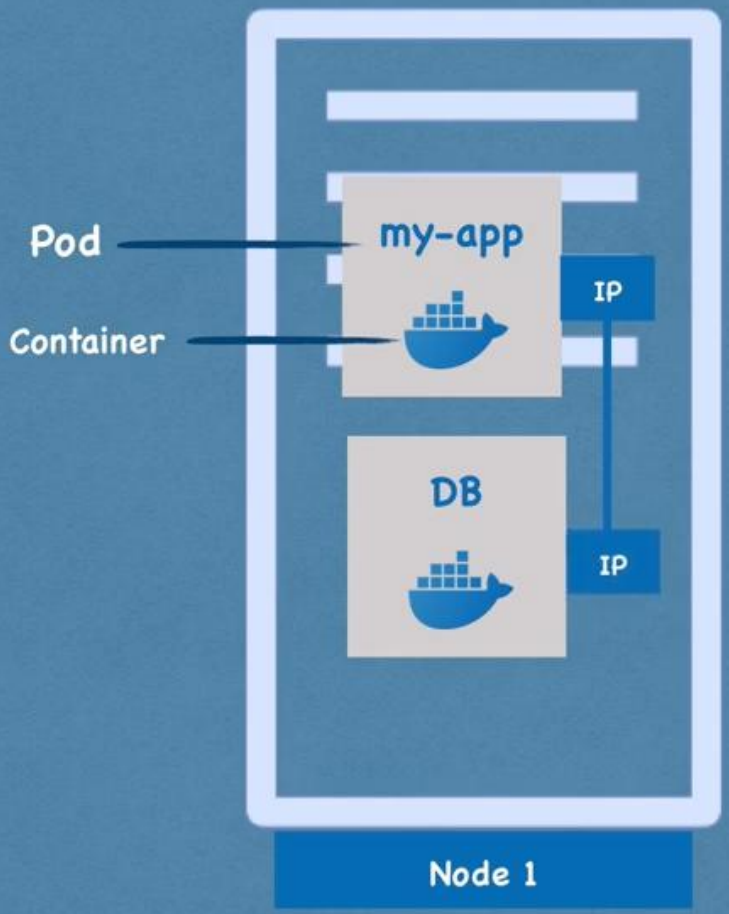
Pod:

- Smallest unit of K8s
- Abstraction over container
- Usually 1 application per Pod
- Each Pod gets its own IP address



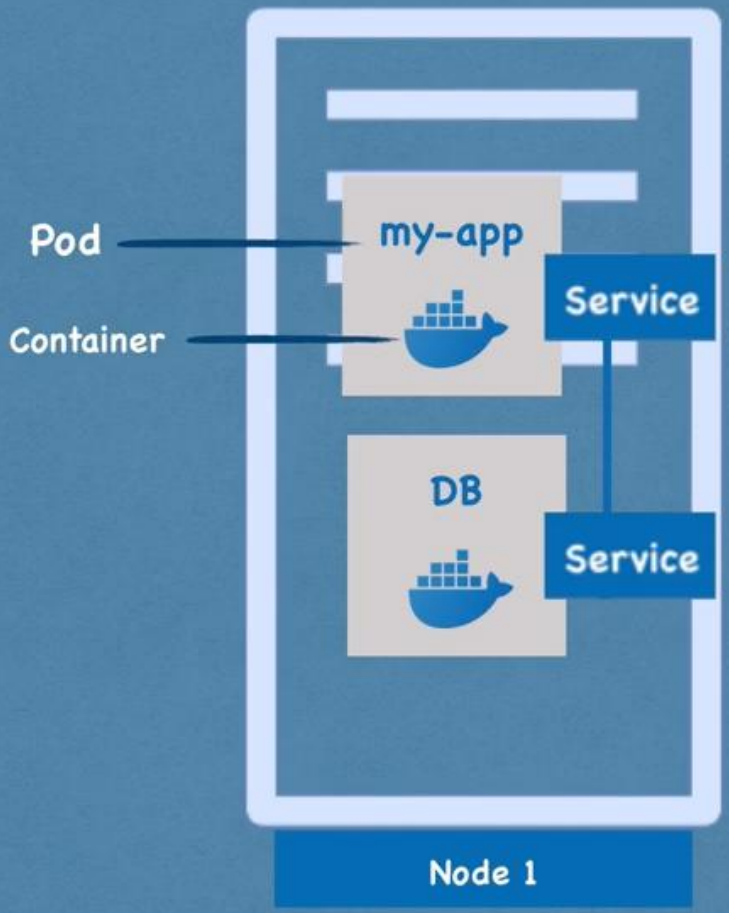
Pod:

- Smallest unit of K8s
- Abstraction over container
- Usually 1 application per Pod
- Each Pod gets its own IP address



Pod:

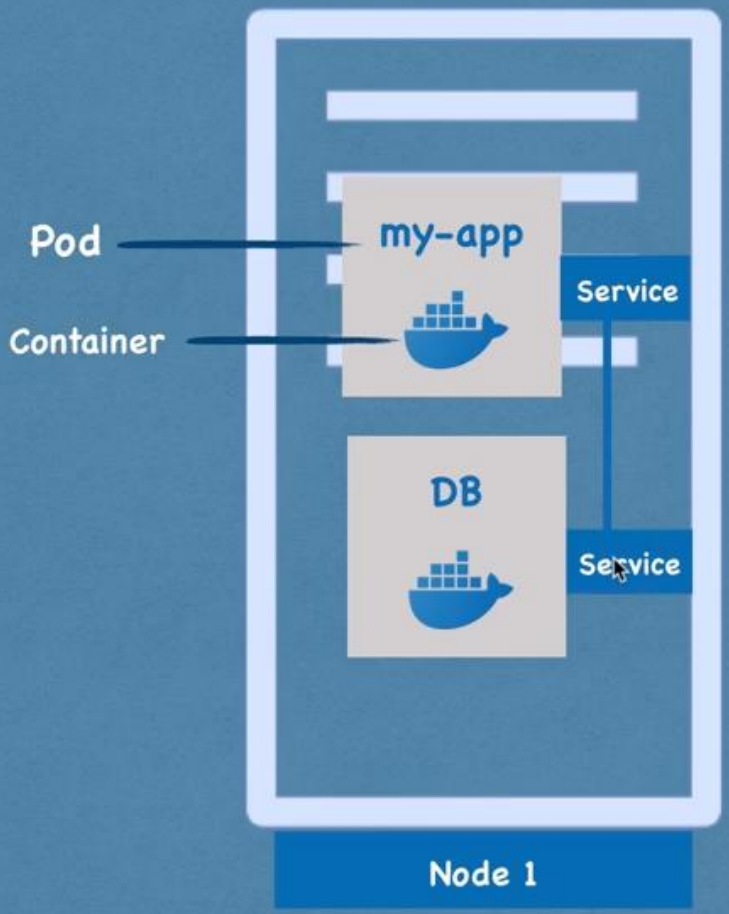
- Smallest unit of K8s
- Abstraction over container
- Usually 1 application per Pod
- Each Pod gets its own IP address
- New IP address on re-creation



Pod:

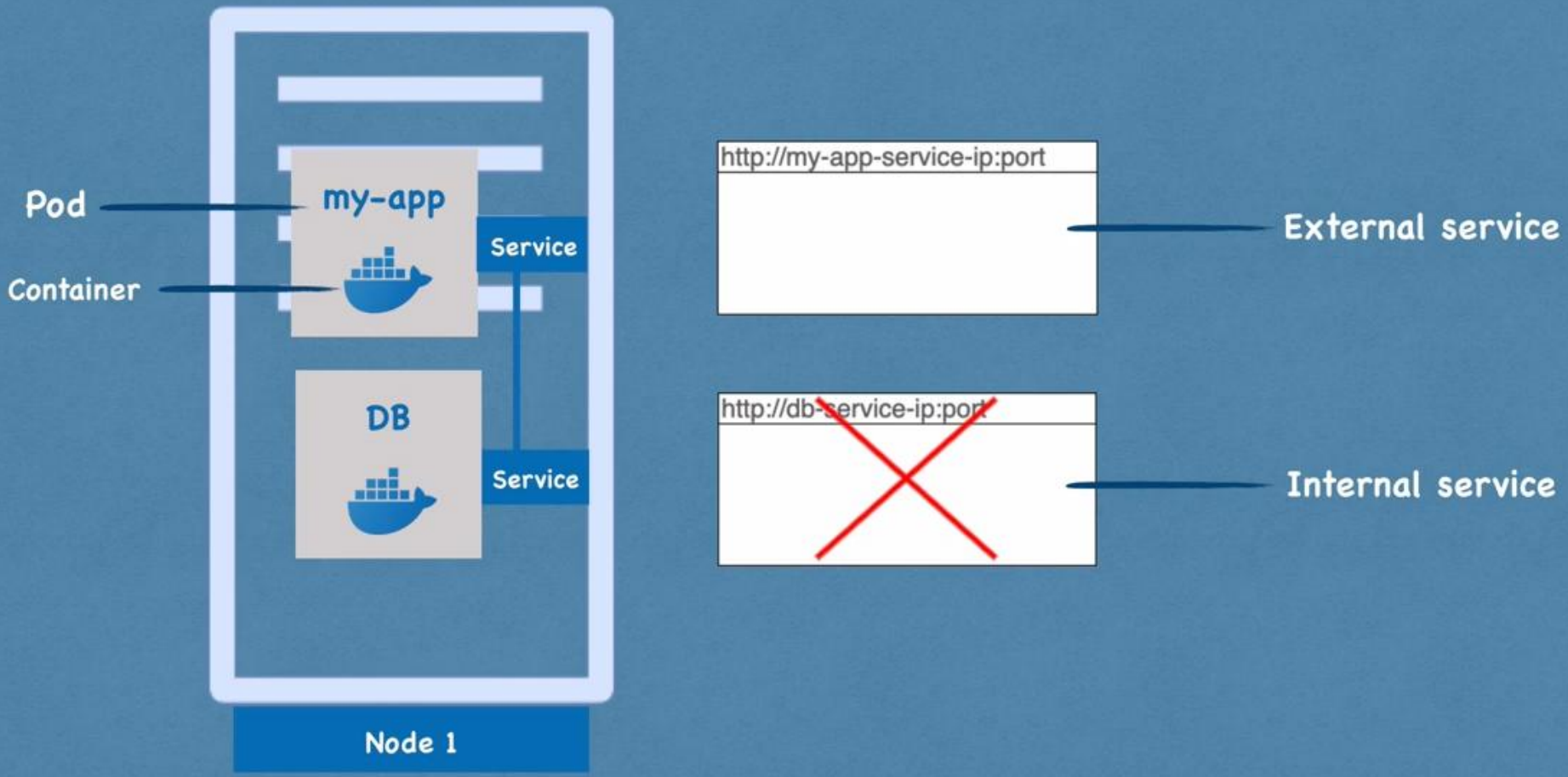
- Smallest unit of K8s
- Abstraction over container
- Usually 1 application per Pod
- Each Pod gets its own IP address
- New IP address on re-creation

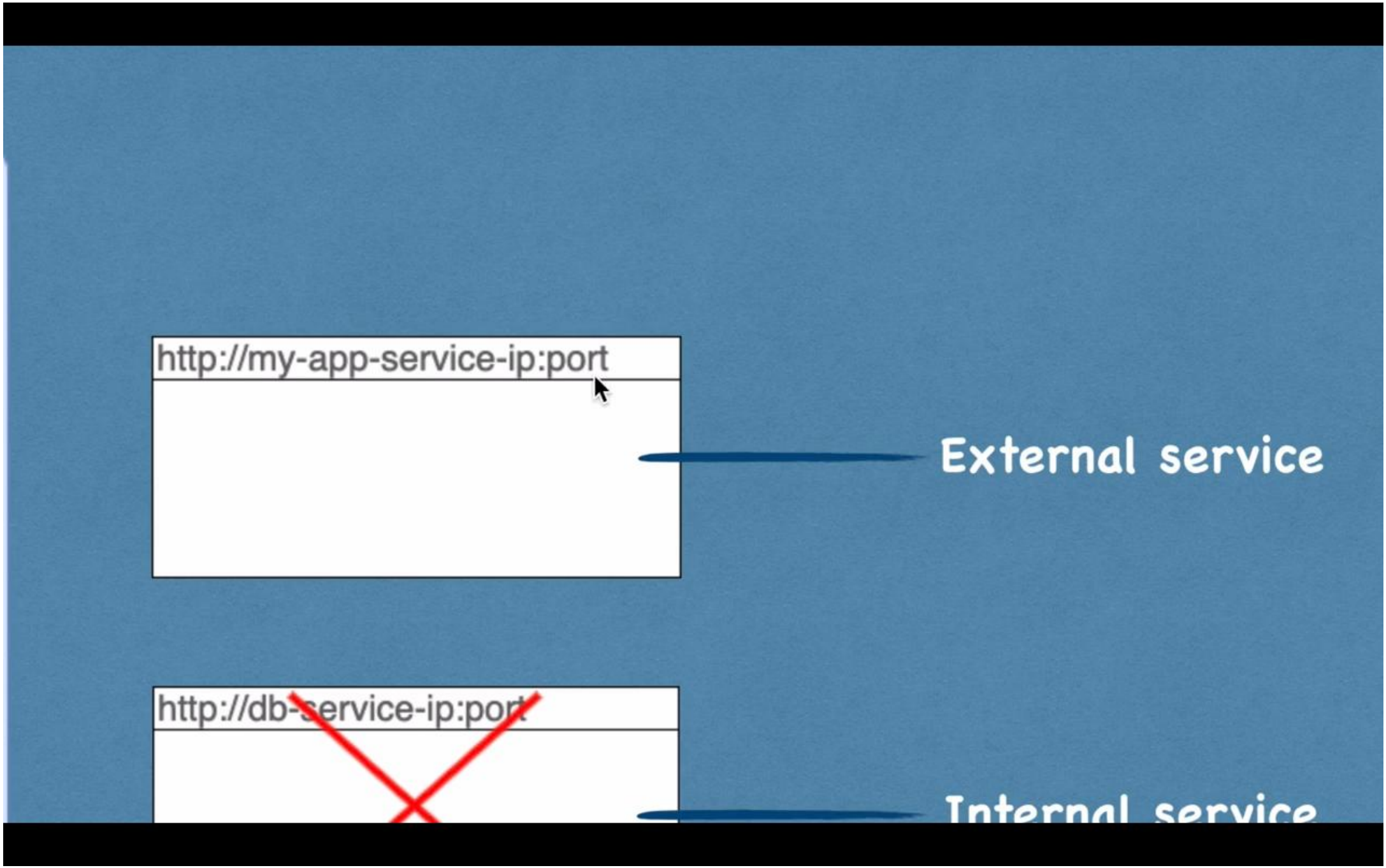
Service and Ingress

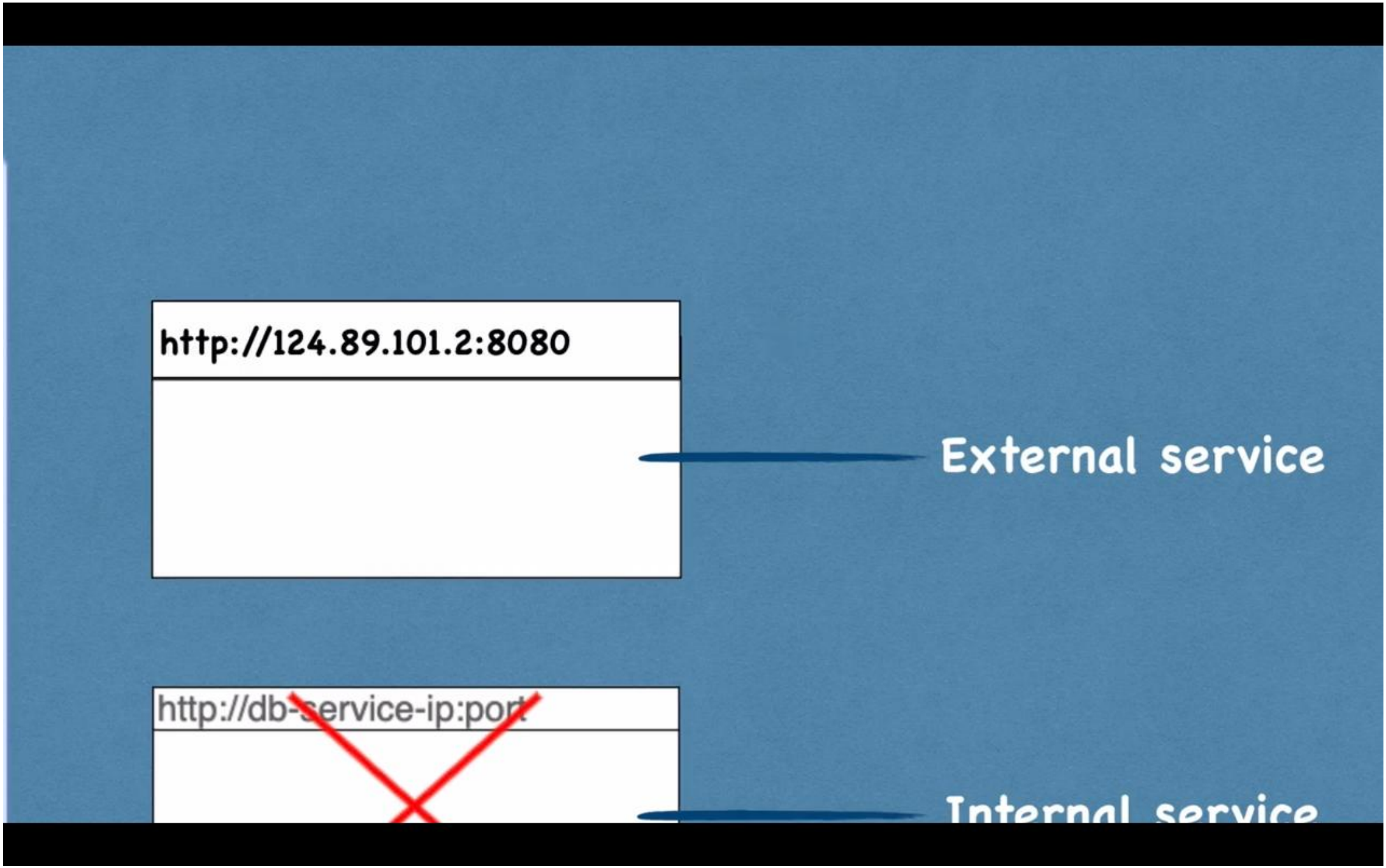


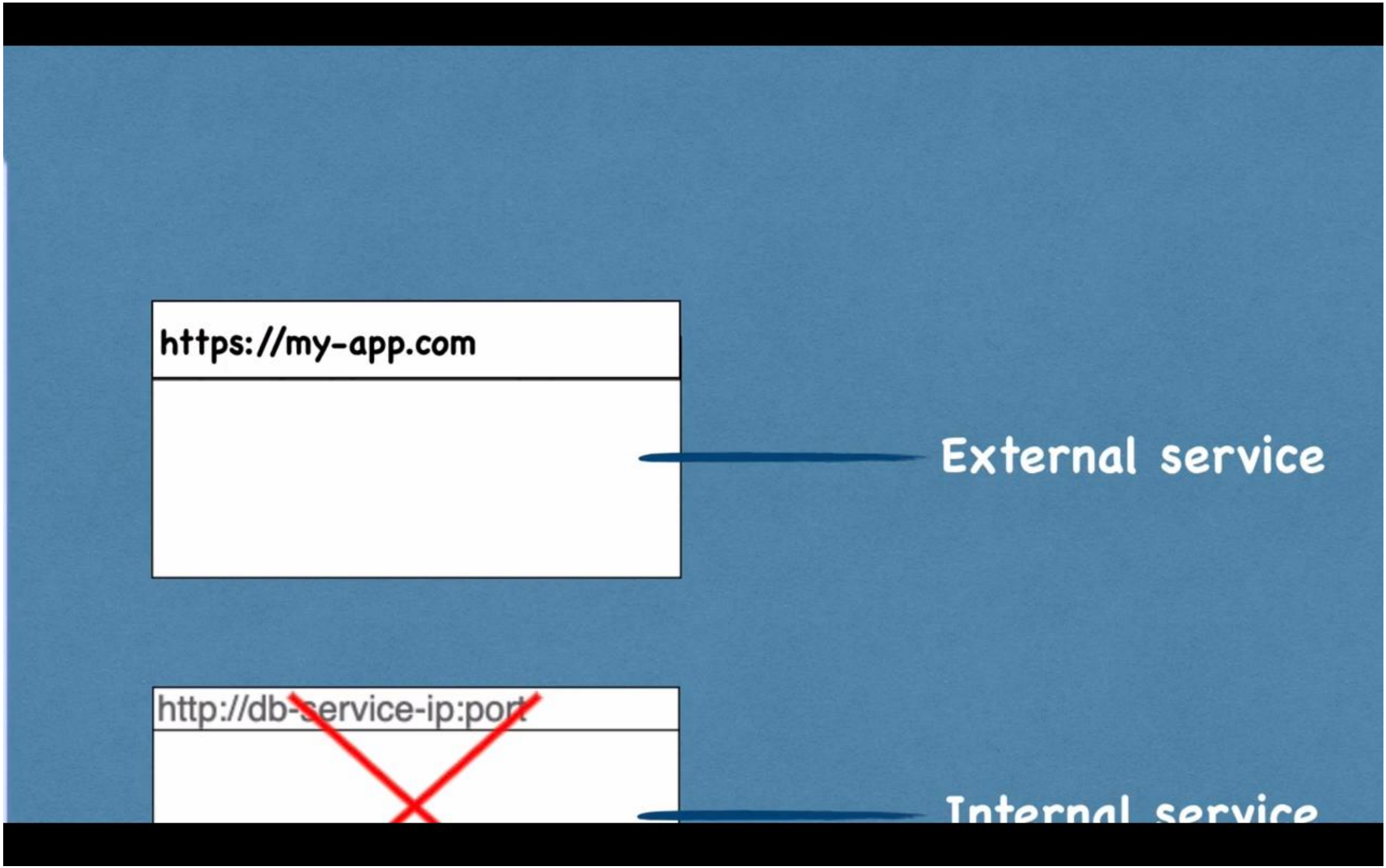
Service:

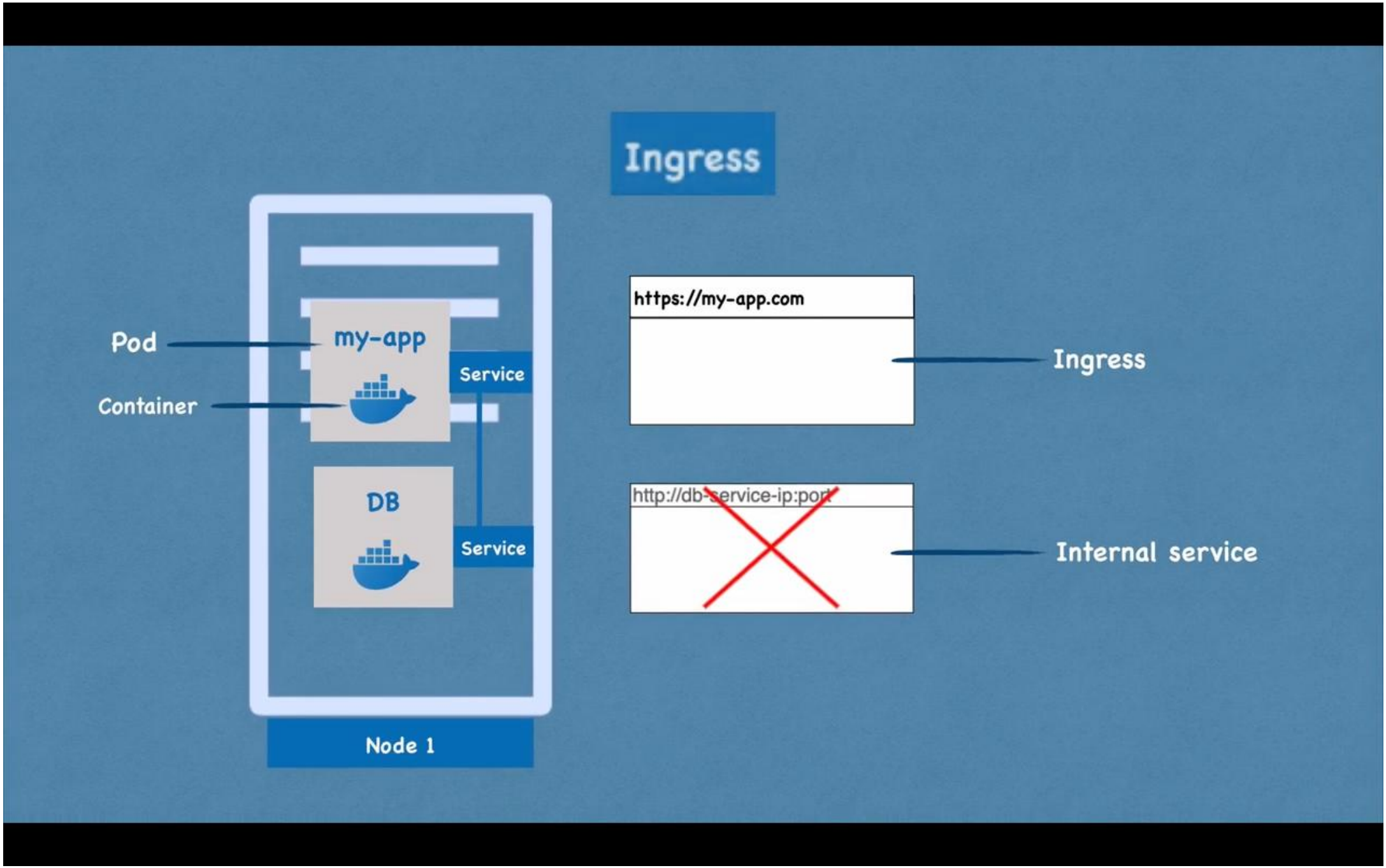
- permanent IP address

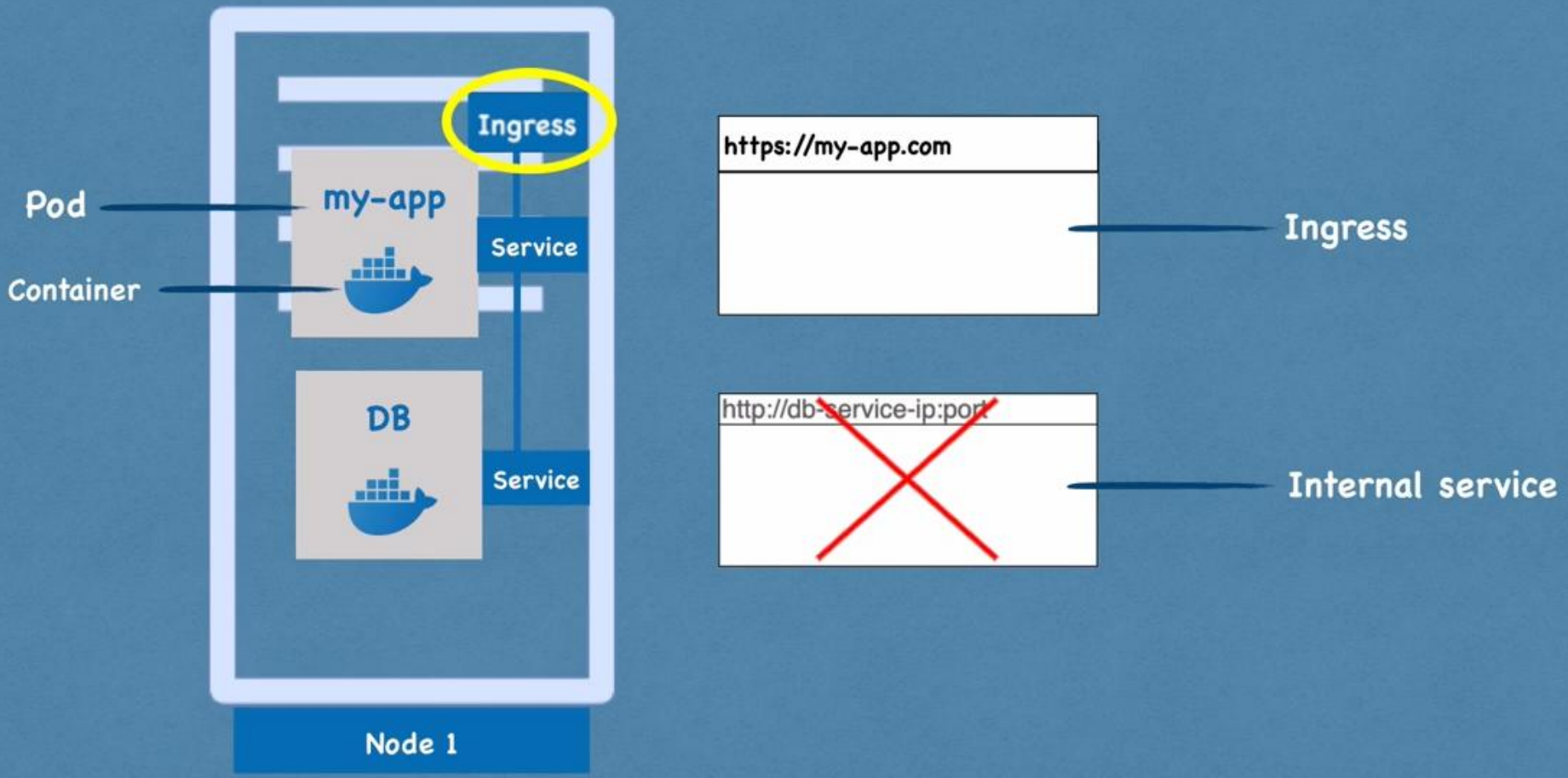

Pod
Container
my-app
Service
DB
Service
Node 1
http://my-app-service-ip:port
External service
http://db-service-ip:port
Internal service

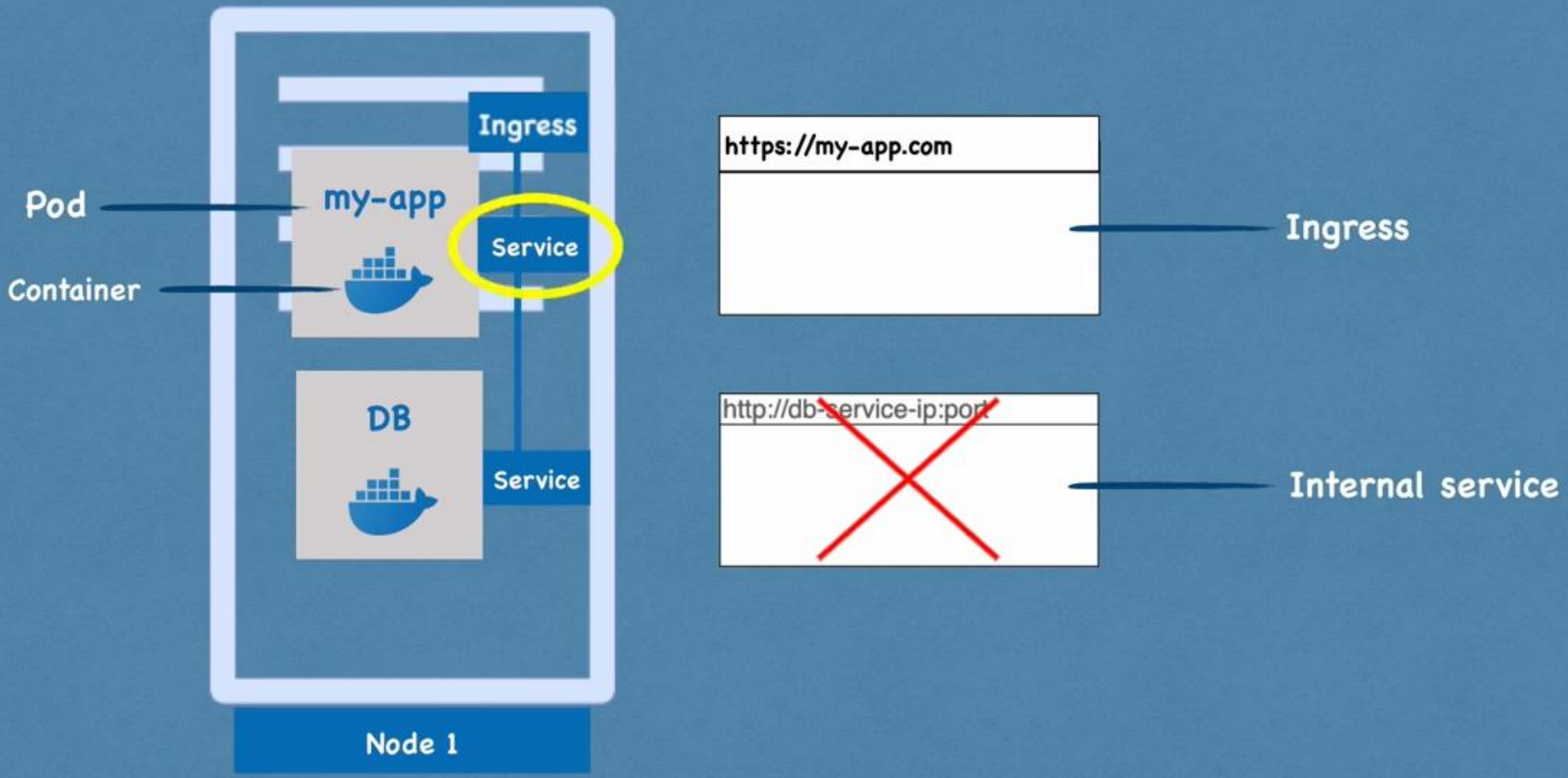

http://my-app-service-ip:port
External service
http://db-service-ip:port
Internal service


http://124.89.101.2:8080
External service
http://db-service-ip:port
Internal service

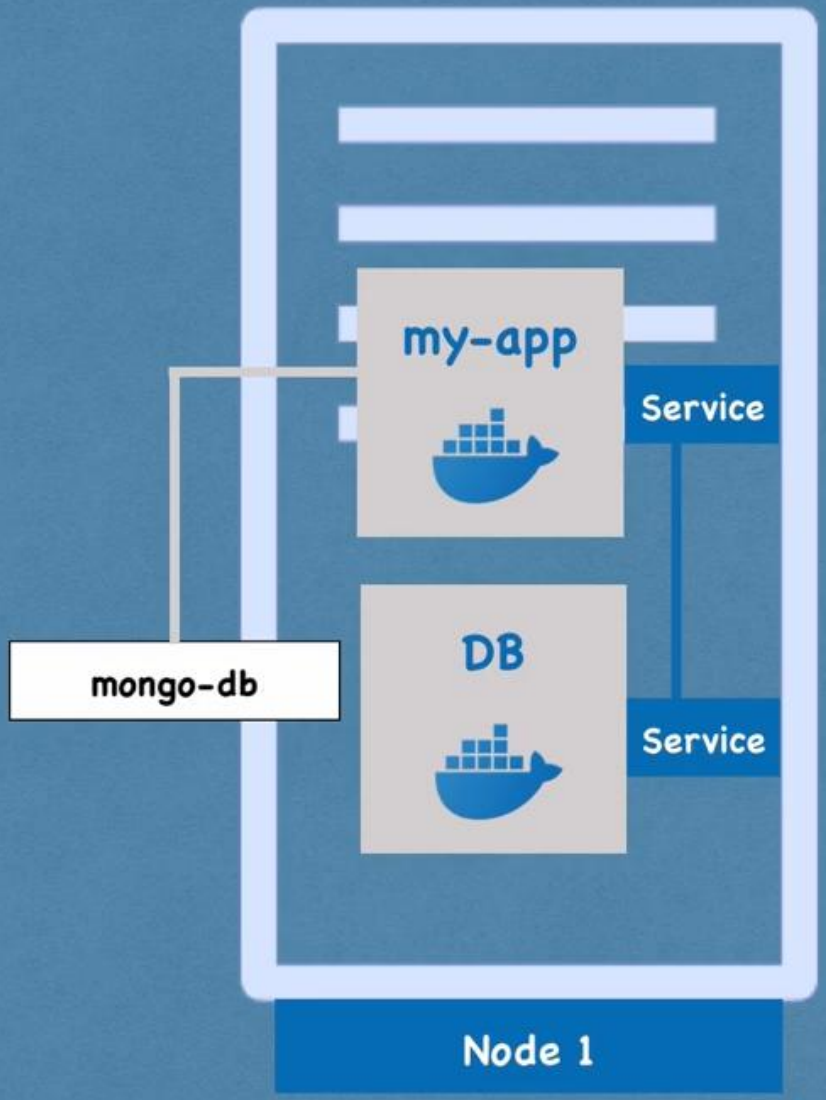

https://my-app.com
External service
http://db-service-ip:port
Internal service


Ingress
Pod
my-app
Service
Container
DB
Service
Node 1
https://my-app.com
Ingress
http://db-service-ip:port
Internal service

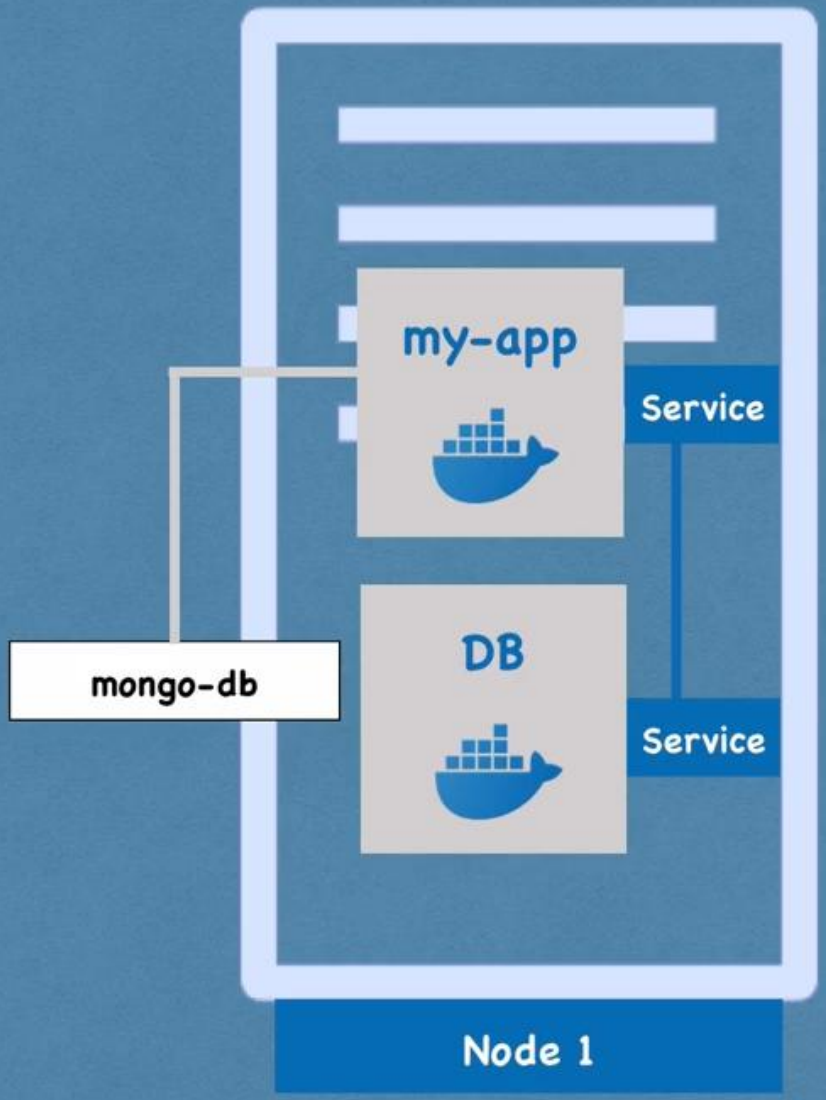

Ingress
Pod
my-app
Service
Container
DB
Service
Node 1
https://my-app.com
Ingress
http://db-service-ip:port
Internal service


Ingress
Pod
my-app
Service
Container
DB
Service
Node 1
https://my-app.com
Ingress
http://db-service-ip:port
Internal service

ConfigMap and Secret



Database Url usually in the built application!



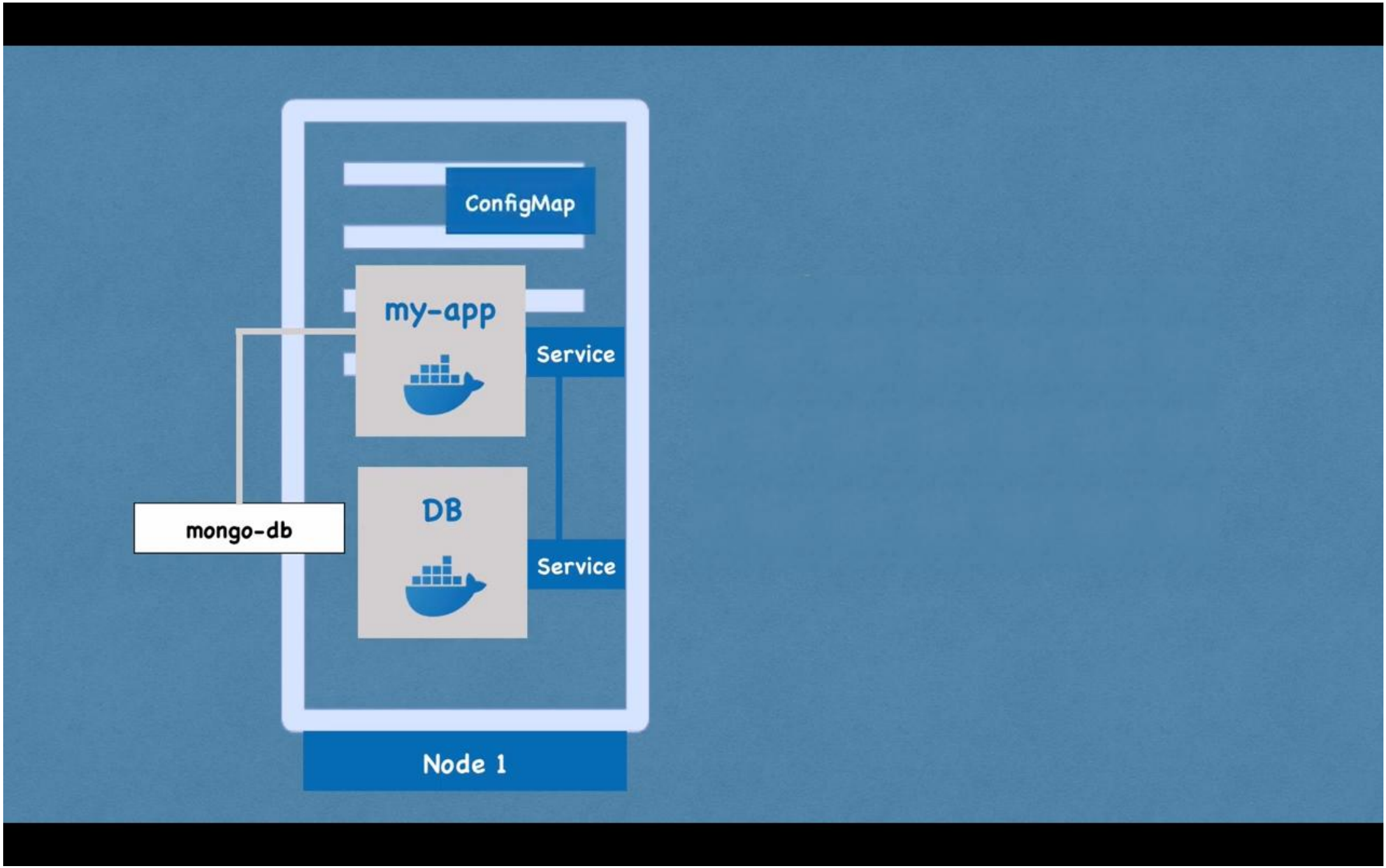
Database Url usually in the built application!

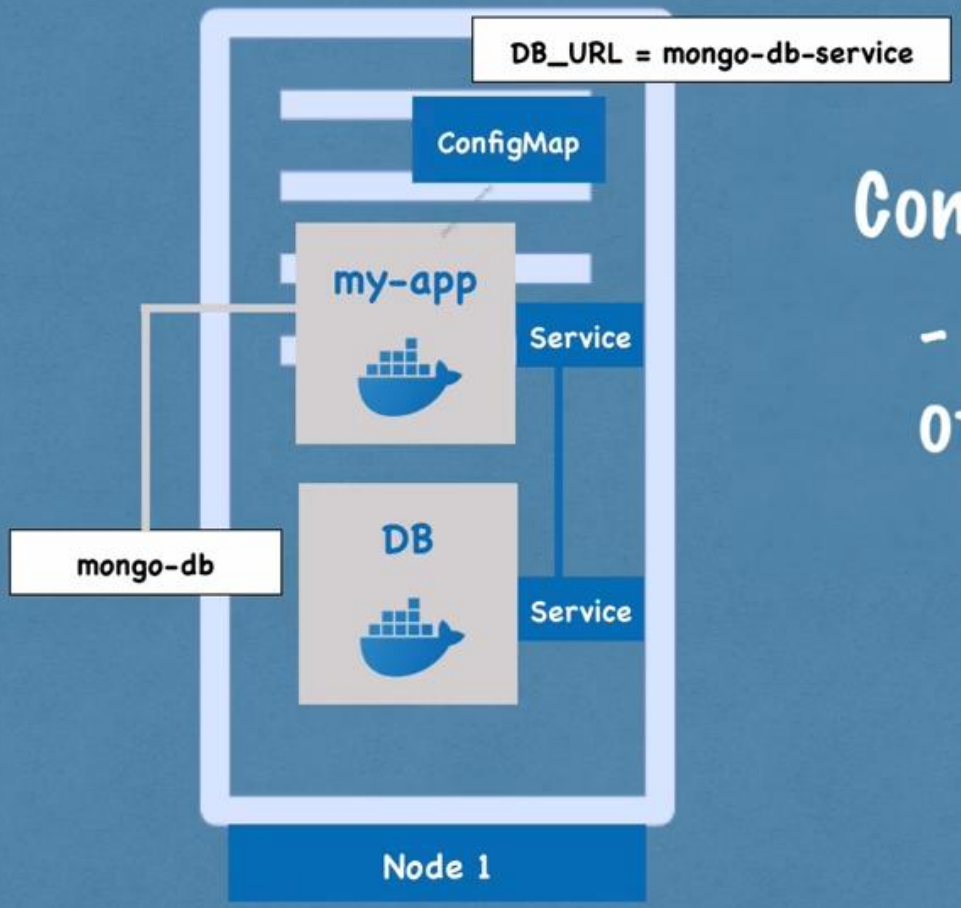
Repository

re-build

push it to repo

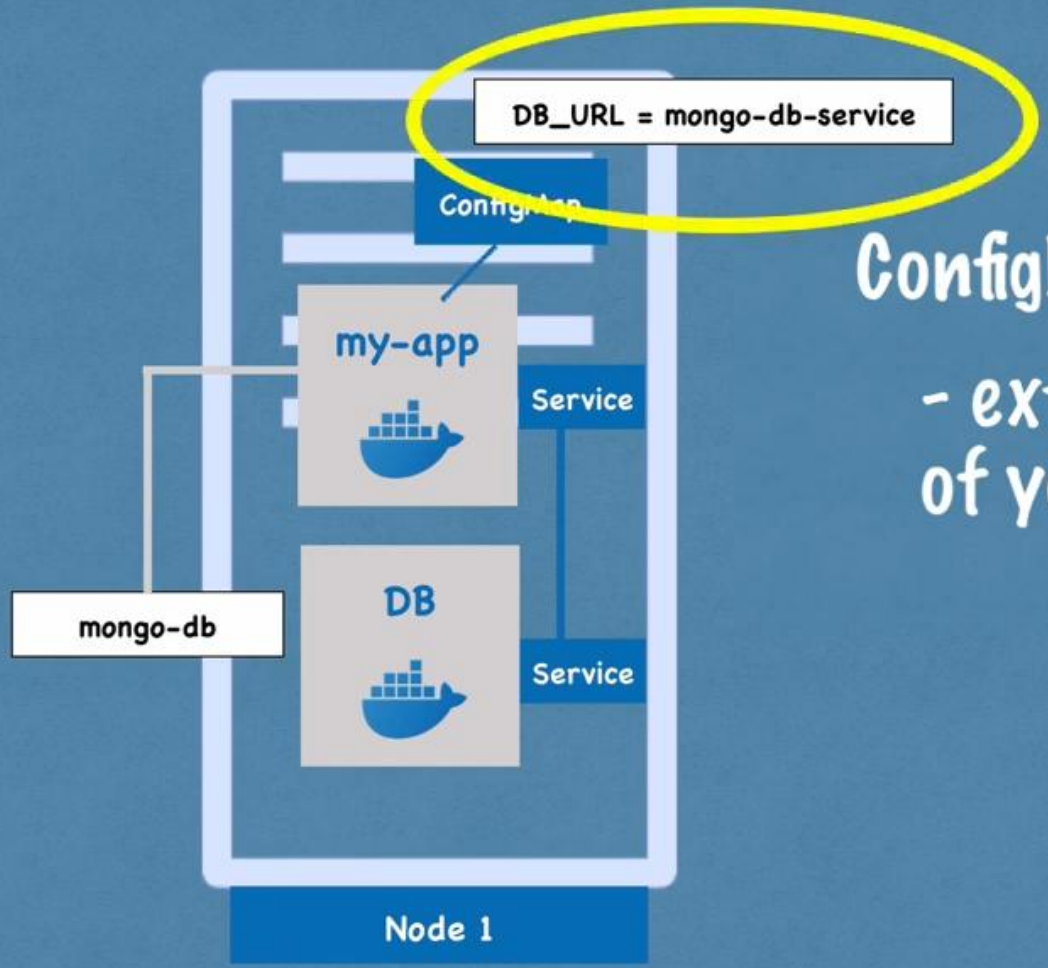
pull it in your pod


ConfigMap
my-app
Service
DB
Service
mongo-db
Node 1



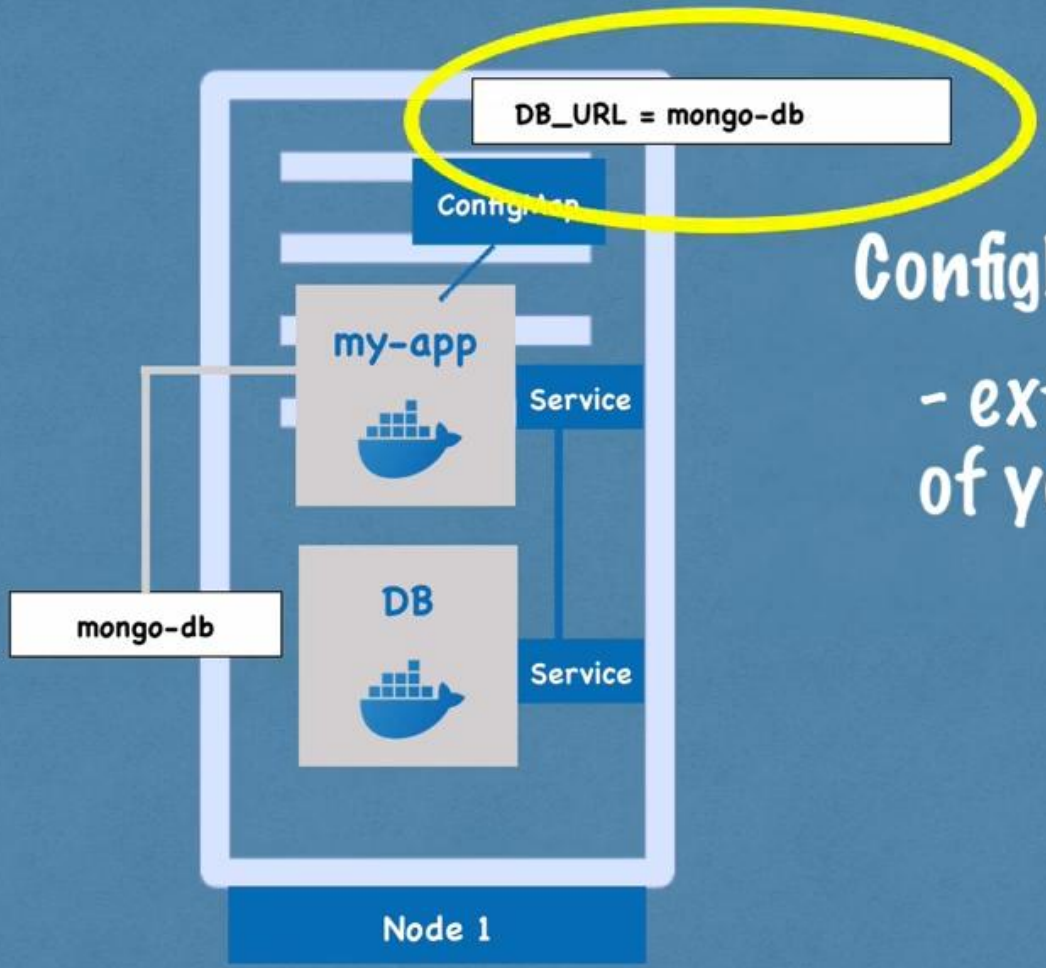
ConfigMap:

- external configuration of your application



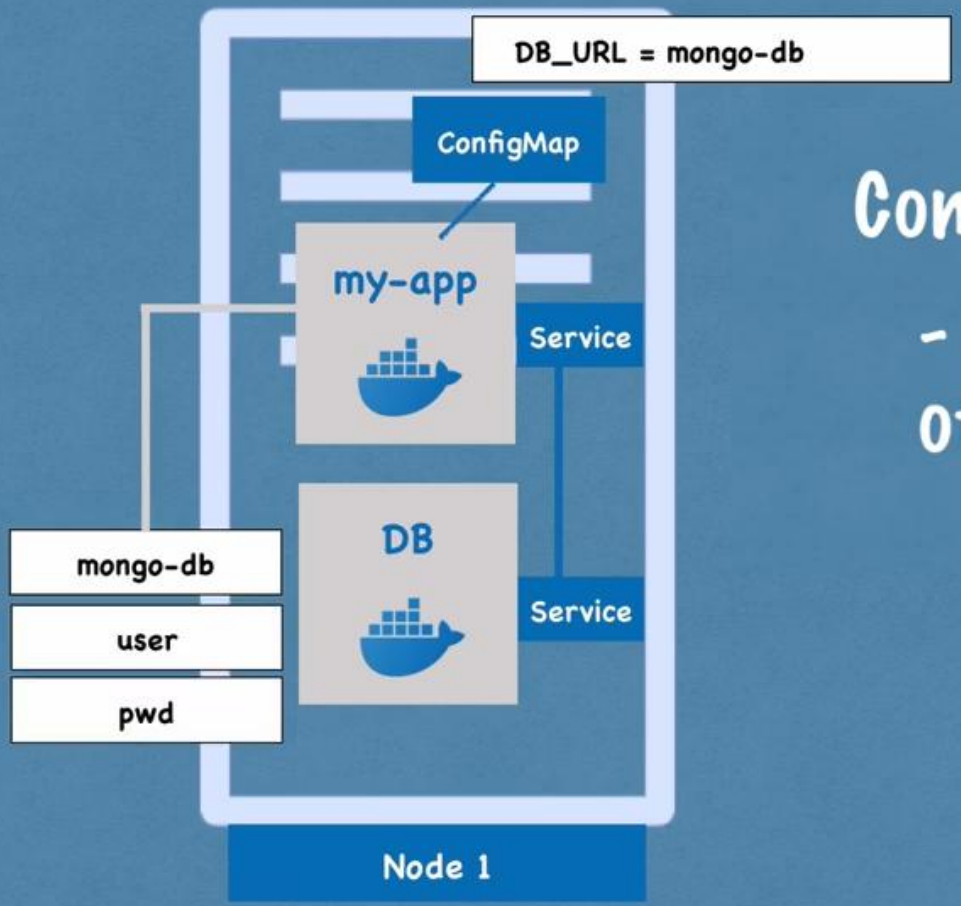
ConfigMap:

- external configuration of your application



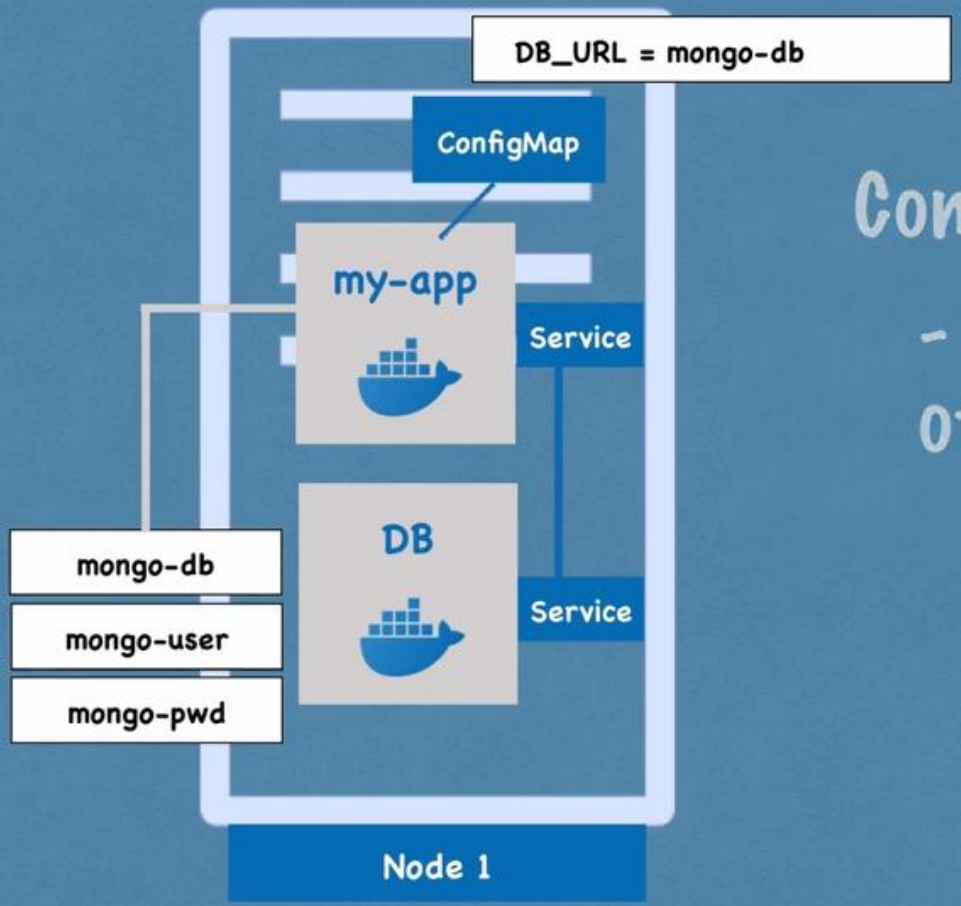
ConfigMap:

- external configuration of your application



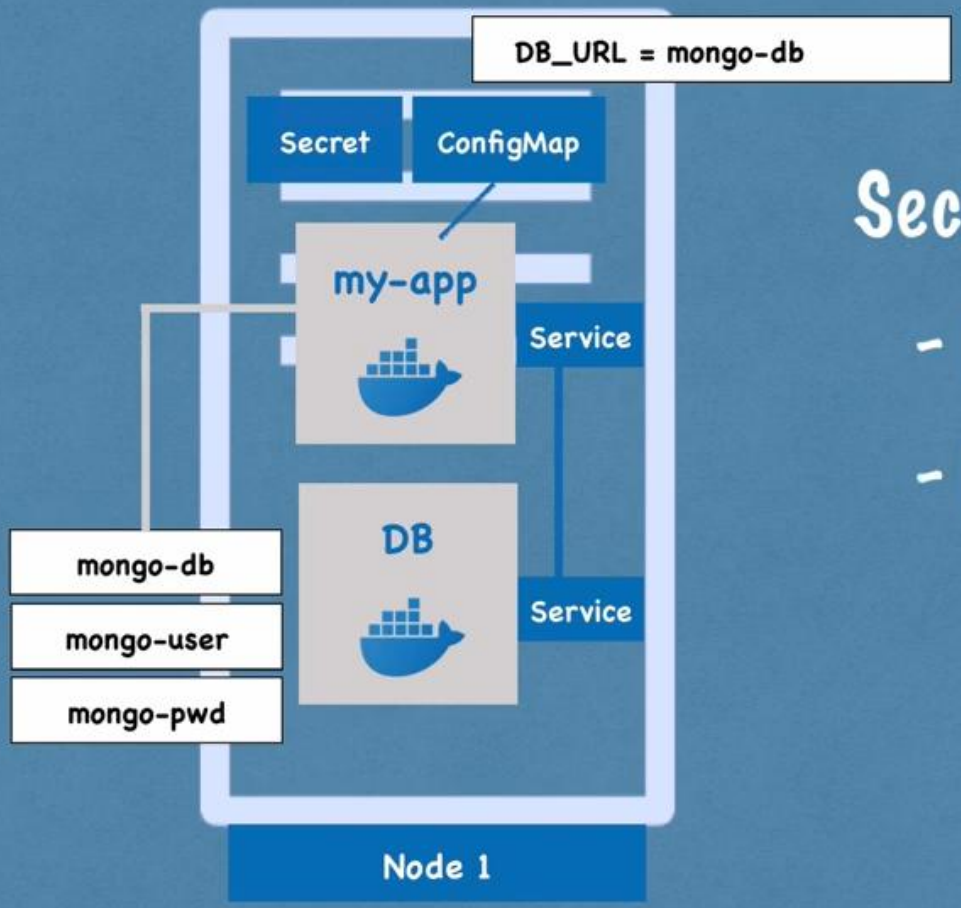
ConfigMap:

- external configuration of your application



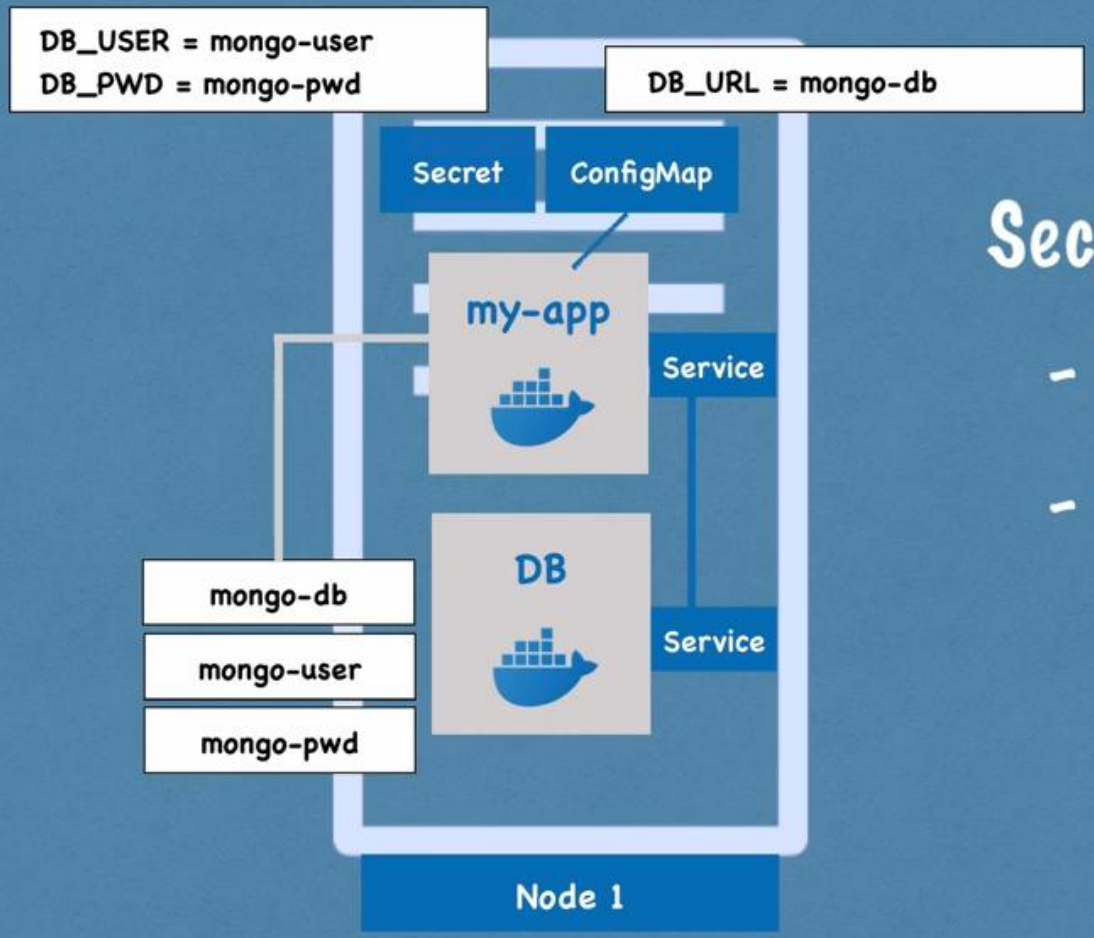
ConfigMap:

- external configuration of your application



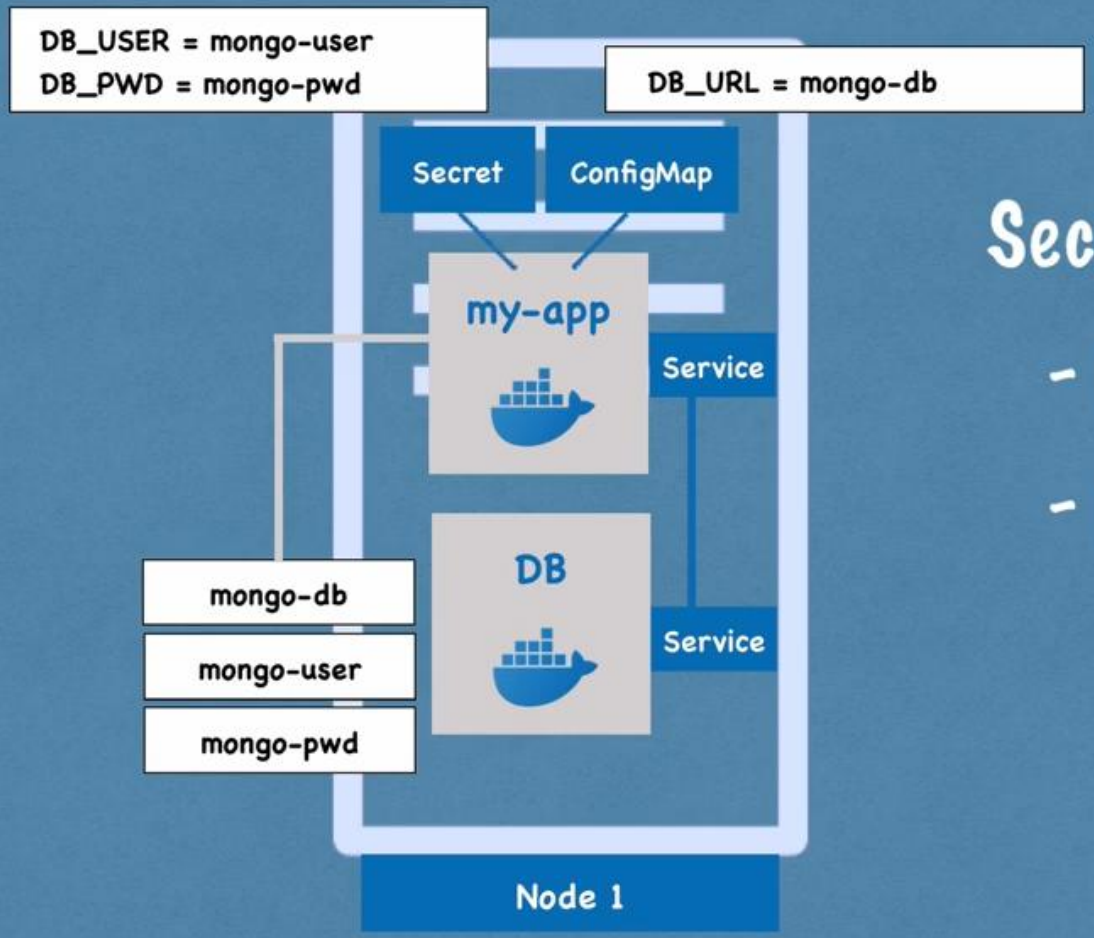
Secret:

- used to store secret data
- base64 encoded



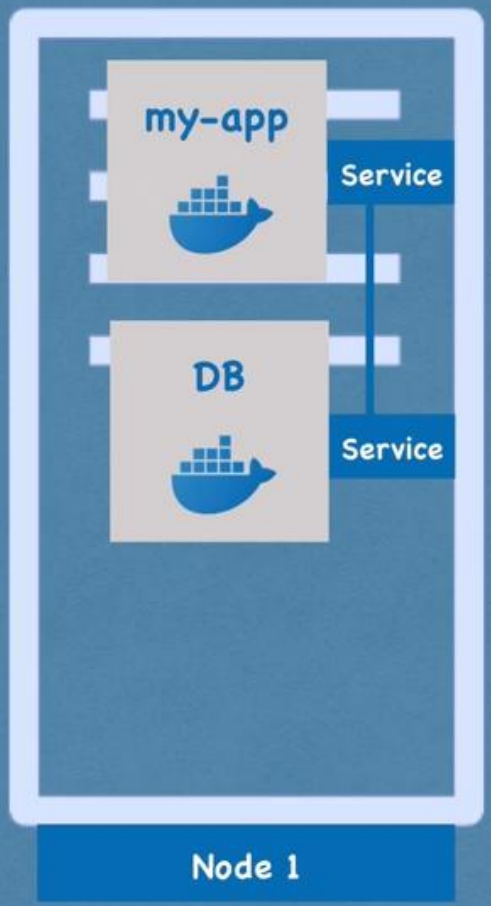
Secret:

- used to store secret data
- base64 encoded



Secret:

- used to store secret data
- base64 encoded

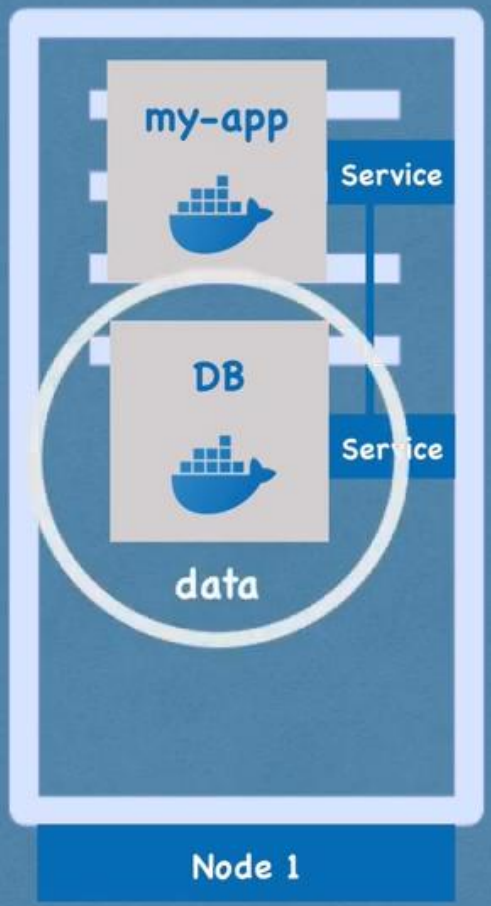


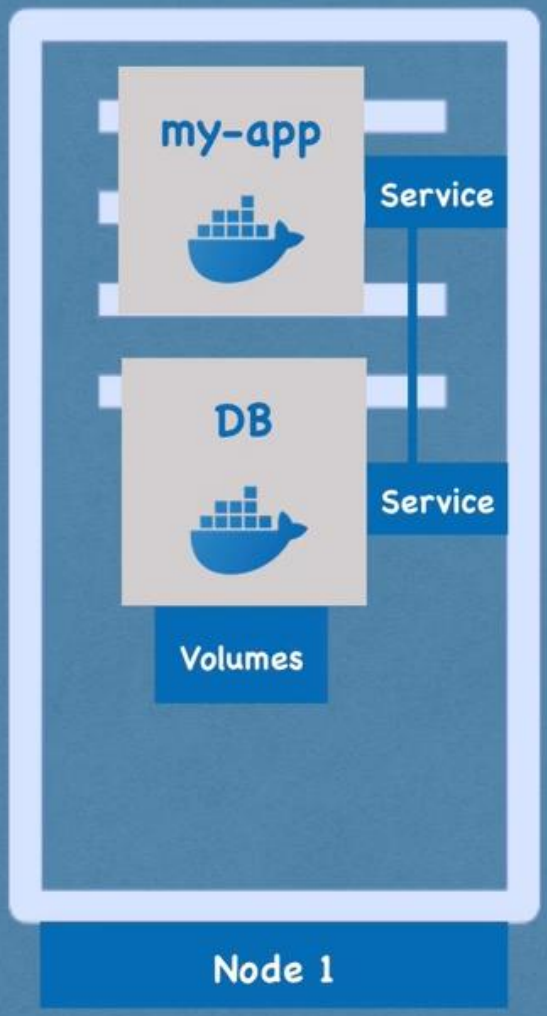
Pod

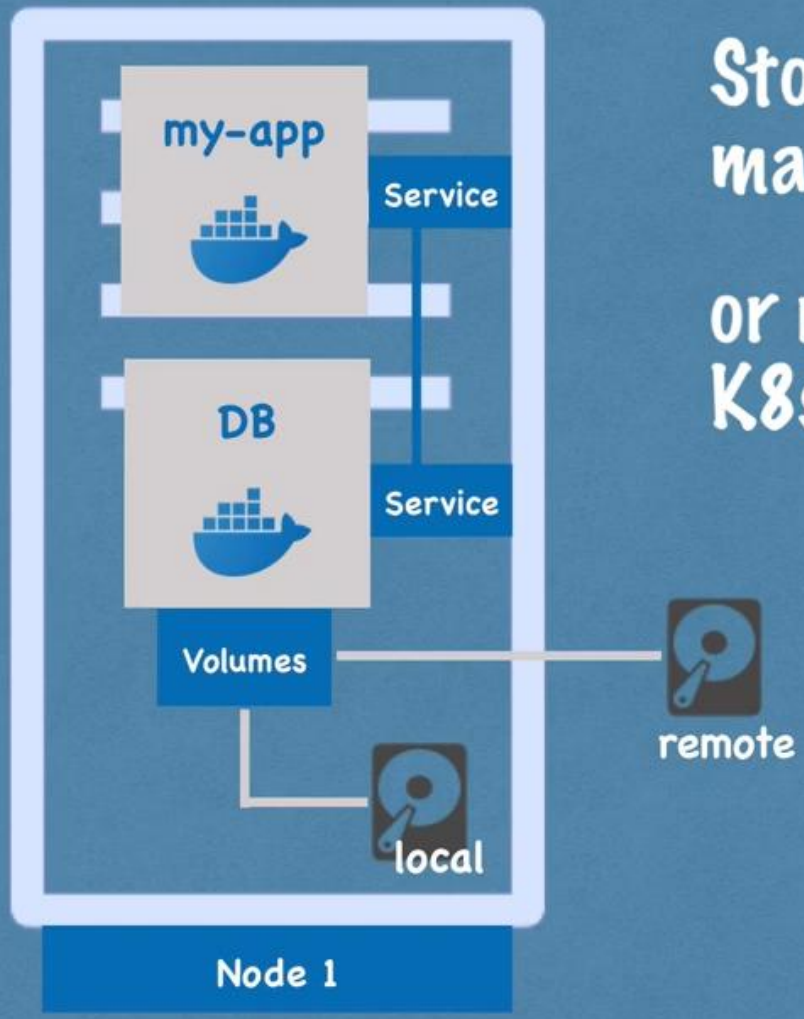
Service Ingress

ConfigMap Secrets

Volumes

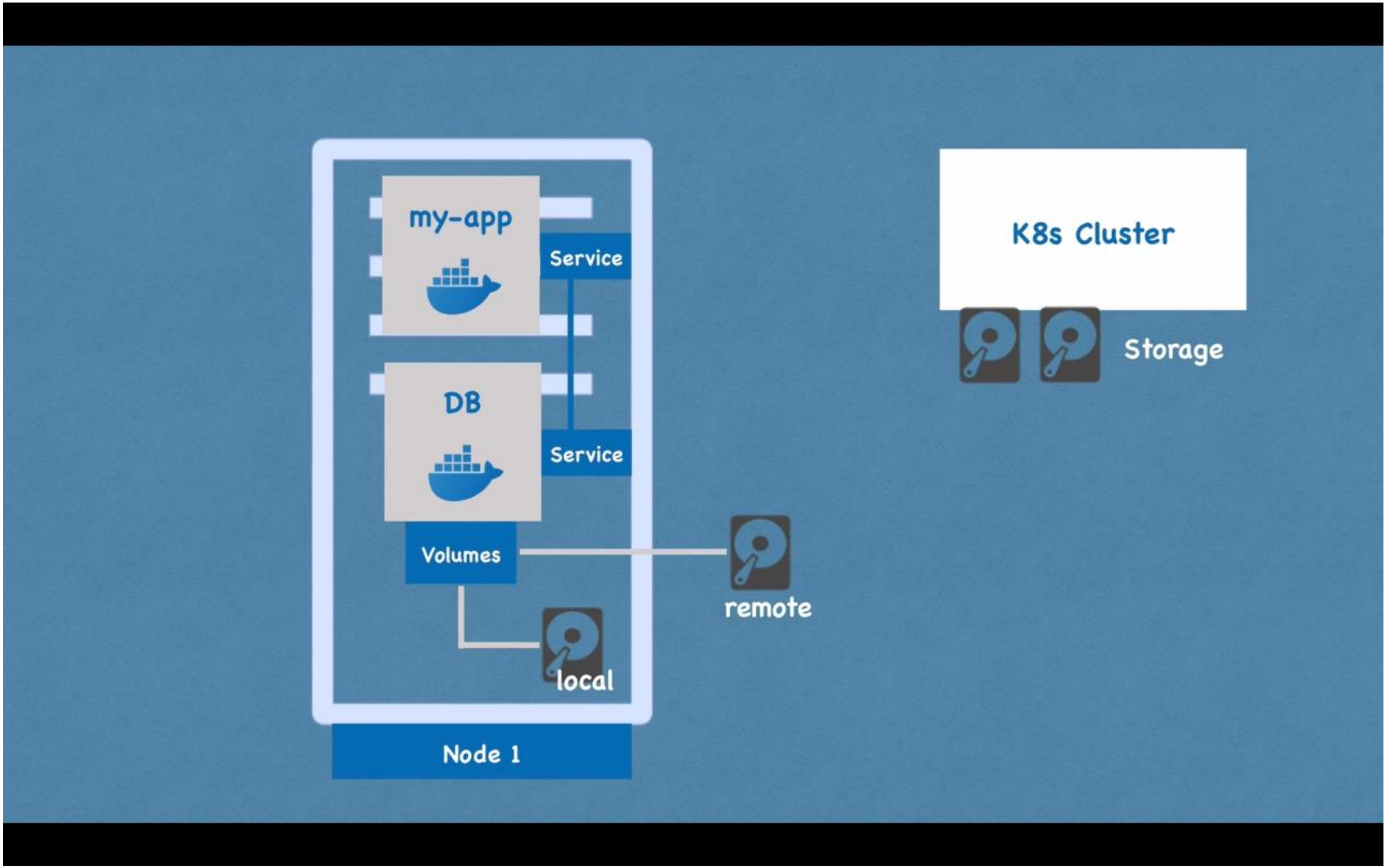

my-app
Service
DB
Service
data
Node 1

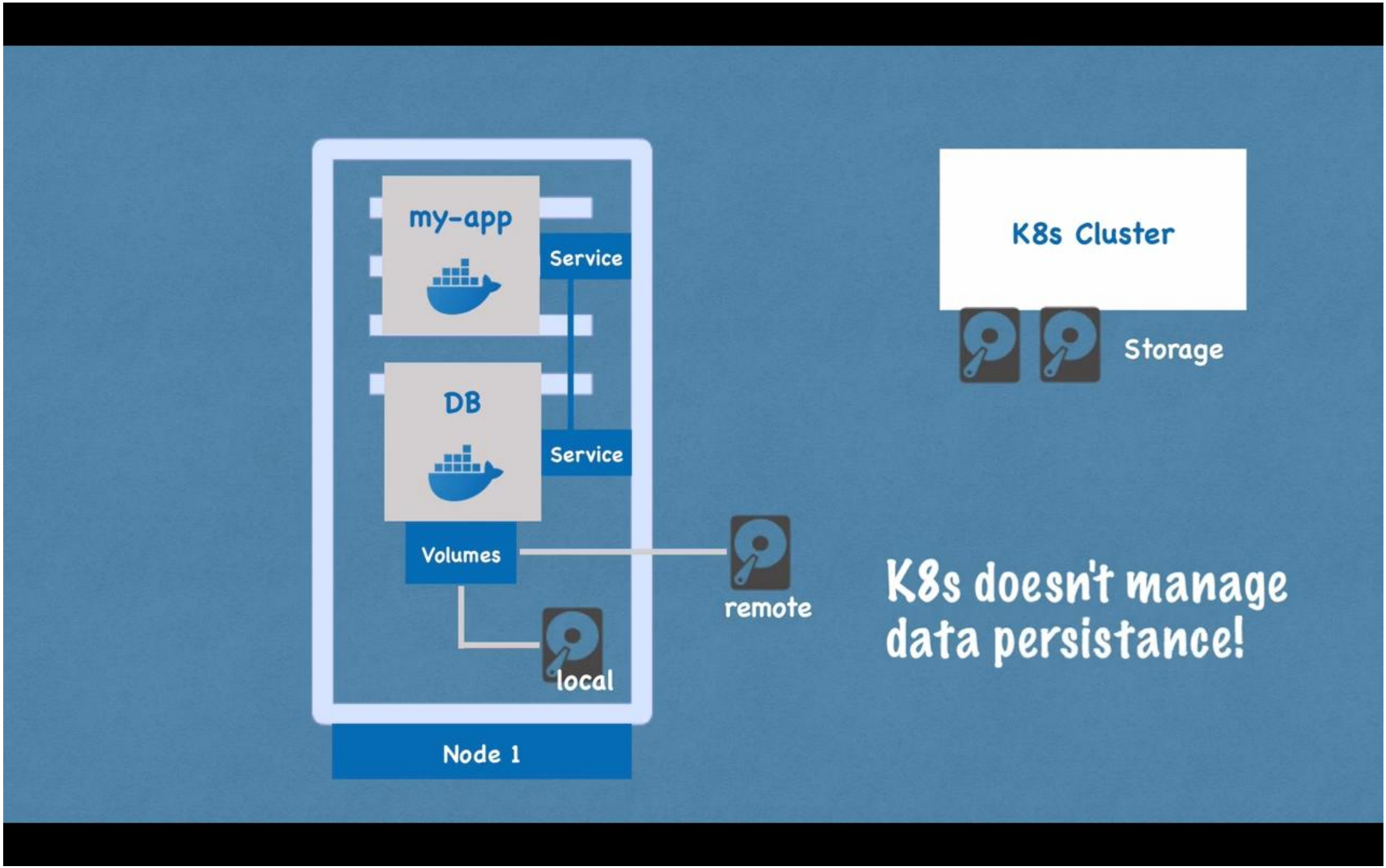

my-app
Service
DB
Service
Volumes
Node 1



Storage can be on a local machine

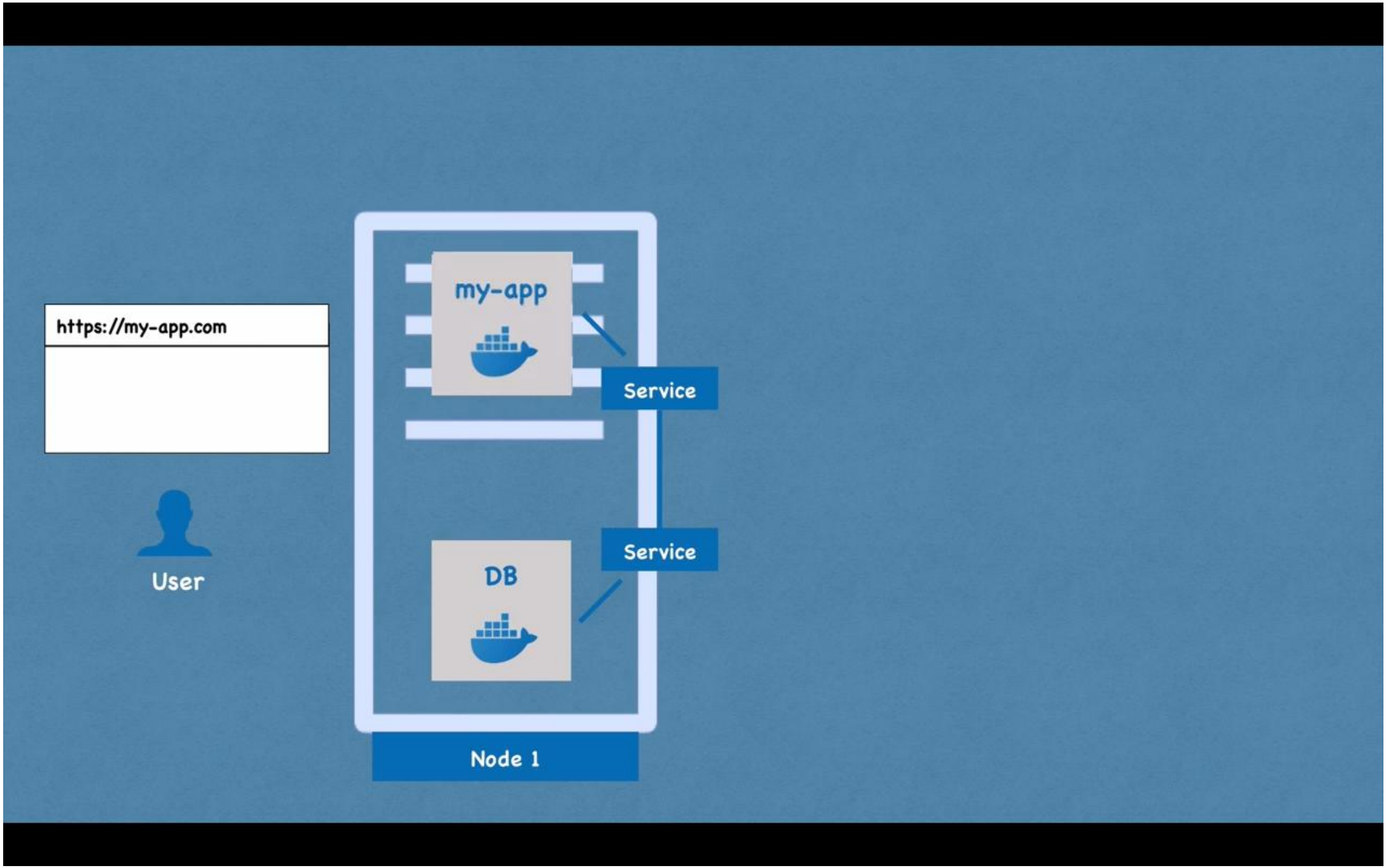
or remote, outside of the K8s cluster

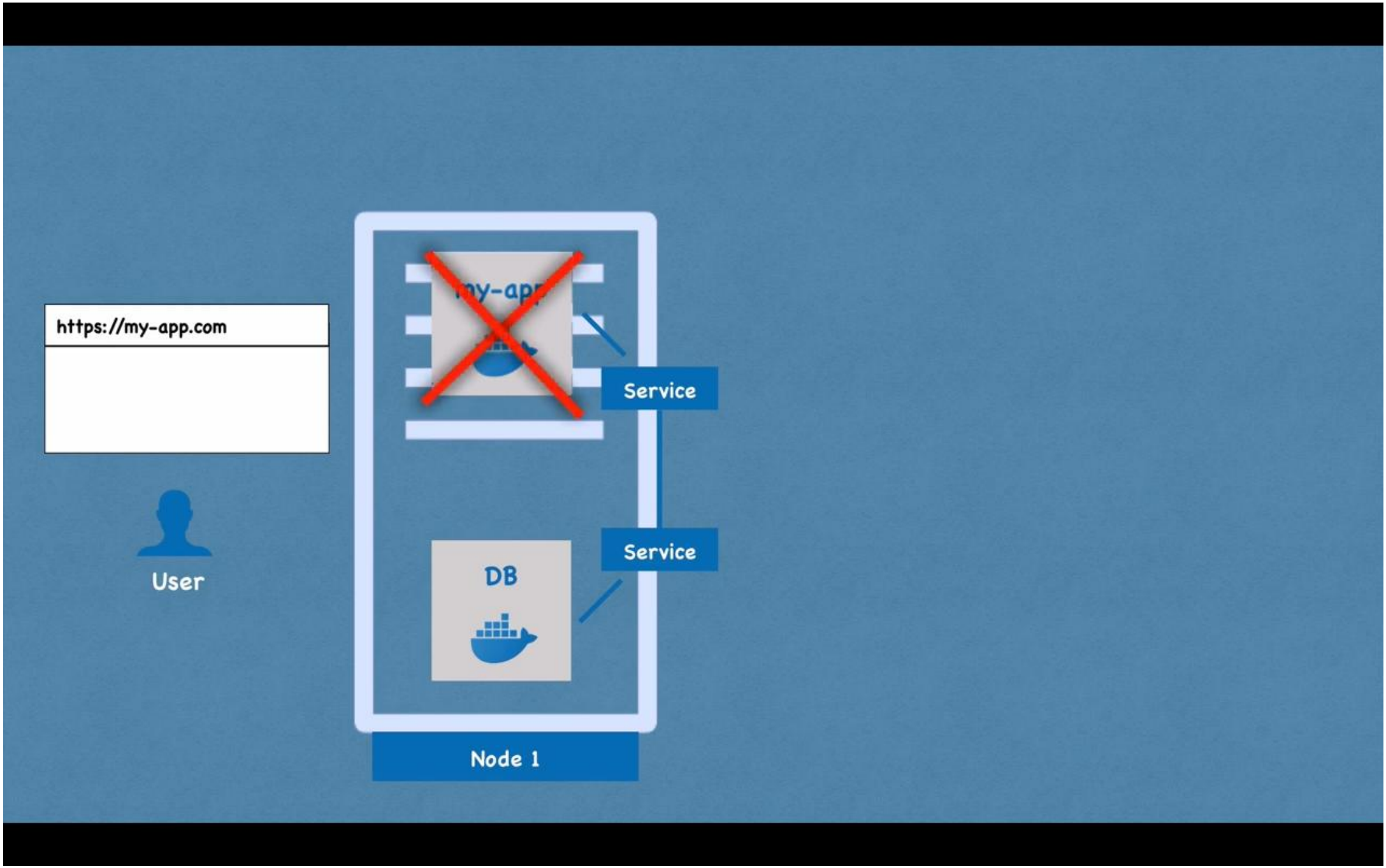

my-app
Service
DB
Service
Volumes
remote
local
Node 1
K8s Cluster
Storage

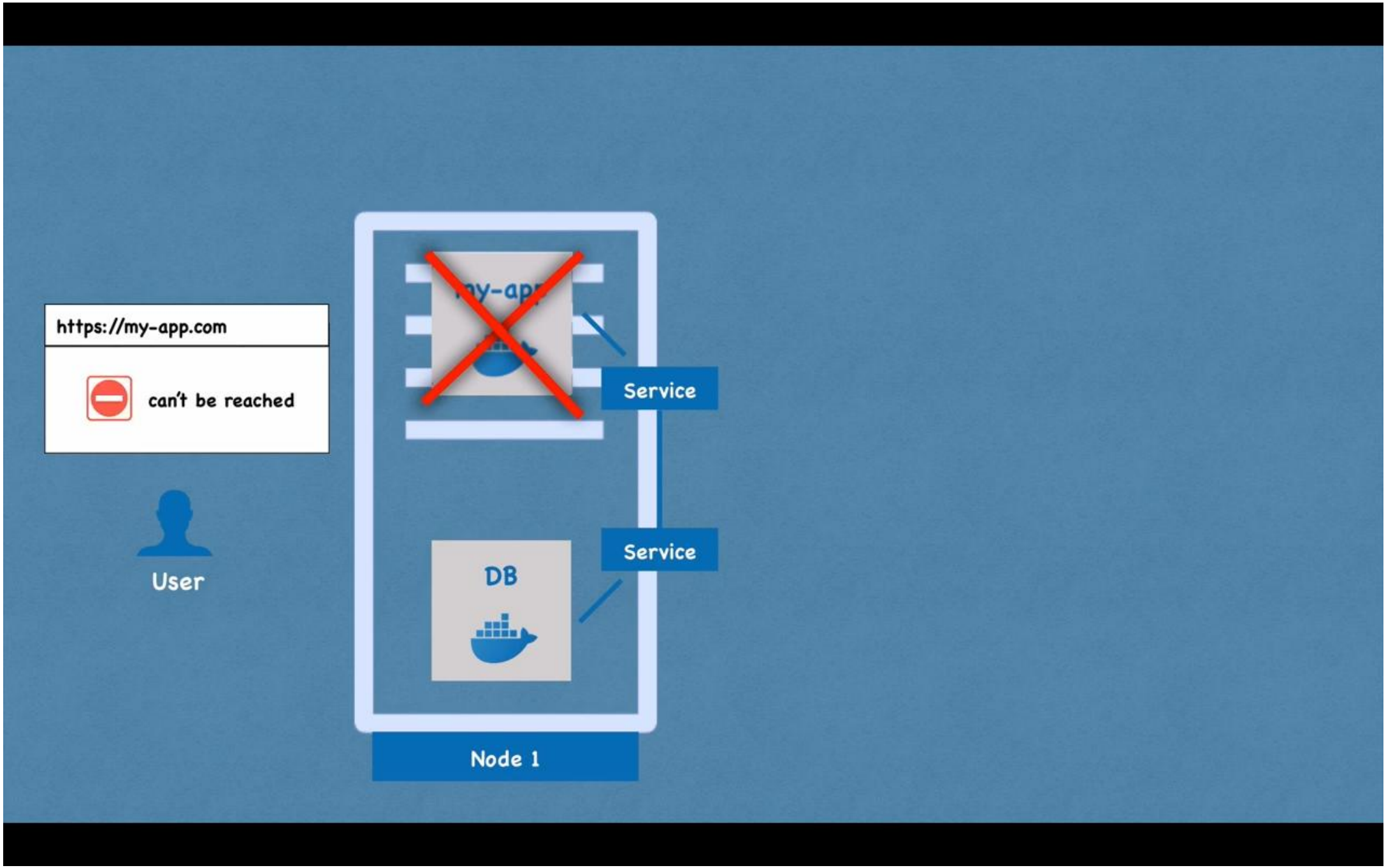

my-app
Service
DB
Service
Volumes
remote
local
Node 1
K8s Cluster
Storage
K8s doesn't manage
data persistance!

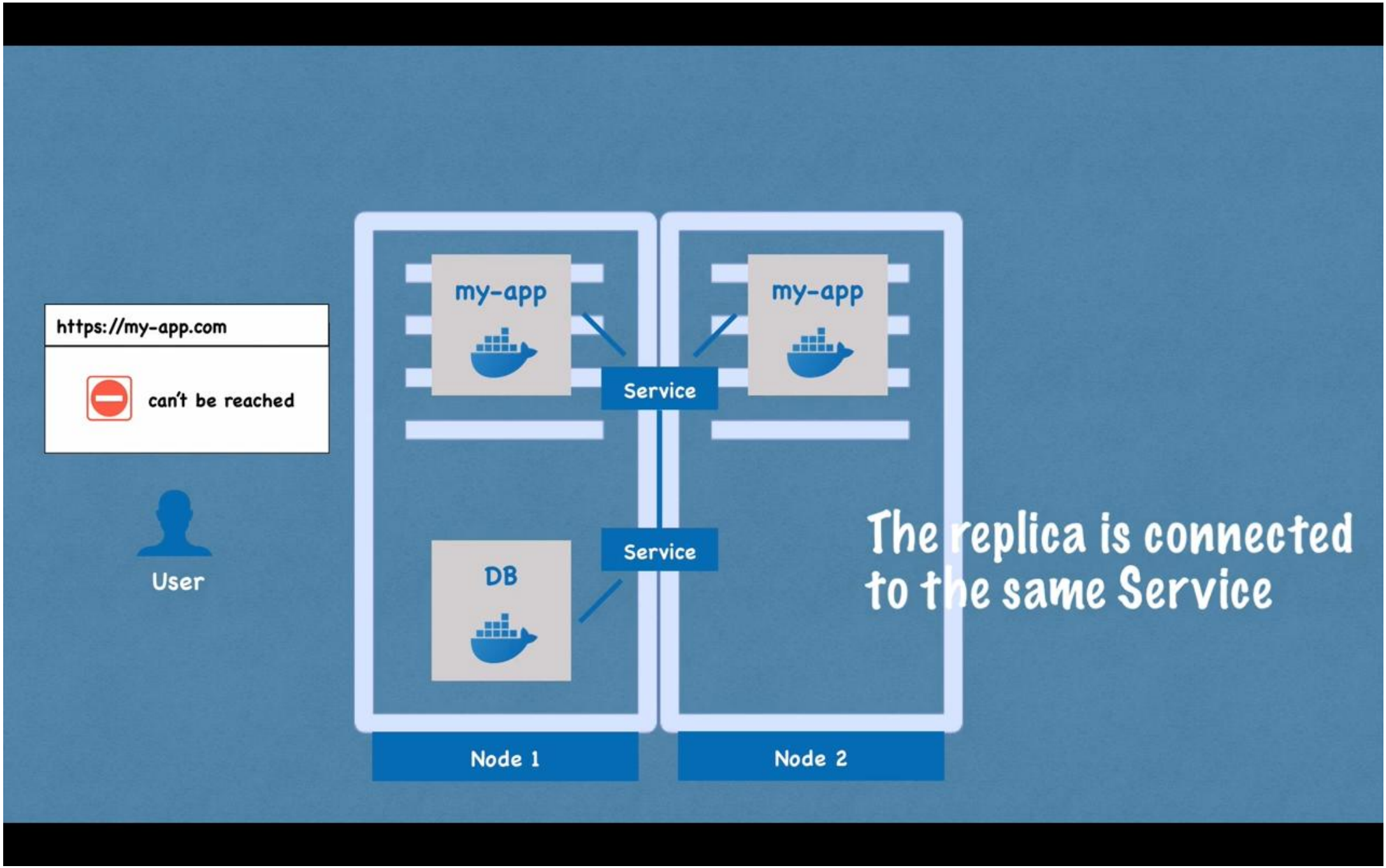
Deployment and Stateful Set

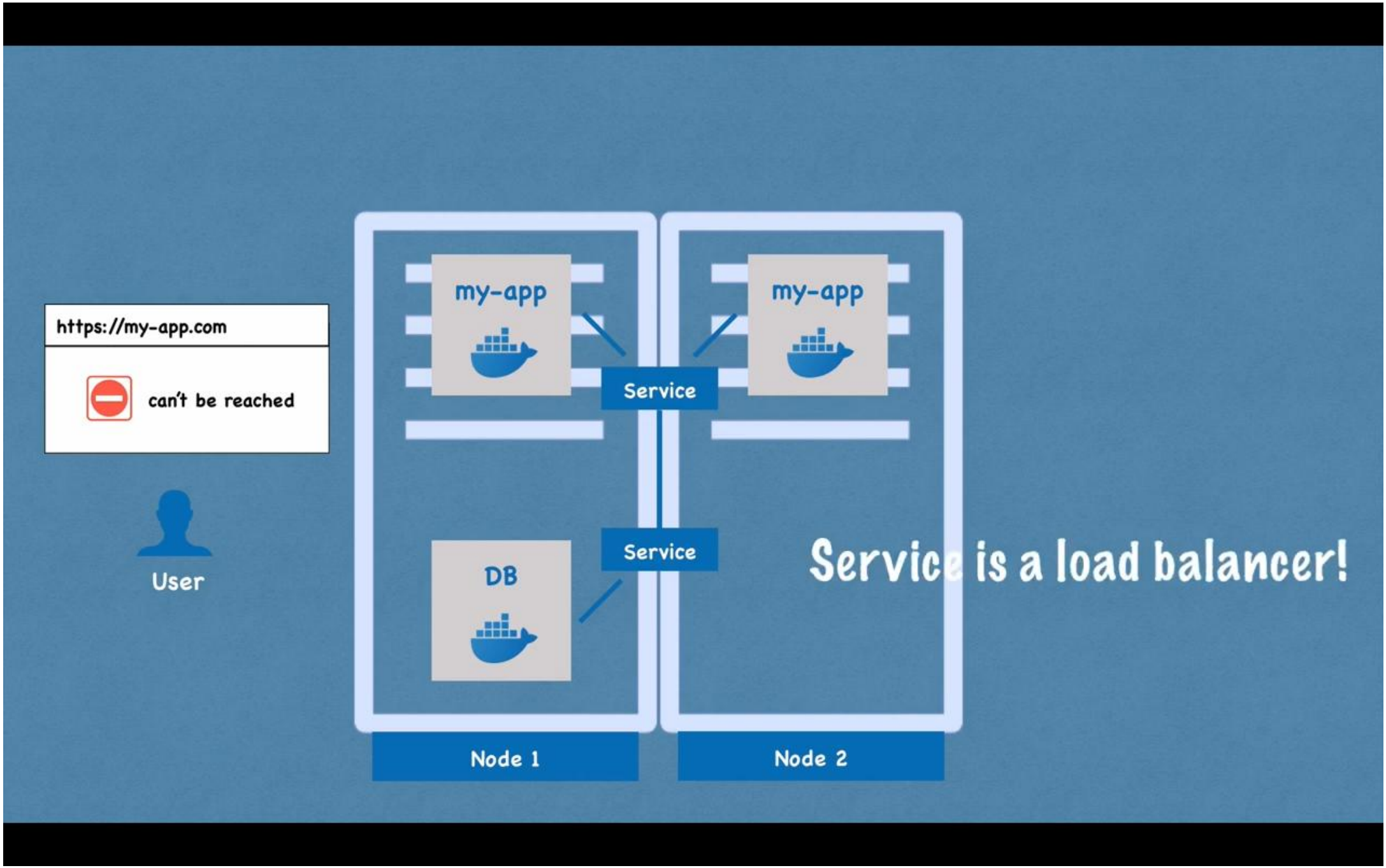
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app
          image: my-image
          env:
            - name: SOME_ENV
              value: $SOME_ENV
          ports:
            - containerPort: 8080
```

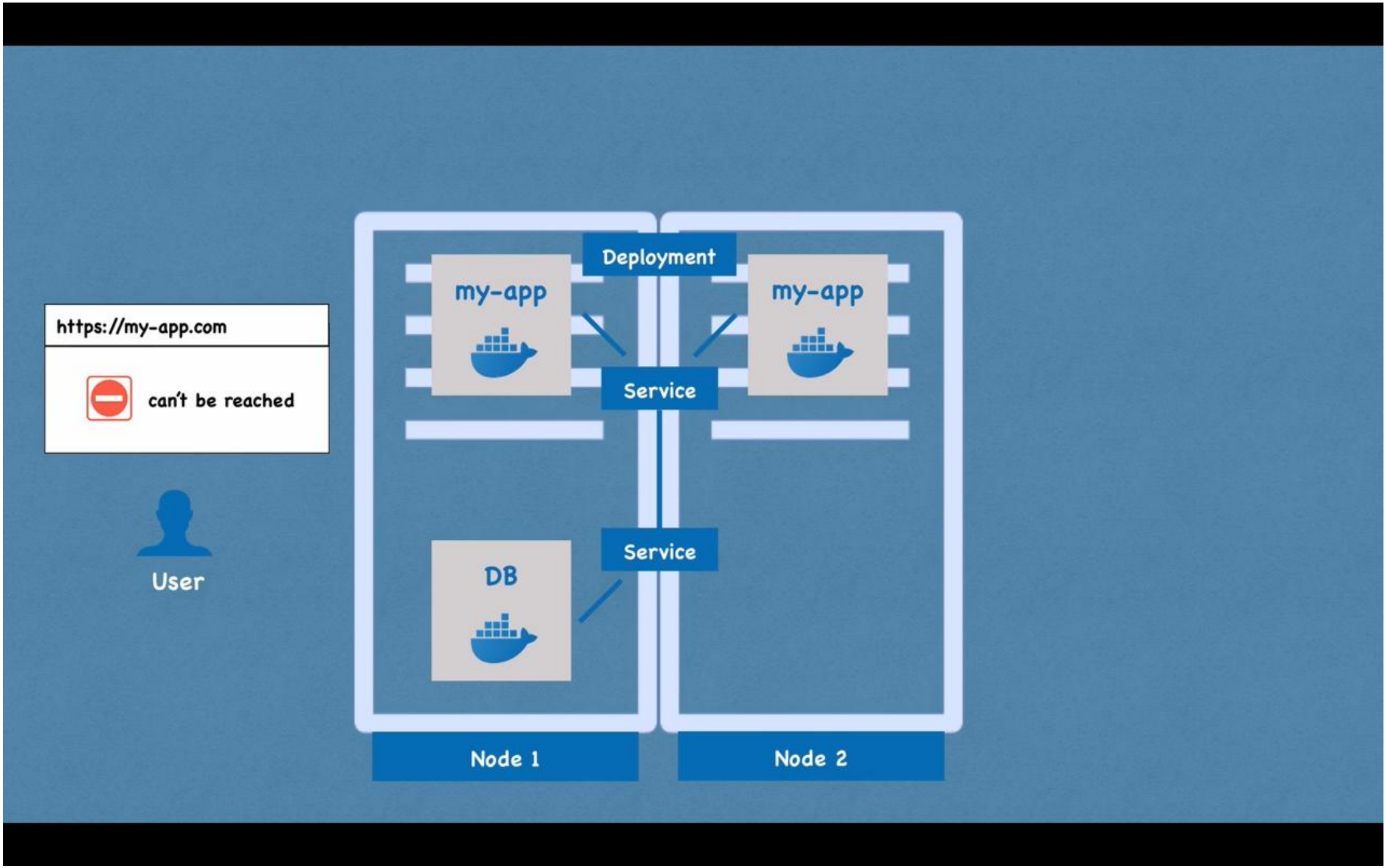

https://my-app.com
User
my-app
Service
DB
Service
Node 1

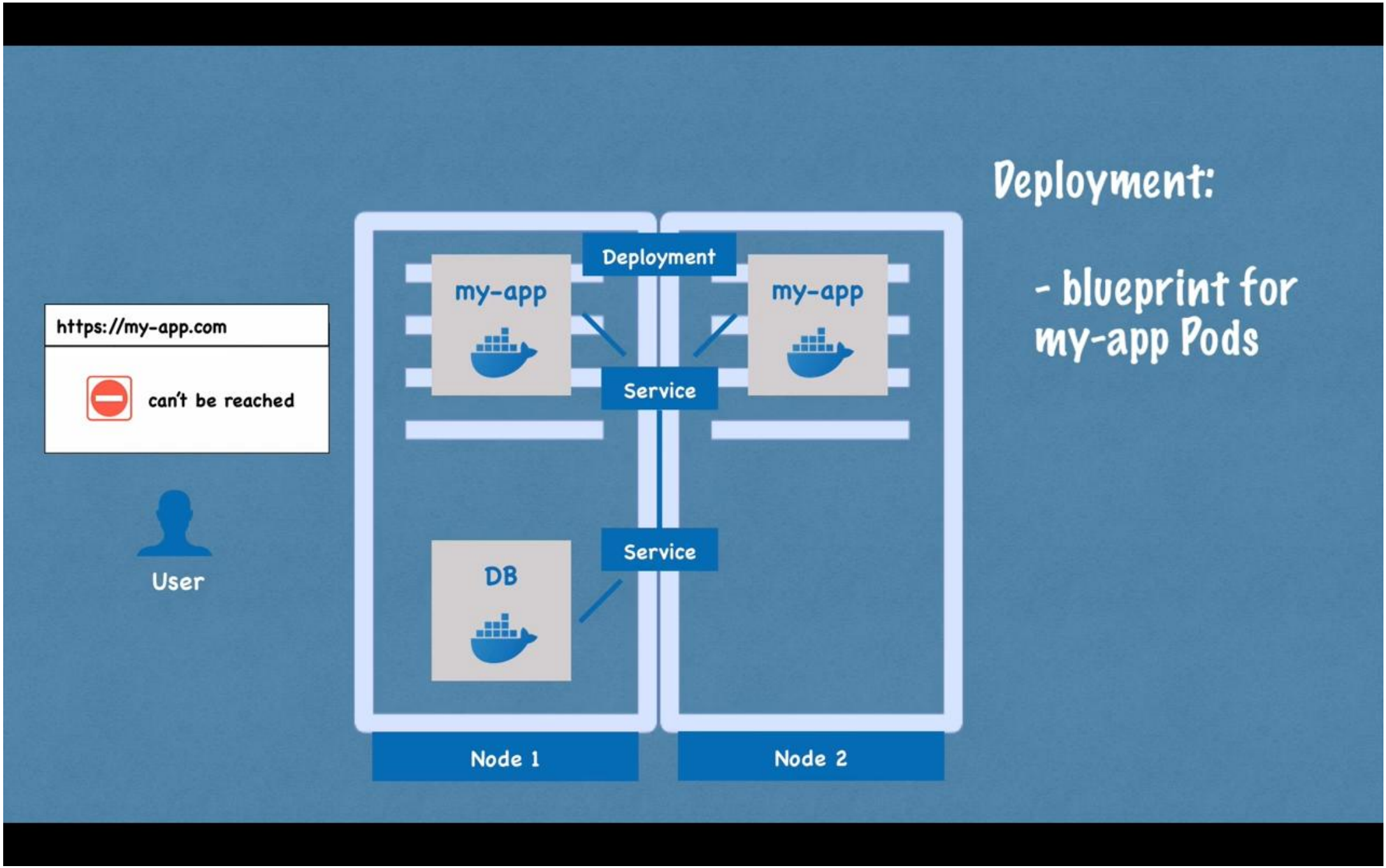

https://my-app.com
User
Service
Service
DB
Node 1

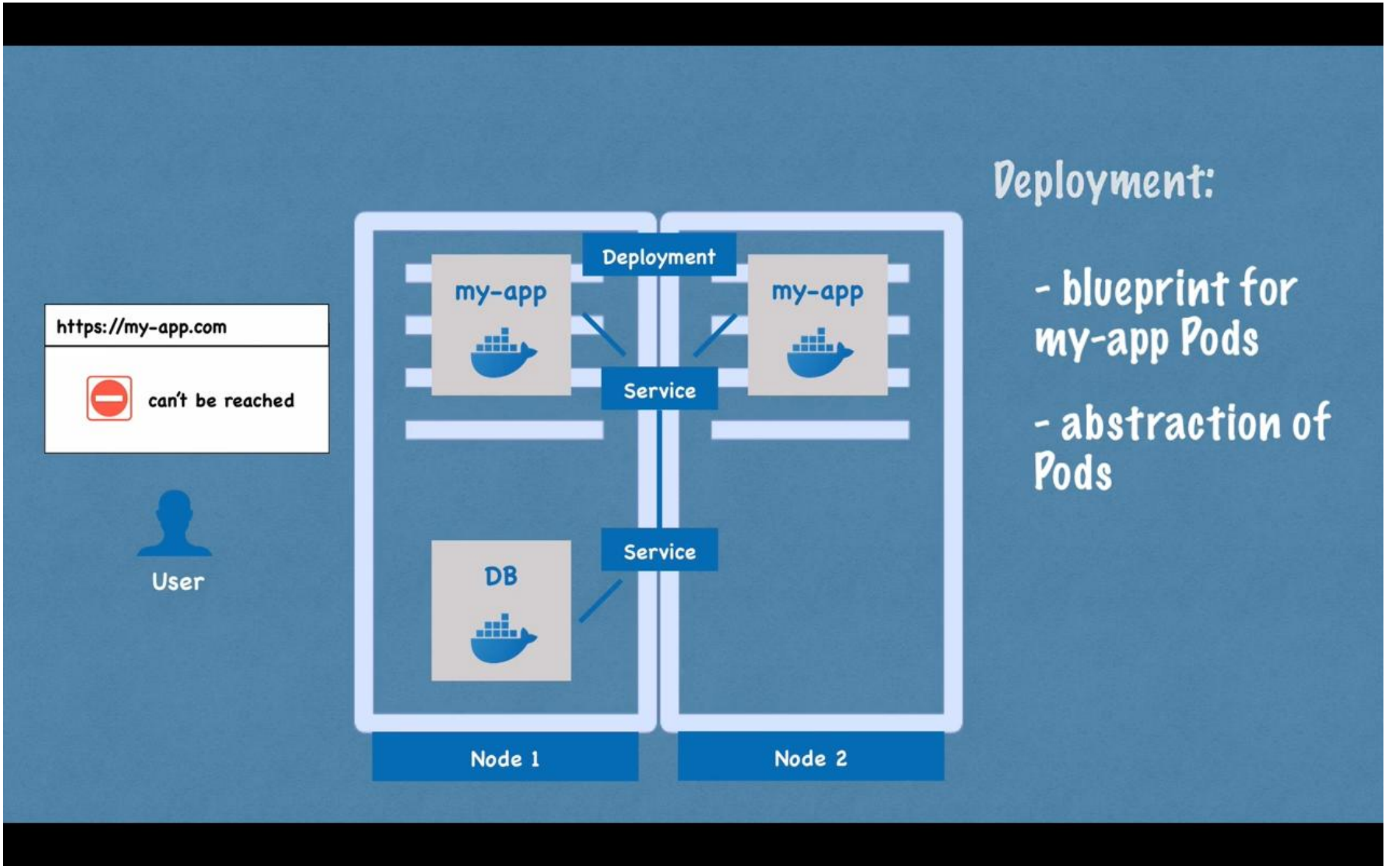

https://my-app.com
can't be reached
User
Service
Service
DB
Node 1

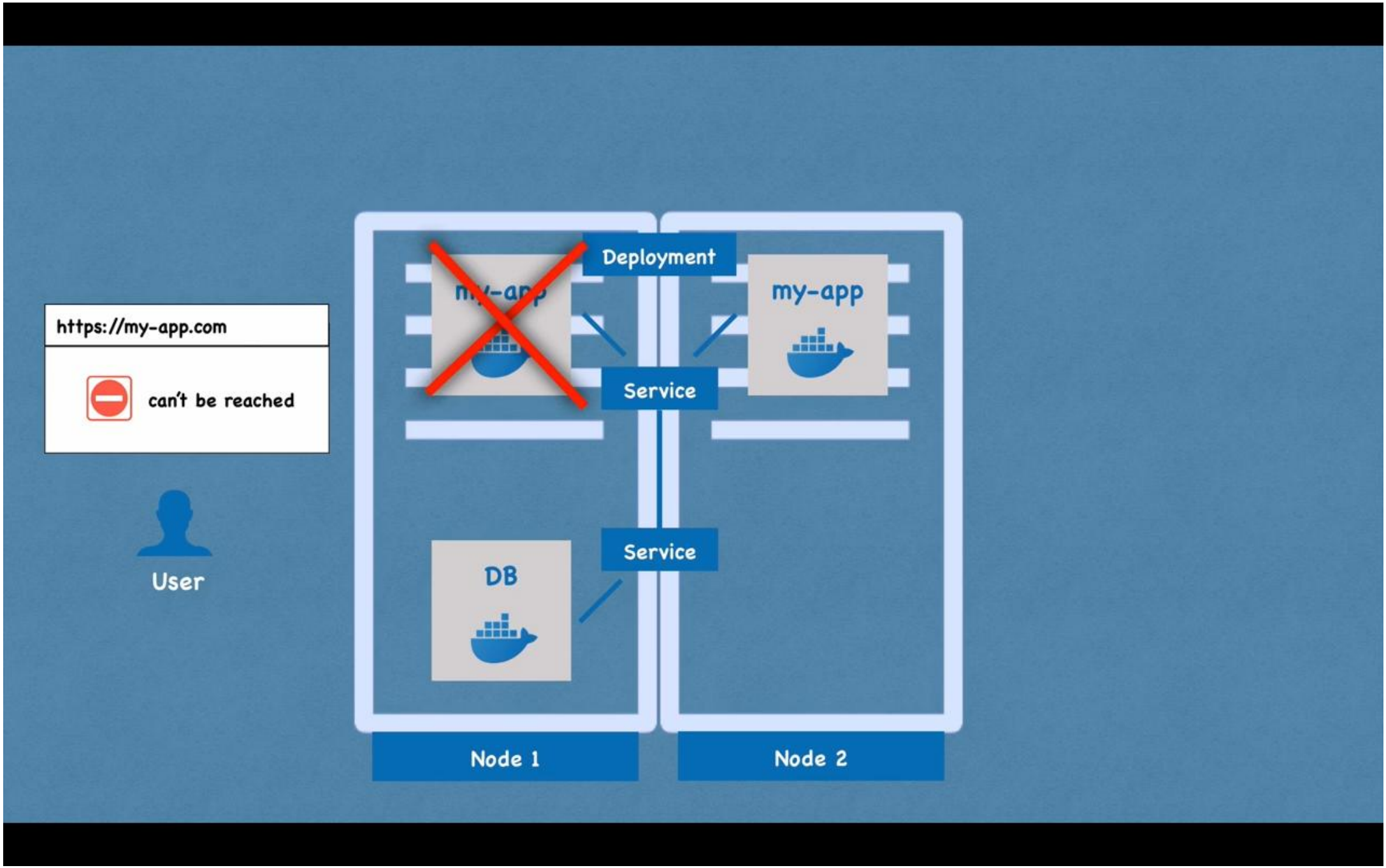

https://my-app.com
can't be reached
User
my-app
my-app
Service
Service
DB
Node 1
Node 2
The replica is connected to the same Service

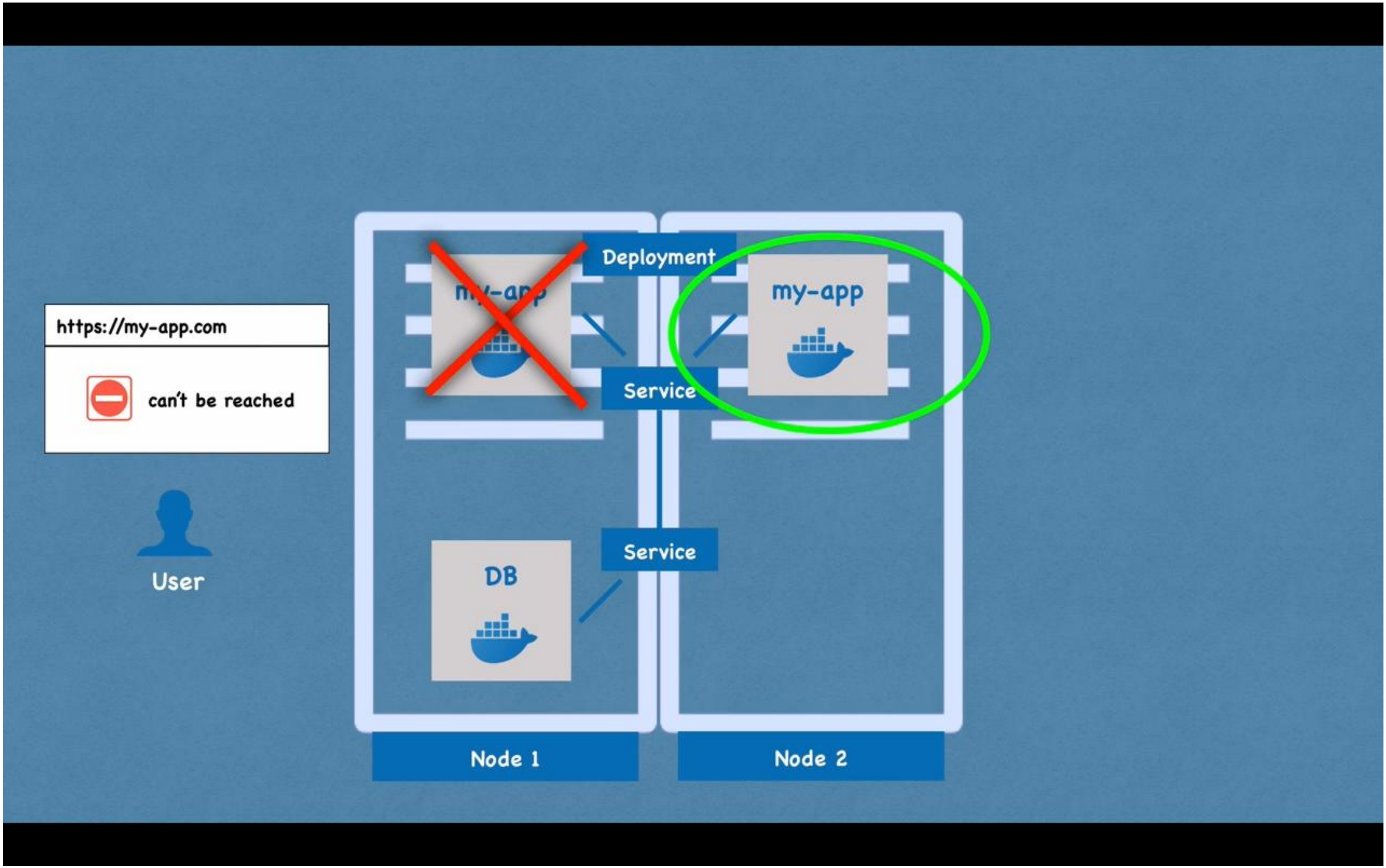

https://my-app.com
can't be reached
User
my-app
my-app
Service
Service
DB
Node 1
Node 2
Service is a load balancer!

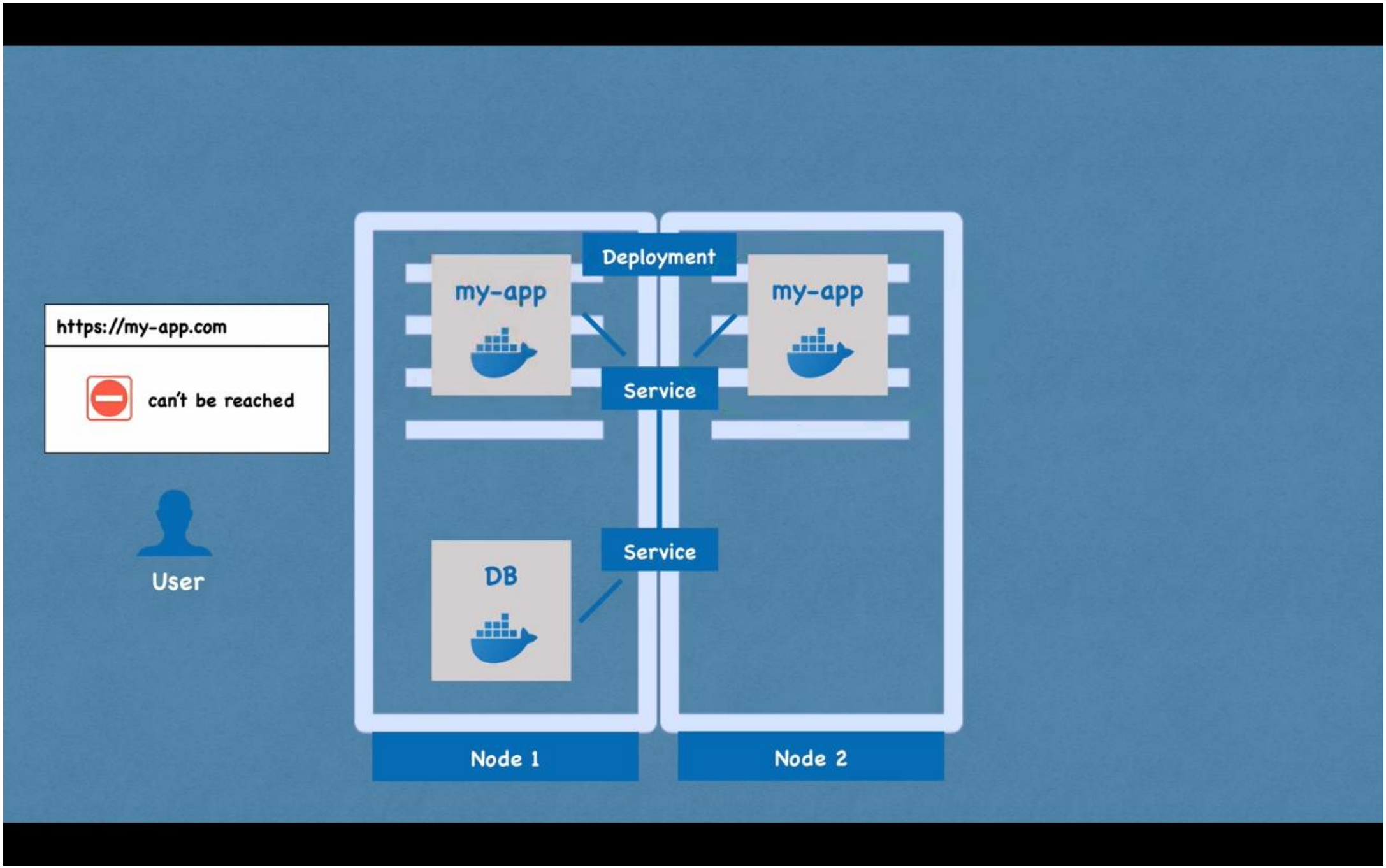

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
Node 1
Node 2

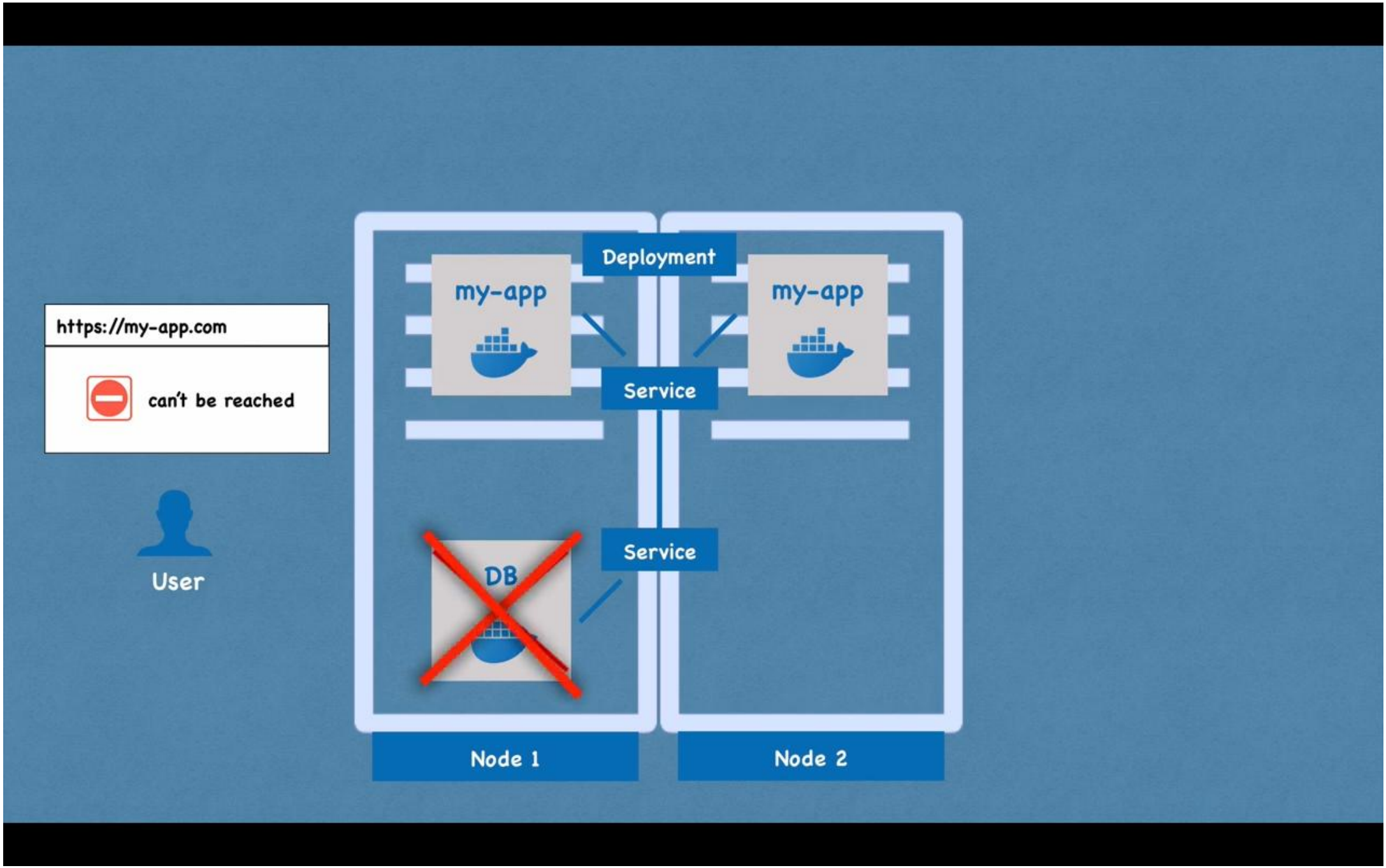

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
Node 1
Node 2
Deployment:
- blueprint for my-app Pods

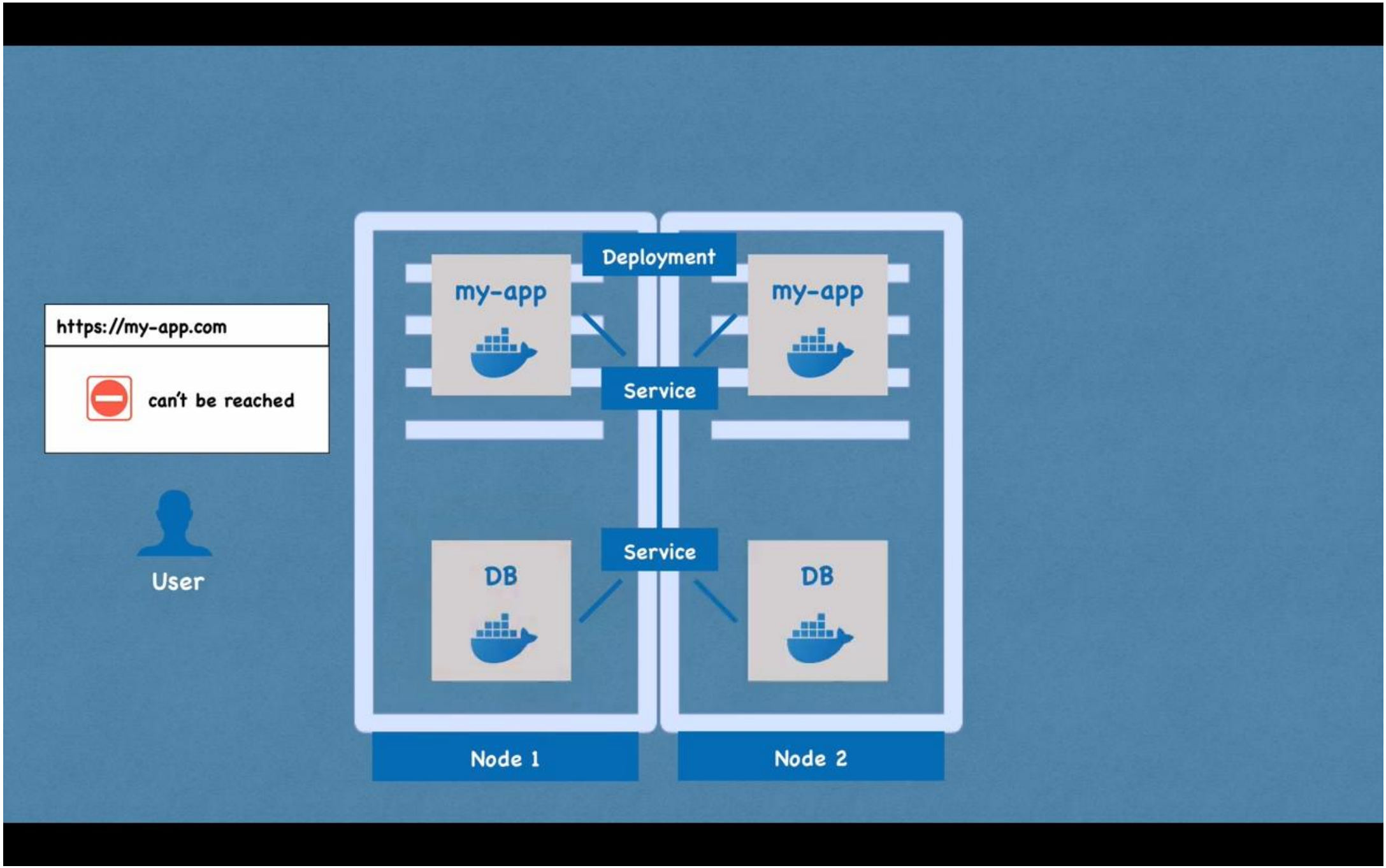

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
Node 1
Node 2
Deployment:
- blueprint for my-app Pods
- abstraction of Pods


https://my-app.com
can't be reached
User
Deployment
my-app
Service
my-app
Service
DB
Node 1
Node 2

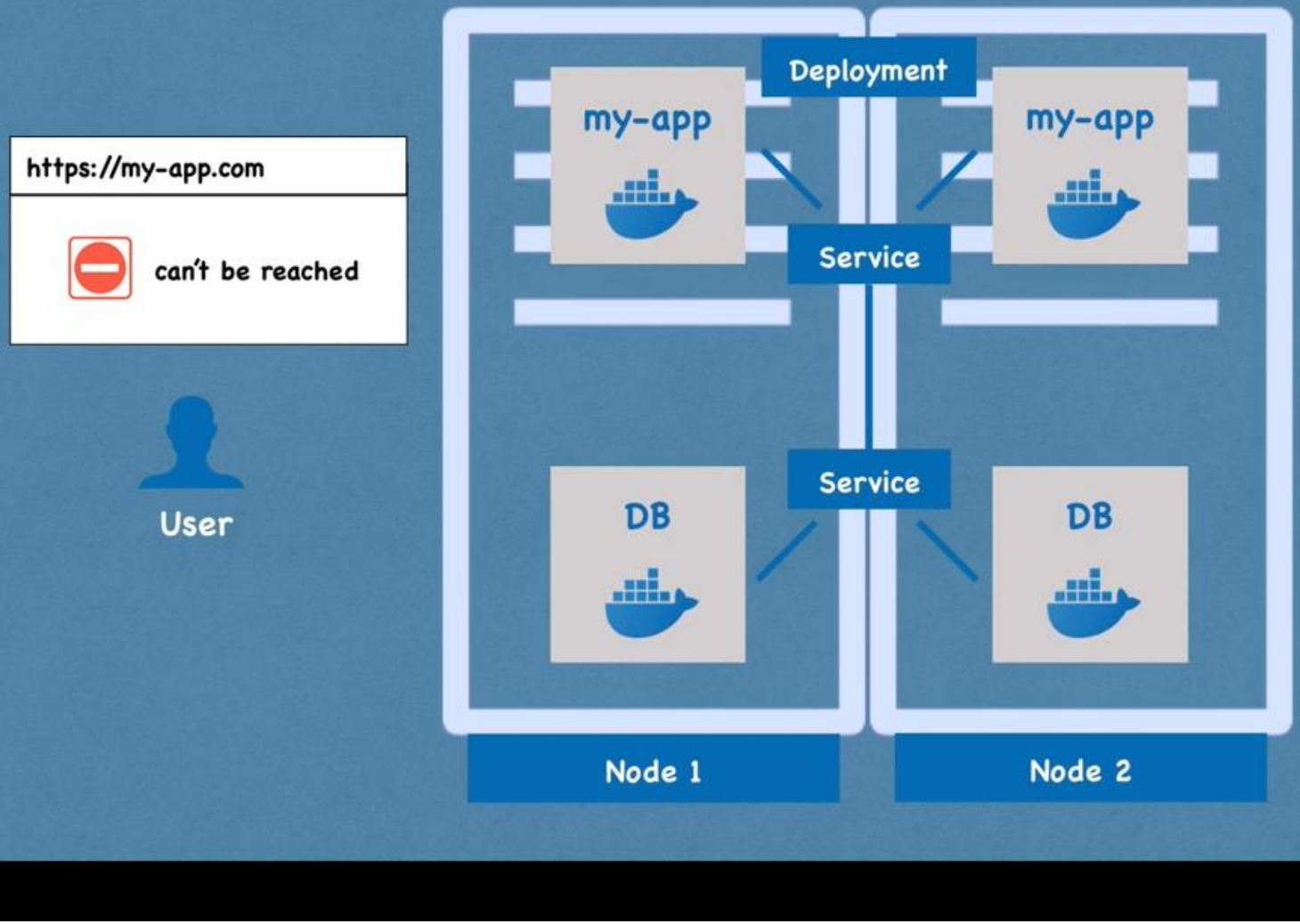

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
Node 1
Node 2

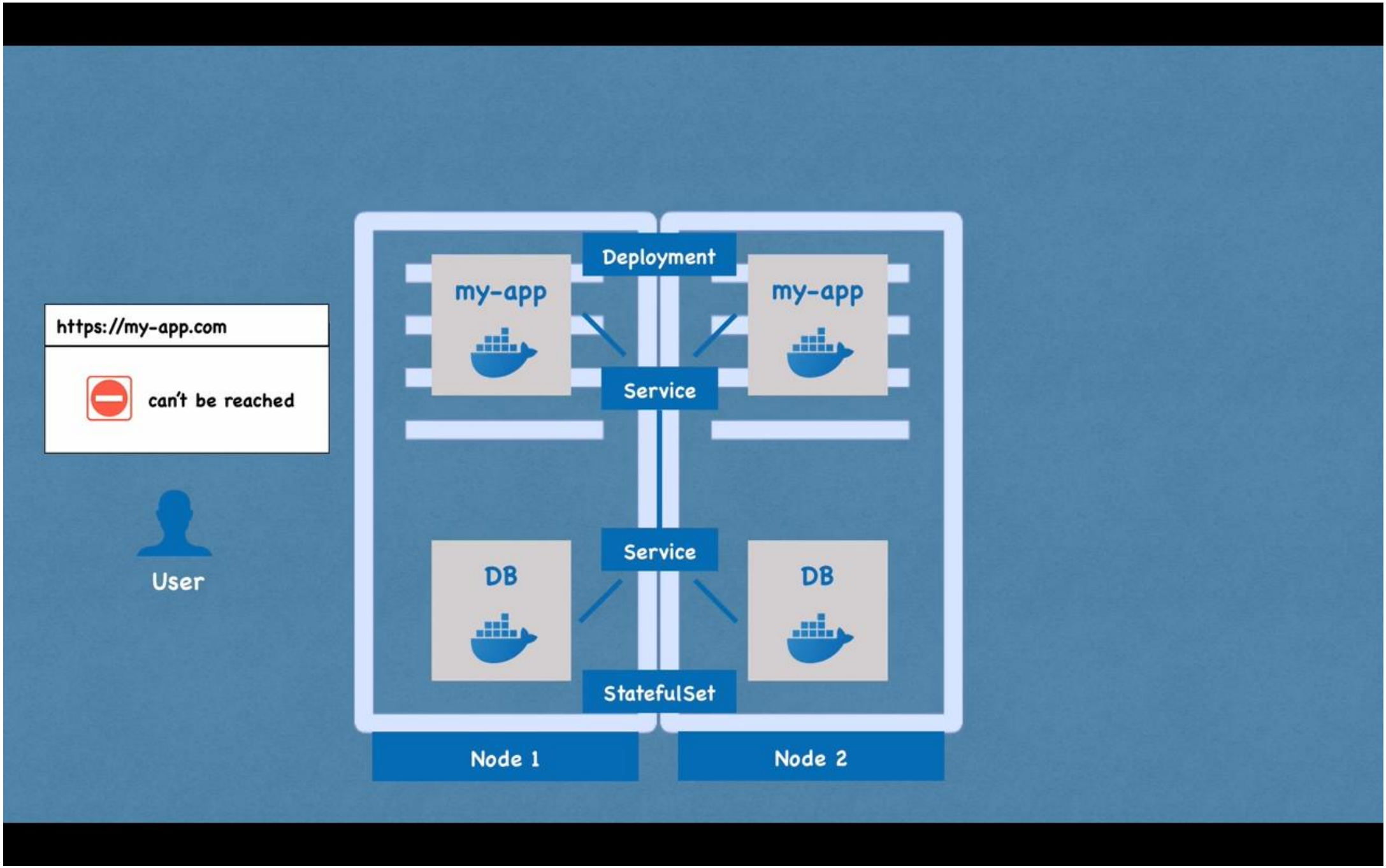

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
Node 1
Node 2

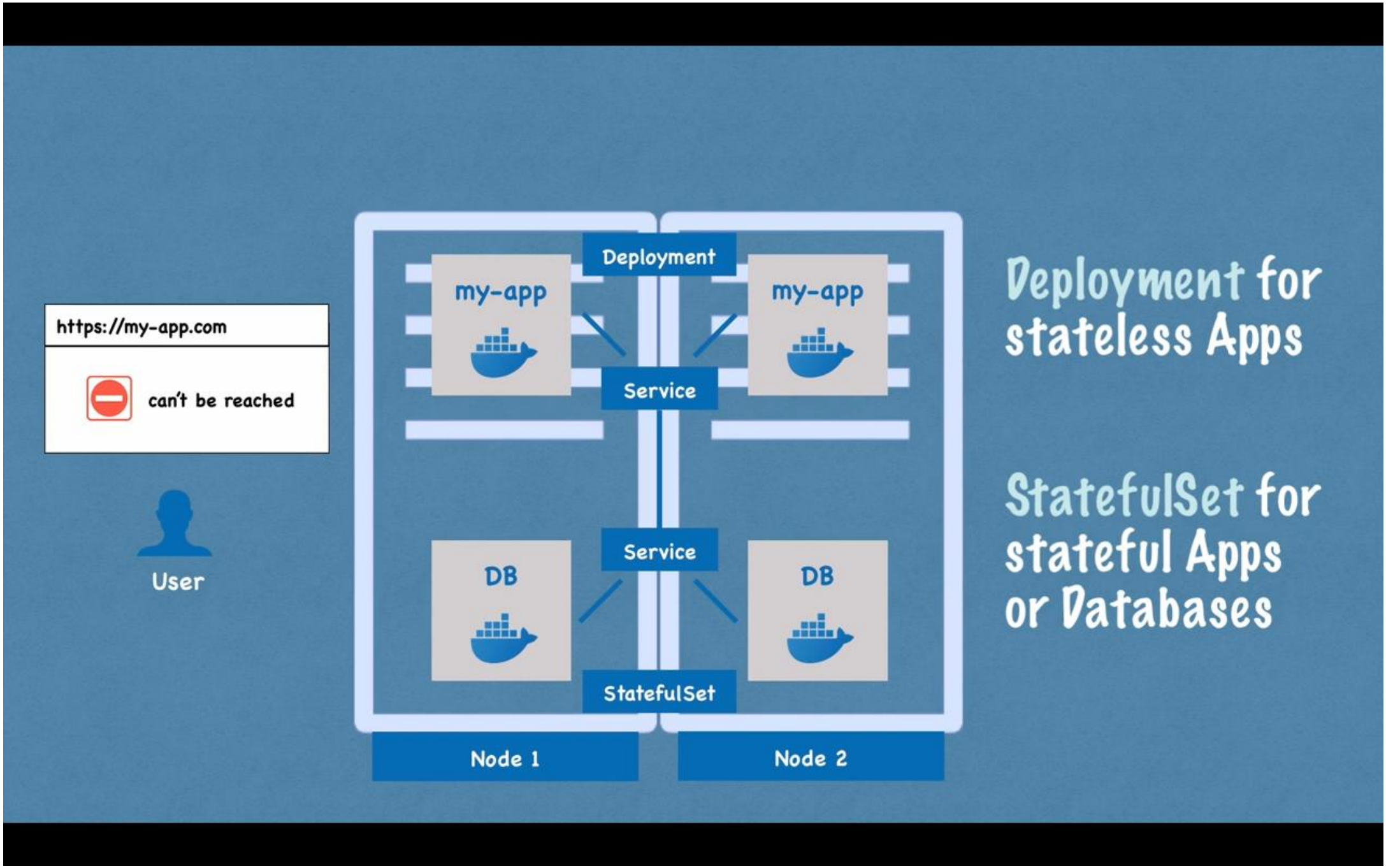

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
Node 1
Node 2


https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
DB
Node 1
Node 2

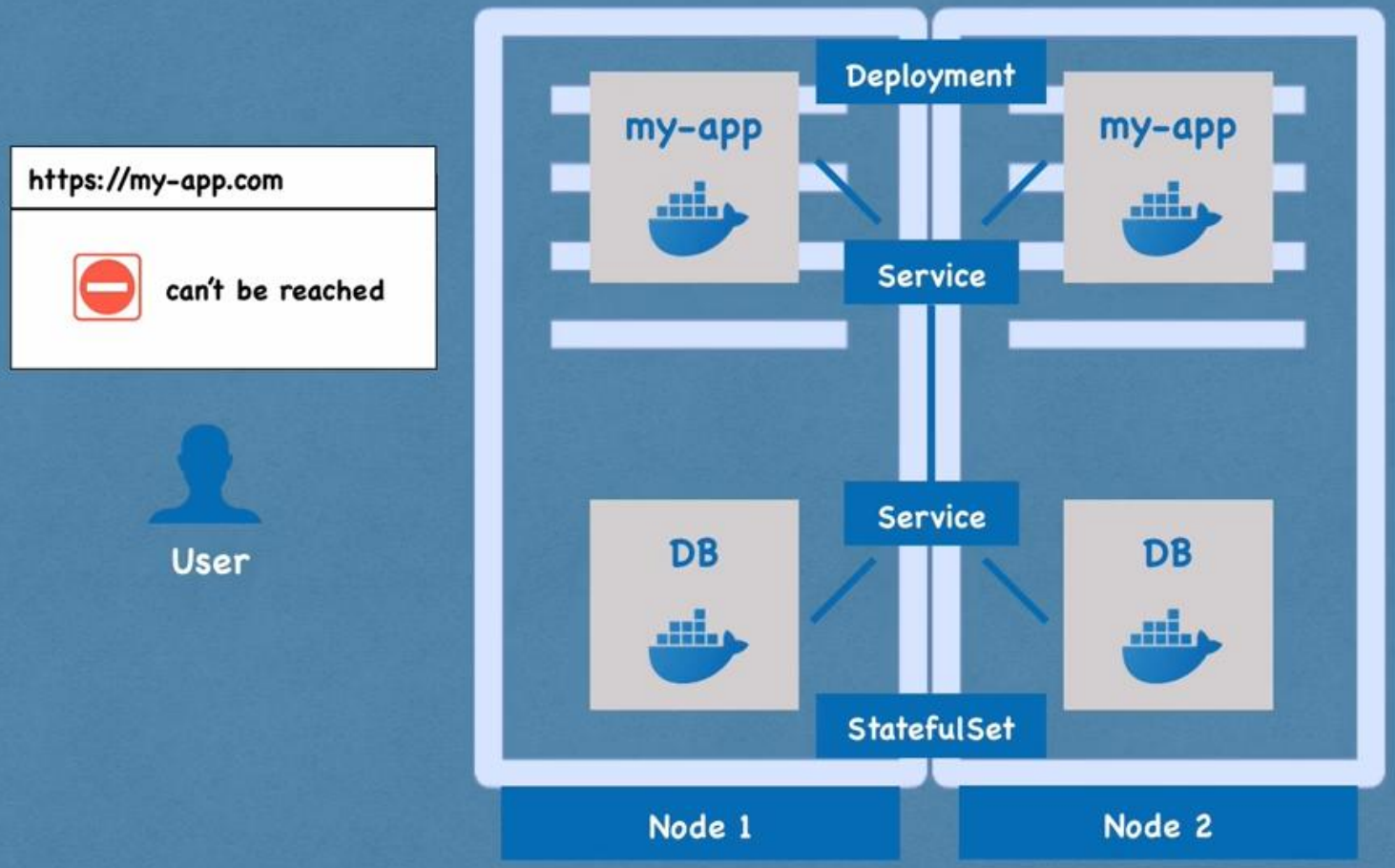
DB can't be replicated via Deployment!

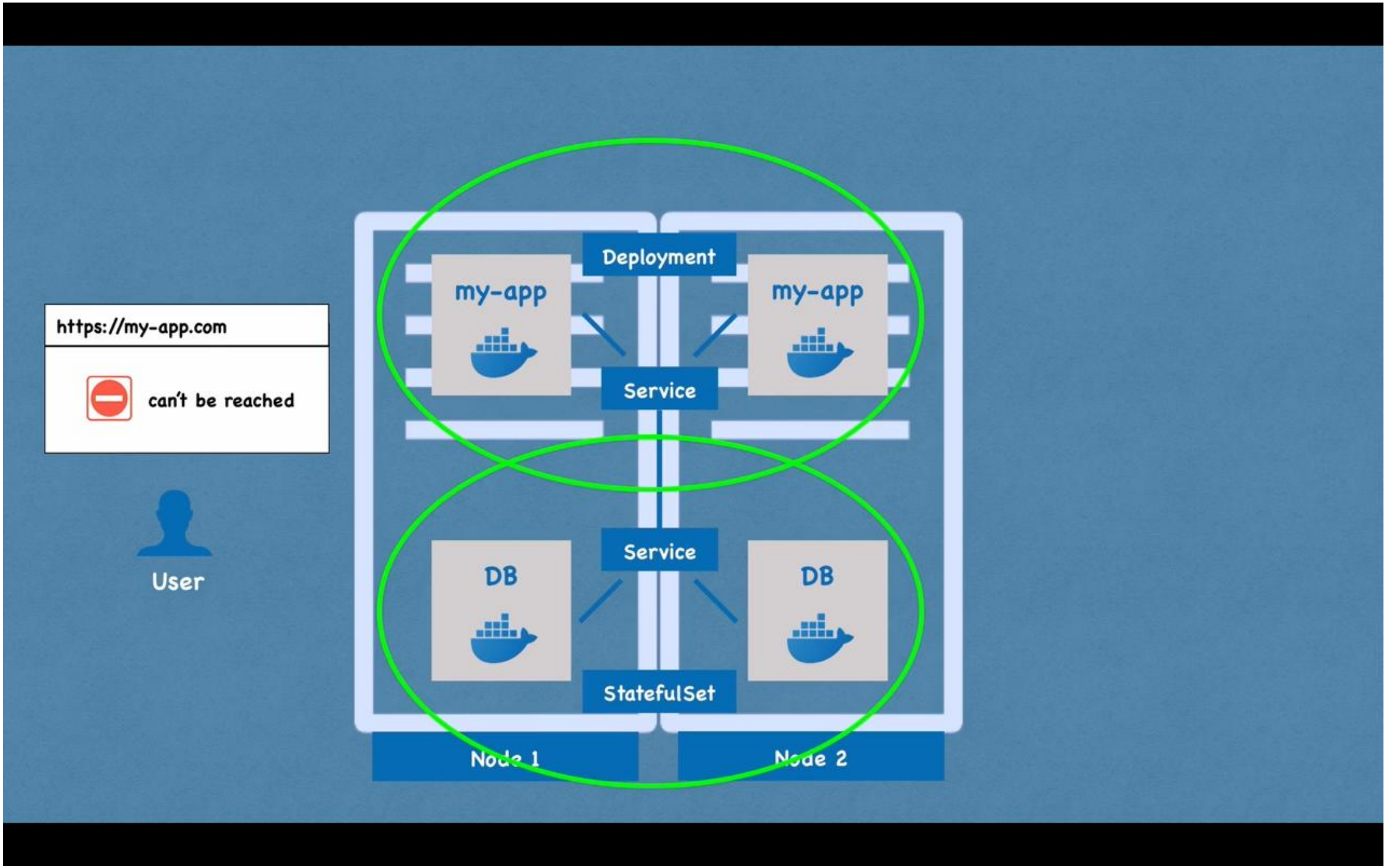


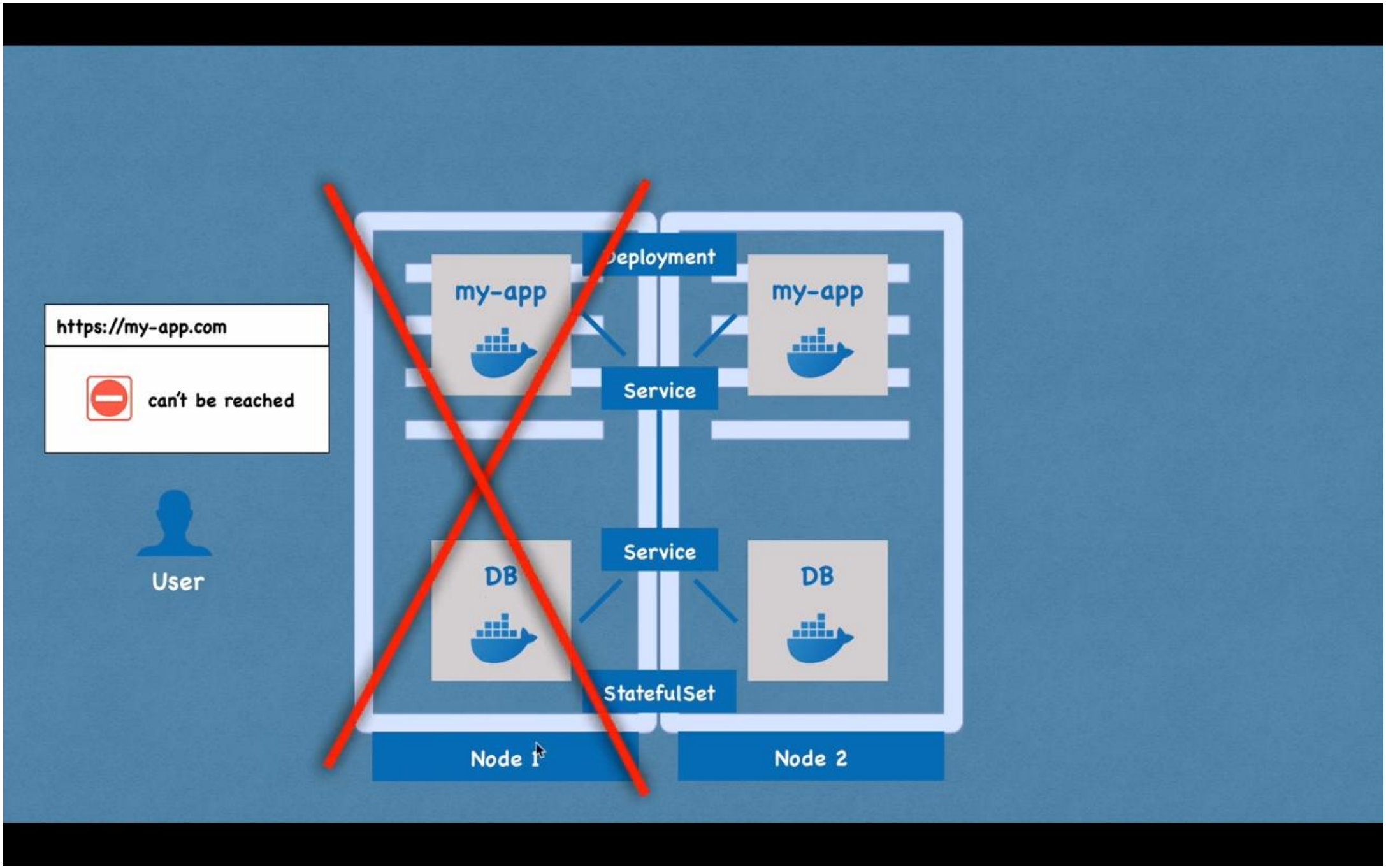

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
DB
StatefulSet
Node 1
Node 2

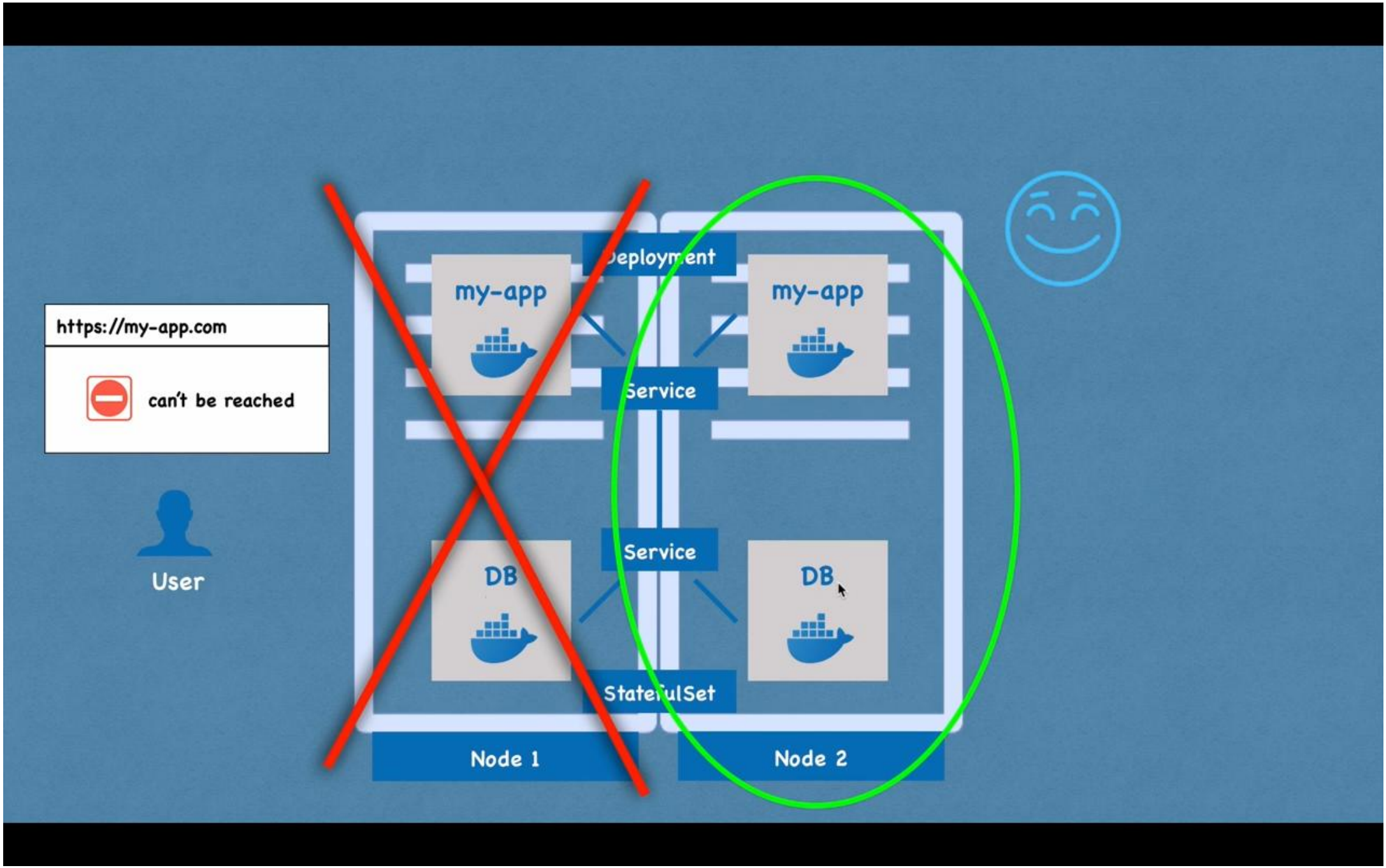

https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
DB
StatefulSet
Node 1
Node 2
Deployment for stateless Apps
StatefulSet for stateful Apps or Databases

DB are often hosted outside of the K8s cluster




https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
DB
StatefulSet
Node 1
Node 2


https://my-app.com
can't be reached
User
Deployment
my-app
my-app
Service
Service
DB
DB
StatefulSet
Node 1
Node 2


https://my-app.com
can't be reached
User
my-app
my-app
Service
Service
DB
DB
Node 1
Node 2

Main Kubernetes Components summarized

Pod

Service

Ingress

Volumes

Deployment

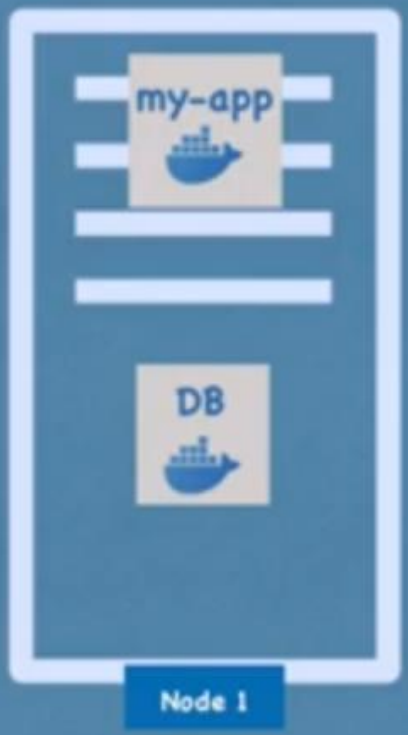
ConfigMap

StatefulSet

Secrets

Node processes

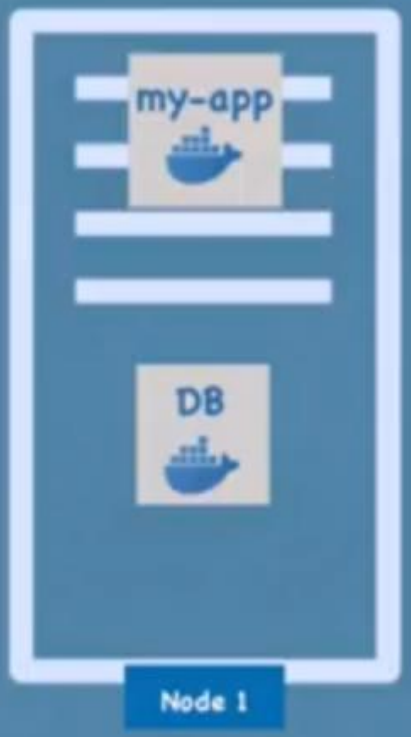
Worker machine in K8s cluster



Worker machine in K8s cluster

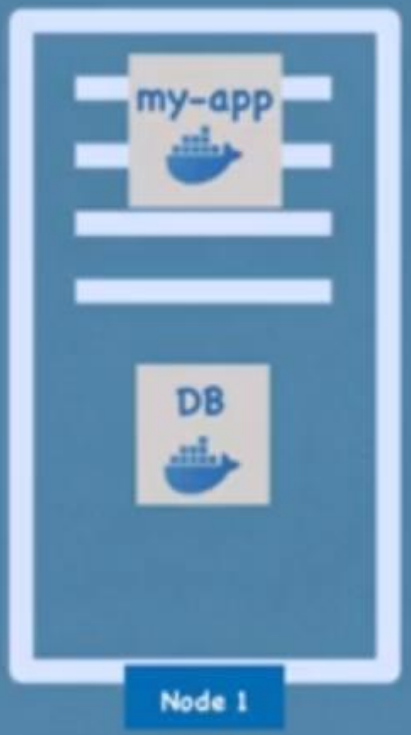
- each Node has multiple Pods on it

Worker machine in K8s cluster



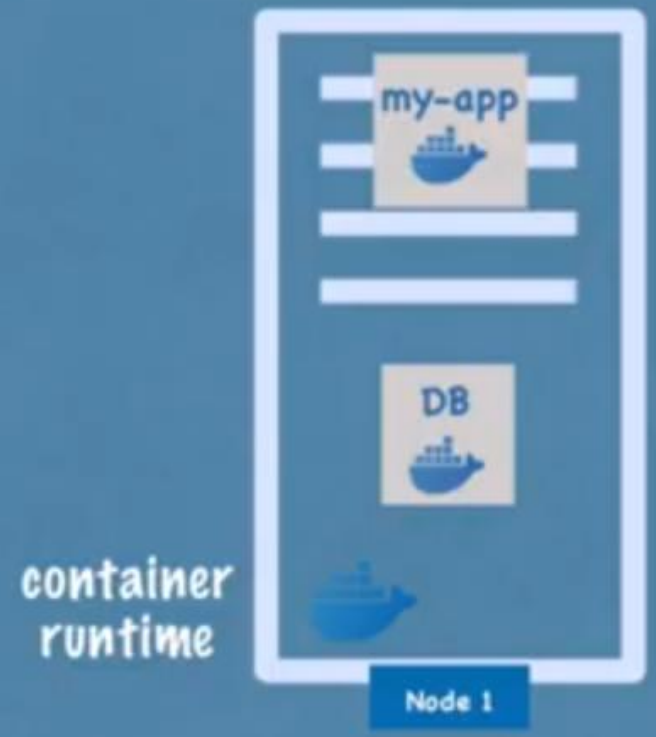
- each Node has multiple Pods on it
- 3 processes must be installed on every Node

Worker machine in K8s cluster

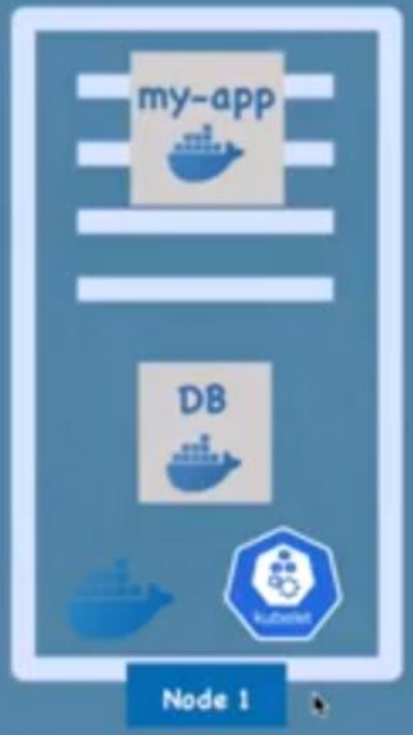


- each Node has multiple Pods on it
- 3 processes must be installed on every Node
- Worker Nodes do the actual work

Worker machine in K8s cluster



Worker machine in K8s cluster



container runtime

Kubelet

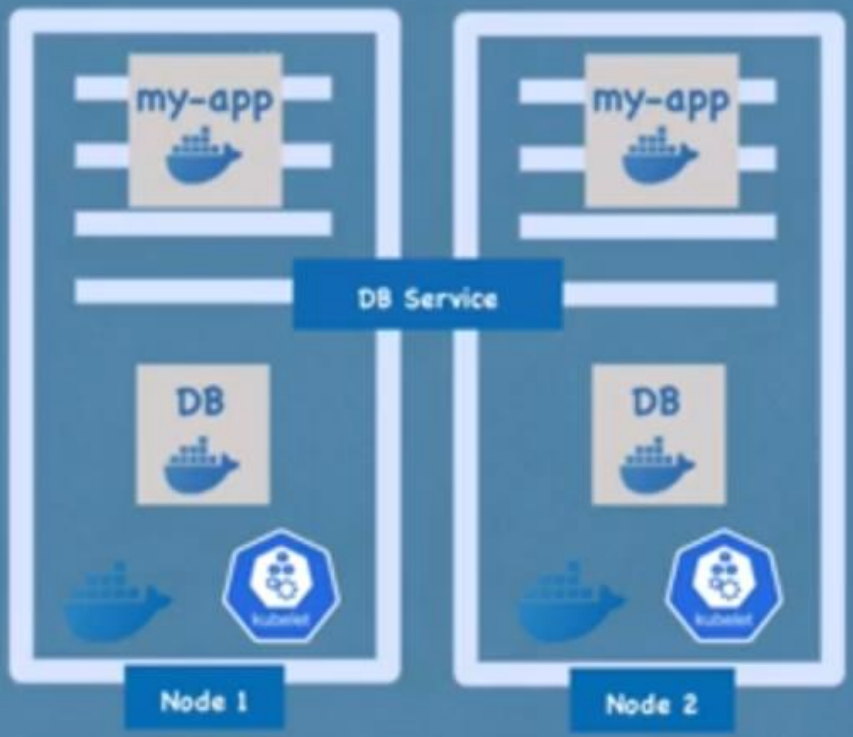
Kubelet interacts with both - the container and node

Kubelet starts the pod with a container inside

Worker machine in K8s cluster

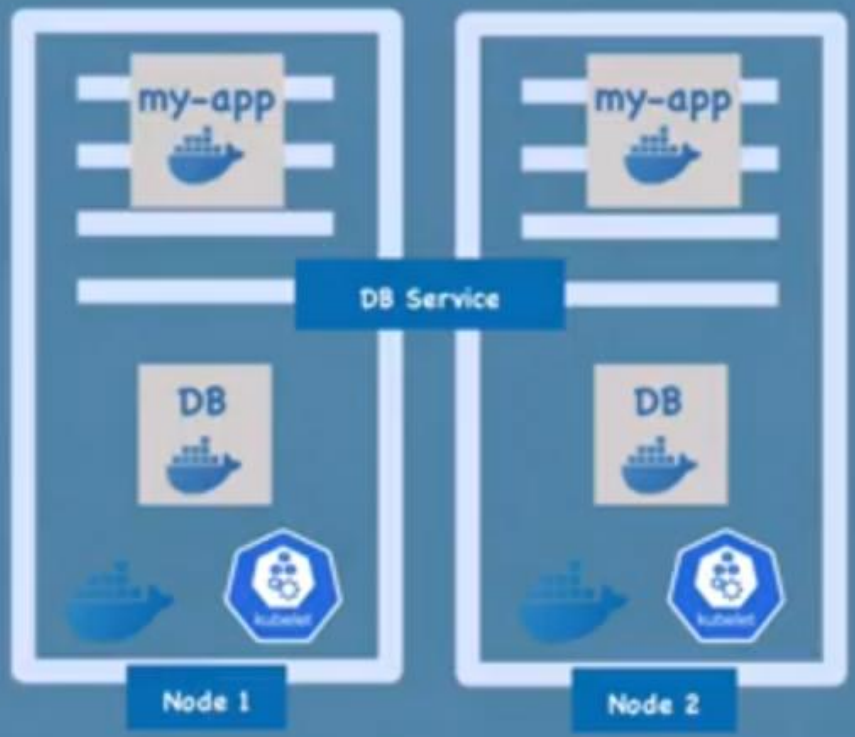
Communication via Services

container runtime



Kubelet

Worker machine in K8s cluster



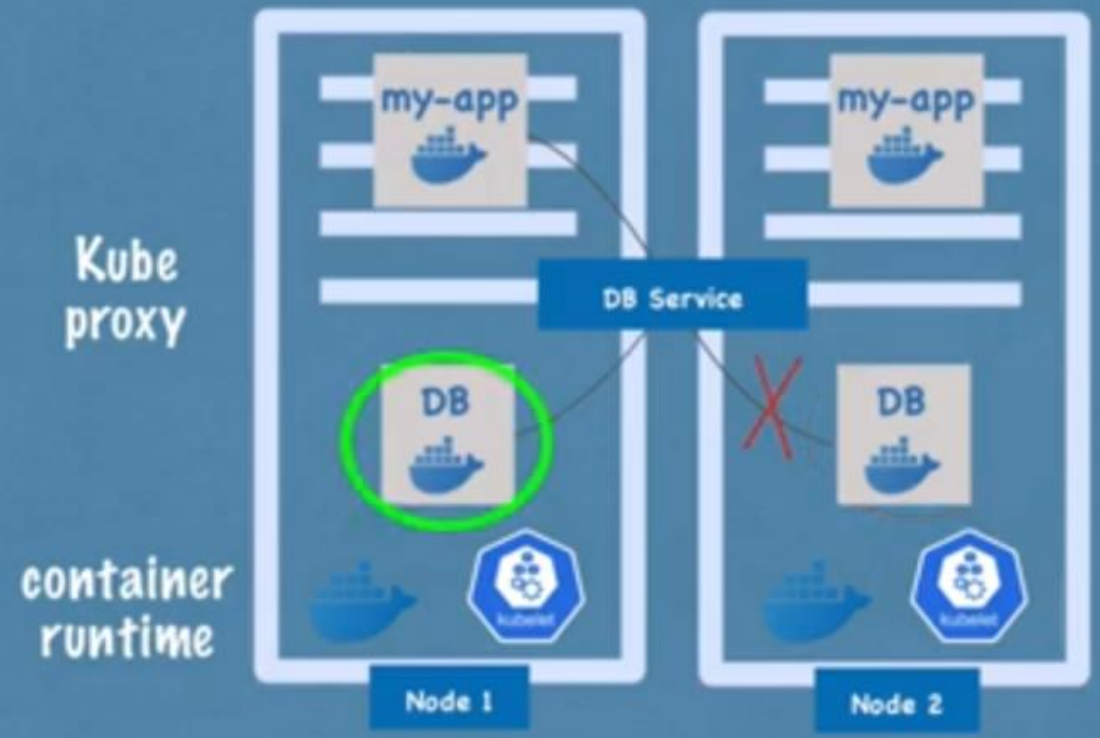
Kube proxy

container runtime

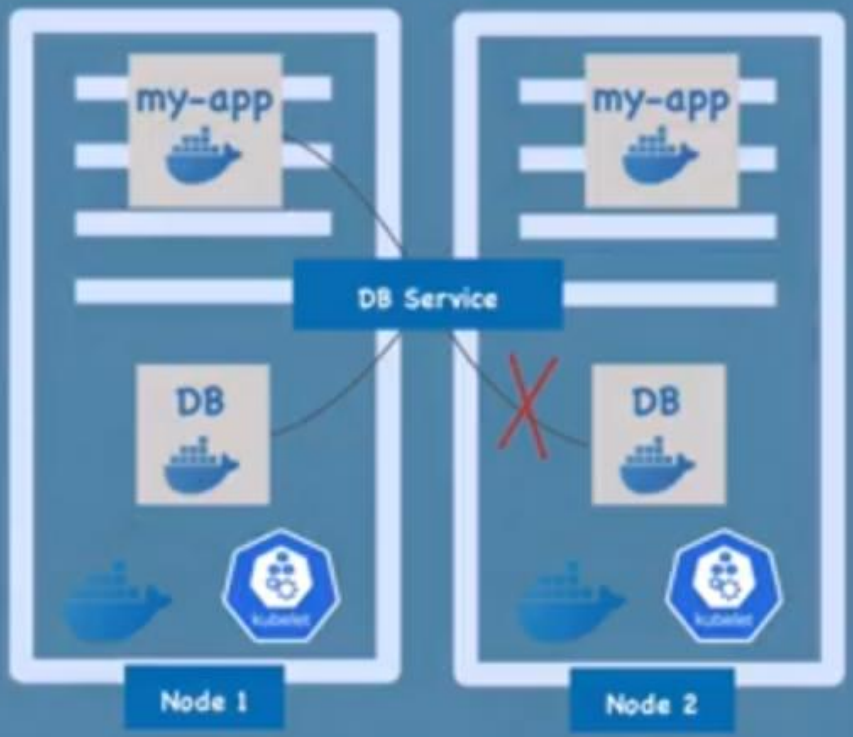
Kubelet

Kube Proxy forwards the requests

Worker machine in K8s cluster



Worker machine in K8s cluster



3 node processes:

1. Kubelet
2. Kube Proxy
3. Container runtime

So, how do you interact with this cluster?

How to:

-> schedule pod?

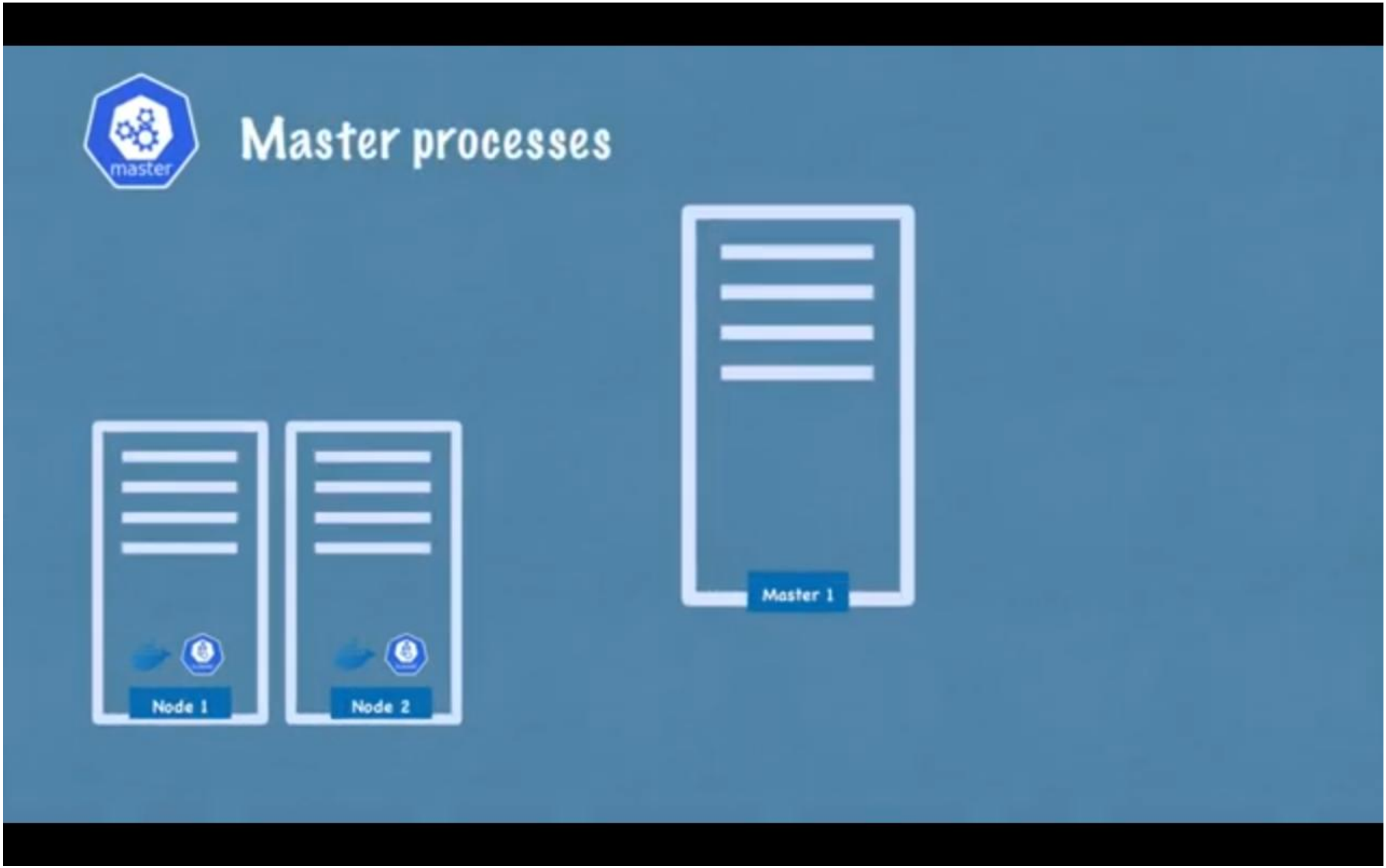
-> monitor?

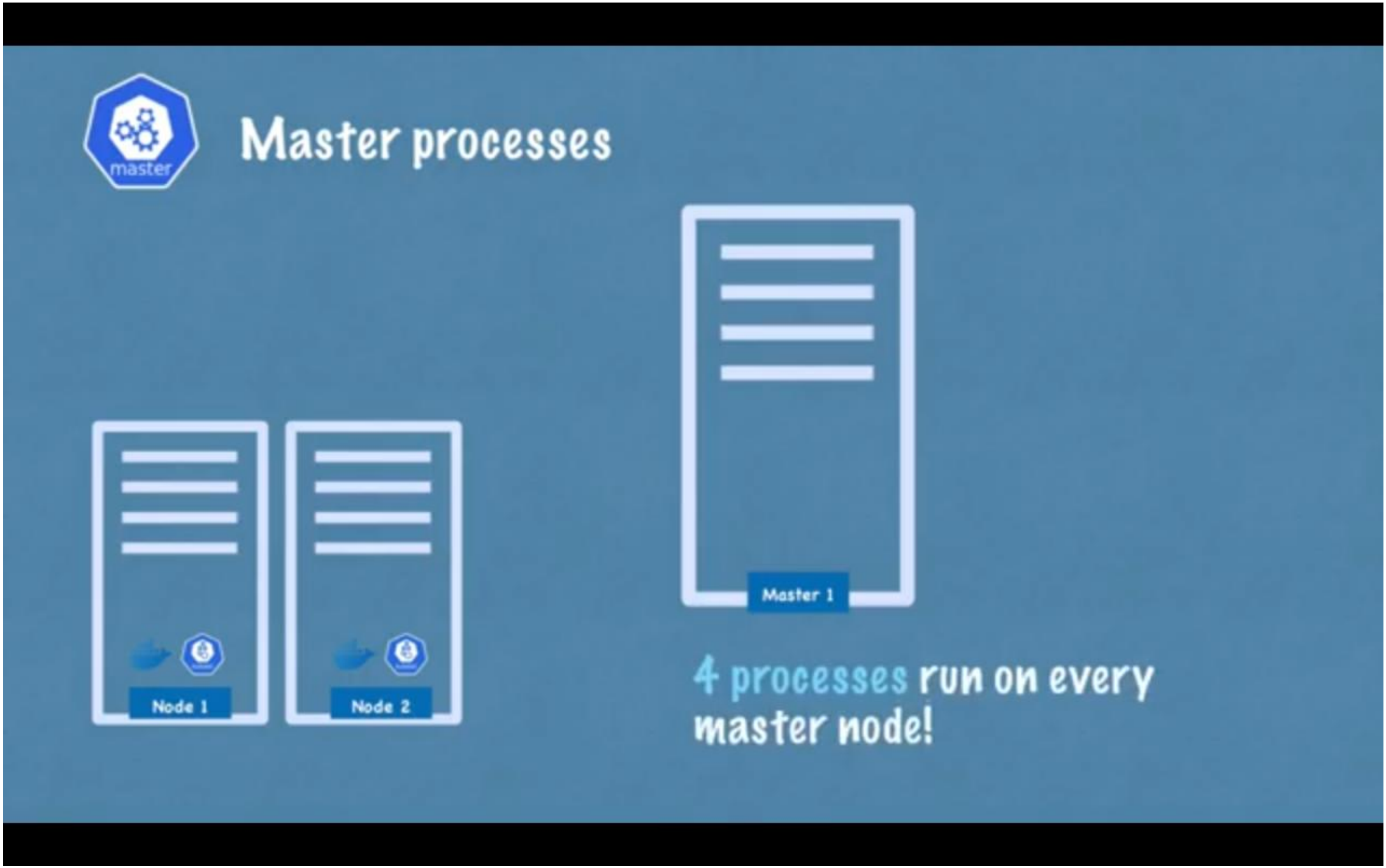
-> re-schedule/re-start pod?

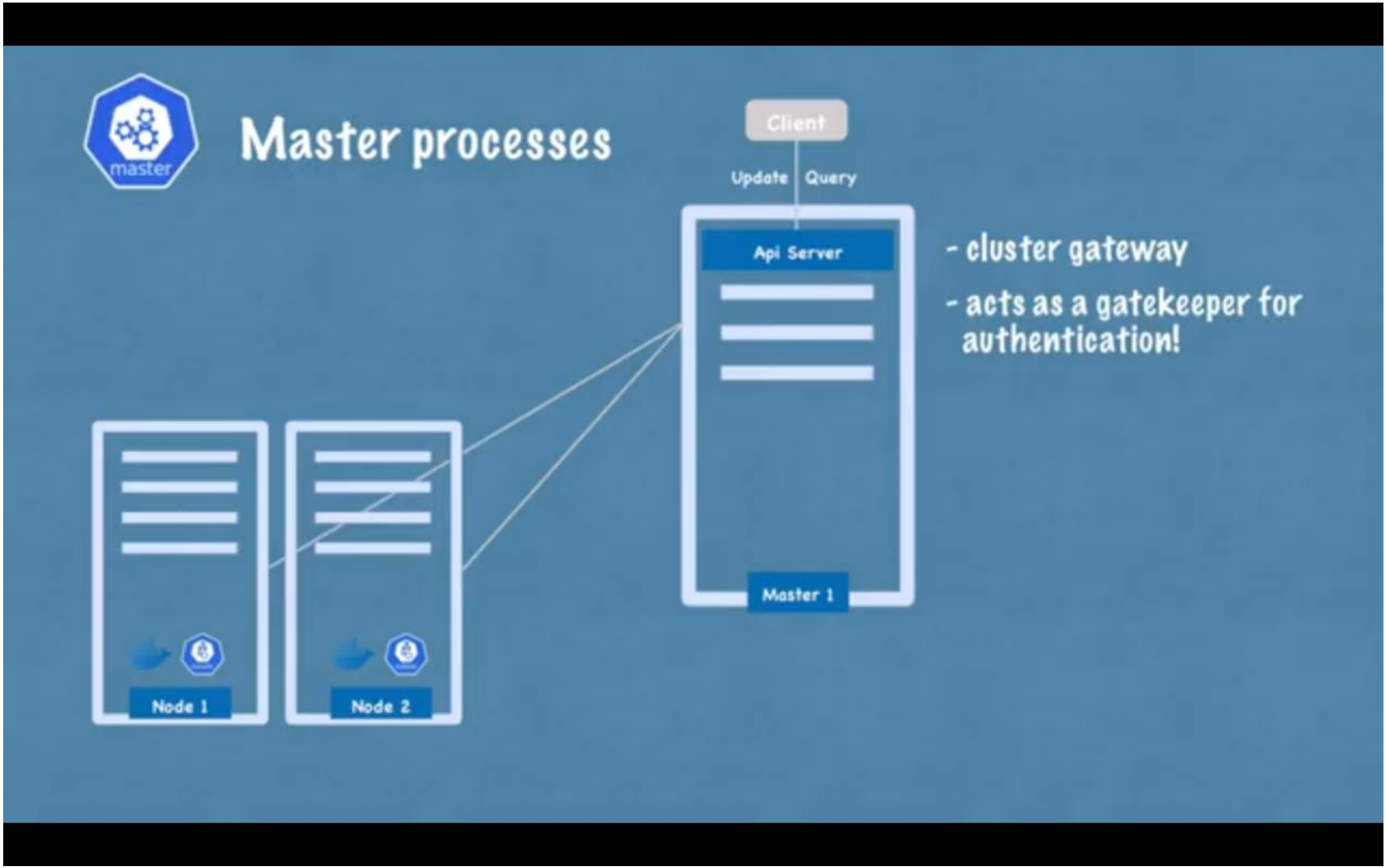
-> join a new node?

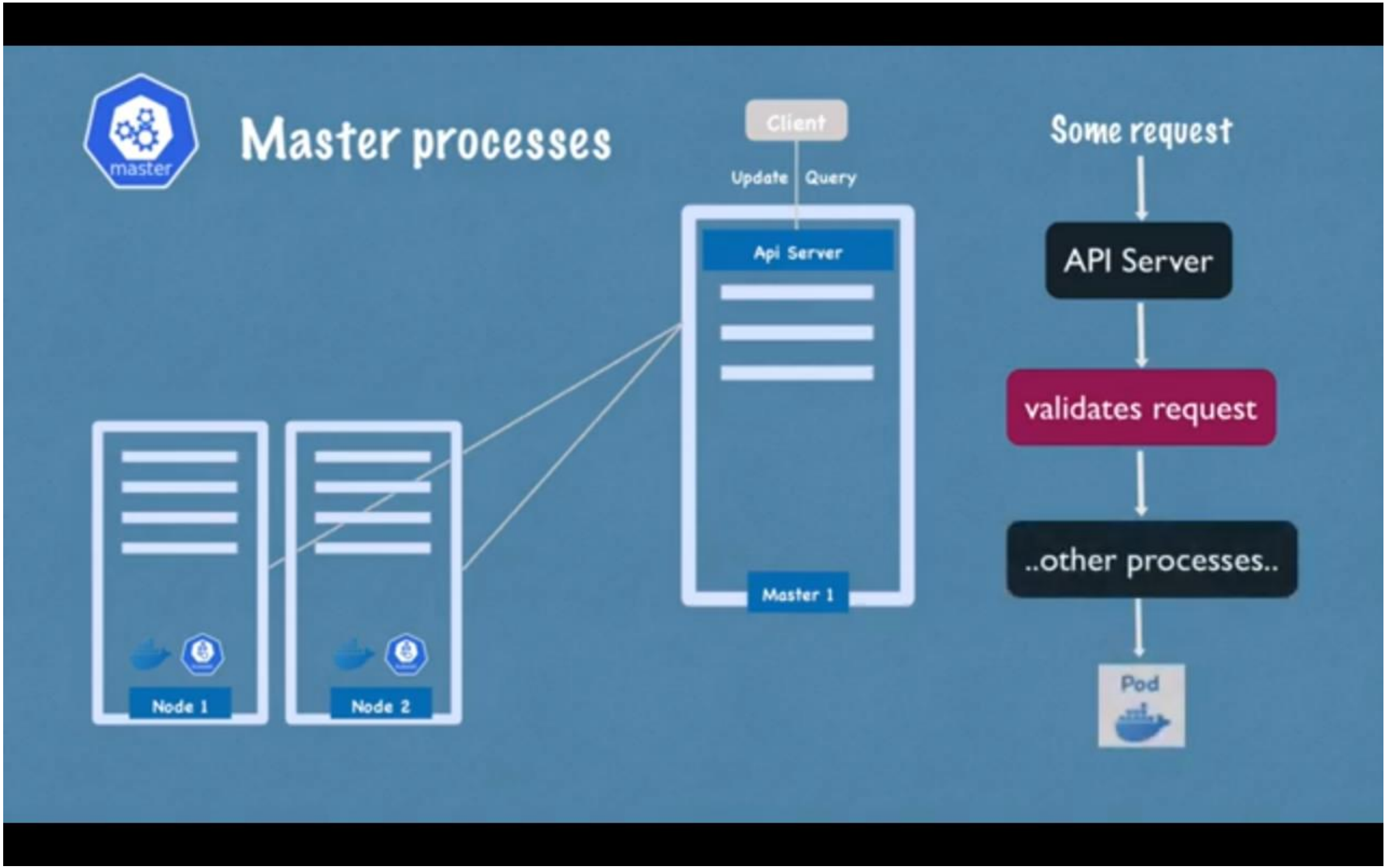
Managing processes are done by
Master Nodes

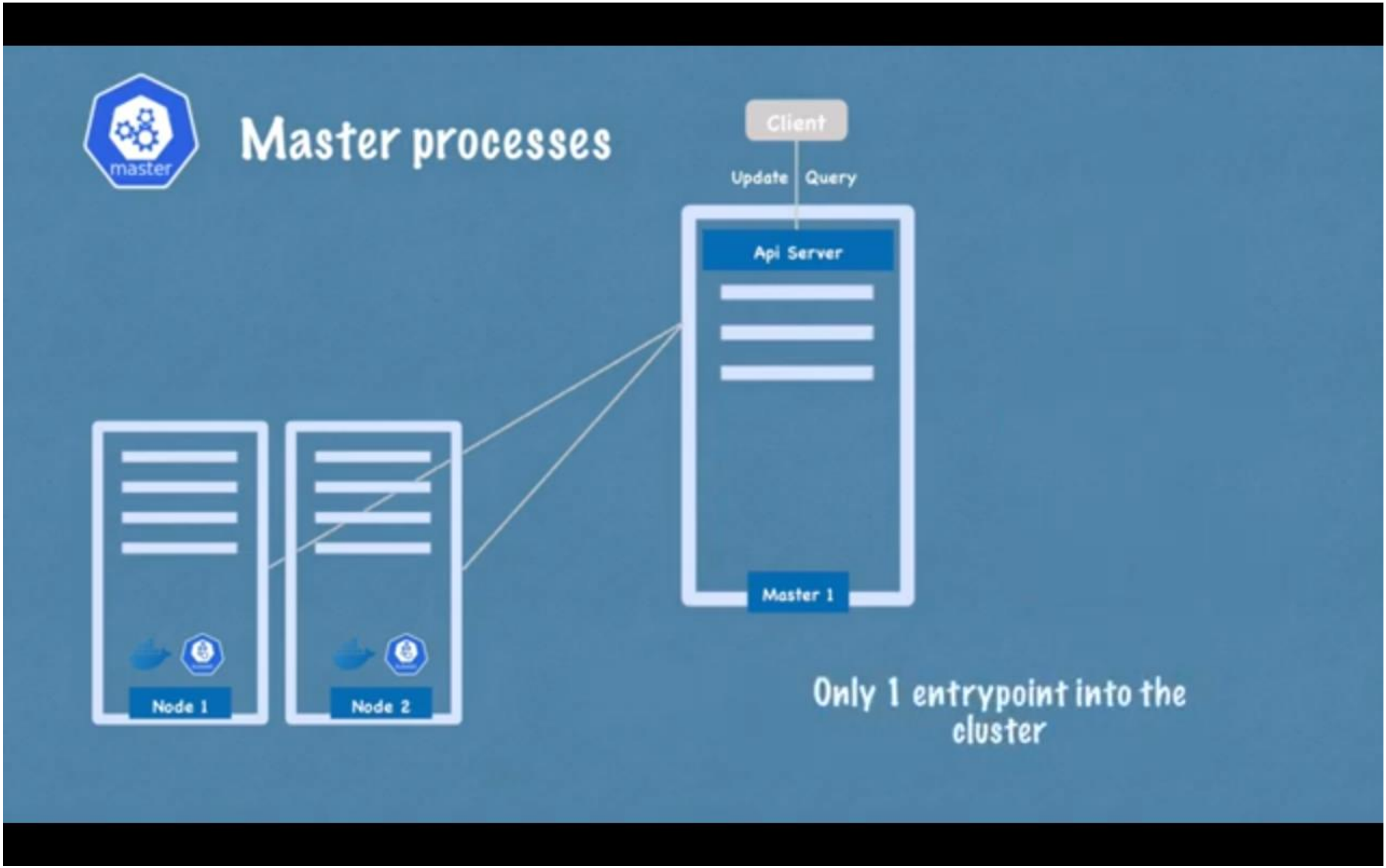
Master processes

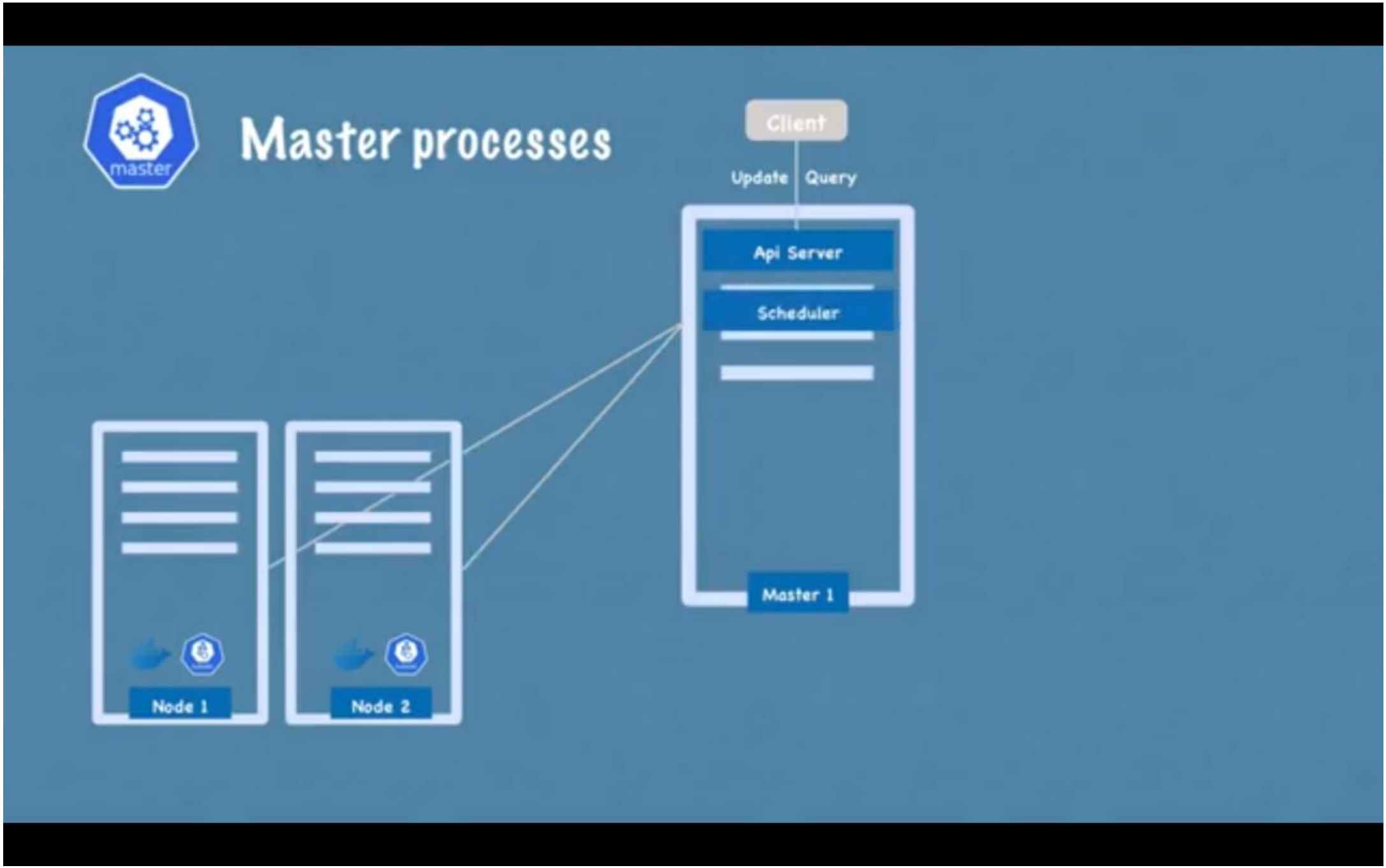

master
Master processes
Node 1
Node 2
Master 1

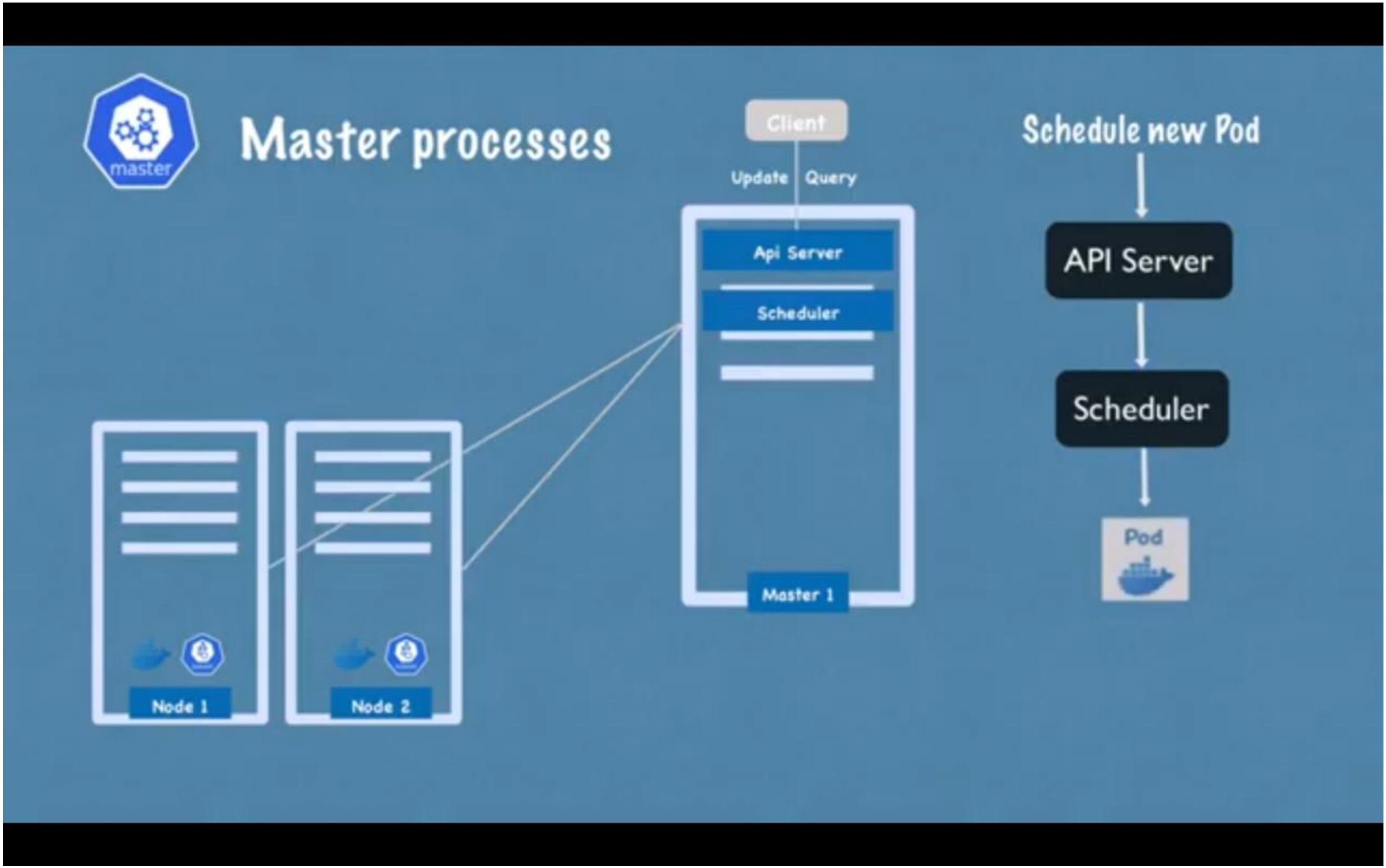

master
Master processes
Node 1
Node 2
Master 1
4 processes run on every master node!

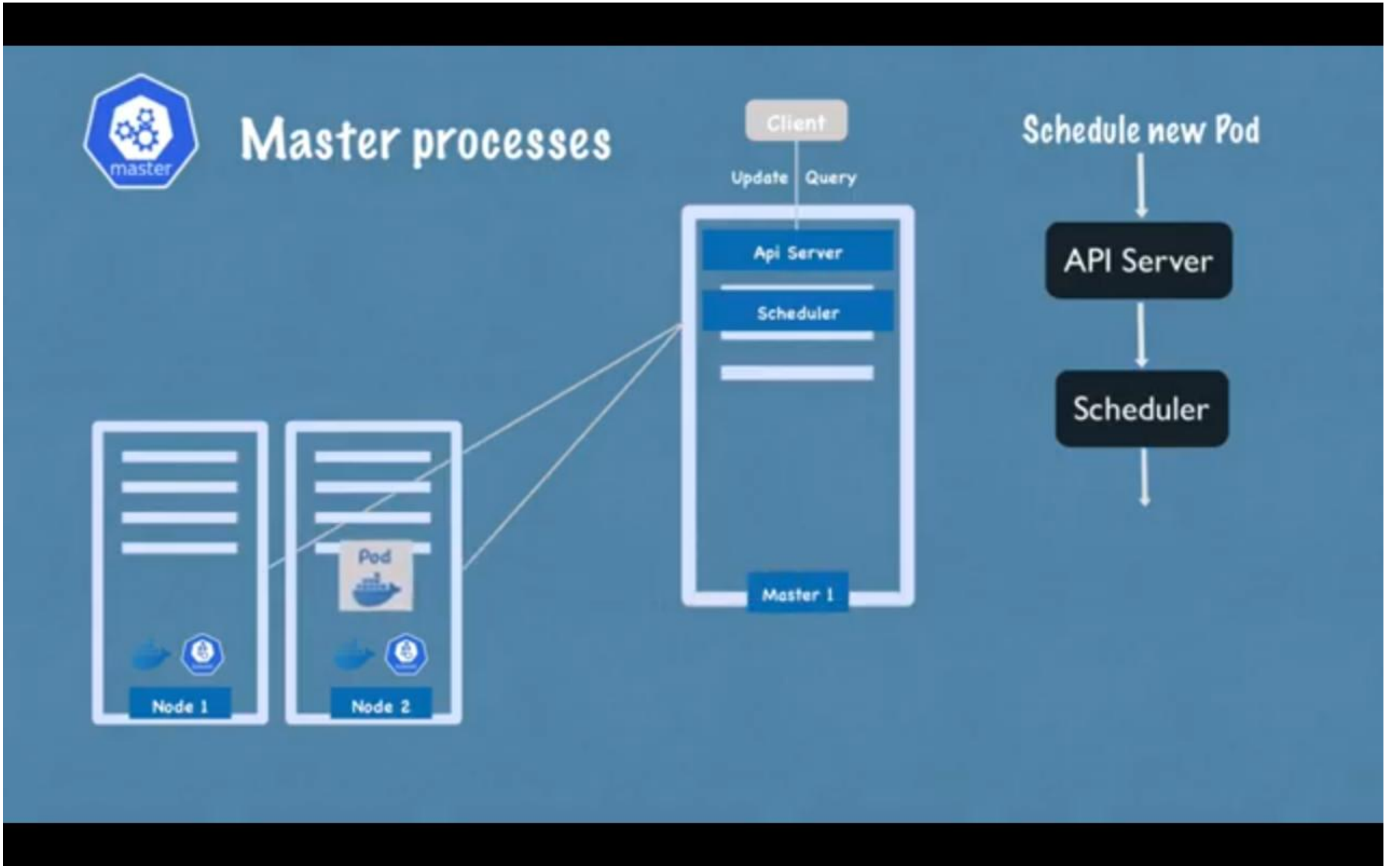

master
Master processes
Client
Update
Query
Api Server
- cluster gateway
- acts as a gatekeeper for authentication!
Master 1
Node 1
Node 2

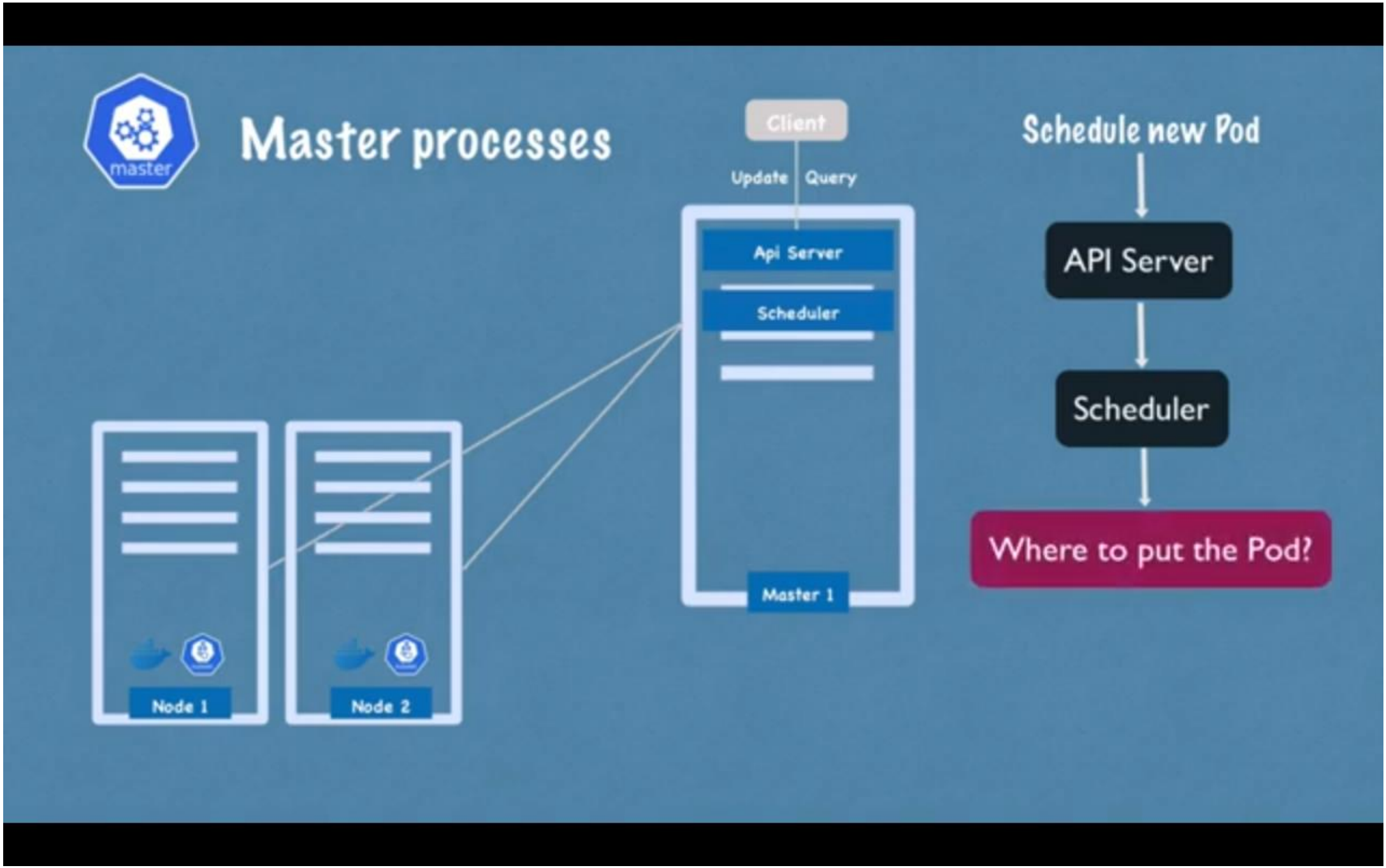

master
Master processes
Client
Update
Query
Api Server
Master 1
Node 1
Node 2
Some request
API Server
validates request
..other processes..
Pod

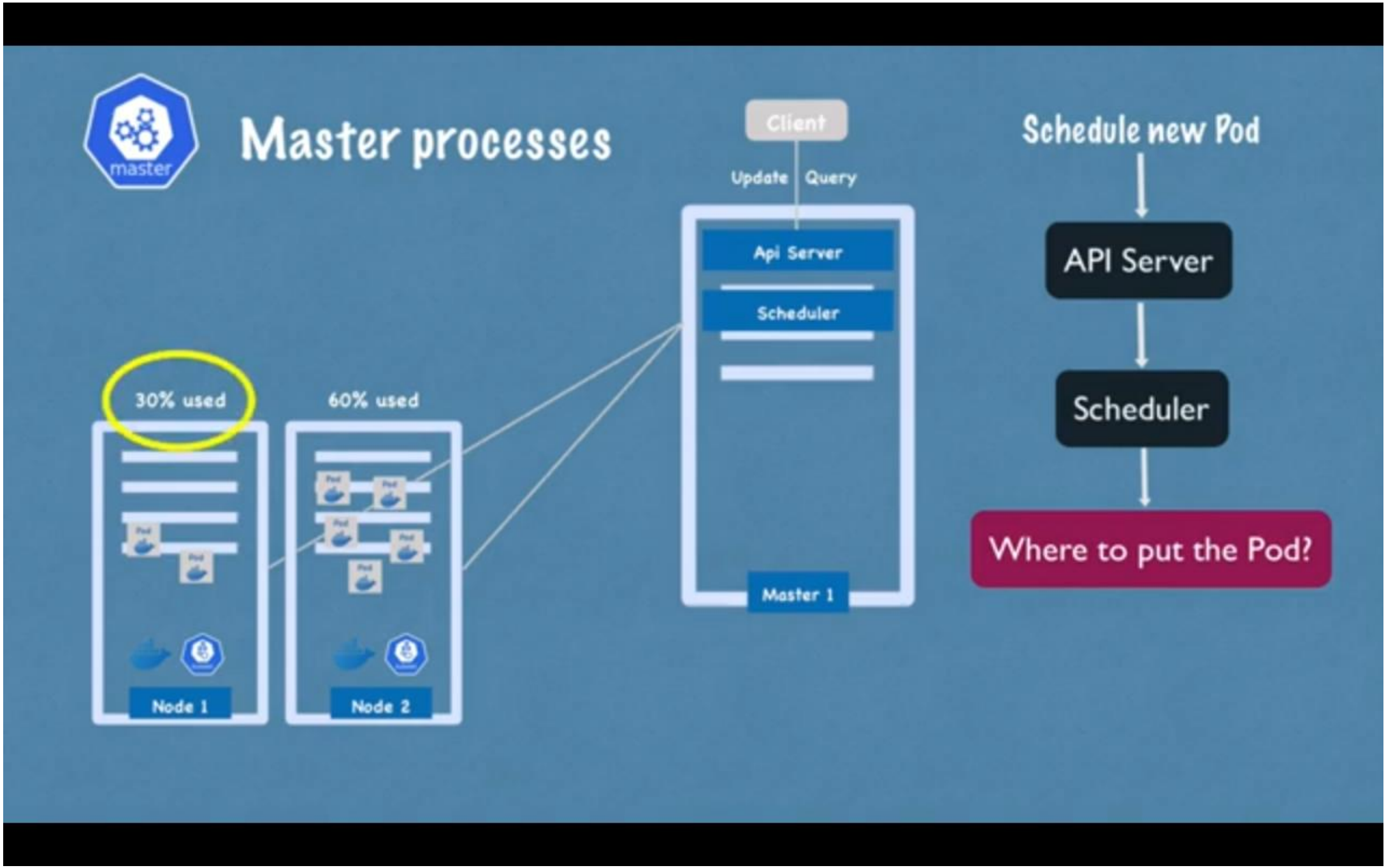

Master processes
master
Client
Update
Query
Api Server
Master 1
Node 1
Node 2
Only 1 entrypoint into the cluster

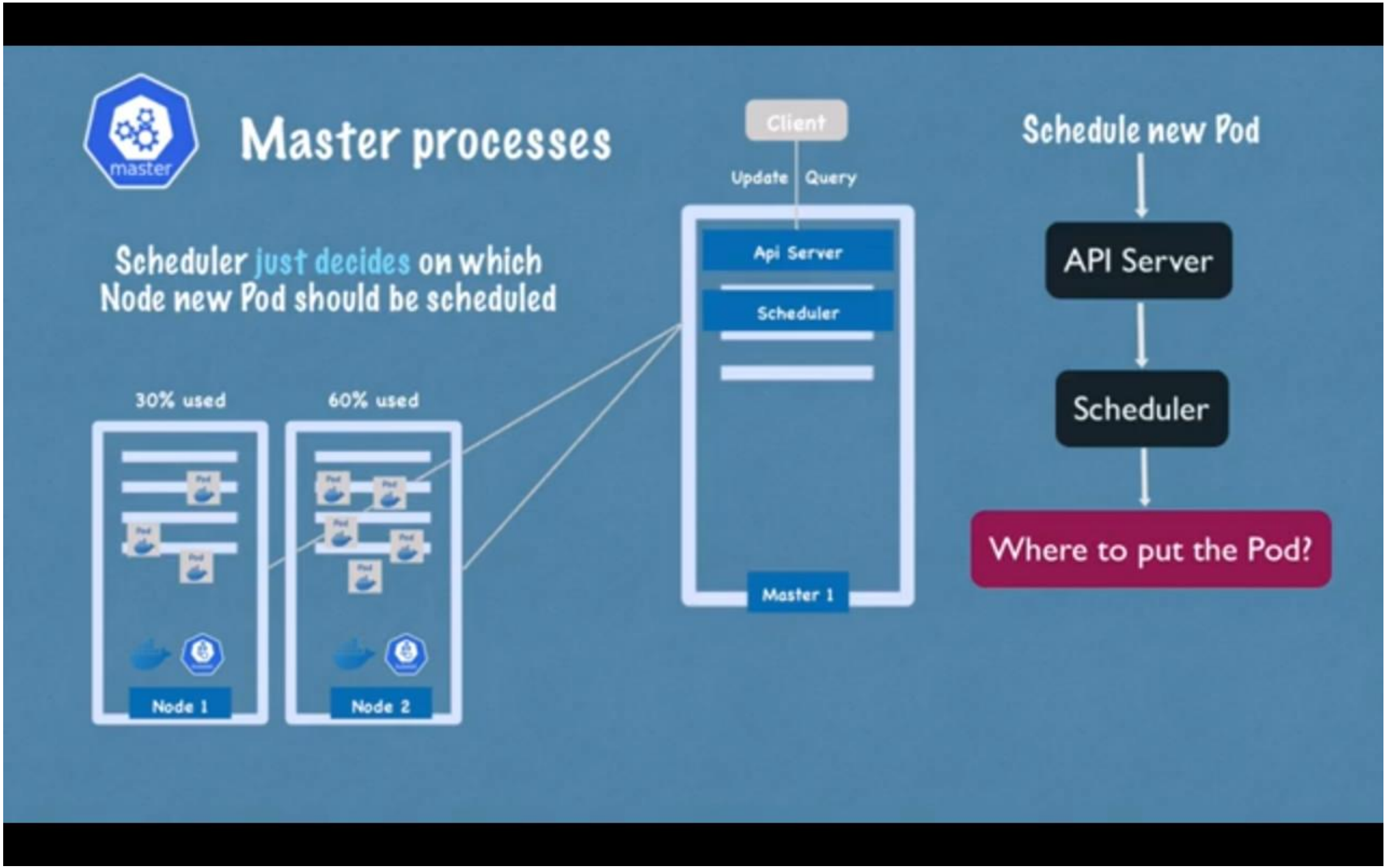

Master processes
master
Client
Update
Query
Api Server
Scheduler
Master 1
Node 1
Node 2

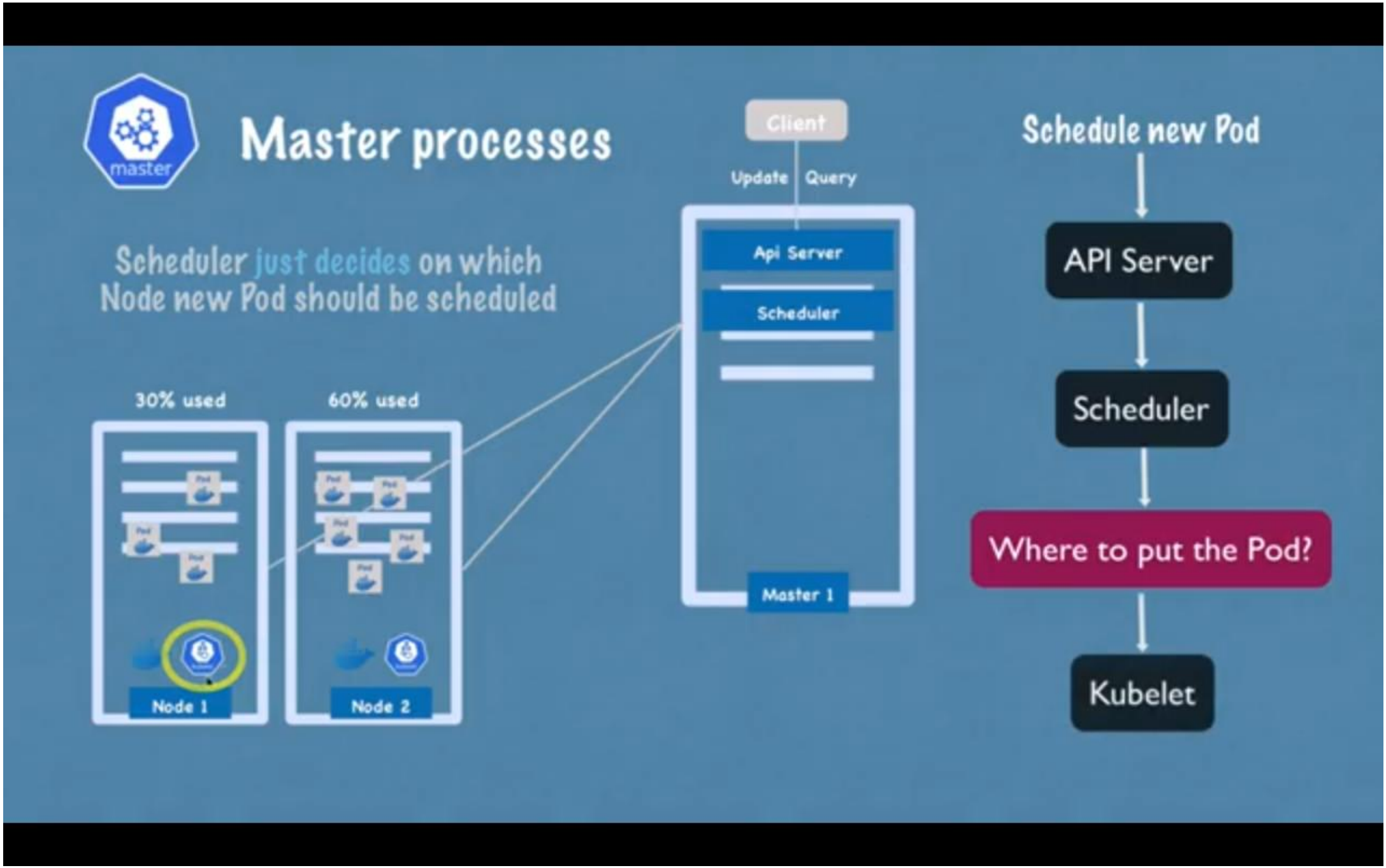

master
Master processes
Client
Update
Query
Api Server
Scheduler
Master 1
Node 1
Node 2
Schedule new Pod
API Server
Scheduler
Pod

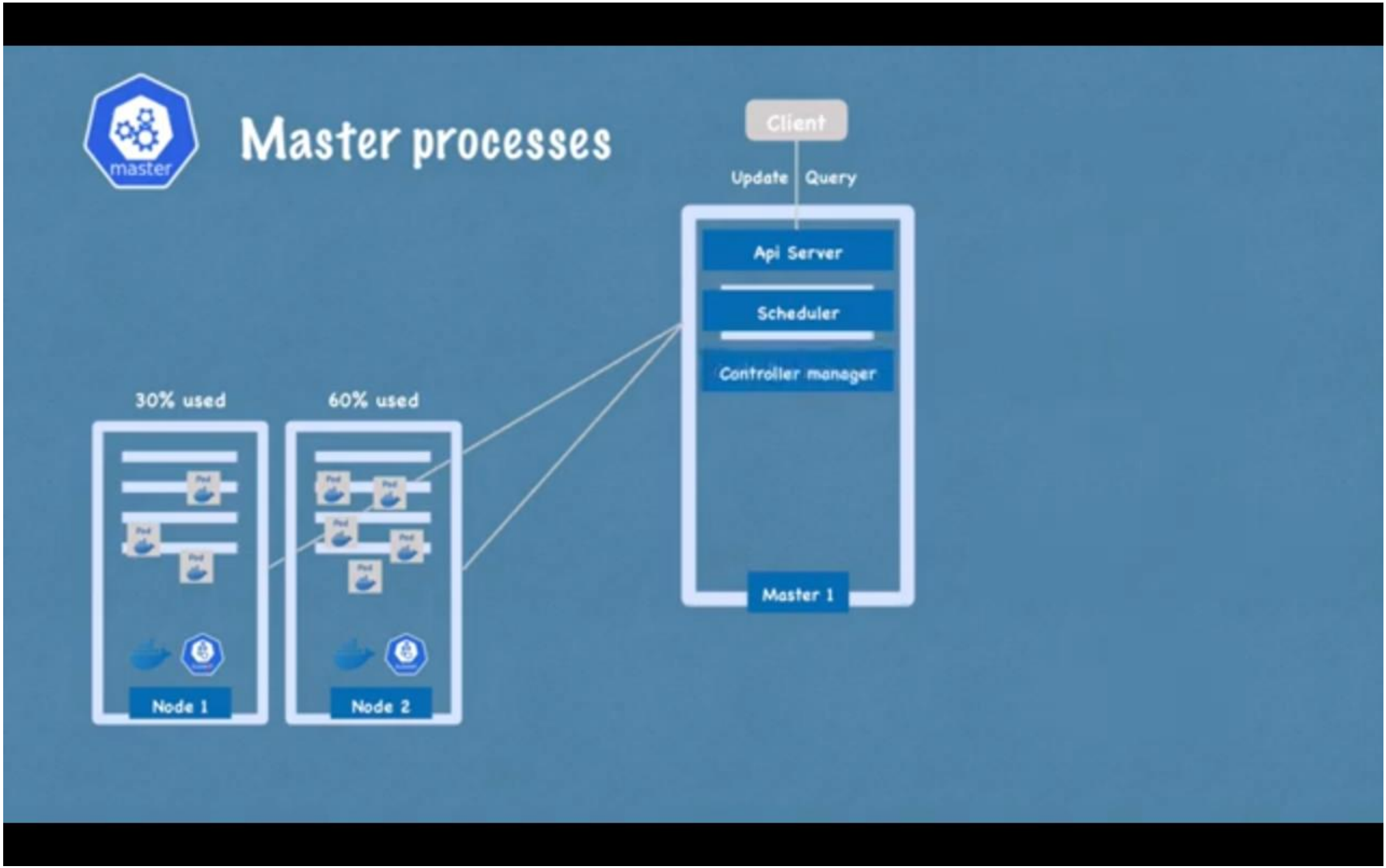

master
Master processes
Client
Update
Query
Api Server
Scheduler
Master 1
Node 1
Node 2
Pod
Schedule new Pod
API Server
Scheduler

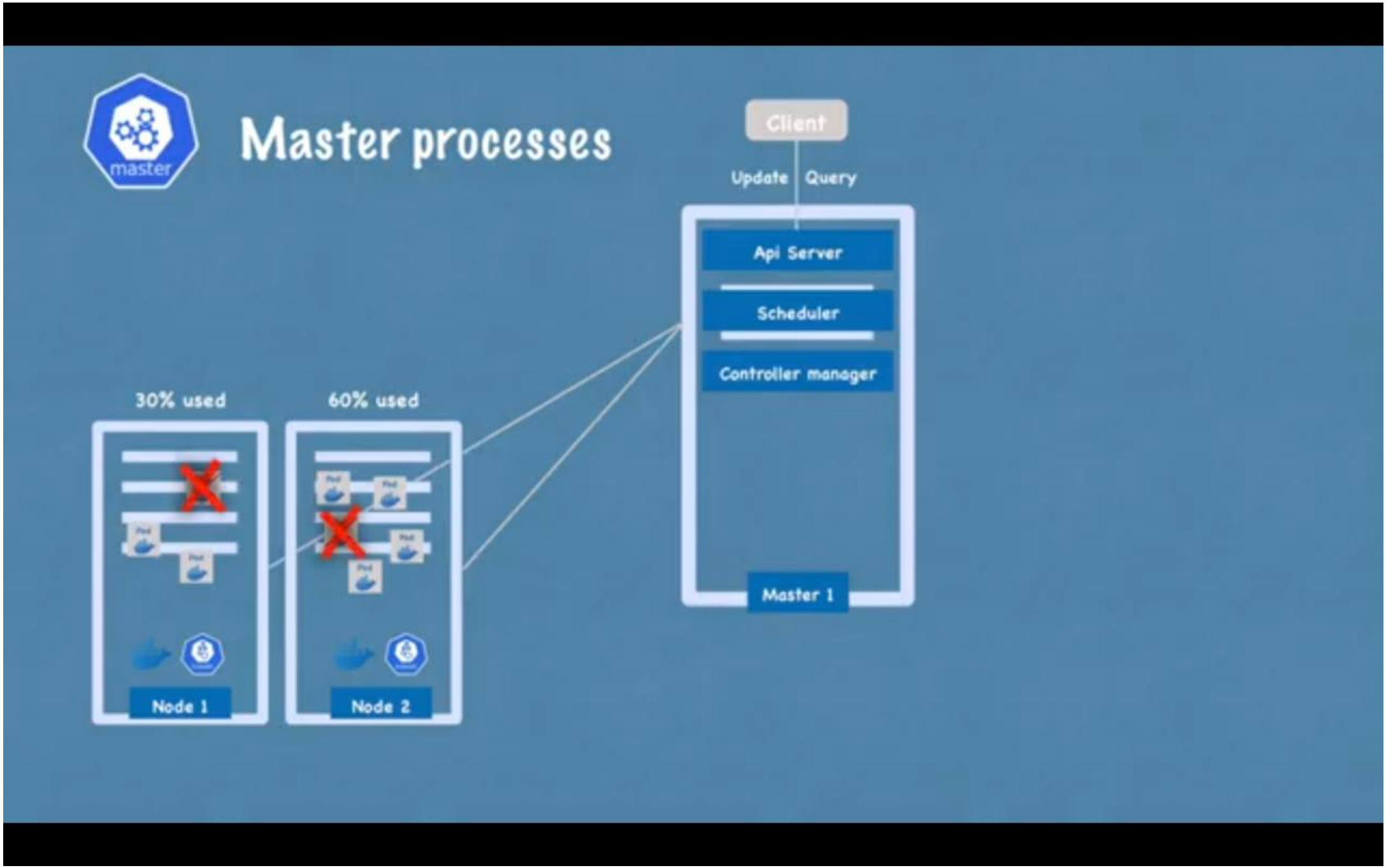

master
Master processes
Client
Update
Query
Api Server
Scheduler
Master 1
Node 1
Node 2
Schedule new Pod
API Server
Scheduler
Where to put the Pod?

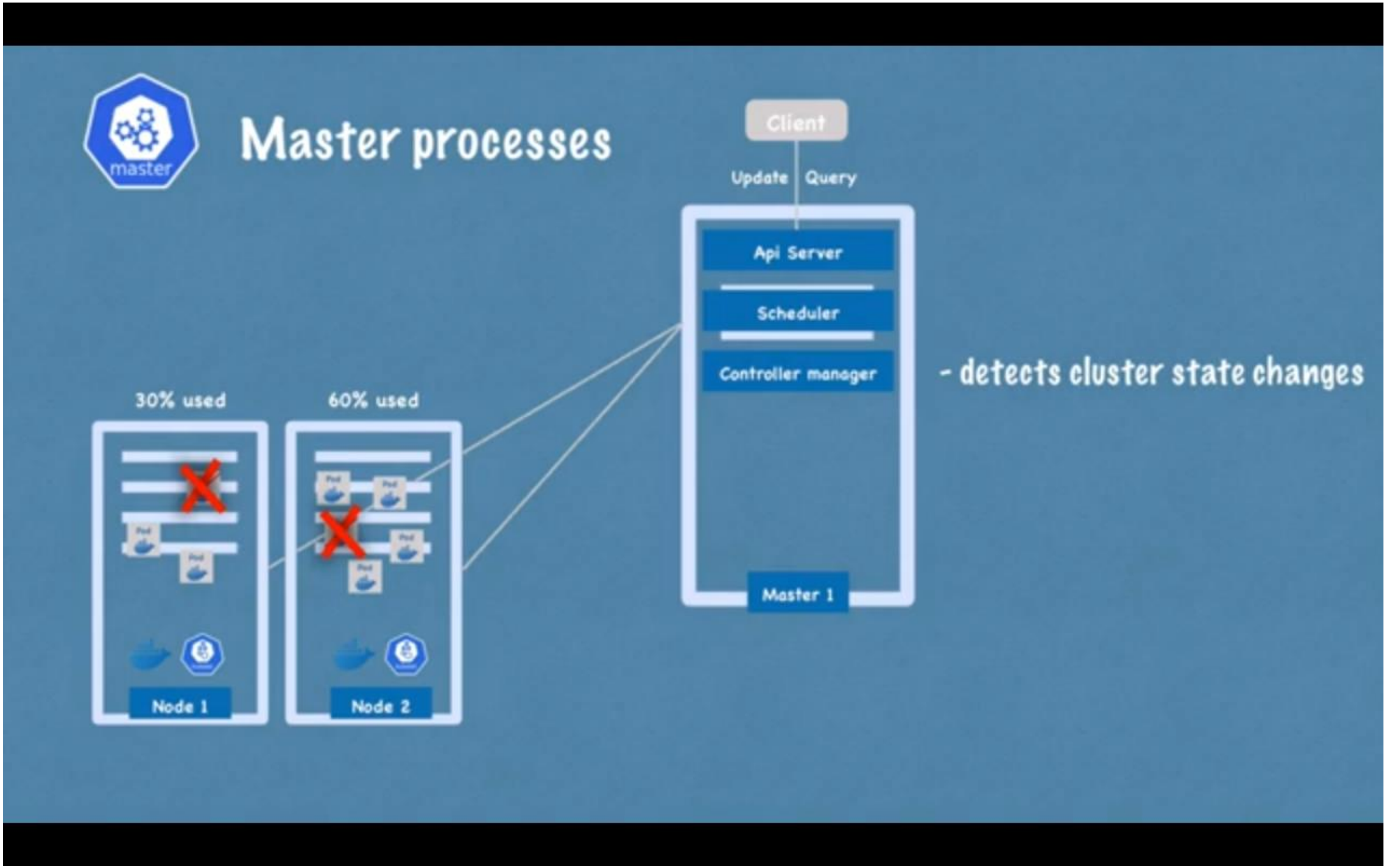

Master processes
master
Client
Update
Query
Api Server
Scheduler
Master 1
30% used
60% used
Node 1
Node 2
Schedule new Pod
API Server
Scheduler
Where to put the Pod?

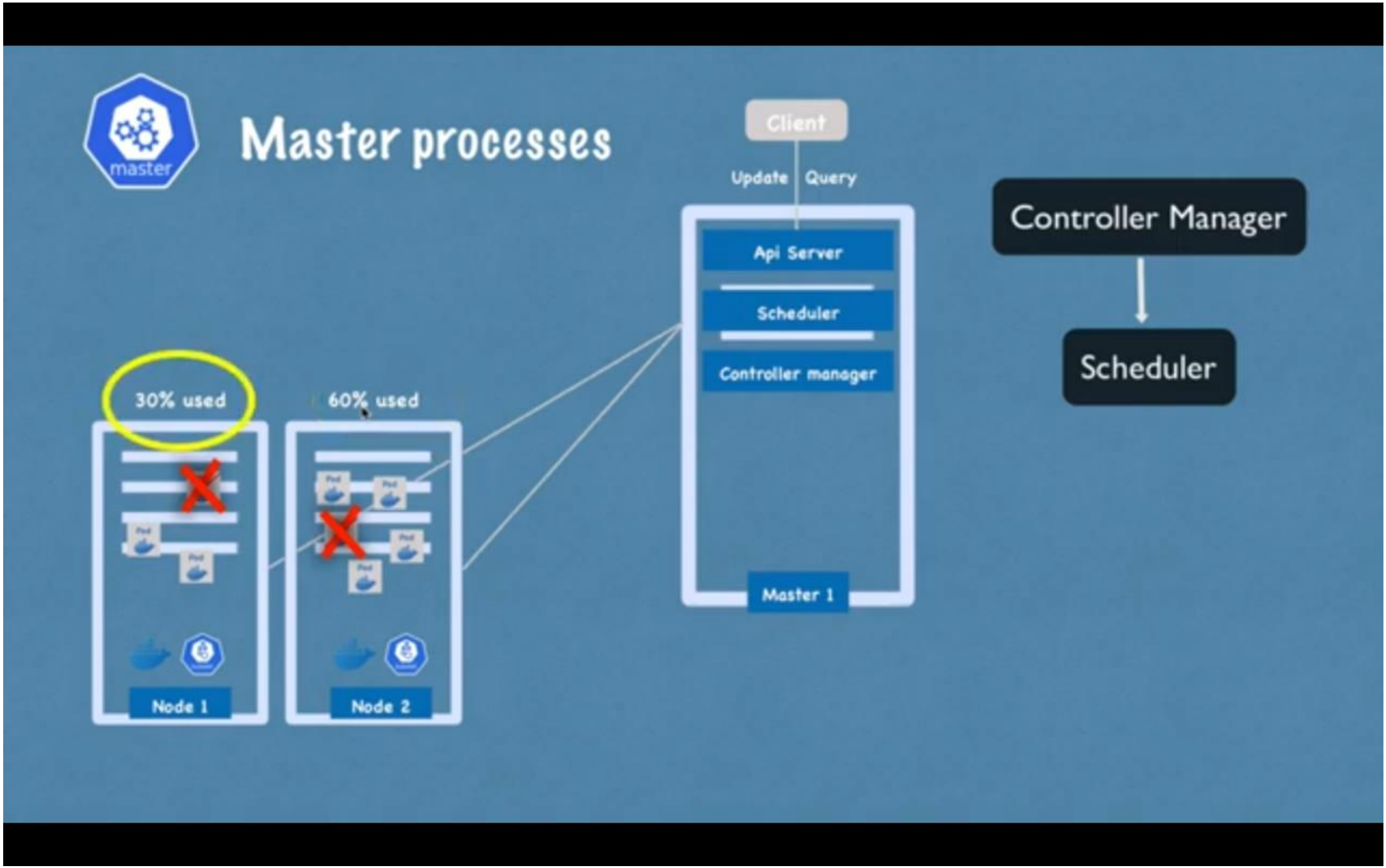

Master processes
Scheduler just decides on which Node new Pod should be scheduled
30% used
60% used
Node 1
Node 2
Client
Update
Query
Api Server
Scheduler
Master 1
Schedule new Pod
API Server
Scheduler
Where to put the Pod?

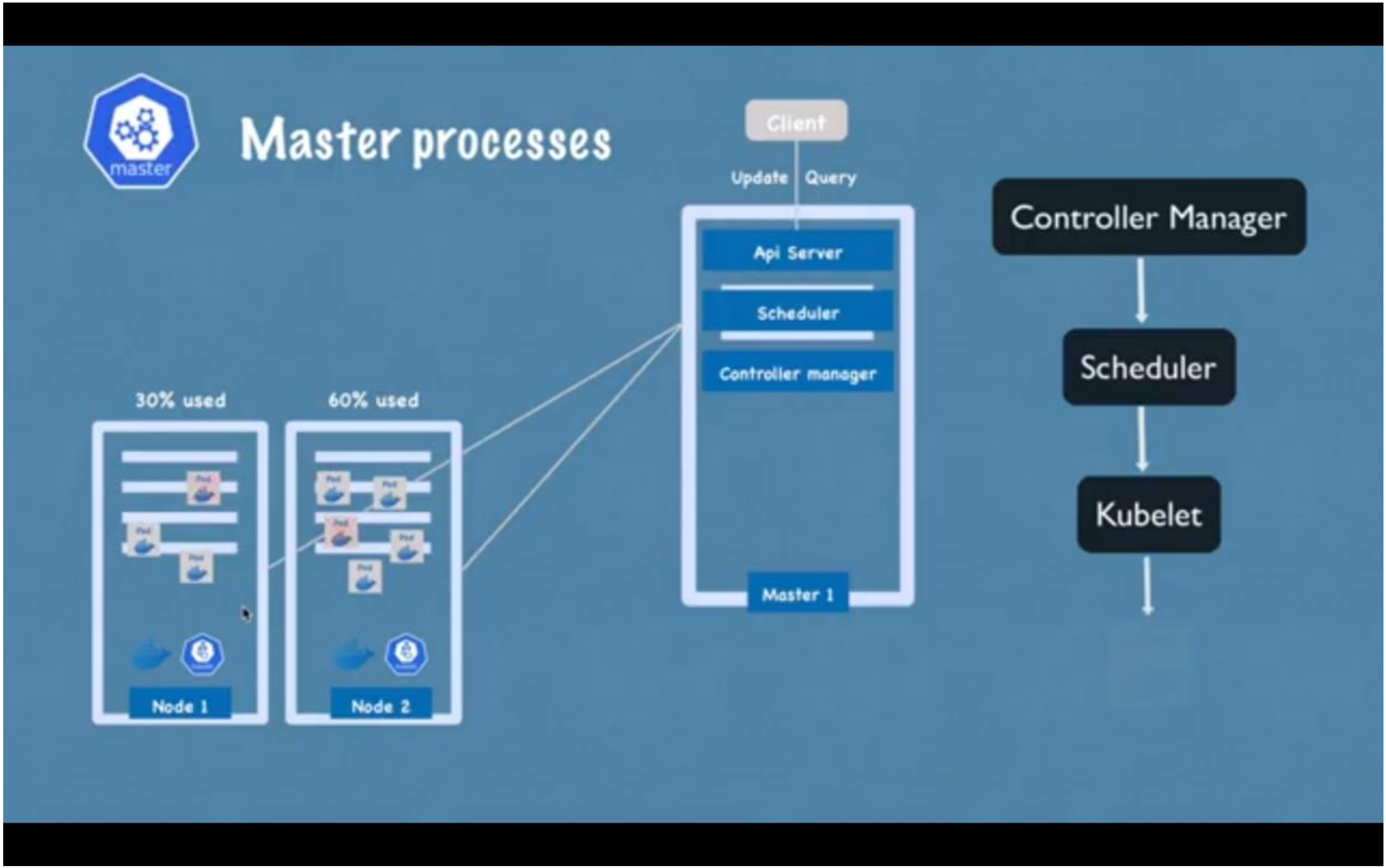

Master processes
Scheduler just decides on which Node new Pod should be scheduled
30% used
60% used
Node 1
Node 2
Client
Update
Query
Api Server
Scheduler
Master 1
Schedule new Pod
API Server
Scheduler
Where to put the Pod?
Kubelet

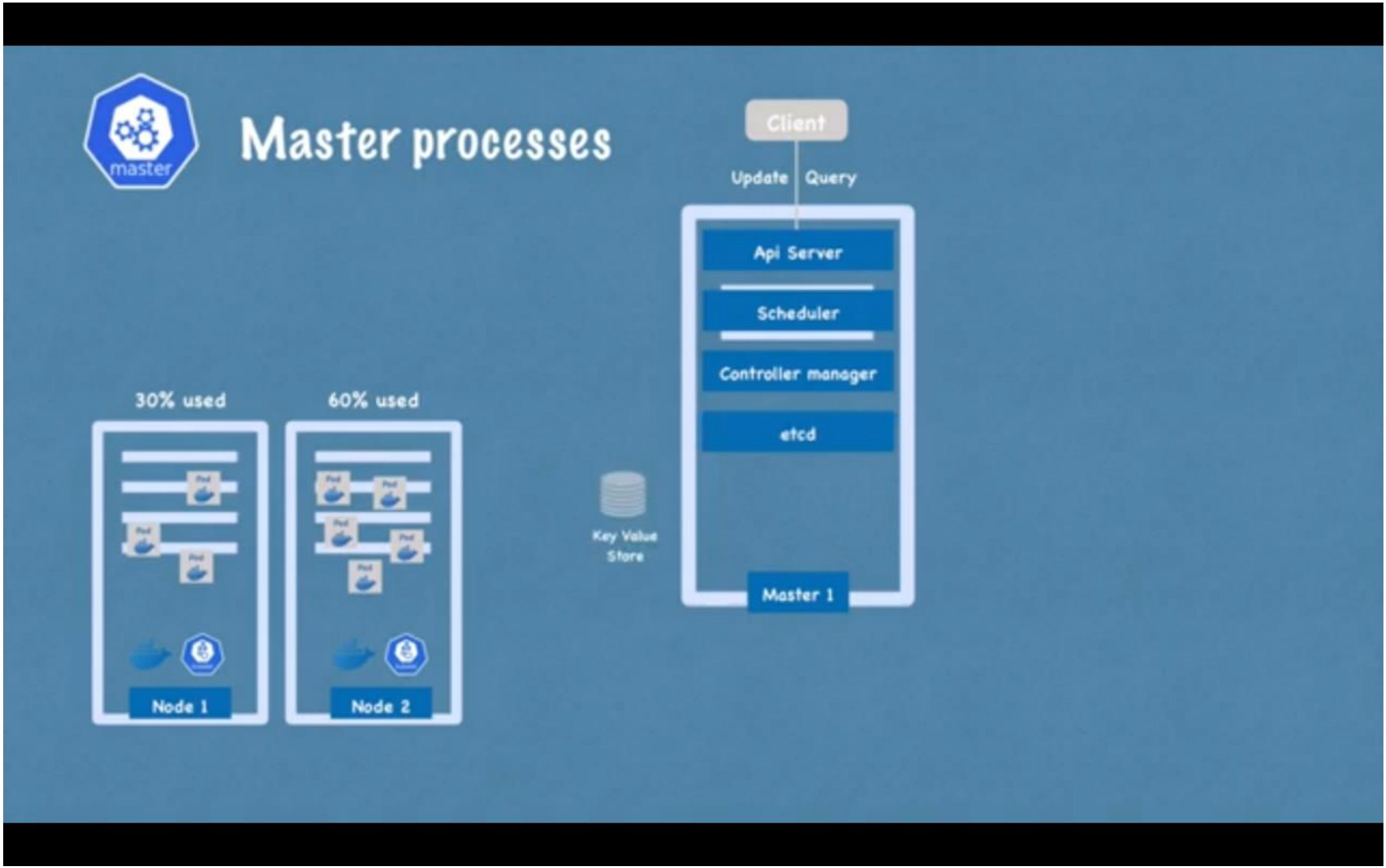

Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
Master 1
30% used
60% used
Node 1
Node 2


Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
Master 1
30% used
60% used
Node 1
Node 2

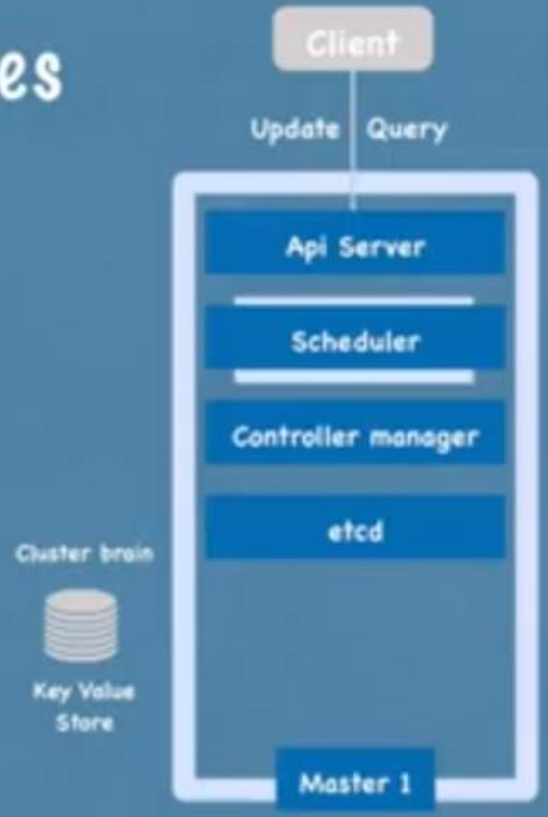

Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
Master 1
- detects cluster state changes
30% used
60% used
Node 1
Node 2


Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
Master 1
Controller Manager
Scheduler
30% used
60% used
Node 1
Node 2


master
Master processes
Client
Update
Query
Api Server
Scheduler
Controller manager
Master 1
30% used
60% used
Node 1
Node 2
Controller Manager
Scheduler
Kubelet

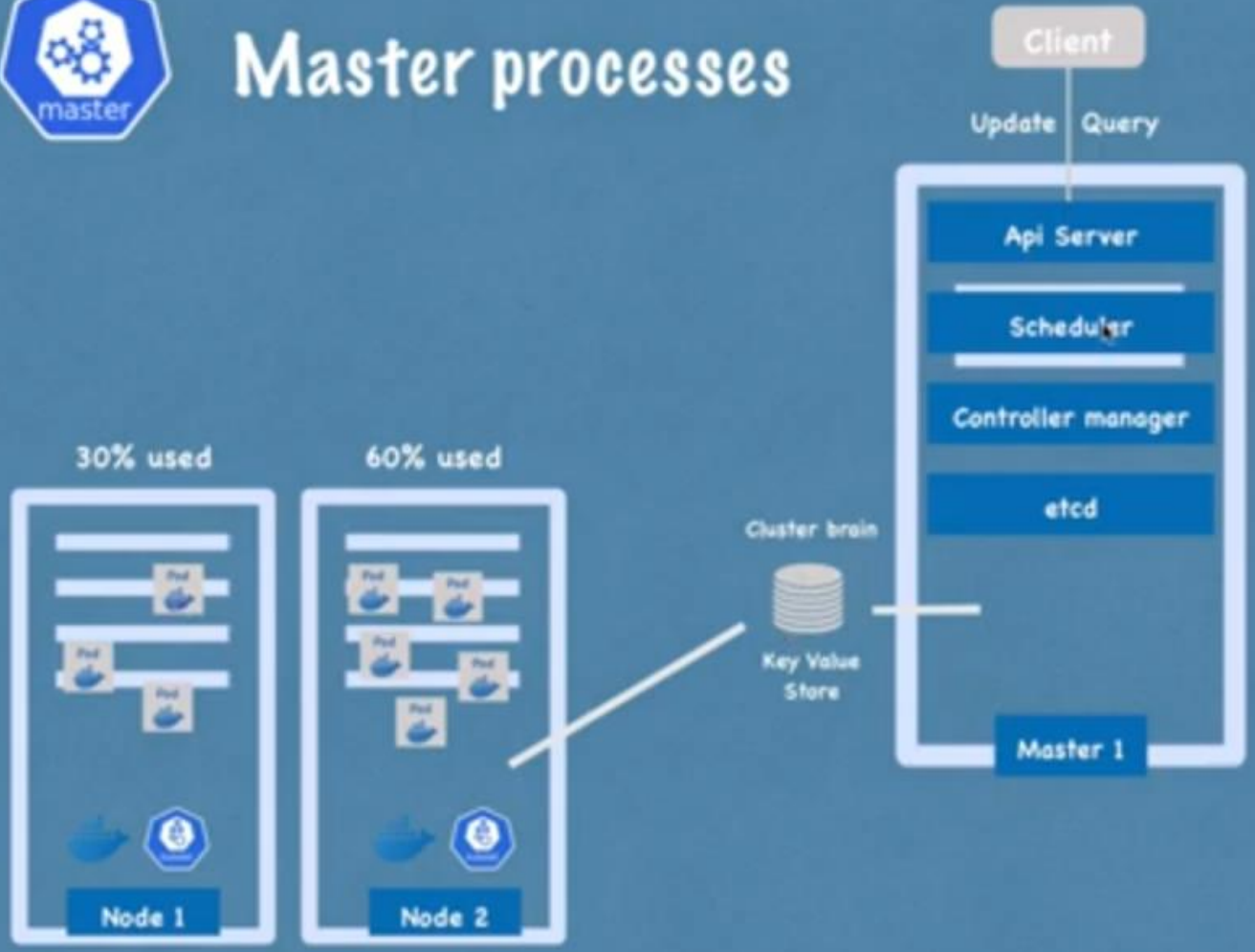

Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
etcd
Master 1
Key Value Store
30% used
60% used
Pod
Node 1
Node 2

Master processes

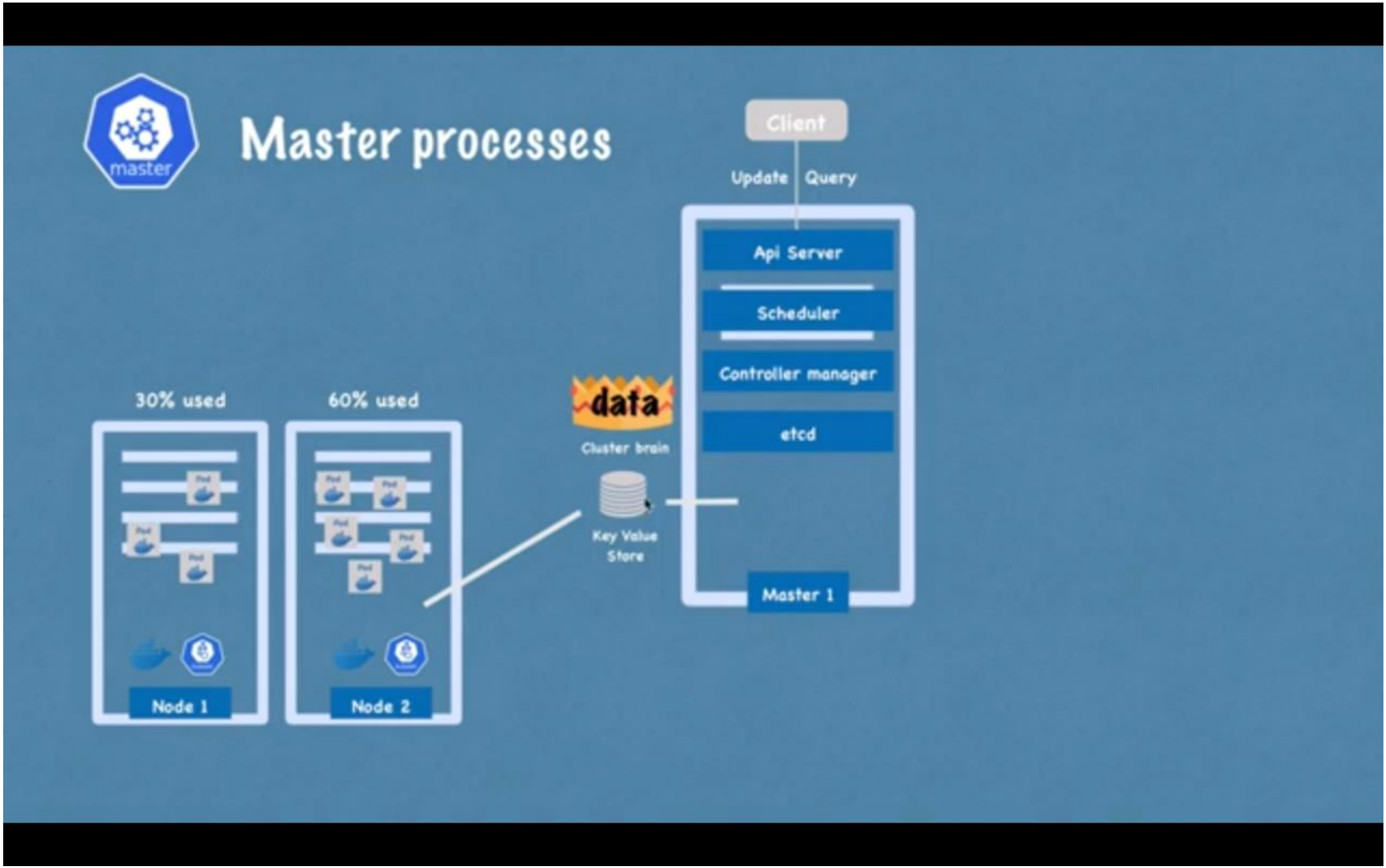


- etcd is the cluster brain!

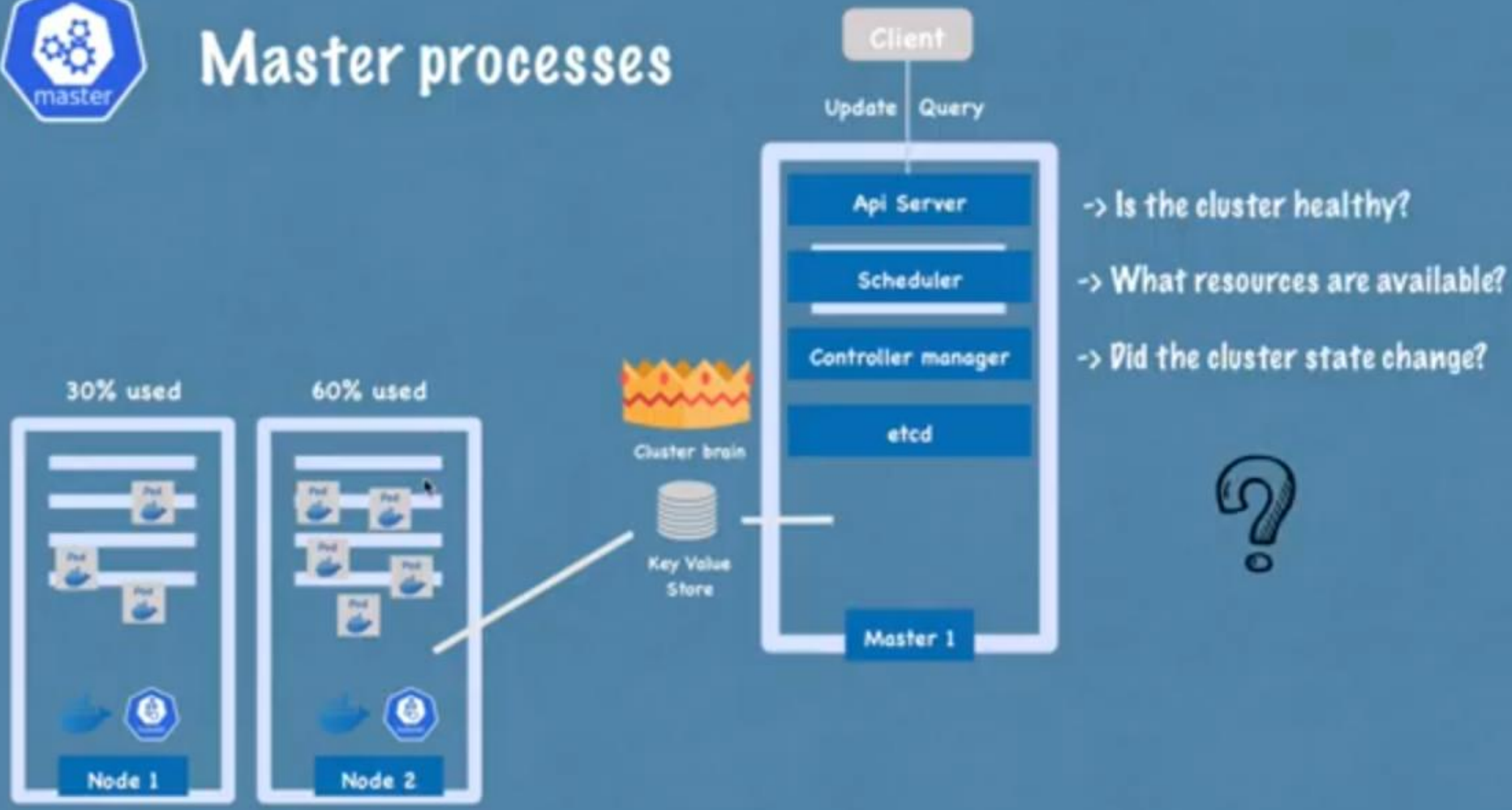
Master processes

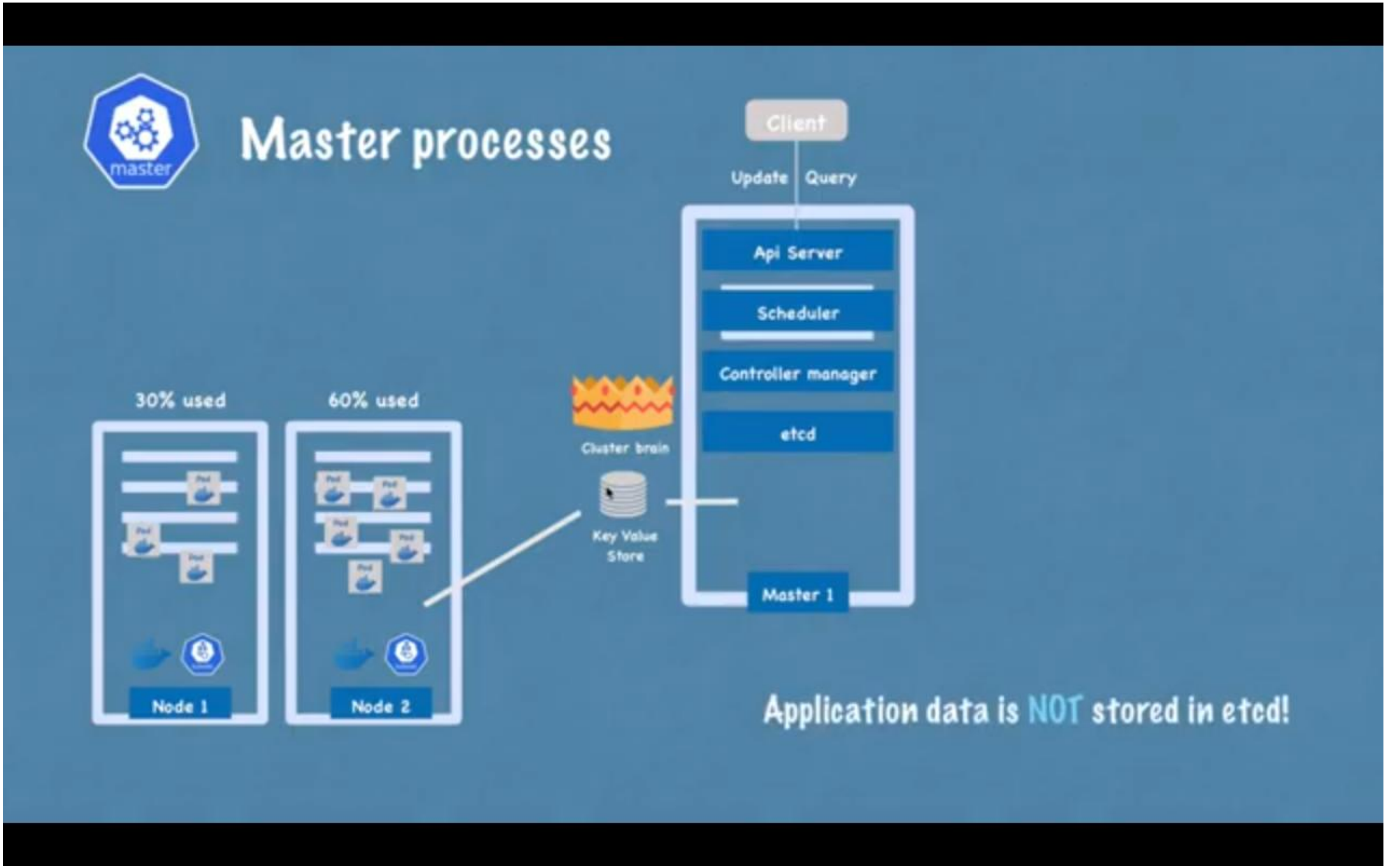


- etcd is the cluster brain!
- Cluster changes get stored in the key value store

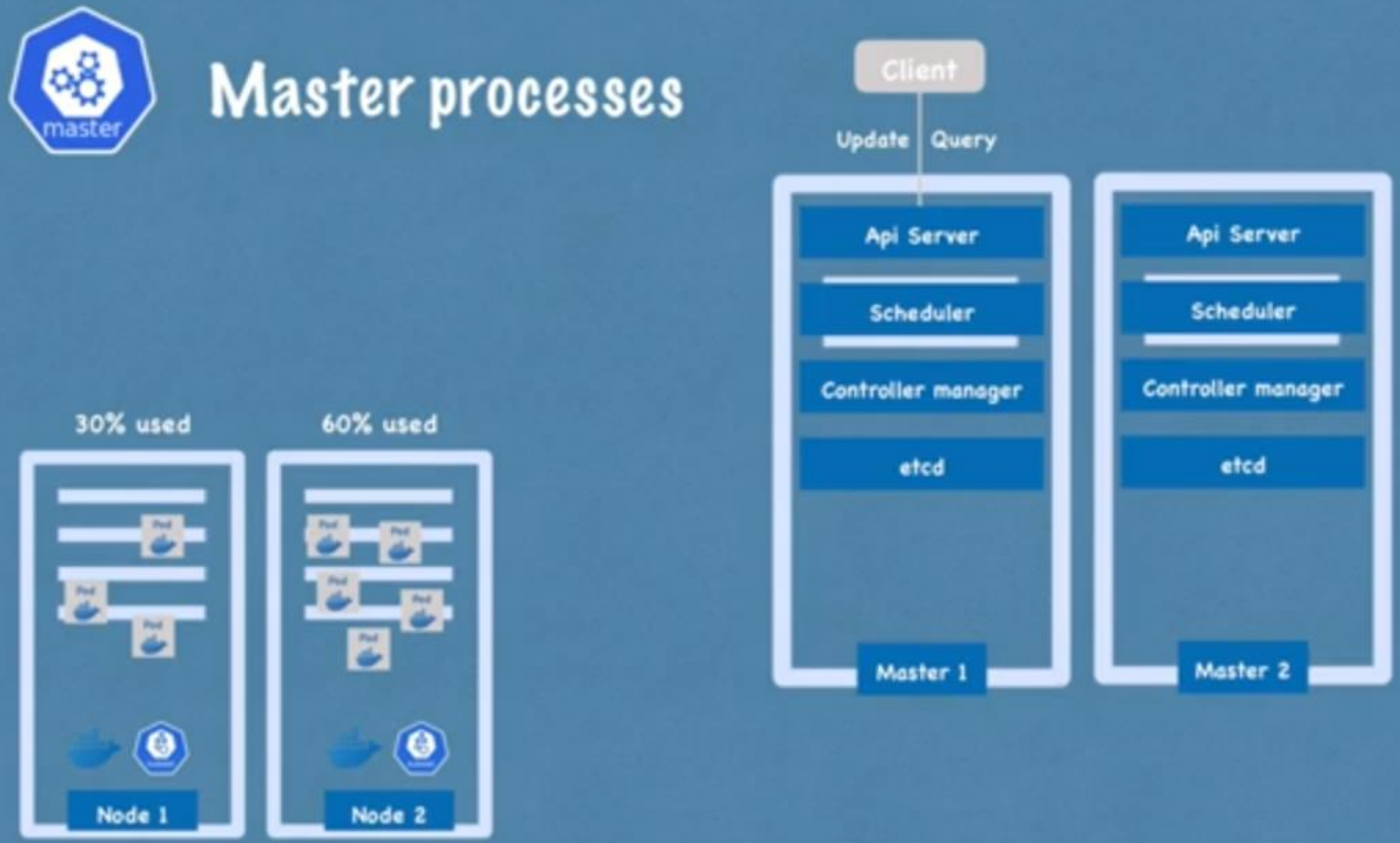

Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
etcd
Master 1
data
Cluster brain
Key Value Store
30% used
60% used
Node 1
Node 2

Master processes

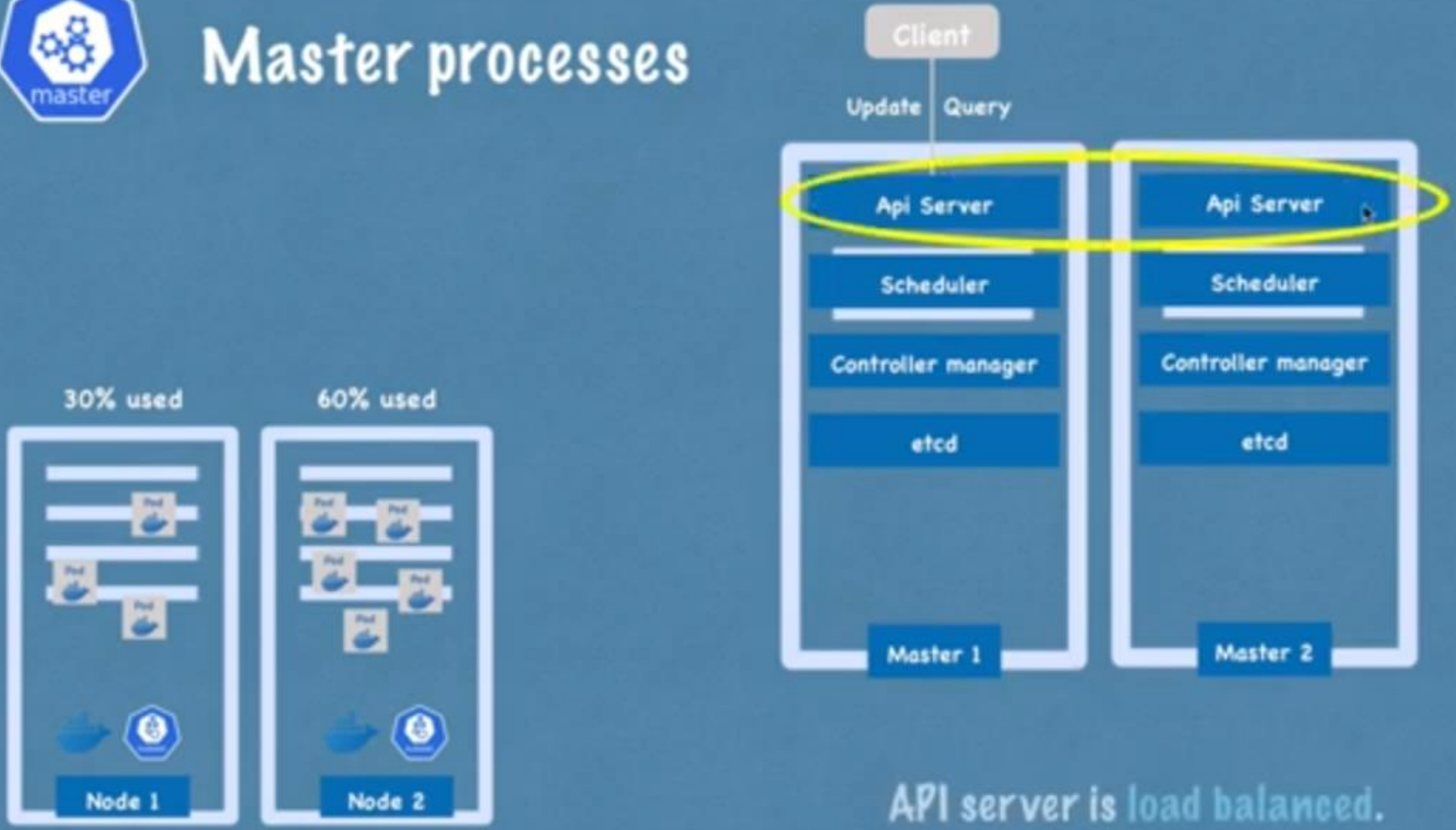



Master processes
master
Client
Update
Query
Api Server
Scheduler
Controller manager
etcd
Master 1
Cluster brain
Key Value Store
30% used
60% used
Node 1
Node 2
Application data is NOT stored in etcd!

Master processes

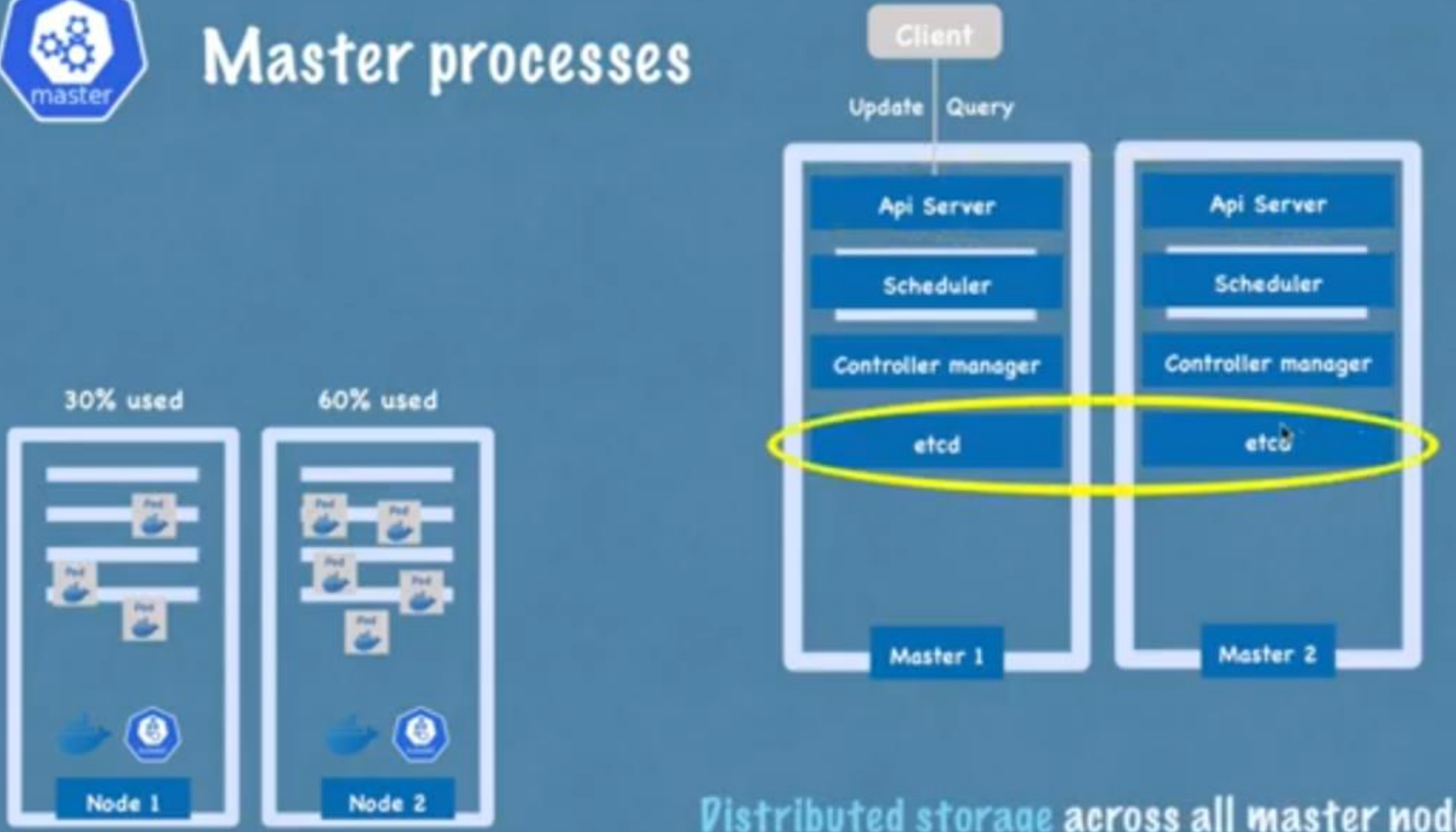


Master processes



API server is load balanced.

Master processes



Distributed storage across all master nodes

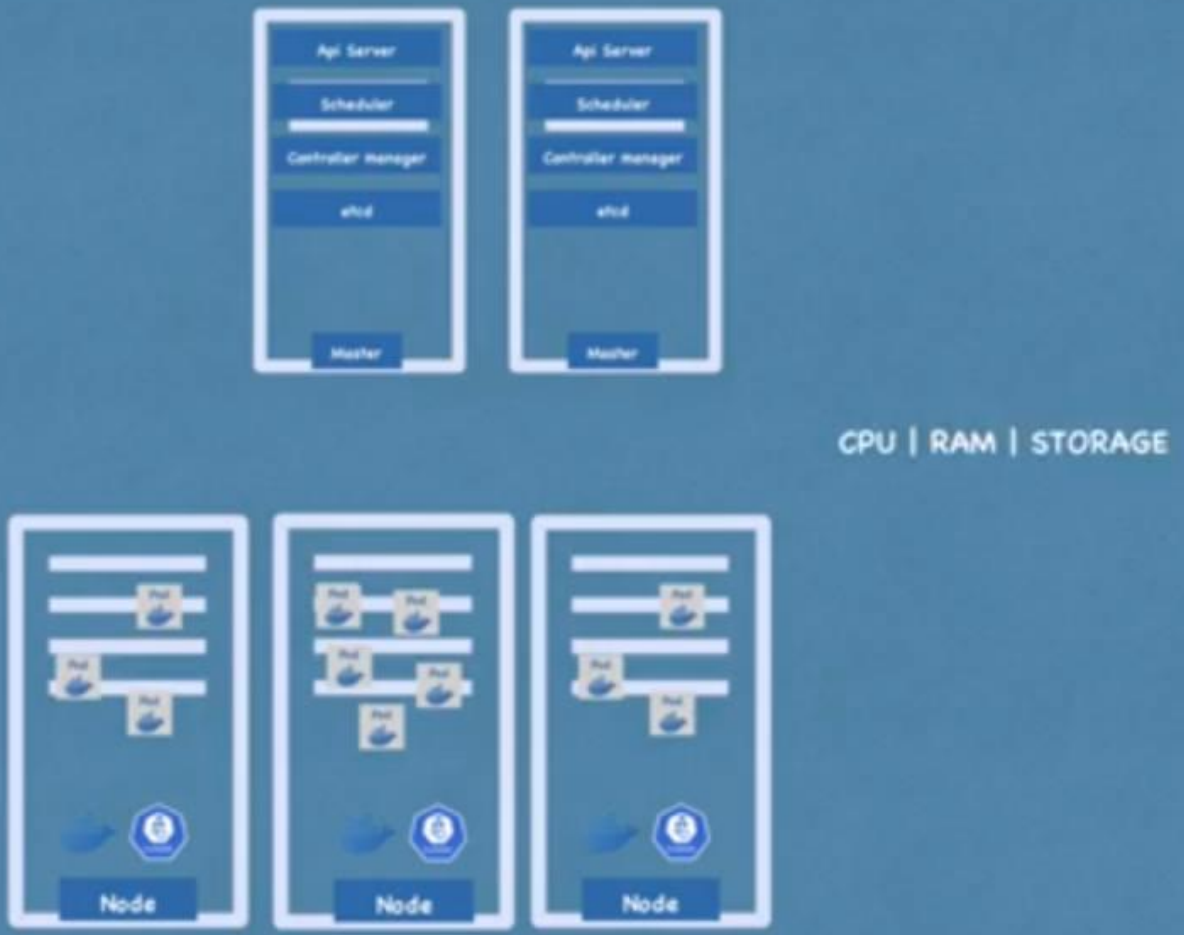

Example cluster setup

Example Cluster Set-Up

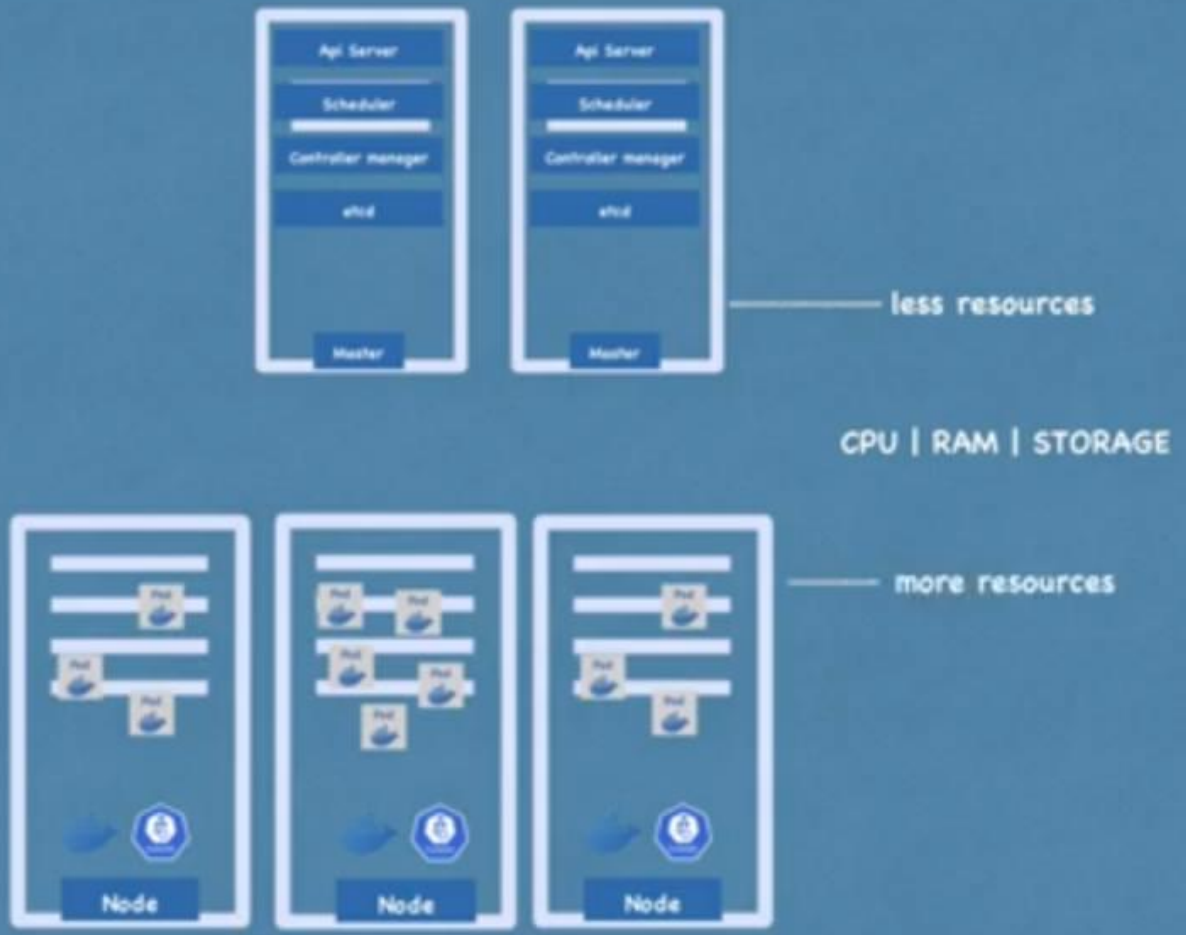
2 Master Nodes

3 Worker Nodes

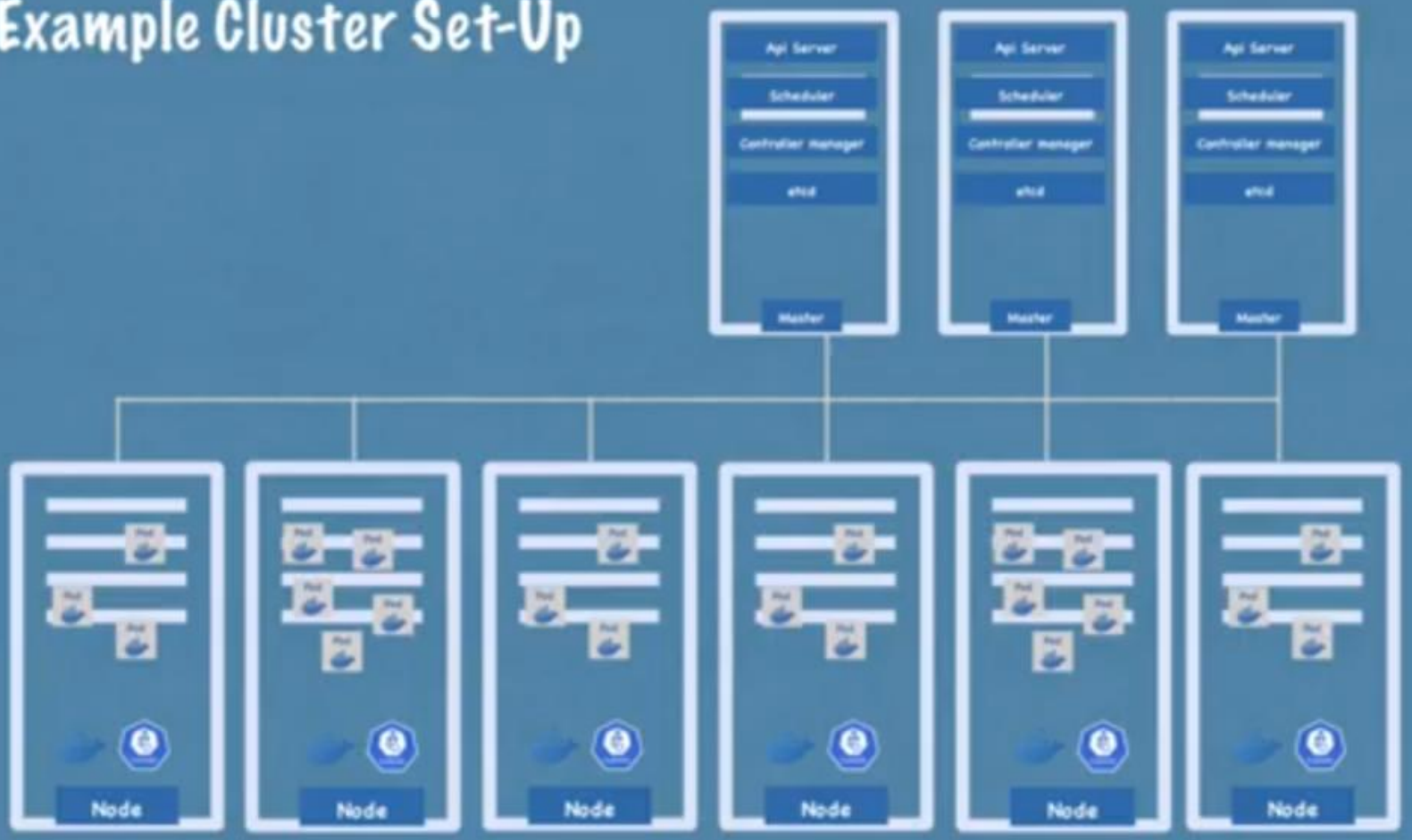
Example Cluster Set-Up



Example Cluster Set-Up



Example Cluster Set-Up

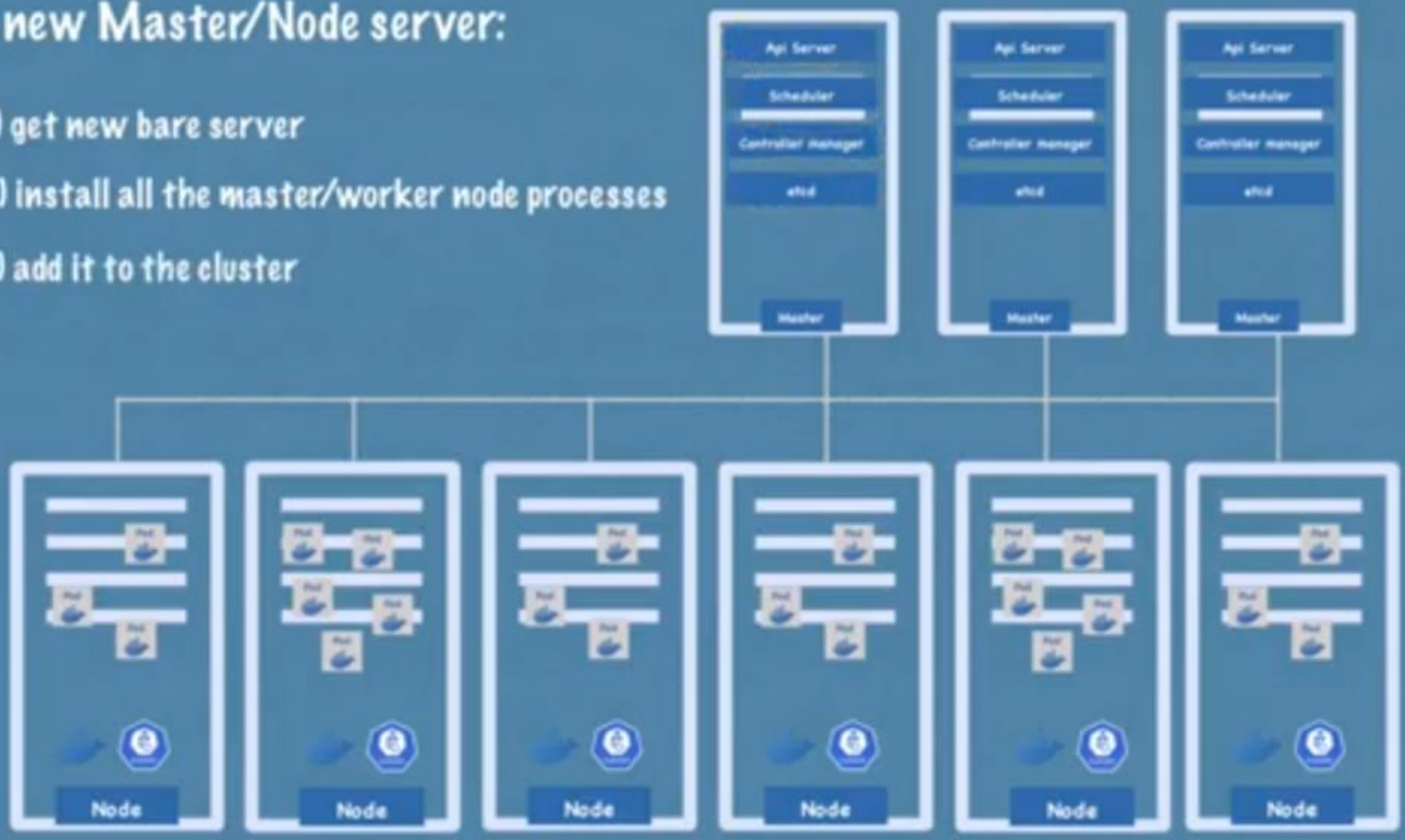


Add new Master/Node server:

1) get new bare server

2) install all the master/worker node processes

3) add it to the cluster



Kubernetes is an orchestration tool

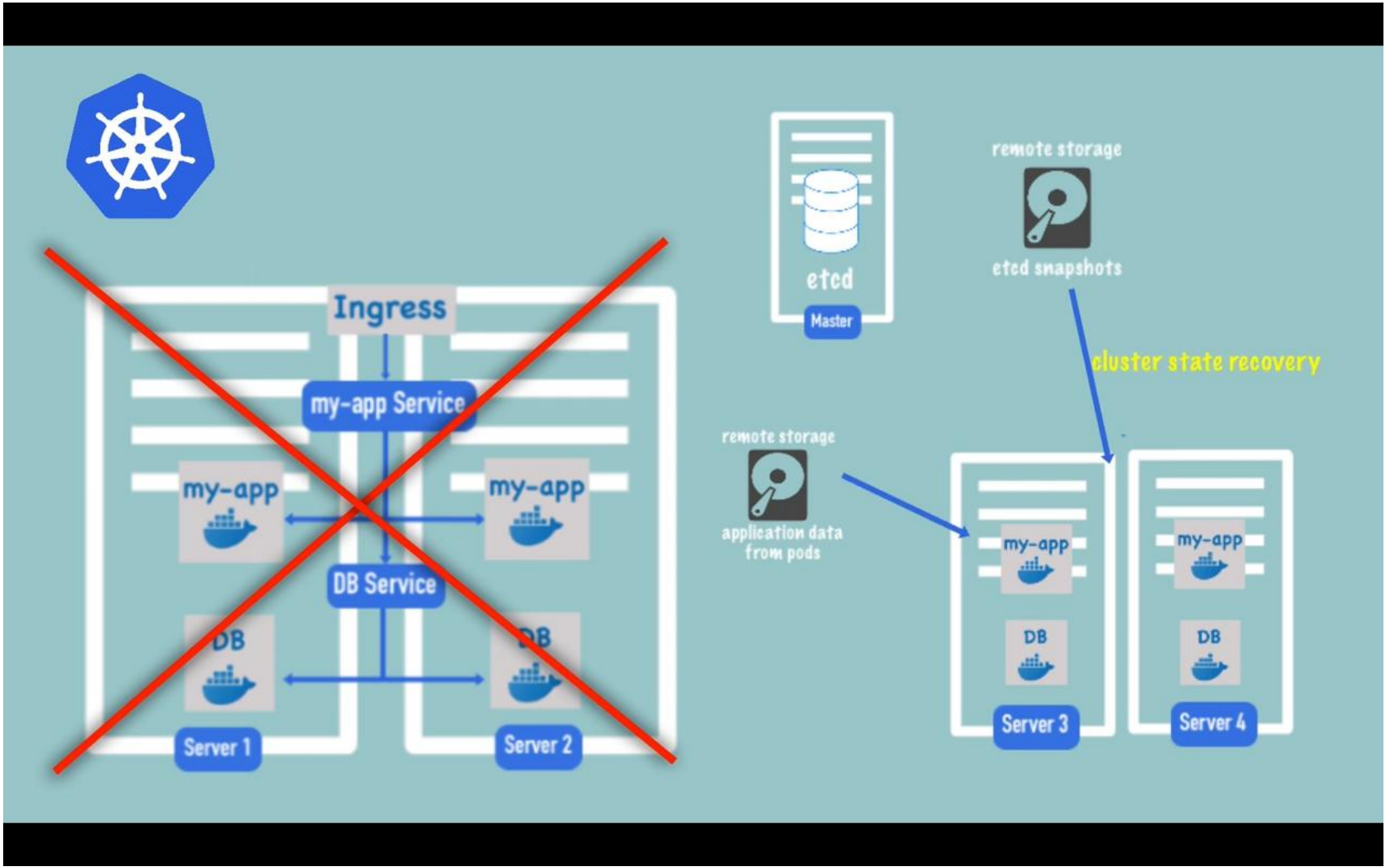

high availability


scalability
high availability
disaster recovery

How K8s actually makes

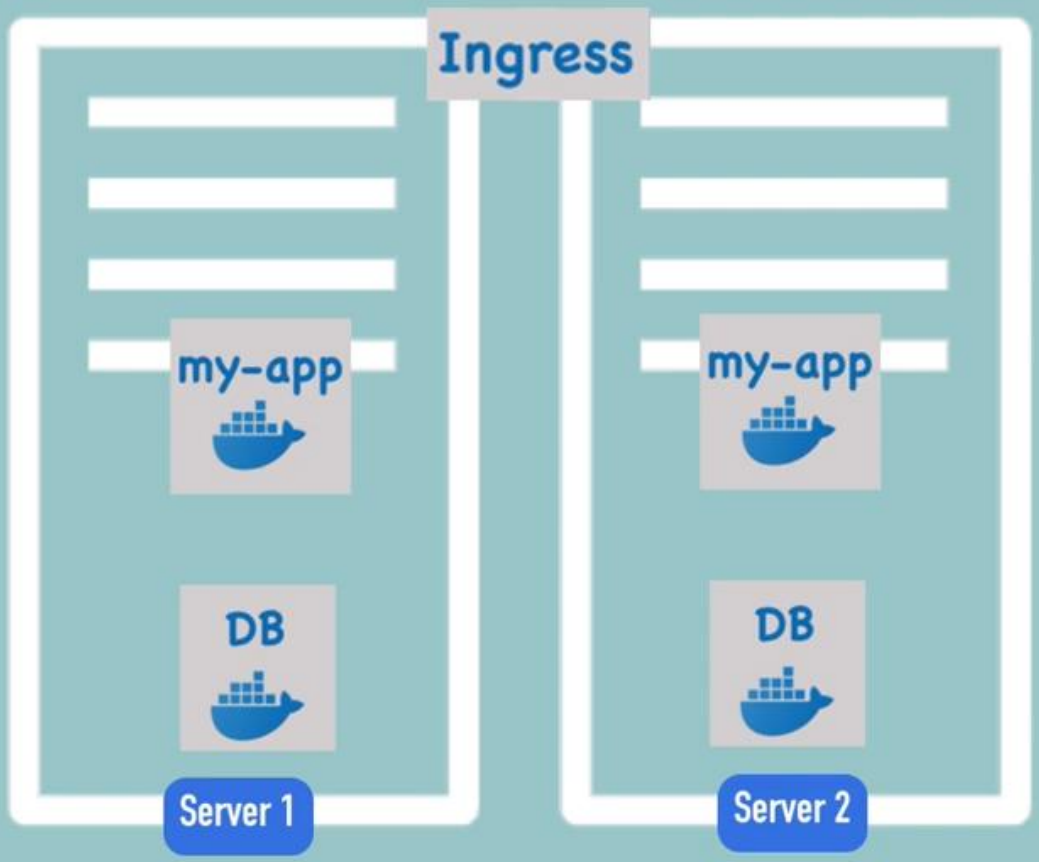
- high availability
- scalability
- disaster recovery

possible?


Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
cluster state recovery
remote storage
application data
from pods
my-app
DB
Server 3
my-app
DB
Server 4

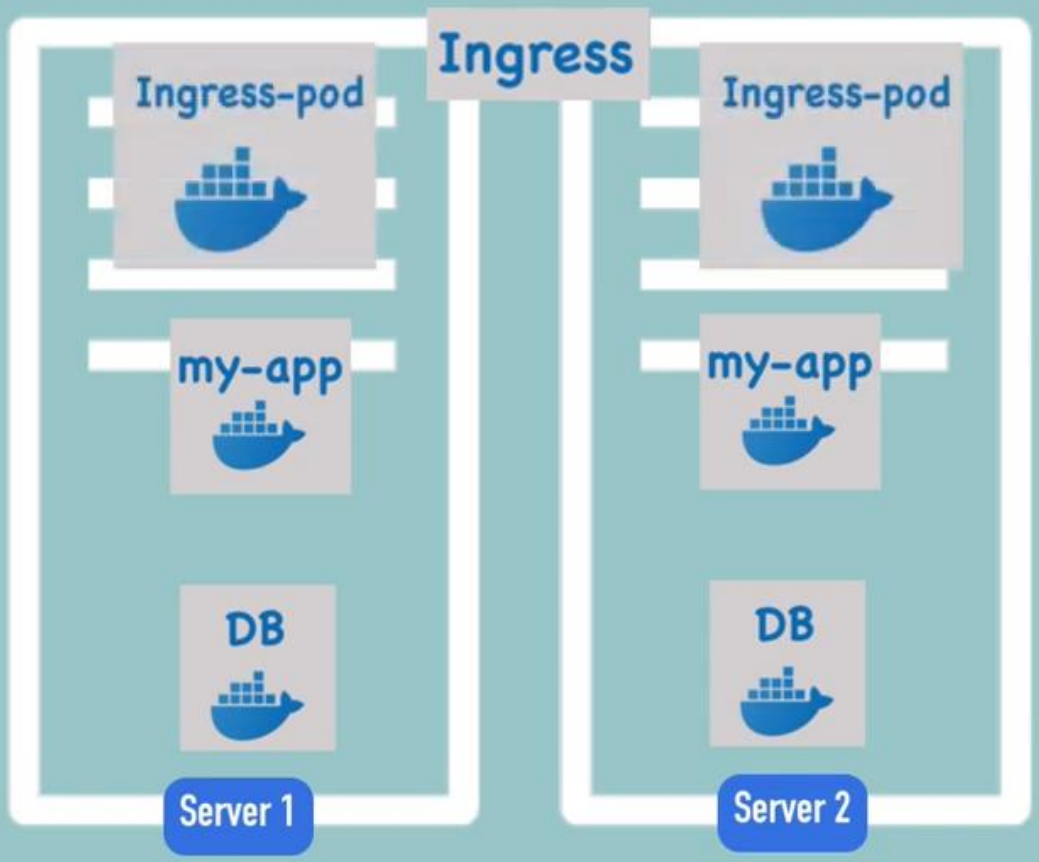
High availability and scalability

High availability and scalability



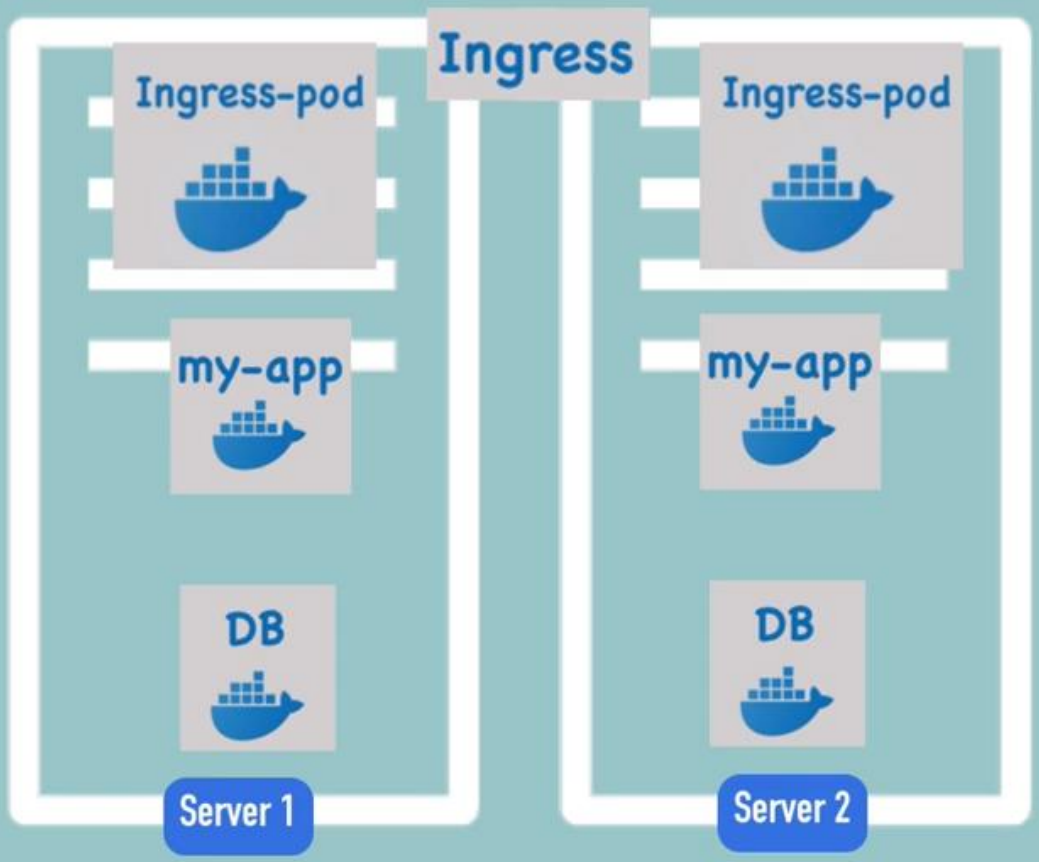
Ingress handles every incoming request

High availability and scalability



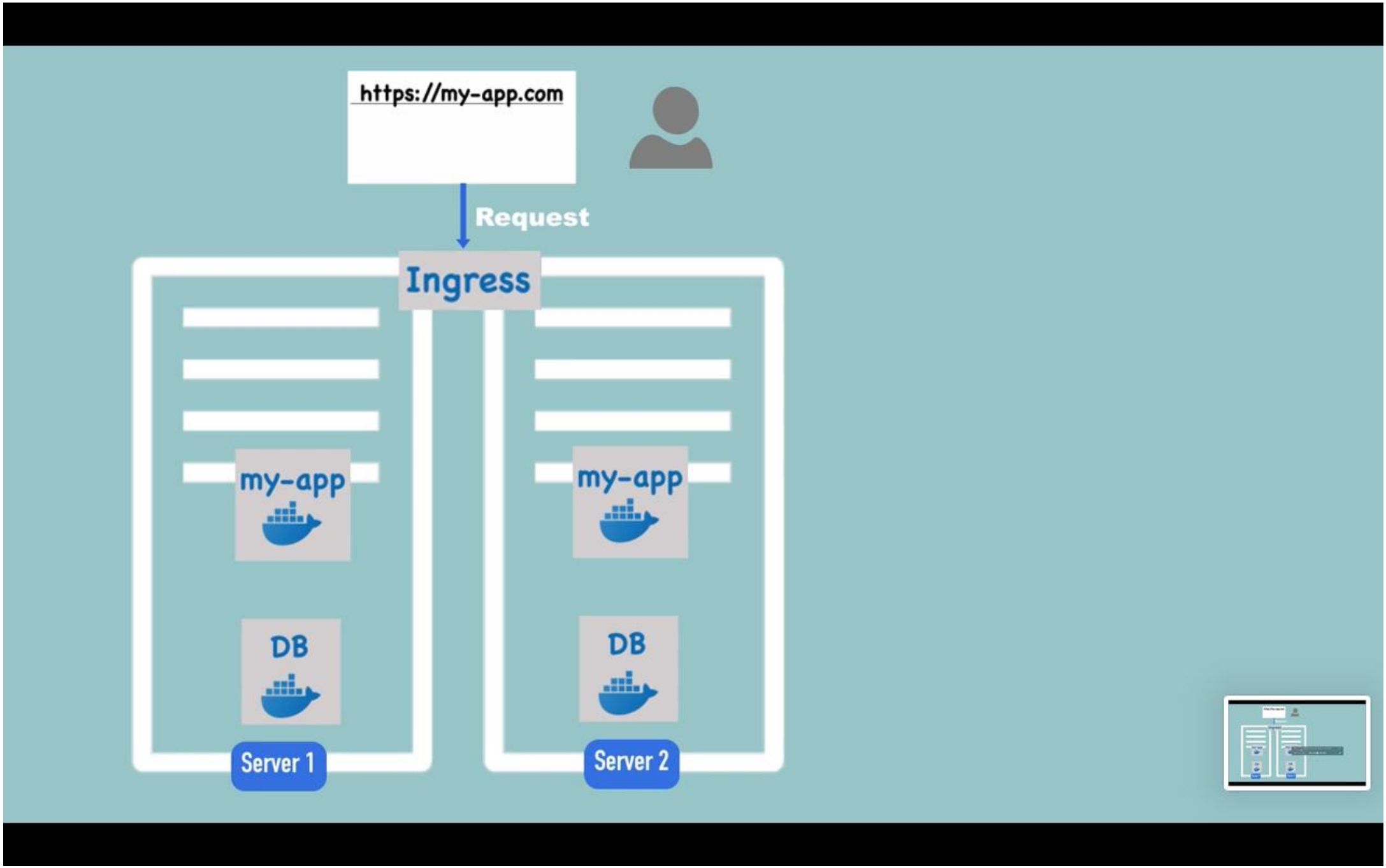
Ingress handles every incoming request

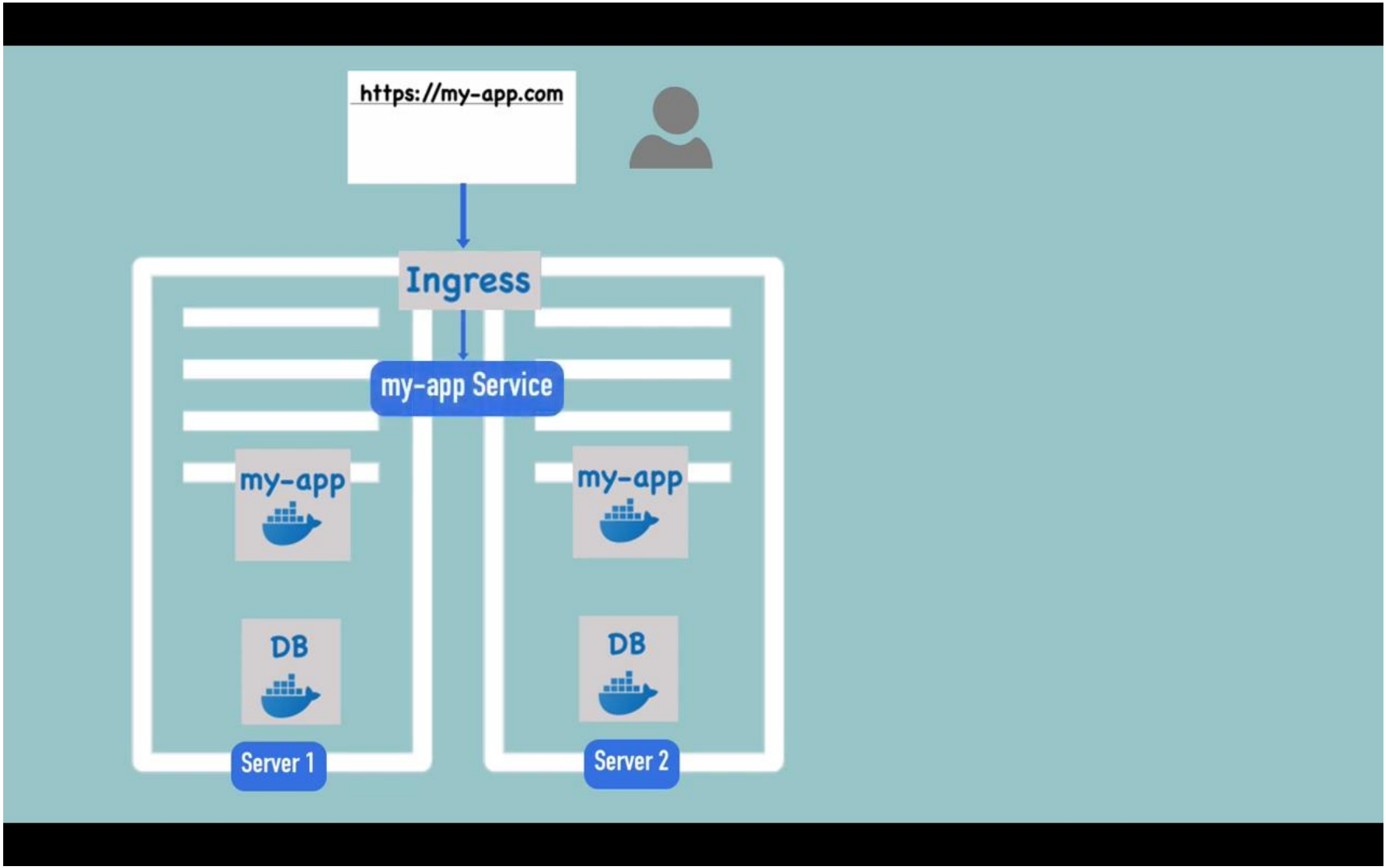
High availability and scalability

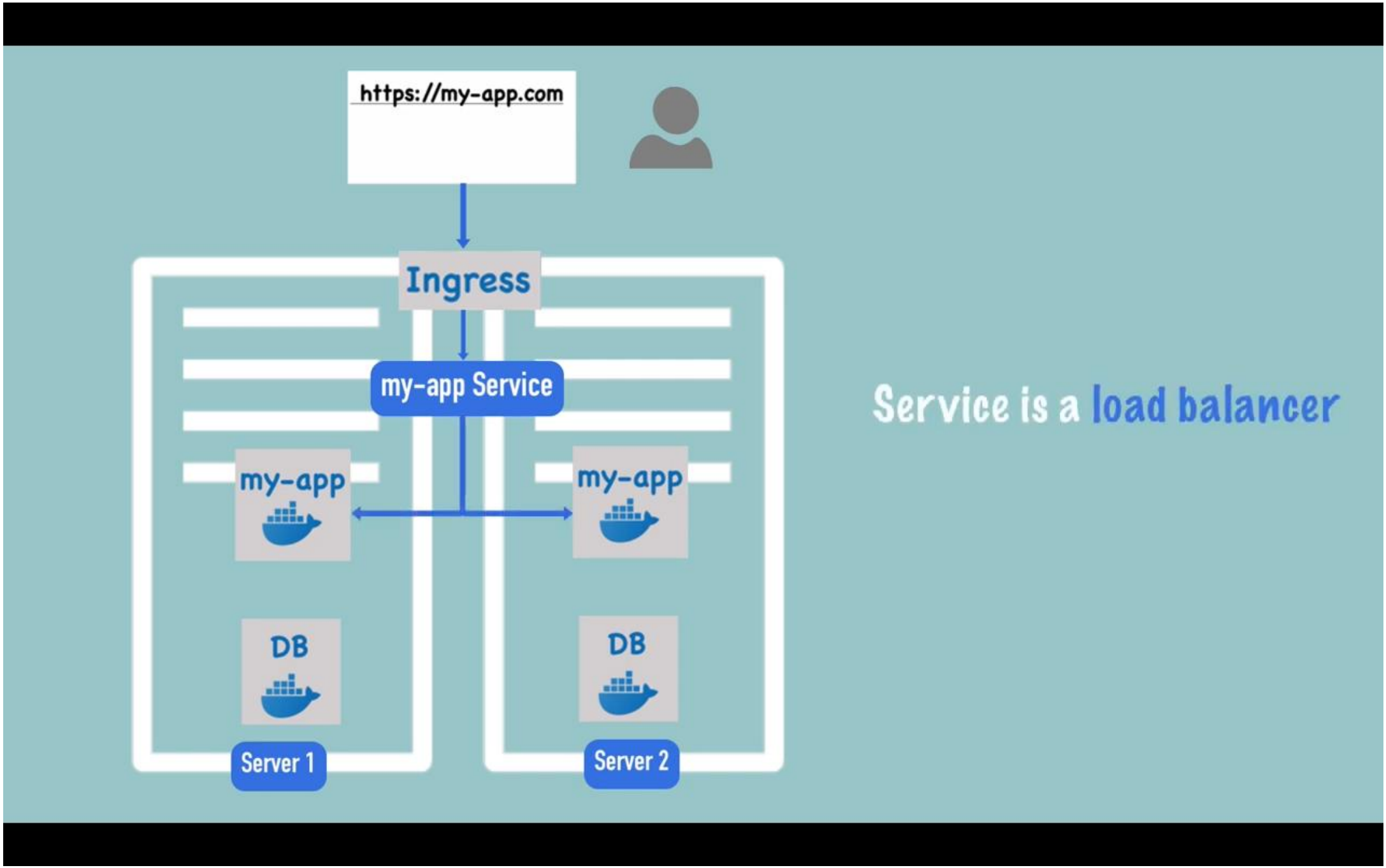


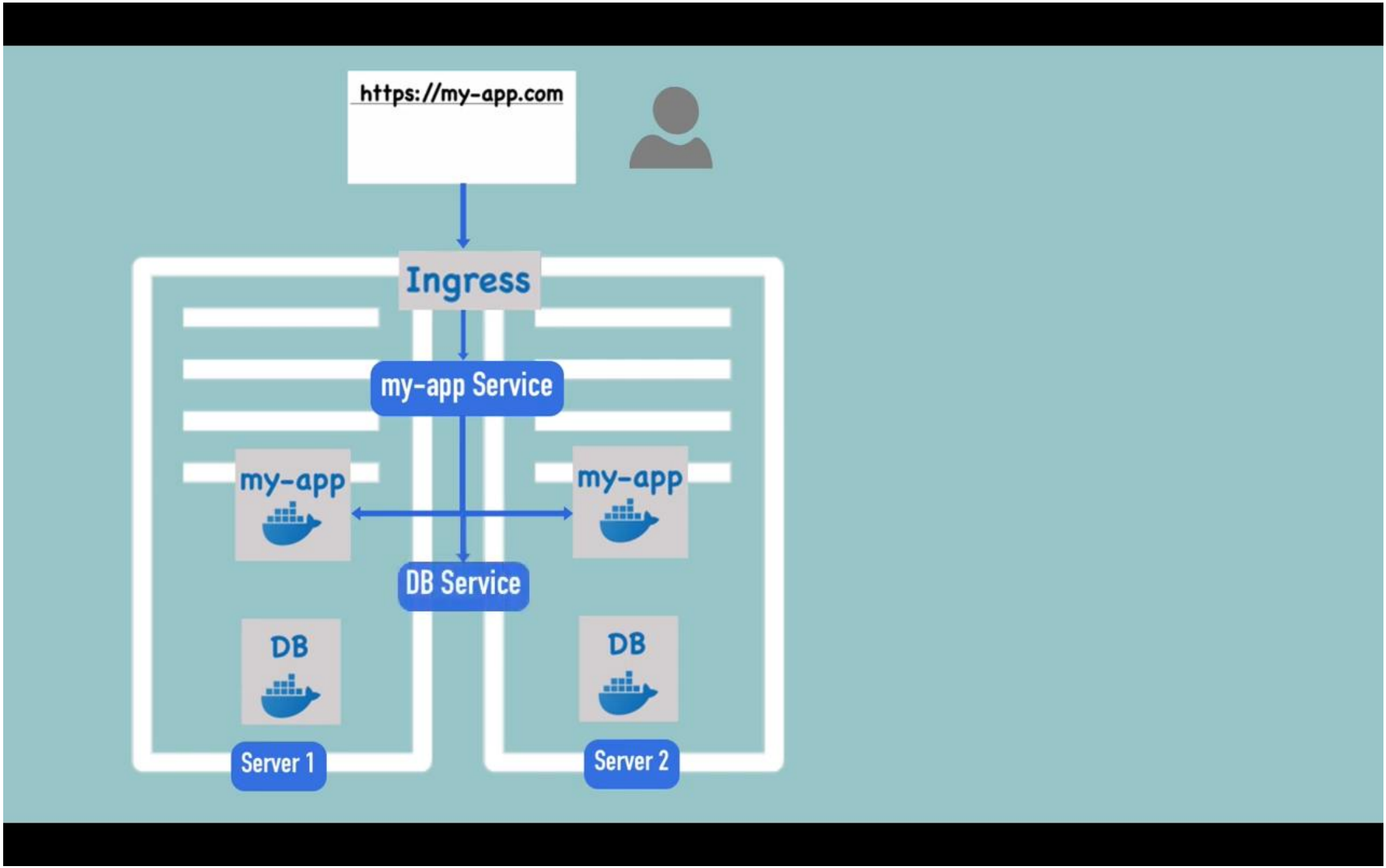
Ingress handles every incoming request

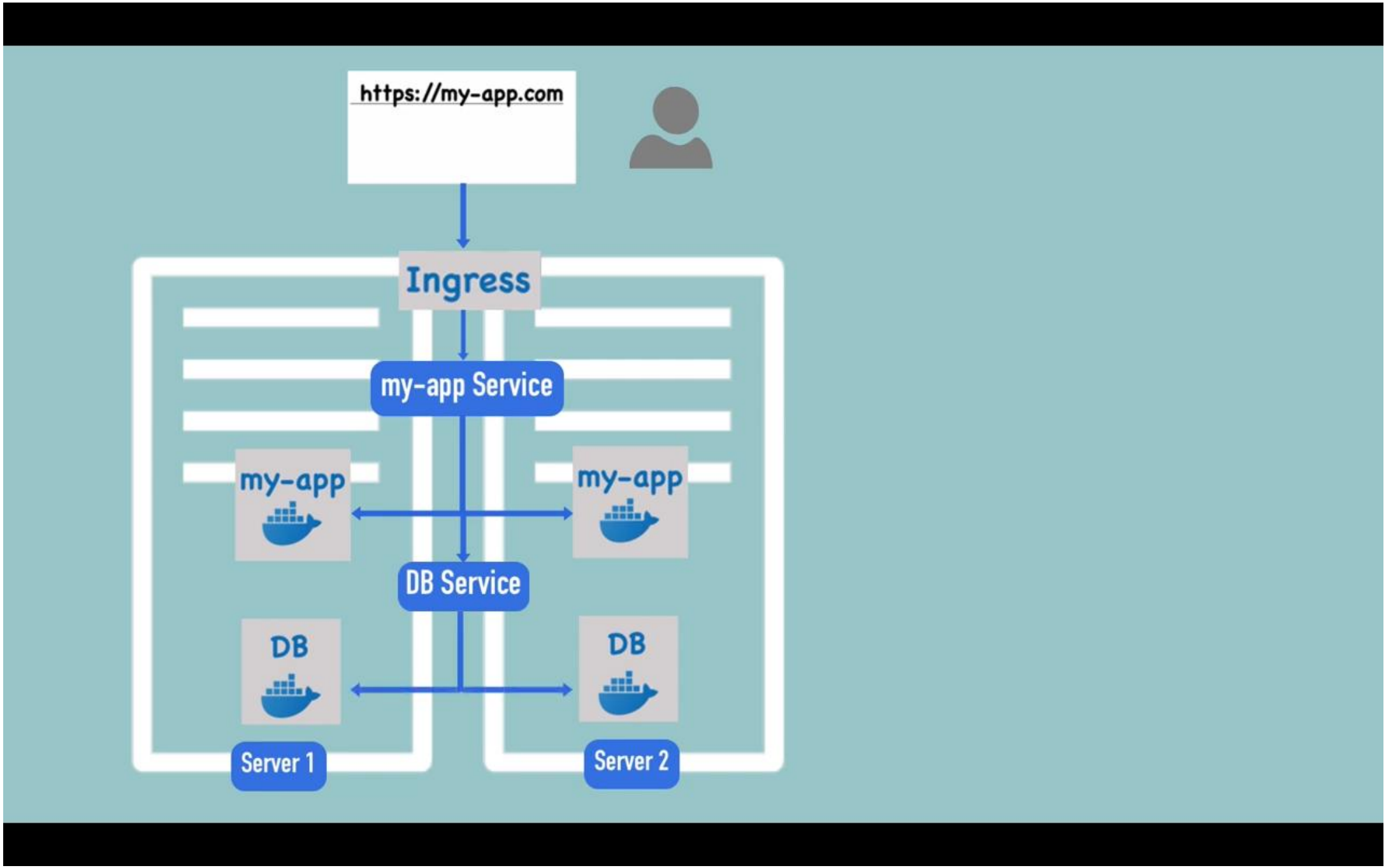
Ingress is also replicated.

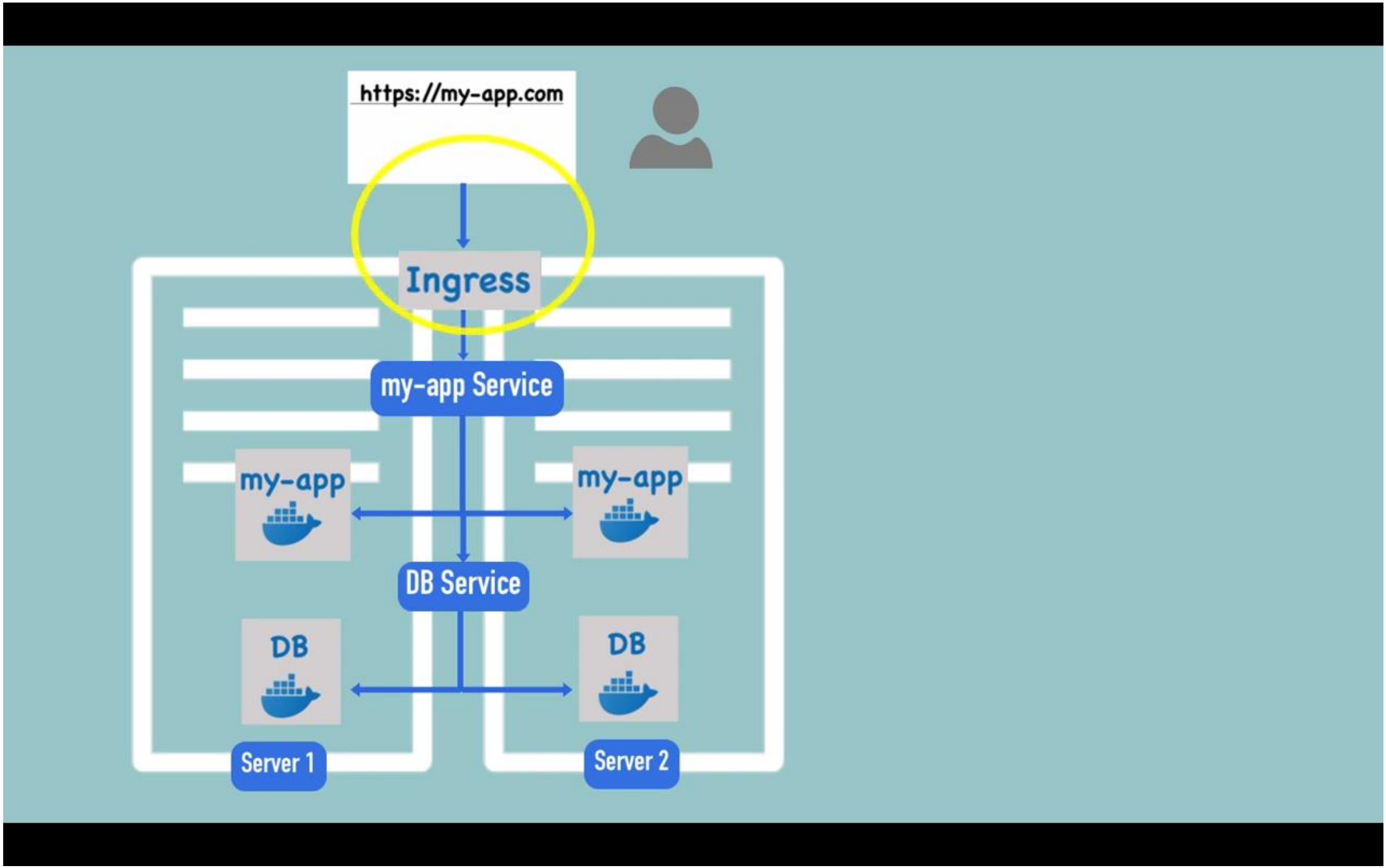

https://my-app.com
Request
Ingress
my-app
my-app
DB
DB
Server 1
Server 2

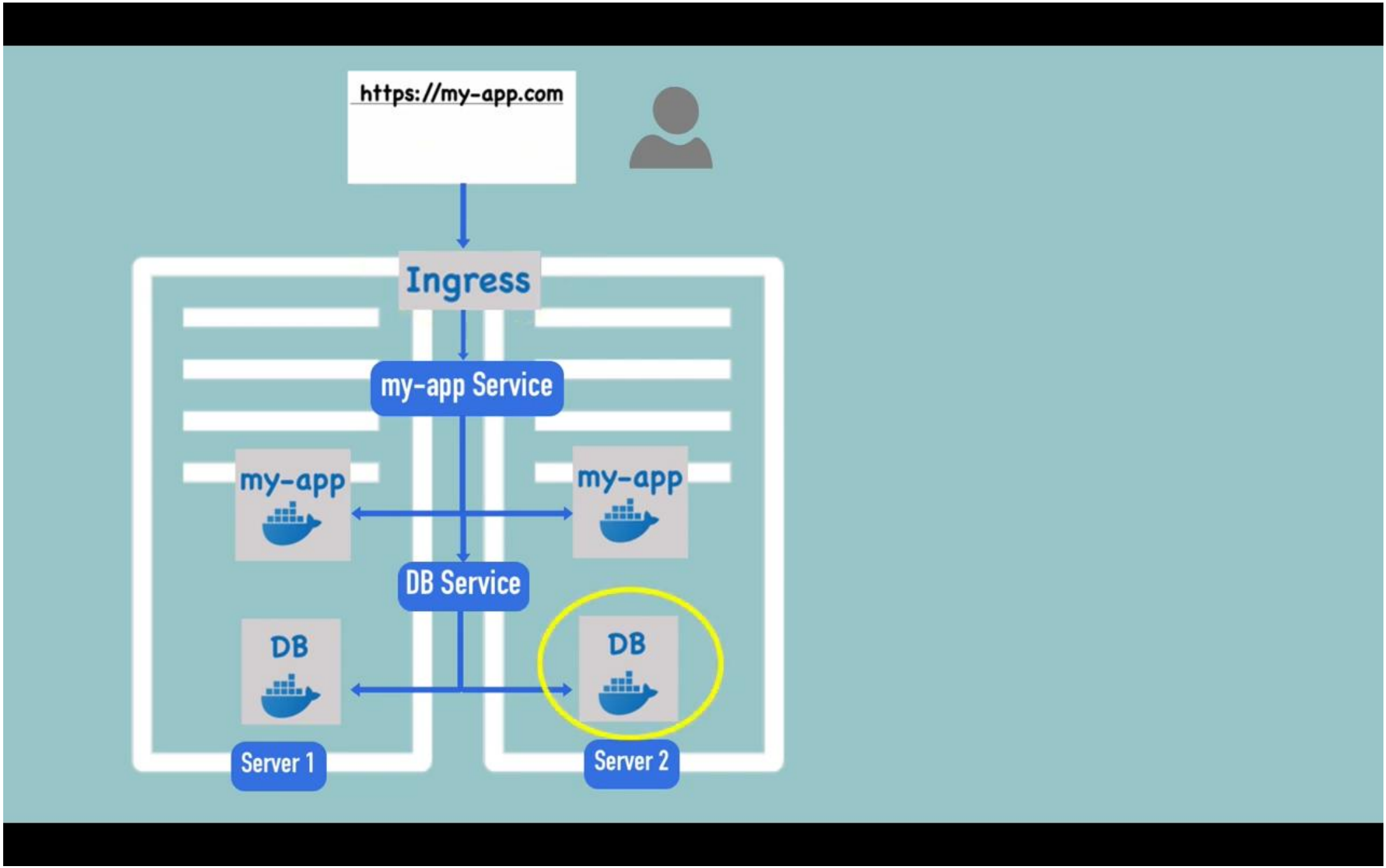

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB
DB
Server 1
Server 2

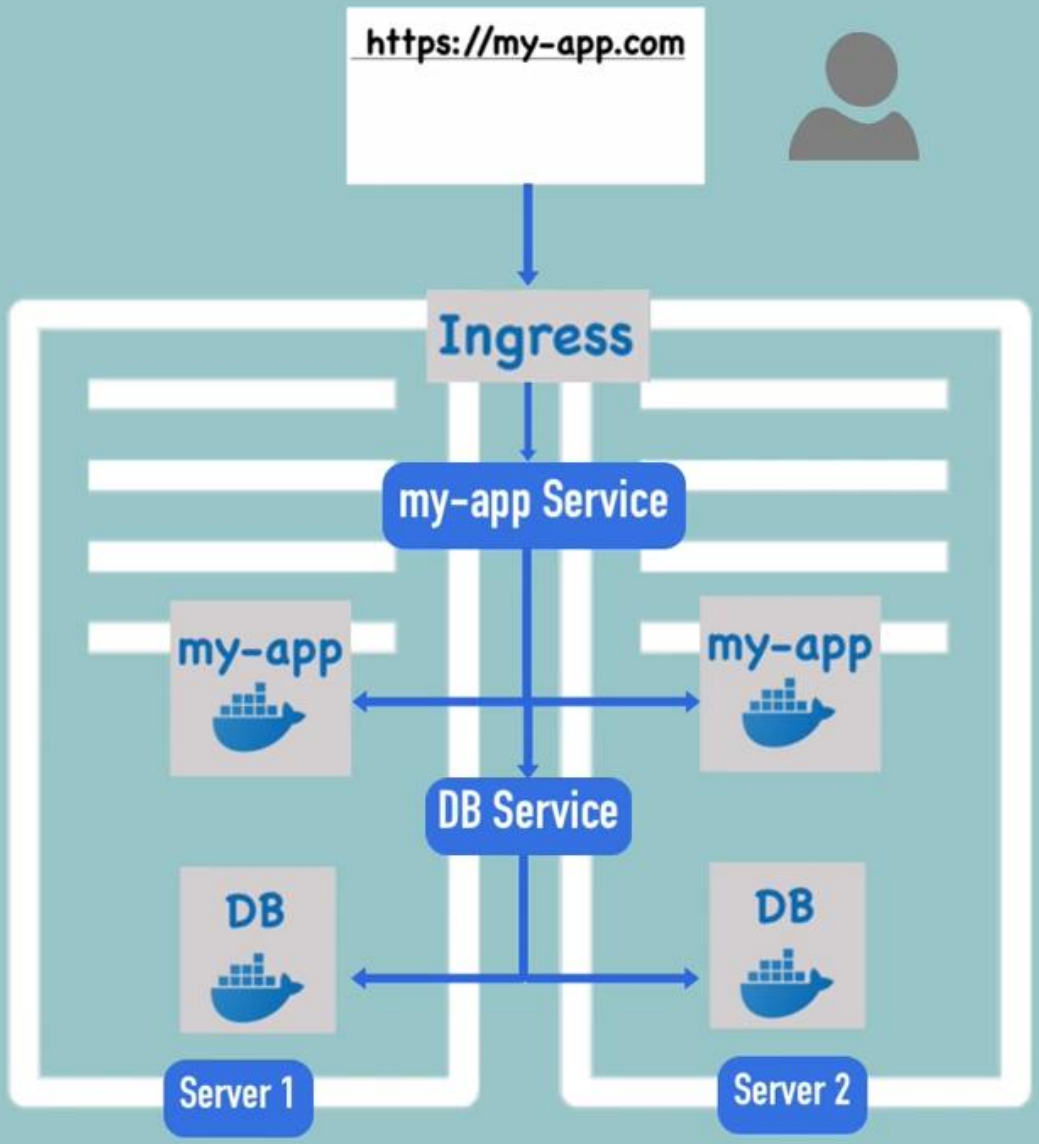

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB
DB
Server 1
Server 2
Service is a load balancer


https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2

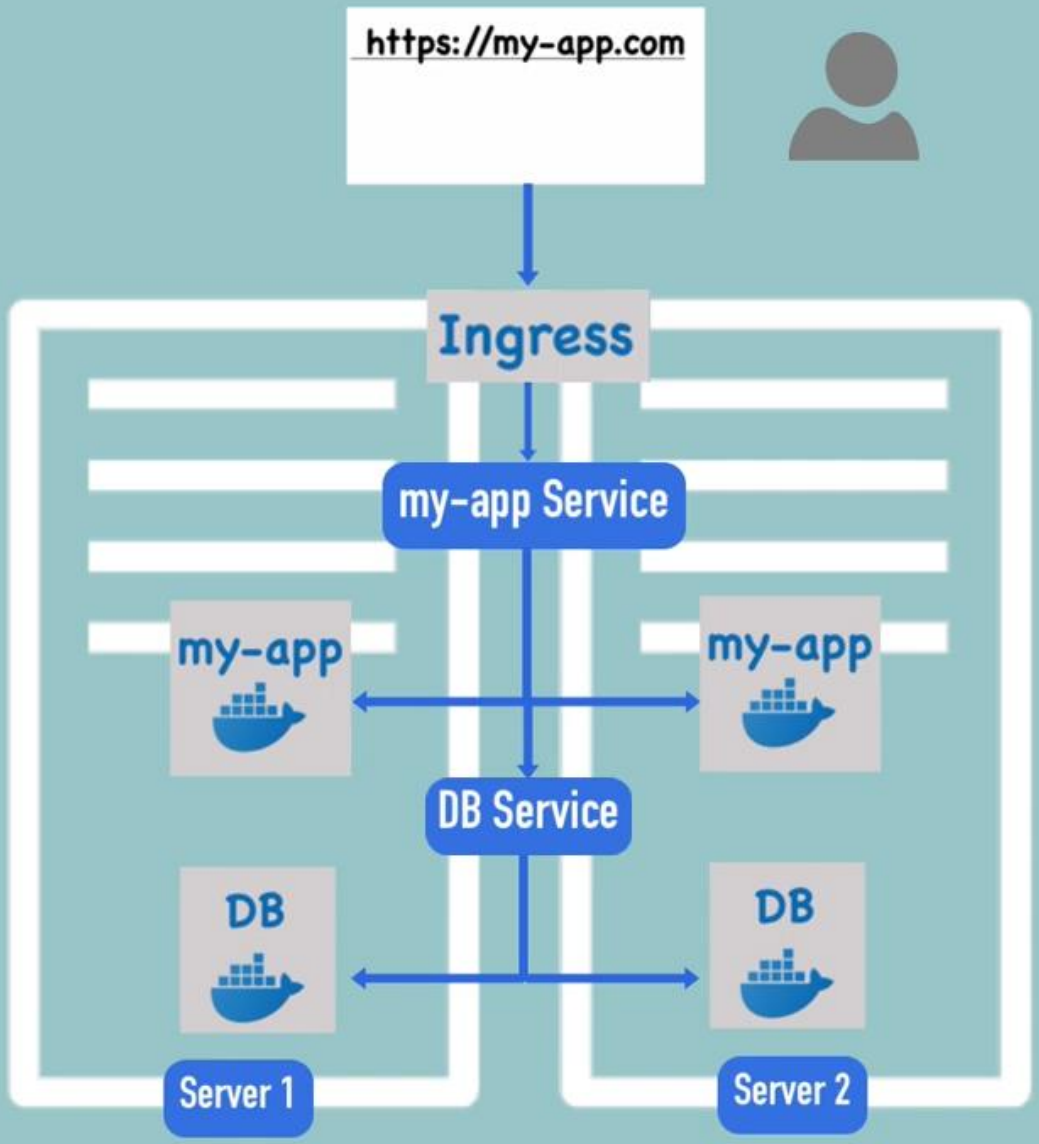

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2


https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2

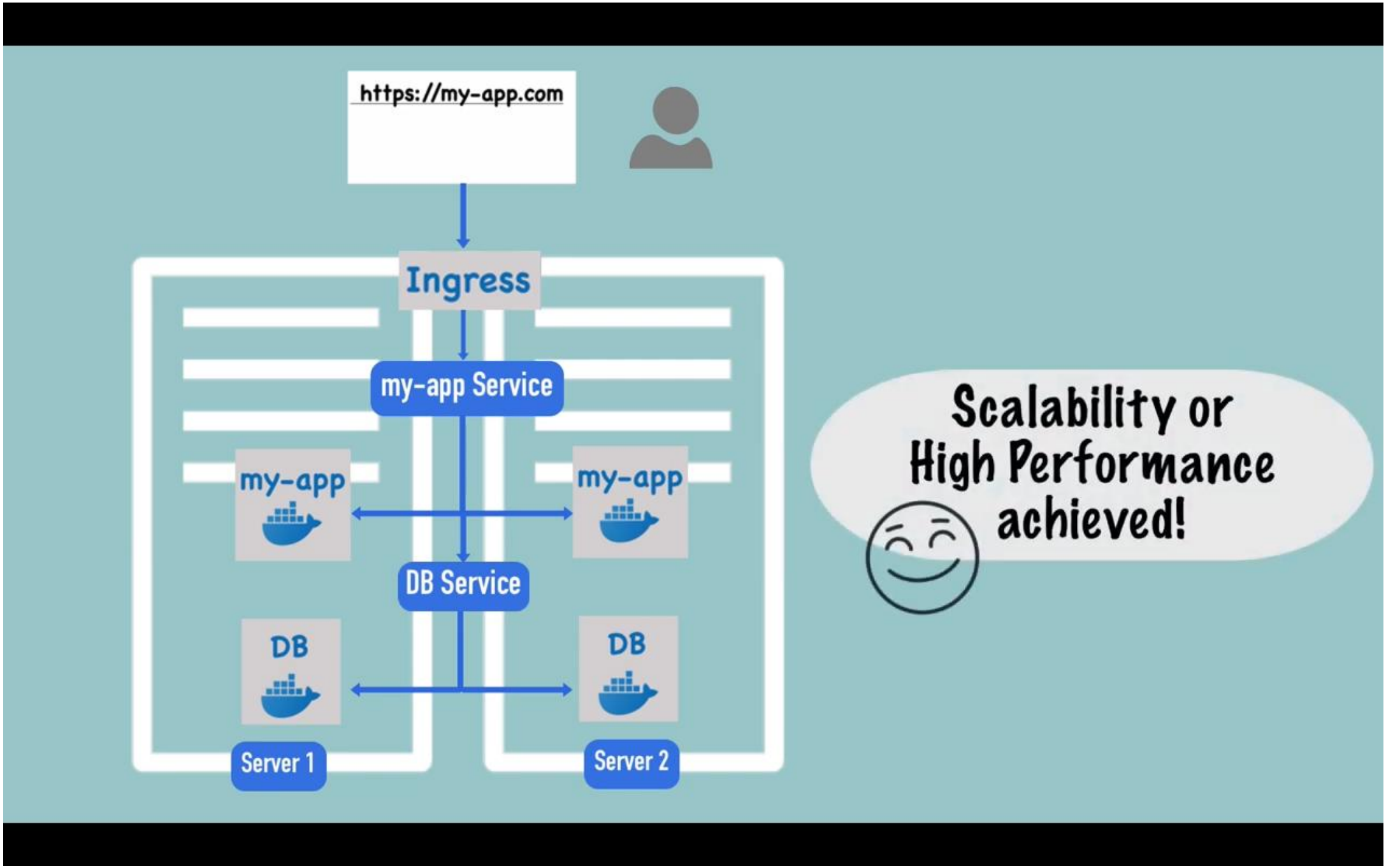

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2

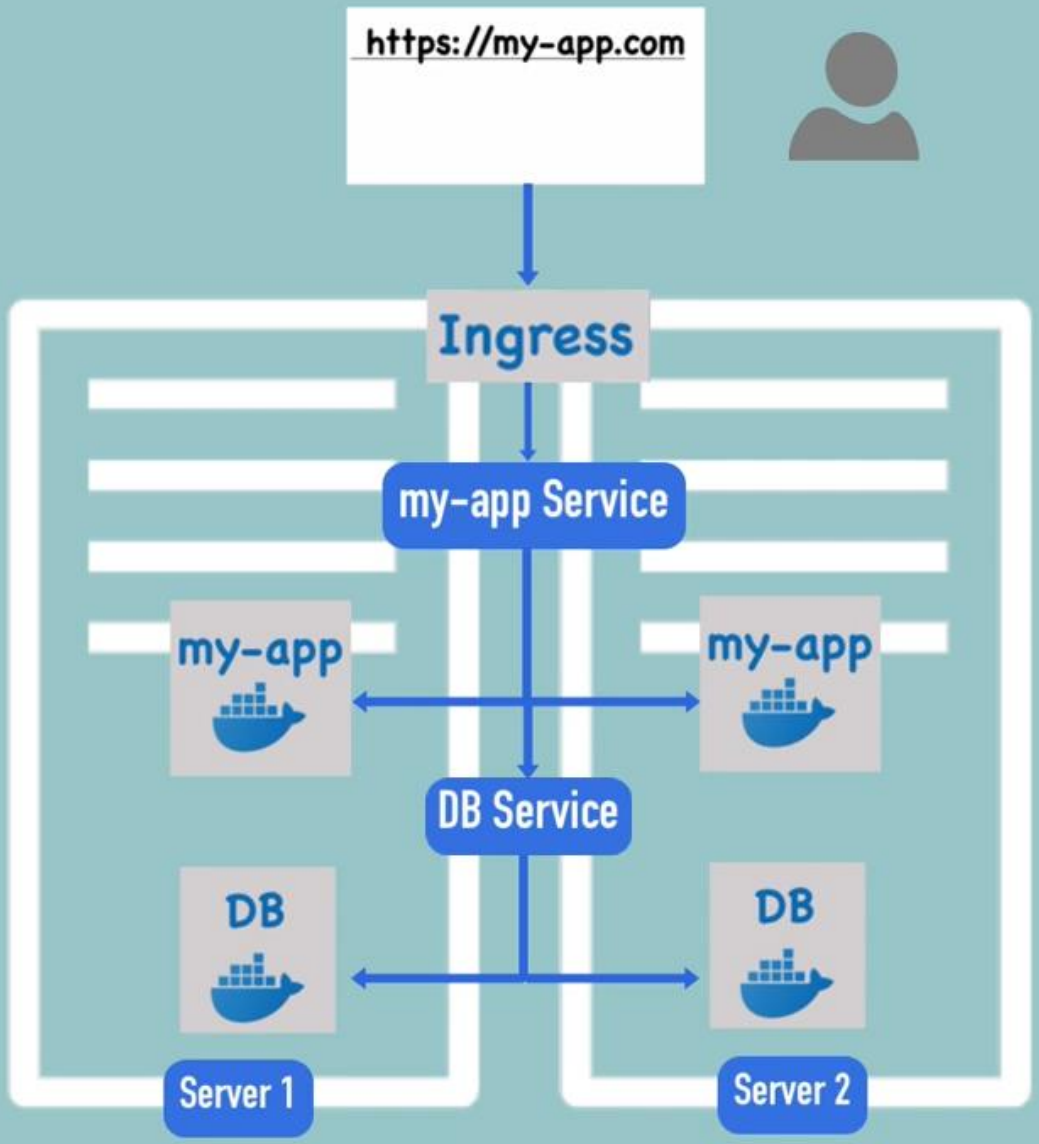


- EVERY component is replicated
- load balanced

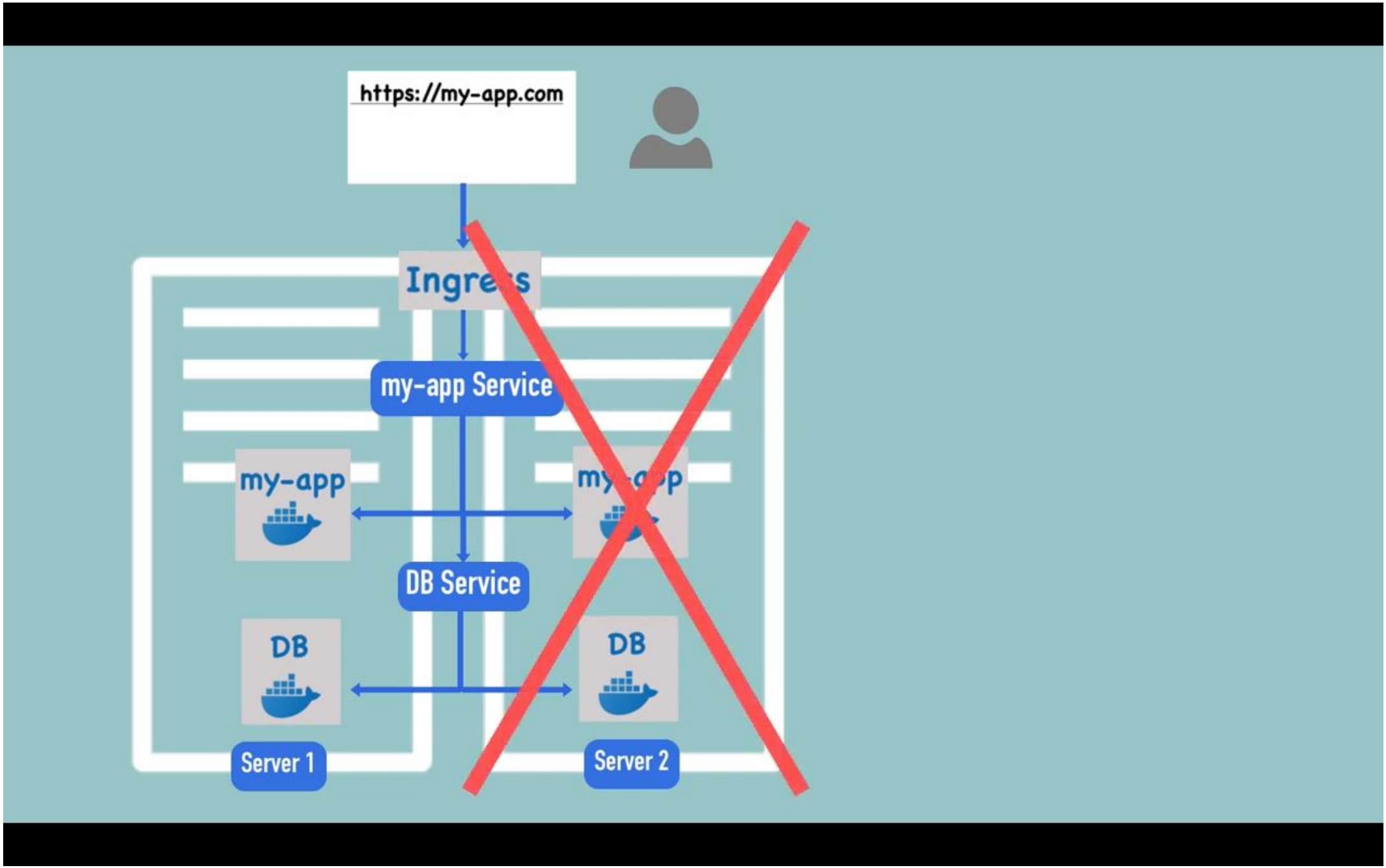


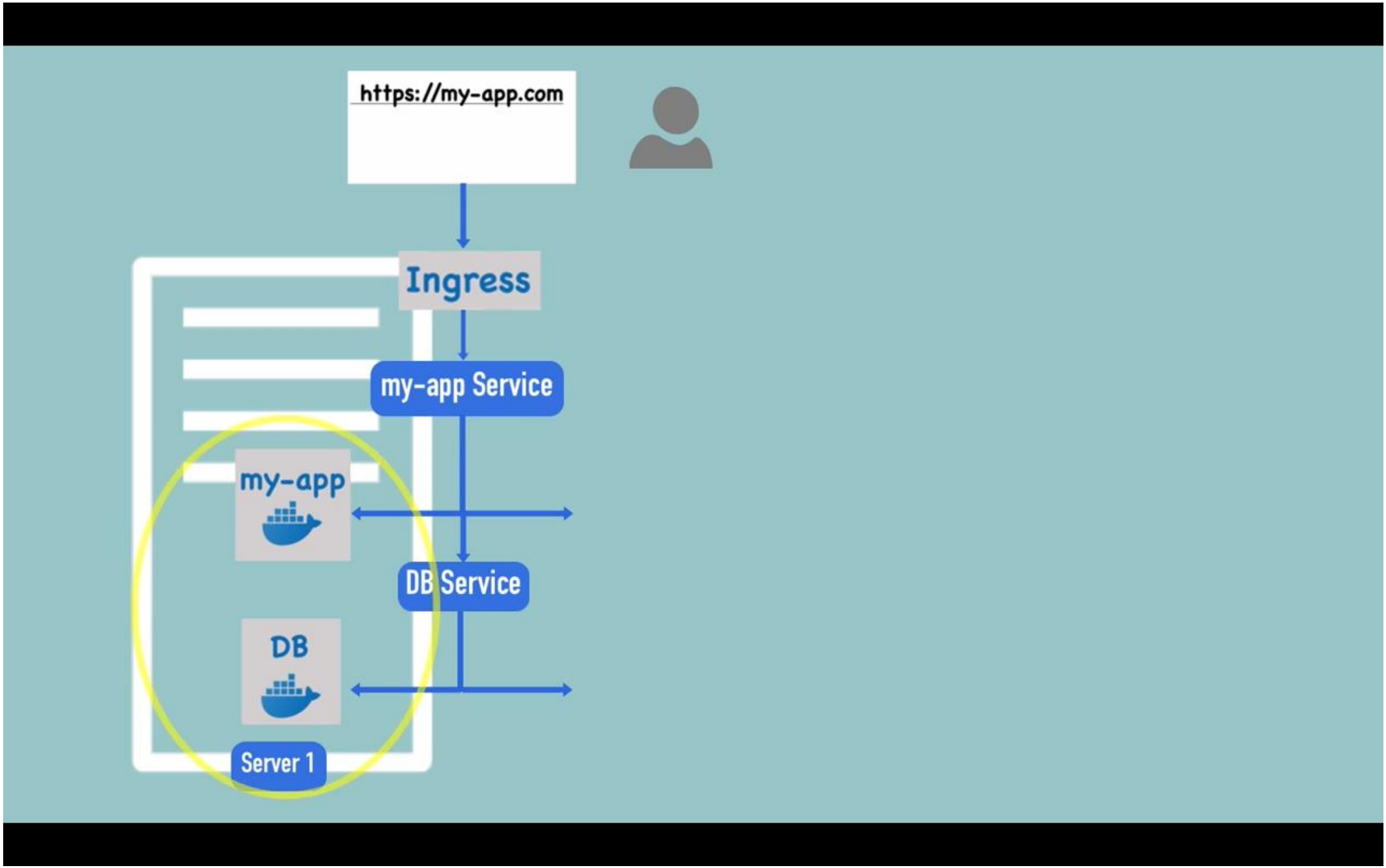
- EVERY component is replicated
- load balanced
- no bottleneck that slows down the responses

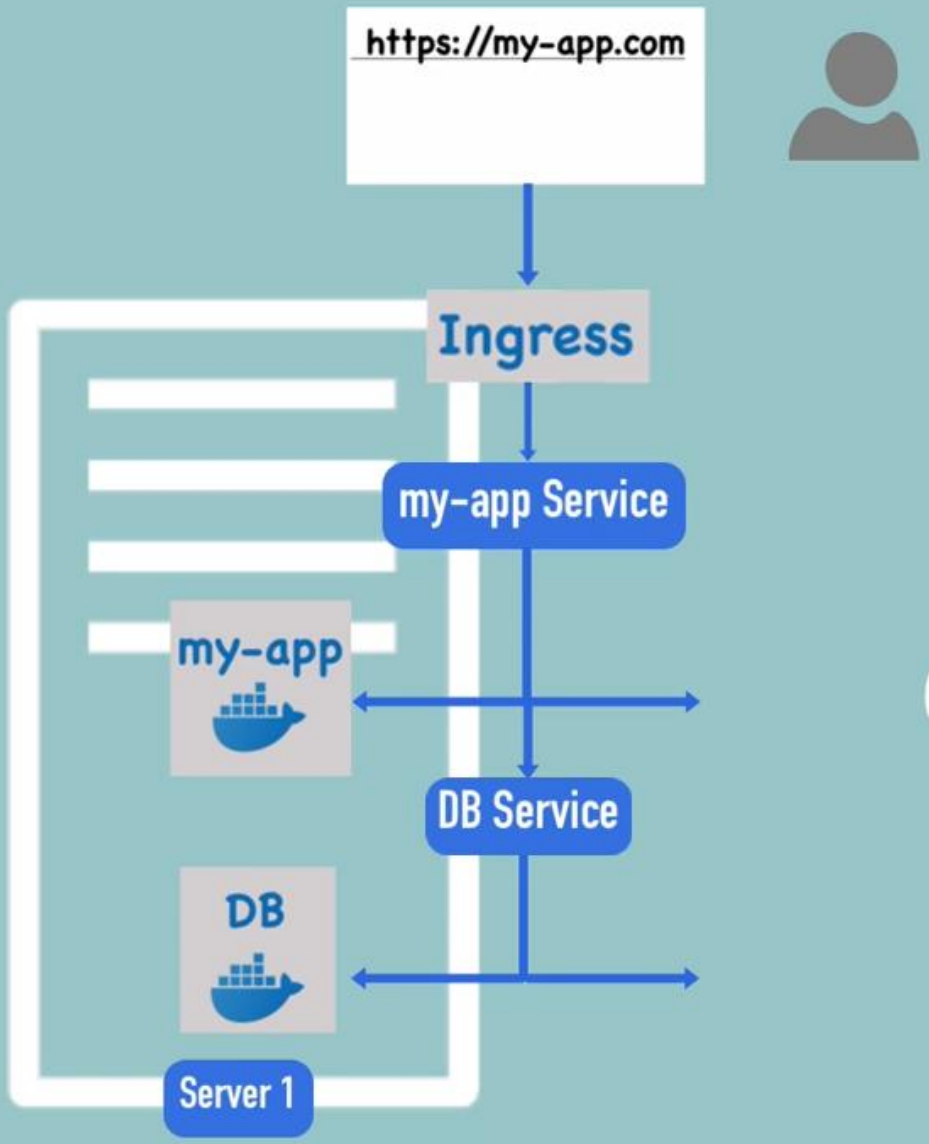

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
Scalability or
High Performance
achieved!



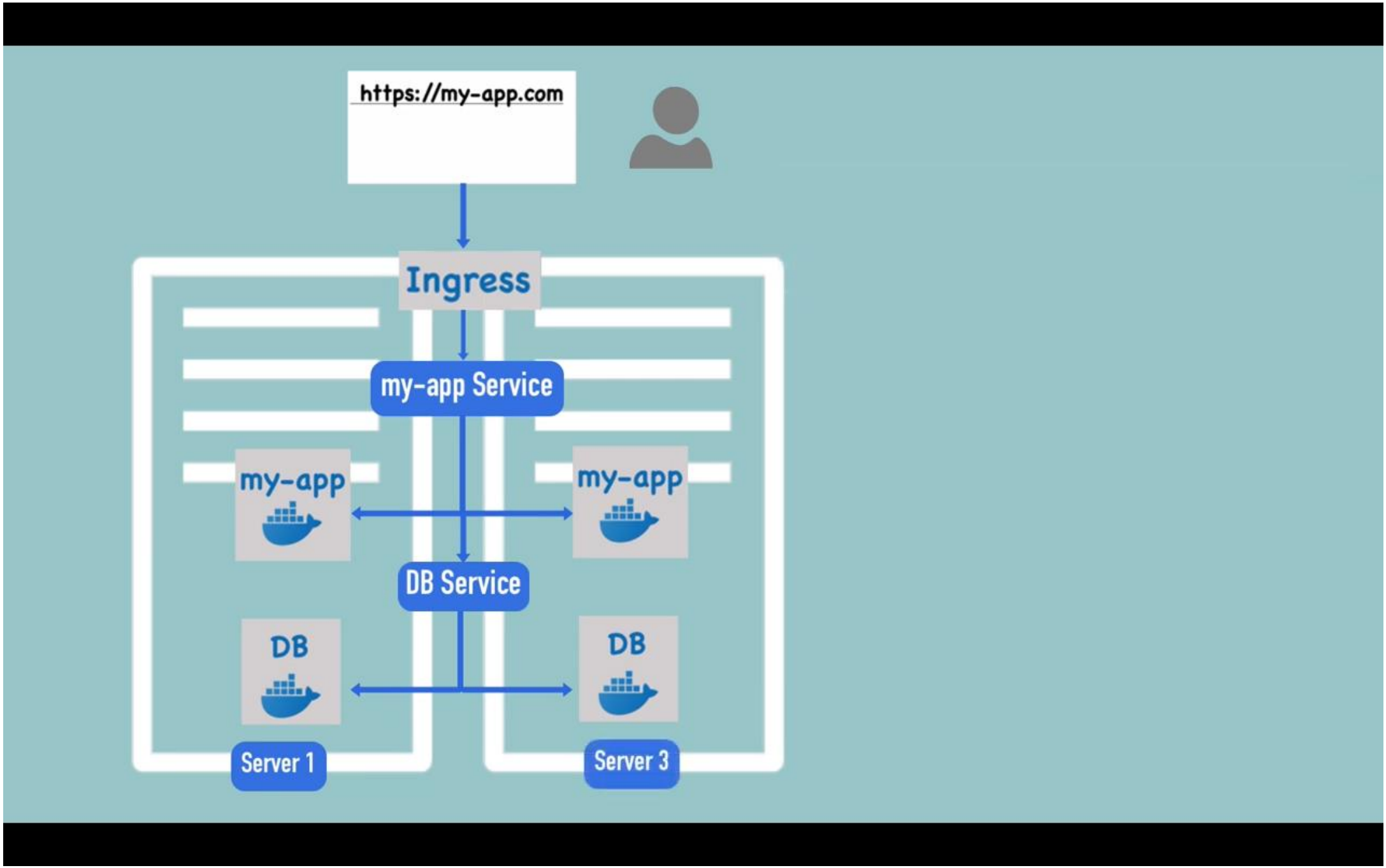
Nonetheless, you need a
properly designed application!

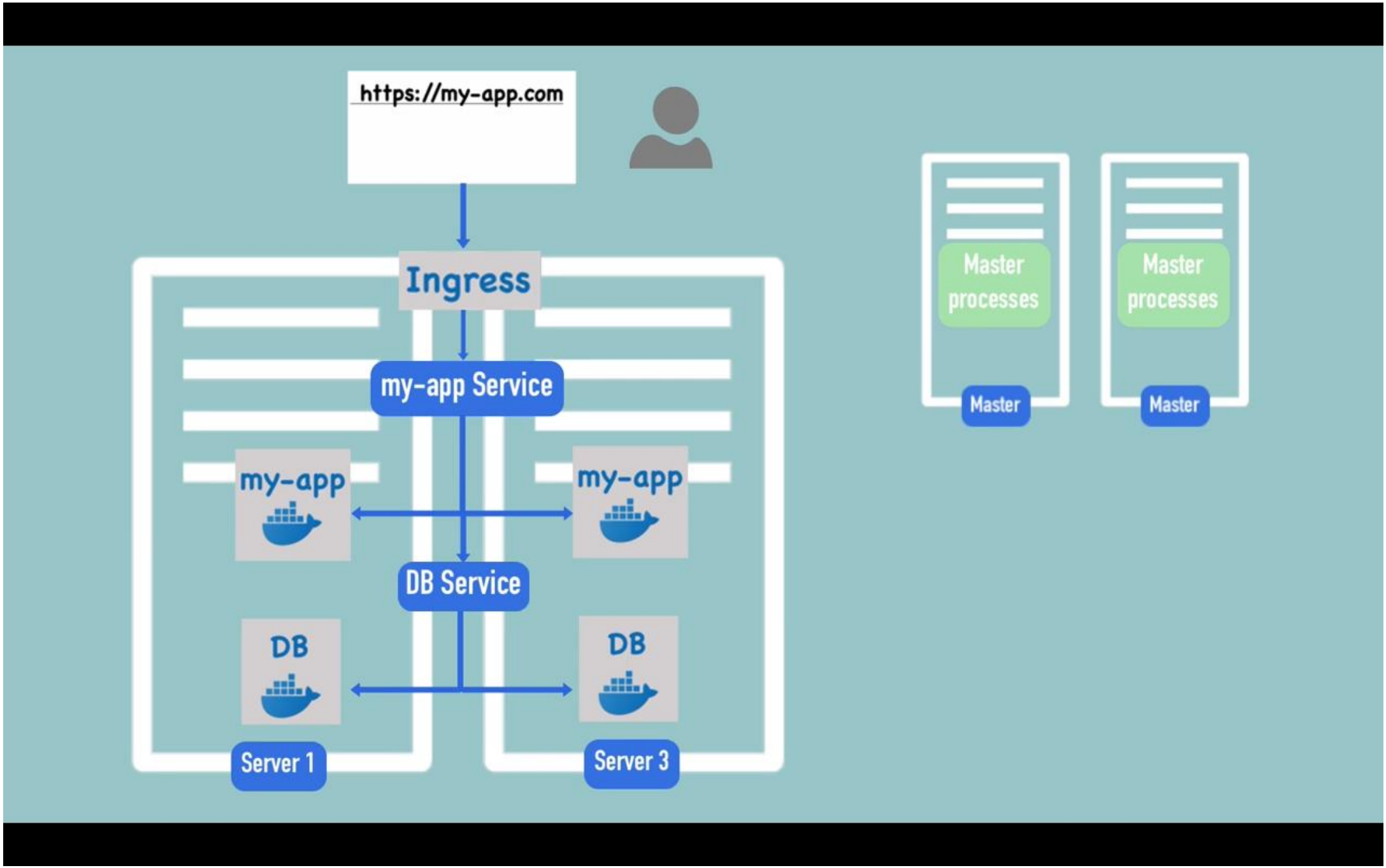

https://my-app.com
Ingress
my-app Service
my-app
DB Service
DB
DB
Server 1
Server 2

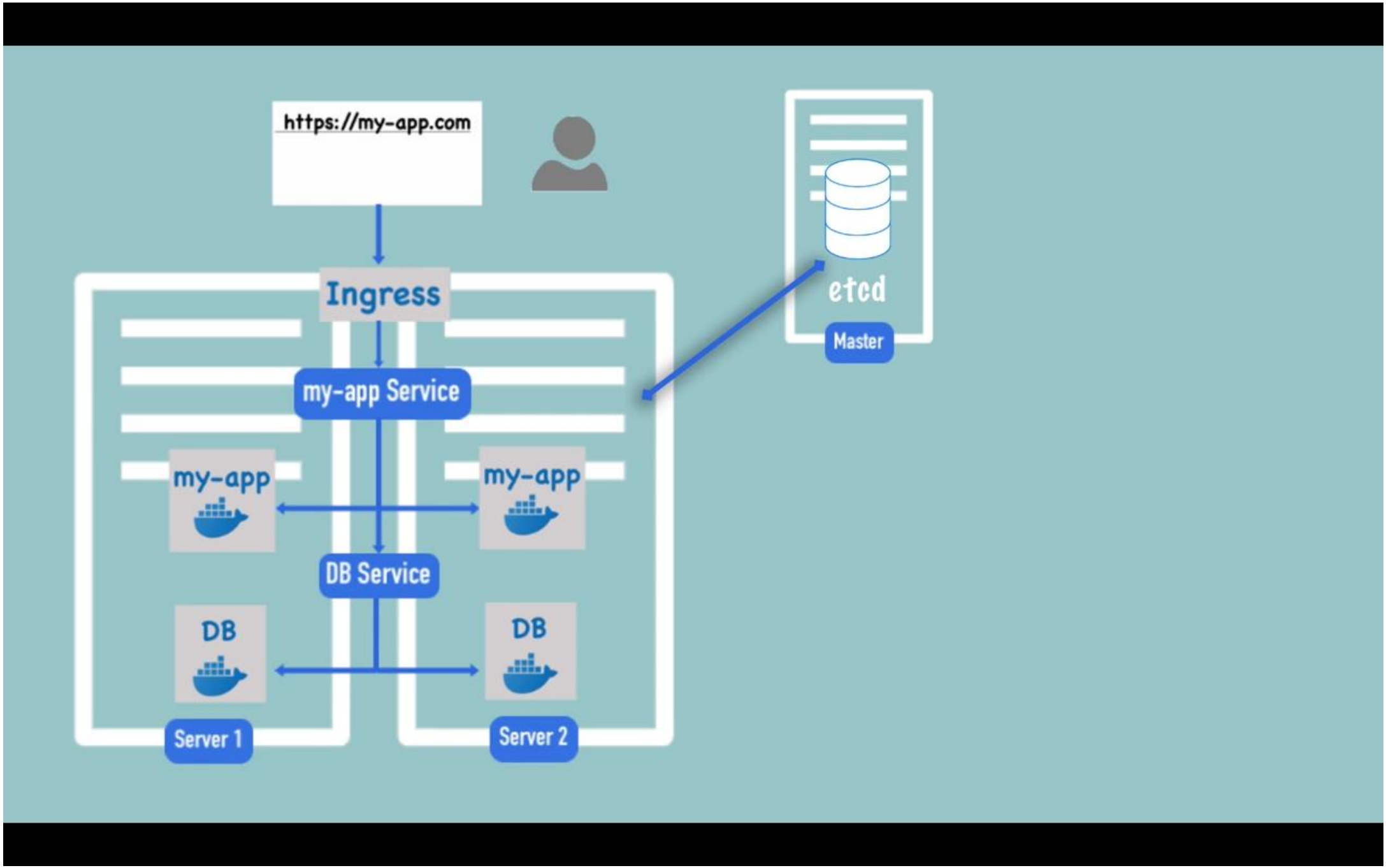

https://my-app.com
Ingress
my-app Service
my-app
DB Service
DB
Server 1

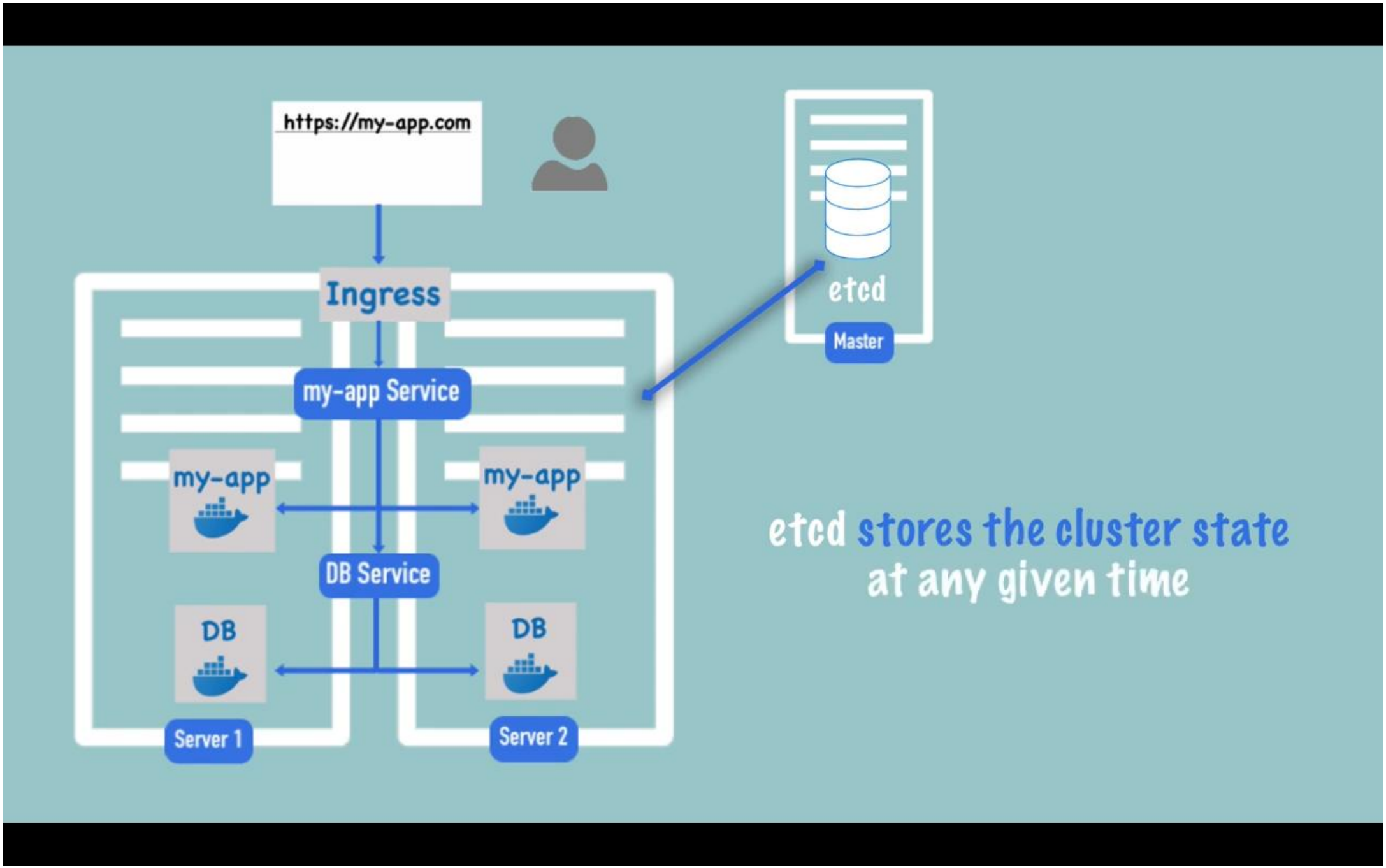


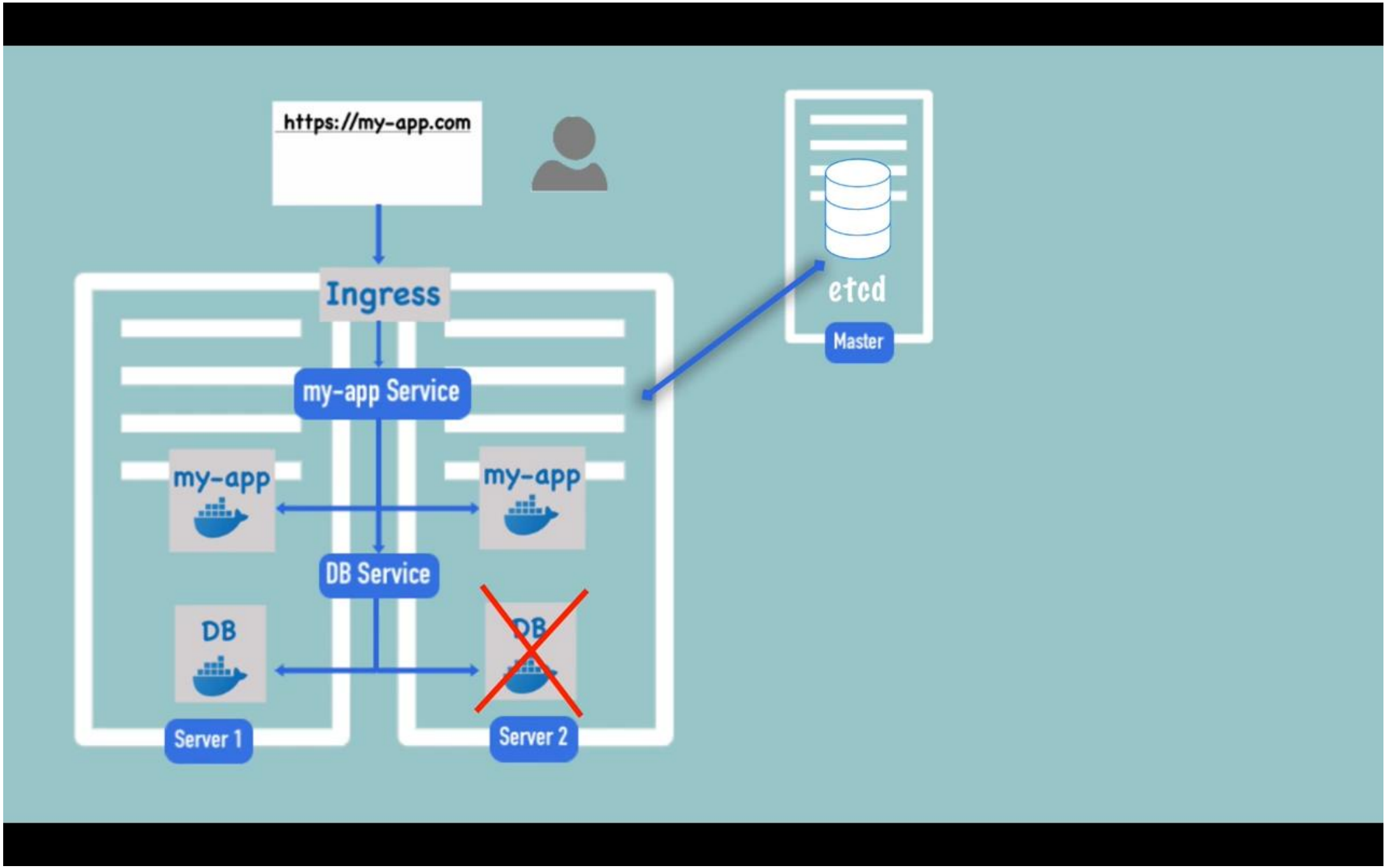
Controller Manager schedules new replicas

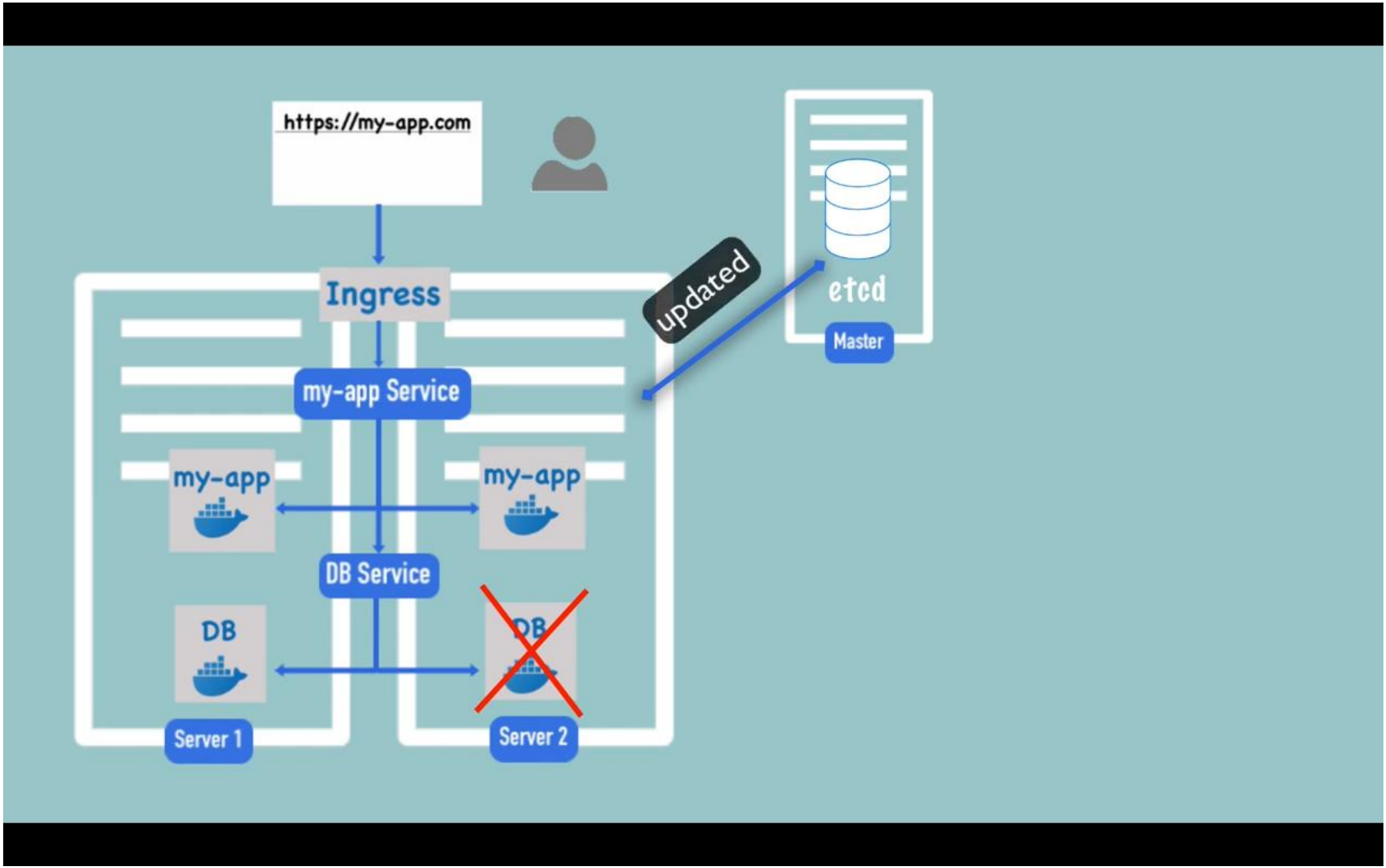

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 3

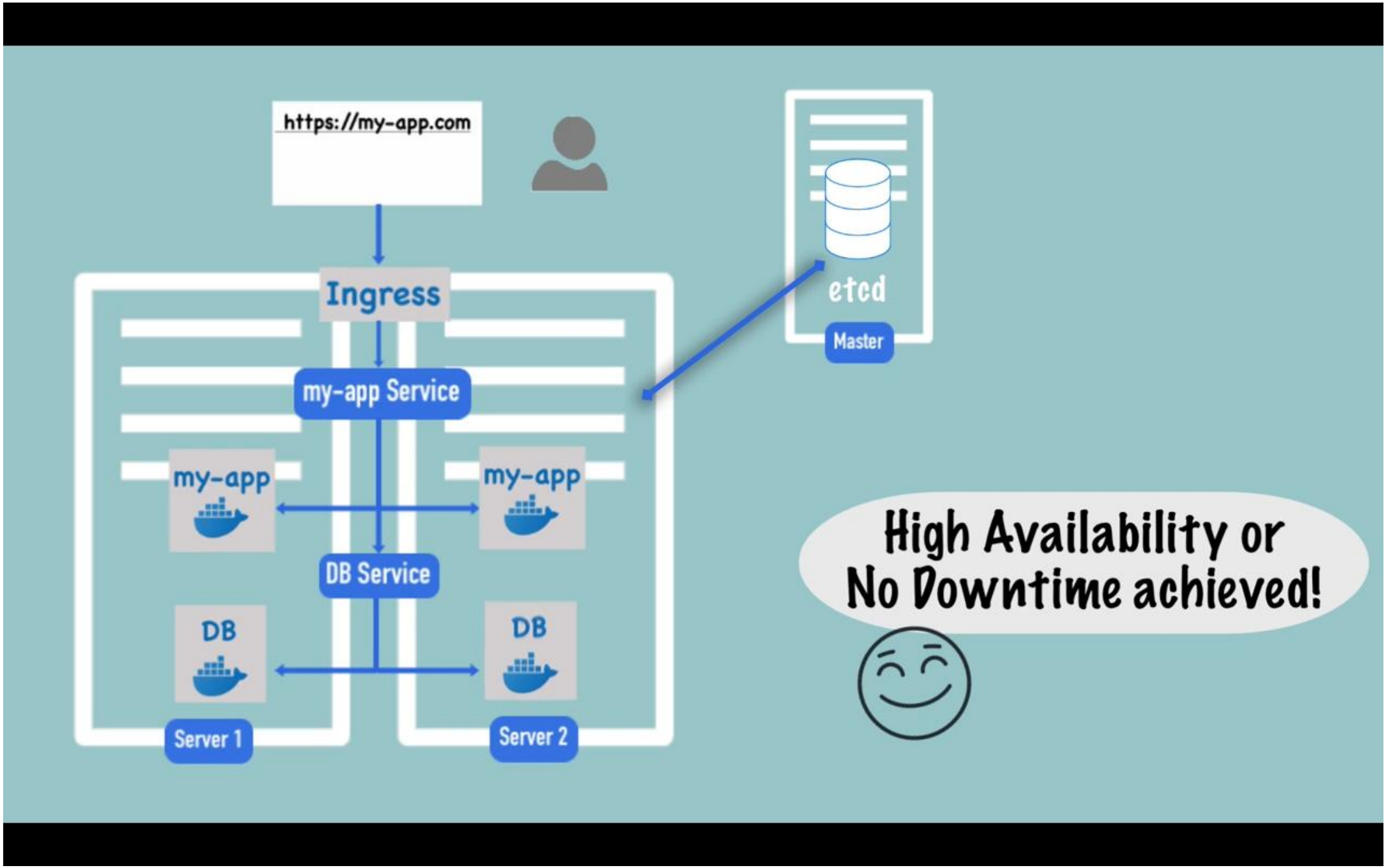

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 3
Master processes
Master processes
Master
Master

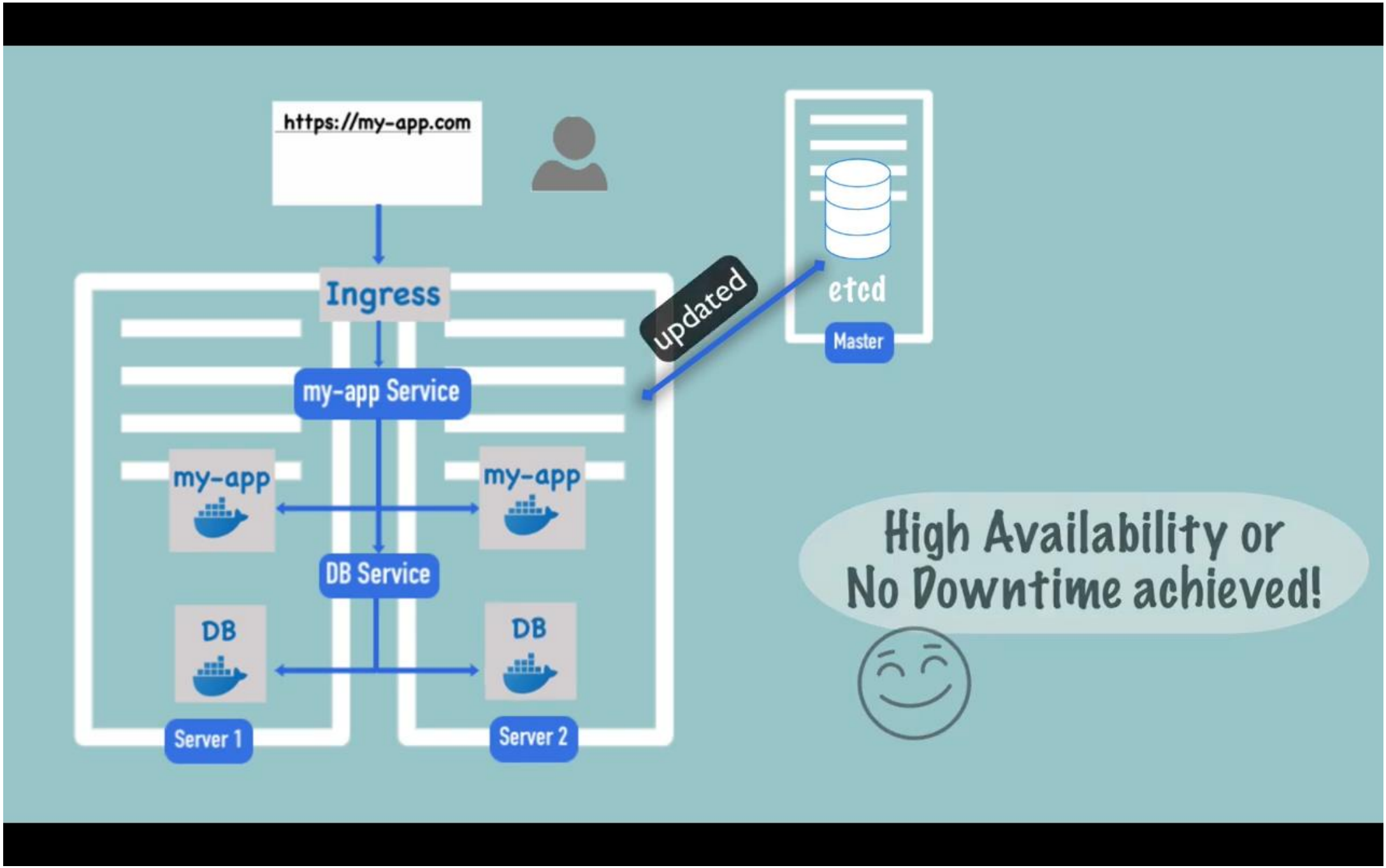

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master


https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
etcd stores the cluster state
at any given time

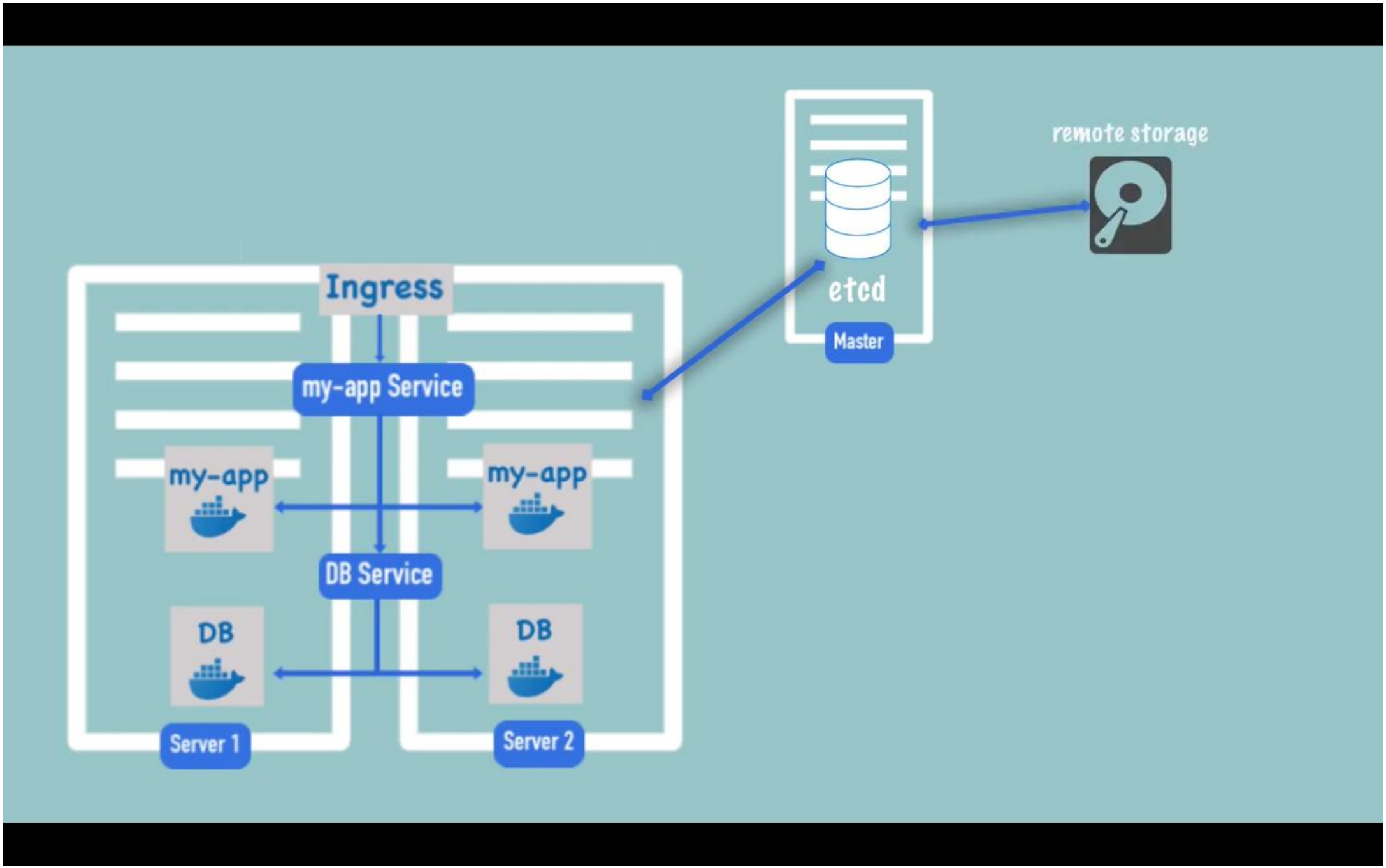

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master

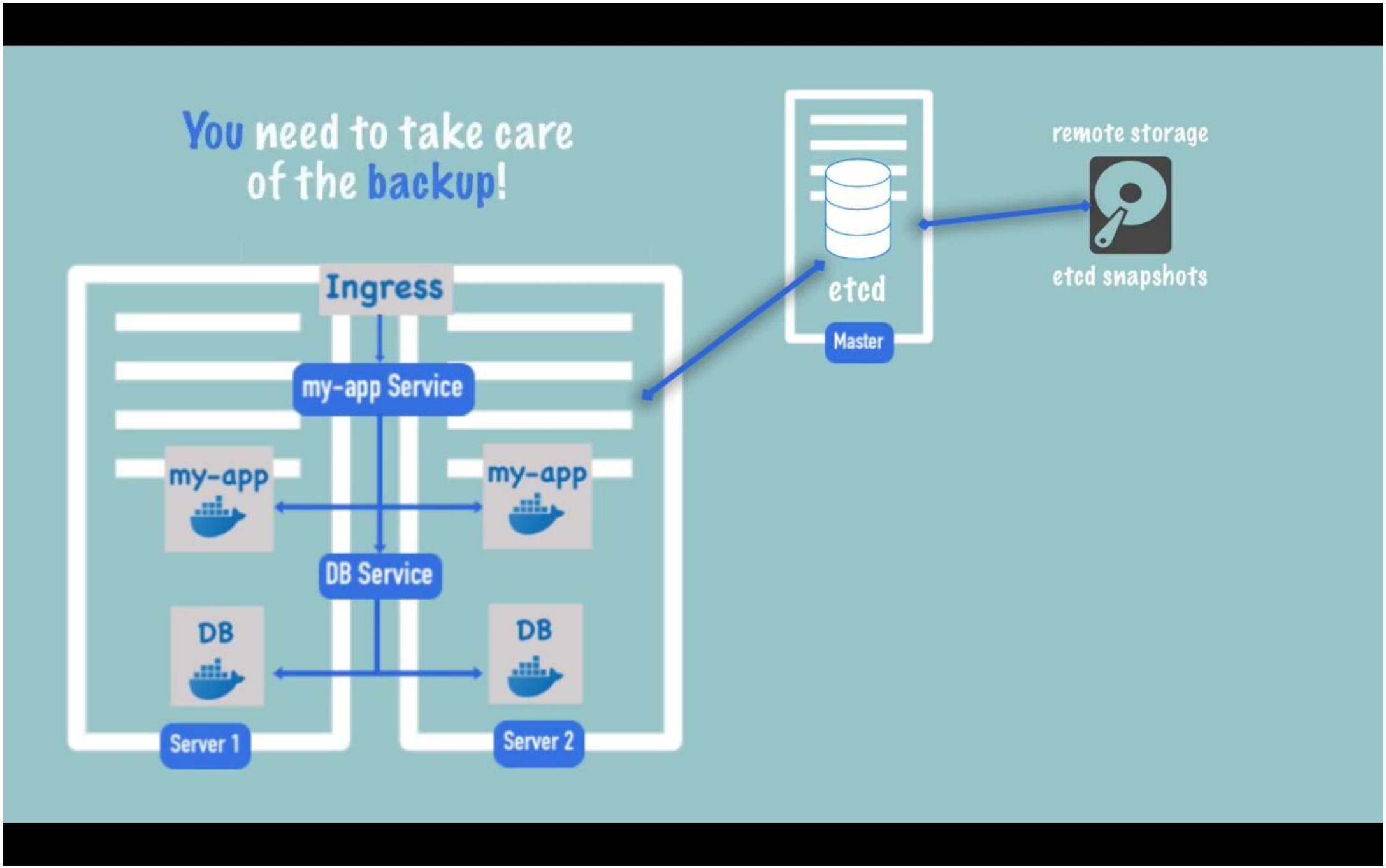

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
updated
etcd
Master

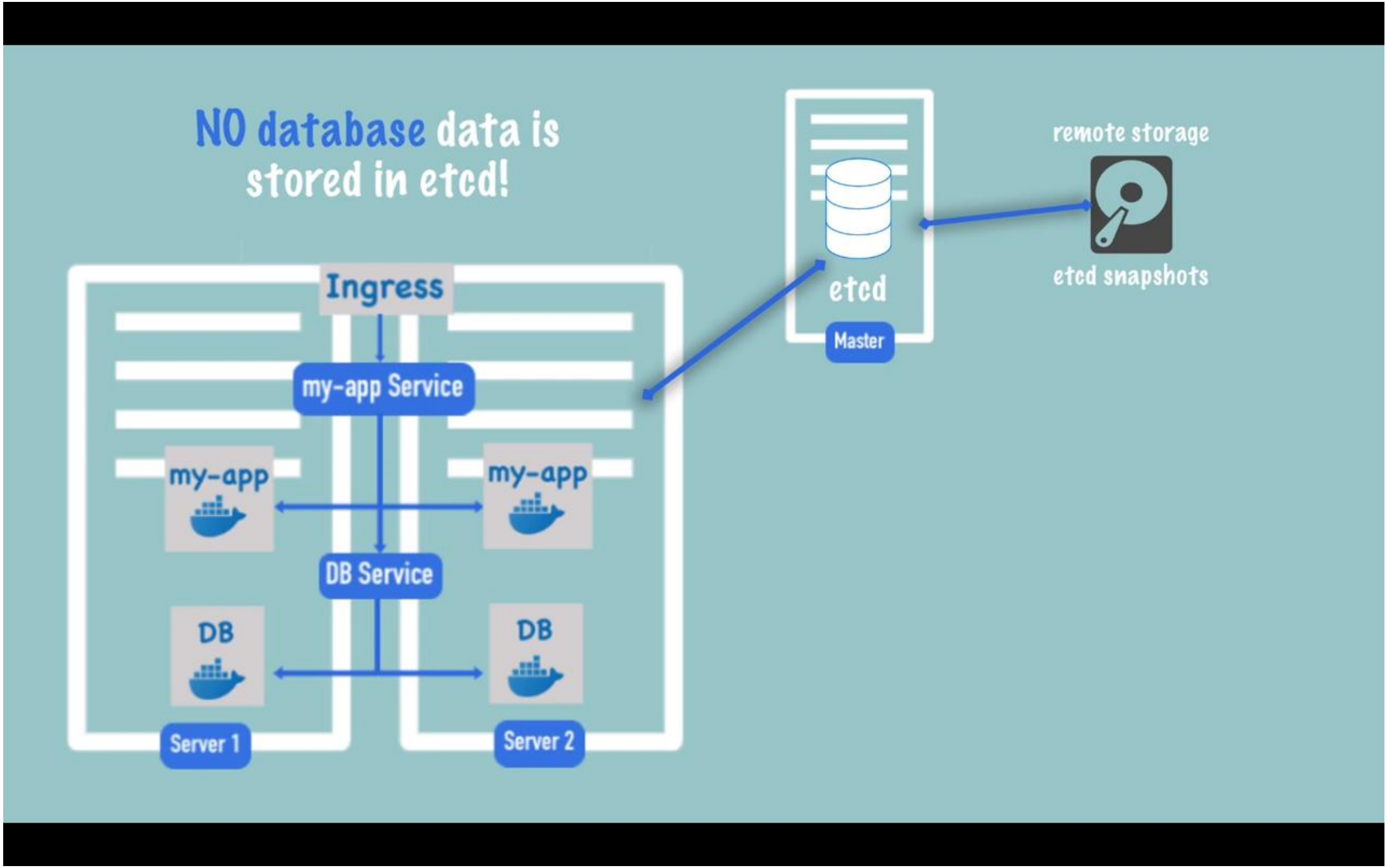

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
High Availability or
No Downtime achieved!

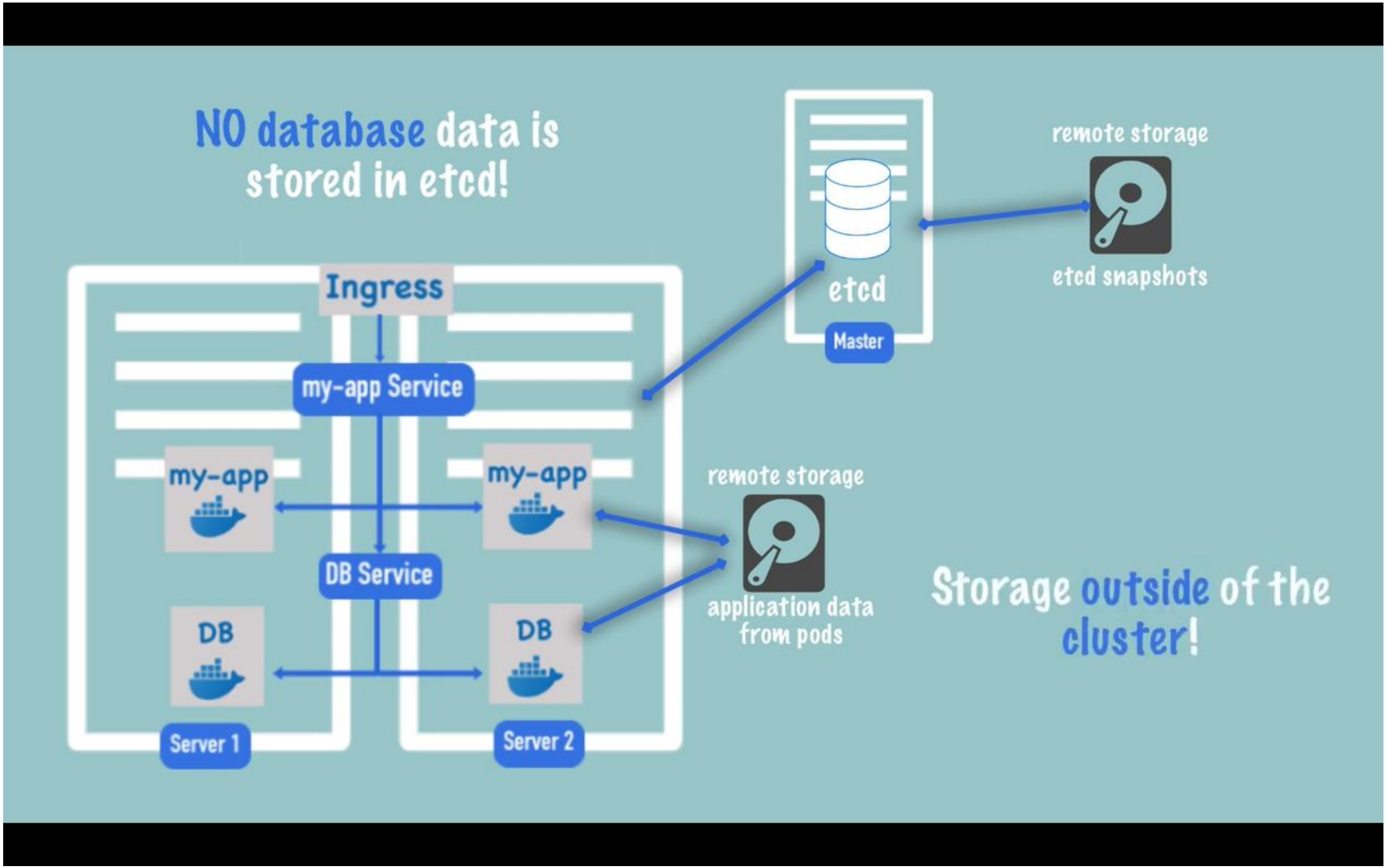

https://my-app.com
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
updated
etcd
Master
High Availability or
No Downtime achieved!

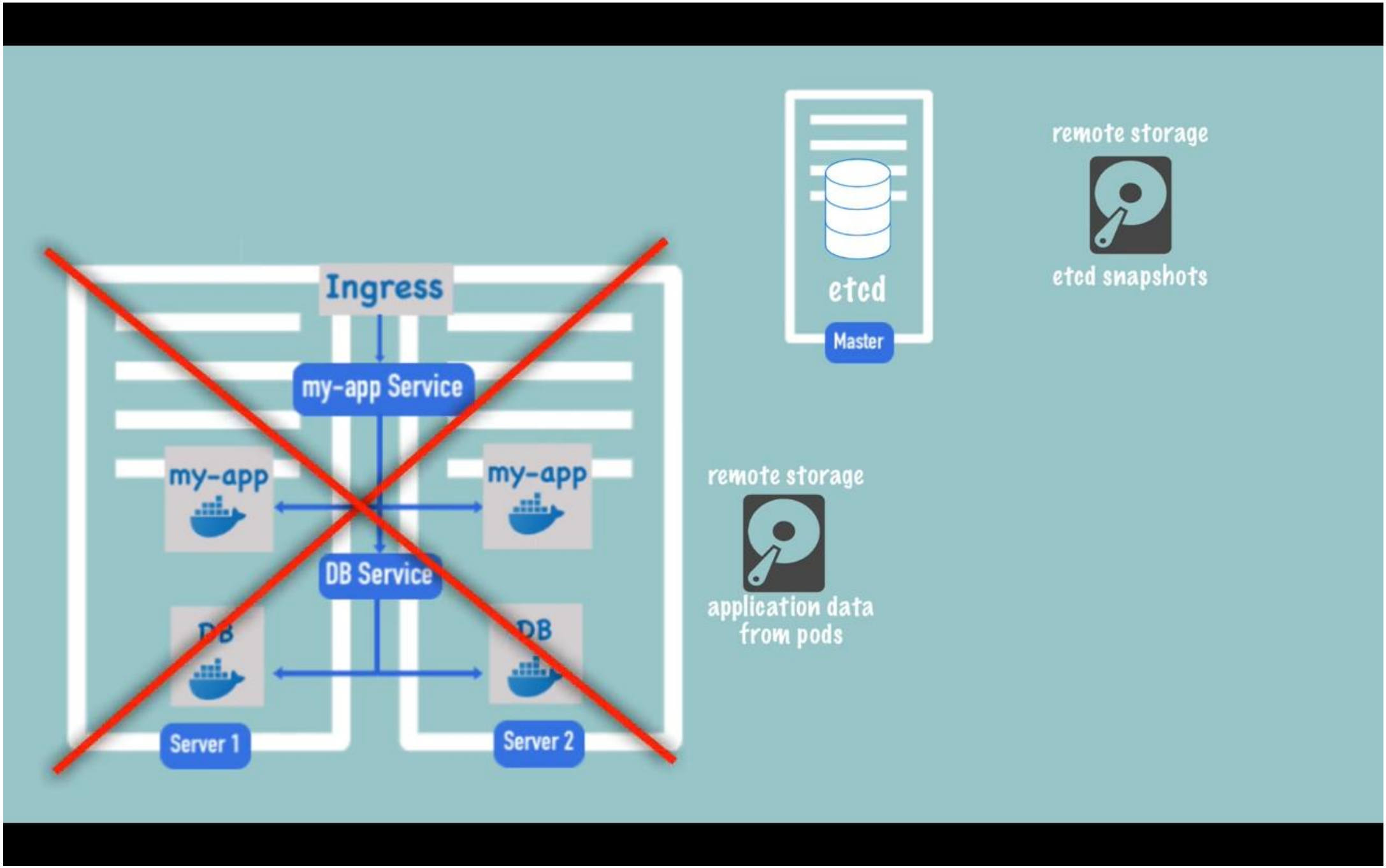
Disaster recovery

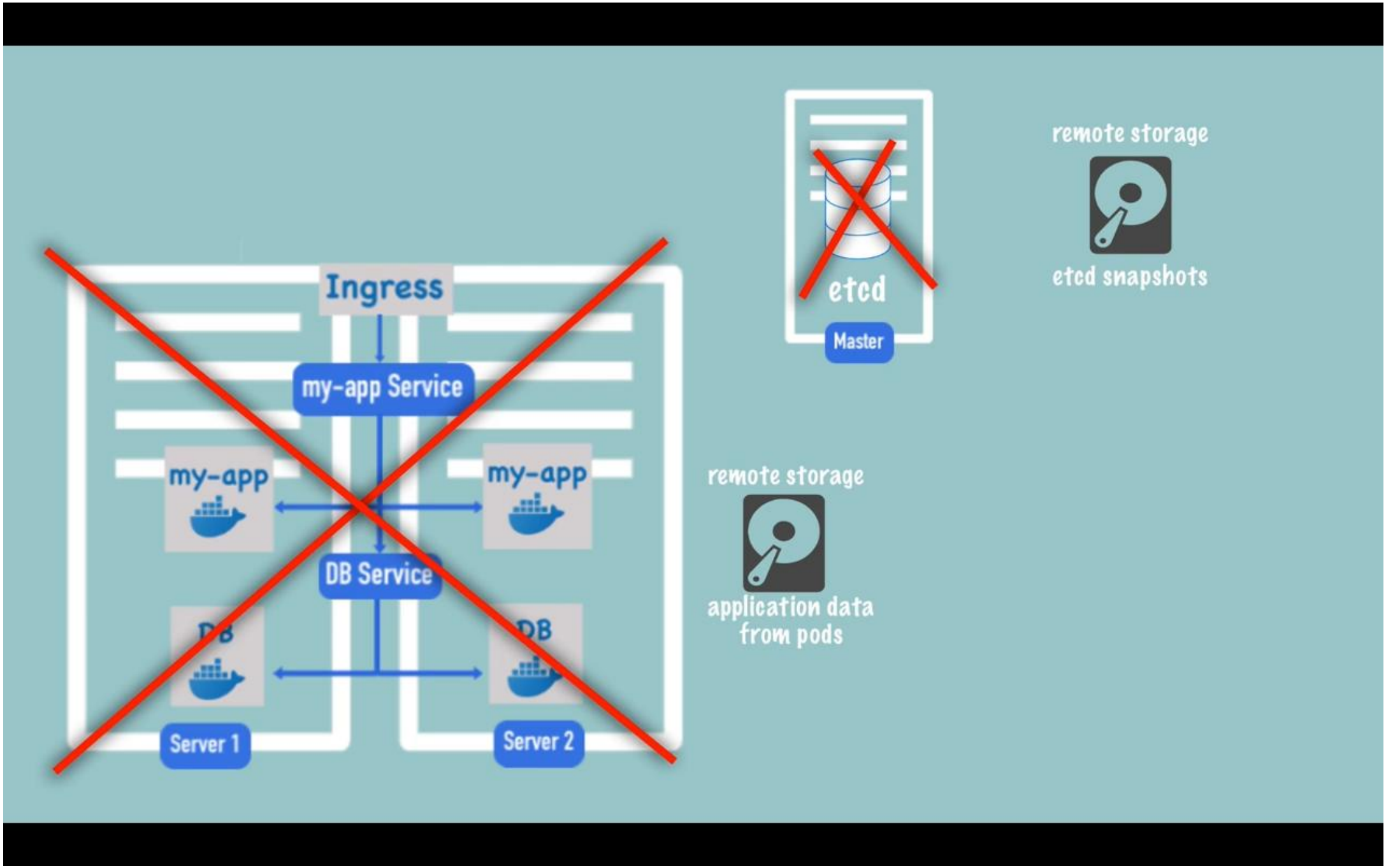

remote storage
etcd
Master
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2

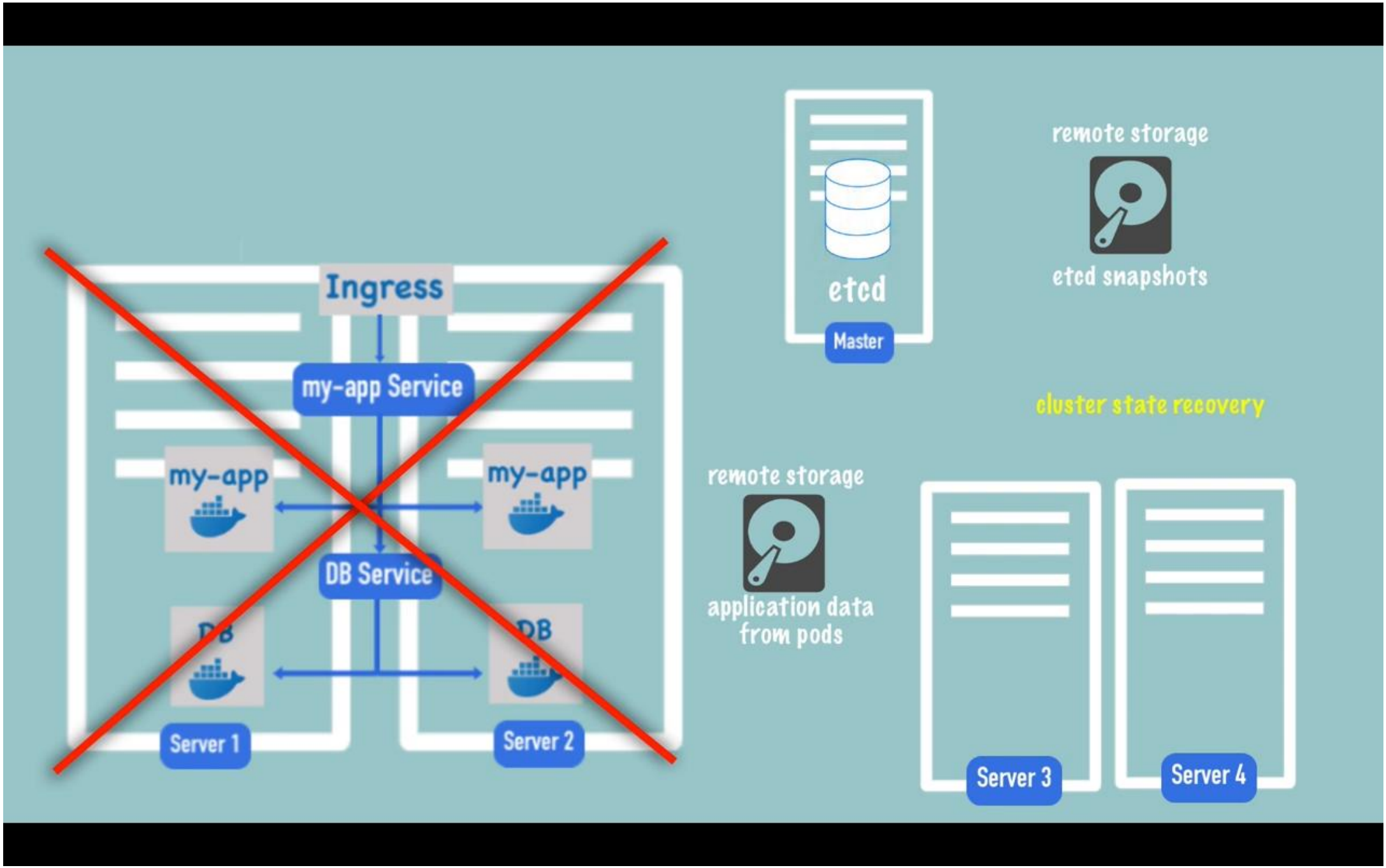

You need to take care of the backup!
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots

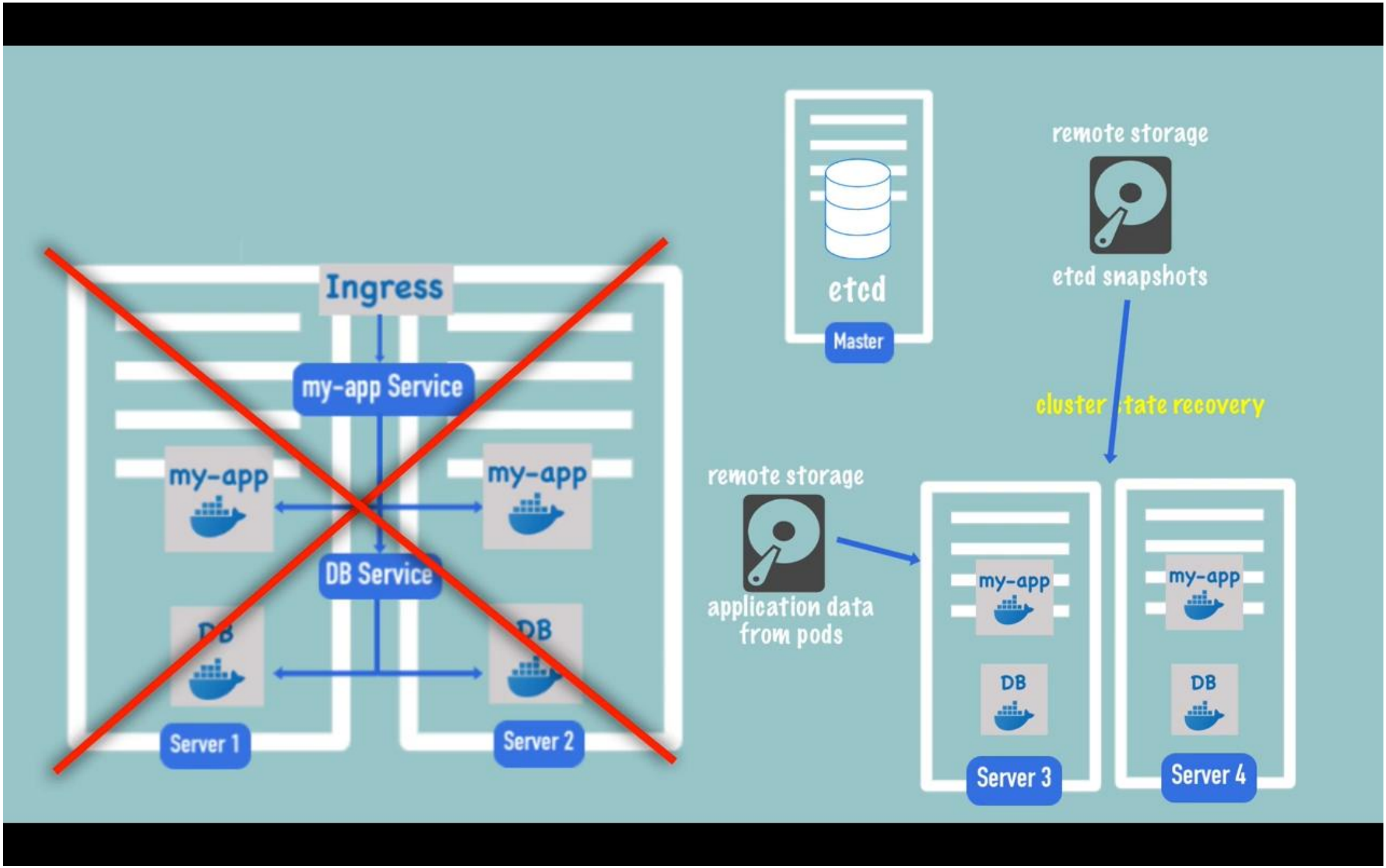

NO database data is stored in etcd!
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots

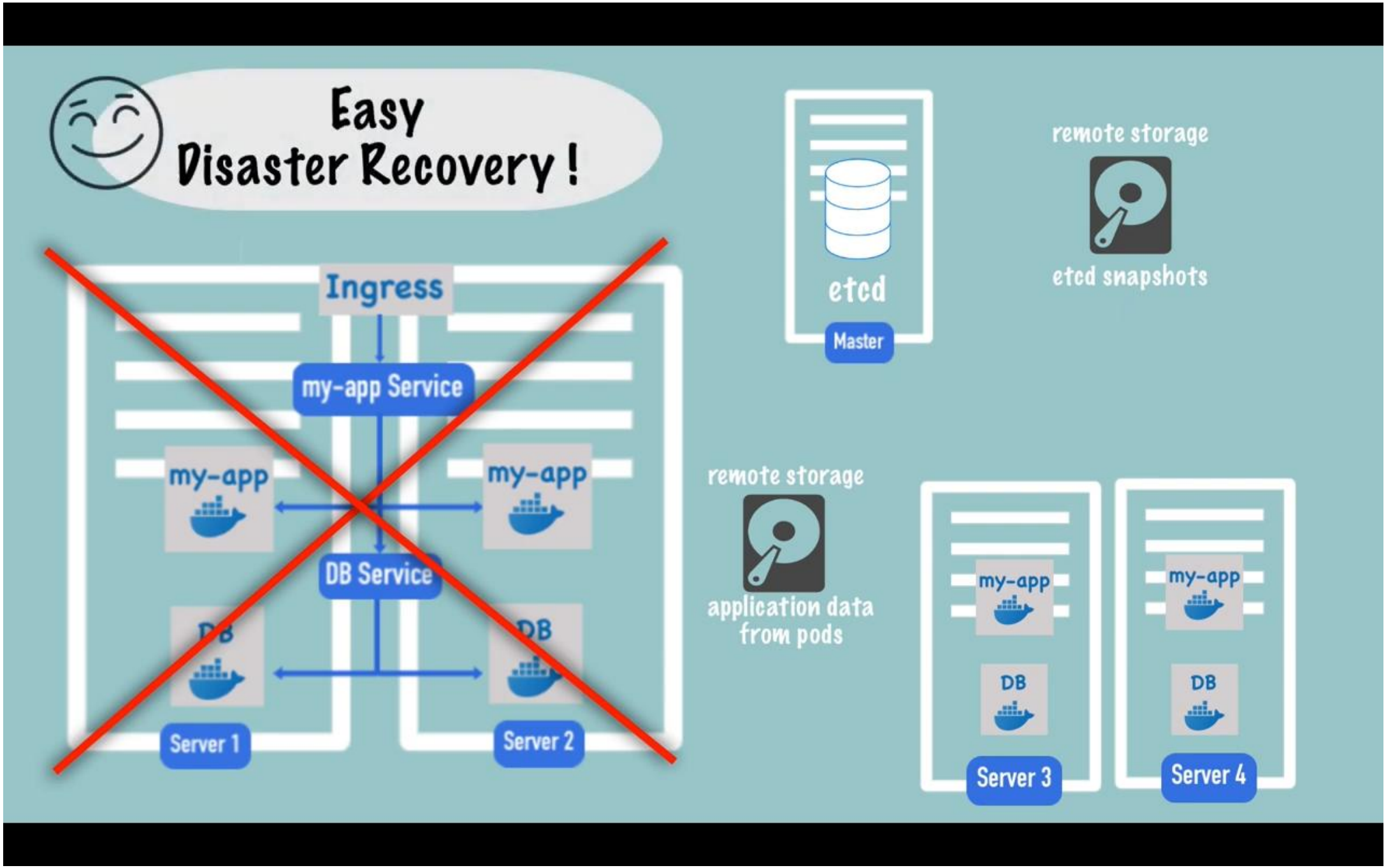

NO database data is stored in etcd!
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
remote storage
application data from pods
Storage outside of the cluster!

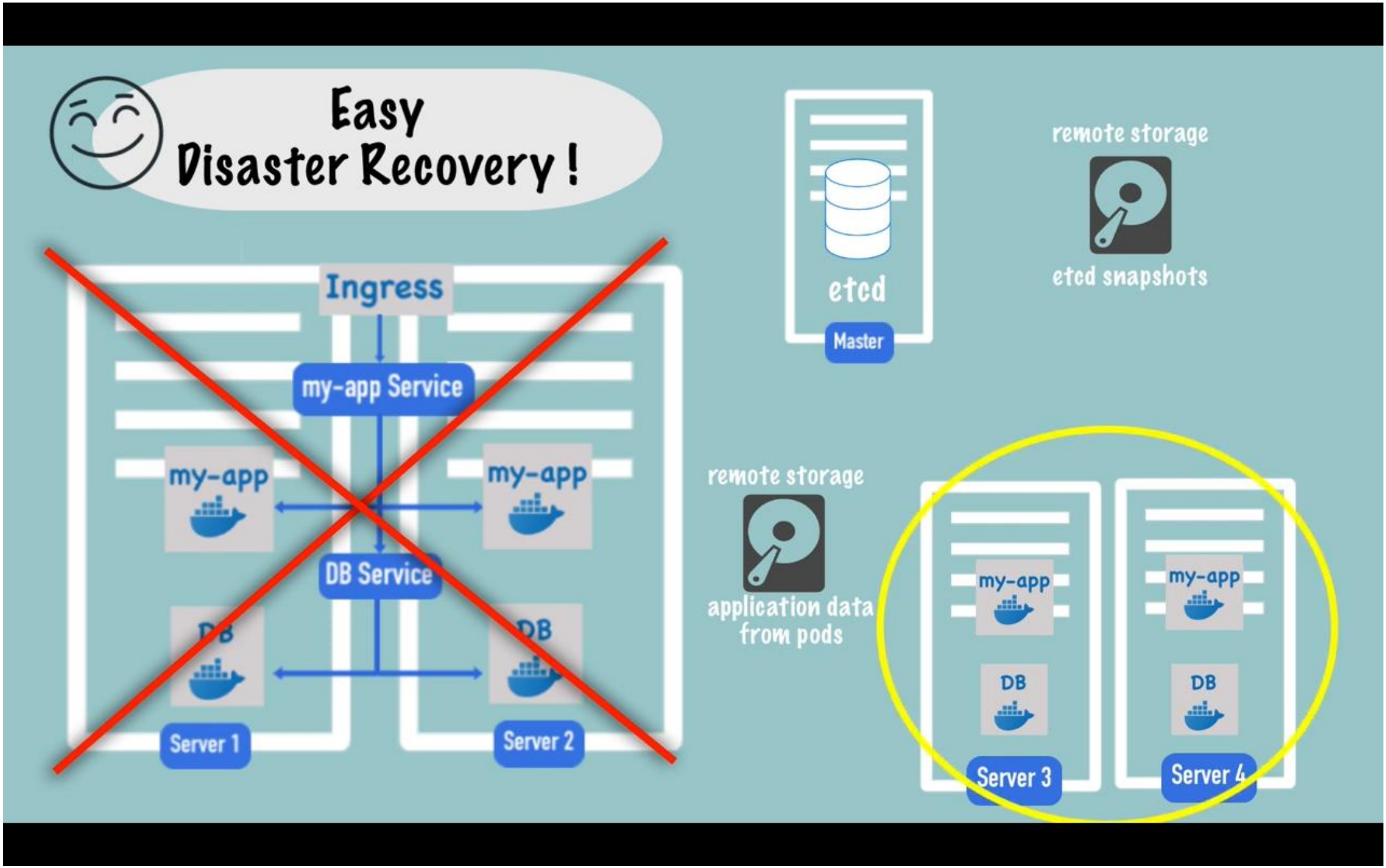

Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
remote storage
application data
from pods


Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
remote storage
application data
from pods


Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
cluster state recovery
remote storage
application data
from pods
Server 3
Server 4


Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
cluster state recovery
remote storage
application data
from pods
my-app
DB
Server 3
my-app
DB
Server 4


Easy
Disaster Recovery !
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
remote storage
application data
from pods
my-app
DB
Server 3
my-app
DB
Server 4


Easy
Disaster Recovery !
Ingress
my-app Service
my-app
my-app
DB Service
DB
DB
Server 1
Server 2
etcd
Master
remote storage
etcd snapshots
remote storage
application data
from pods
my-app
my-app
DB
DB
Server 3
Server 4

Kubernetes vs. AWS

Advantages of Kubernetes

- replication is much easier

- self-healing of K8s

amazon web services

or other platforms

or own setup

Advantages of Kubernetes

- replication is much easier

- self-healing of K8s

amazon web services

or other platforms

or own setup

Advantages of Kubernetes

- replication is much easier

- self-healing of K8s

amazon web services

or other platforms

or own setup

Advantages of Kubernetes

- replication is much easier
- self-healing of K8s
- smart scheduling

amazon web services

or other platforms

or own setup

Advantages of Kubernetes

- replication is much easier
- self-healing of K8s
- smart scheduling

amazon web services

or other platforms

or own setup