Docker From The Ground Up Part 3

Multihost, Swarm and Production

Thanks to our Sponsor!

BlackCat /

Who's this guy?

Matt Todd

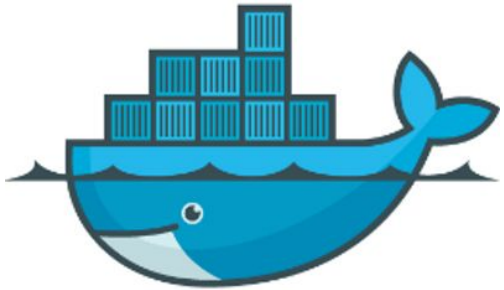
Principal Architect @ Resonate.tech

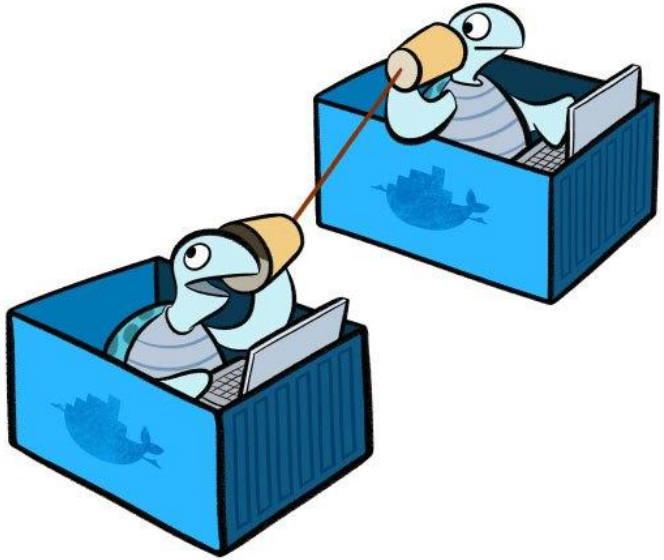
Cloud Native Advocate

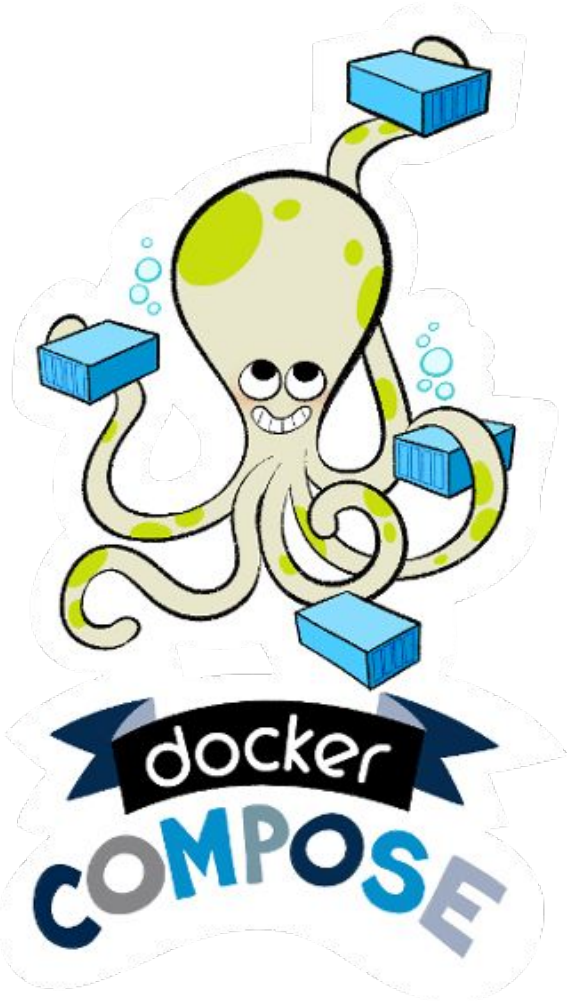
`github.com/mattjtodd`

`hub.docker.com/u/mattjtodd`

Recap!






docker
COMPOSE

Containers

- Linux Namespaces and CGroups
- Process resource isolation and constraint definition
- Share host CPU / Memory
- Lighter weight than VMs
- Fast startup times
- Docker Portability / Abstraction

Networking

- Namespaced networking stack per container (can be shared)
- Isolated by design
- Extensive use of bridge networking (Virtual switches)
- Multi-host capability (with swarm)
- Portable and able to work in many networking stacks
- Network Types (None, Host, Bridge, User-defined, MacVlan, Overlay)

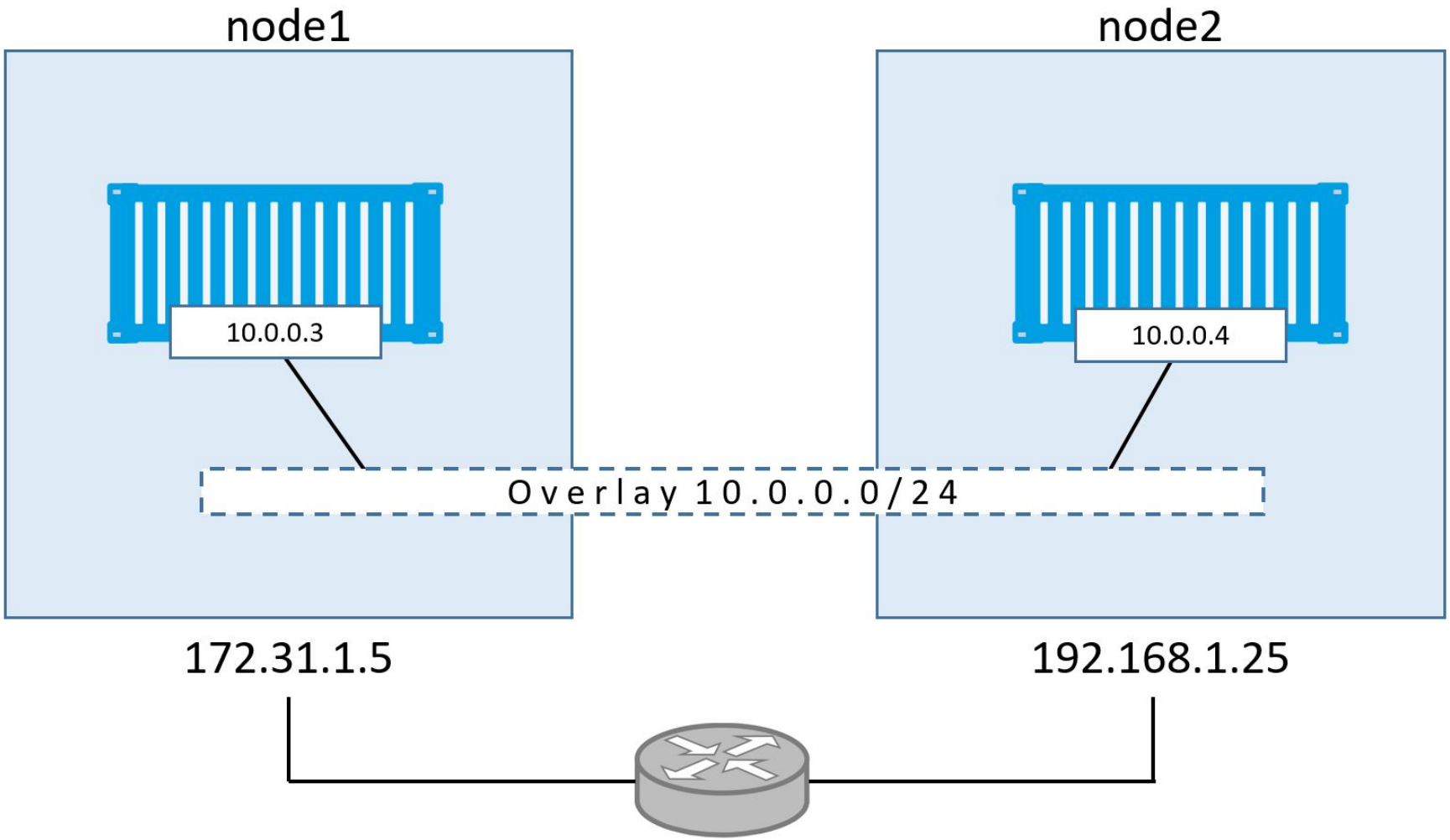
Docker Compose

- A way of defining and managing multiple containers and their resources
- Use YAML templates to define multiple services including networks, volumes
- Start and develop applications using an iterative lifecycle
- Namespaced resources provides services stack isolation
- Single host but can be used with Swarm
- Installed with Mac / Windows, separately on Linux

Into Production!

Overlay Networks

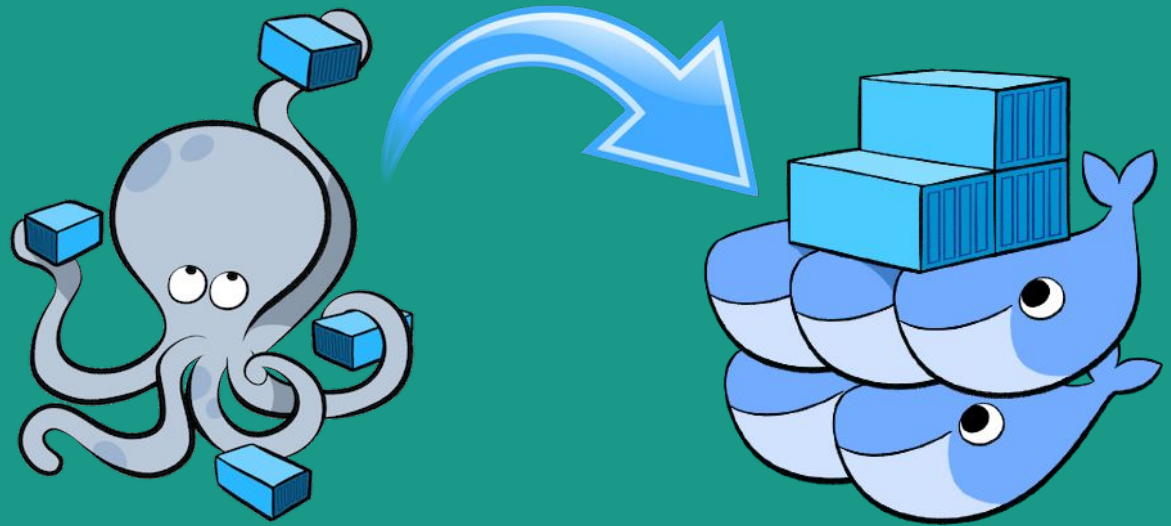
- A distributed network between hosts
- Requires swarm mode to be enabled for participant nodes
- VXLANs to create a virtual Layer 2 network on top of an existing Layer net
- Can be encrypted via IPSEC tunnels via driver config
- Creates and ***ingres*** (default) and ***docker_gwbridge*** networks


node1
node2
10.0.0.3
10.0.0.4
Overlay 10.0.0.0/24
172.31.1.5
192.168.1.25

Tips!

- Check for IP range conflicts in ***ingres*** and ***docker_gwbridge***
- Firewall configured:
  - TCP port 2377 for cluster management communications
  - TCP and UDP port 7946 for communication among nodes
  - UDP port 4789 for overlay network traffic
- Always test performance of your network!
  - IPSec / Kernel issues!
- Swarm initialized? :)

Docker Swarm



https://github.com/mattjtodd

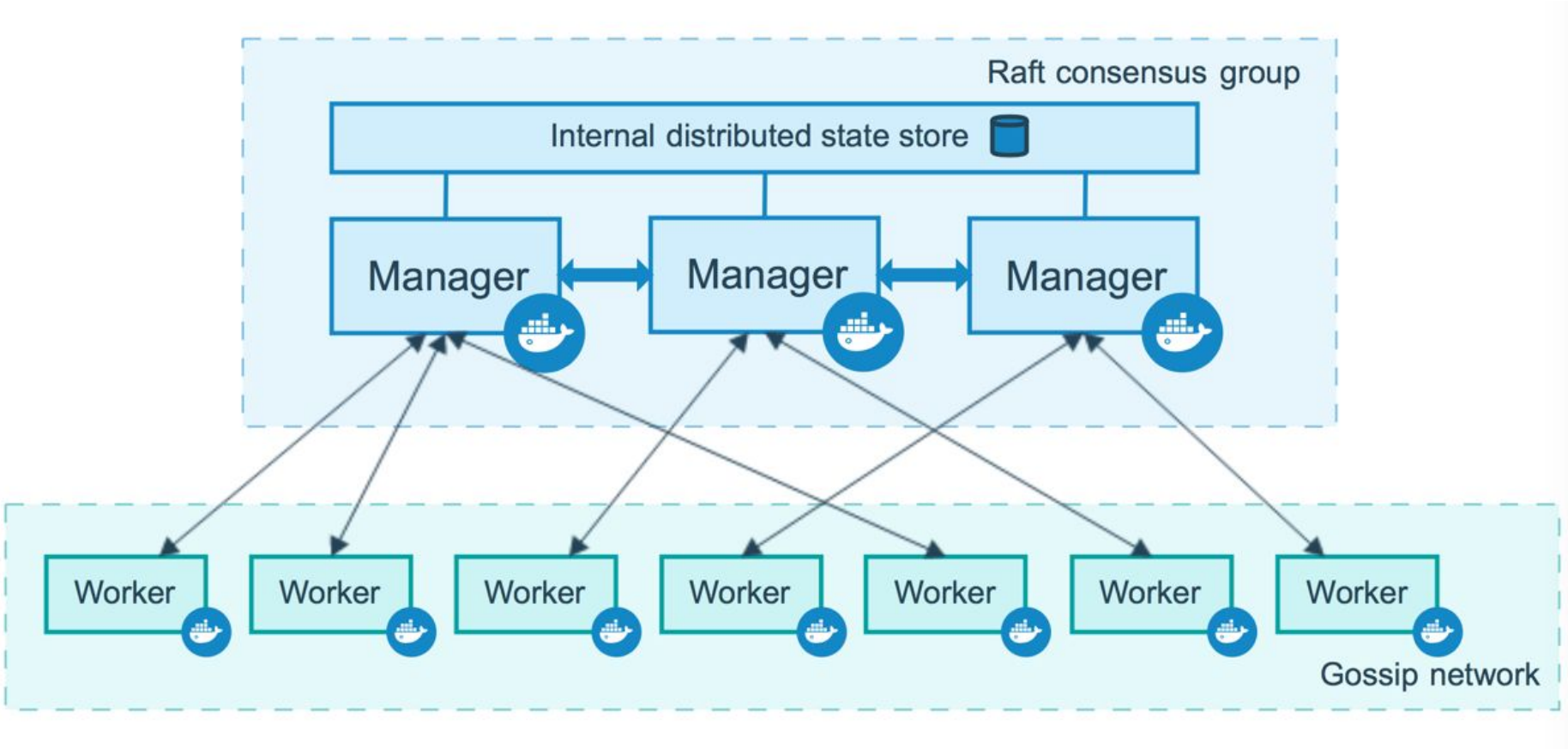
docker-from-the-ground-up-part-3

Desirable Characteristics

- Resilient to failure
- Secure in the cloud
- Horizontally scalable
- Easily managed
- Operator Friendly
- Heterogeneous resource capability
- Service Discovery

Features

- Built into Engine; batteries included!
- Orchestration via cli driven services / compose stacks
- Multi-host service mesh routing / load balancing
- DNS based Service discovery
- Config / Secrets management
- Rolling update / rollback / spread policies


Raft consensus group
Internal distributed state store
Manager
Manager
Manager
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Gossip network

Swarm

- A cluster will be at least one node (preferably more)
- A node can be a manager or a worker
- A manager actively takes part in the Raft consensus, and keeps the Raft log
- One manager is elected as the leader; other managers merely forward requests to it
- The workers get their instructions from the managers
- Both workers and managers can run containers

Initialisation

- Initialize a seed manager node
- Join other nodes with the join tokens
- Specify interface ip if needed
- Vagrantfile in **vagrant/docker**

Initialisation

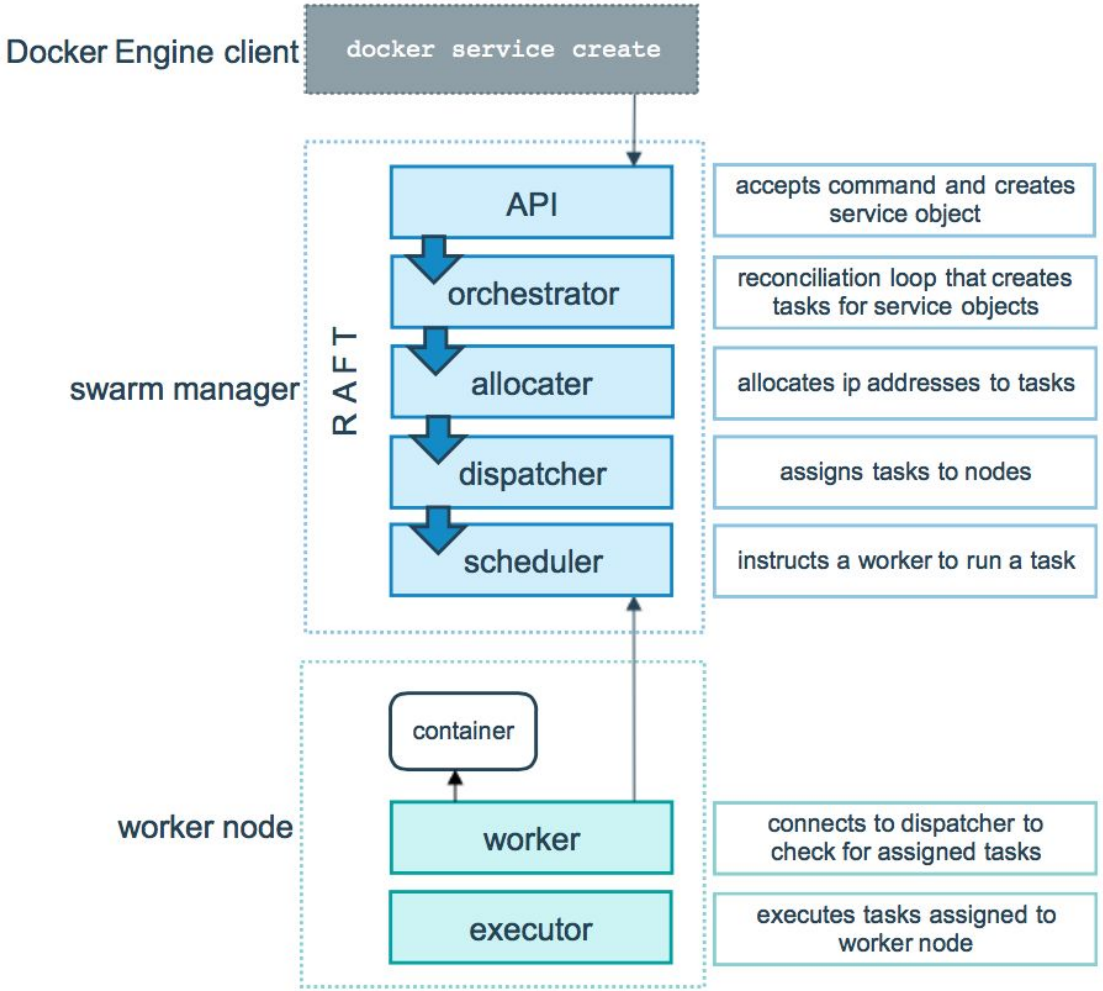
```
$ vagrant up
$ vagrant ssh node-1
$ docker swarm init --advertise-addr=192.168.50.150
$ docker node ls
Copy the join token command and exit the ssh session
$ vagrant ssh node-2
$ <paste token join command in here>
$ docker node ls # we're a worker, so can't see this
$ exit
$ vagrant ssh node-2
$ docker node promote
```

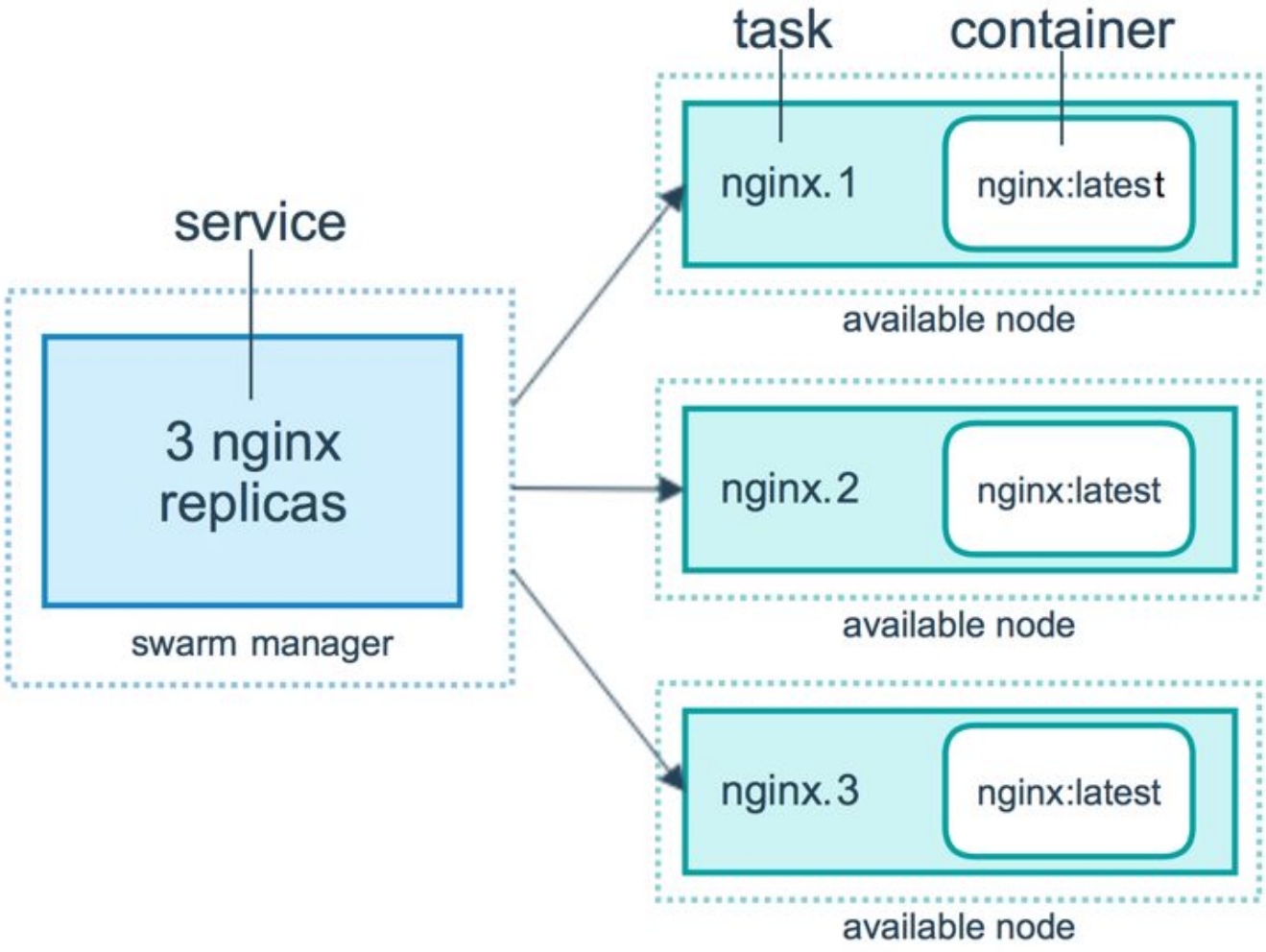
Orchestration

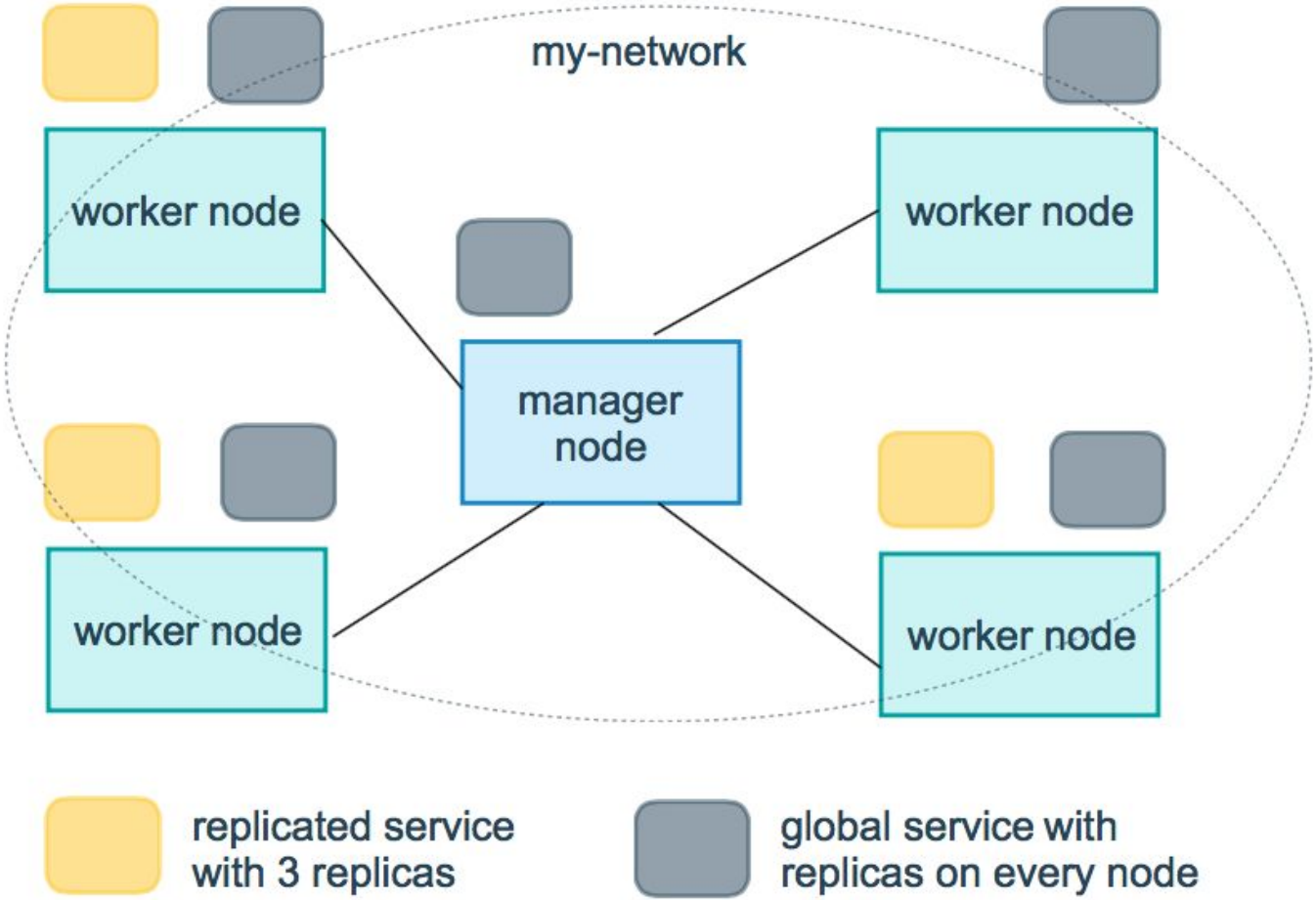
- Manager(s) job:
  - **Schedule and maintain containers on a cluster of workers**
- Worker(s) job:
  - **Execute containers and report back on the status**
- Services define the tasks to be run
  - Image, port mappings replicas, ...
- Tasks are the encapsulation for a container
  - Assigned to a node and lives there until failure

Services & Tasks

- Services define the *shape* and *characteristics* of tasks
- Health-checks define when services should be registered in the task pool
- Placement restrictions
- Resource constraints
- Rolling update policies
- Rollback policies
- Modes ***replicated*** and ***global***


Docker Engine client
docker service create
swarm manager
RAFT
API
orchestrator
allocater
dispatcher
scheduler
accepts command and creates service object
reconciliation loop that creates tasks for service objects
allocates ip addresses to tasks
assigns tasks to nodes
instructs a worker to run a task
worker node
container
worker
executor
connects to dispatcher to check for assigned tasks
executes tasks assigned to worker node


task
container
service
nginx.1
nginx:latest
available node
3 nginx
replicas
nginx.2
nginx:latest
available node
swarm manager
nginx.3
nginx:latest
available node


my-network
worker node
worker node
manager node
worker node
worker node
replicated service with 3 replicas
global service with replicas on every node

Service Discovery

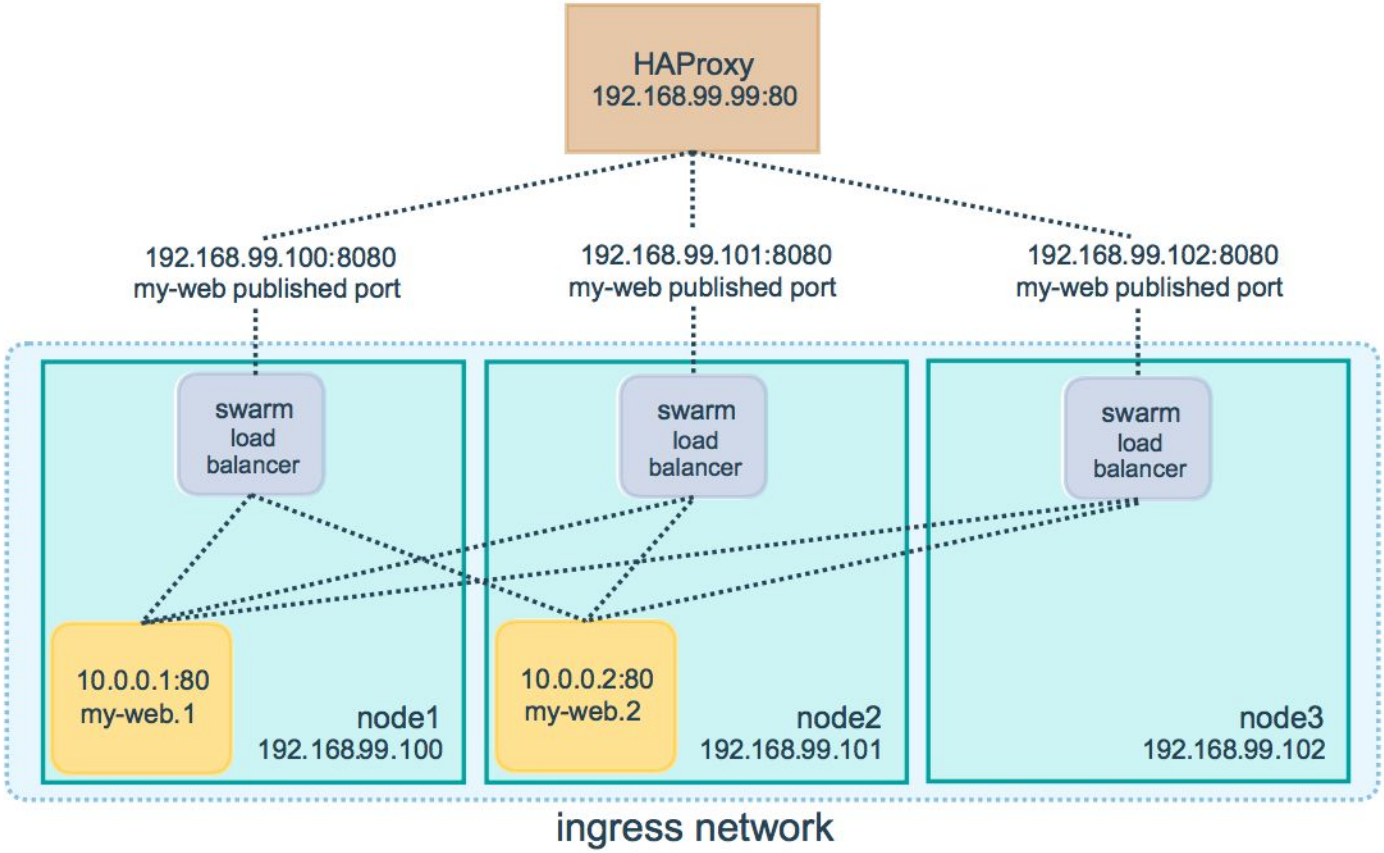
- As with compose Docker uses DNS to register services in a pool
- Only ***healthy*** tasks are registered in the pool
- User Defined bridge networks register named containers here
- Overlay has Two types:
    - VIP (default)
    - DNSRR

Service Discovery

- VIP
  - Resolves to a single VIP which load balances between tasks
  - Can be part of the ingres service mesh
  - Pool DNS lookup takes the form of `tasks.<service-name>`
- DNSRR
  - Cannot be part of the service mesh
  - Round-robin ip of containers
  - Self configured LB based on DNS A query
  - Pool DNS lookup by `<service-name>` directly

Load Balancing / Service Mesh

- Containers can live on any suitable host
- Physical / Virtual LB balances load between nodes
- Ingres ports are mapped to each host
- Traffic routed to service tasks in the pool
- Load balance you nodes!



Routing Mesh

Healthcheck

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost"]
  interval: 1m30s
  timeout: 10s
  retries: 3
  start_period: 40s
```

Deploy Policies

- Restart Policy
  - If and how to restart on exit
- Rollback Config
  - What to do in event of service update failure
- Update Config
  - How a service should be updated
  - E.g. stop /start first, delay etc.

Secrets & Configs

- Persisted by the managers
- Allocated to containers on as needed basis
- Prevents proliferation of sensitive material
- Keep you containers vanilla!
- Mounted in the containers files system
- Only secrets encrypted in managers

Logging Drivers

- Log aggregation remote / local
- Multiple protocols
  - json-file (default)
  - awslogs
  - fluentd
  - gcplogs
  - gelf
  - Splunk

Placements & Preferences

```
deploy:
   placement:
     constraints:
       - node.role == manager
       - engine.labels.operatingsystem == ubuntu 18.04
     preferences:
       - spread: node.labels.az
```

Stackfiles

- Docker compose files version 3+
- Manage service definitions much like compose
- Deploy directive provides service and task properties
- https://docs.docker.com/compose/compose-file/

Sample Stackfile

```
version: '3'

services:

  portainer:
    image: portainer/portainer
    command: -H "tcp://tasks.portainer-agent:9001" --tlsskipverify --no-auth
    ports:
      - "9000:9000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints: [node.role == worker]
```

Production Tips

- Use logging aggregation
- Monitor *all* hosts
- Label and tag everything using a consistent schema
- Keep the managers free from work (only control plane agents)

Demo Lab

Goals

- Stand up a 1 manager 3 worker swarm cluster
- Use Portainer to visualise state
- Deploy EFK stack
- Observe Grafana
- Deploy a small application
- Run some test loads
- Scale in / out

Tools

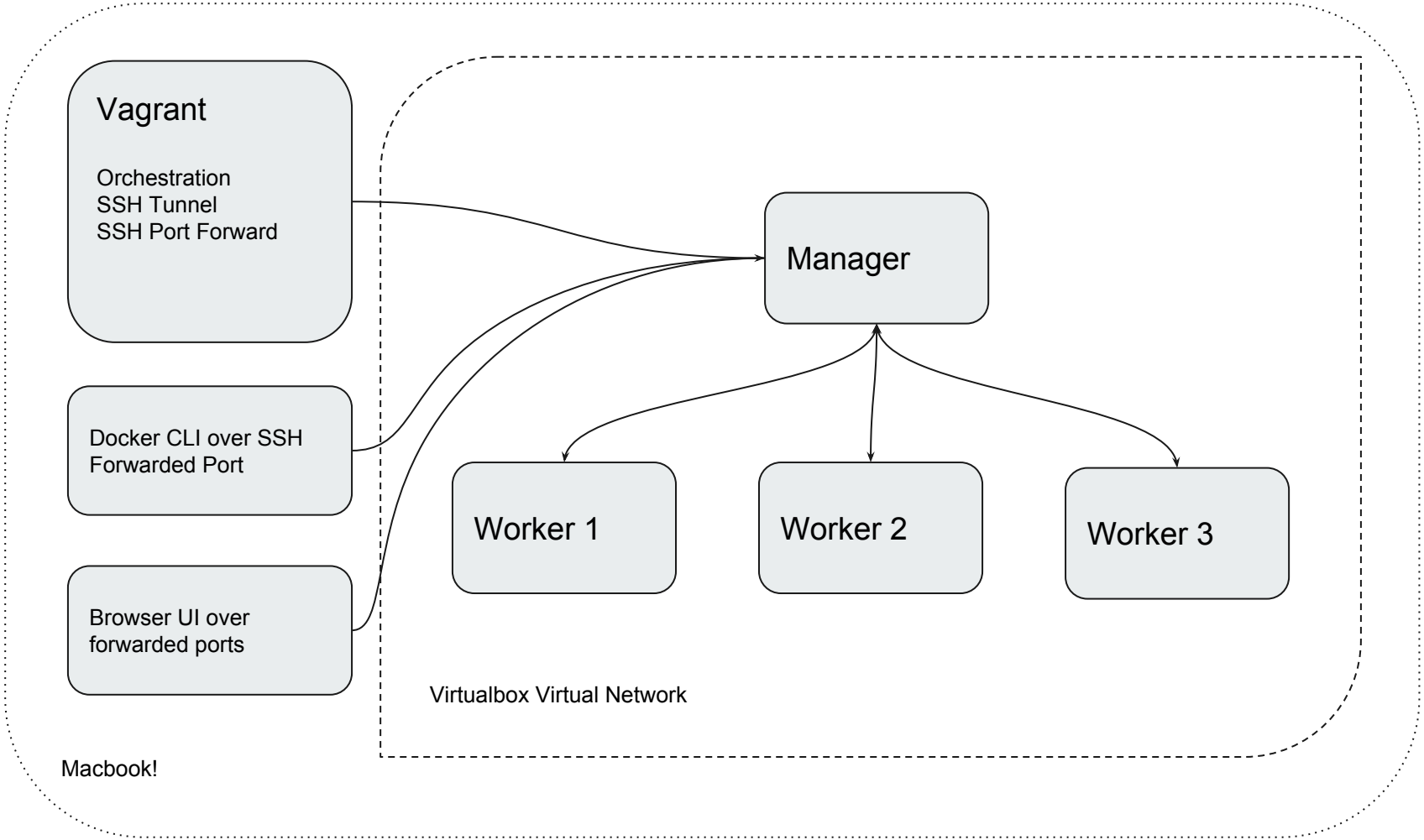
- Vagrant
- Ubuntu Bionic Beaver
- Docker Swarm Mode
- Portainer
- SSH

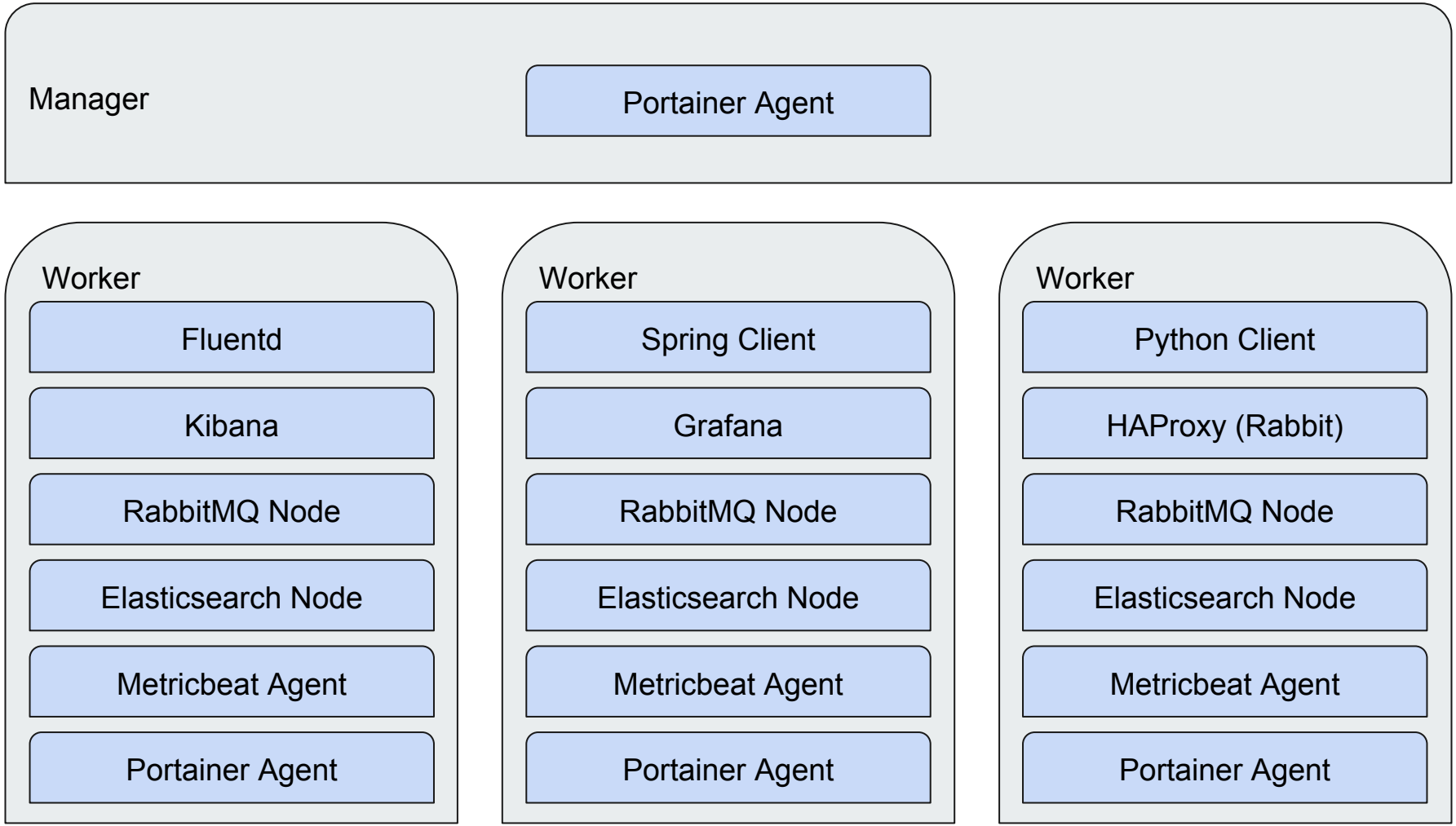
DOCKER SWARM
Manager
192.168.10.1

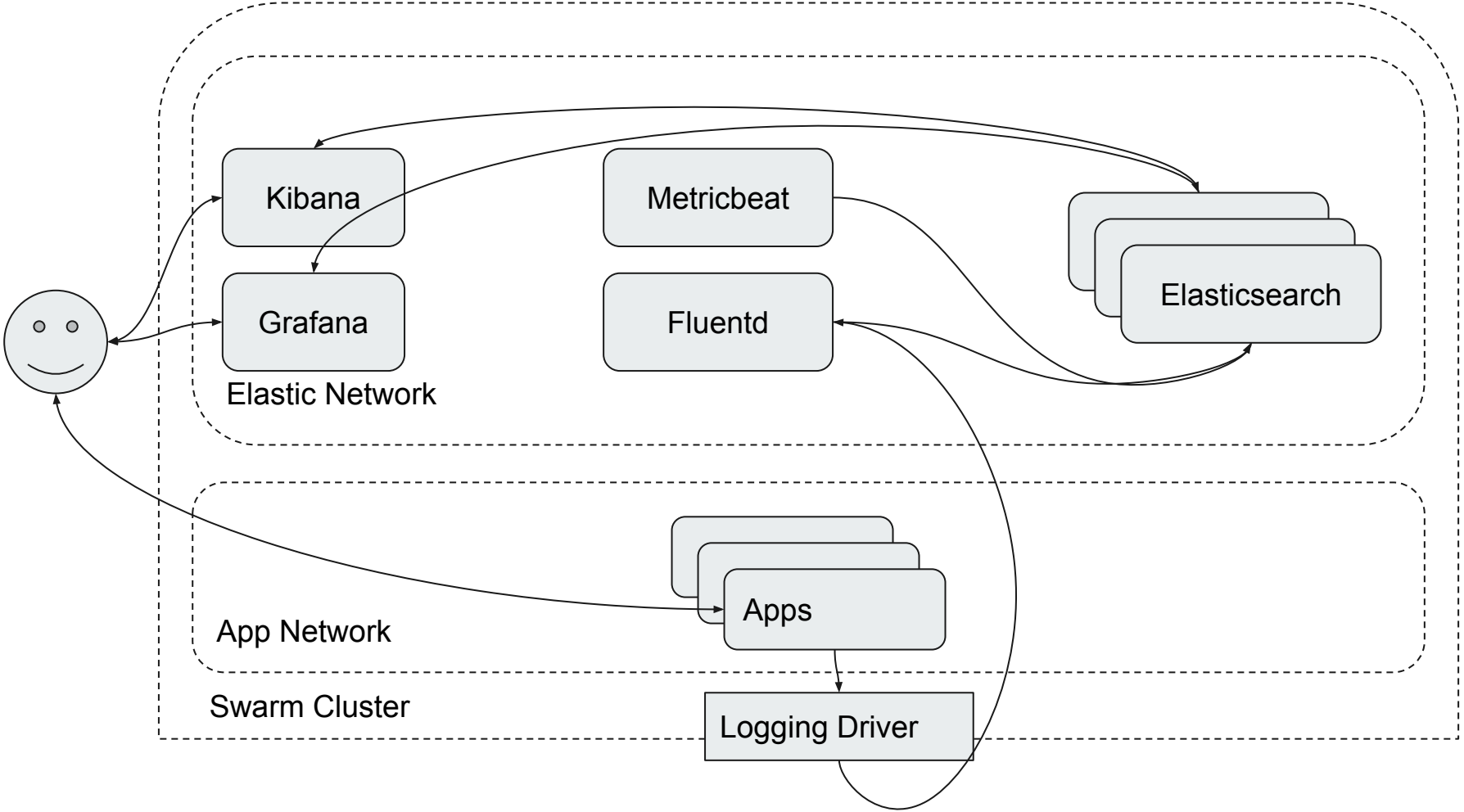
Worker 1
192.168.10.2

Worker 2
192.168.10.3

Worker 3
192.168.10.4


Vagrant
Orchestration
SSH Tunnel
SSH Port Forward
Docker CLI over SSH
Forwarded Port
Browser UI over
forwarded ports
Manager
Worker 1
Worker 2
Worker 3
Virtualbox Virtual Network
Macbook!


Manager
Portainer Agent
Worker
Fluentd
Kibana
RabbitMQ Node
Elasticsearch Node
Metricbeat Agent
Portainer Agent
Worker
Spring Client
Grafana
RabbitMQ Node
Elasticsearch Node
Metricbeat Agent
Portainer Agent
Worker
Python Client
HAProxy (Rabbit)
RabbitMQ Node
Elasticsearch Node
Metricbeat Agent
Portainer Agent


Kibana
Metricbeat
Elasticsearch
Grafana
Fluentd
Elastic Network
Apps
App Network
Swarm Cluster
Logging Driver

SSH-Fu

- Docker 18.09.0 supports connection over SSH tunnels
  - https://github.com/docker/cli/pull/1014
- SSH Port-forwarding will achieve the same
- SSHuttle may also work and resolve hostnames

Getting Started

```
vagrant ssh -- \
  -L9999:/var/run/docker.sock \
  -L9000:localhost:9000 \
  -L5601:localhost:5601 \
  -L3000:localhost:3000 \
  -L15672:localhost:15672 \
  -L5000:localhost:5000 \
  -L8080:localhost:8080
```

Getting Started

```
# Start up the vagrant managed VMS.  This will create the cluster
$ vagrant up

# Create the SSH Port Forwards for access to the published services
$ vagrant ssh -- \
  -L9999:/var/run/docker.sock \
  -L9000:localhost:9000 \
  -L5601:localhost:5601 \
  -L3000:localhost:3000 \
  -L15672:localhost:15672 \
  -L5000:localhost:5000 \
  -L8080:localhost:8080
```

Deploy EFK

Configure the host ENV variable for the docker client:

```
$ export DOCKER_HOST=localhost:9999
```

Listing the processes via the CLI should yield:

```
$ docker ps
CONTAINER ID        IMAGE                     COMMAND                  CREATED             STATUS
PORTS               NAMES
efe073030701        portainer/agent:latest    "./agent"                10 minutes ago      Up 10
minutes
portainer_portainer-agent.3ojifkjtru85qe3kcx2zzday7.icmyg91ylw5rfgfze9wztwnn0
983231ecb5ce        registry:2                "/entrypoint.sh /etc…"   10 minutes ago      Up 10
minutes       5000/tcp            registry-mirror_registry-mirror.1.lkfo7pljrrazsyvkx84qa4s4p
```

Deploy EFK

Move into the **elastic-cluster** dir and run:

```
$ docker stack deploy -c docker-compose.yml efk
```

Deploy EFK

```
$ docker service ls
ID                  NAME                                MODE                REPLICAS
IMAGE                                PORTS
sfwiht8446ch        efk_elasticsearch                   global              0/3
elastic/elasticsearch:6.4.3
d4bgkdj8knz5        efk_fluentd                         replicated          0/1
dockerbirmingham/fluentd:latest   *:24224->24224/tcp
es01tx3x4rjv        efk_grafana                         replicated          0/1
grafana/grafana:latest            *:3000->3000/tcp
r9w8skclnfor        efk_kibana                          replicated          0/1
elastic/kibana:6.4.3              *:5601->5601/tcp
m1hv7iralzbm        efk_metricbeat-host                 global              0/4
elastic/metricbeat:6.4.3
tikymzswhywa        portainer_portainer                 replicated          1/1
portainer/portainer:latest        *:9000->9000/tcp
f5215am43p98        portainer_portainer-agent           global              4/4
portainer/agent:latest
vpqnfcch3d95        registry-mirror_registry-mirror     replicated          1/1
registry:2                        *:5555->5000/tcp
```

Deploy EFK

- Portainer UI
  - http://localhost:9000
- Kibana:
  - http://localhost:5601
- Grafana:
  - http://localhost:3000

Deploy App

Back up to project root and into the **rabbitmq** dir.

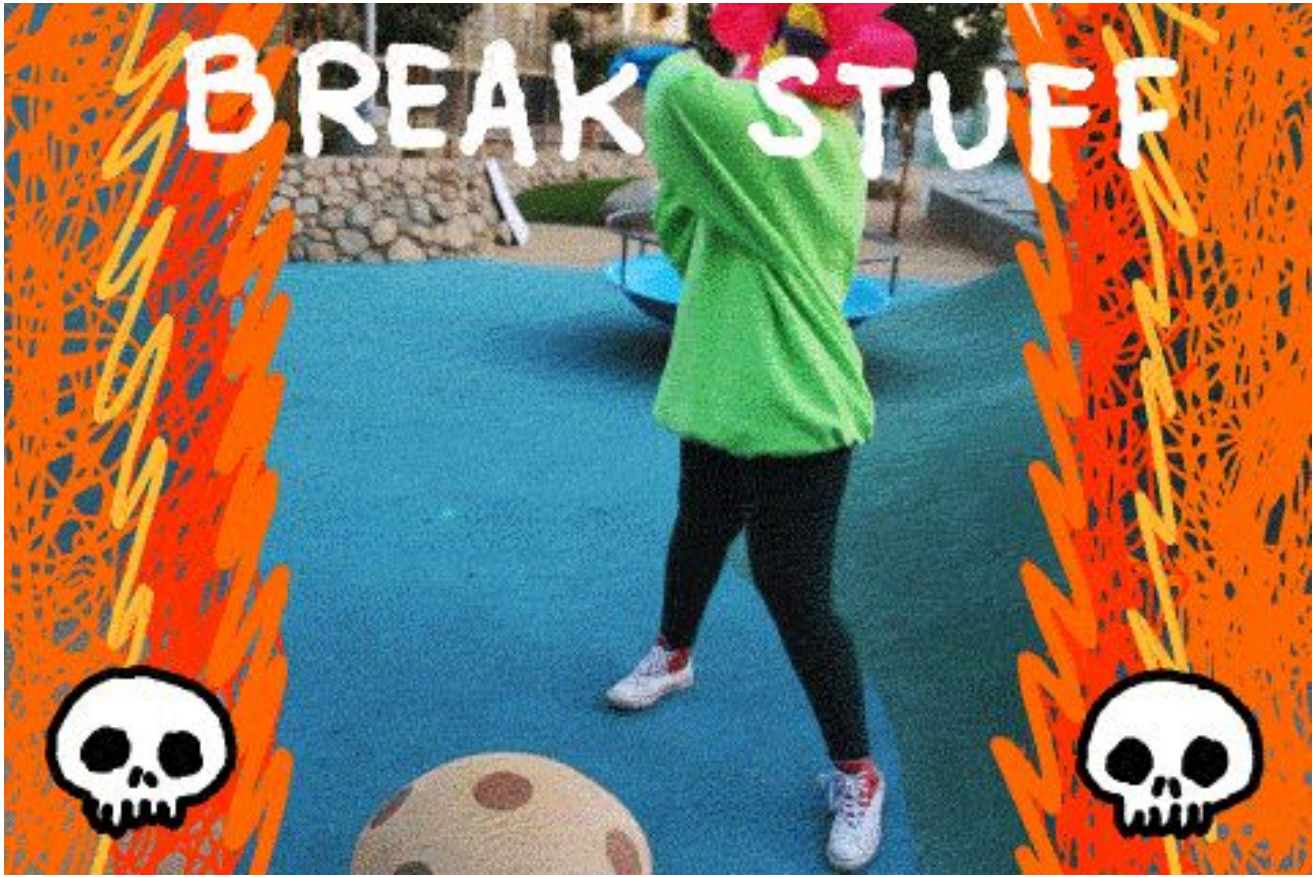
```
$ docker stack deploy -c docker-compose.yml app

$ docker service ls

...
```

Access App

- RabbitMQ Manager
  - Localhost:15672
    - monitor
    - Password
- Python HTTP client
  - http://localhost:5000/greeting?message=Hello
- Spring Boot HTTP Client:
  - http://localhost:8080/greeting?message=Hello


BREAK STUFF

Additional Resources

https://www.katacoda.com/courses/docker-orchestration

https://container.training/swarm-selfpaced.yml.html

https://training.play-with-docker.com/

https://docs.docker.com/engine/swarm/