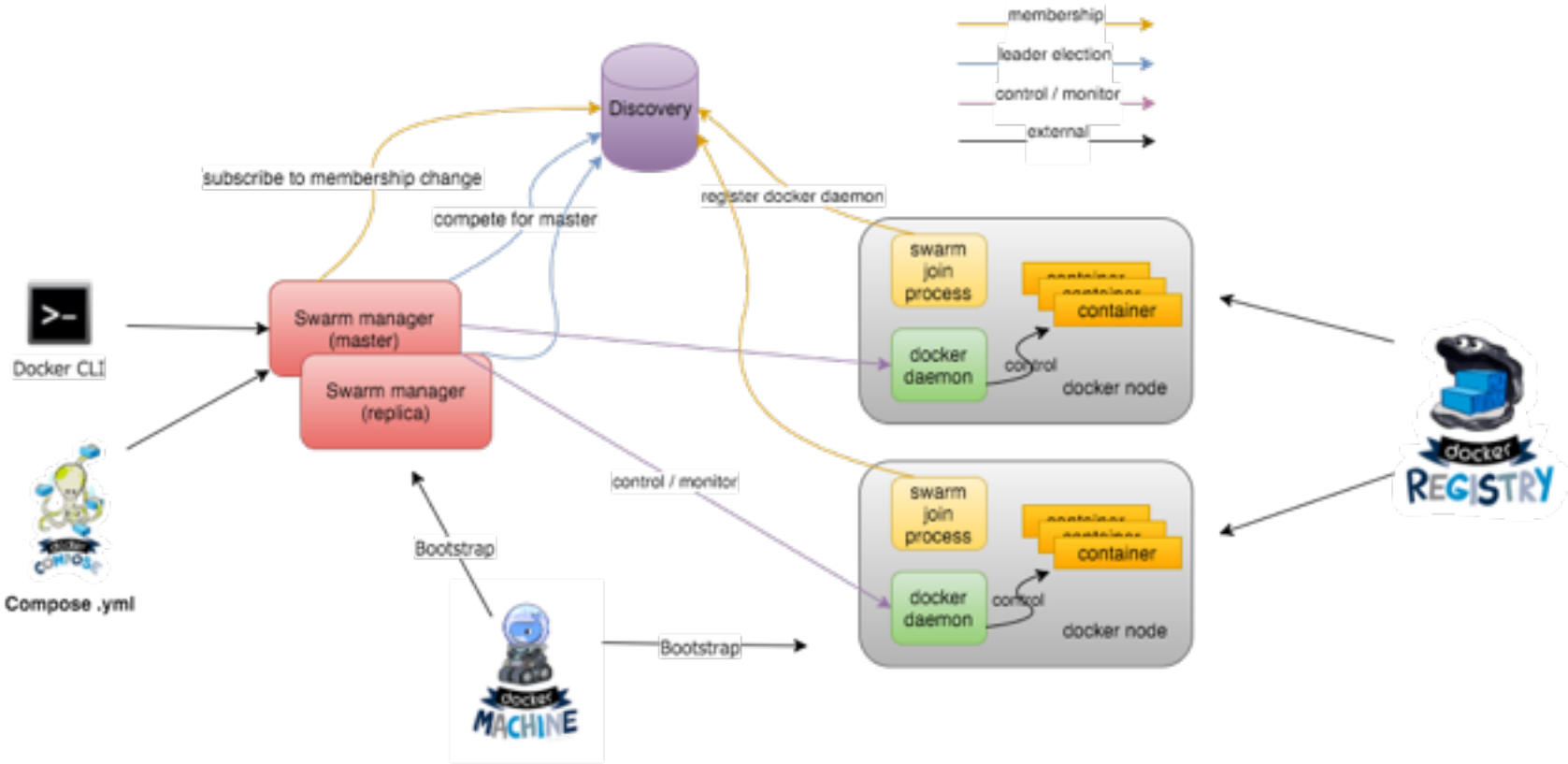
Docker Clustering Update since 1.10

Dongluo (Dong) Chen
March 2016

Overview

- Docker clustering tools include Docker Machine, Docker Compose and Docker Swarm. Each tool resolves one specific problem. Docker machine provisions machines from different cloud providers or local hypervisor. Docker Compose defines a desired service state. Swarm deploys your services to a cluster.
- These tools work together to support microservice architecture. Developers run tests on laptop and Ops deploy services to production can use the same set of tools.
- Since Docker 1.10+ a series of features improves clustering. Including
- New compose file format
- Security
- Dynamic daemon update
- Separation of Docker Engine and Containerd
- Overlay network, DNS, Network alias
- Swarm automatic rescheduling and improved node management
- Docker Clustering in Windows and Azure

Docker clustering tools



Swarm is native Docker clustering tool

- Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host.
- A cluster manages a pool of computer resources, including CPU, memory, storage, bandwidth.
- Swarm supports Docker REST API. Commands to Docker Engine work with Swarm.
- Swarm works seamlessly with Docker components, Compose, Machine, Network, Registry.
- Swarm follows Docker's "battery included but swappable" principle. Swarm can integrate with community tools. One example is you can run Mesos scheduler on top of a Swarm cluster.

New Compose file

- Docker Compose file describes a multi-container service in a single file for deploying to a docker engine or a swarm cluster
- Docker 1.10 introduces version 2 of Compose file where “services, volumes, networks” are top level elements. It also defines dependencies between them
- Networks and volumes can be backed by drivers, e.g., overlay network or flocker volume

```
version: "2"

services:
  web:
    image: myapp
    networks:
      - front
      - back
  redis:
    image: redis
    volumes:
      - redis-data:/var/lib/redis
    networks:
      - back

volumes:
  redis-data:
    driver: flocker

networks:
  front:
    driver: overlay
  back:
    driver: overlay
```


Security update

- Seccomp profiles: lock down container capability to only what they need (docker/docker/#17989)
- User Namespaces: Give containers their own UID/GID. Root in container is not root at host.
- Docker authorization plug-in infrastructure (#15365). Using an authorization plugin, a Docker administrator can configure granular access policies for managing access to Docker daemon.
- Content addressable image IDs. Image IDs is based on a secure hash of the image and layer data. It provides a built-in way to avoid ID collisions and guarantee data integrity after pull, push, load, or save.

```
dchen@vm2:~/temp/demo$ cat chmod.json

{

  "defaultAction": "SCMP_ACT_ALLOW",

  "syscalls": [

    {

      "name": "chmod",

      "action": "SCMP_ACT_ERRNO"

    }                                                    ]

}
```

dchen@vm2:~/temp/demo$ docker run busybox chmod 400 /etc/hostname

dchen@vm2:~/temp/demo$ docker run **--security-opt seccomp:chmod.json** busybox chmod 400 /etc/hostname

chmod: /etc/hostname: **Operation not permitted**

Dynamic update

- Separation of Docker Engine and containerd in Docker 1.11. Containerd manages the runtime of containers. Restarting Engine would not interrupt the containers.
- Docker **update** command can change container's resource requirement (CPU/mem/blkio/…) without starting container. You can dynamically change resource allocation to different jobs
- Daemon label and debug option can be changed without restarting daemon.
- User can change Daemon label to indicate a node's capability or state

```
sudo /usr/bin/docker daemon --config-file=daemon.json -H
172.19.143.233:4444

#change daemon.json
…
[ec2-user@docker]$ cat daemon.json
{
   "labels":["maintenance=true"],
   "debug": true
}

[ec2-user@docker]$ sudo kill -1 25218
[ec2-user@docker]$ docker info
…
ID:
QLVS:VHZN:TCDE:CRMW:NEBS:MPI6:OQJN:MAVW:NMJJ:
6GZ6:4MID:4F7X
Debug mode (server): true
 File Descriptors: 17
 Goroutines: 50
 System Time: 2016-02-24T19:45:08.463633129Z
…
 Docker Root Dir: /var/lib/docker
Labels:
 maintenance=true
Cluster store: consul://manager_wily:8500/swarm
Cluster advertise: 172.19.143.233:4444
```


Docker network update

- Docker 1.10 adds `--ip` support so administrator can assign IP address to containers
- Add an embedded DNS resolver
- Support for network-scoped alias using `--net-alias` (#19242)
- From 1.11 resolver supports DNS-based load balancing to multiple containers when they are set with same net-alias

```
ubuntu@manager:~$ docker run -d --name container1
--net mynet --net-alias busytop busybox top
16d989a5557d084ea8e64a9b853adc2c7715a47b8fec7b1b5
8c80e21200b6d51
ubuntu@manager:~$ docker run -d --name container2
--net mynet --net-alias busytop busybox top
533a69f9bfd6cd2940a98d9a34d96a7d93d857b9b07935564
9e38e62d93c034c

# nslookup busytop
Server:                 127.0.0.11
Address:    127.0.0.11#53
Non-authoritative answer:
Name:       busytop
Address: 10.0.3.2
Name:       busytop
Address: 10.0.3.3

# ping -c 1 busytop
PING busytop (10.0.3.3) 56(84) bytes of data.
64 bytes from container2.mynet (10.0.3.3):
icmp_seq=1 ttl=64 time=0.076 ms

# ping -c 1 busytop
PING busytop (10.0.3.2) 56(84) bytes of data.
```


Docker Events

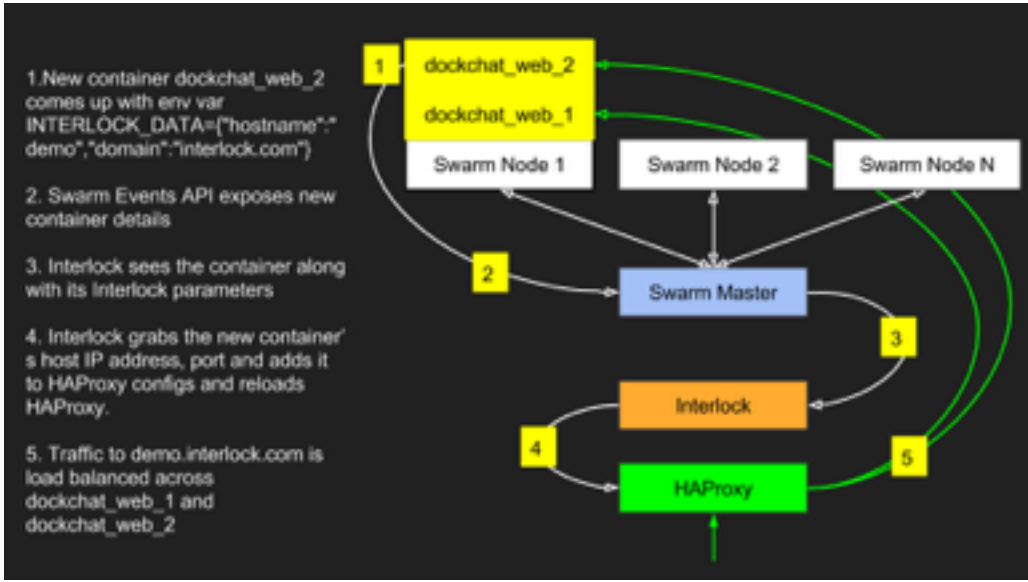
- In Docker 1.10 Daemon normalizes event stream. Events are now consistently structured with a resource type and the action being performed against that resource
- Events have been added for operations on volumes and networks.
- Docker swarm improves event stream reliability
- Docker compose starts to support events

ubuntu@manager:~$ docker -H dong-node-0:2377 **run** -d --name test_container --net=mynet busybox top
f3daac3a4cefd23fea9a5f3a4a3572077a2b2bd7fce5d271f426b91c7c2e2d2a

ubuntu@ip-172-19-95-33:~$ docker -H dong-node-0:2377 **events**
2016-04-13T05:00:56.017368491Z **container create** f3daac3a4cefd23fea9a5f3a4a3572077a2b2bd7fce5d271f426b91c7c2e2d2a (image=busybox, name=test_container)
2016-04-13T05:00:56.219161965Z **network connect** 42d1bbb167caf866b7d19e0707144b1d69d022bfad57d9136ce2063e1b453a9e (container=f3daac3a4cefd23fea9a5f3a4a3572077a2b2bd7fce5d271f426b91c7c2e2d2a, name=mynet, type=overlay)
2016-04-13T05:00:56.913129711Z **container start** f3daac3a4cefd23fea9a5f3a4a3572077a2b2bd7fce5d271f426b91c7c2e2d2a (image=busybox, name=test_container)

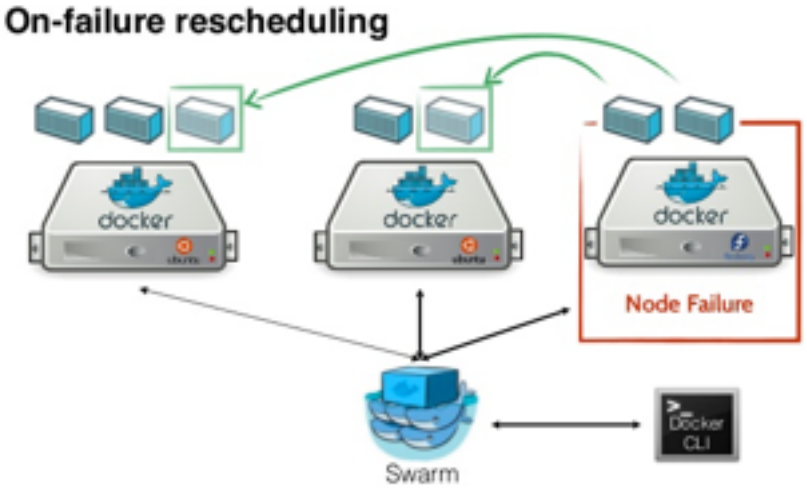
Software load balancer using events

- Use Nginx/HAProxy as load balancer. The key is to update load balancer configuration when there are changes
- Run a HAProxy and an Interlock instance on a node with shared data volume. HAProxy's configuration file is on the shared volume.
- Interlock (github.com/ehazlett/interlock) listens on Swarm events to understand change in web instances.
- One missing feature is health monitoring which is under development in Docker 1.12 (docker issue 21142, 21143).



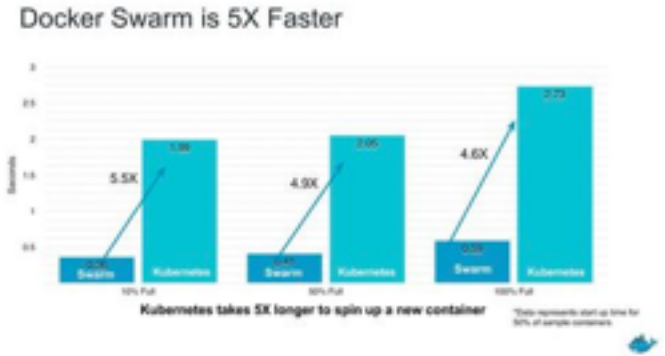
Container Rescheduling in Swarm

- Docker 1.10 adds container rescheduling to Swarm.
- When a node crashes or disconnects from network, Swarm manager reschedule the containers if they have the corresponding configuration
- Health monitoring decides a node is disconnected and triggers event “node disconnected”
- Upon such event, Swarm master goes over containers in the failing node, and reschedule containers according.



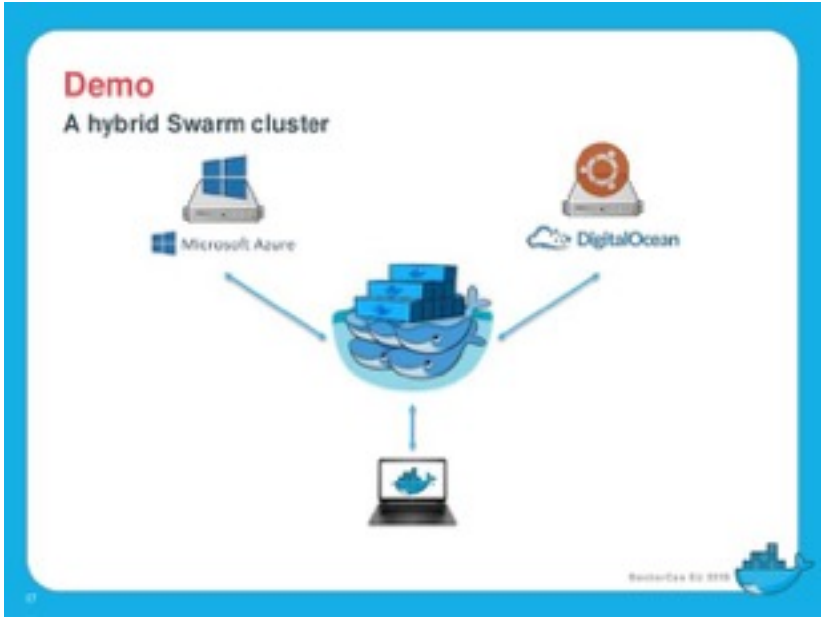
Scalability and Performance

- How to evaluate a platform's scalability and performance? Node count, container count, schedule latency, burst traffic
- Recent Swarm tests on 1k nodes, 30k containers. Swarm scales well due to its simple architecture. Some problems found include TCP link leakage, dead peer detection, go routine max thread count
- Potential bottleneck: network, manager failover, request spikes



Swarm managing Windows node

- DockerConEu demo by Arnaud Porterie
- Microsoft has provided open source templates to support Docker and Docker Swarm in Azure
- Docker machine driver for Azure has been rewritten to improve authentication and resource recreation models
- Swarm.exe can be built from source and run on Windows server
- Docker containers can be run on Windows Server Technical Preview 4 with Container support



Running Containers Windows node

azureuser@swarm-master-0 ~ $ docker -H swarm-master-1:2375 info
Containers: 2
Images: 6
Role: replica
Primary: 10.0.0.5:2375
Strategy: spread
Filters: health, port, dependency, affinity, constraint
Nodes: 5
docker-tp4-2: 192.168.0.8:2375
└ Status: Healthy
└ Containers: 0
└ Reserved CPUs: 0 / 2
└ Reserved Memory: 0 B / 7.35 GiB
└ Labels: executiondriver=
Name: Windows 1854
Build: 1.10.0-dev 59a341e
Default Isolation: process, kernelversion=10.0 10586 (10586.0.amd64fre.th2_release.151029-1700), operatingsystem=**Windows Server 2016 Technical Preview 4**, storagedriver=windowsfilter
…

azureuser@swarm-master-0 ~ $ docker -H swarm-master-1:2375 images

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
nginx	latest	6f62f48c4e55	3 weeks ago	190.5 MB
windowsservercore	10.0.10586.0	6801d964fda5	5 months ago	0 B
windowsservercore	latest	6801d964fda5	5 months ago	0 B

azureuser@swarm-master-0 ~ $ docker -H swarm-master-1:2375 run **windowsservercore tasklist**

Image Name	PID	Session Name	Session#	Mem Usage
System Idle Process	0		0	4 K
System	4		0	136 K
smss.exe	3420		0	1,132 K
csrss.exe	3288	Services	3	3,728 K
wininit.exe	1000	Services	3	4,672 K
services.exe	3200	Services	3	5,388 K
lsass.exe	3052	Services	3	8,988 K

dockercon.com