Scaling Docker Swarm on Azure

Rui Carmo (@rcarmo)

Questions?

#MicrosoftJWS

Agenda

Overview

- Infrastructure Lifecycle
  - Traditional
  - Cloud Scale
  - With Containers

Docker Orchestration Options

- Kubernetes
- Mesos
- Rancher
- Docker Swarm

Docker Swarm Deep Dive (Demo)

- Building an Azure template
- Virtual Machine Scale Sets
- Provisioning with cloud-init
- Scaling a Swarm cluster

Overview

Managing containers (literally) at scale

The Challenge

Docker is changing the way we develop software

- Easier to package, distribute and deploy services
- Managing containers in production is becoming a (reasonably) well-understood problem
- Container orchestration tools are coming of age
- But managing the resources that support those containers (CPU, RAM, storage) is still a big challenge

So let's see how we can begin to address that

Traditional Infrastructure Lifecycle

1. ## Sizing

   Capacity Planning

2. ## Provisioning

   Buying hardware (servers, networking gear, etc.)

3. ## Deployment

   Putting it all together
   Base OS and runtime installation
   Deploying your services

4. ## (No) Scaling

   Under-used (need to buy and assemble more)
   Over-capacity

Cloud Infrastructure Lifecycle

Windows Azure

1 Sizing

Minimum required resources

No need to reserve capacity

2 Provisioning

Allocating CPU and storage
Networking (IP, DNS, etc.)

Everything ready with a few clicks

3 Deployment

Deploy your app stack
Deploy your software

Focus on **your** software

4 Scaling

Vertically (bigger VMs)
Horizontally (more VMs)

Scale what you need, when you need it

Cattle vs Pets

Automation

Container Infrastructure Lifecycle

1 **Sizing**

Minimum required resources (but which ones?)

2 **Provisioning**

Containers don't exist in a vacuum – they still require infrastructure

3 **Deployment**

Container deployment is instantaneous, but how long does it take to set up a cluster of machines?

4 **Scaling**

"Oh, we can just add more containers, right?"
"On which machines?"
"What about when we run out of CPU?"

Container Management at Scale

Eight key aspects to consider in running a container infrastructure

- Cluster Deployment and Management
- Scheduling and Automation
- Service Discovery
- Container Registry
- Container placement / Resource management
- Configuration Management
- Continuous Integration / Continuous Deployment
- Monitoring and Logging

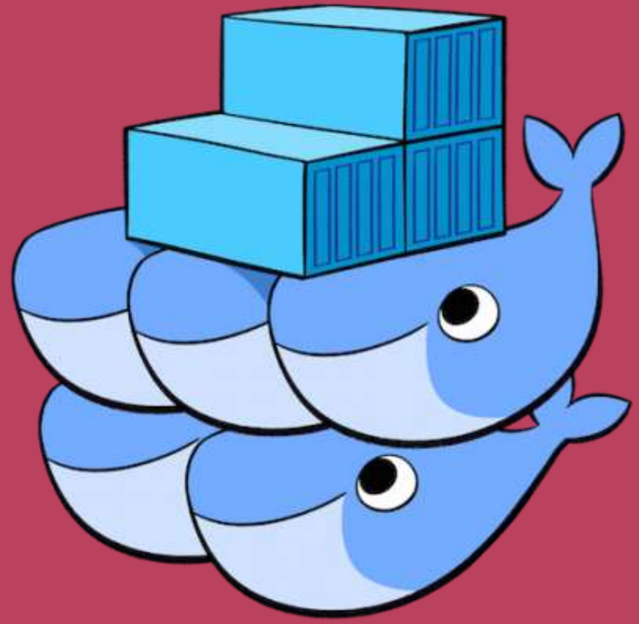
Container Management at Scale: Infrastructure

And these two need to be closely coupled to make effective use of infrastructure resources:

Cluster Deployment and Management	Scheduling and Automation	Service Discovery	Container Registry
Container placement / Resource management	Configuration Management	Continuous Integration / Continuous Deployment	Monitoring and Logging

Orchestration

Container Orchestration



Current Options

Mesos (supported in Azure ACS)
Docker Swarm (also in Azure ACS, now **built-in to Docker 1.12**)
Kubernetes (now also supported in ACS)
Rancher (also supports Kubernetes and Mesos)

Core Functionality

Deploy on pre-defined infrastructure
Schedule containers on (specific) hosts
Basic load balancing, monitoring, logging, etc.

Hot Topics

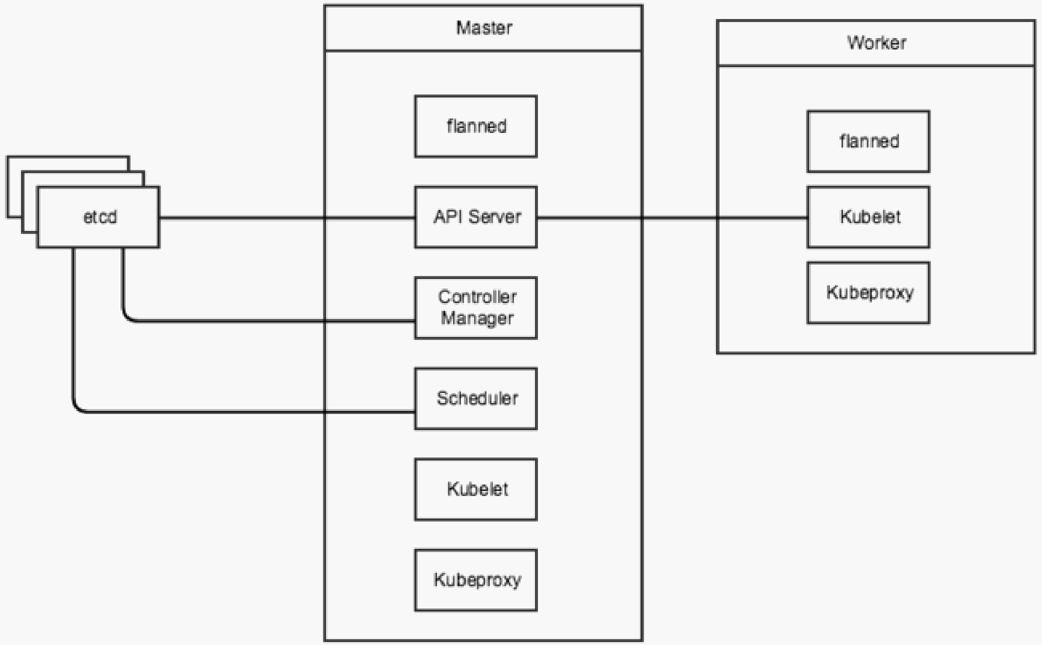
Networking (performance, security, management)
Storage management (volumes and data persistence)
Applications as groups of containers (“pods”, “stacks”, etc.)

Kubernetes

Highlights

Master/worker nodes (with external **etcd** cluster)
Tries to be independent of networking layer
Integrated secret store, DNS server for service discovery

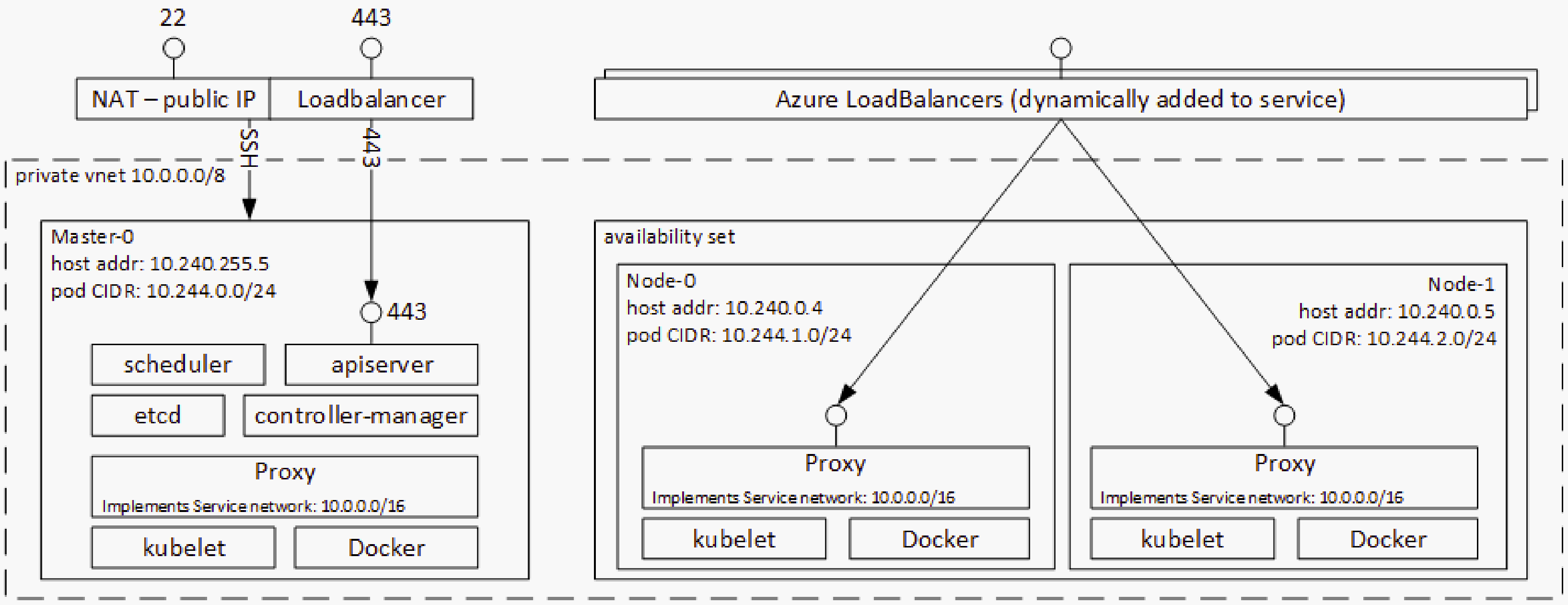
Very complex to set up (uses its own CLI and terminology)
Schedules pods (sets of containers), not single containers
Supports rolling updates, rollbacks and health checks



Kubernetes

Update:

Now available on Azure Container Service!

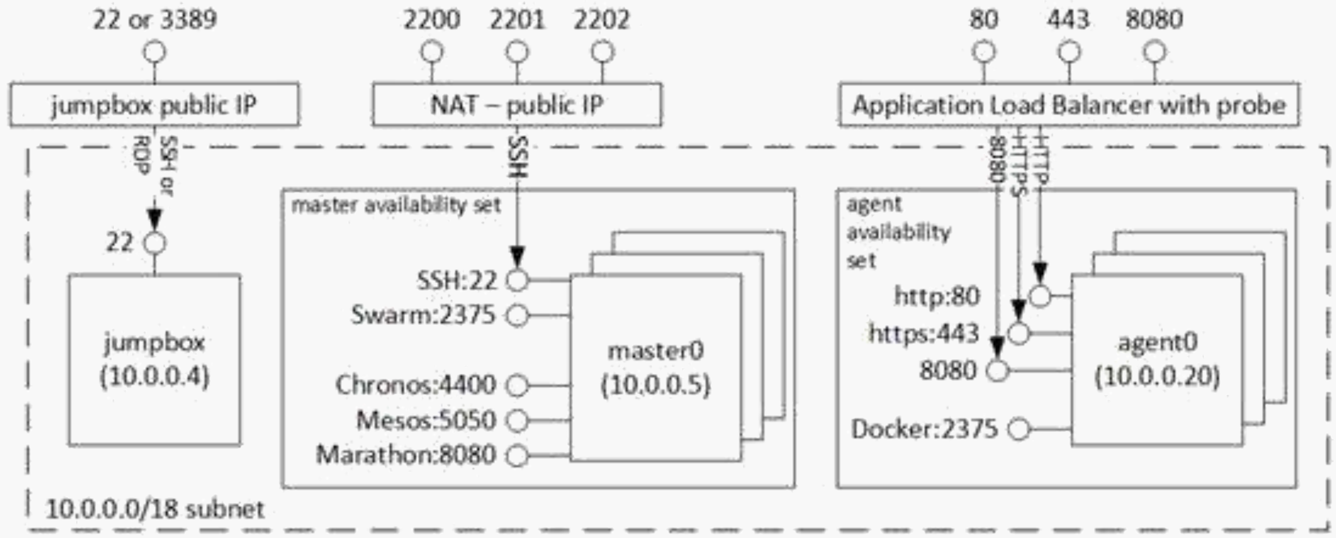


Mesos DC/OS (Marathon)

Highlights

Mesos is a large scale generic cluster management system
Marathon is a container scheduler that uses Mesos
Features like load balancing, DNS and authentication are additional Mesos services

Also uses its own CLI, GUI and terminology
Supports scheduling and scheduling of container groups
Supports rolling and blue-green deployments, app catalogue
Available in Azure Container Service



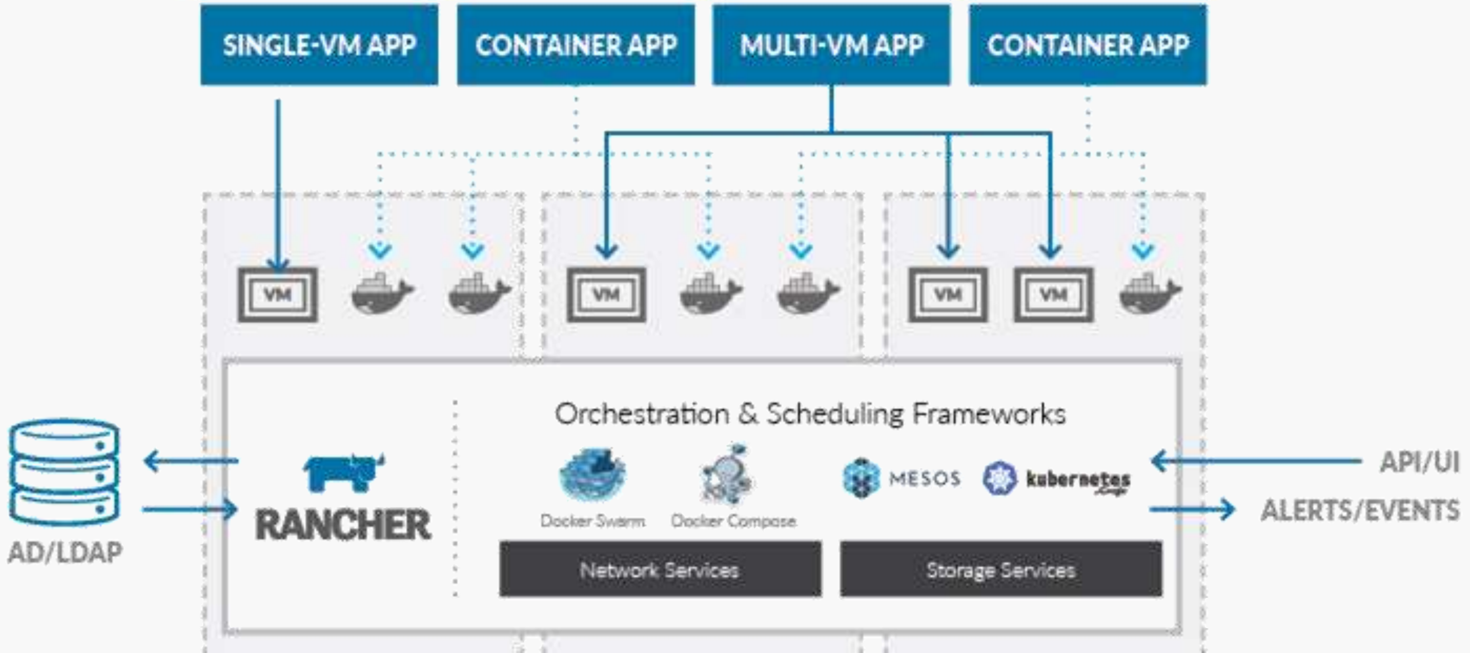
Rancher (Cattle)

RANCHER

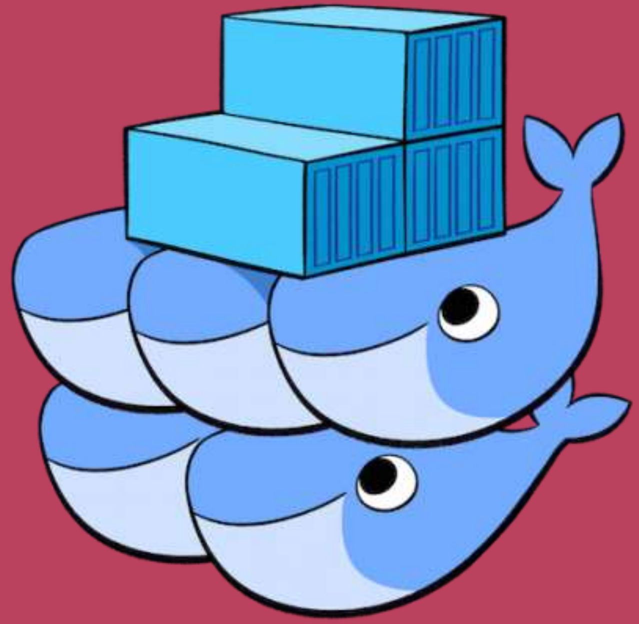
Highlights

Built atop Docker core, simple to deploy and manage
Supports Kubernetes and Mesos as well as Cattle
Provides networking, service discovery, APIs

Cattle uses Docker Compose tooling and terminology
Cattle can schedule and scale “stacks” of containers
GUI for managing multiple environments, projects, access levels
Application catalogue, remote shell to containers in GUI



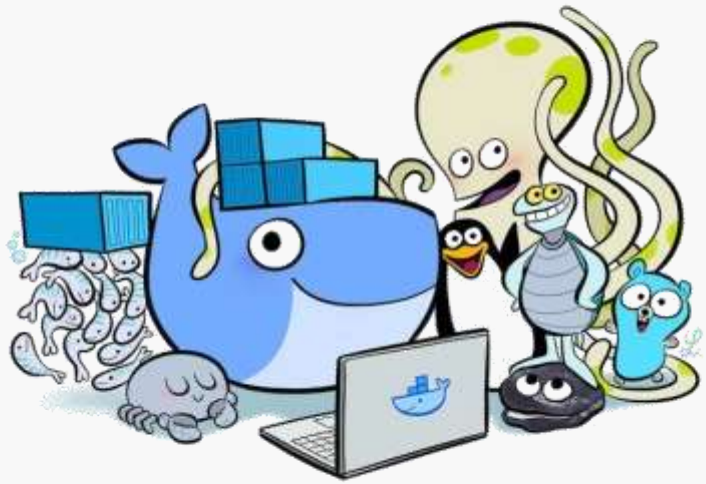
Docker Swarm



Highlights

Now available out-of-the box
Built-in master consensus, networking, security…

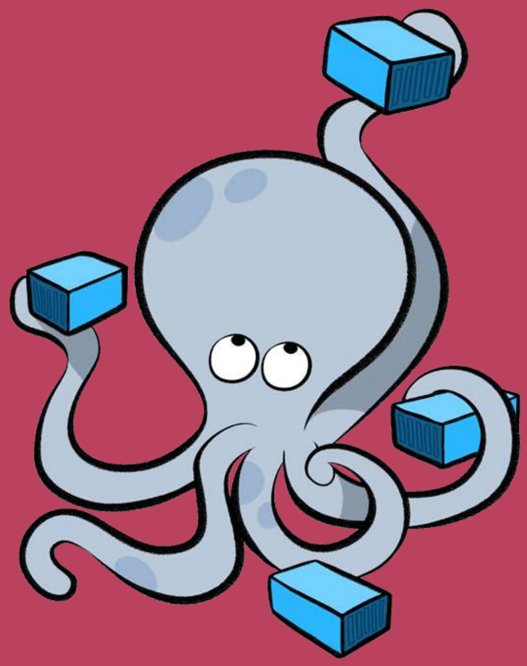
CLI is an extension of docker
Basic scheduling (replicated/global services)
Rudimentary rolling updates (no rollbacks, very basic health checks)
Also available in Azure Container Service



Lots of interest

(we'll dive into it next)

Container Orchestration (the missing bits)



Capacity Planning

Initially: Pre-sized clusters. No ability to add or remove nodes
Now: (basic) ability to add/remove nodes from running clusters

Scaling Live Clusters

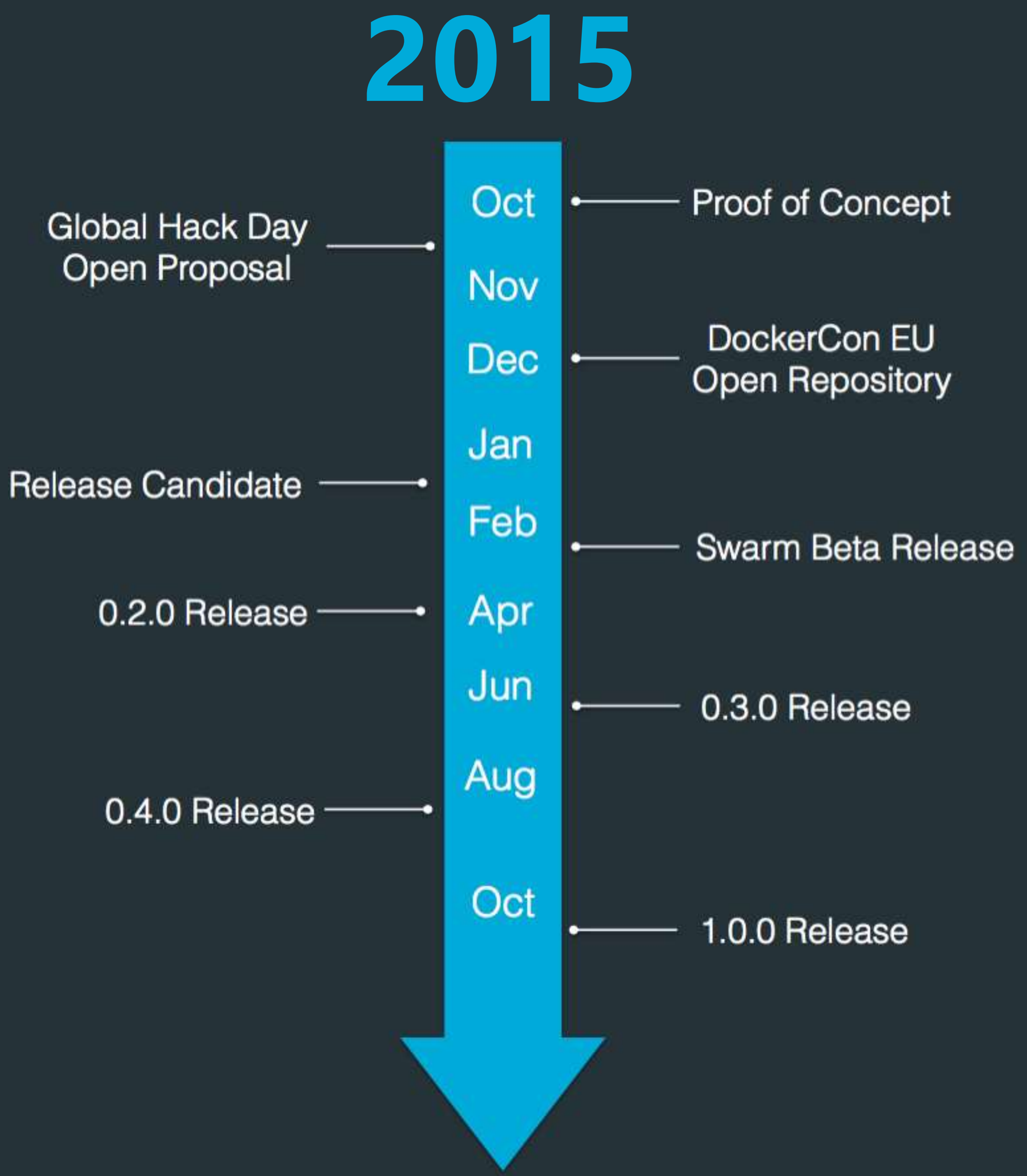
Aligning VM provisioning with container node provisioning
Container migration (maintaining state, quorums, etc.)
Networking (load balancing, security, etc.)
Storage (volume management, replication, etc.)

This is why we're here today

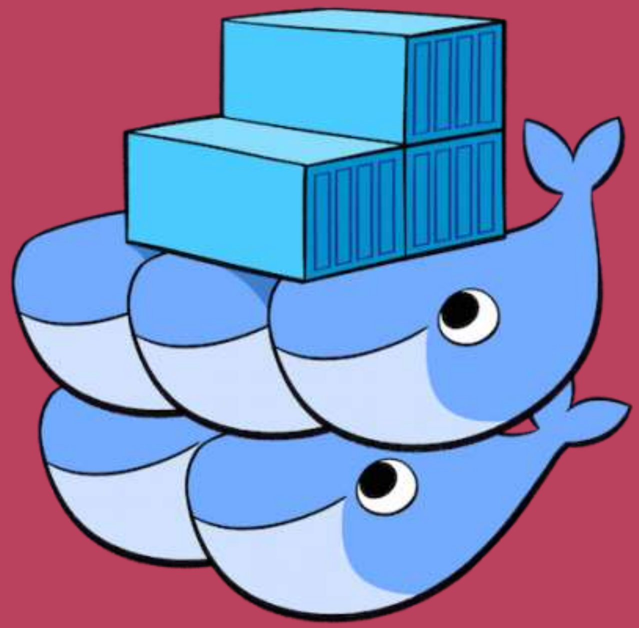
Docker Swarm

Background and features

Timeline (pre 1.12)



Features (1.12)



Clustering

Decentralized design (master election using Raft protocol)
Simple setup with docker CLI

Orchestration

Desired state application reconciliation
Rolling upgrades, scaling, health checks
Service discovery

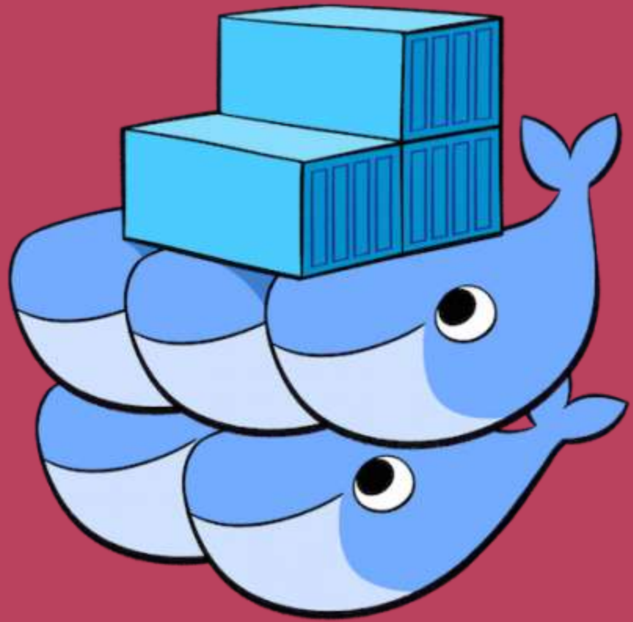
Networking

Built-in load balancing and routing mesh
Container-native overlay networks

Security

Built-in CA, end-to-end TLS security by default

Feature Coverage



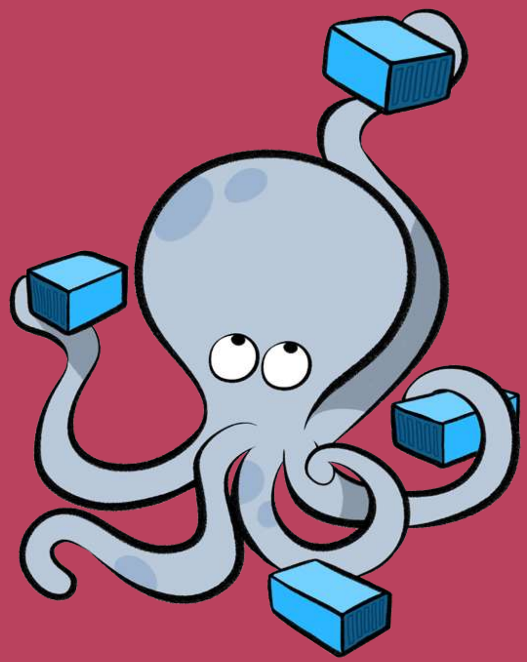
Scoreboard

Cluster Deployment and Management	Scheduling and Automation	Service Discovery	Container Registry
Container placement / Resource management	Configuration Management	Continuous Integration / Continuous Deployment	Monitoring and Logging

Instantiation

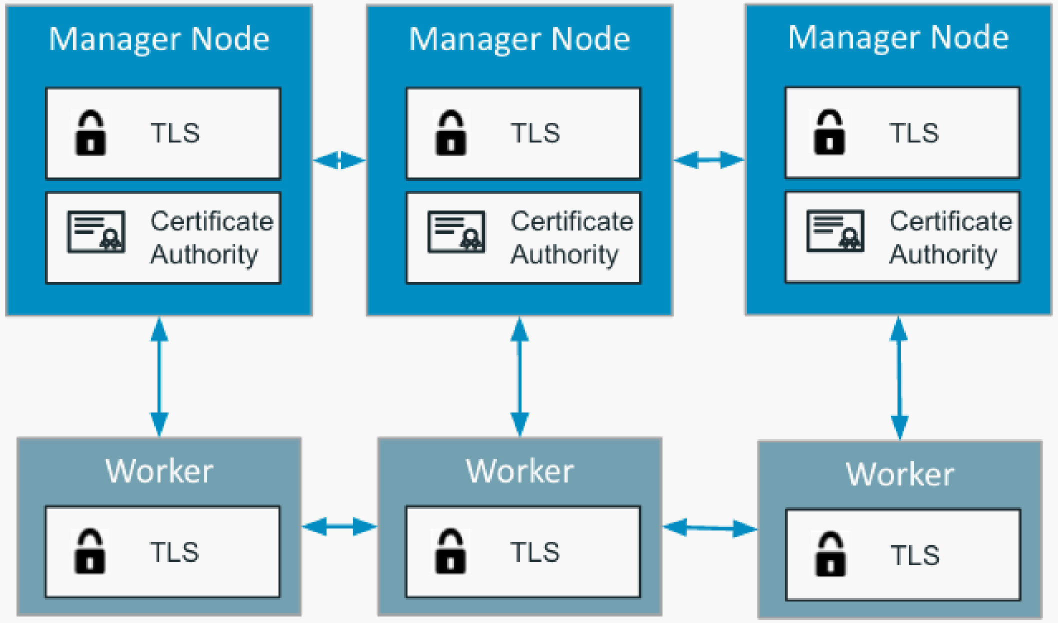
```
cluster@master0:~$ docker swarm init
Swarm initialized: current node
(a2jhx8l3lpkvhvmnhz7jk09kq) is now a manager.

To add a worker to this swarm, run the following
command:

    docker swarm join \
    --token SWMTKN-1-2iefk4nlxpua1802wtf4dk… \
    10.1.0.10:2377

To add a manager to this swarm, run 'docker
swarm join-token manager' and follow the
instructions.
```

So what just happened?

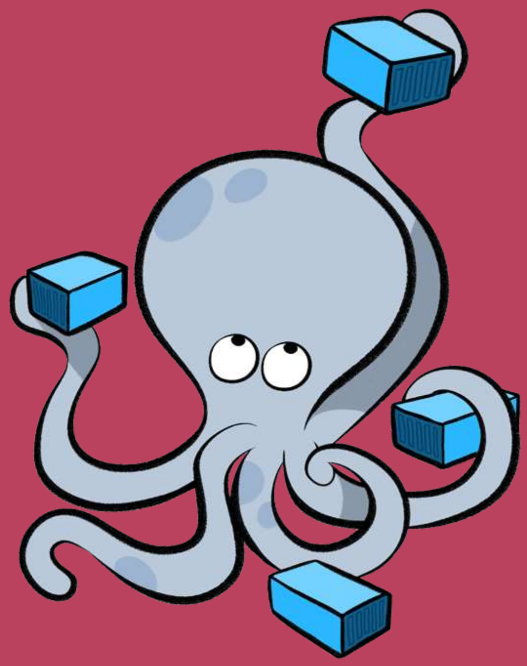


The Docker daemon created:

- A memory-backed state store to keep track of swarm state
- An internal Certificate Authority to secure communications
- A manager service using the Raft protocol to turn the state store and the CA into a secure distributed system



But in real life…



Deploy master0

- Create VM
- Install Docker
- Swarm init
- **Get token**

Deploy worker0

- Create VM
- Install Docker
- **Apply token**

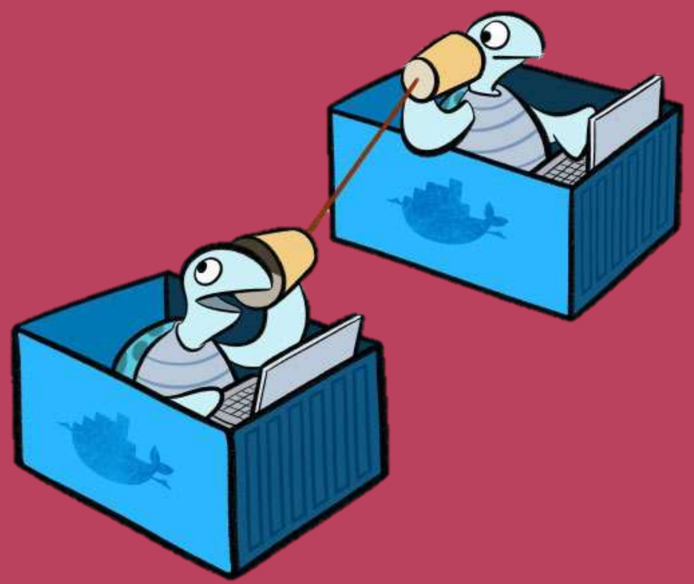
Deploy worker1

- Create VM
- Install Docker
- **Apply token**

…and so on…

Automation

Networking



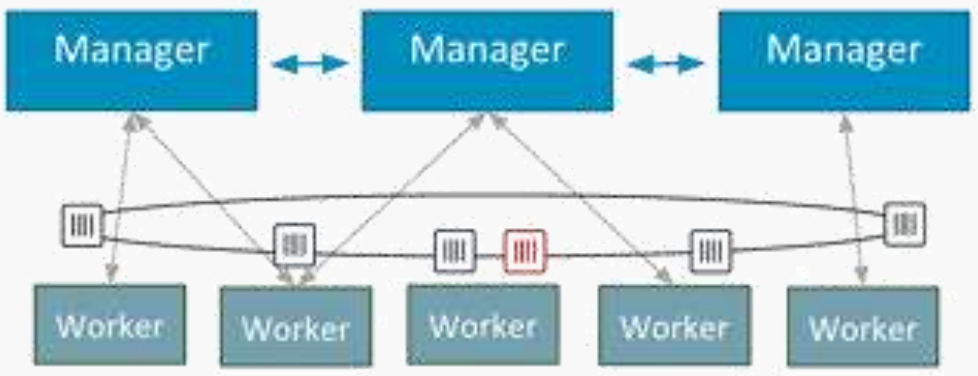
Named, isolated overlay networks

You bind a service to the network when creating it

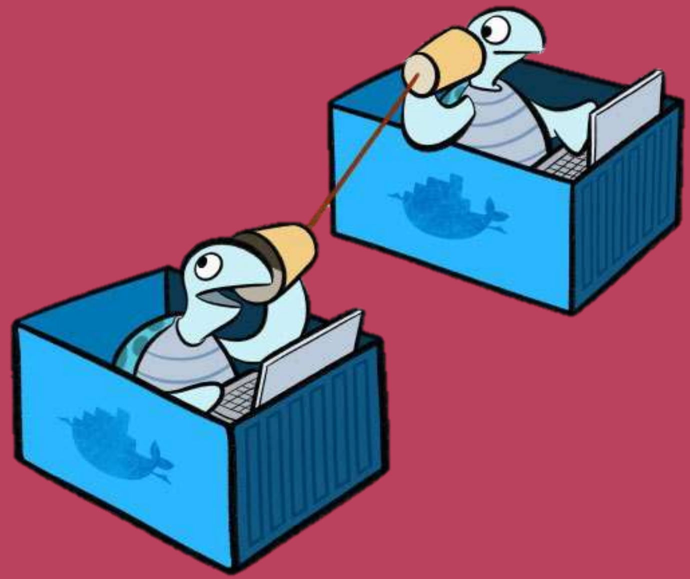
Layer 4 load-balancing
(no HTTPS termination or HTTP awareness built-in)

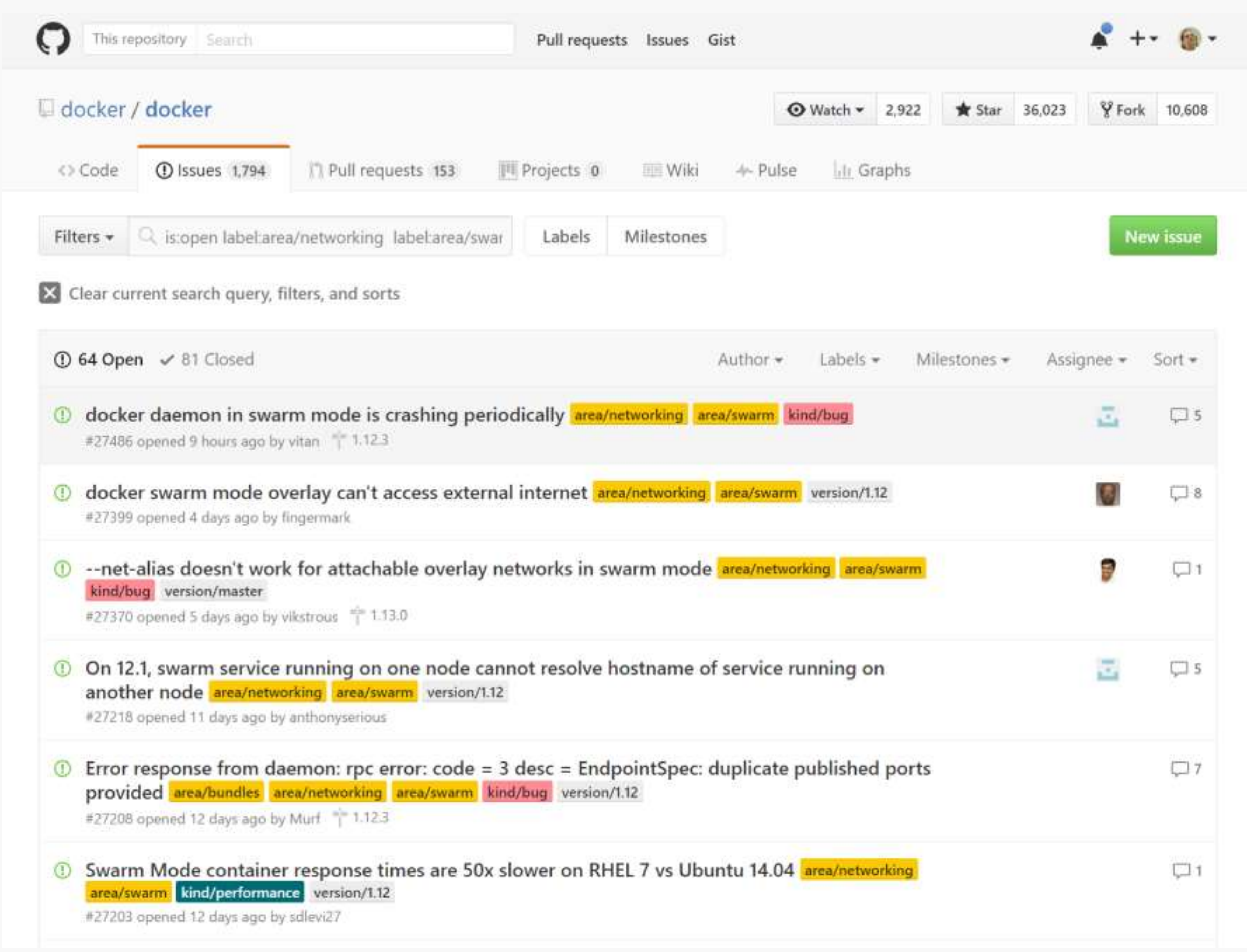
When a service is deployed, **published** ports are open on **every** host

Based on IPVS/netfilter (very high performance)



Networking (caveats)

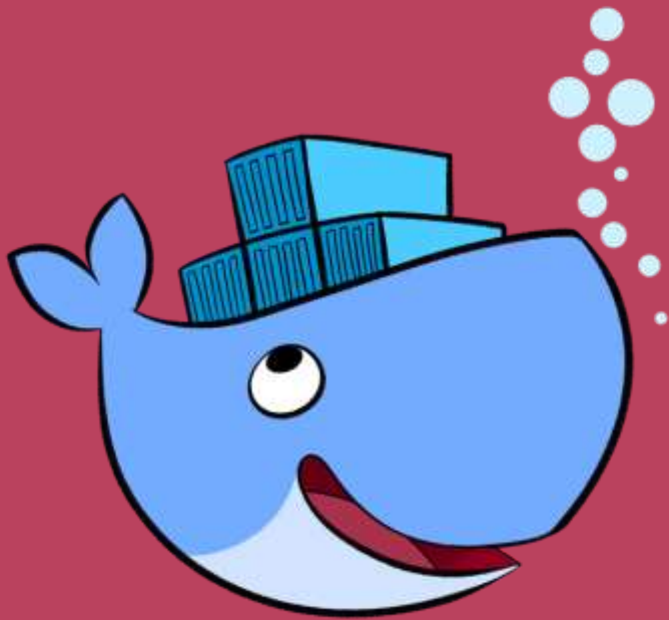


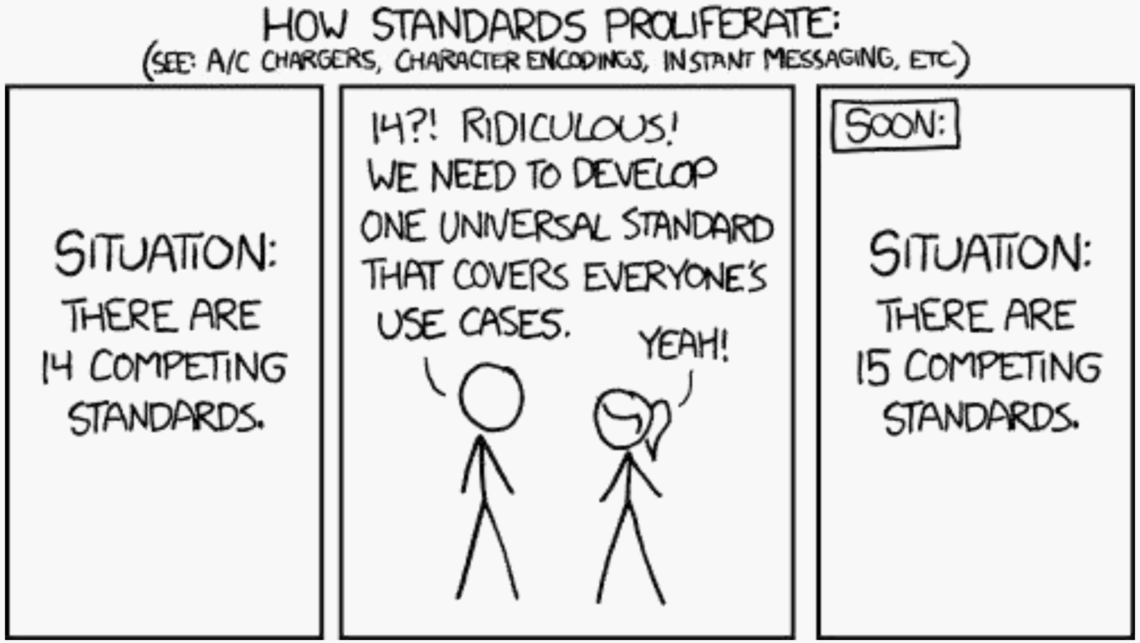


Bundles

A Docker Bundle file is a declarative specification of a set of services that mandates:

- What specific image revision to run
- What networks to create
- How containers in those services must be networked to run





Let's do it!

- Provision Compute and Networking resources using ARM
- Deploy and configure Docker via `cloud-init`
- Automatic Cluster Scaling using Azure VM Scalesets

Building a scalable Swarm cluster

Cluster Template

Network	Storage	Compute	Packages

↓

Deploy Infrastructure

OS	Docker Swarm

↓

Deploy Applications

Swarm Visualizer	Stateless App

↓

Scale

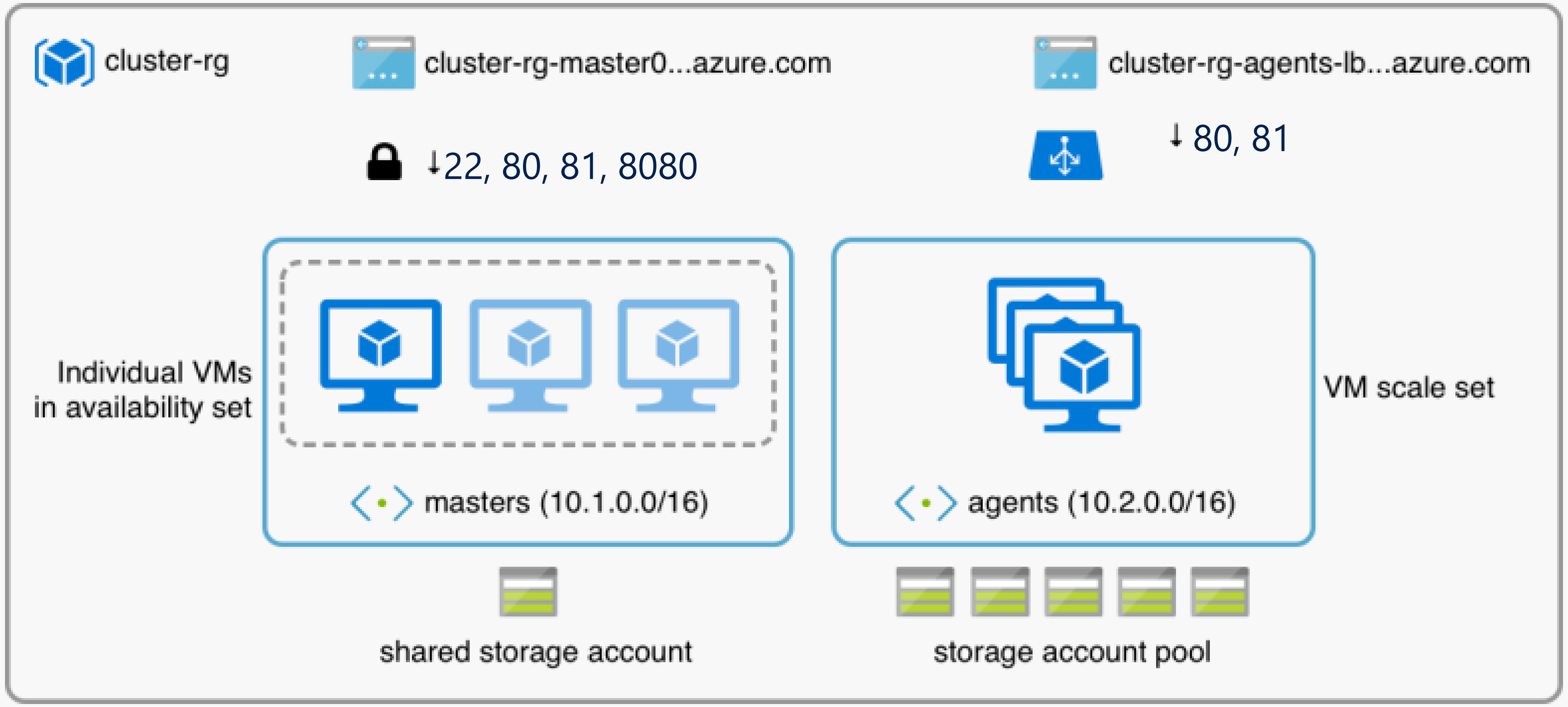
Containers	VMs

Resource Provisioning

Building our infrastructure

Azure Resource Manager Template

Infrastructure as (JSON) code
Compute, storage, networking and security definitions
Sizing, addressing, dependencies between components, etc.



VM Scale Sets

Cattle vs Pets

Pets: Named resources with unique charactristics

Cattle: Numbered, inherently replaceable, interchangeable

- Identical VMs provisioned from the same image and generic settings
- Scale in or out with a simple REST call
- Capacity can be 0 – 100
- VMs balanced across:
  Azure fault/update domains
  Storage Accounts
- Load Balancer integration

VM Scale Sets (autoscaling)

Define thresholds and triggers

- Max – Min VMs
- Trigger – action rules

Standard audit/email notifications

Webhooks

Azure Automation Runbooks


```
    "enabled": true,
    "name": "autoscalewad",
    "targetResourceUri": "/subscriptions/97ad66a
/virtualMachineScaleSets/guyboas28",
    "notifications": [
      {
        "operation": "Scale",
        "email": {
          "sendToSubscriptionAdministrator": tru
          "sendToSubscriptionCoAdministrators":
          "customEmails": [
            "guybo@microsoft.com"
          ]
        },
        "webhooks": [
          {
            "serviceUri": "https://events.pagerd
            "properties": {
              "key1": "custommetric",
              "key2": "scalevmss"
            }
          }
        ]
      }
    ]
```


Windows Azure

VM Scale Sets (provisioning)

Flexible Provisioning and Updates

Provisioning

- Baked-in custom OS images
- Runtime deployment with VM extensions
- **`CustomData/cloud-init`**
- Higher level cluster manager / container service / Config Managers (Chef/Puppet)

Application Updates

- VM extension
- VM image update
- Chef/Puppet
- **Containers**

Server Configuration

cloud-init

cloud-init.readthedocs.io

Bootstrapping using cloud-init

Declare list of package sources and requirements
Ensure specific commands are run upon instance (re)creation

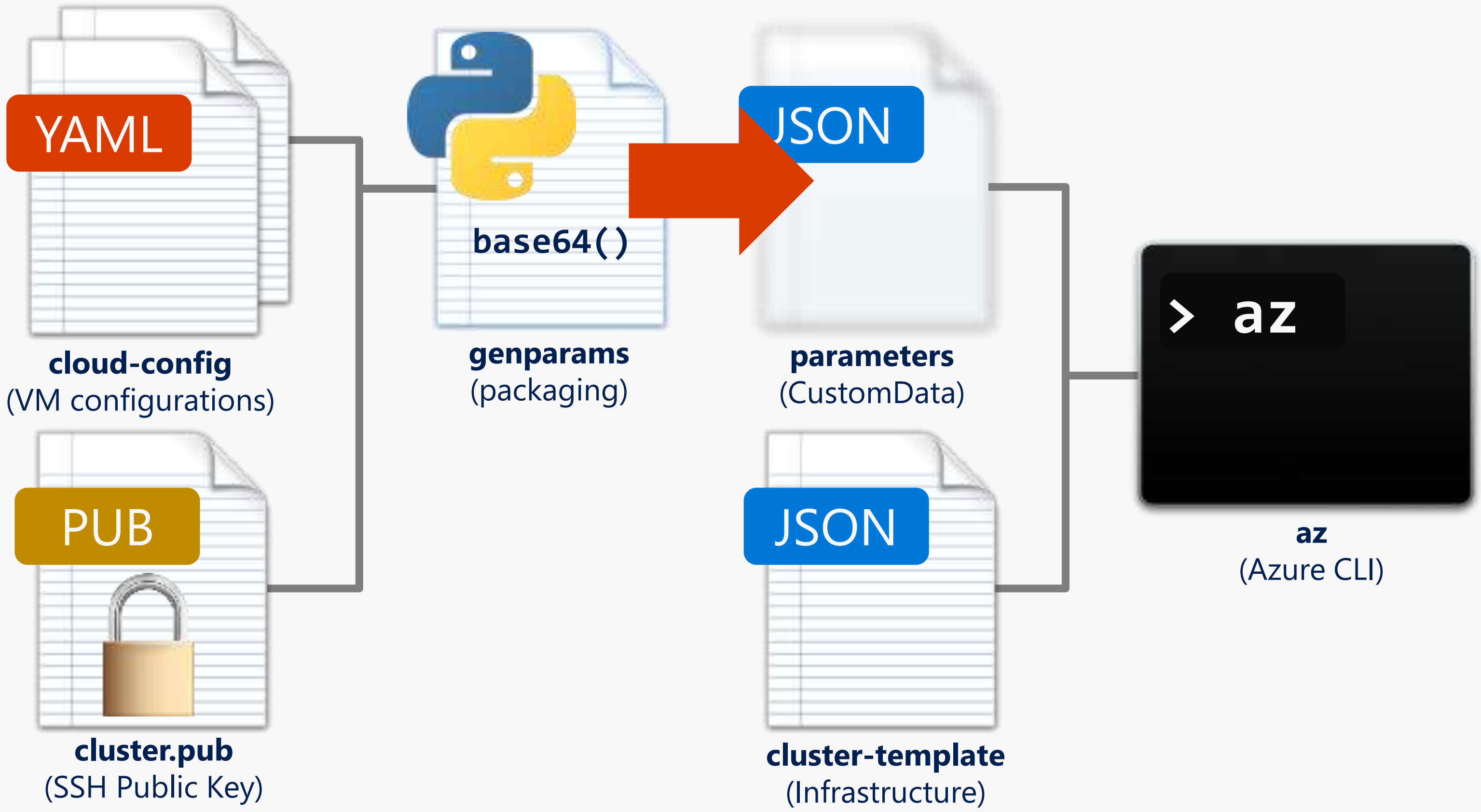
```
apt:
  sources:
    docker.list:
      source: "deb https://apt.dockerproject.org/repo ubuntu-xenial main"
      keyserver: p80.pool.sks-keyservers.net
      keyid: 58118E89F3A912897C070ADBF76221572C52609D

packages:
  - ntp
  - docker-engine

runcmd:
  - usermod -G docker cluster
  - systemctl enable docker
  - systemctl start docker
```

Preparing CustomData

`cloud-init`

Creating Template Parameters

To create the parameters file, we simply pack the SSH key and the cloud-config files into the `CustomData` field using `base64()`



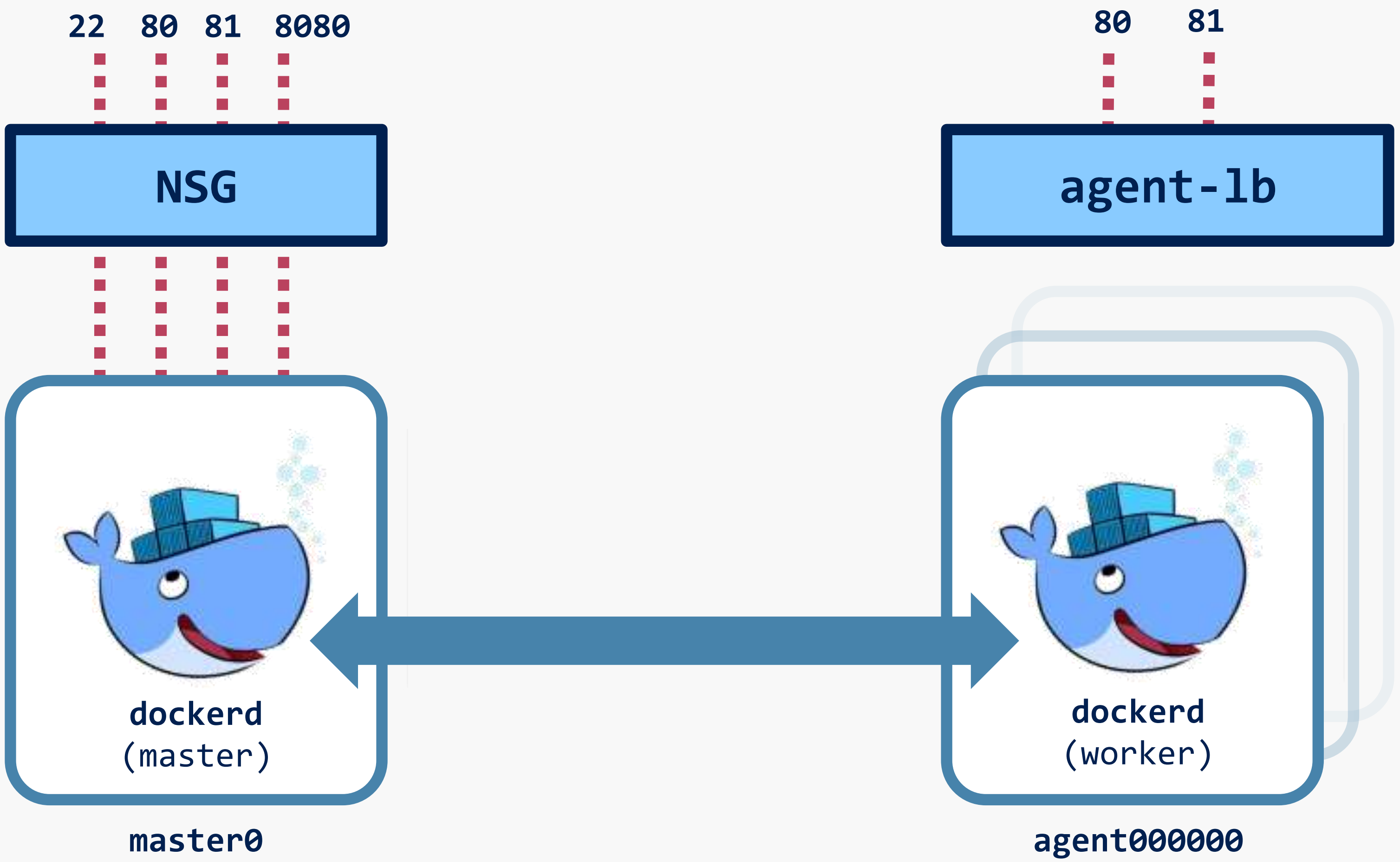
Launching our cluster

Demo

What is running now?

Open Ports

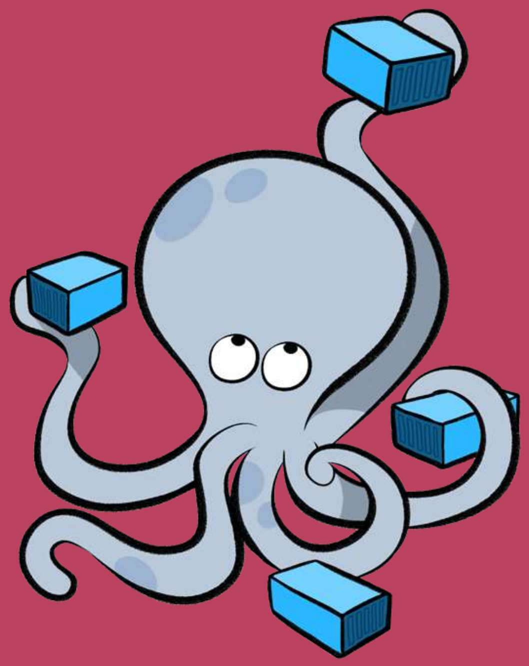
The master is protected by an Azure Network Security Group, and agents are behind an Azure Load Balancer



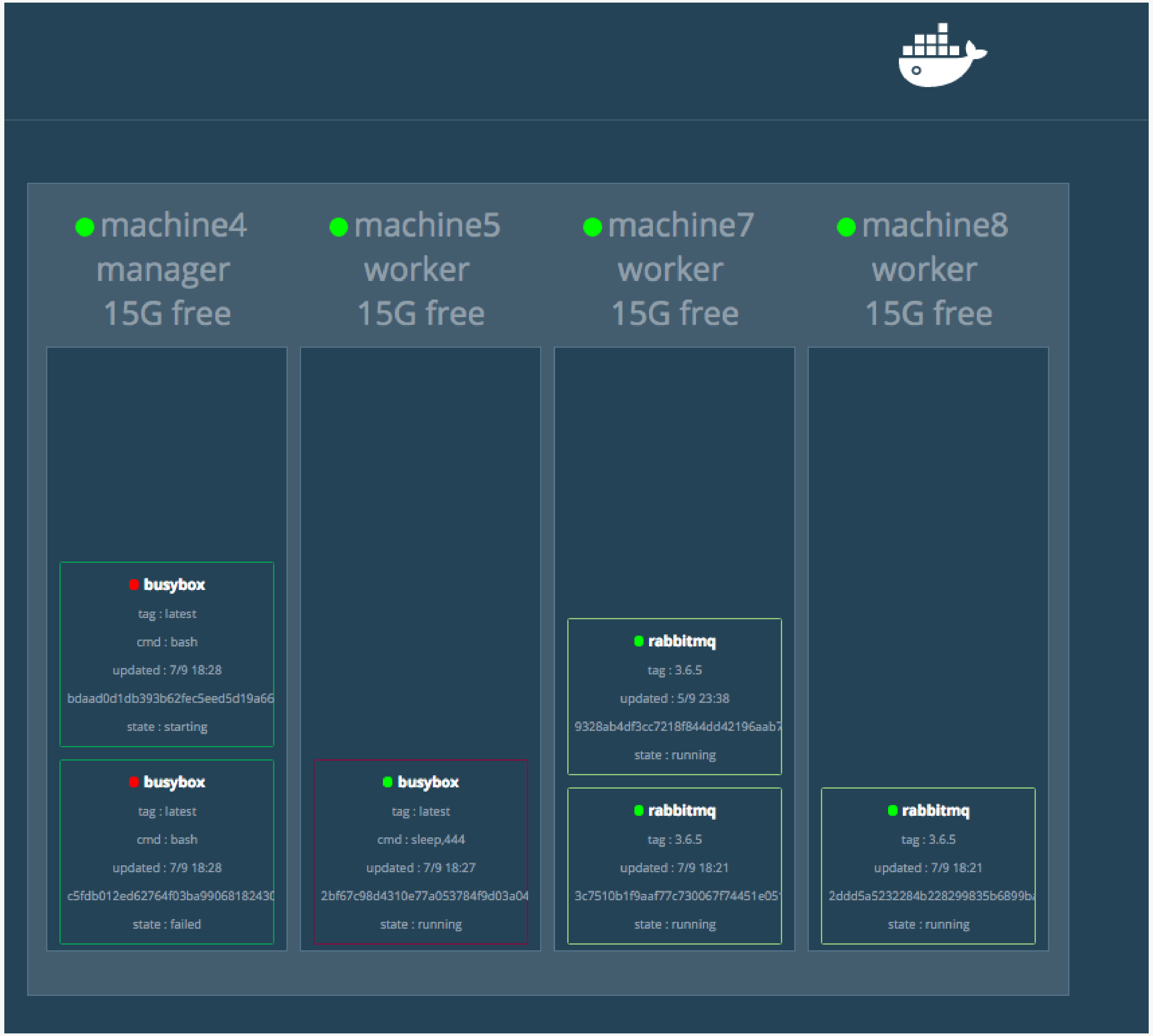
Seeing what's inside

Demo

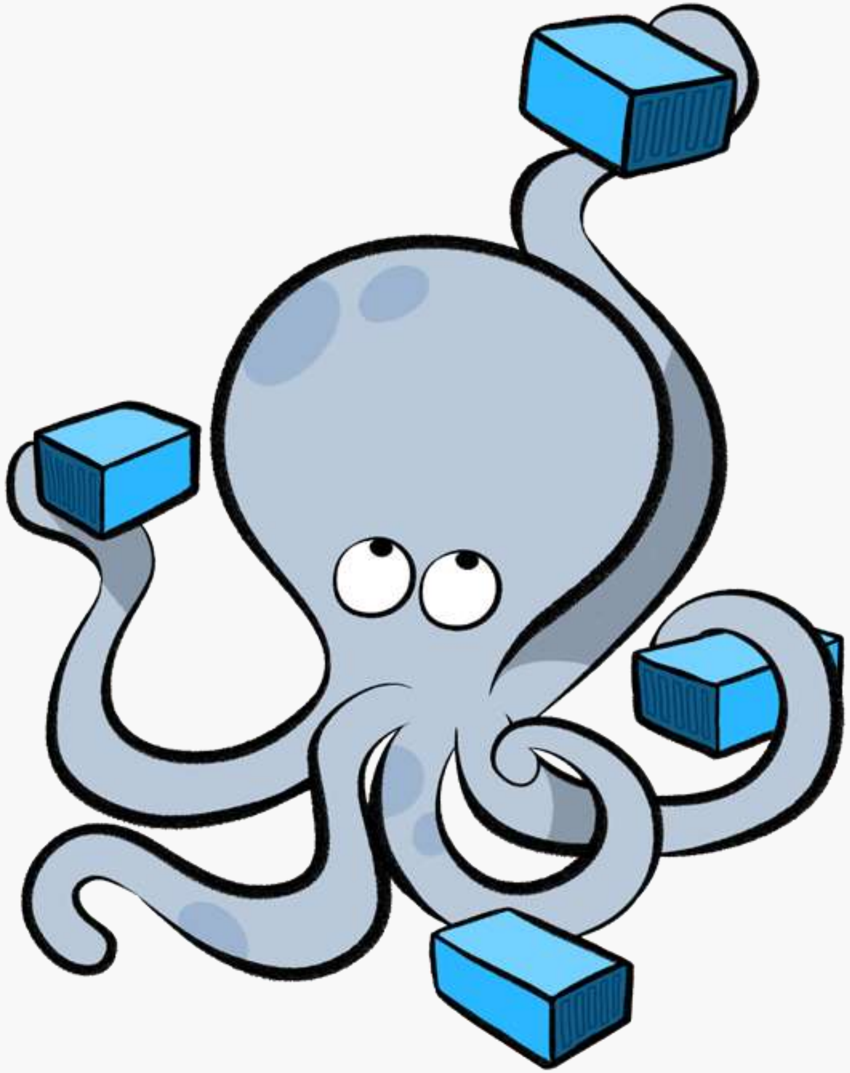
Deploying a visualizer



```
docker run -d -p 8080:8080 … manomarks/visualizer
```



Deploying Services



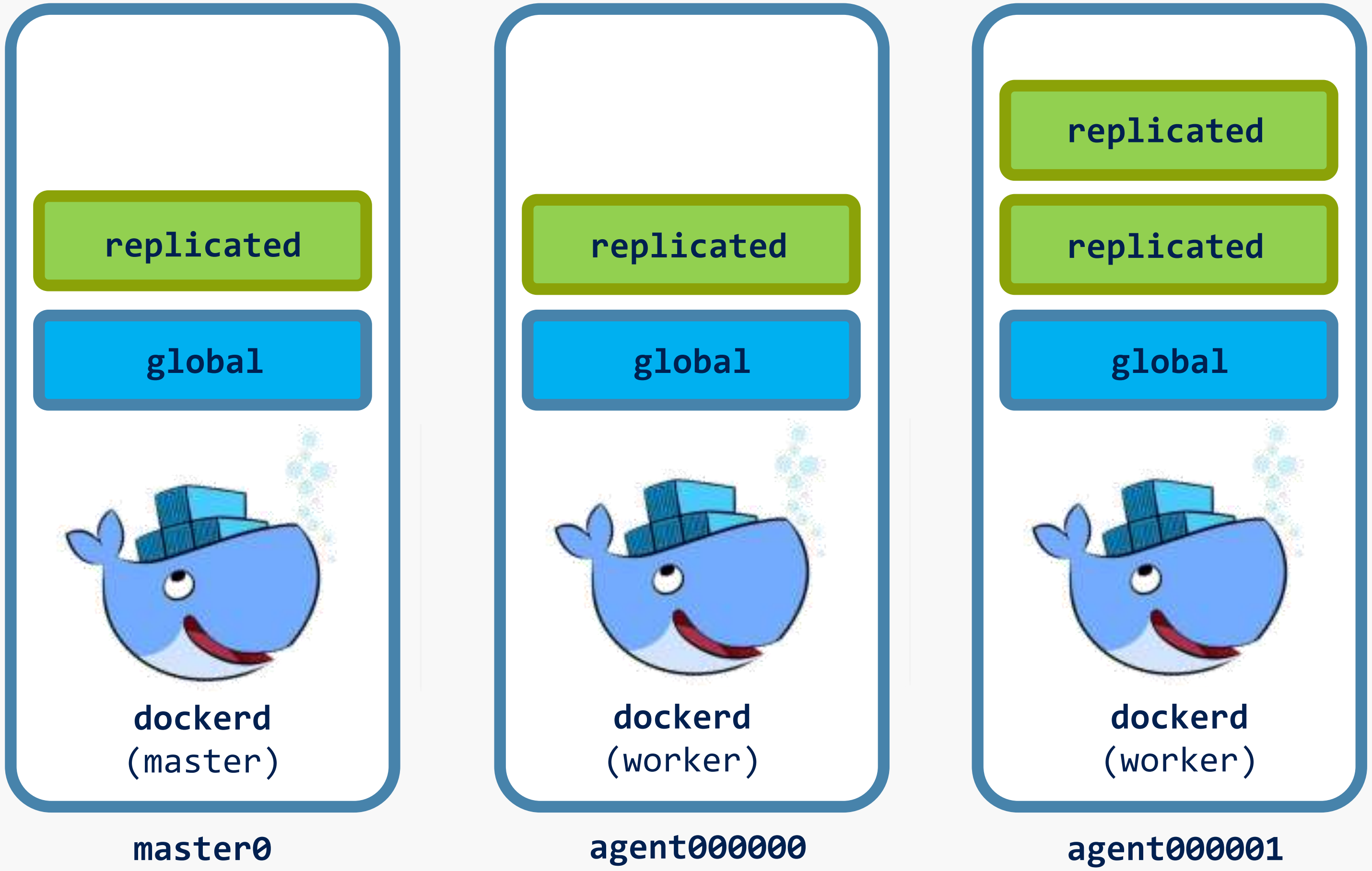
Demo

What is running now?

Global vs Replicated Services

Global services run one instance on every node

Replicated services run as many instances as required (4 in this case), regardless of location



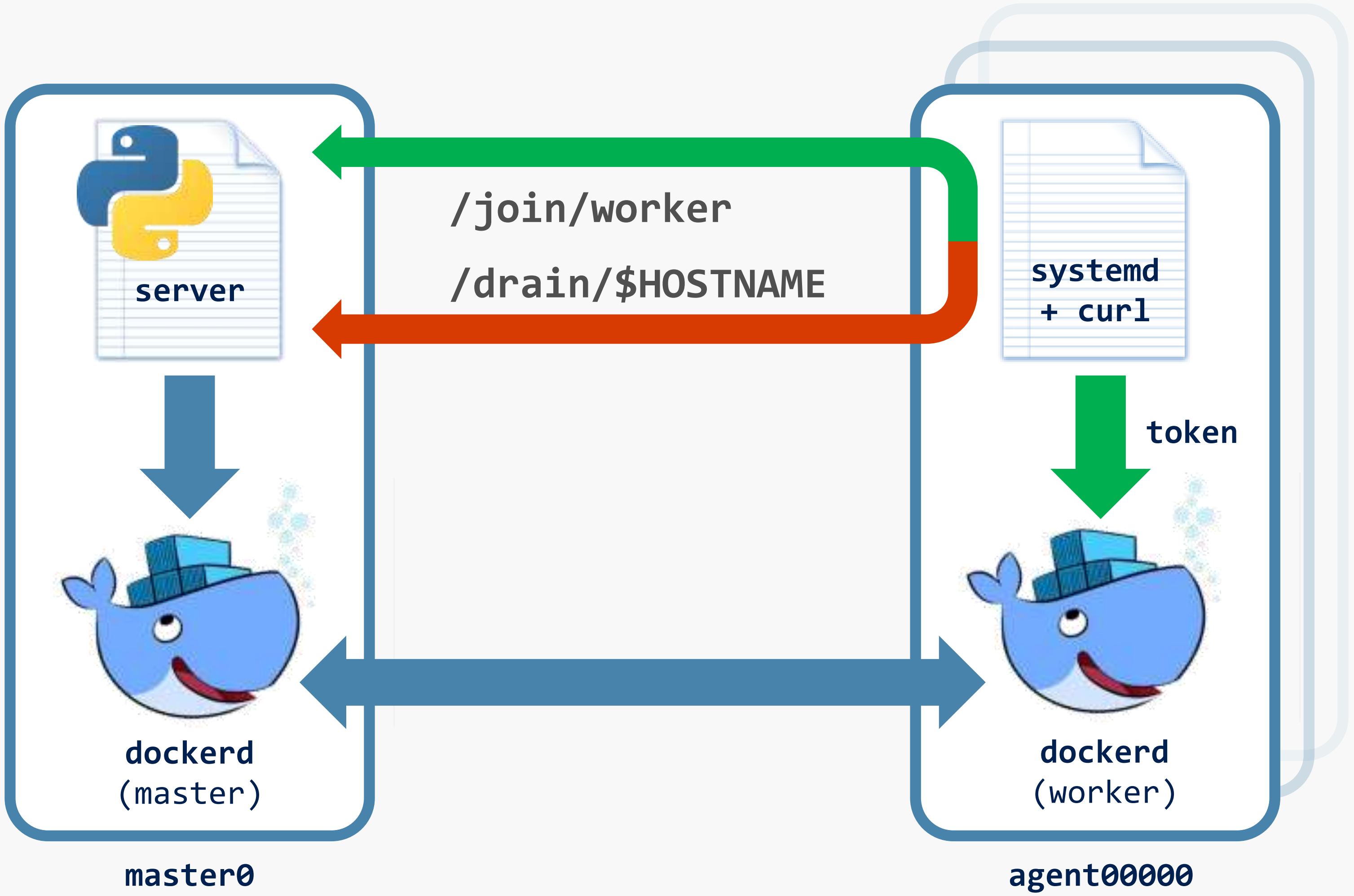
Scaling our cluster

Demo

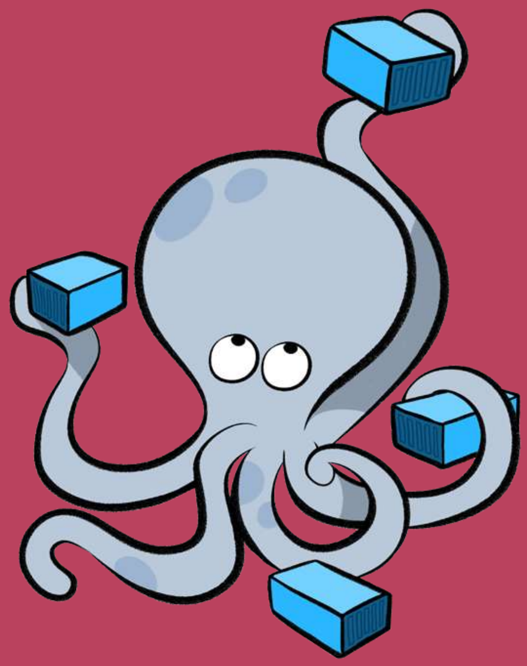
What is running now?

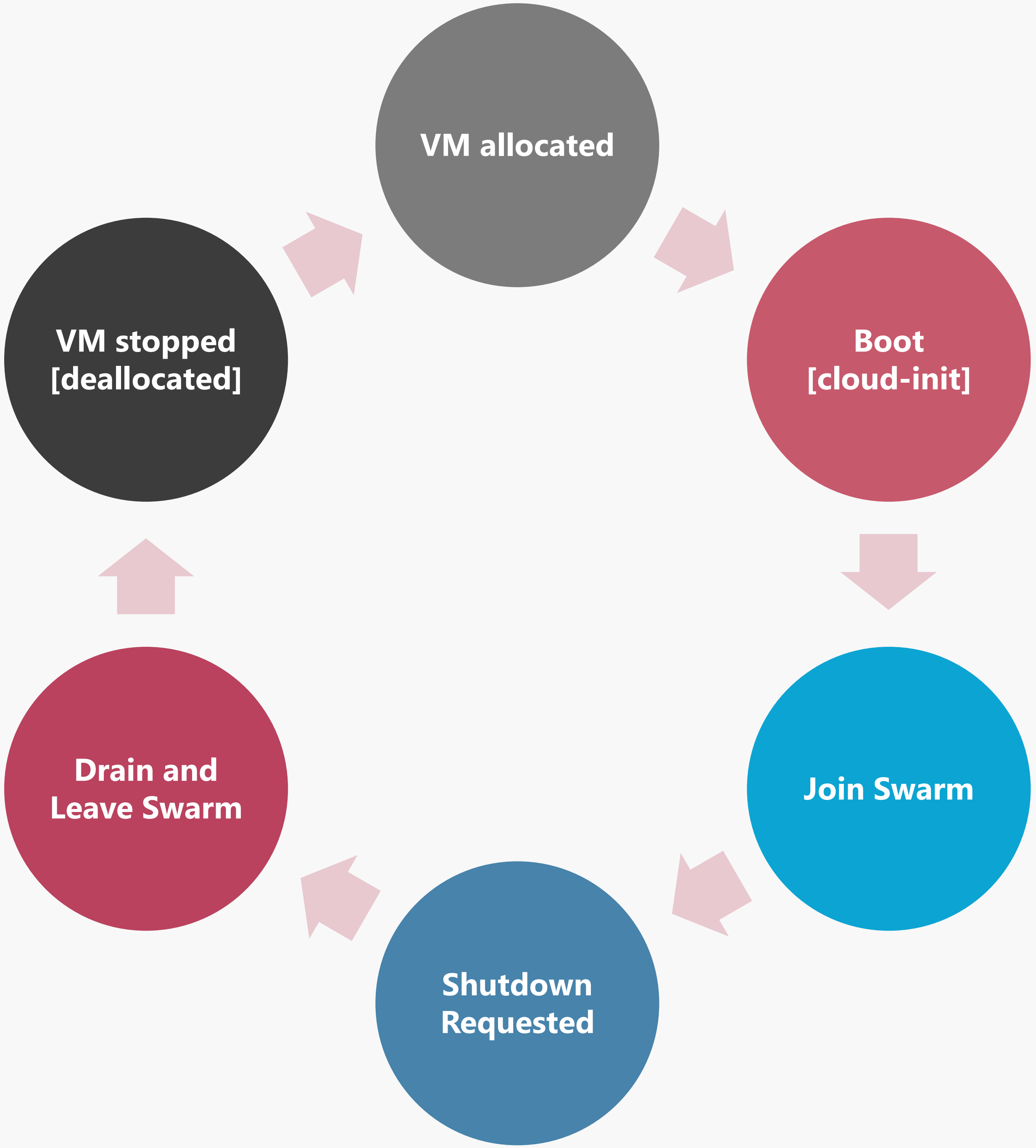
Poor Man's Cluster Automation

When an agent starts, it asks a custom server for the swarm token
When it shuts down, it signals the server to drain the containers



Agent Lifecycle

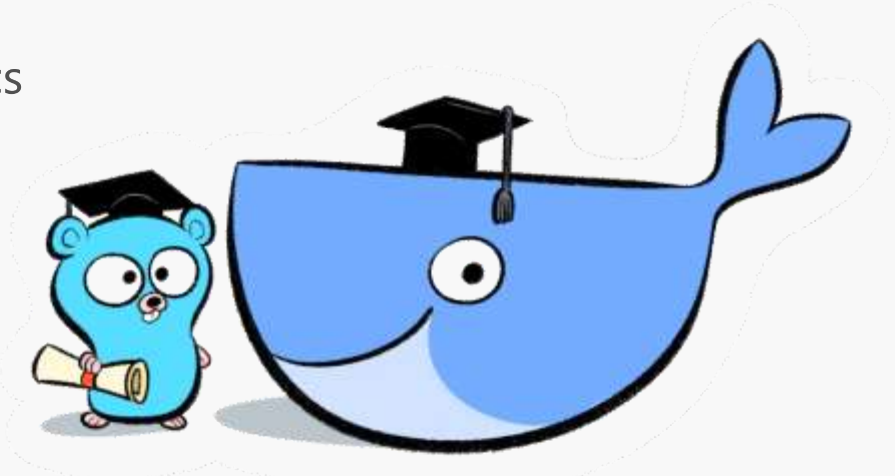




Resources

- https://github.com/rcarmo/azure-docker-swarm-cluster
    - ARM template
    - cloud-init scripts
    - Deployment quickstart
- https://github.com/rcarmo/azure-toolbox
    - The Linux environment used in the demo
    - Complete Azure toolbox in a Docker container

Docker artwork by @laurelcomics



Now that class is over…

Thank you!

Check out upcoming masterclasses at
http://aka.ms/jwsmasterclasses