Docker From the Ground Up Part 2

Networking and Compose

Thanks to our Sponsor!

BlackCat /

Who's This Guy?

Matt Todd

Principal Archiect @ Resonate.tech

Cloud Native Advocate

`github.com/mattjtodd`

`hub.docker.com/u/mattjtodd`

Recap!

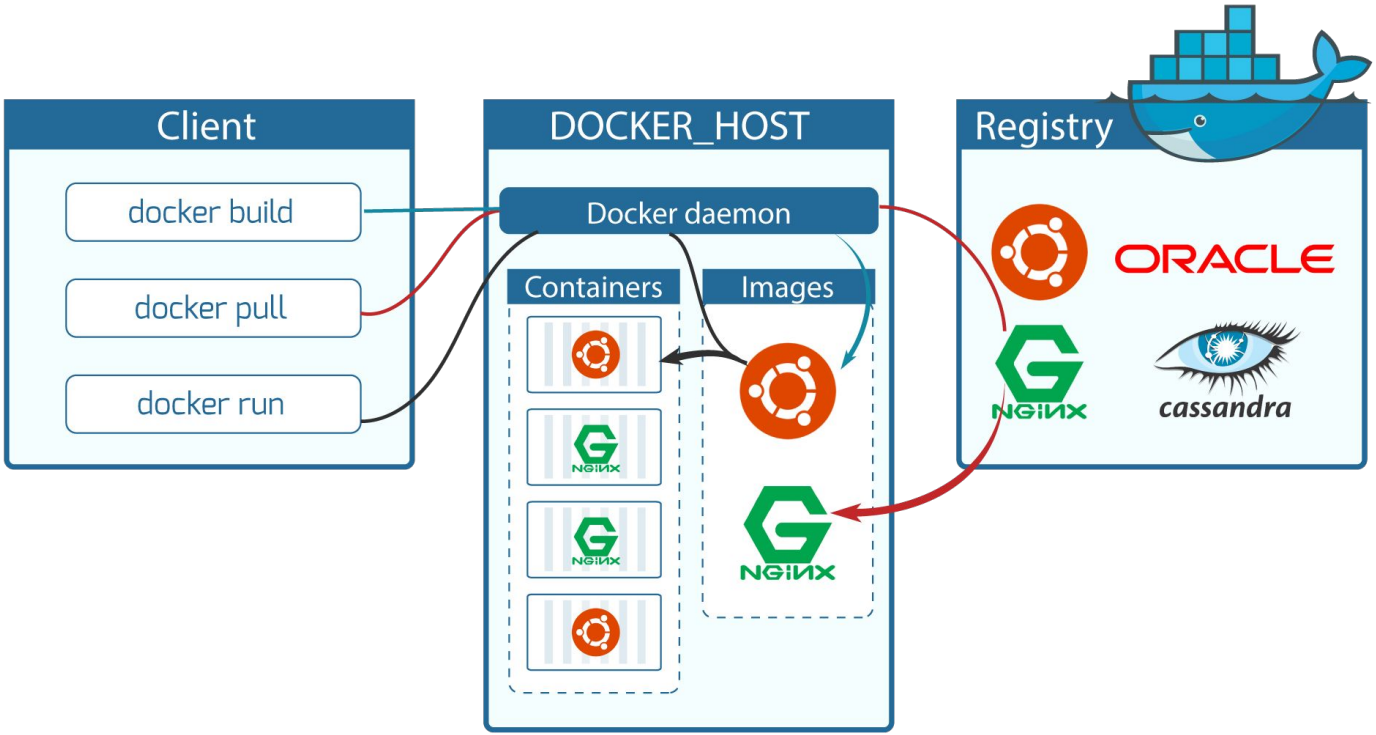
Containers

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

Docker / Containerisation

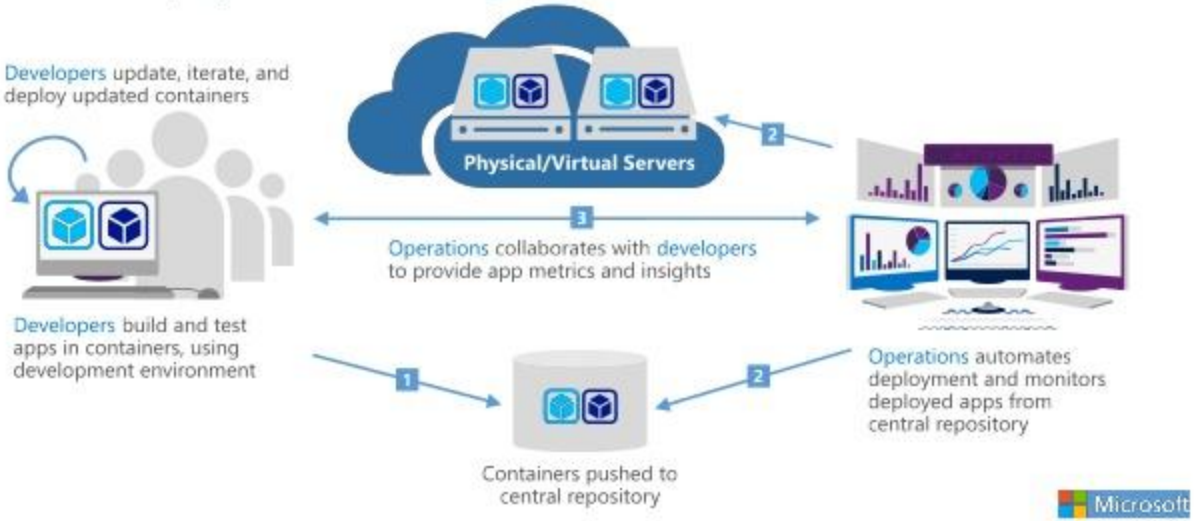
- Runtime Standardisation
- Scalability
- Dependency Management & Versioning
- Lightweight & Resource Isolated
- Shared FS Layering
- Repeatability of Builds
- Portability
- Cloud Native (CNCF)
- Immutability (Cattle Not Pets)
- Rich Sharing Ecosystem

DOCKER COMPONENTS



DevOps and Containers

Creation, deployment, and management



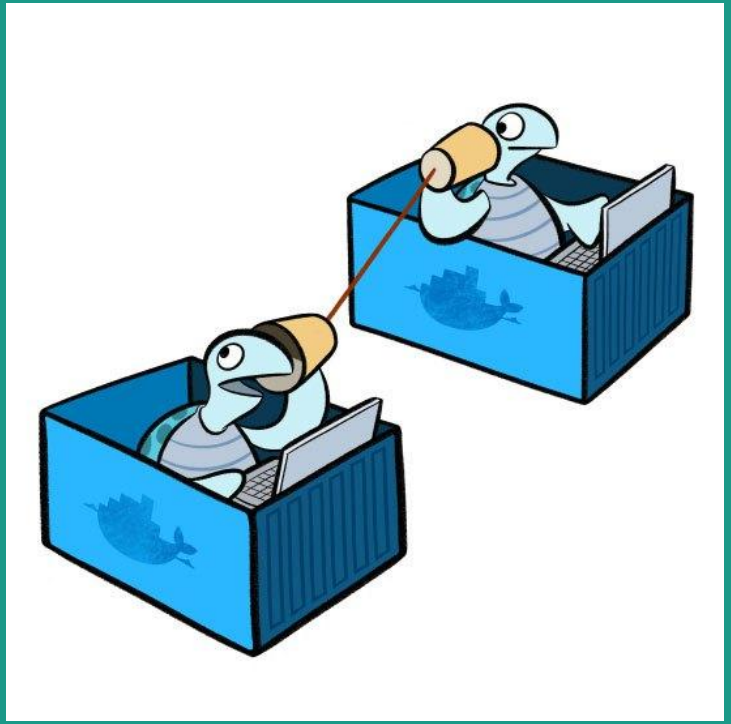

WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net

Poll: Dev / Ops spread?

Resources

- https://github.com/mattjtodd/docker-from-the-ground-up-part-2
- Meetup / Slack channel images

Docker Networking



Networking

- Namespaced networking stack per container
- Isolated by design
- Extensive use of bridge networking (Virtual switches)
- Multi-host capability
- Portable and able to work in many networking stacks
- Pluggable drivers

Network Drivers

- Bridge
- Host
- Overlay
- Macvlan
- None
- Third party plugins (Weave, Contiv, ...)

Networking CLI runthrough

- Consistent naming with the other management commands

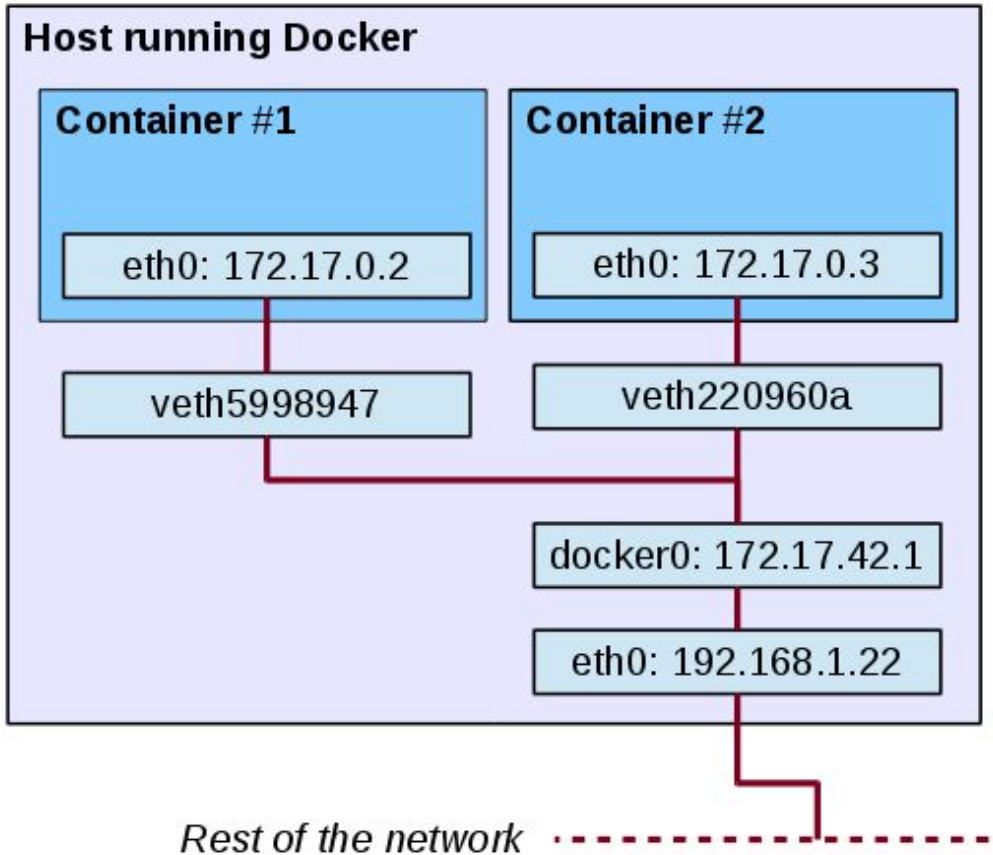
```
connect     Connect a container to a network
create      Create a network
disconnect  Disconnect a container from a network
inspect     Display detailed information on one or more networks
ls          List networks
prune       Remove all unused networks
rm          Remove one or more networks
```

CLI Runthrough

- `$ docker network ls`
- `$ docker network inspect none host bridge`
- `$ docker network create testnet`
- `$ docker network inspect testnet`
- `$ docker network rm testnet`

Bridge

- Link Layer device which forwards traffic between network segments
- Allow containers connected to the same bridge network to communicate
- On startup Docker creates a default bridge network (*bridge*)
- Networks IP address ranges are defined using CIDR notation
- See https://tools.ietf.org/html/rfc1918


Host running Docker
Container #1
Container #2
eth0: 172.17.0.2
eth0: 172.17.0.3
veth5998947
veth220960a
docker0: 172.17.42.1
eth0: 192.168.1.22
Rest of the network

Default Bridge

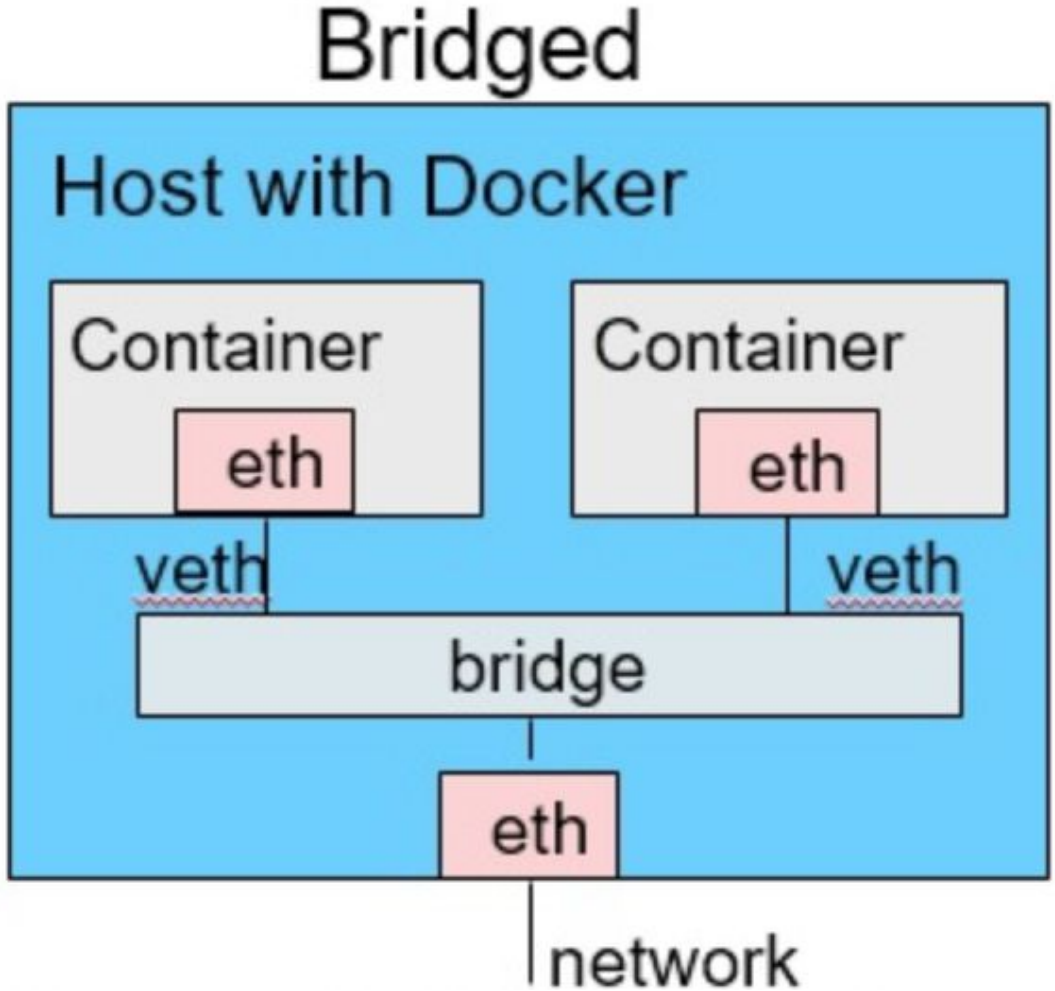
- By default containers are launched into this network
- Per-host global configuration
- Can be changed on startup of engine only
- Limited segmentation possibilities
- Communication via IP or legacy --link option

Default Bridge Example

```
# Create two containers and background them
$ docker run --name one -d -it alpine sh
$ docker run --name two -d -it alpine sh
# Attempt ping ont to two via name
$ docker exec -it one ping two
# Inspect the bridge network and get two's IP
$ docker network inspect bridge
# Ping two from one
$ docker exec -it one ping <ip-from-network-for-two>
$ docker rm -f one two
```

User Defined Bridge

- Provides a DNS service by container name
- Attach / re-attach networks to containers
- Micro-segmentation of networking architecture
- Separate definition of bridge networks config
- Some similarity to AWS VPC subnets


Bridged
Host with Docker
Container
eth
Container
eth
veth
veth
bridge
eth
network

User Defined Bridge Example

```
$ docker network create -d bridge testnet
$ docker network inspect testnet
$ docker run --name one --network testnet -d -it alpine sh
$ docker run --name two -d -it alpine sh
$ docker exec -it one ping two
$ docker network connect testnet two
$ docker exec -it one ping two
$ docker rm -f one two
$ docker network rm testnet
```

Publishing Ports

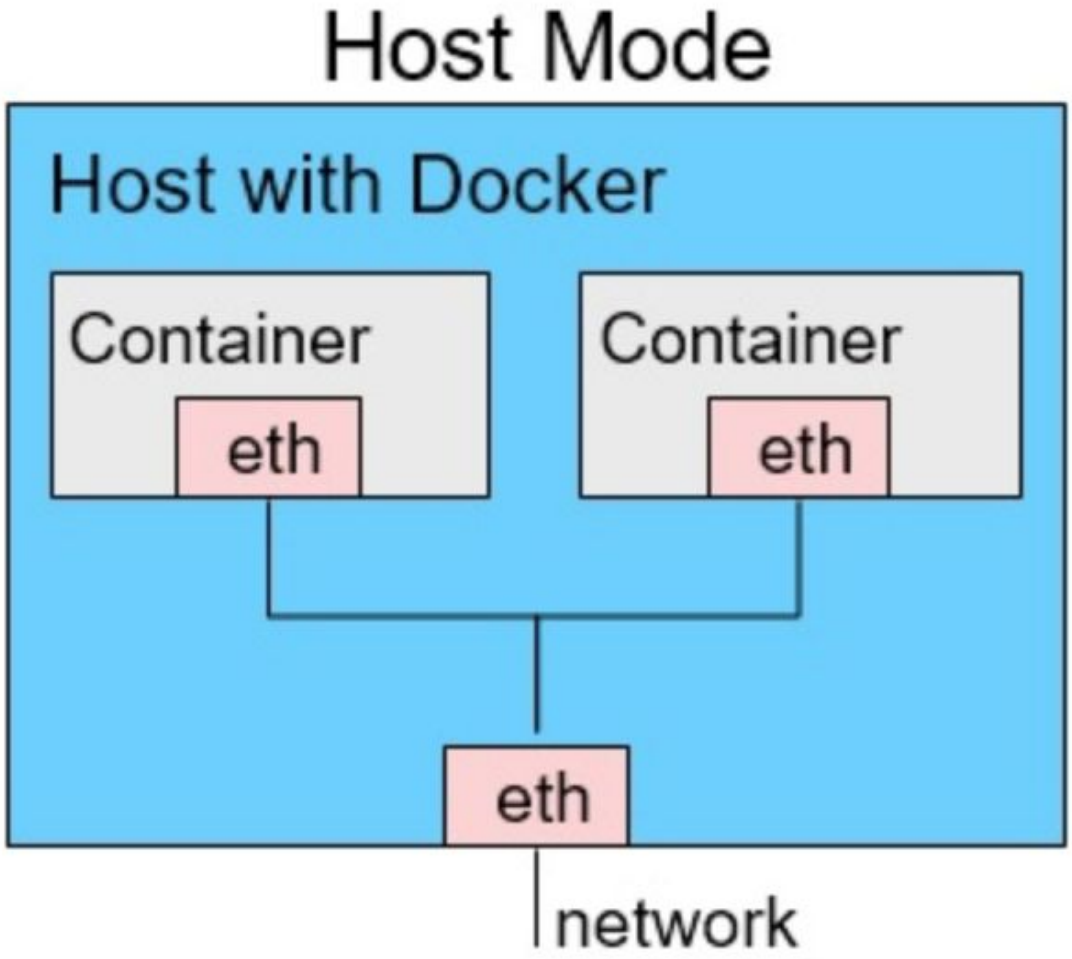
- Container ports can be mapped to physical ports on the hosts
- They can be allocated specifically or dynamically
- Physical constraints apply, once a port is allocated, it’s gone!
- Can publish single, multiple and a range of ports
- Multiple protocols udp, tcp
- Similar to a simple Firewall / Reverse Proxy

Public container port mapping

```
$ docker run --rm -it -p 127.0.0.1:8888:80 nginx:alpine

$ curl localhost:8888

<!DOCTYPE html>
...

$ docker run --rm -it -p 127.0.0.1::80 nginx:alpine

$ curl localhost:?
```

Host

- Containers networking stack is not isolated from the hosts
- Ports published on from the container are available directly on the host
- Docker for Win / Mac will bind to the *hypervisor* networking
- User *docker-machine* to easily exercise in isolation


Host Mode
Host with Docker
Container
eth
Container
eth
eth
network

Container Mode

- Share / Inherit the networking of another container
- All containers have the Same IP address
- Used by Kubernetes PODS
- Only ports exposed as part of the first container network can be made available
- Can’t publish new ports via the “bound” container

Container mode example

```
#
$ docker network create testnet
$ docker run --rm --name server -it --network testnet dockerbirmingham/iperf3 -s
$ docker run --rm --name client -it --network container:server
dockerbirmingham/iperf3 -c localhost -i 1 -t 30
$ docker stop server client
$ docker network rm testnet
```

Overlay

- Allows host to host container networking
- Requires docker swarm mode
- Provides IPSEC encryption and routing mesh
- Recc. max 255 ips addresses per network
- Will cover next time!

Macvlan

- Provides ability to connect directly to the physical network
- Caution, needs close config attention!
- Could cover in specific session, if there is interest

Overriding defaults

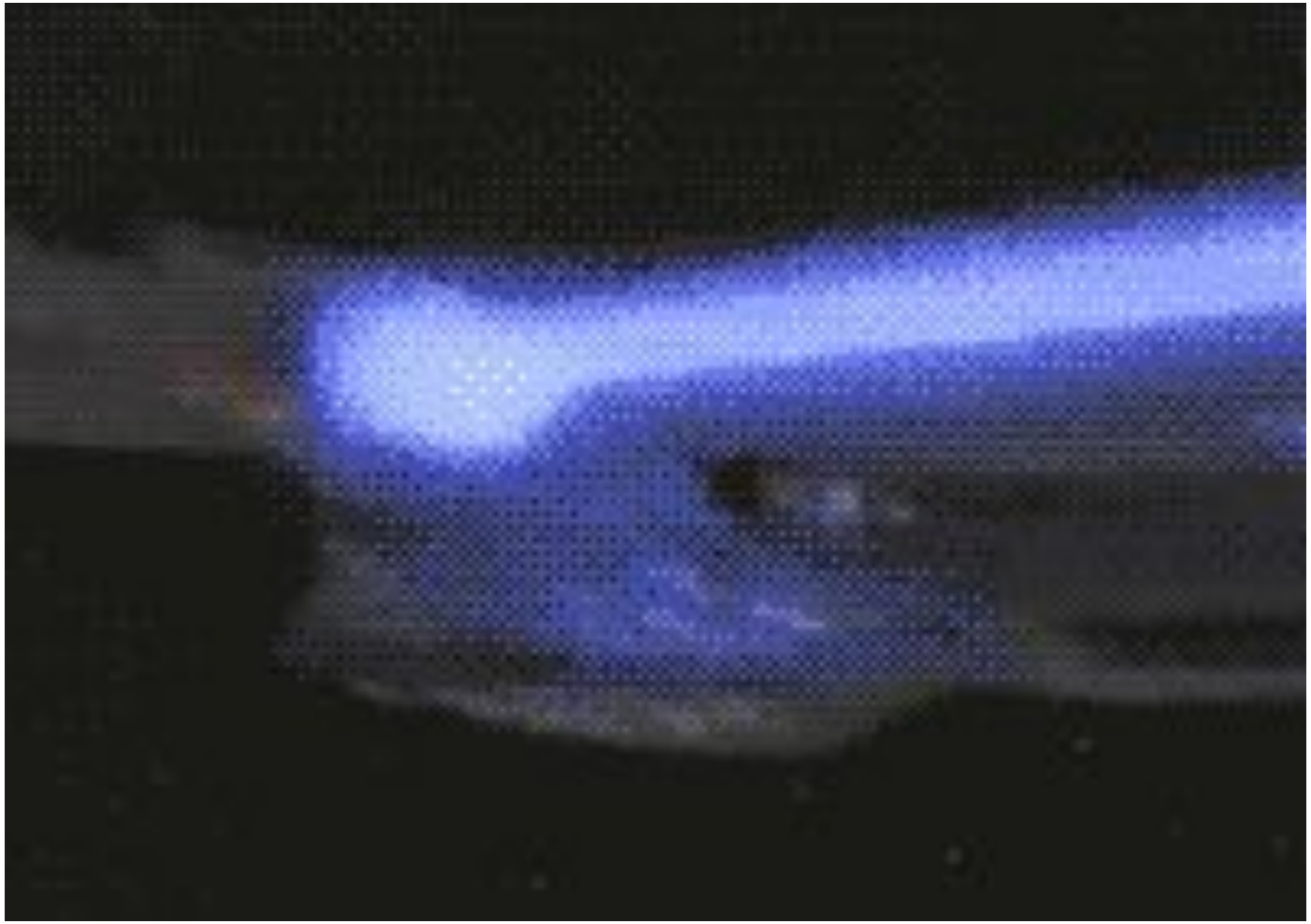
- IP address range conflicts can cause trouble (private network conflicts)
- All are configurable using the CLI tools.
- Understand CIDR notation and Private IP address ranges! (RFC1918)
- https://tools.ietf.org/html/rfc1918

Netshoot

- https://github.com/nicolaka/netshoot
- Many networking diagnostics / utilities
- Adapt and push for yourselves!

More Info

- https://success.docker.com/article/networking



Docker Compose

Overview

- A way of defining and managing multiple containers and their resources
- Use YAML templates 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

Use Cases

- Development environments
- CI / CD environments
- Proof of concept work
- Networking POCS
- Distributed Computing POCS
- VCS Infrastructure as Code

Compose CLI

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Composify Iperf3

Create a simple stack to run a client and server Iperf3 stack:

- Small Base image
- Tests container to container networking as before
- Uses Compose
- Committed to version control
- Shared on Docker Hub

Iperf3 Dockerfile

```
FROM alpine:3.8

RUN apk add --no-cache iperf3

ENTRYPOINT ["iperf3"]
```

Iperf3 Compose File

```
version: "3"

services:
 server:
   image: dockerbirmingham/iperf3-alpine
   build: ./
   command: -s

 client:
   image: dockerbirmingham/iperf3-alpine
   build: ./
   command:  -c server -i 1 -t 30
```

Build, Run, Ship!

```
# stand up the stack and detach tty from output
$ docker-compose up -d
# attach the tty to the log output from all containers and tail
$ docker-compose logs -f
# stop all containers defined the in stackfile
$ docker-compose down
# publish images to public repo
$ docker-compose push
```

Telemetry!

- Time Series Databases
  - Prometheus
  - Elastic
  - InfluxDb
- Grafana
- Cadvisor
- Netdata
- Many more!

Netdata

https://my-netdata.io/

- Compose stack, let’s deploy it!
- Let’s look at this in more detail……..

Netdata Stackfile

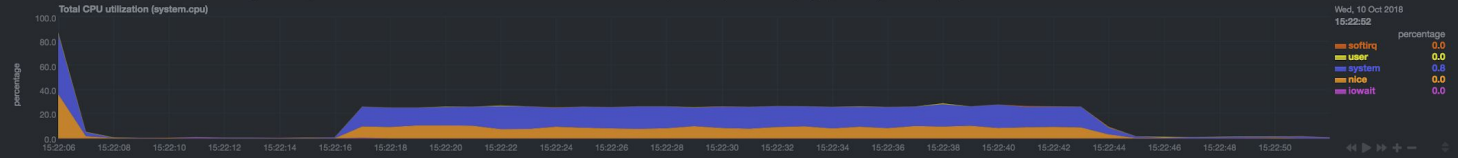
```
version: '3'
services:
  netdata:
    image: netdata/netdata
    hostname: example.com # set to fqdn of host
    ports:
      - 19999:19999
    cap_add:
      - SYS_PTRACE
    security_opt:
      - apparmor:unconfined
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
```

System Overview

Overview of the key system metrics.

cpu

Total CPU utilization (all cores). 100% here means there is no CPU idle time at all. You can get per core usage at the **CPUs** section and per application usage at the **Applications Monitoring** section.
Keep an eye on **iowait** (0.00%). If it is constantly high, your disks are a bottleneck and they slow your system down.
An important metric worth monitoring, is **softirq** (0.00%). A constantly high percentage of softirq may indicate network driver issues.

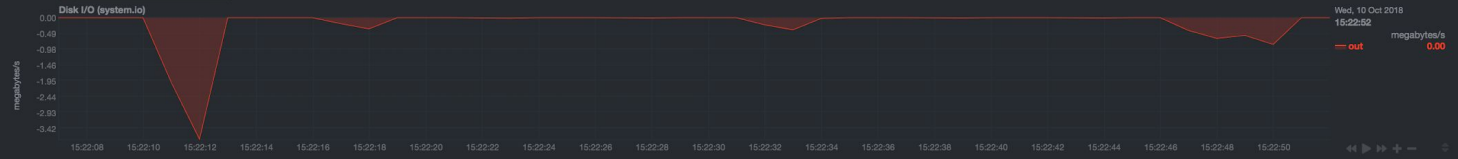


load

Current system load, i.e. the number of processes using CPU or waiting for system resources (usually CPU and disk). The 3 metrics refer to 1, 5 and 15 minute averages. The system calculates this once every 5 seconds. For more information check **this wikipedia article**

disk

Total Disk I/O, for all physical disks. You can get detailed information about each disk at the **Disks** section and per application Disk usage at the **Applications Monitoring** section. Physical are all the disks that are listed in `/sys/block`, but do not exist in `/sys/devices/virtual/block`.



Memory paged from/to disk. This is usually the total disk I/O of the system.

System Overview
cpu
load
disk
ram
swap
network
processes
idlejitter
interrupts
softirqs
softnet
entropy
ipc semaphores
uptime

CPUs
Memory
Disks
IP Virtual Server
Networking Stack
IPv4 Networking
IPv6 Networking
Network Interfaces
Firewall (netfilter)
Applications
User Groups
Users
Example Charts
Netdata Monitoring

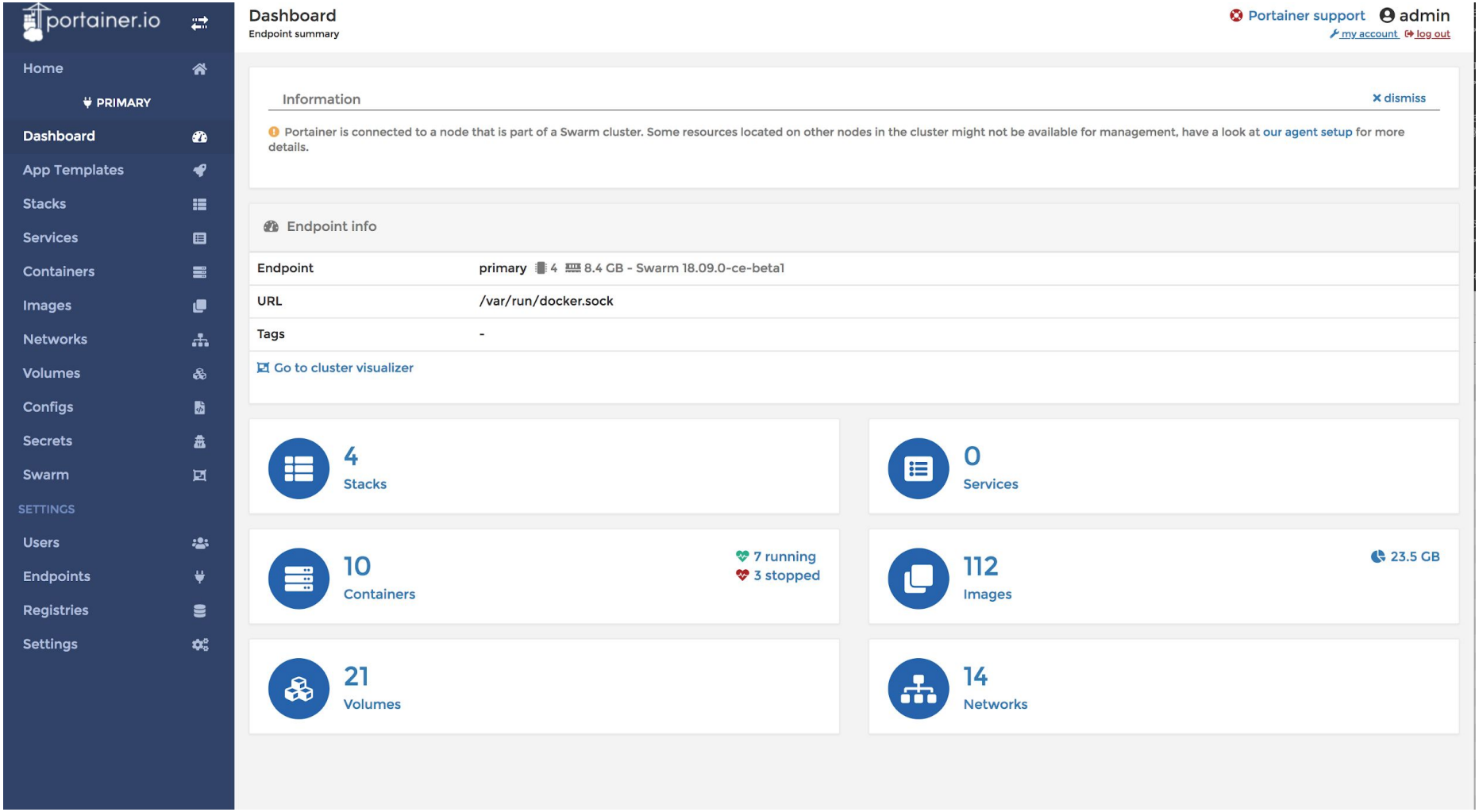
add more charts
add more alarms

netdata on **example.com**, collects every second **956** metrics, presented as **226** charts and monitored by **0** alarms, using 16 MB of memory for 1 hour, 6 mins and 36 secs of real-time history.

netdata
v1.10.1_rolling

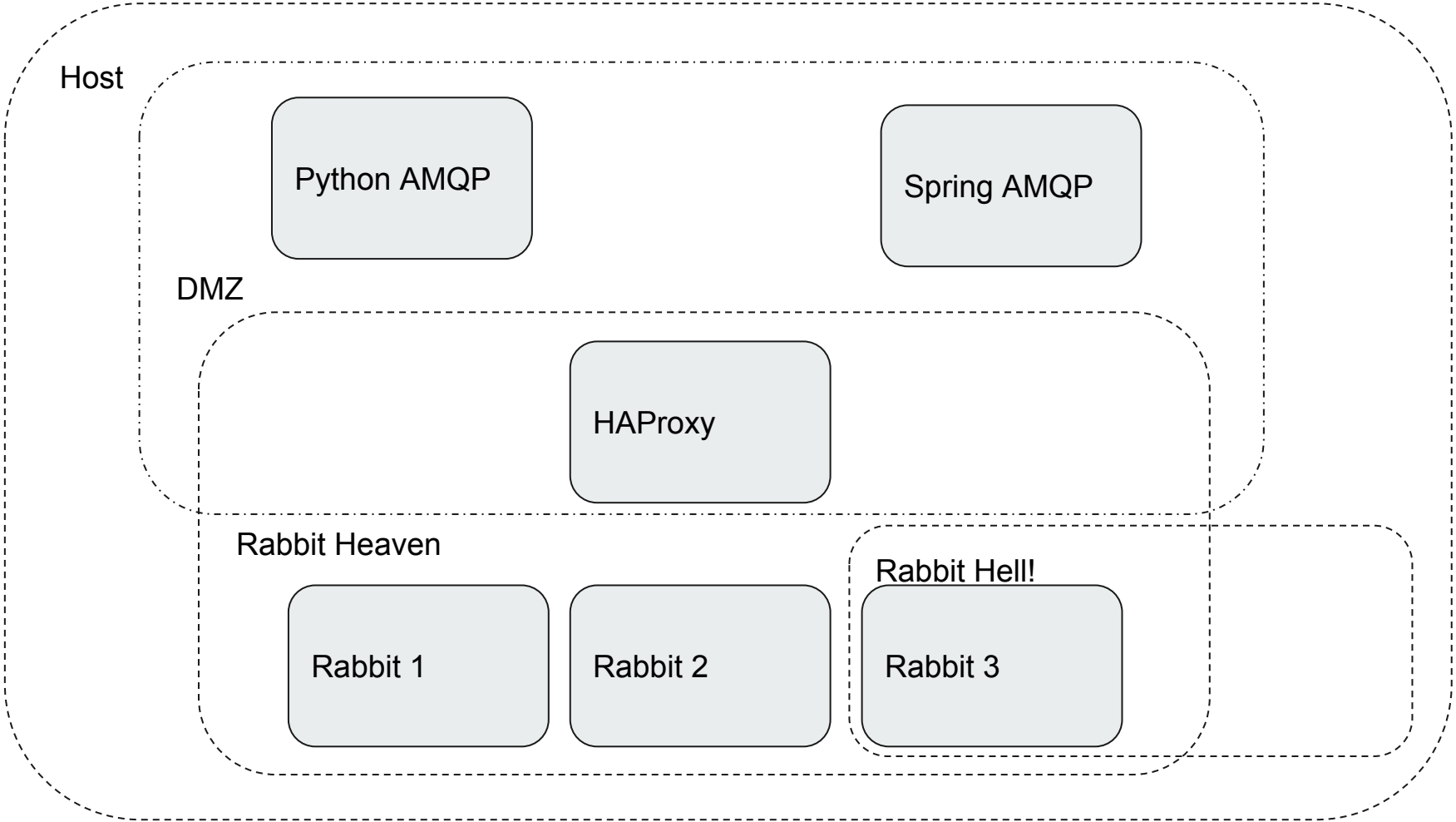
Portainer

```
version: '3'
services:
  portainer:
    image: portainer/portainer
    ports:
      - "127.0.0.1:9000:9000"
    command: -H unix:///var/run/docker.sock
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data

volumes:
  portainer_data:
```


portainer.io
Home
PRIMARY
Dashboard
App Templates
Stacks
Services
Containers
Images
Networks
Volumes
Configs
Secrets
Swarm
SETTINGS
Users
Endpoints
Registries
Settings
Dashboard
Endpoint summary
Portainer support
admin
my account
log out
Information
dismiss
Portainer is connected to a node that is part of a Swarm cluster. Some resources located on other nodes in the cluster might not be available for management, have a look at our agent setup for more details.
Endpoint info
Endpoint
primary 4 8.4 GB - Swarm 18.09.0-ce-beta1
URL
/var/run/docker.sock
Tags
-
Go to cluster visualizer
4
Stacks
0
Services
10
Containers
7 running
3 stopped
112
Images
23.5 GB
21
Volumes
14
Networks

RabbitMQ Cluster

- 3 Node Rabbit MQ cluster
- HAProxy reverse proxy load balancer
- Python and Spring boot clients
- Each exposes a simple HTTP endpoint to send load


Host
Python AMQP
Spring AMQP
DMZ
HAProxy
Rabbit Heaven
Rabbit Hell!
Rabbit 1
Rabbit 2
Rabbit 3

Elasticsearch Host Monitor Cluster

- Dynamically scalable elasticsearch cluster
- Kibana
- Grafana
- Metricbeat monitoring host and docker engine on the host