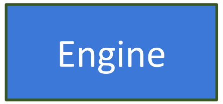
What’s New in Docker 1.12

(Spoiler alert: a lot!)

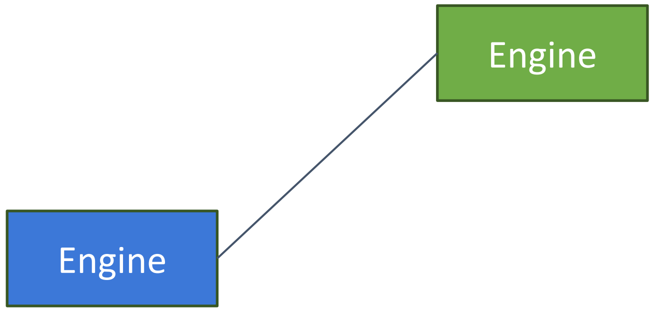
Mike Goelzer

Swarm Mode



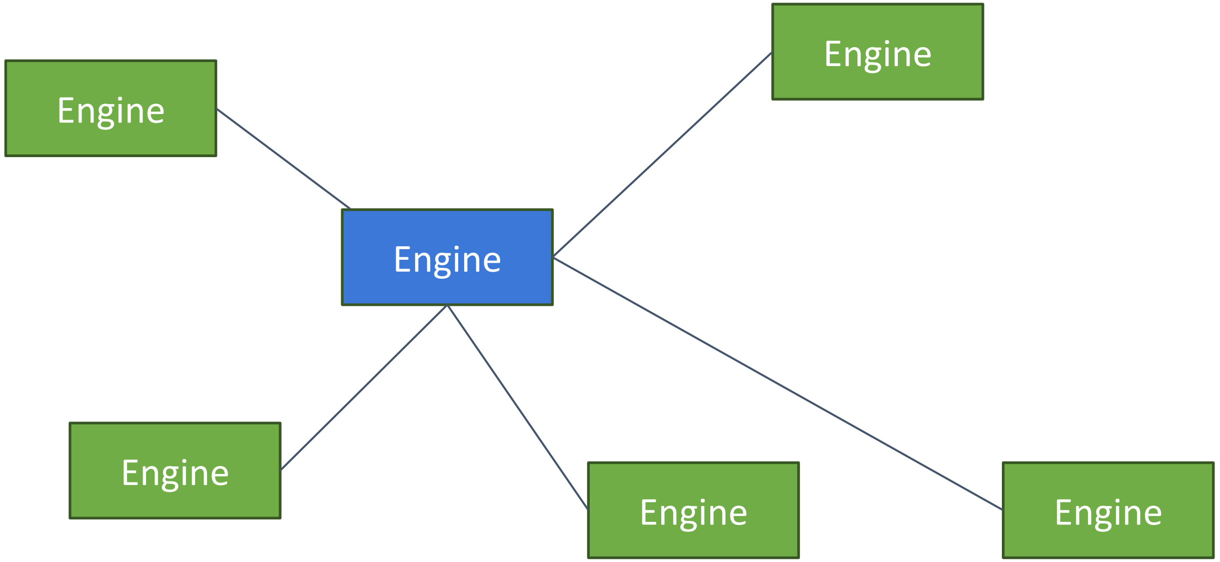
🟦
```
$ docker swarm init
```

Swarm Mode



🟦
```
$ docker swarm init
```
🟩
```
$ docker swarm join <IP of manager>:2377
```

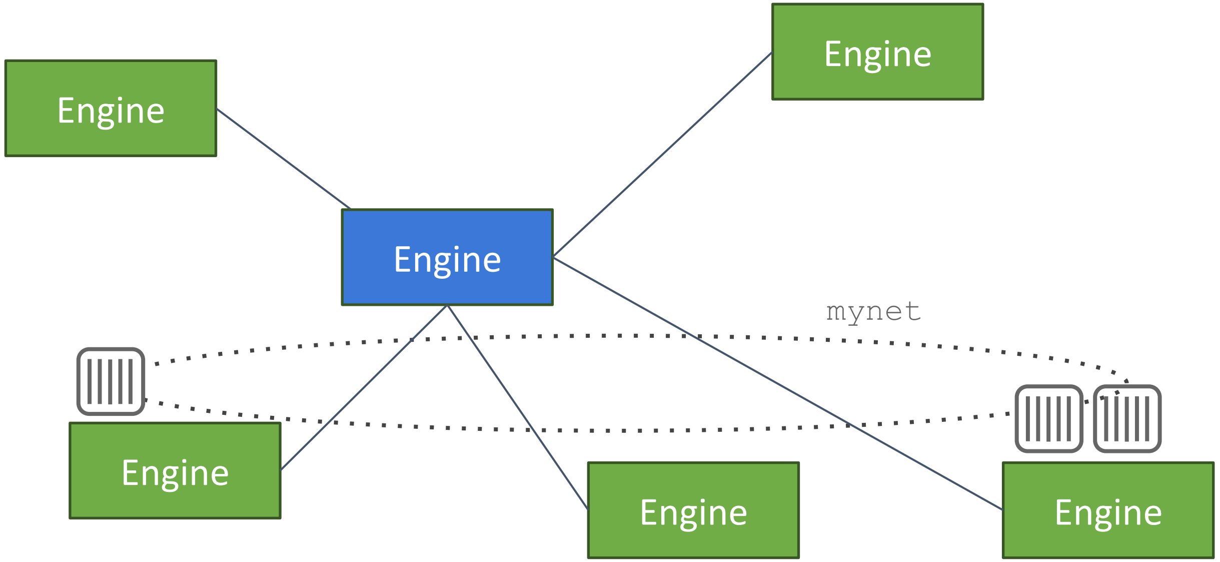
Swarm Mode



🟦
```
$ docker swarm init
```

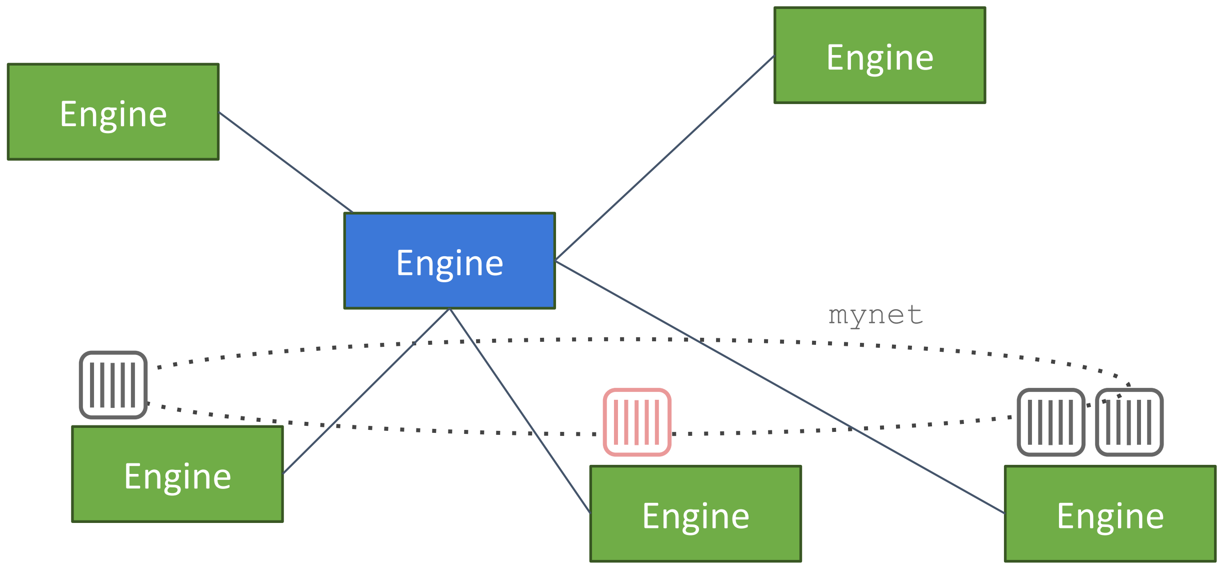
🟩
```
$ docker swarm join <IP of manager>:2377
```

Services



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

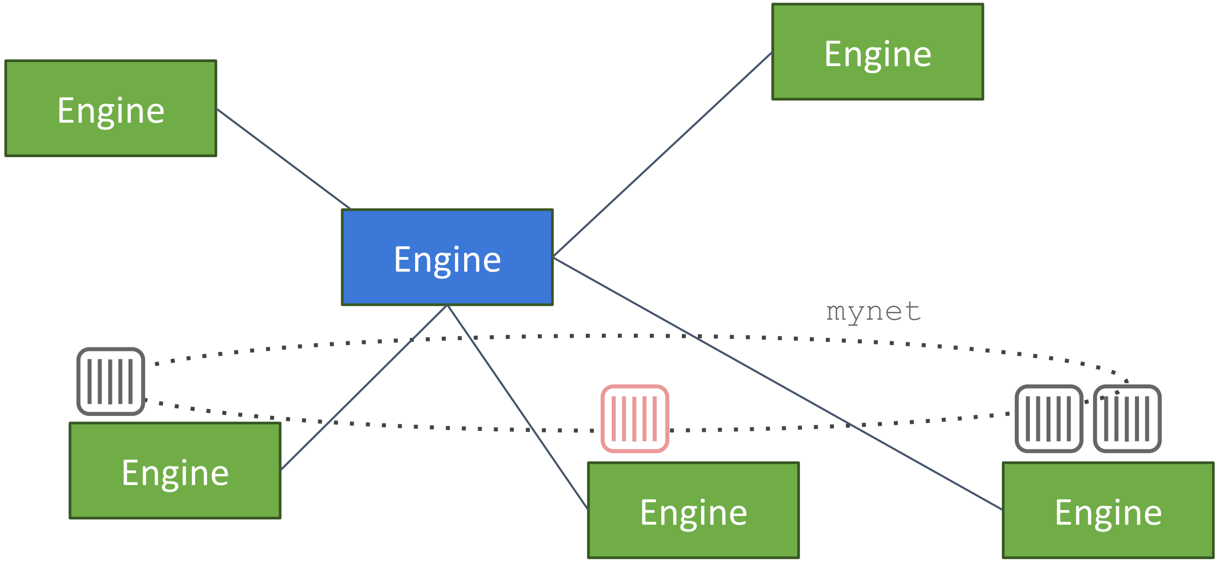
Services



```
$ 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
```

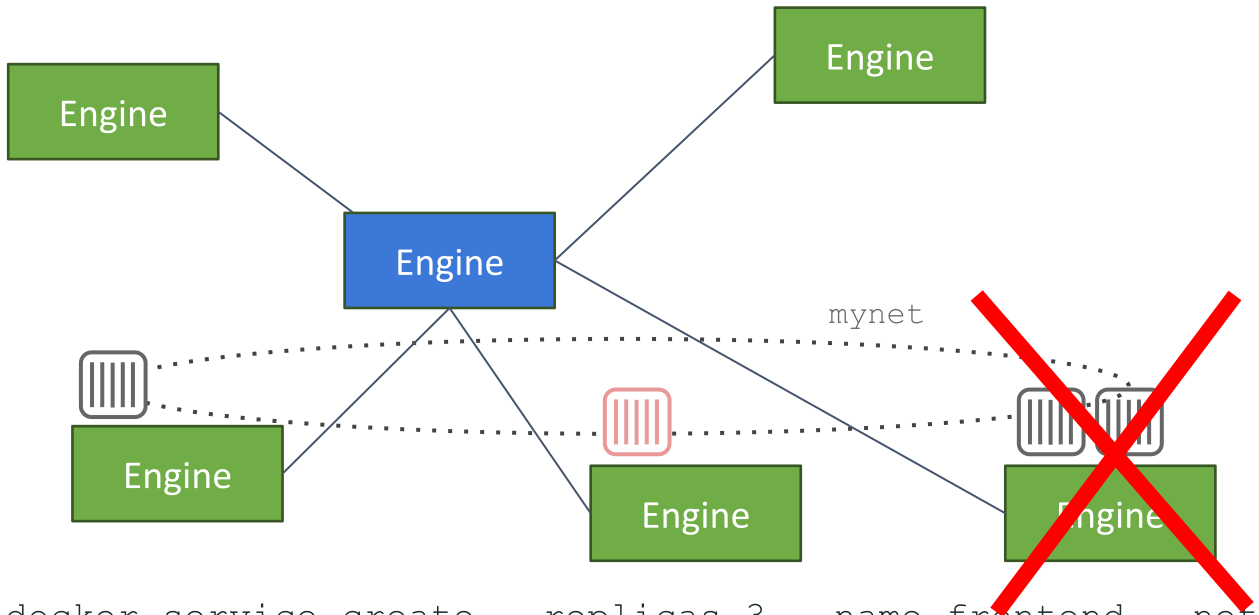
Node Failure



```
$ 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
```

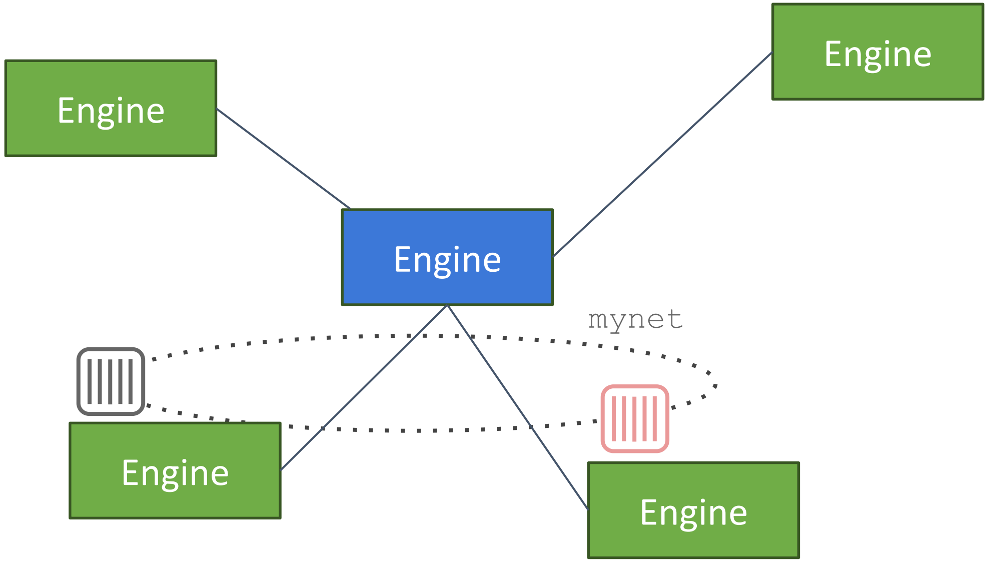
Node Failure



```
$ 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
```

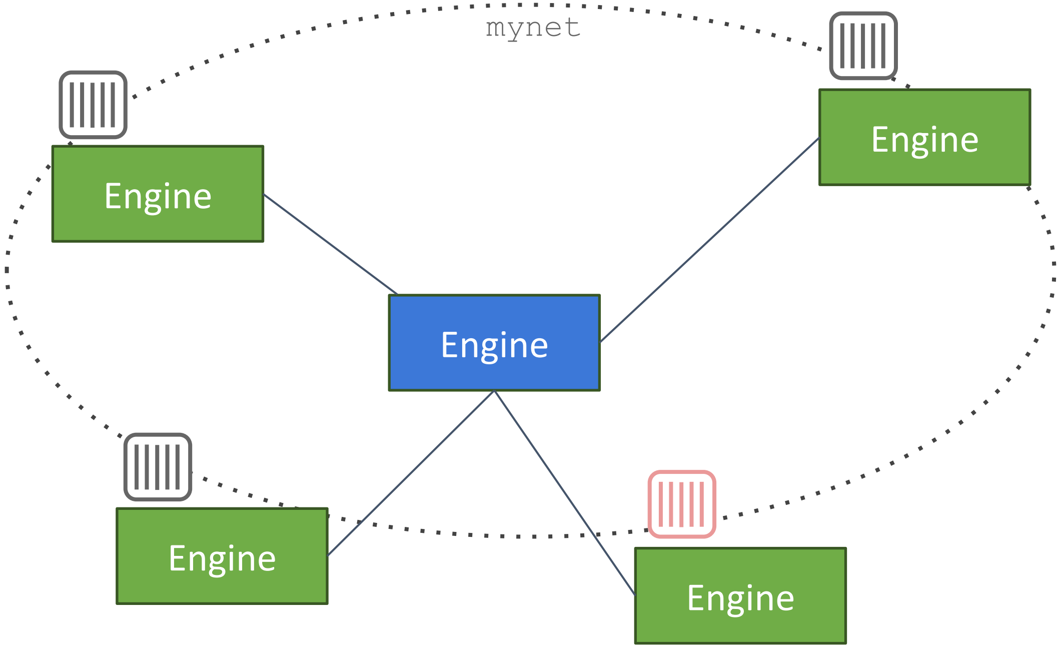
Desired State ≠ Actual State



```
$ 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
```

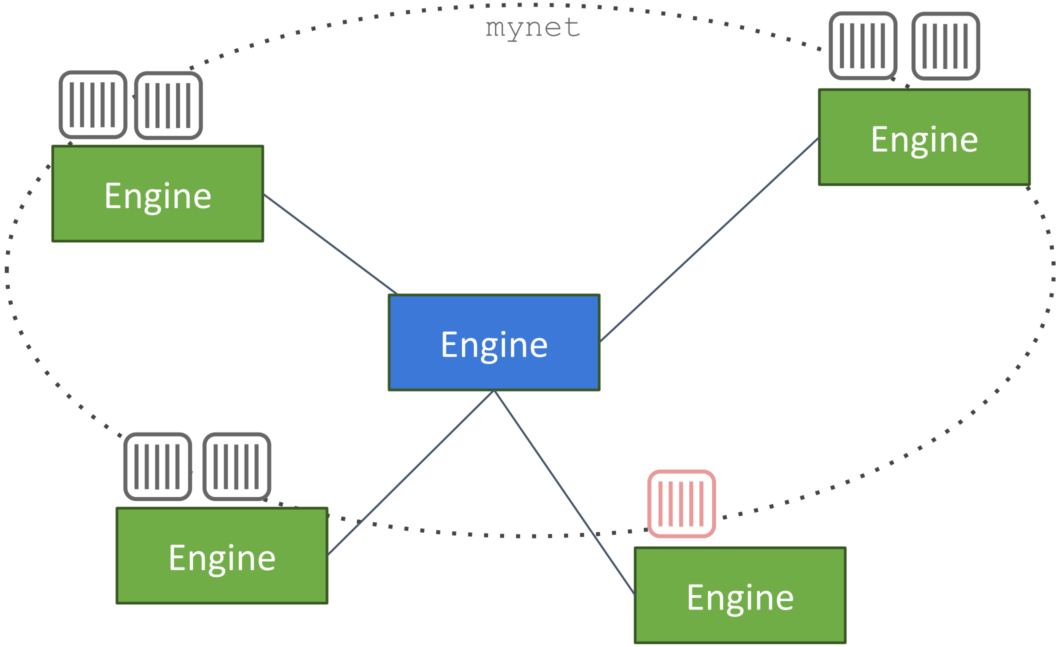
Converge Back to Desired State



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

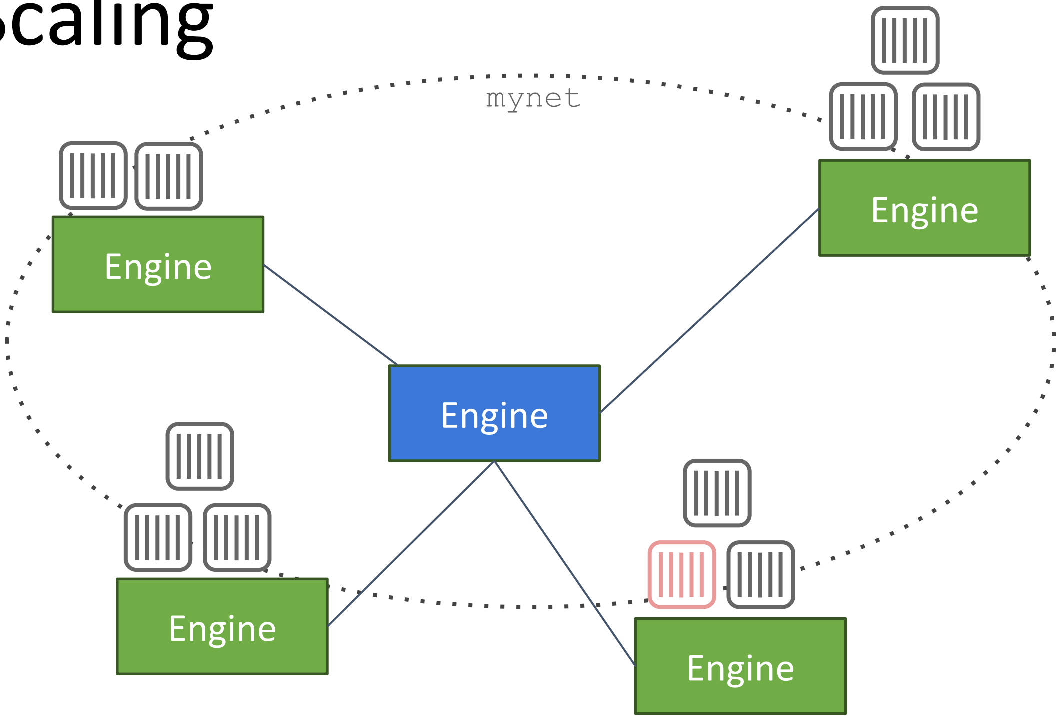
```
$ docker service create --name redis --network mynet redis:latest
```

Scaling



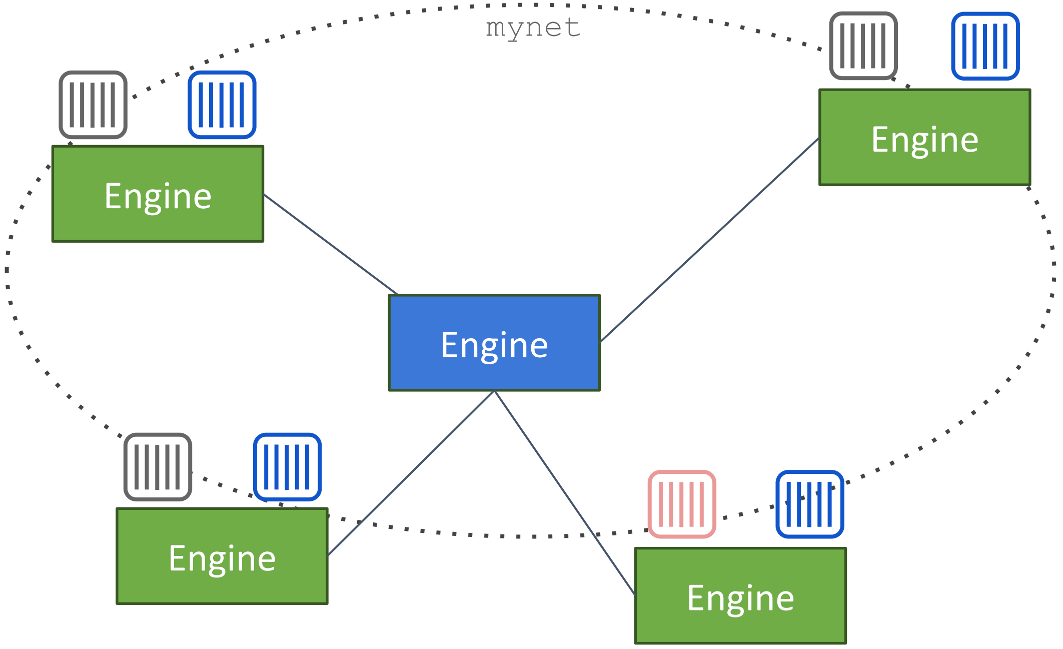
```
$ docker service scale frontend=6
```

Scaling



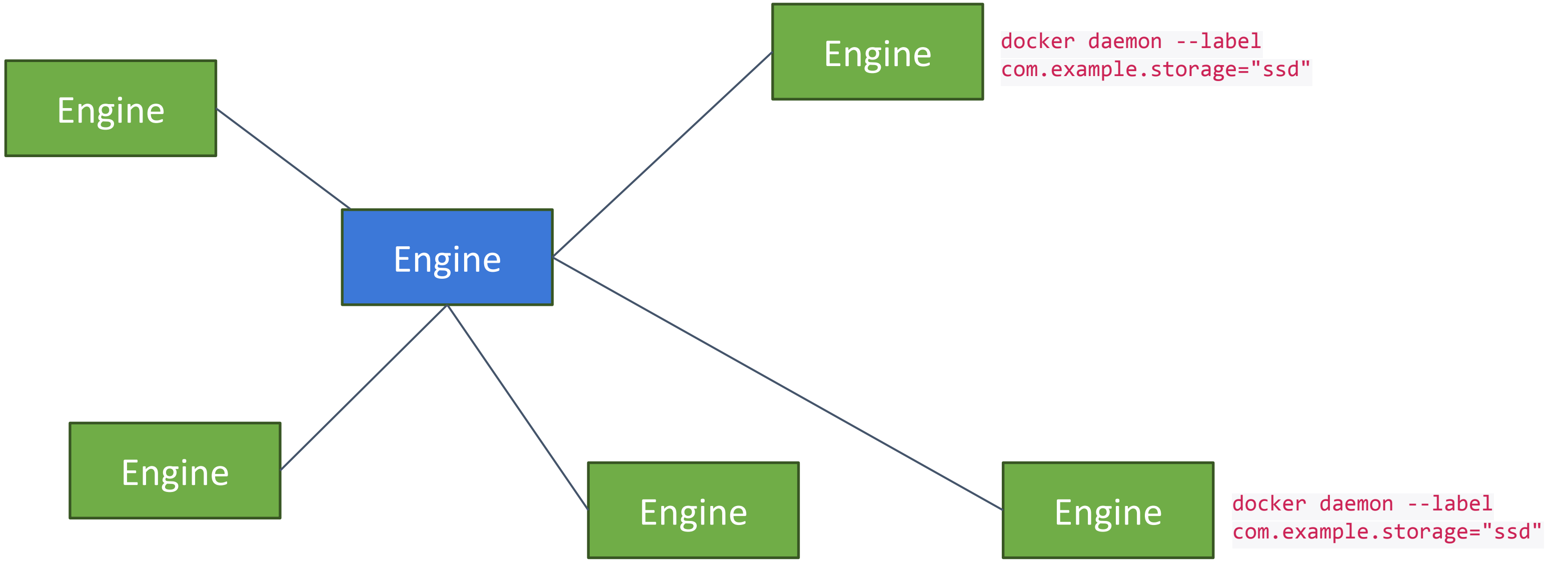
```
$ docker service scale frontend=10
```

Global Services



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

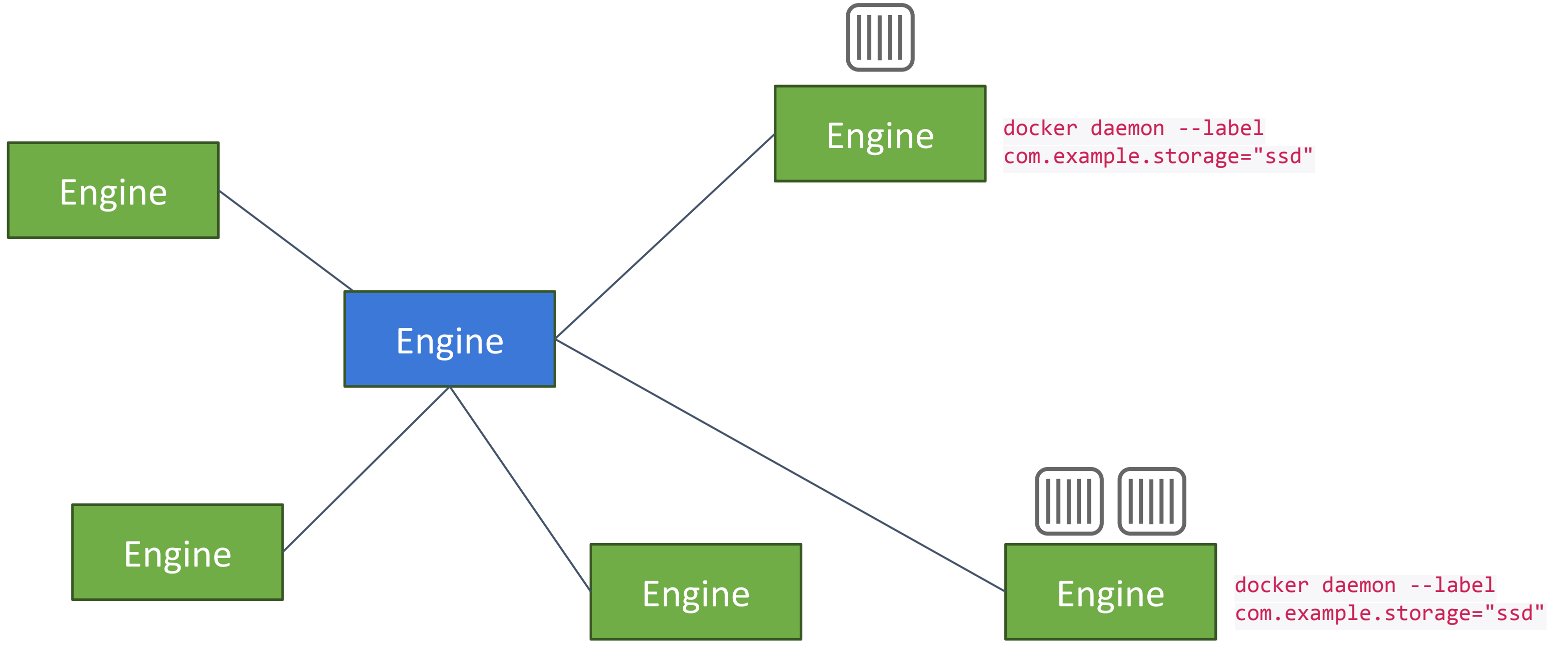
Constraints



```
docker daemon --label
com.example.storage="ssd"
```

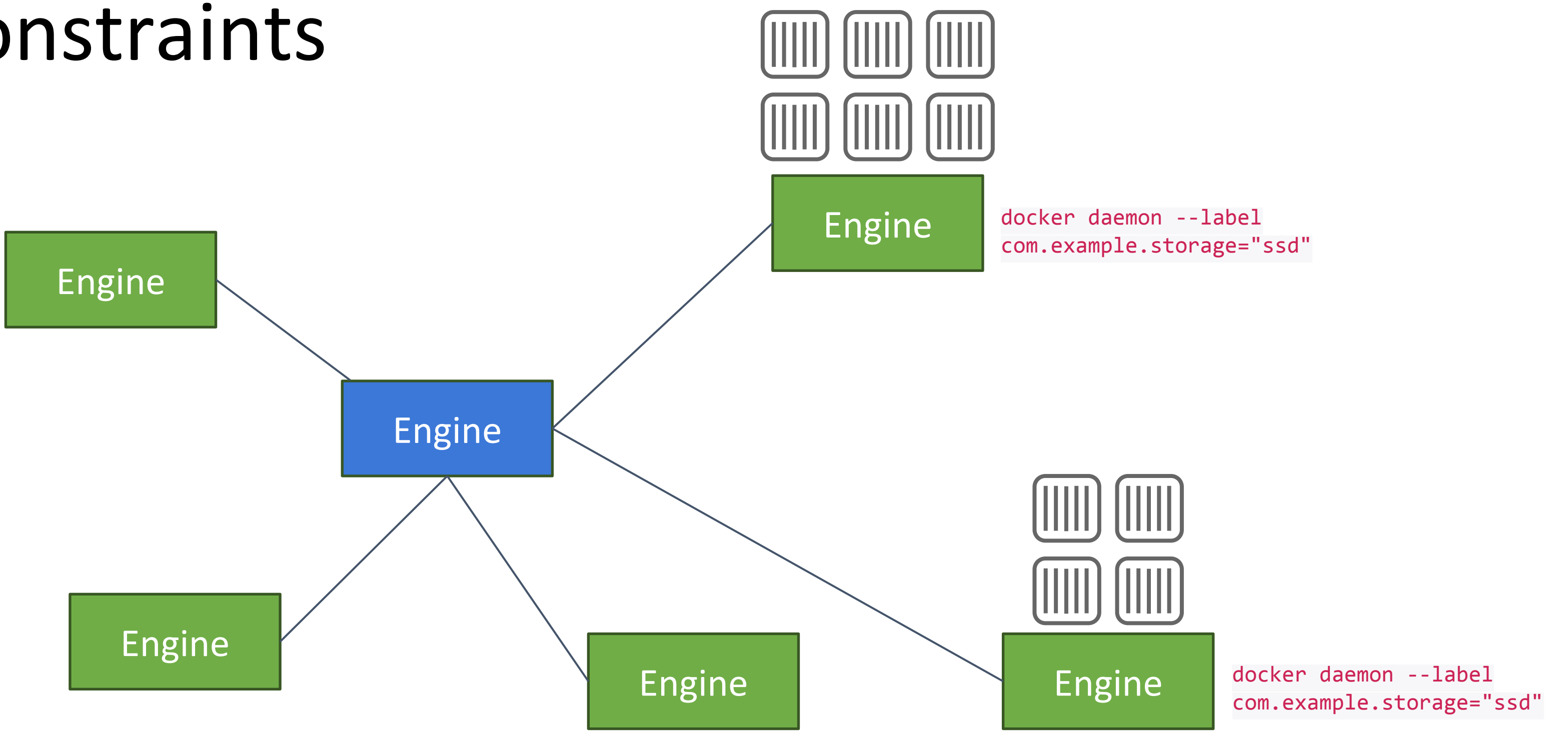
```
docker daemon --label
com.example.storage="ssd"
```

Constraints


Engine
Engine
Engine
Engine
Engine
Engine
docker daemon --label com.example.storage="ssd"
docker daemon --label com.example.storage="ssd"

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

Constraints

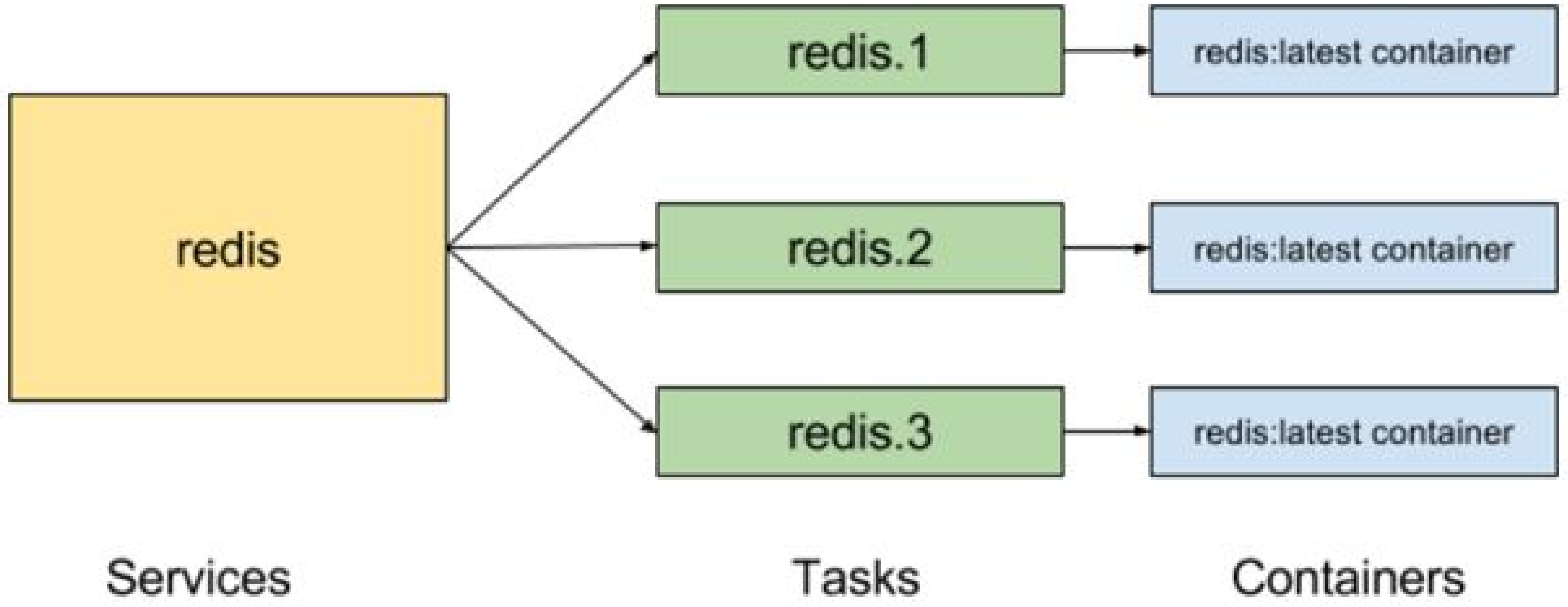


```
docker daemon --label
com.example.storage="ssd"
```

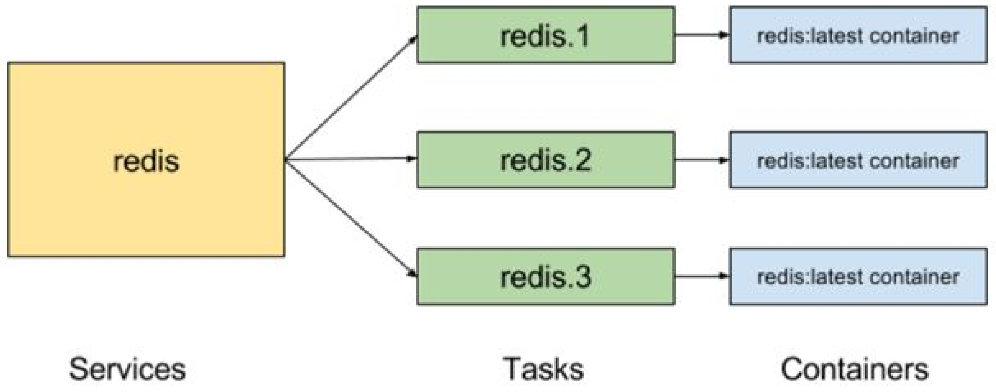
```
docker daemon --label
com.example.storage="ssd"
```

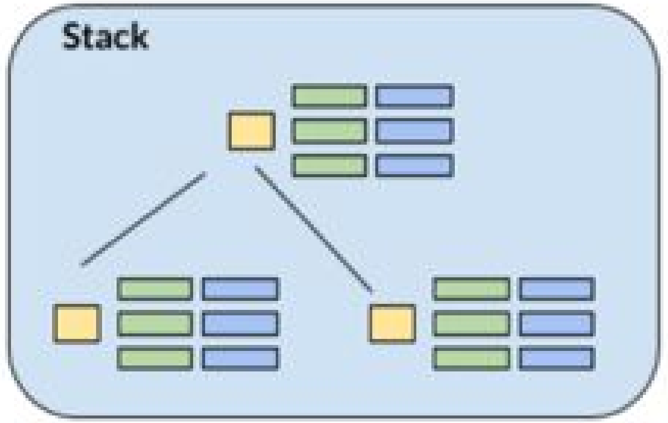
```
$ docker service create --replicas 3 --name frontend --network mynet
 --publish 80:80/tcp --constraint com.example.storage="ssd"
frontend_image:latest
$ docker service scale frontend=10
```

Services

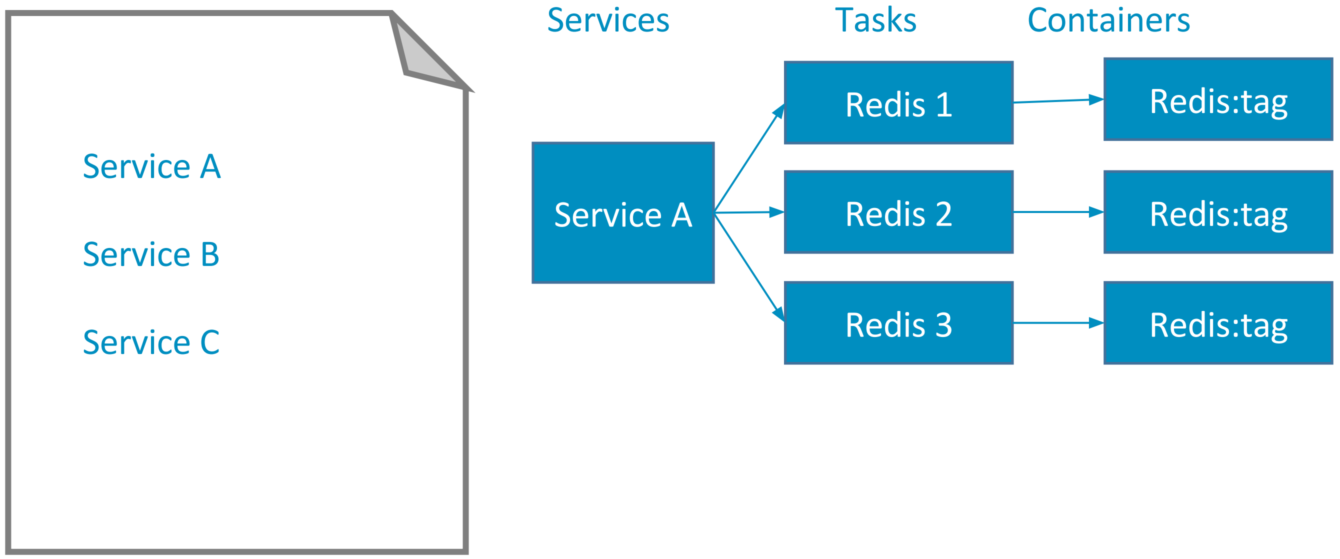

redis
redis.1
redis.2
redis.3
redis:latest container
redis:latest container
redis:latest container
Services
Tasks
Containers

Services are grouped into stacks





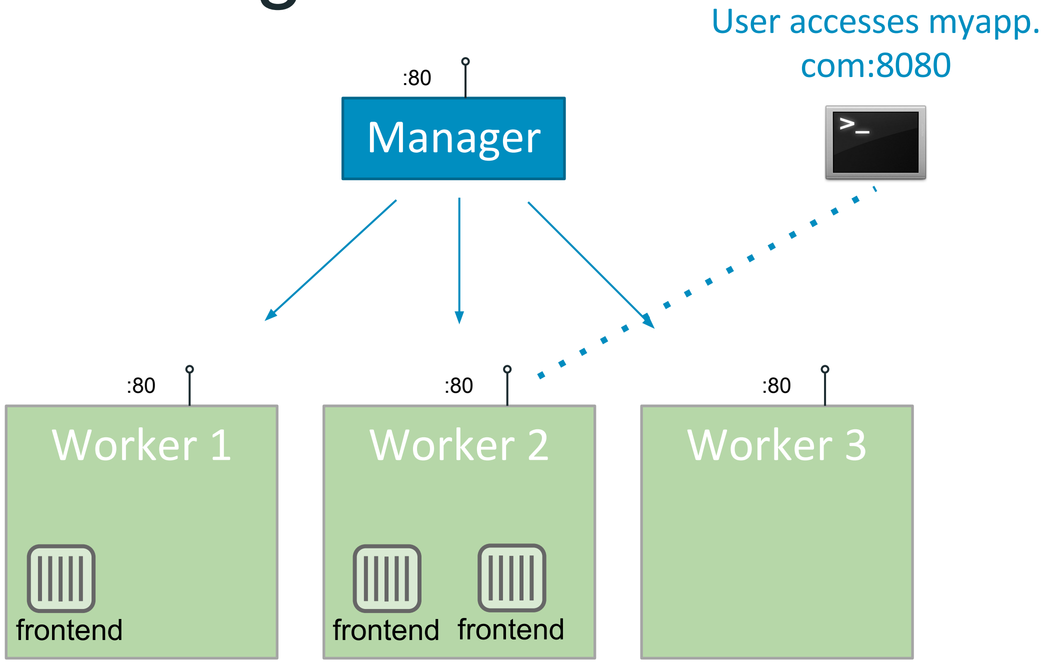
Distributed Application Bundle (.dab) declares a stack



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

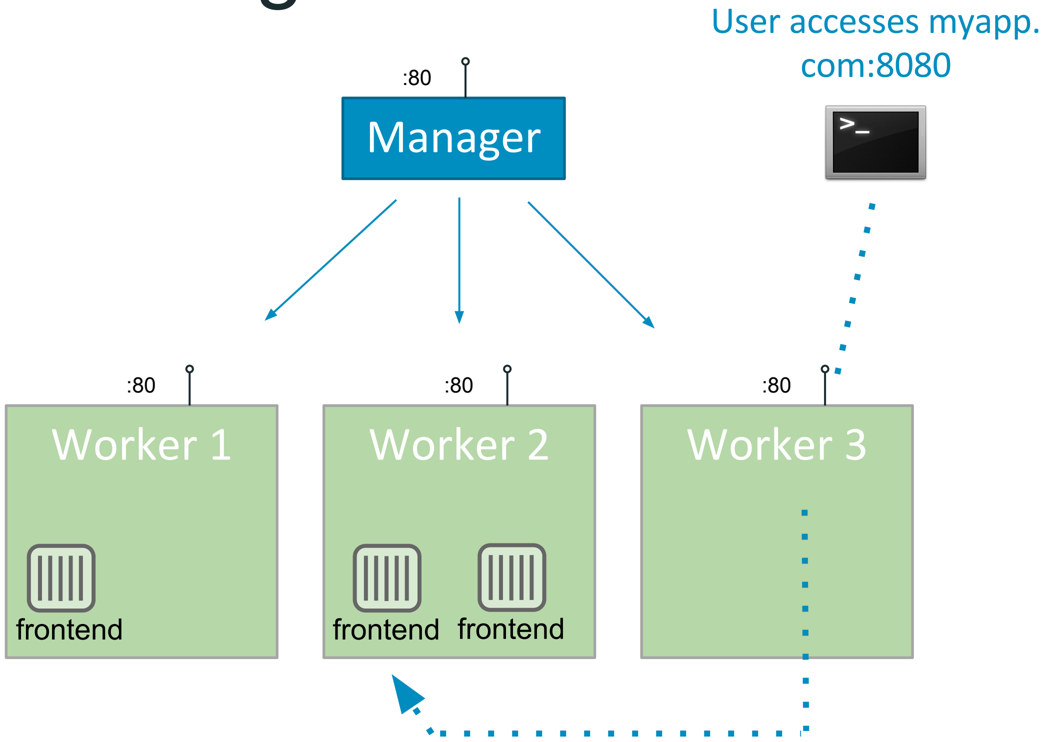
Routing Mesh



- Operator reserves a swarm-wide ingress port (80) for `myapp`
- Every node listens on 80
- 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
  --publish 80:80/tcp frontend_image:latest
```

Routing Mesh: Published Ports



- Operator reserves a swarm-wide ingress port (80) for `myapp`
- Every node listens on 80
- 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
 --publish 80:80/tcp 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.

New Plugin Subcommands

```
docker plugin install tiborvass/no-remove

docker plugin enable no-remove

docker plugin disable no-remove
```

Plugin Permissions Model

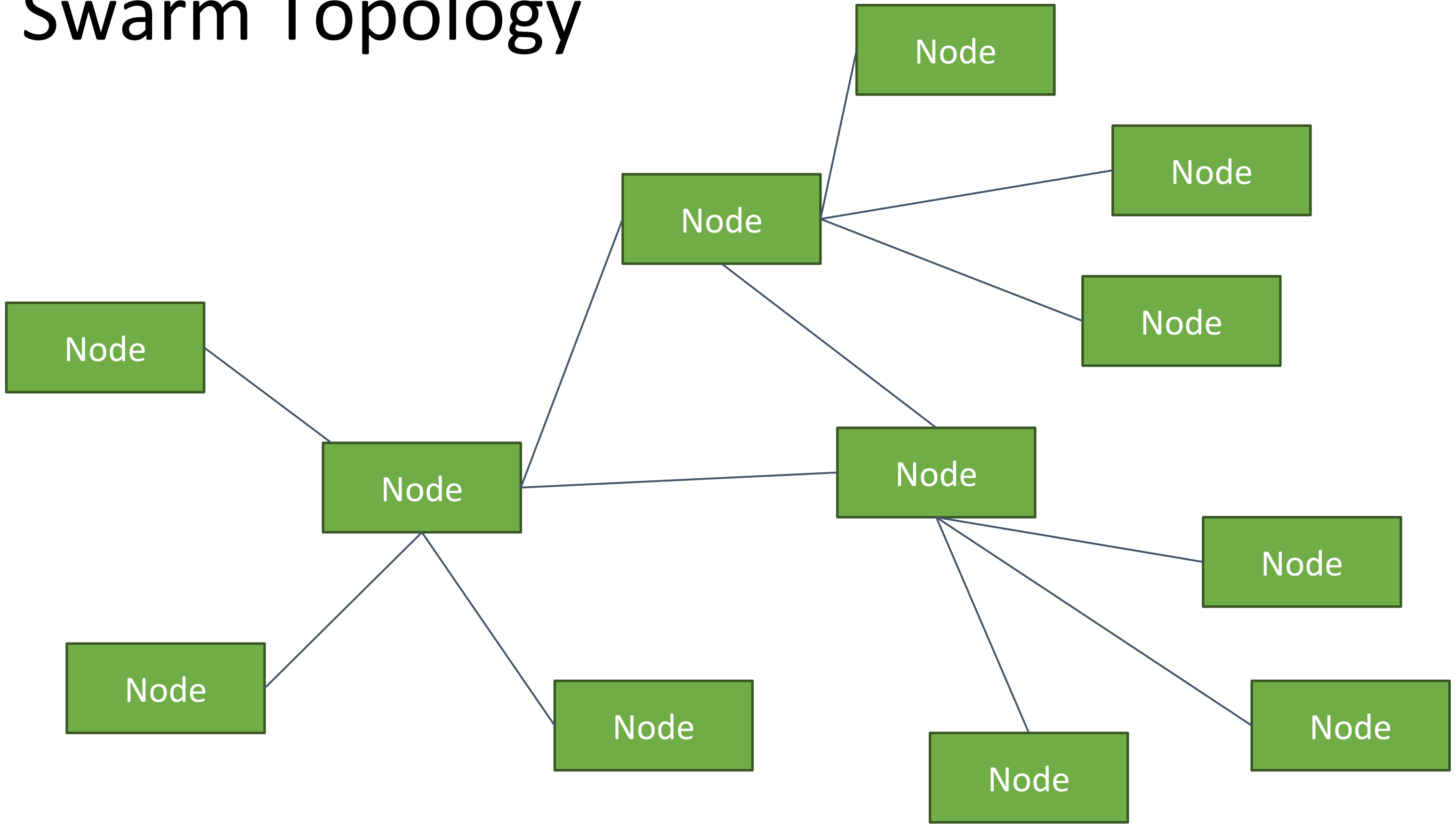
```
$ docker plugin install tiborvass/no-remove
Plugin "mikegoelzer/myplugin:latest"
requested the following privileges:
 - Networking: host
 - Mounting host path: /data
Do you grant the above permissions? [y/N]
```

Orchestration Deep Dive

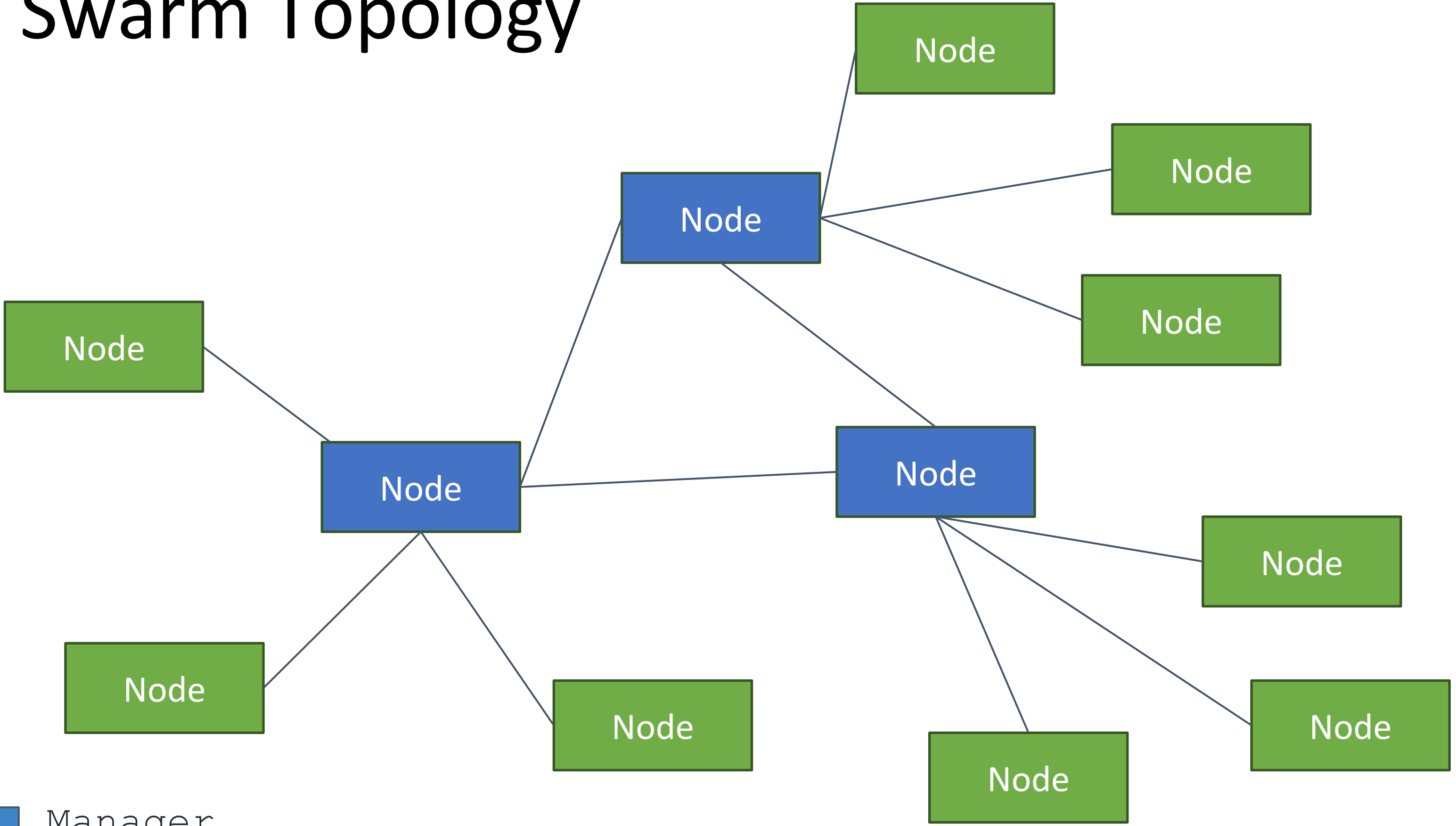
DockerCon 2016

Andrea Luzzardi

Swarm Topology



Swarm Topology

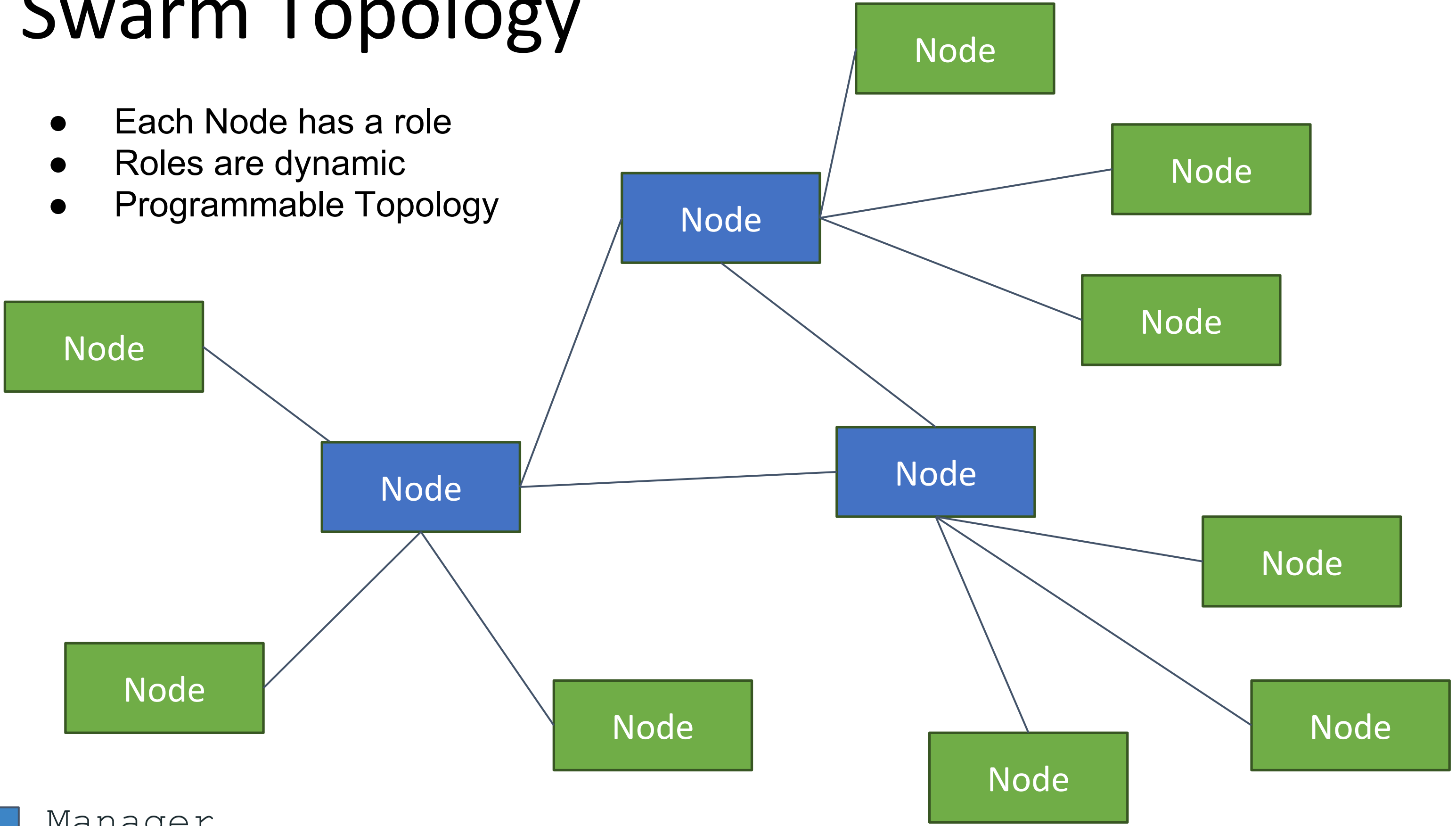


🟦 Manager

🟩 Worker

Swarm Topology

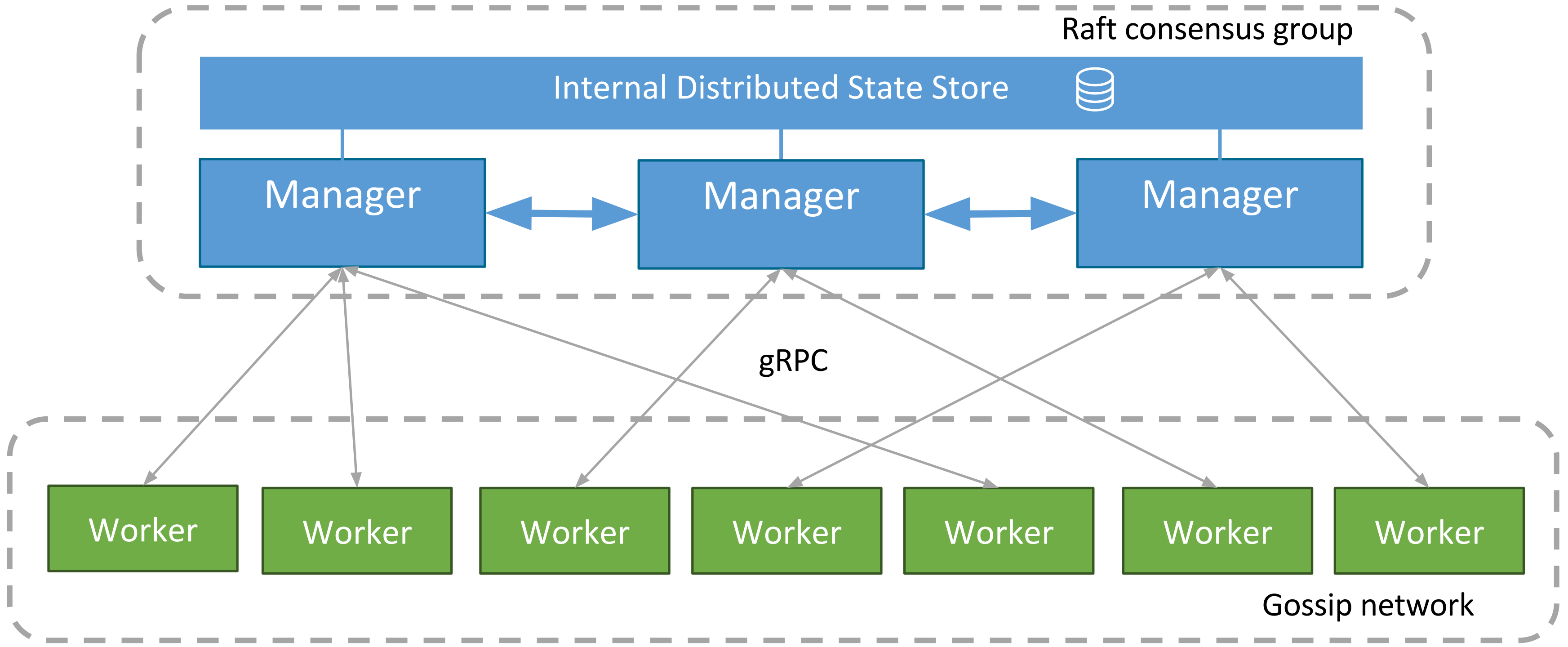
- Each Node has a role
- Roles are dynamic
- Programmable Topology



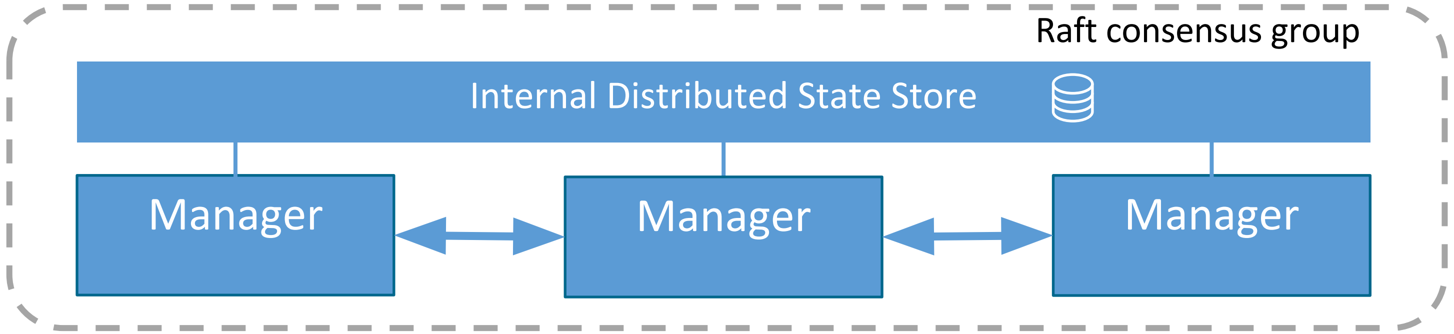
🟦 Manager

🟩 Worker

Docker Swarm Communication Internals

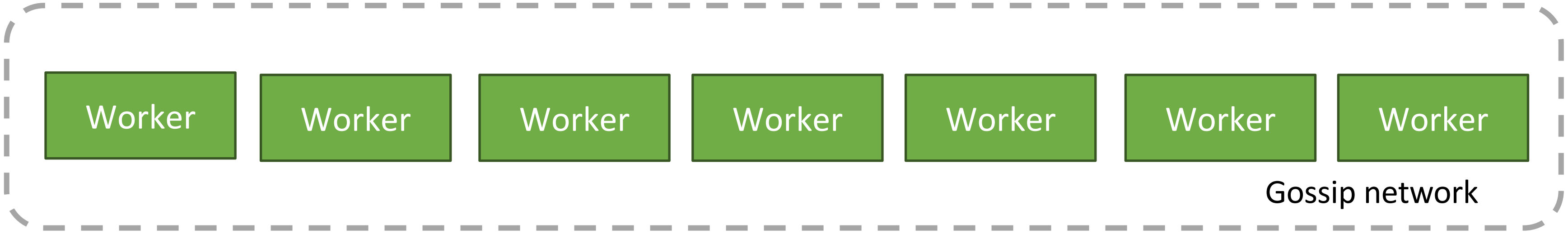


Quorum Layer



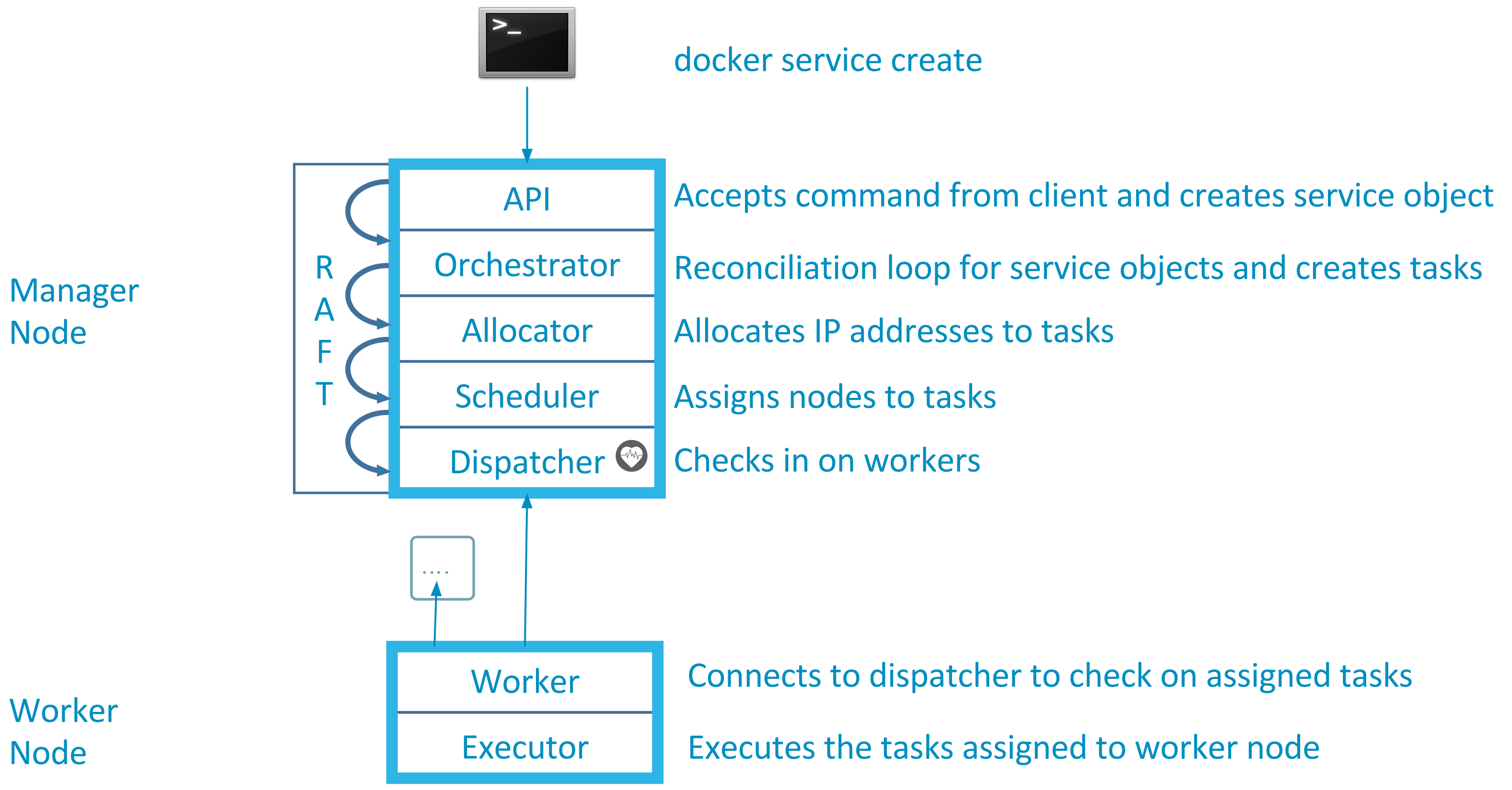
- Strongly consistent: Holds desired state
- Simple to operate
- Blazing fast (in-memory reads, domain specific indexing, ...)
- Secure

Worker-to-Worker Gossip



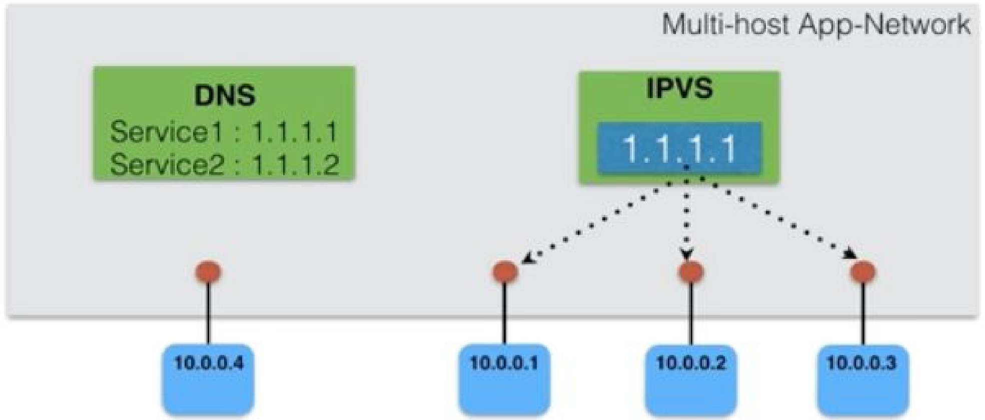
- Eventually consistent: Routing mesh, load balancing rules, ...
- High volume, p2p network between workers
- Secure: Symmetric encryption with key rotation in Raft

Node Breakdown



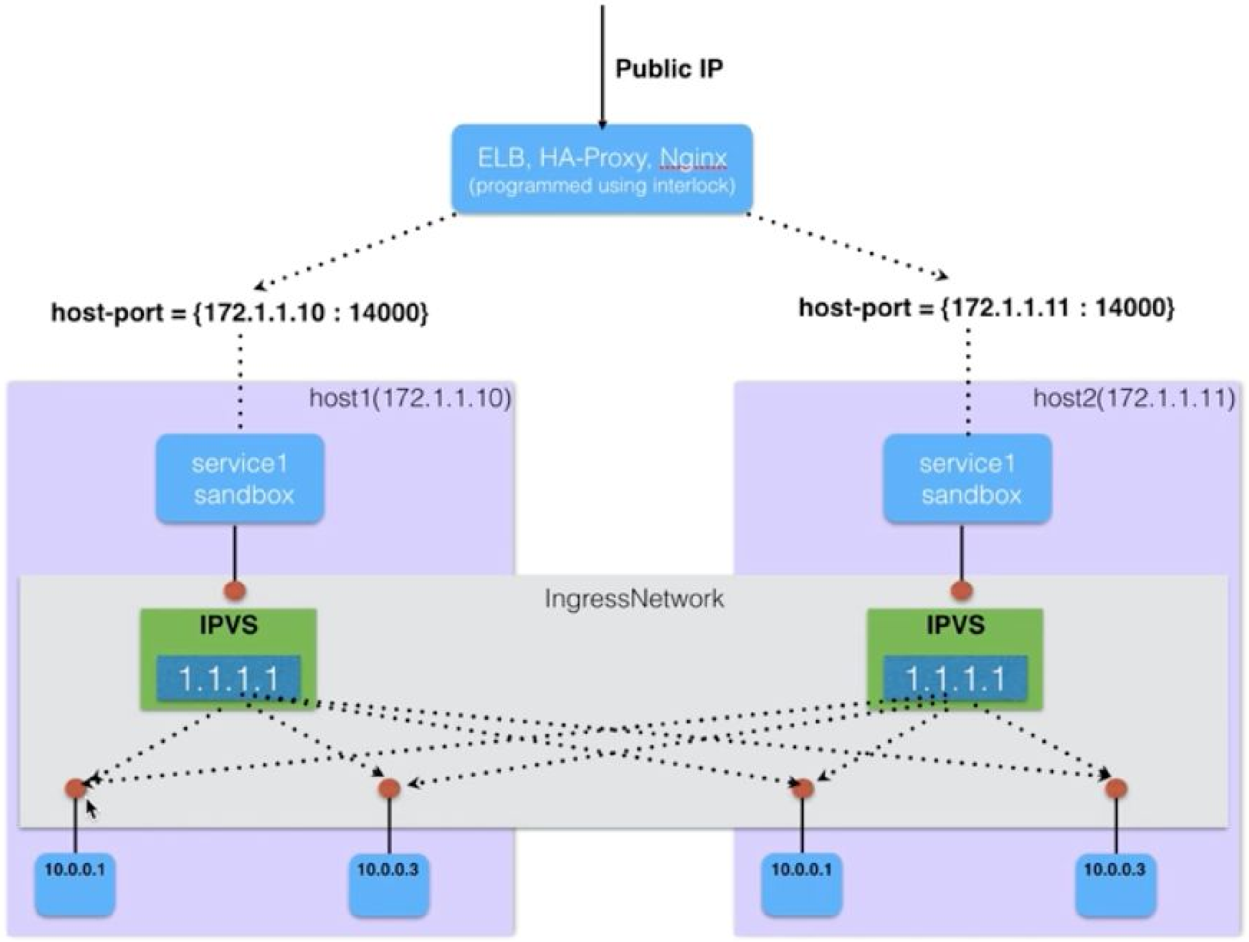
docker service create
Manager Node
RAFT
API
Accepts command from client and creates service object
Orchestrator
Reconciliation loop for service objects and creates tasks
Allocator
Allocates IP addresses to tasks
Scheduler
Assigns nodes to tasks
Dispatcher
Checks in on workers
....
Worker Node
Worker
Connects to dispatcher to check on assigned tasks
Executor
Executes the tasks assigned to worker node

Internal Load-Balancer

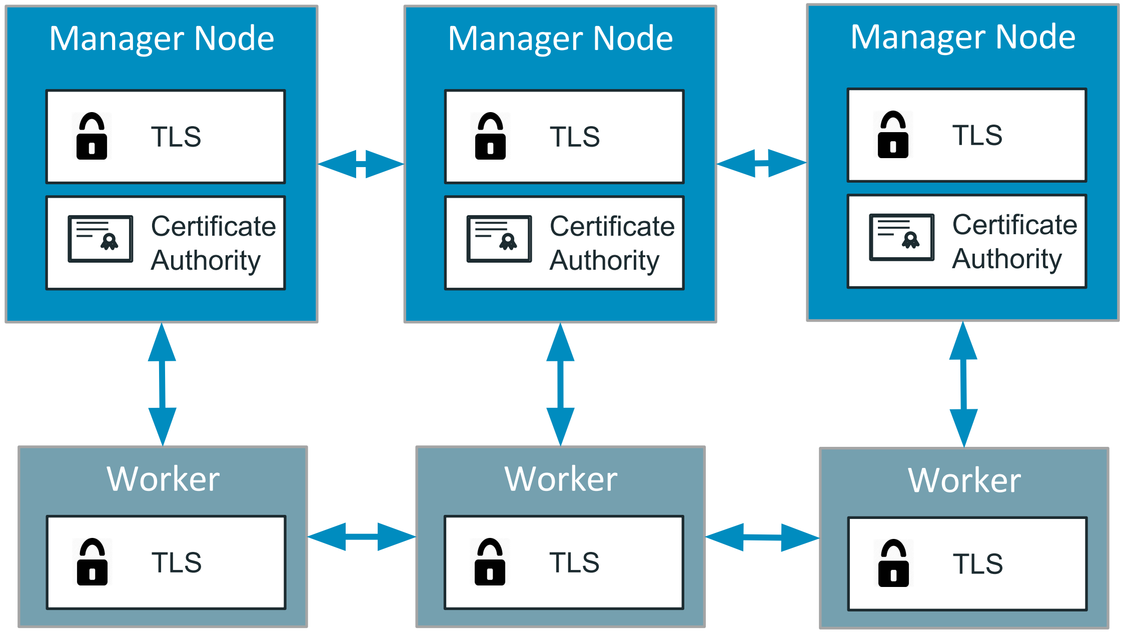


- **Load-balancer is designed as an integral part of CNM**
  - Works on top of CNM constructs (network, endpoint, sandbox, SD)
  - Every Service gets a Virtual-IP
  - Built-in SD resolves Service-Name -> VIP
  - Service VIP -> Container IP load balancing achieved using IPVS

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

Learn more about 1.12

Monday 5:20 pm @ Ballroom 6E

- Docker Security Deep Dive

Tuesday 3:55 pm @ Ballroom 6E

- Docker for Ops: Networking Deep Dive, Considerations and Troubleshooting

Questions?

Mike Goelzer
mgoelzer@docker.com / @mgoelzer

Andrea Luzzardi
al@docker.com / @aluzzardi

Thank You

Mike Goelzer
mgoelzer@docker.com / @mgoelzer

Andrea Luzzardi
al@docker.com / @aluzzardi