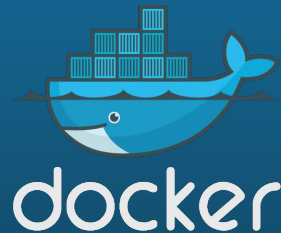


Orchestration in Docker

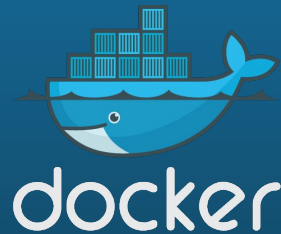
Swarm mode, Docker services and declarative application deployment

Mike Goelzer & Victor Vieux
Docker



Orchestration Overview

Mike Goelzer / mgoelzer@docker.com / gh: mgoelzer



Orchestration in Docker

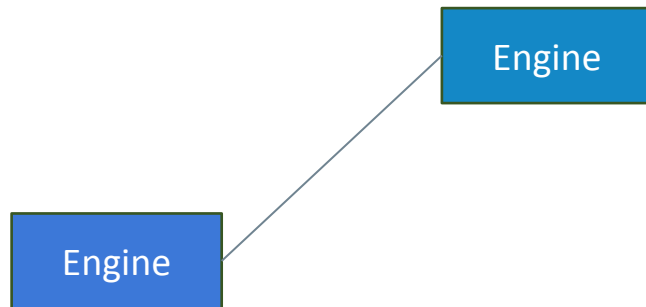
- Orchestration
 - Swarm Mode
 - Docker Services
 - Security
 - Routing mesh
- Container Healthcheck

Swarm Mode

Engine

■ `$ docker swarm init`

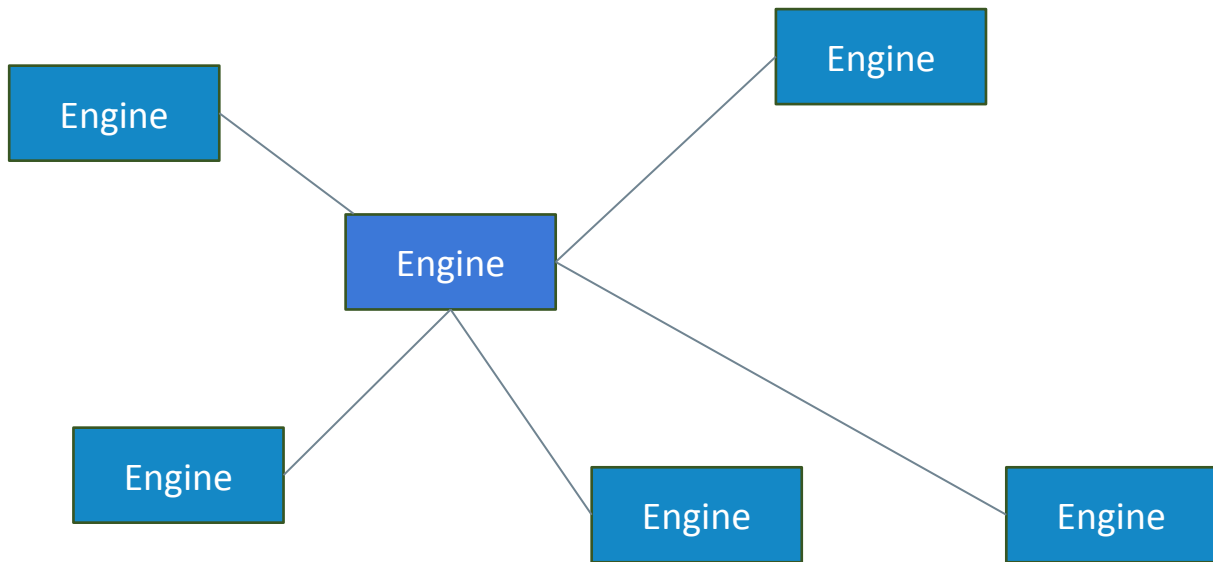
Swarm Mode



■ `$ docker swarm init`

■ `$ docker swarm join <IP of manager>:2377`

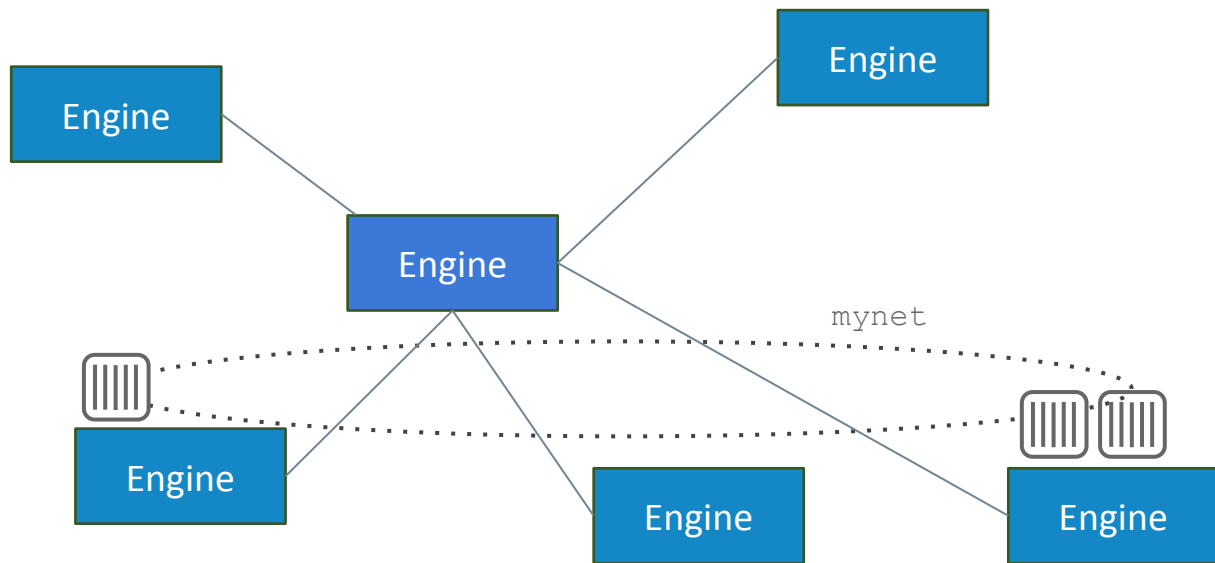
Swarm Mode



■ `$ docker swarm init`

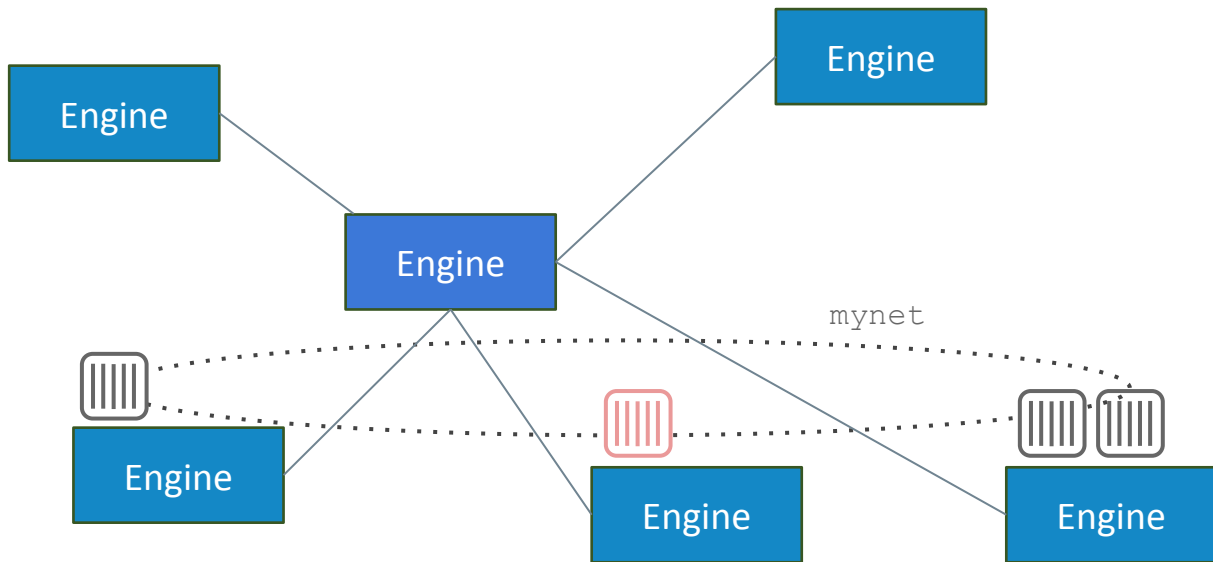
■ `$ docker swarm join <IP of manager>:2377`

Services



```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp frontend_image:latest
```

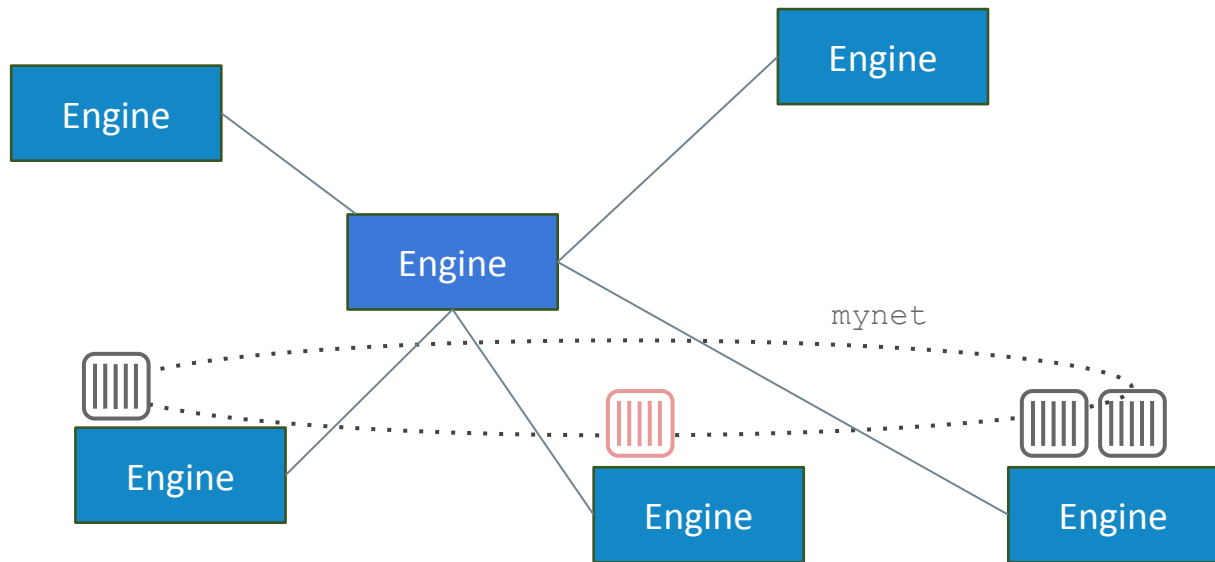
Services



```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp frontend_image:latest
```

```
📦 $ docker service create --name redis --network mynet redis:latest
```

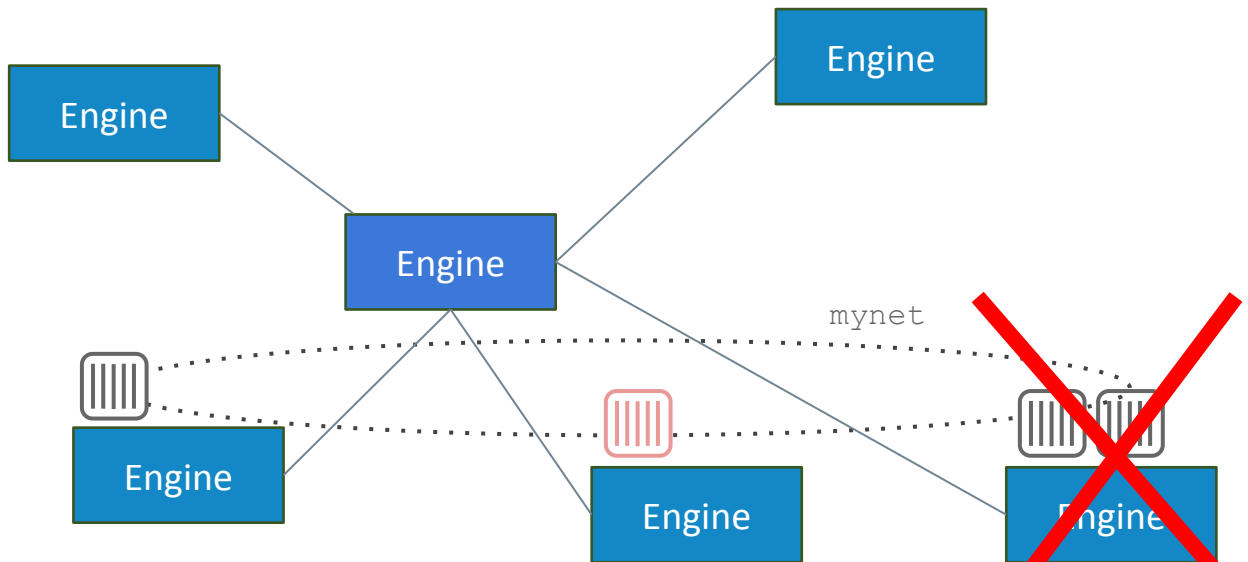

Node Failure



```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp frontend_image:latest
```

```
📦 $ docker service create --name redis --network mynet redis:latest
```

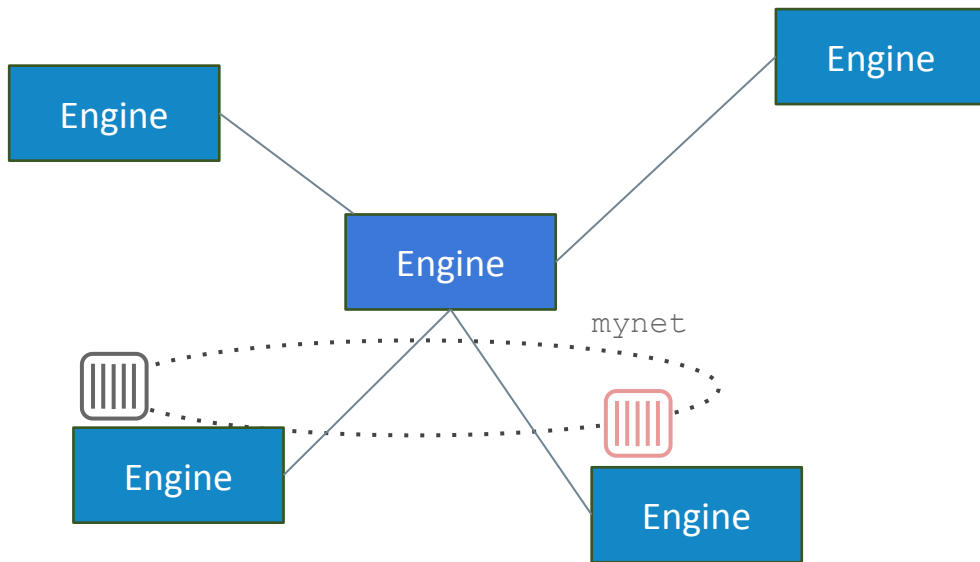
Node Failure



```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp frontend_image:latest
```

```
📦 $ docker service create --name redis --network mynet redis:latest
```

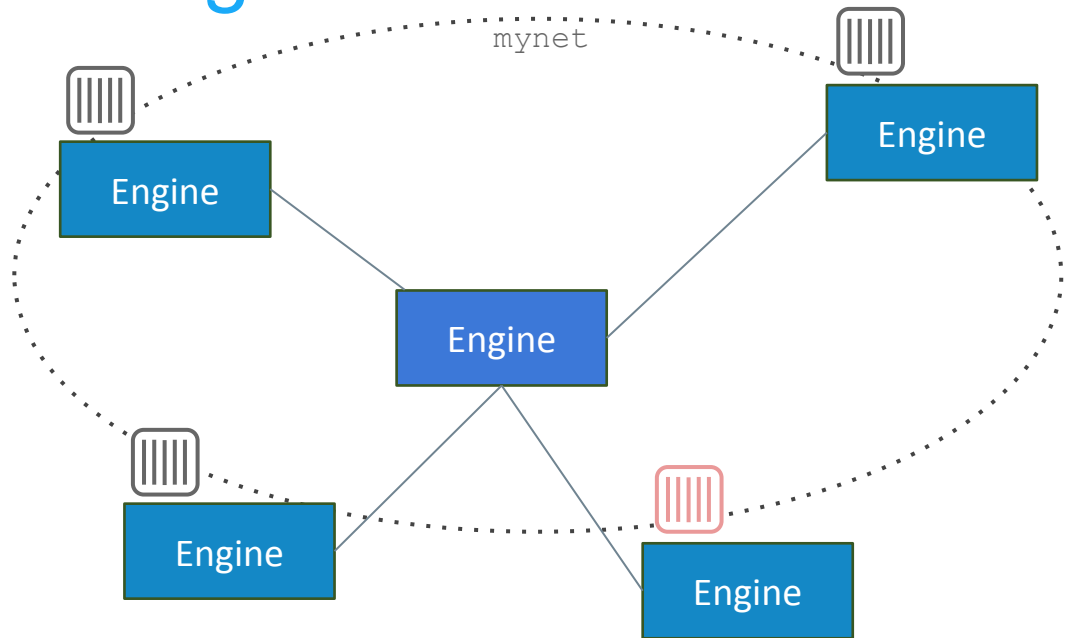
Desired State \neq Actual State



```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp frontend_image:latest
```

```
📦 $ docker service create --name redis --network mynet redis:latest
```

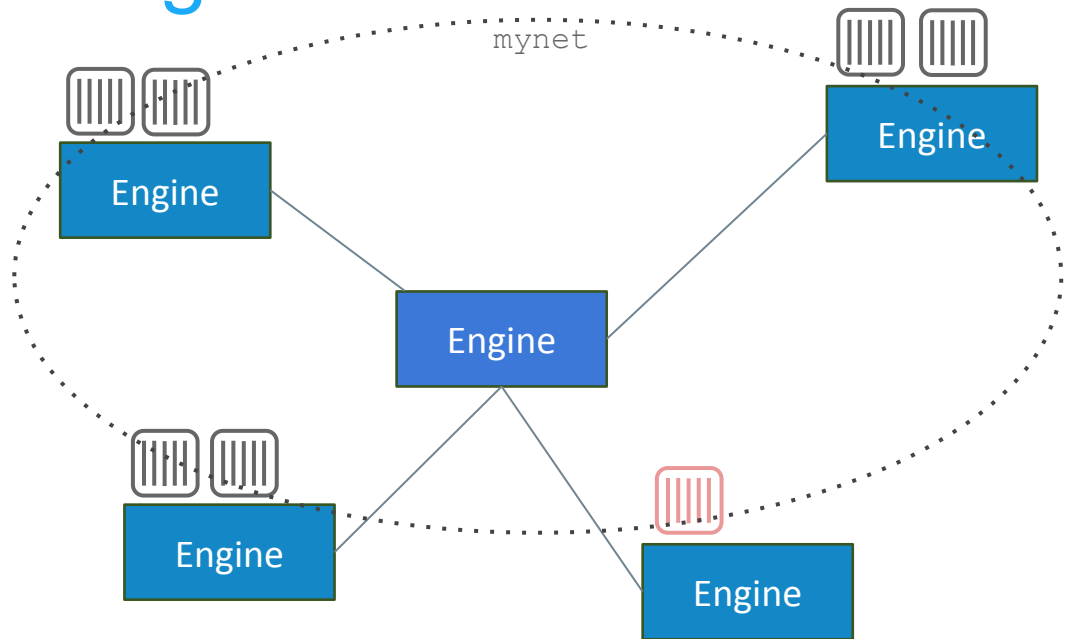
Converge Back to Desired State



```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp frontend_image:latest
```

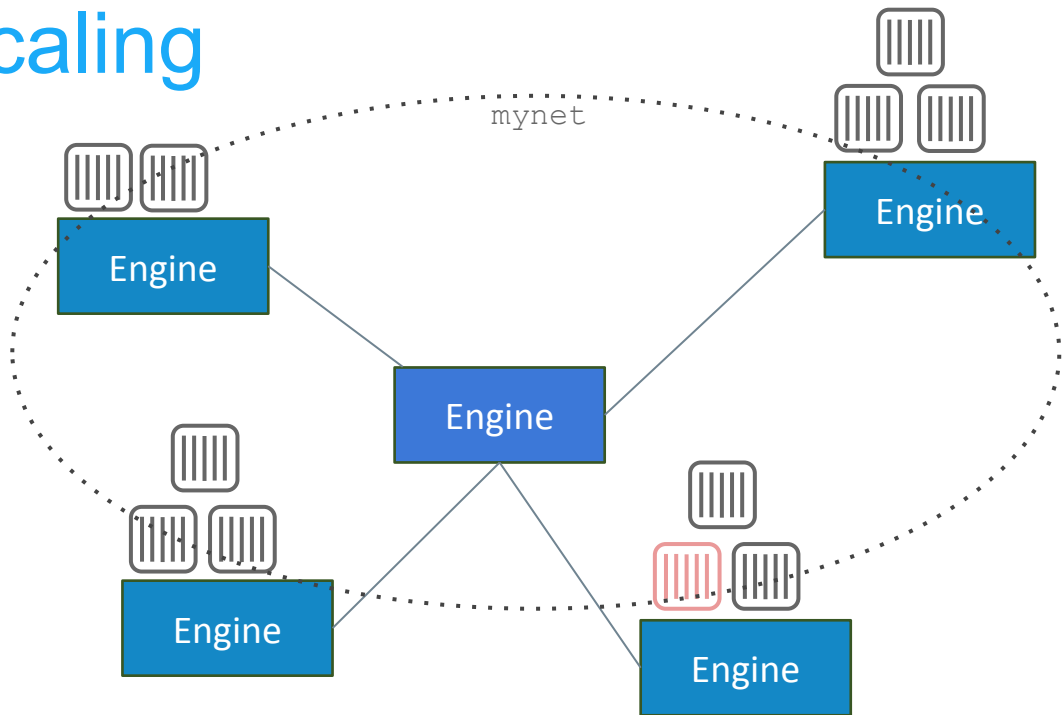
```
📦 $ docker service create --name redis --network mynet redis:latest
```

Scaling



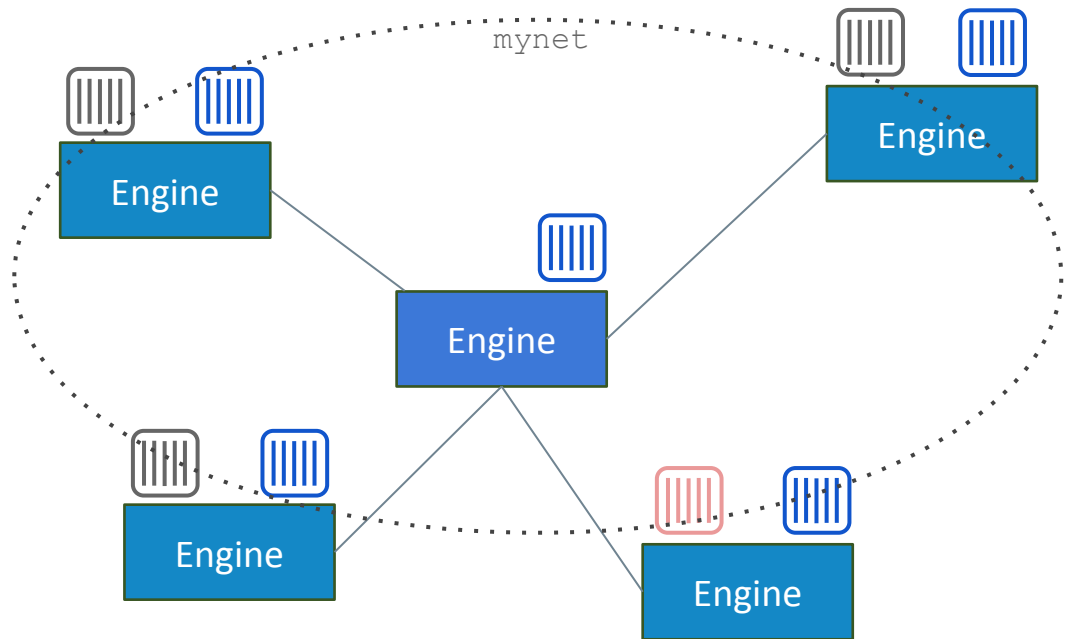
```
$ docker service scale frontend=6
```


Scaling



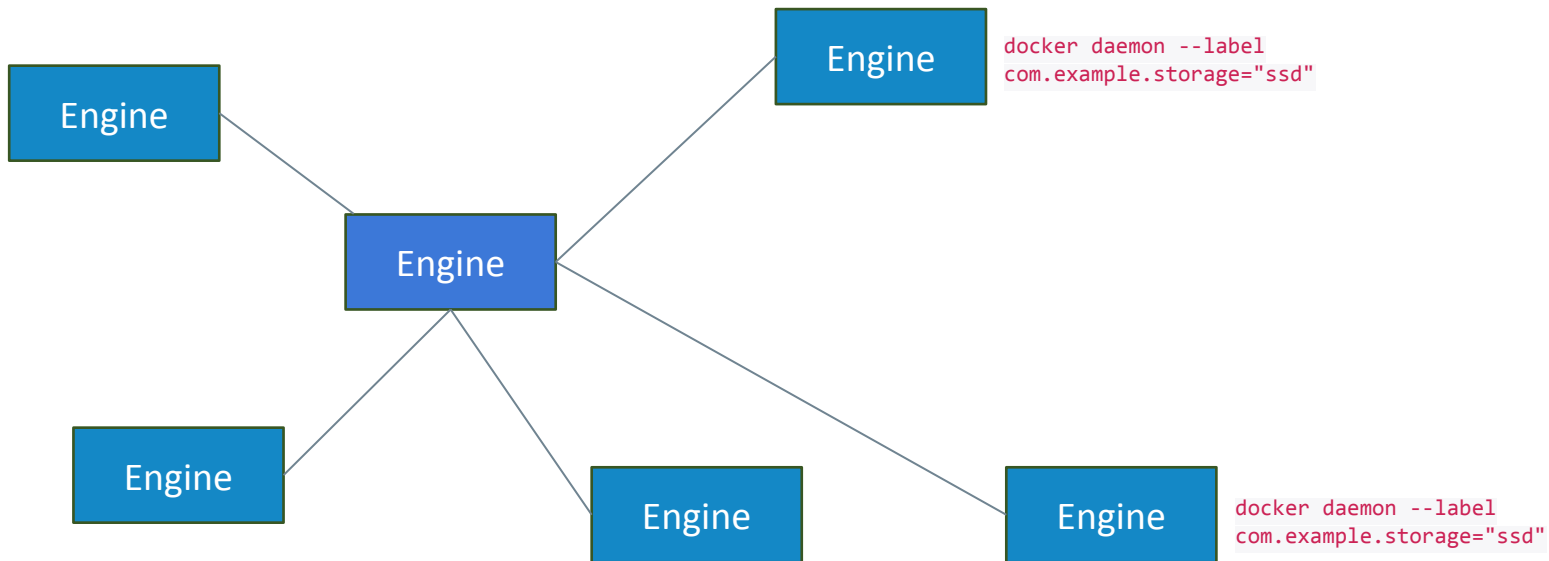
```
$ docker service scale frontend=10
```

Global Services

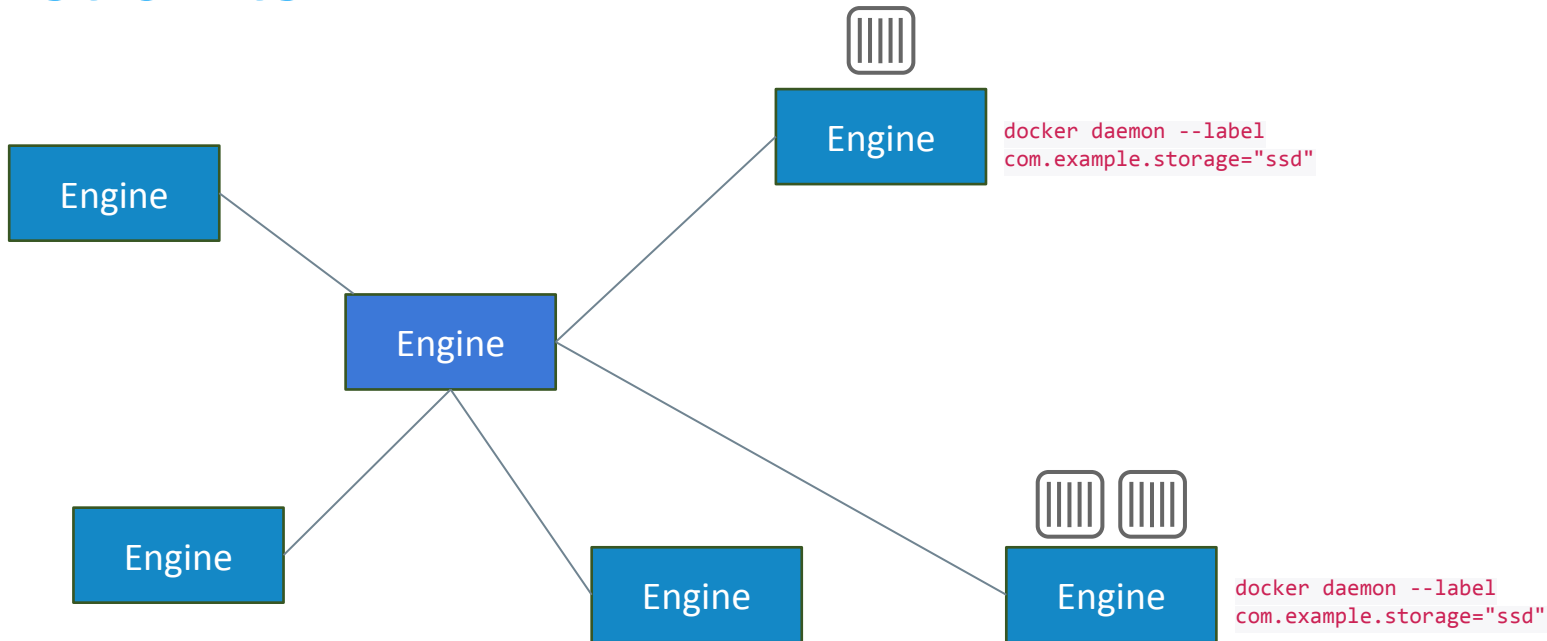


```
 $ docker service create --mode=global --name prometheus  
prom/prometheus
```

Constraints

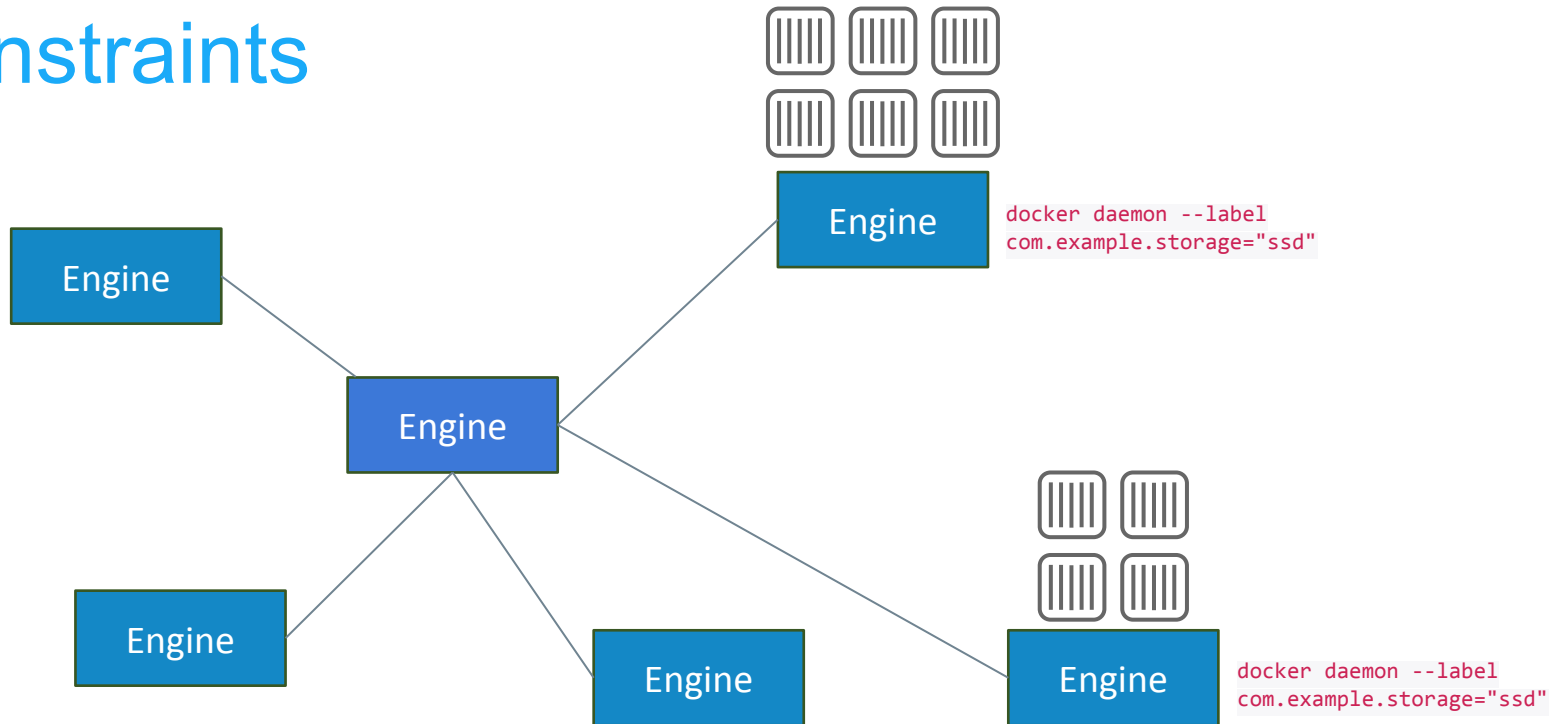


Constraints



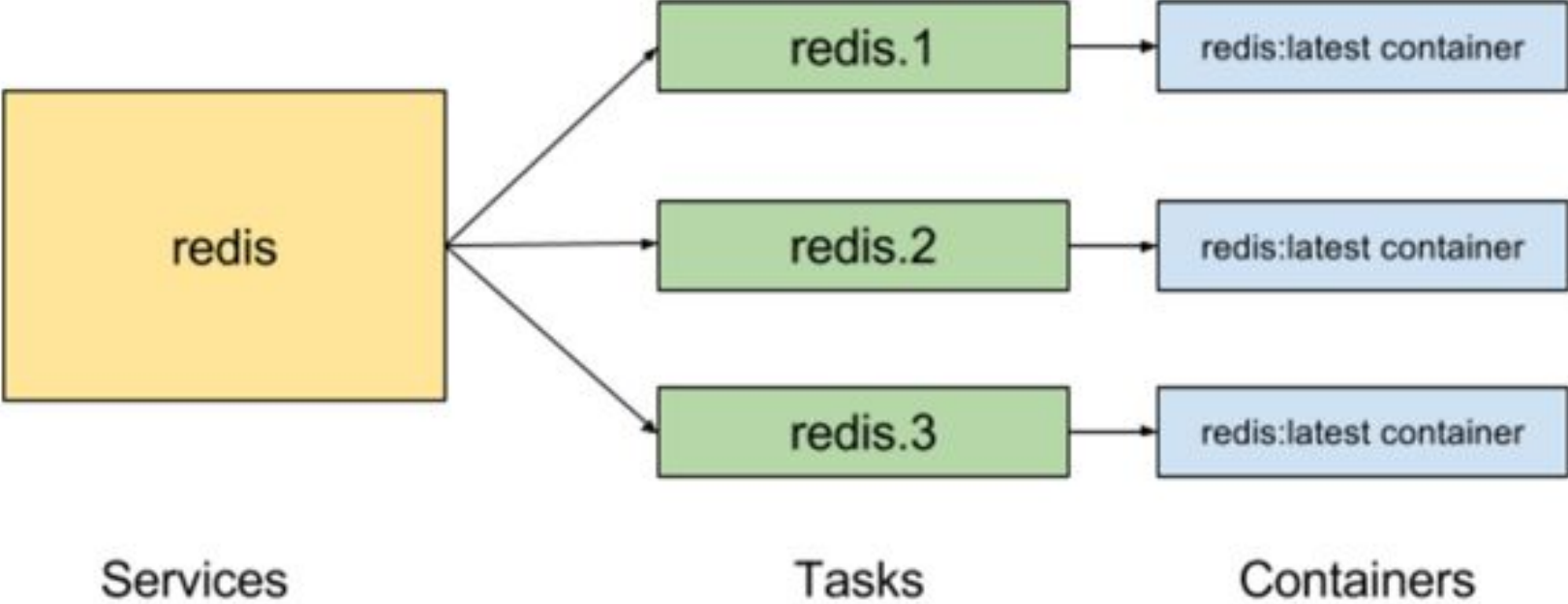
```
🗄️ $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp --constraint  
engine.labels.com.example.storage==ssd frontend_image:latest
```

Constraints

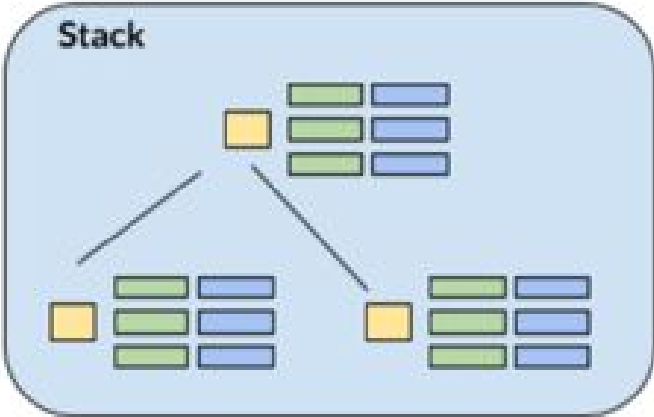
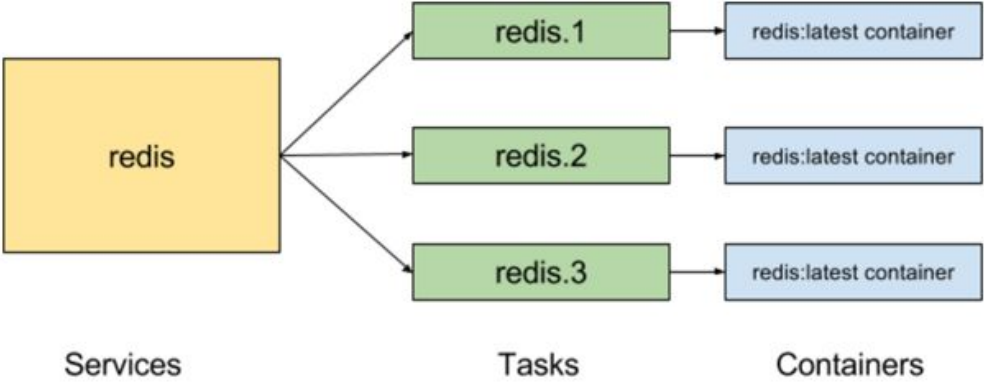


```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
--publish 80:80/tcp --constraint  
engine.labels.com.example.storage==ssd frontend_image:latest  
$ docker service scale frontend=10
```

Services



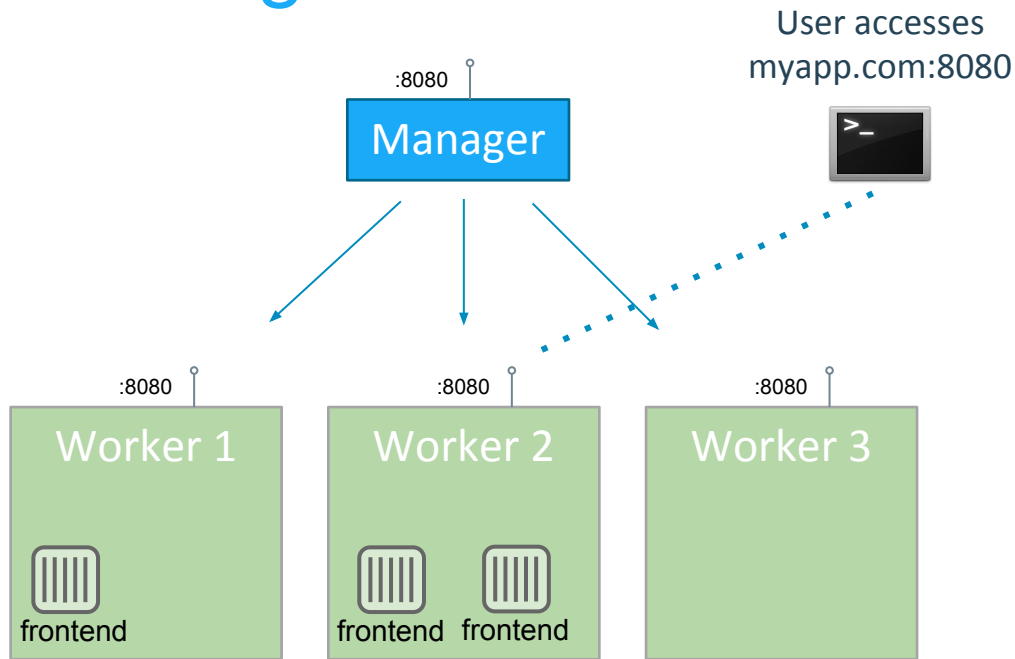
Services are grouped into stacks



Swarm mode orchestration is optional

- You don't have to use it
- 1.12 is fully backwards compatible
- Will not break existing deployments and scripts

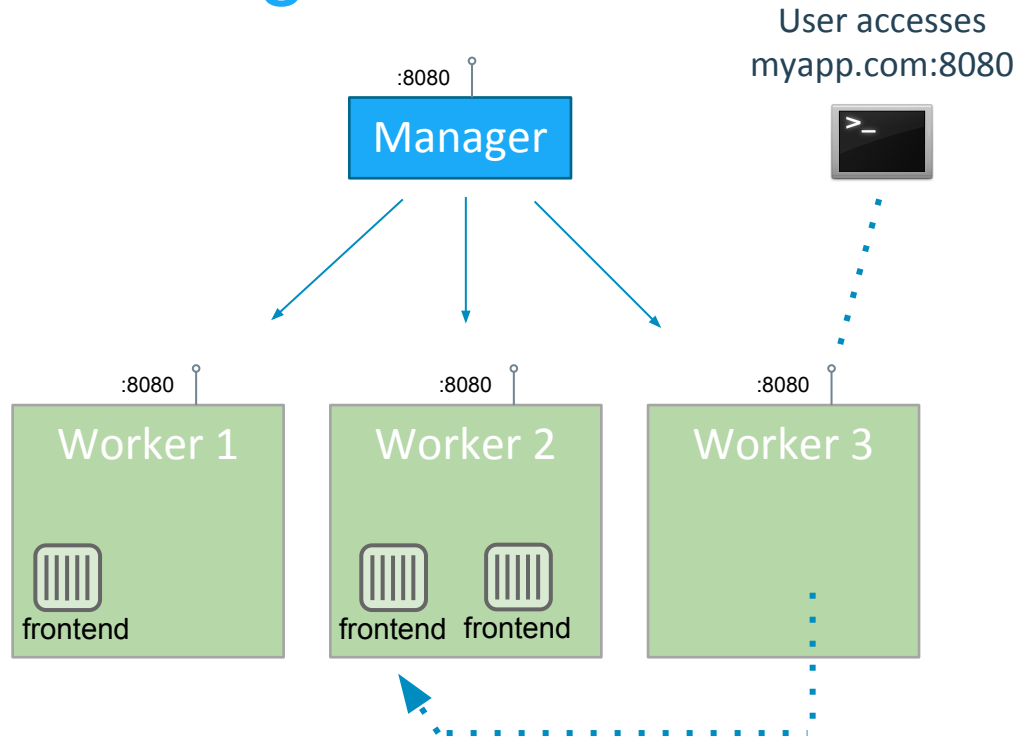
Routing Mesh



- Operator reserves a swarm-wide ingress port (8080) for `myapp`
- Every node listens on 8080
- Container-aware routing mesh can transparently reroute traffic from Worker3 to a node that is running container
- Built in load balancing into the Engine
- DNS-based service discovery

```
 $ docker service create --replicas 3 --name frontend --network mynet  
-p 8080:80 frontend_image:latest
```

Routing Mesh: Published Ports



- Operator reserves a swarm-wide ingress port (8080) for `myapp`
- Every node listens on 8080
- Container-aware routing mesh can transparently reroute traffic from Worker3 to a node that is running container
- Built in load balancing into the Engine
- DNS-based service discovery

```
📦 $ docker service create --replicas 3 --name frontend --network mynet  
-p 8080:80 frontend_image:latest
```

Security out of the box

Cryptographic Node Identity

- Workload segregation (think PCI)

There is no “insecure mode”:

- TLS mutual auth
- TLS encryption
- Certificate rotation

Container Health Check in Dockerfile

```
HEALTHCHECK --interval=5m --timeout=3s
  --retries 3
  CMD curl -f http://localhost/ || exit 1
```

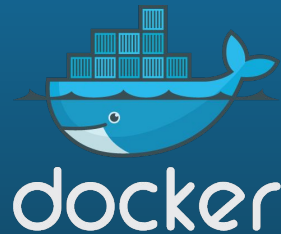
Checks every 5 minutes that web server can return index page within 3 seconds.

Three consecutive failures puts container in an unhealthy state.

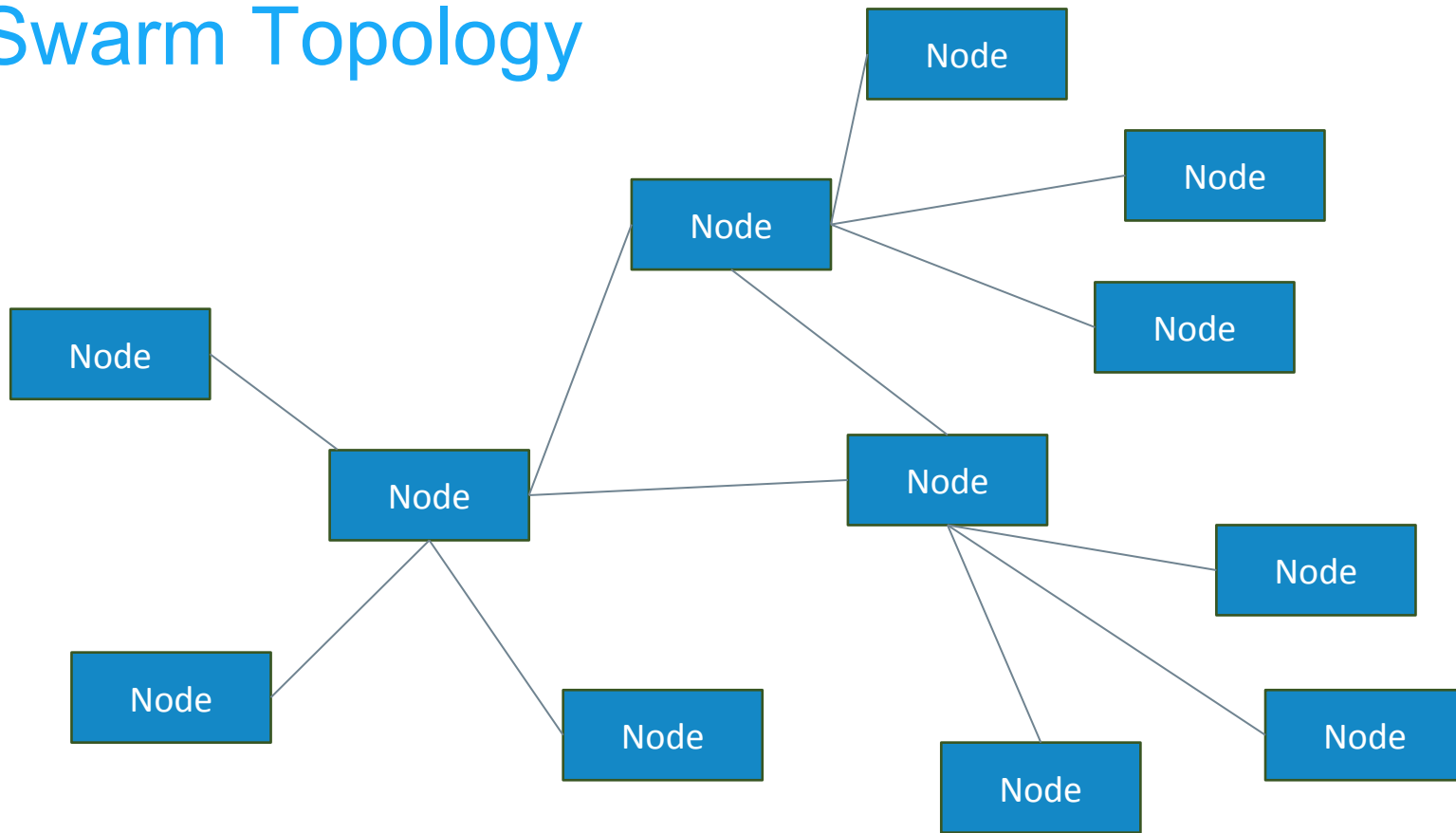
Orchestration Deep Dive

+ demo at the end

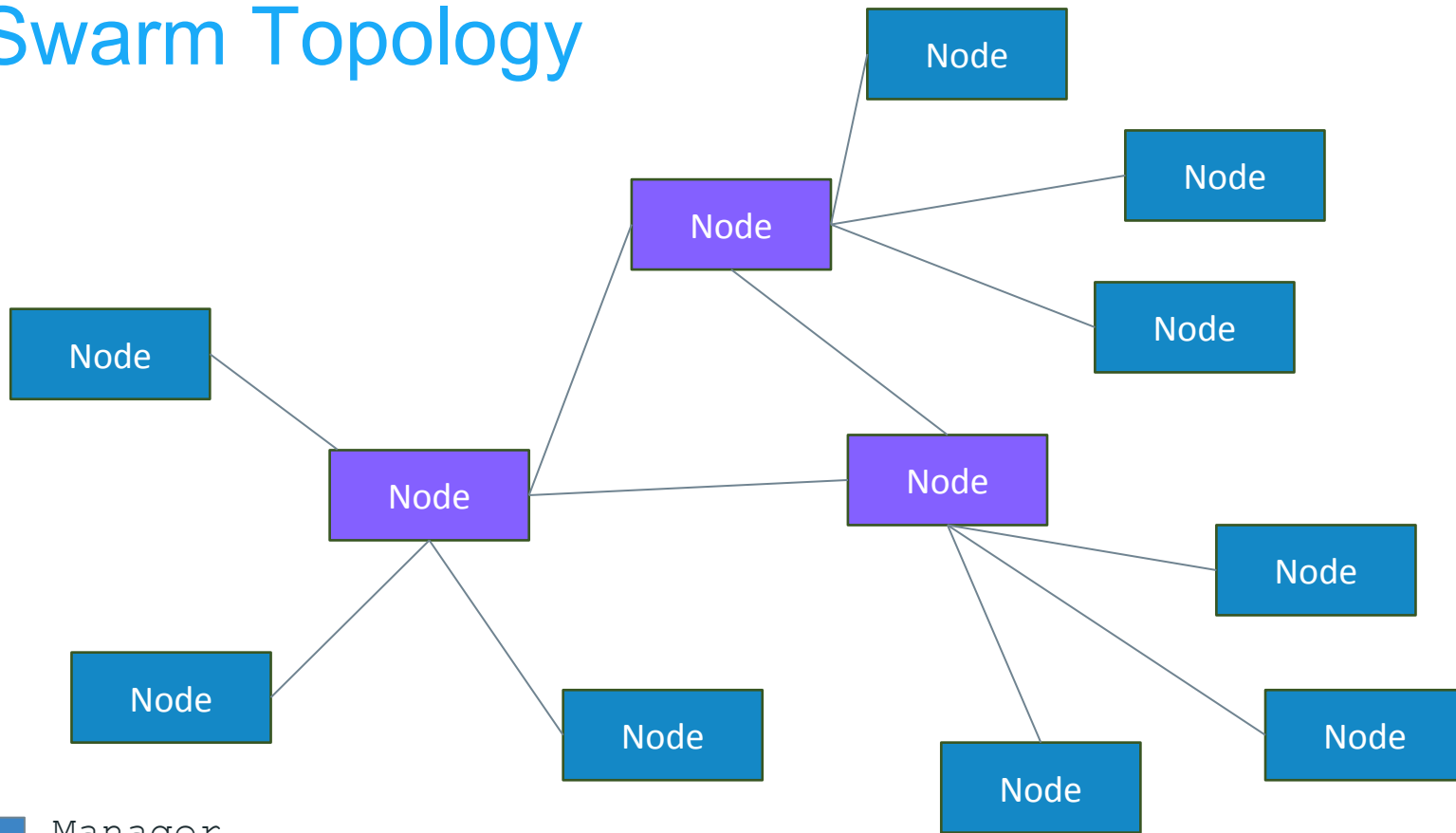
Victor Vieux / vieux@docker.com / gh: vieux



Swarm Topology



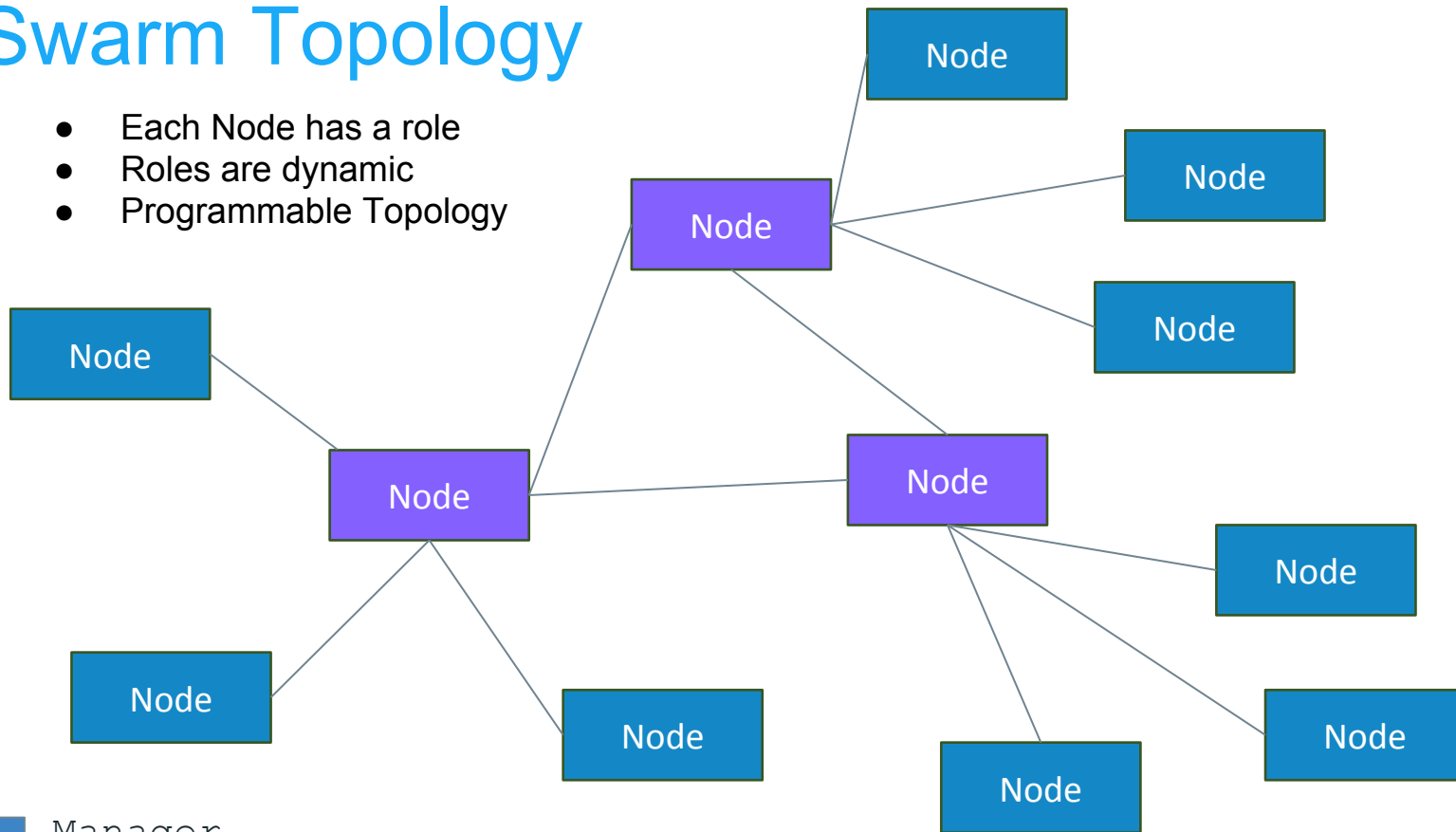
Swarm Topology



- Manager
- Worker

Swarm Topology

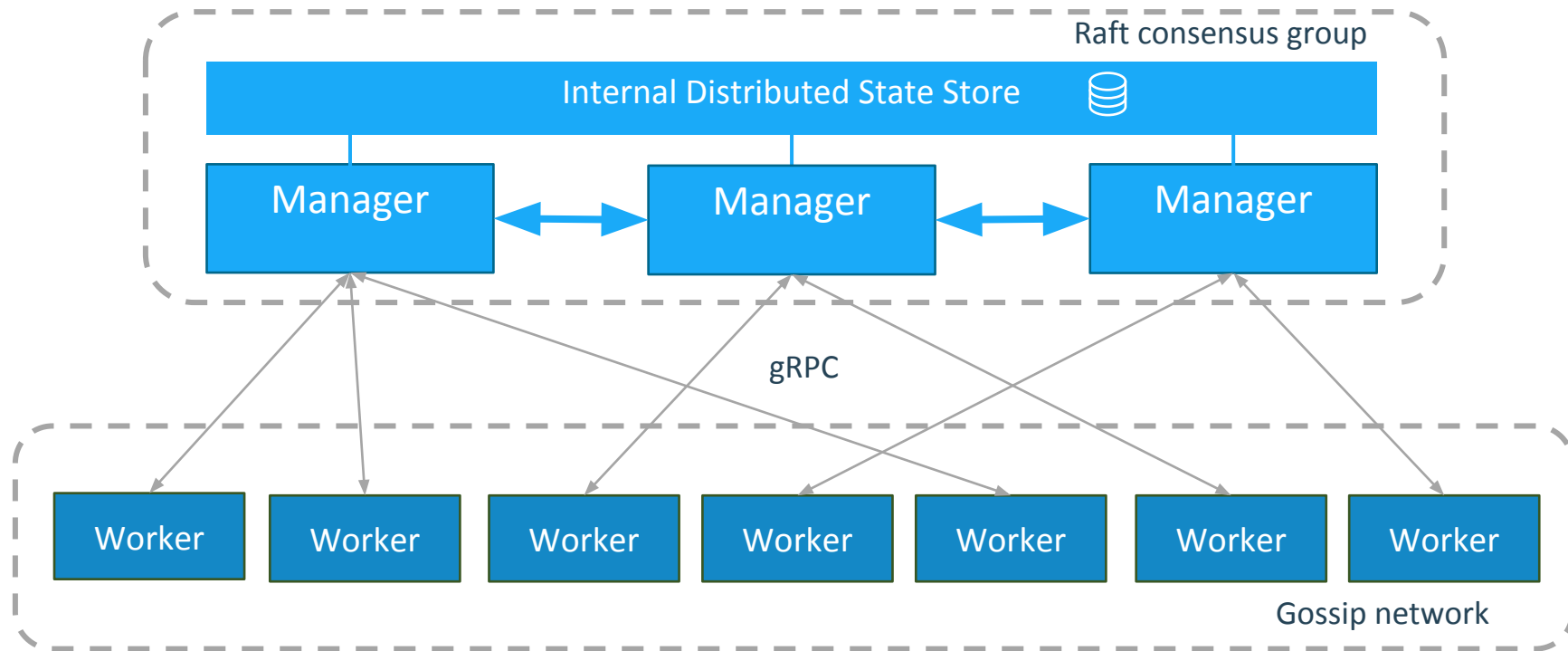
- Each Node has a role
- Roles are dynamic
- Programmable Topology



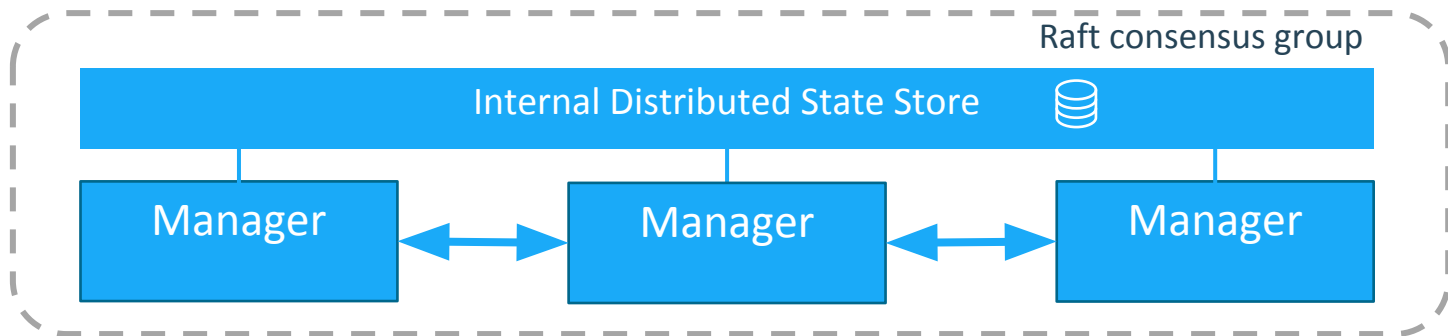
■ Manager

■ Worker

Docker swarm mode communication internals

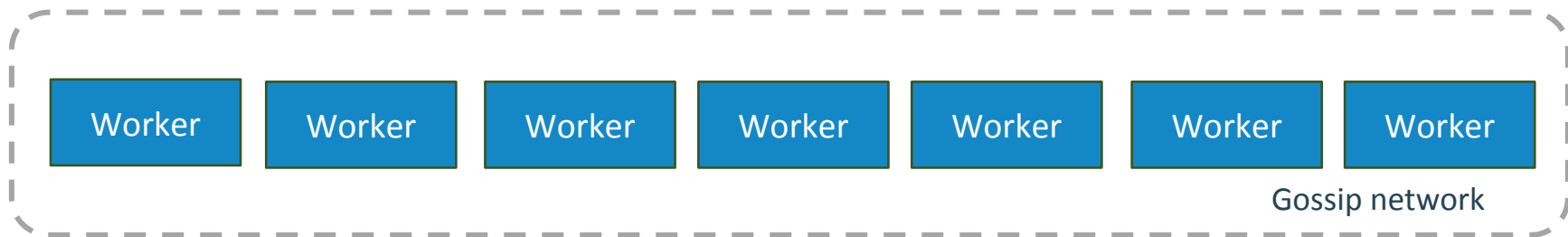


Quorum Layer



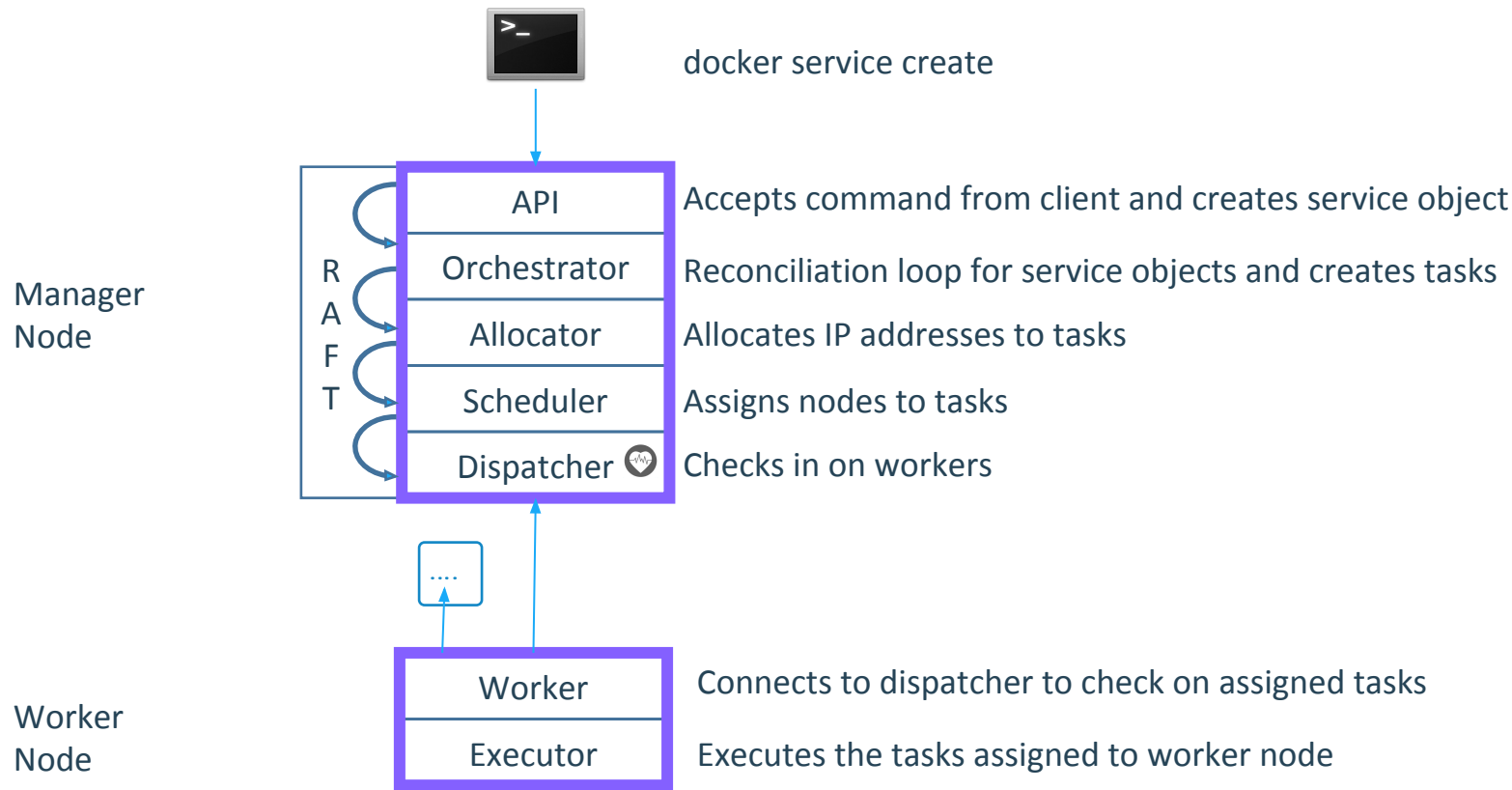
- Strongly consistent: Holds desired state
- Simple to operate
- Blazing fast (in-memory reads, domain specific indexing, ...)
- Secure

Worker-to-Worker Gossip

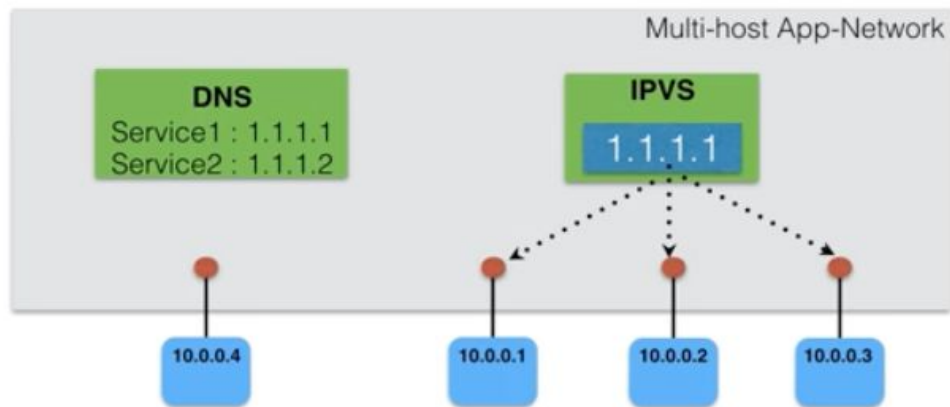


- Eventually consistent: Routing mesh, load balancing rules, ...
- High volume, p2p network between workers
- Secure: Symmetric encryption with key rotation in Raft

Node Breakdown

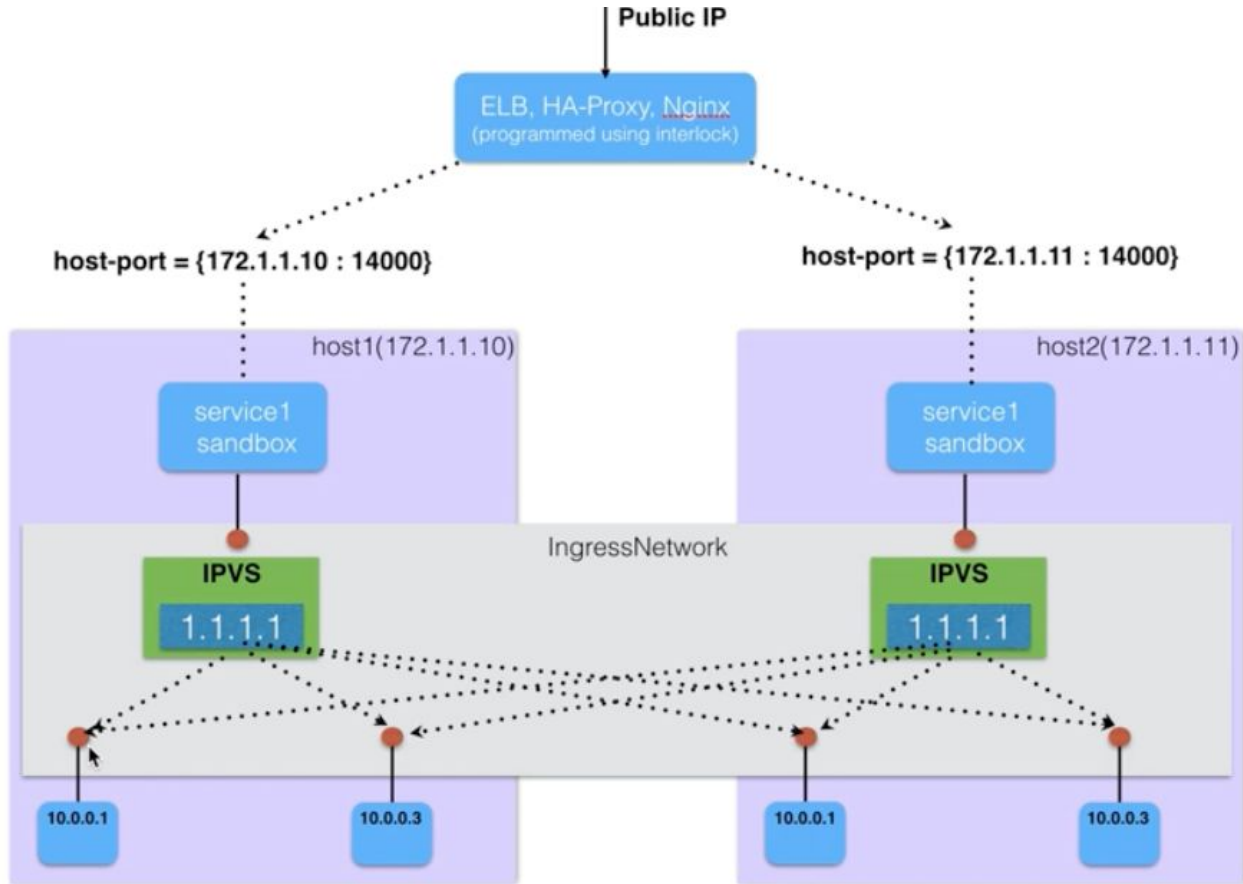


Internal Load Balancer

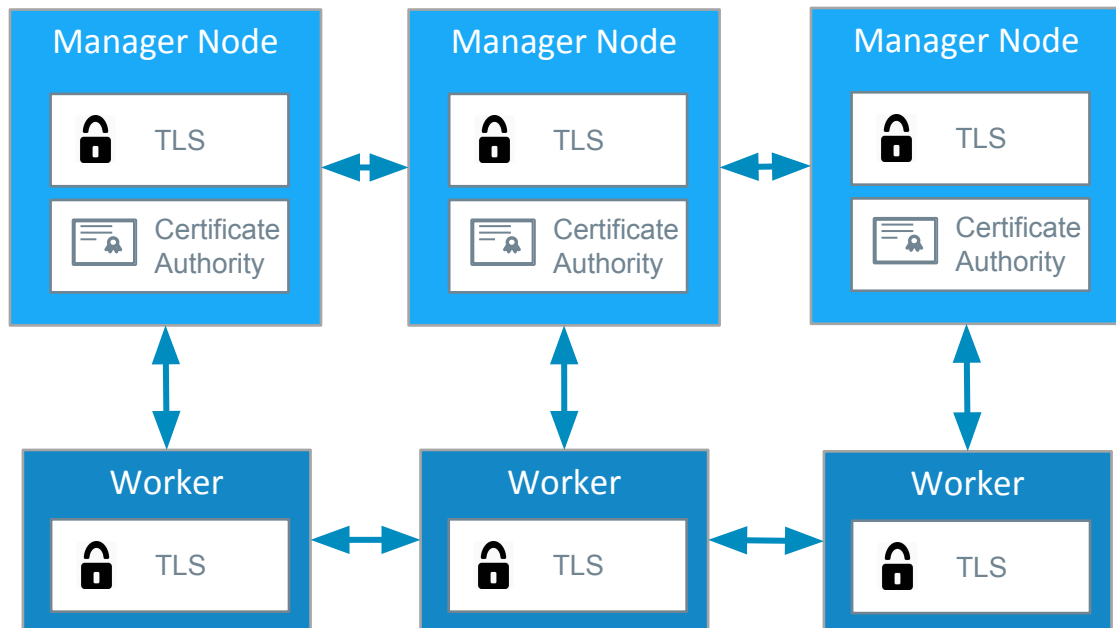


- **Load-balancer is designed as an integral part of CNM**
 - Works on top of CNM constructs (network, endpoint, sandbox, SD)
 - Every Service gets a Virtual-IP
 - Built-in SD resolves Service-Name -> VIP
 - Service VIP -> Container IP load balancing achieved using IPVS

Ingress Load Balancer



Secure by default with end-to-end encryption



- Cryptographic node identity
- Automatic encryption and mutual auth (TLS)
- Automatic cert rotation
- External CA integration

DEMO

Questions?

Victor Vieux
vieux@docker.com / @vieux

Mike Goelzer
mgoelzer@docker.com / @mgoelzer





docker

Victor Vieux

vieux@docker.com / @vieux

Mike Goelzer

mgoelzer@docker.com / @mgoelzer