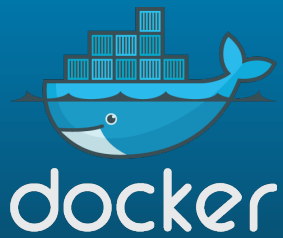
Orchestrating Docker Containers with SwarmKit

Docker Meetup Bangalore

January 7, 2017

Nishant Totla, Core Team Software Engineer, Docker Inc.

@nishanttotla



Nishant Totla

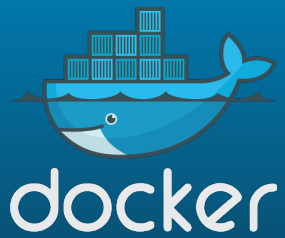
Docker, Inc.
nishant@docker.com
github.com/nishanttotla
@nishanttotla

docker

DockerCon 2017 CFPs open!

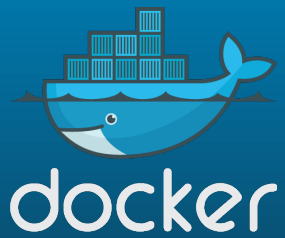
Deadline is January 14

More info: http://2017.dockercon.com/



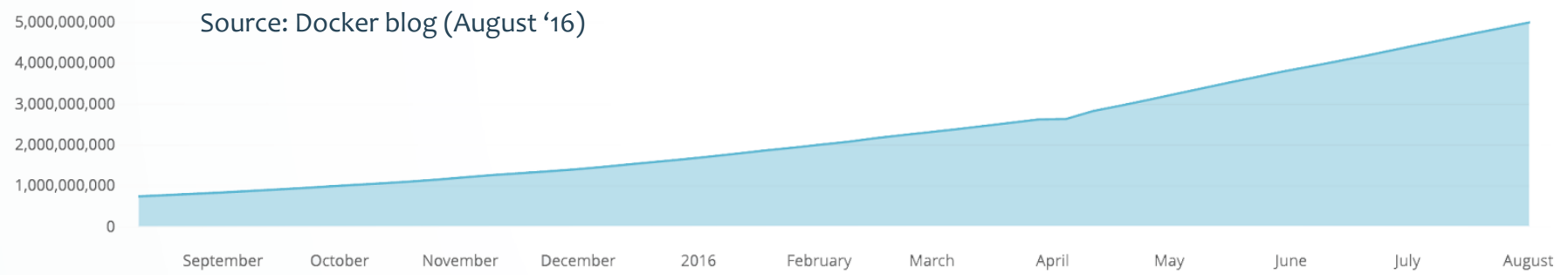
What is a Linux Container?

- A Linux container is a self-contained execution environment with isolated resources
- The technology has been around for a while - LXC, FreeBSD Jails, Solaris Zones
- The Docker project, open-sourced in 2013 made containers extremely popular and significantly easier to use



What is a Linux Container?

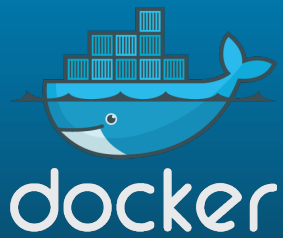
- A Linux container is a self-contained execution environment with isolated resources
- The technology has been around for a while - LXC, FreeBSD Jails, Solaris Zones
- The Docker project, open-sourced in 2013 made containers extremely popular and significantly easier to use



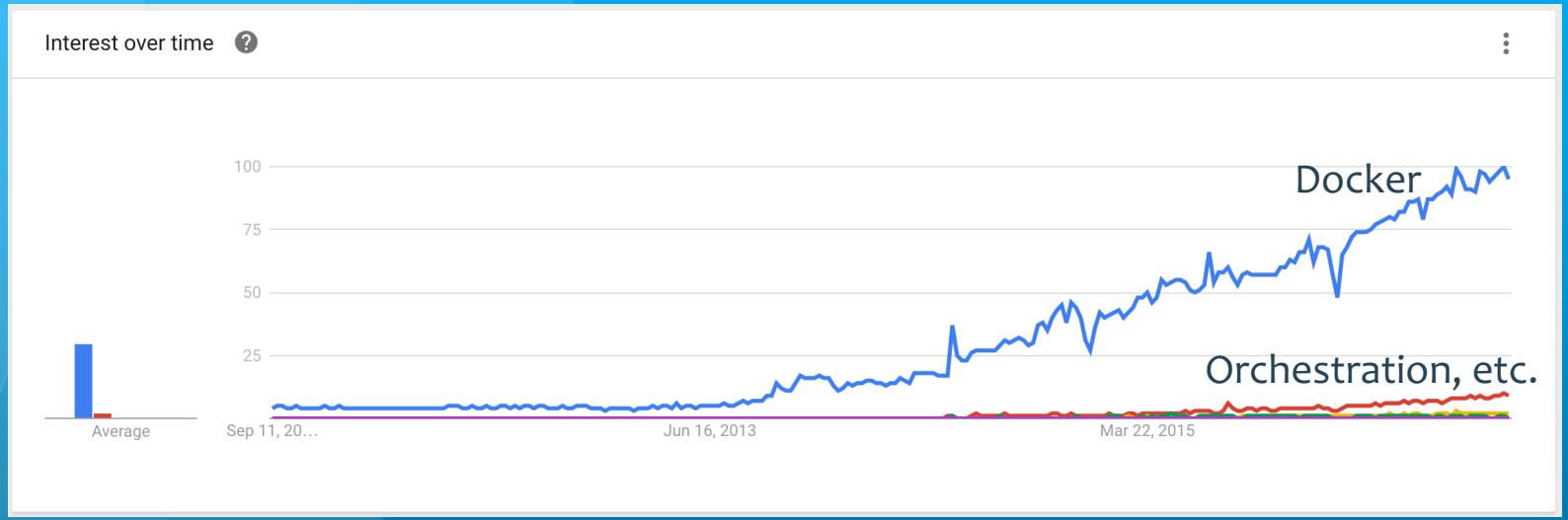
docker

Real-world Applications

- The growth of containers has brought a fresh perspective on deploying containerized applications, particularly microservices
- This brings up interesting problems
  - Infrastructure
  - Deployment
  - Security
  - Networking
  - Scaling
  - Fault-tolerance
- New tools and abstractions needed
- Container Orchestration is the key!



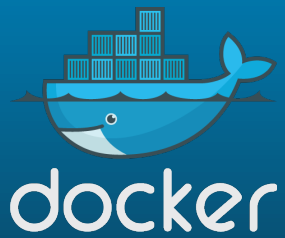
Trends



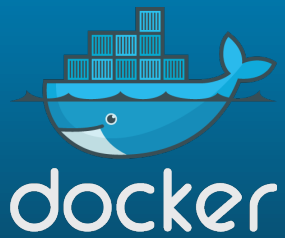
docker

Orchestration is Hard

- Distributed Systems are hard to get right
- Failures happen all the time!
  - Machines go down
  - Networks can fail or drop packets
  - Latency can cause unexpected scenarios

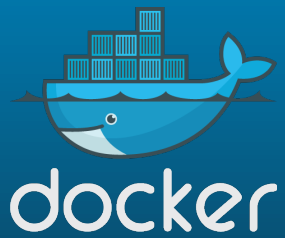


Can we make it easier to orchestrate real-world containerized applications?



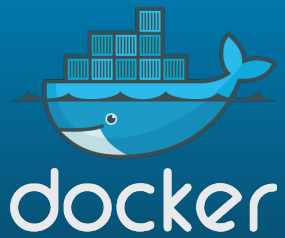
Can we make it easier to orchestrate real-world containerized applications?

→ Swarm Mode in Docker 1.12



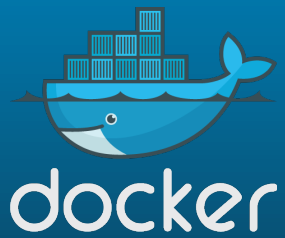
Swarm Mode in Docker 1.12

- Goals
  - It should be simple to setup and use a distributed cluster
  - It should be seamless to move from development to deployment



Swarm Mode in Docker 1.12

- Goals
  - It should be simple to setup and use a distributed cluster
  - It should be seamless to move from development to deployment
- Starting with 1.12, Docker now has inbuilt orchestration
  - The easiest way to orchestrate Docker containers is now Docker
  - Multiple Docker Engines can be brought together to form a Swarm cluster
  - Really easy to manage applications on the cluster

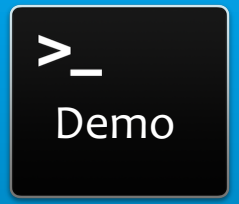


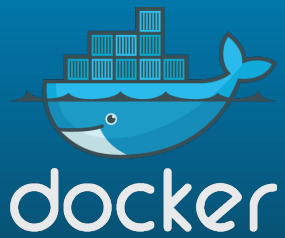
The Docker approach to container orchestration

1. Swarm Topology

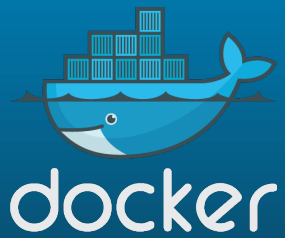
2. Managing Cluster State

3. Managing Application State





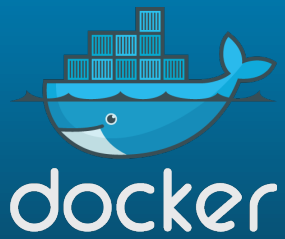
Part 1: Swarm Topology



Single Node Cluster

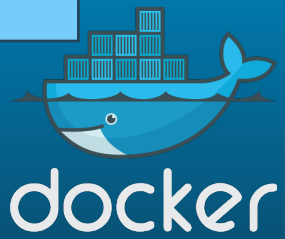
Engine

```
$ docker swarm init --advertise-addr 198.211.109.17
```

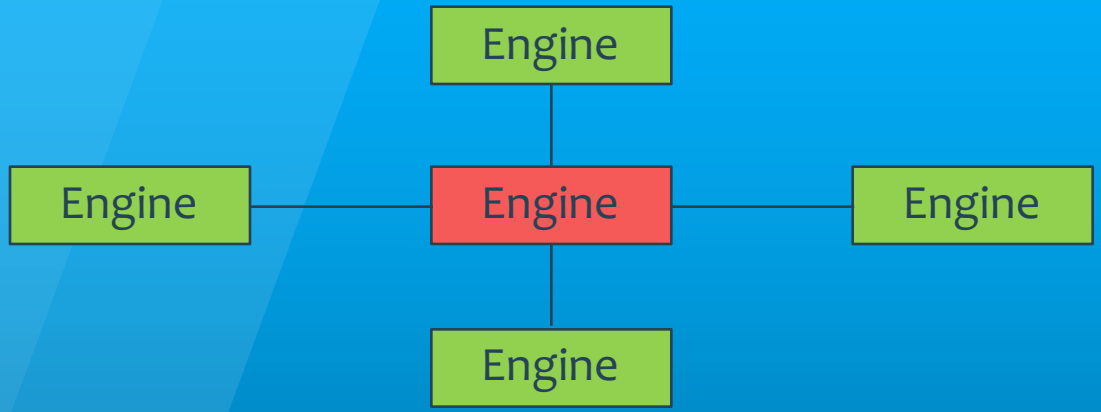


Adding another Node

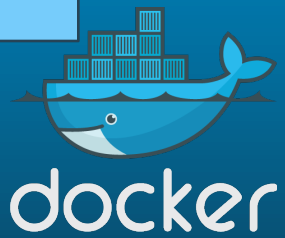

```
$ docker swarm init --advertise-addr 198.211.109.17
$ docker swarm join 198.211.109.17:2377
```



More Nodes

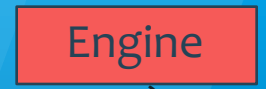


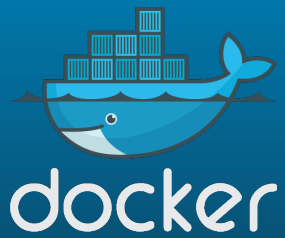

```
$ docker swarm init --advertise-addr 198.211.109.17
$ docker swarm join 198.211.109.17:2377
```



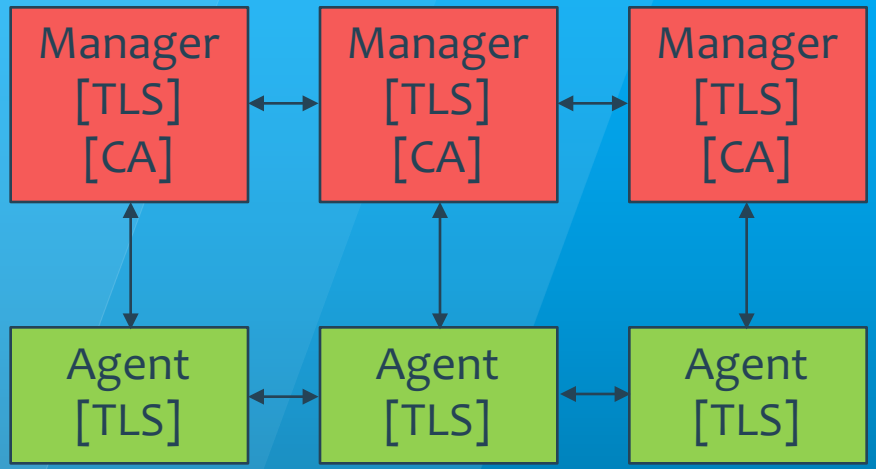
Managers and Workers

- Swarm nodes can be workers or managers
- Worker nodes
  - Run Tasks (containers)
- Manager nodes
  - Run Tasks (containers)
  - Accept specifications from the user
  - Reconcile desired and and actual cluster state
- Workers can be promoted to managers
- Managers can be demoted to workers

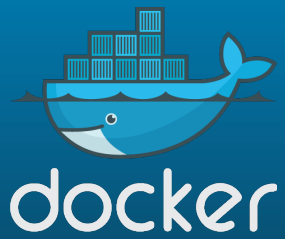




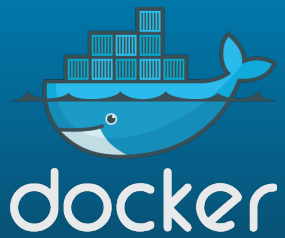
Secure by Default with end to end Encryption



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

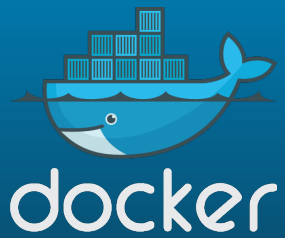


Part 2: Managing Cluster State

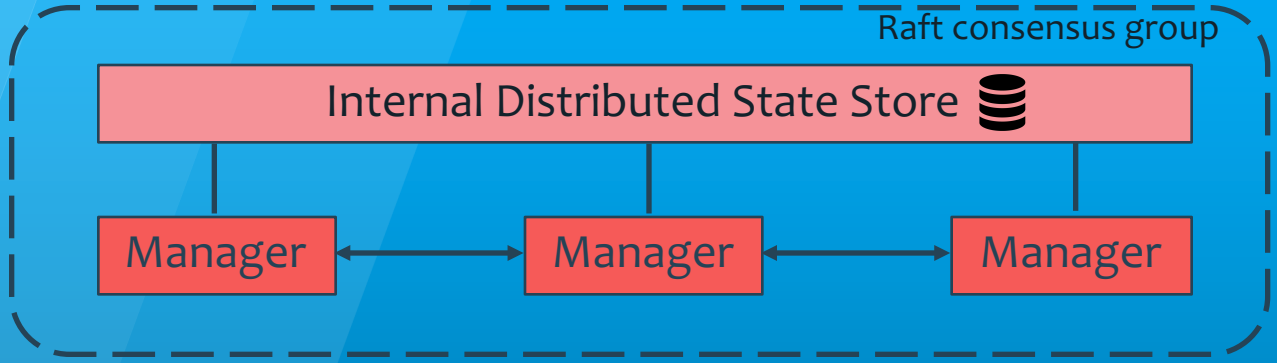


The Raft Algorithm

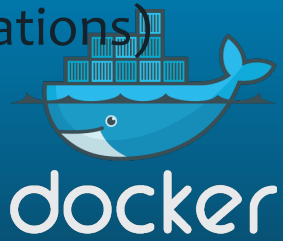
- Raft is a popular distributed consensus algorithm
- Features
  - Single leader performs all updates
  - Leader election – elect a leader
  - Log replication – replicate the log of cluster operations
- Consensus is maintained as long as a ***majority*** of nodes are working



Raft Consensus Layer

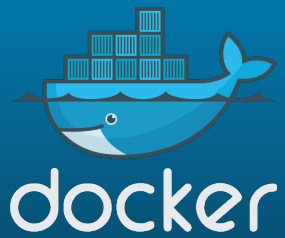


- Built on top of the etcd Raft library
- Strongly consistent - holds desired state
- Fast (in-memory reads, domain-specific indexing, batched operations)
- Secure

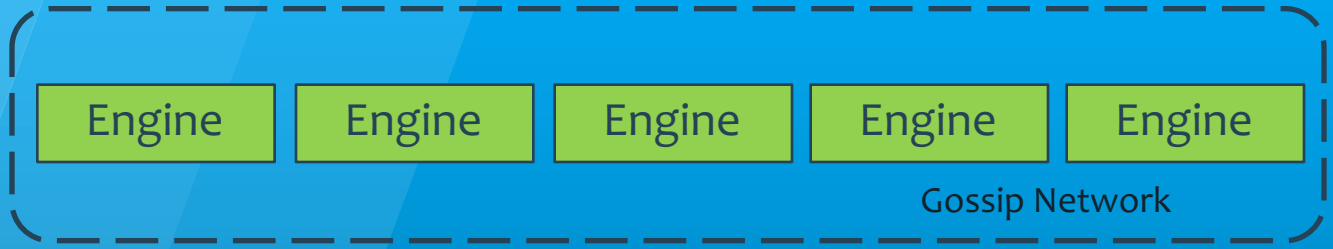


Raft Consensus Layer

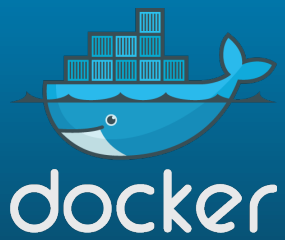
- Swarm managers are crucial
- Manager nodes form a Raft consensus group
- No need to set up an external KV store
- As long as a majority of managers are running, the cluster will function normally (should start with an odd number)
- All managers can accept API requests (HA, no single point of failure)



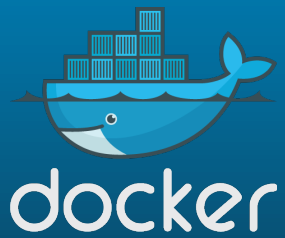
Worker to Worker Gossip



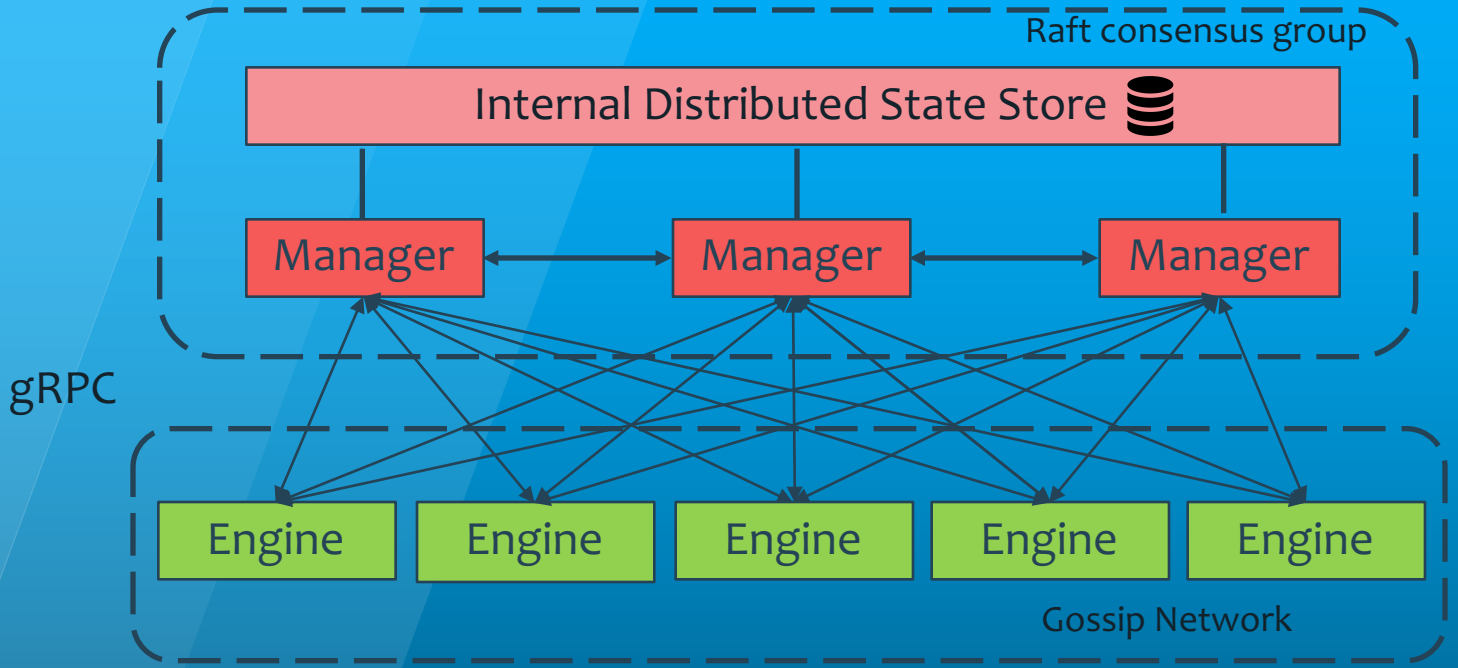
- Different from Raft to keep Raft concise
- High volume, p2p network between workers
- Eventually consistent – networking, load balancing rules, etc.
- Secure: symmetric encryption with key rotation in Raft



Putting it All Together

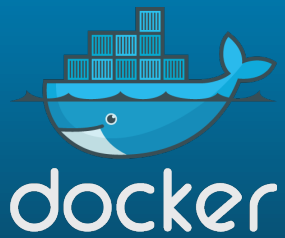


Full Architecture



docker

Part 3: Managing Application State



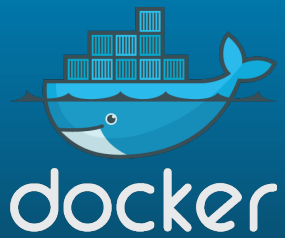
A Higher Level of Abstraction

- From containers to services

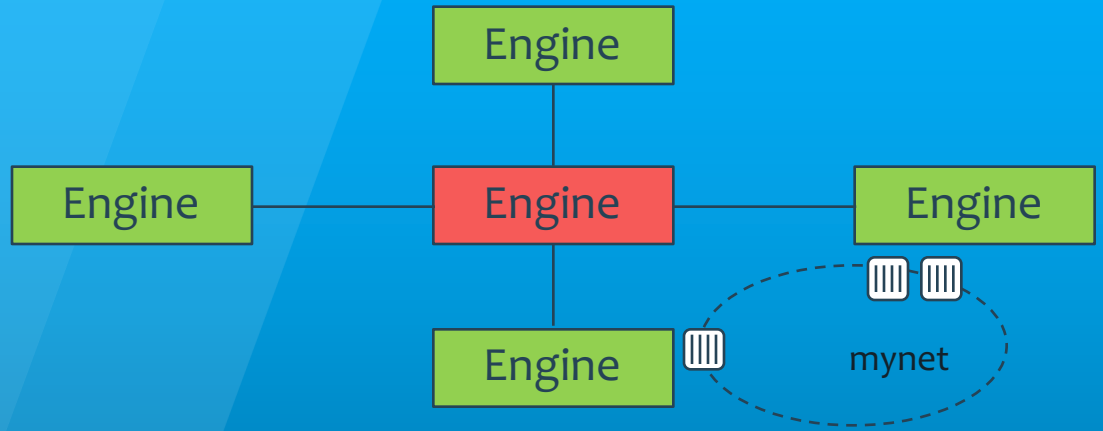
```
$ docker run
```

```
$ docker service create
```

- Docker services
  - Allow declarative specification
  - Work seamlessly on any number of nodes
  - Perform desired state reconciliation
  - Can be scaled or updated easily



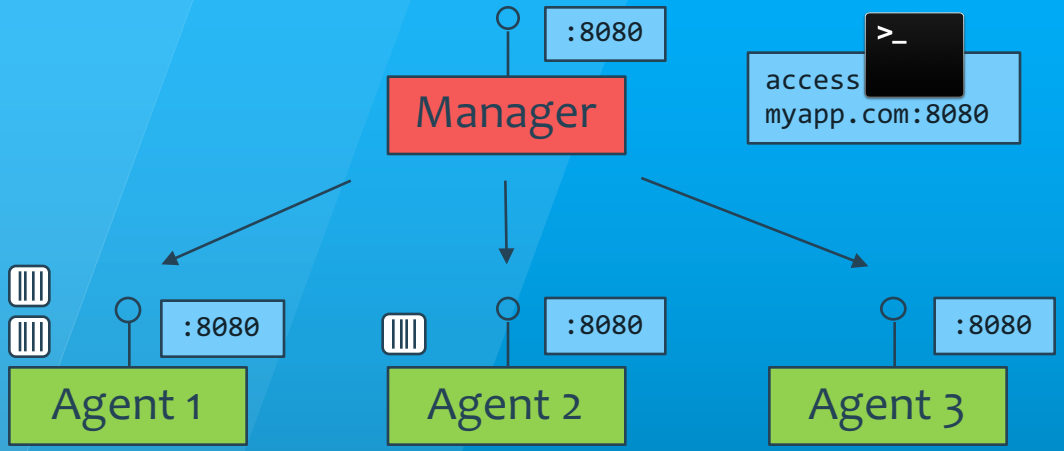
Services




```
$ docker service create \
  --replicas 3 \
  --name frontend \
  --network mynet \
  --publish 8080:80 \
  frontend_image:1.0
```

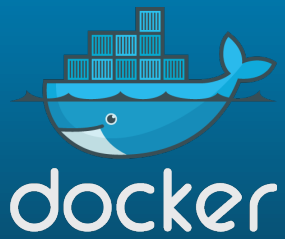
docker

Swarm Routing Mesh

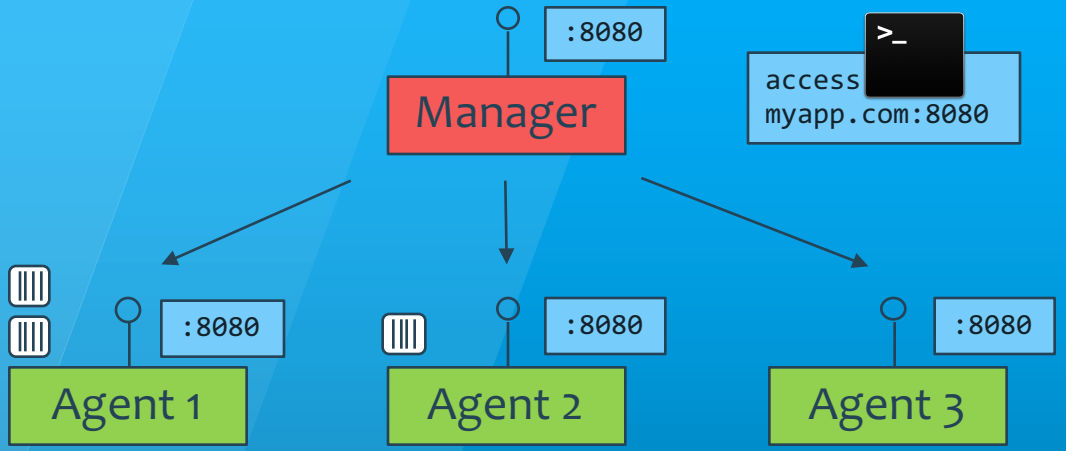



```
$ docker service create \
  --replicas 3 \
  --name frontend \
  --network mynet \
  --publish 8080:80 \
  frontend_image:1.0
```

- All nodes expose port 8080 for myapp
- Routing mesh ensures traffic is routed transparently to a running container of the service

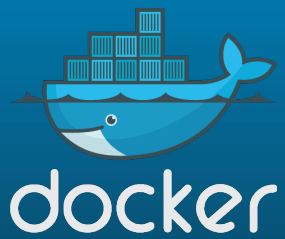


Swarm Routing Mesh

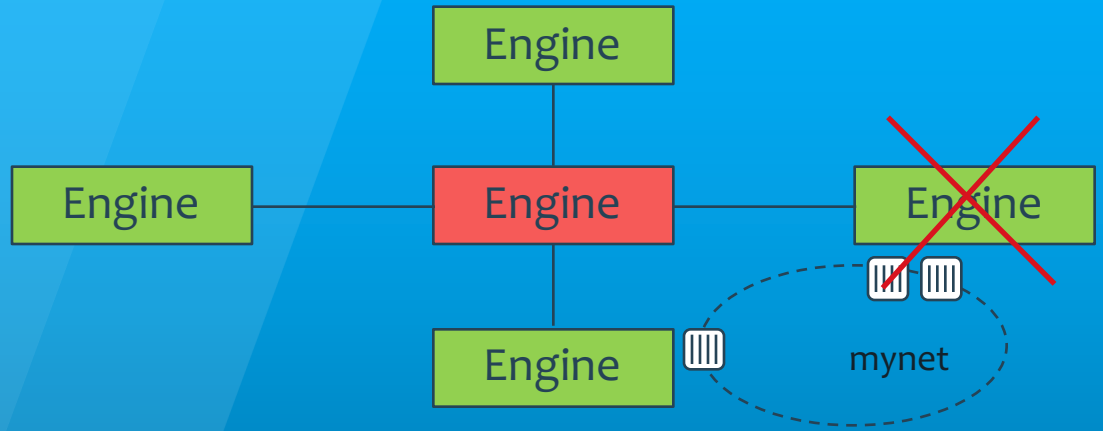



```
$ docker service create \
  --replicas 3 \
  --name frontend \
  --network mynet \
  --publish 8080:80 \
  frontend_image:1.0
```

- Every service gets assigned a virtual IP (VIP)
- Built-in DNS-based service discovery: service name resolves to service VIP
- VIP round-robins between container IPs using Linux IPVS



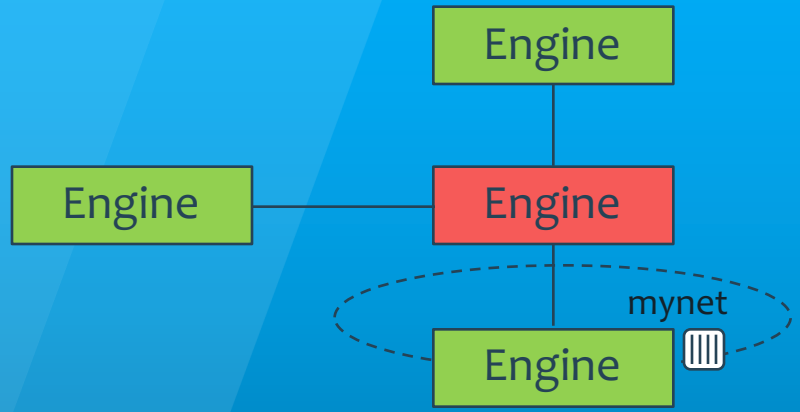
Node Failure



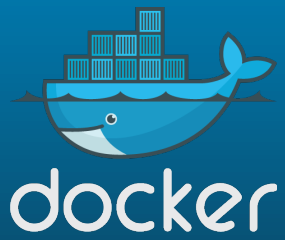

```
$ docker service create \
  --replicas 3 \
  --name frontend \
  --network mynet \
  --publish 8080:80 \
  frontend_image:1.0
```

docker

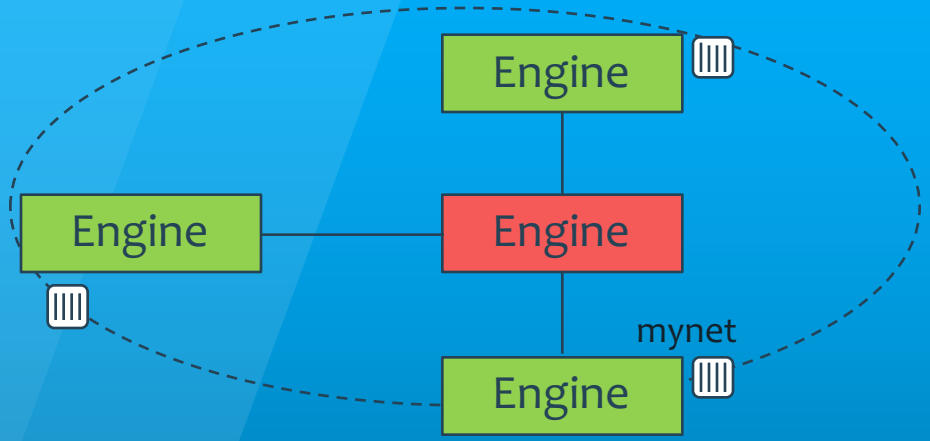
Desired State ≠ Actual State



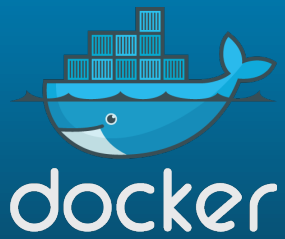

```
$ docker service create \
  --replicas 3 \
  --name frontend \
  --network mynet \
  --publish 8080:80 \
  frontend_image:1.0
```



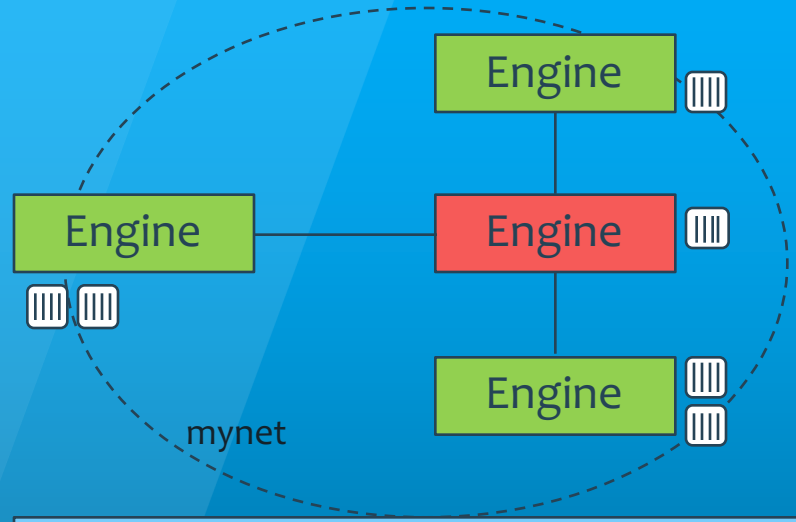
Converge back to Desired State



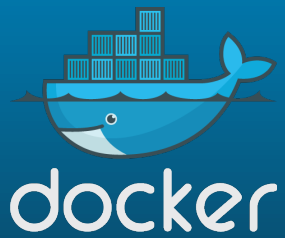

```
$ docker service create \
  --replicas 3 \
  --name frontend \
  --network mynet \
  --publish 8080:80 \
  frontend_image:1.0
```



Service Scaling

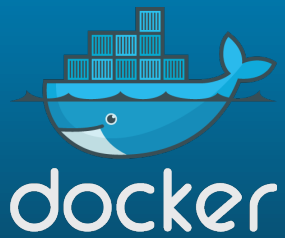



```
$ docker service scale frontend=6
```

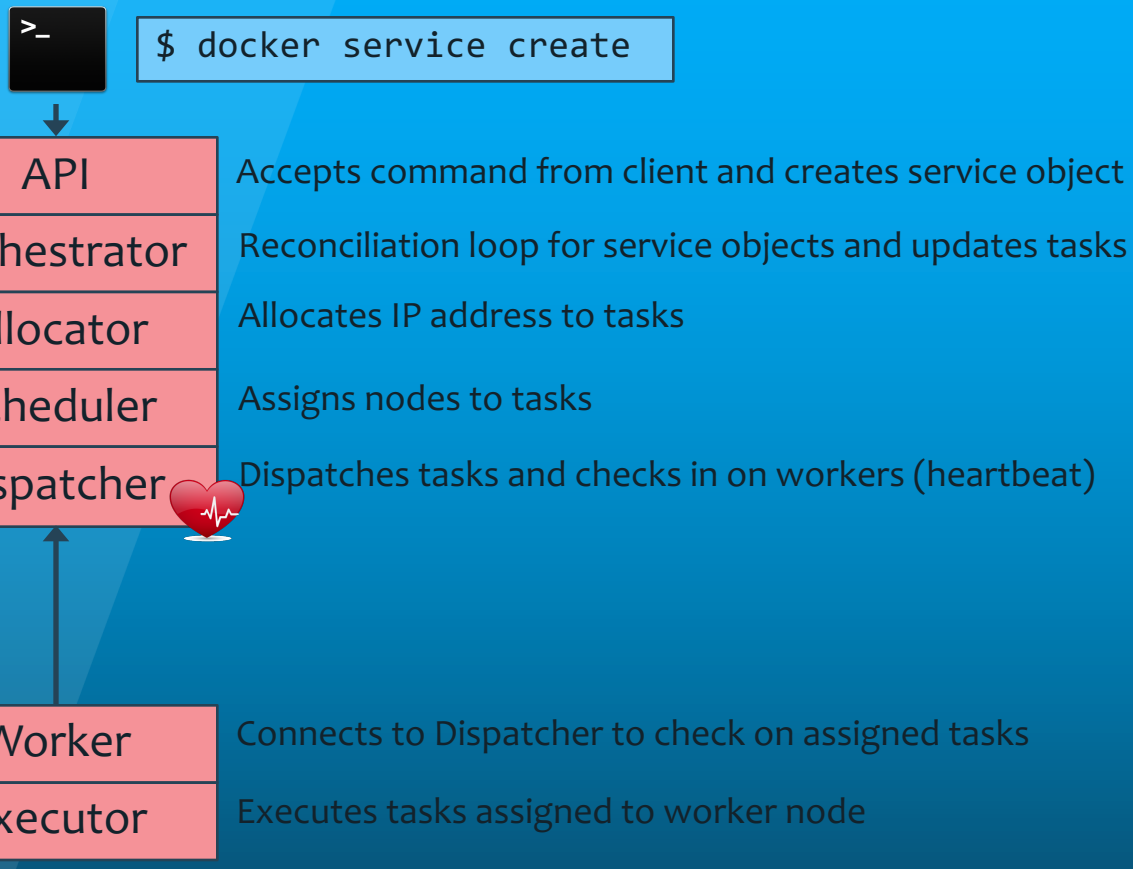


And more!

- Global services
- Configurable rolling updates
- Constraints
- Restart policies
- Resource aware scheduling



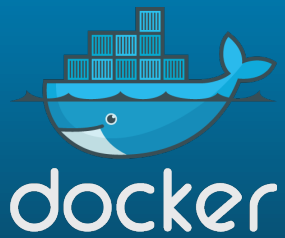
Behind the scenes



docker

Node Failure

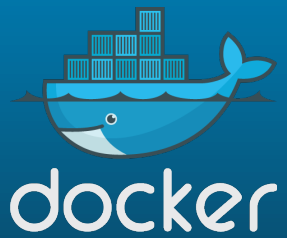
- Manager failure
  - Handled by the Raft algorithm, as long as consensus is maintained
- Worker failure
  - Dispatcher detects a node is down (failed heartbeat)
  - Orchestrator reconciles tasks
  - Allocator, Scheduler, and Dispatcher redo the previous steps to set up new tasks



If Everything Fails

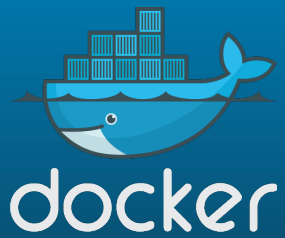
```
$ docker swarm init \
  --advertise-addr 198.211.109.17 \
  --force-new-cluster
```

- The flag forces an existing node that was part of a lost quorum to restart as a single node Manager without losing its data



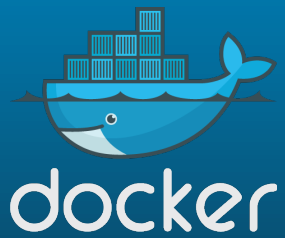
To Conclude

- Swarm mode in Docker makes it very easy to create and manage a failure-tolerant cluster
- Networking and security are handled out of the box
- The service abstraction allows for a declarative specification of how the application is expected to run
- Much more coming up in the next few releases!

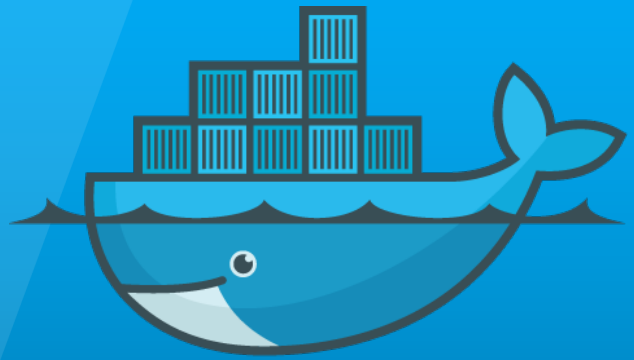


Contributions welcome!

- Docker open-source on GitHub
- github.com/docker/swarmkit | github.com/docker/docker



Thank You!

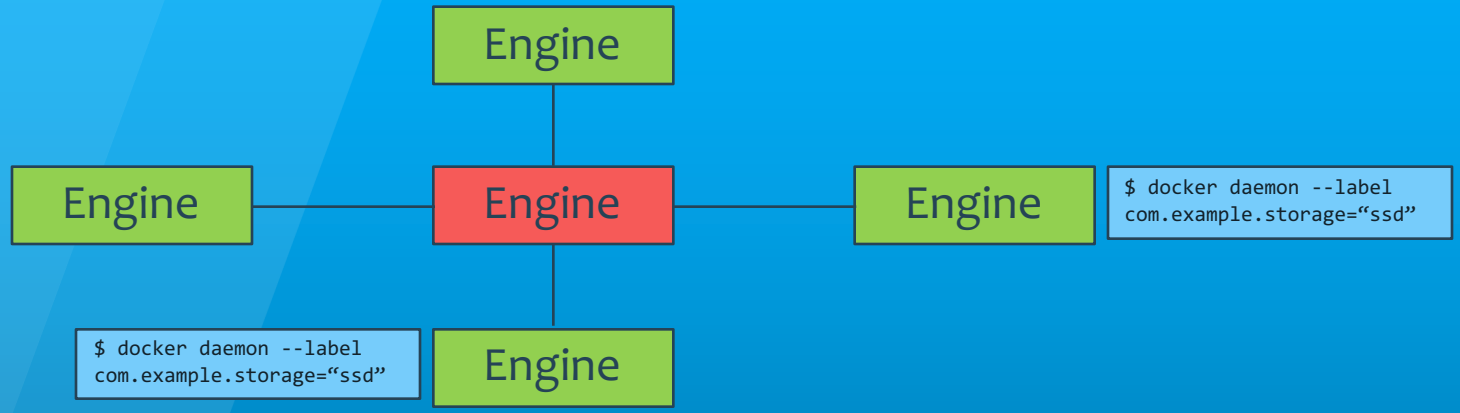


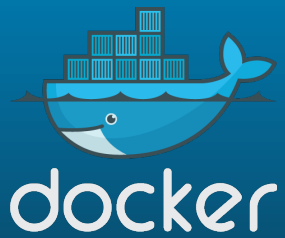
docker

Nishant Totla

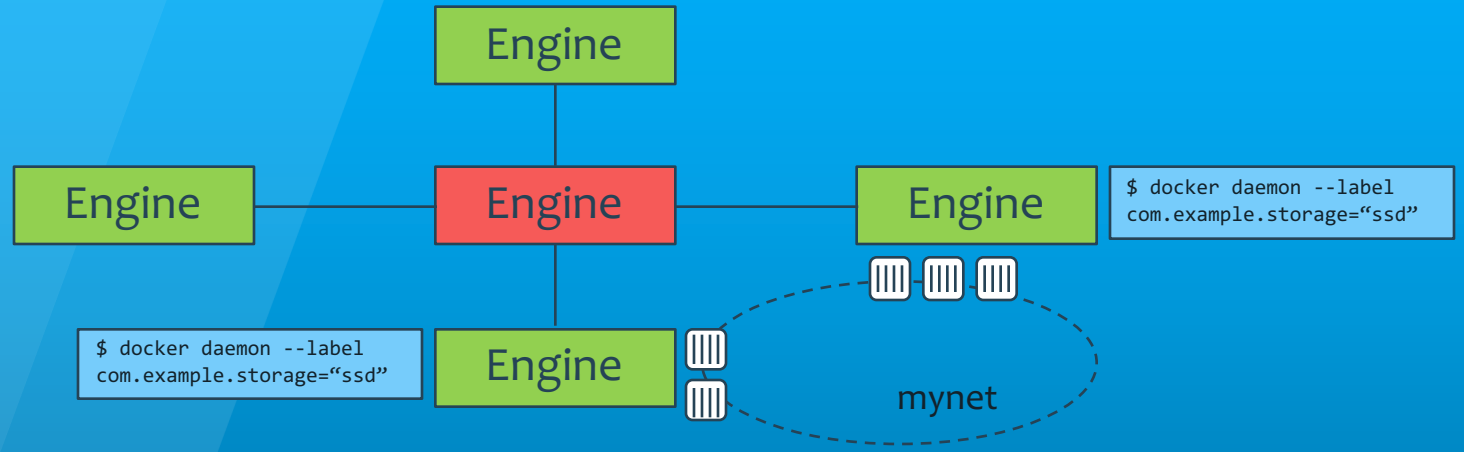
@nishanttotla

Constraints

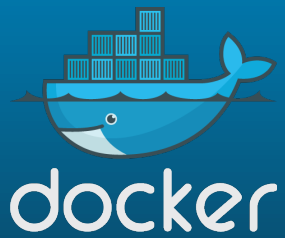




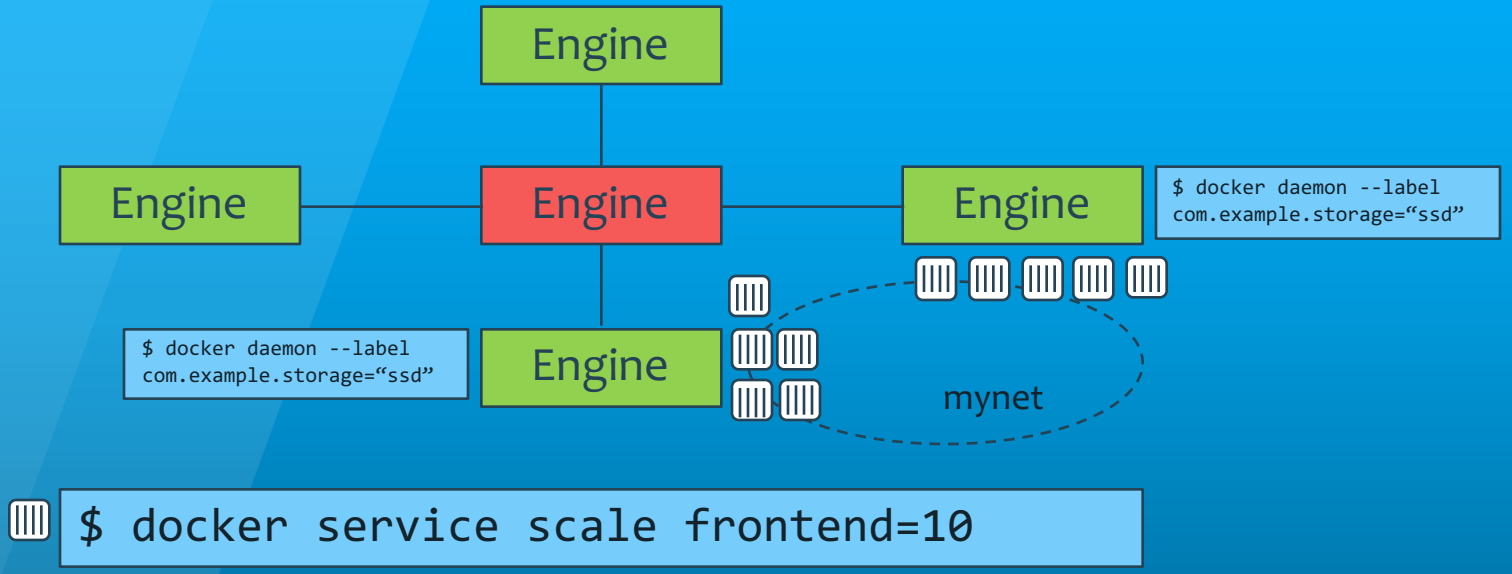
Constraints




```
$ docker service create \
  --replicas 5 \
  --name frontend \
  --network mynet \
  --publish 8080:80 \
  --constraint engine.labels.com.example.storage==ssd
  frontend_image:1.0
```



Constraints




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