Is Docker Infrastructure or Platform? & Cloud Foundry intro

A Lecture for InstallFest 2017

by
Ing. Tomáš Vondra
Cloud Architect at
HOME AT CLOUD

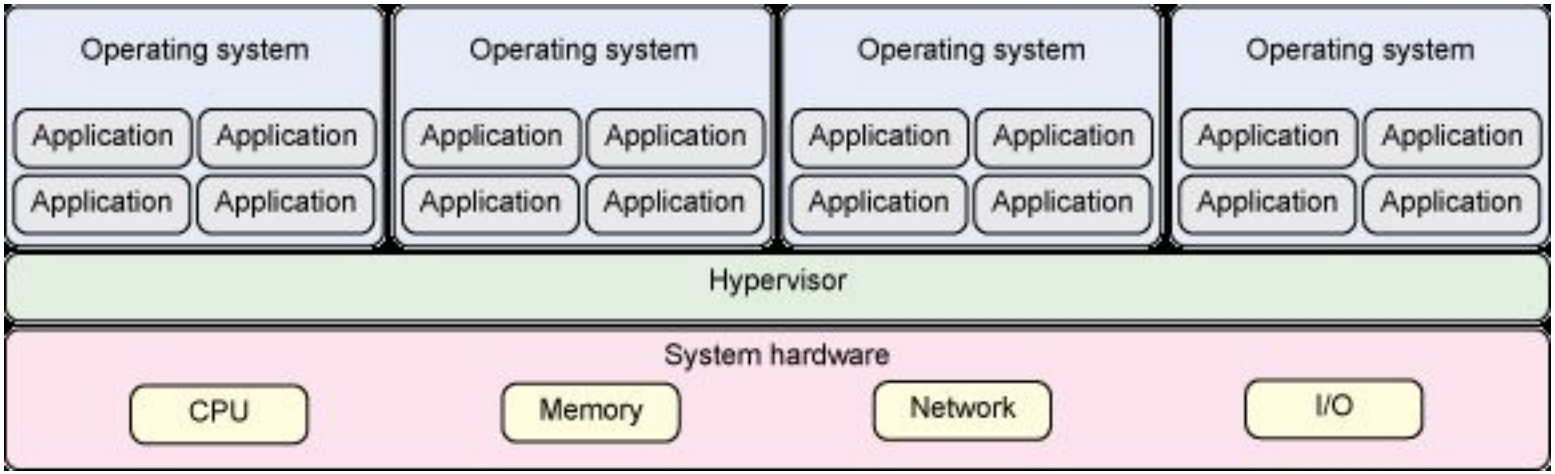
Outline

- Virtualization and IaaS
- PaaS
- Docker
- Problems with Docker
- Cloud Foundry
- Demo

Virtualization

- First used in 1969 by IBM
- On PC platform since 1999 (Vmware)
  - Useful to run an OS on another
- Server virtualization since 2001
  - Aims to increase utilization in datacenters

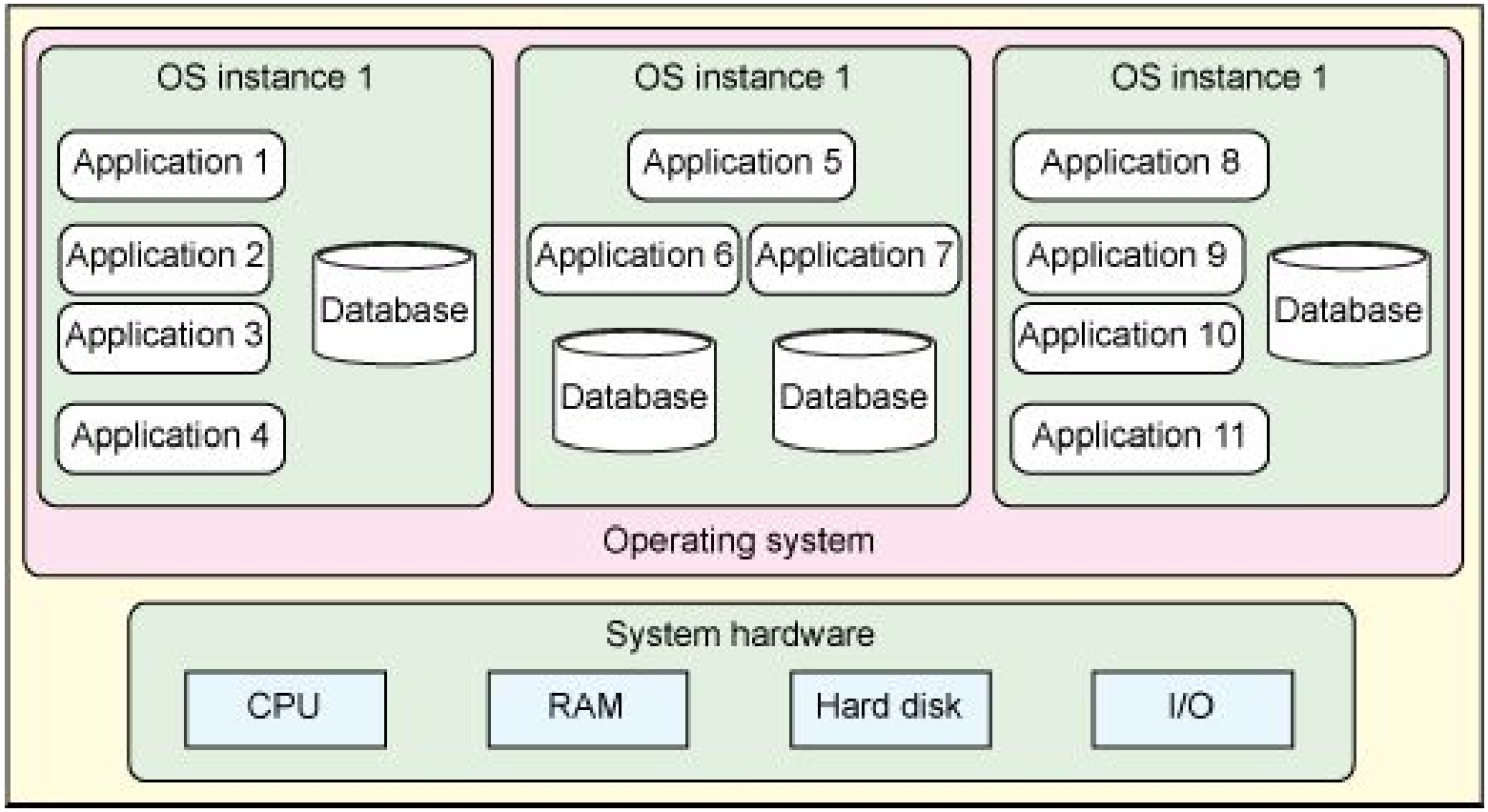
Hardware Virtualization



Virtualization

- OS level virtualization aka. Containers
  - Pros: no overhead at all, high memory efficiency
    - Shared libraries and caches
  - Cons: all guests share one kernel
    - Still possible to have different distributions
  - Uses kernel facilities for high separation of containers
    - namespaces for user IDs, processes, network sockets, filesystems
    - control groups for resource quotas
  - Parallels (commercial), OpenVZ (being phased out), LXC, Docker, runC, Rocket, nSpawn, Warden

Containers


OS instance 1
Application 1
Application 2
Application 3
Application 4
Database
OS instance 1
Application 5
Application 6
Application 7
Database
Database
OS instance 1
Application 8
Application 9
Application 10
Application 11
Database
Operating system
System hardware
CPU
RAM
Hard disk
I/O

Virtualization

- Advantages of server virtualization
  - Increased utilization
  - Power savings
  - Separation of applications
  - Higher flexibility
  - Fast server deployment
  - Load balancing
  - Error resilience

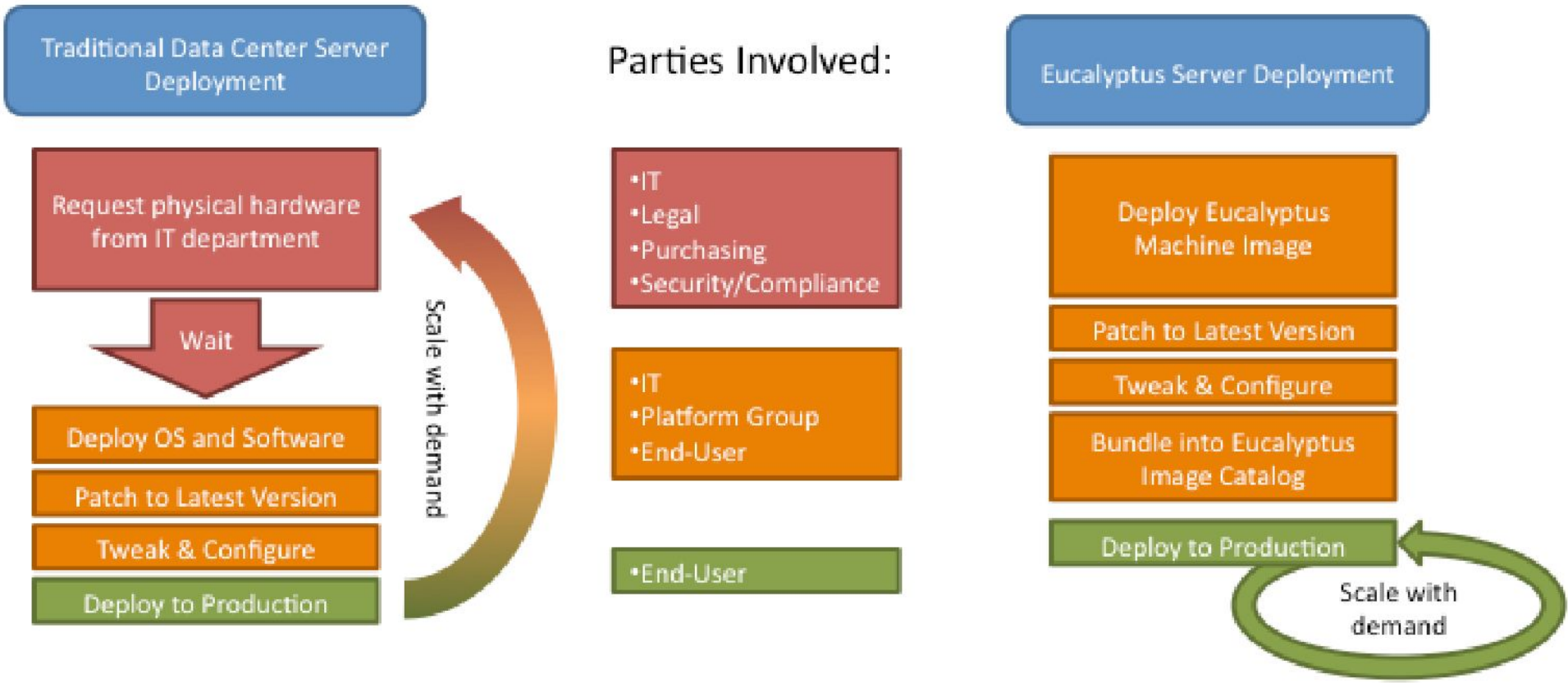
Infrastructure as a Service

- An upgrade to virtualization
- First layer of Cloud Computing
  - > general cloud properties
  - Automation
  - Elasticity
  - Self-service and web services
  - Pay per use
- Private, public and hybrid

Infrastructure as a Service

- What's a service? Computing power.
  - Rationed in units of VM Instances
    - An instance has fixed CPU and RAM
    - There may be pre-defined types or user-configurable
    - Can't modify when running -> horizontal scaling
- Storage
  - File storage
  - Volumes / Virtual disks (on central storage)
- Network connectivity (In/Out, between VMs)
- Usage of some APIs (autoscaling, monitoring)

Scaling process in private IaaS



Webhosting

- Provider does all hardware and software administration
- Service usually includes domain registration and e-mail
- Limits usable programming languages
  - Most have PHP and ASP/.NET, some Perl and Python, very few Java and Ruby
- Changes to the environment only through the provider's service personnel

Webhosting

- Three types
  - Free – mostly without scripting or with ads
  - Shared – good for low traffic sites
  - No information about how many sites on one server
    - Hostings are compared only by latency
  - Multitenancy security measures mostly minimal
  - Managed
    - eq. Server rental with administration
    - Terms can be arranged quite individually

Platform as a Service

- Similar to webhosting in concept
  - Used mostly to run web applications
- Second layer of Cloud Computing
  - > general cloud properties
    - Automation
    - Elasticity
    - Self-service and web services
    - Pay per use

Platform as a Service

- Similarities to webhosting
  - Takes care of software platform administration
  - Limits available programming languages
    - Selection is different, with regard to scalability
    - mostly Ruby, Java, Python, PHP, Node.JS
    - Often includes services like SQL and noSQL databases, queue services, caches, etc.

Platform as a Service

- Two types of PaaS
  - on IaaS
    - Uses a layered approach
      - Depends on IaaS for multitenancy
        - » And for the servers themselves
    - Adds application deployment and scaling
  - Direct
    - Platform built from scratch, own hardware
    - May or may not contain virtualization
      - Must secure multitenancy somehow else
      - > using containers in recent versions

Platform as a Service

- Added value
  - Development tools
    - From a command-line tool to deploy apps
    - To a web dashboard with monitoring
    - Or even a click-up-your-own-app web IDE
  - Special services and APIs
    - To use platform features, databases, ..
  - Using platform specifics induces risk of vendor-lock in
    - Open-source platforms have several providers

Where to get PaaS

- Public
  - Google App Engine, Microsoft Azure, Amazon Elastic Beanstalk, SalesForce Heroku, AppFog, RedHat OpenShift, ActiveState Stackato, CloudBees, IBM BlueMix, Pivotal
- Private (few mature projects)
  - Pivotal Cloud Foundry, RedHat OpenShift, Tsuru
  - Wouldn’t waste time with the rest (Cloudify didn’t work in dipl. thesis)

DevOps

- Also known as Infrastructure as Code
  - Server configuration is scripted
- Fills the gap between developers and system administrators
- Repeatable processes that let you scale out quickly
  - Even if you start small, you write the scaling
- Examples (by age): CFEngine, Puppet, Chef, Ansible, SaltStack
  - Commercial: RightScale, Amazon OpsWorks

Docker

- Recently, container virtualization experienced a boom
- Docker platform took the lead in 2013
  - LXC has been here since 2008, OpenVZ 2005
- Why did it create a market disruption?
- Let’s have a look at its design:

The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center

Public Cloud

Disaster recovery

Production Servers

Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly?

docker

Results in N X N compatibility nightmare

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

docker

Cargo Transport Pre-1960

Multiplicity of Goods

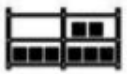
Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing

Can I transport quickly and smoothly (e.g. from boat to train to truck)

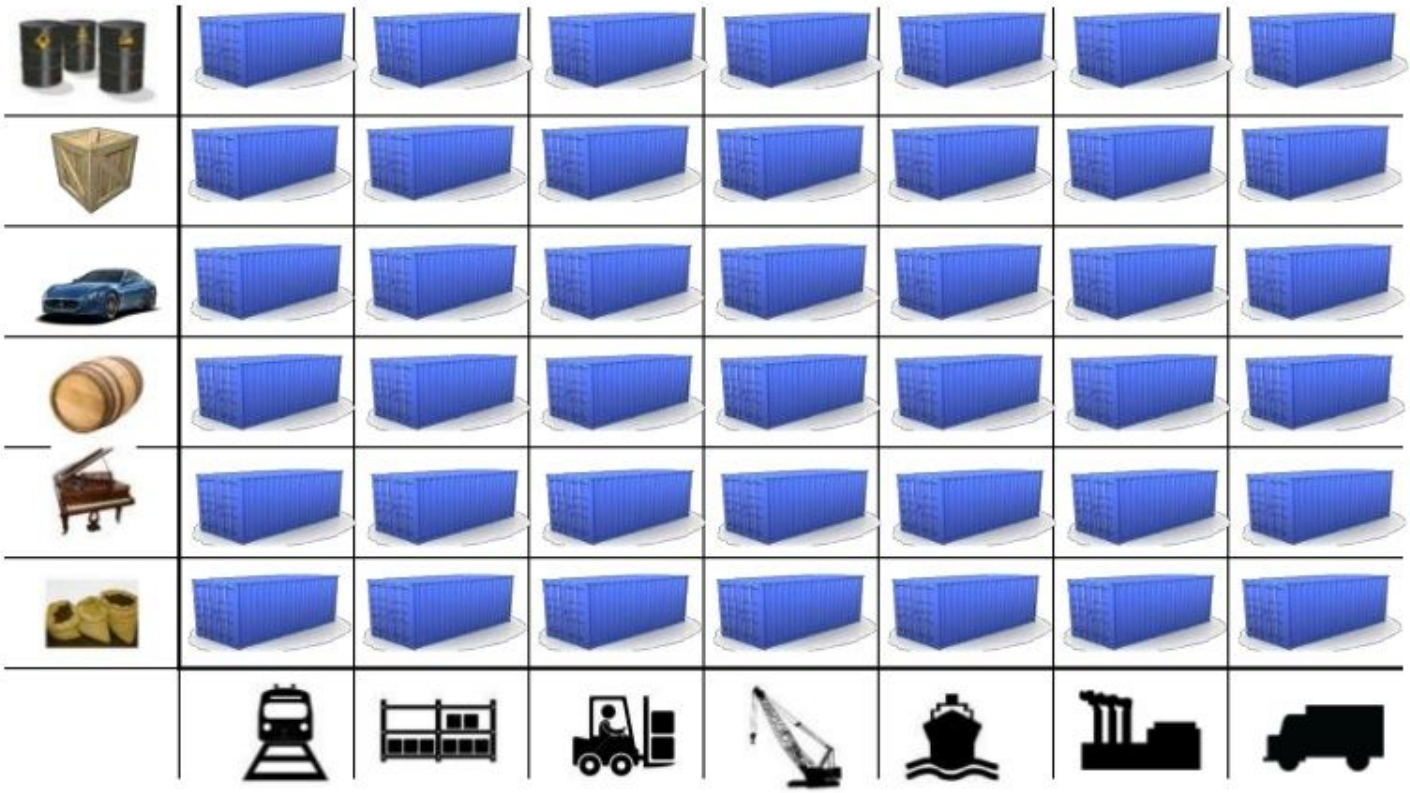
docker

Also an NxN Matrix

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

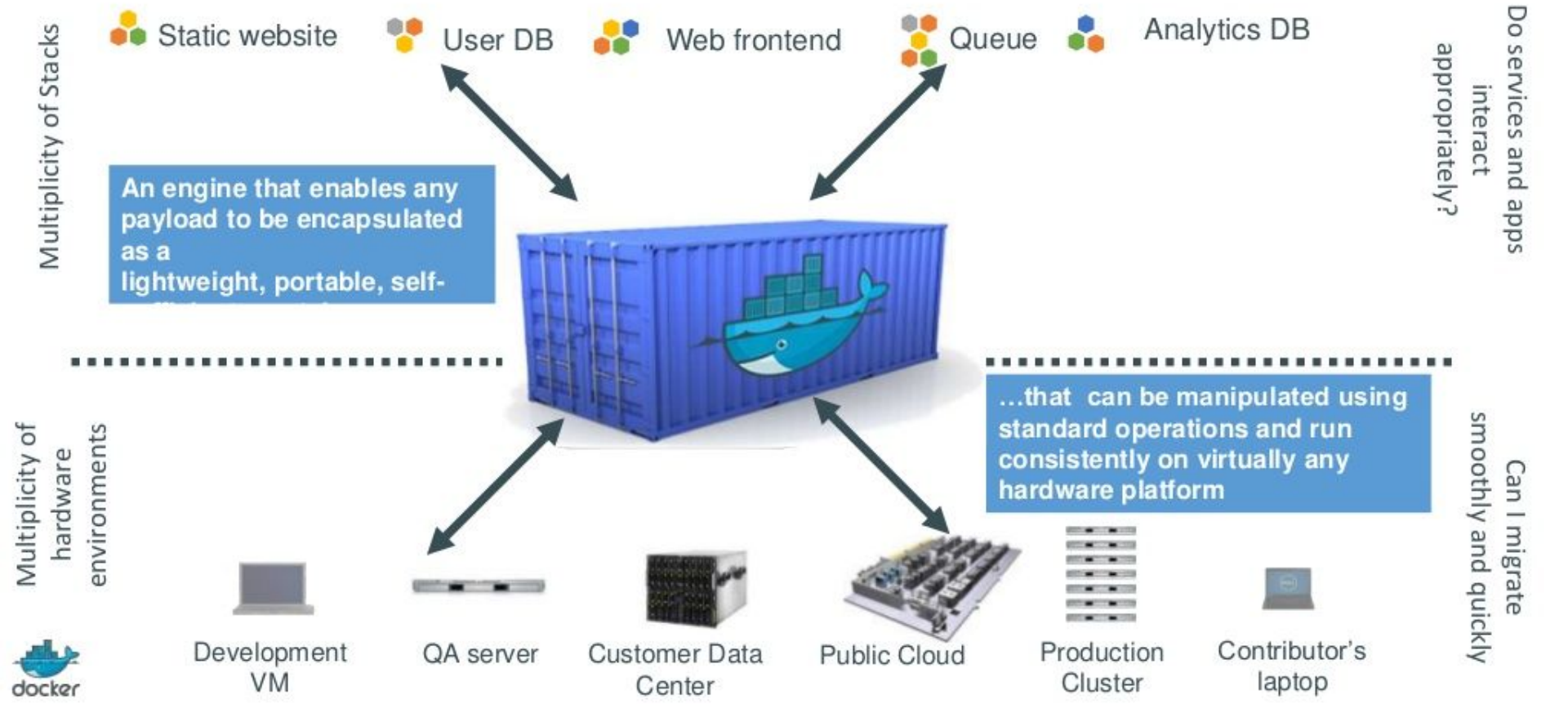
Solution: Intermodal Shipping Container

This eliminated the NXN problem…

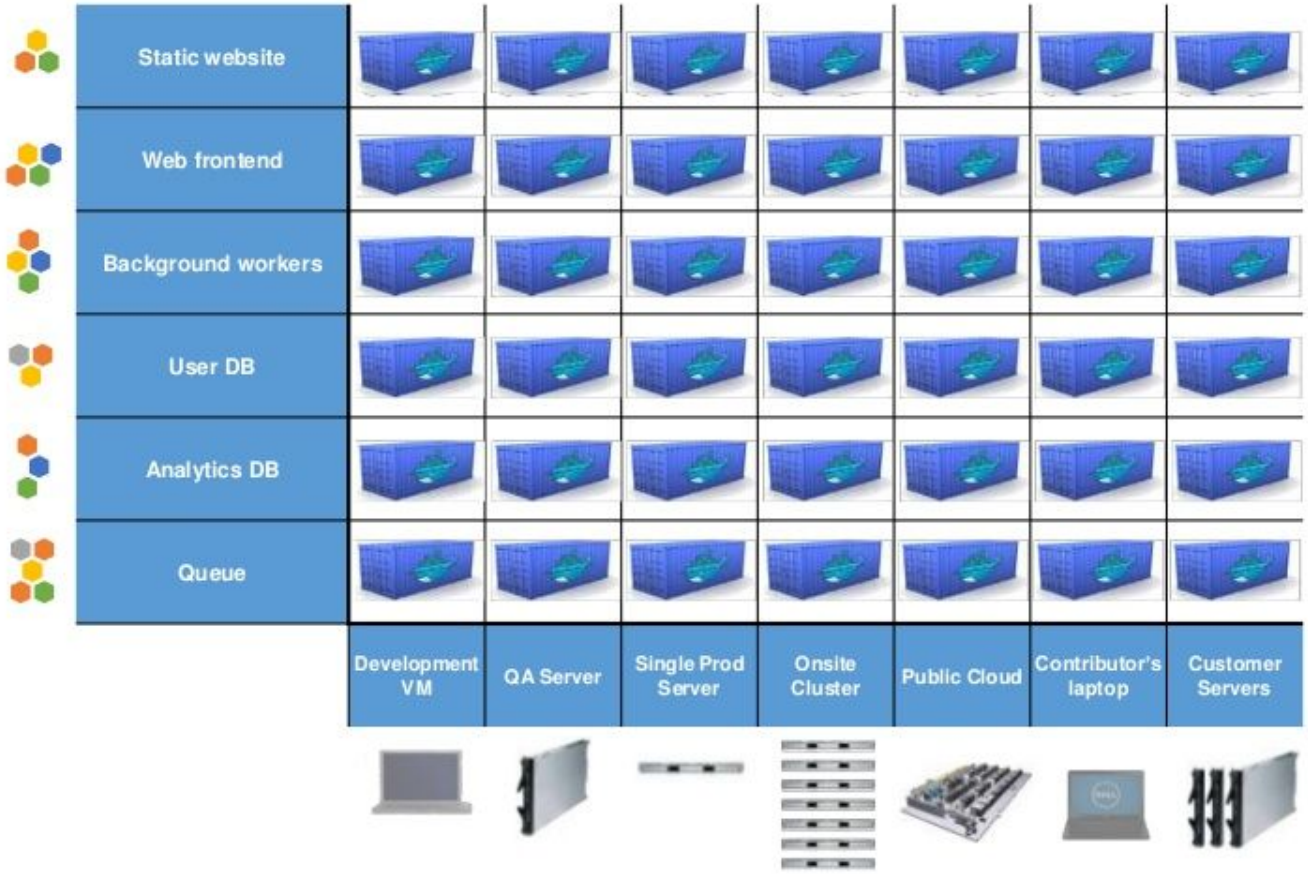


docker

Docker is a shipping container system for code



Docker solves the NXN problem



docker

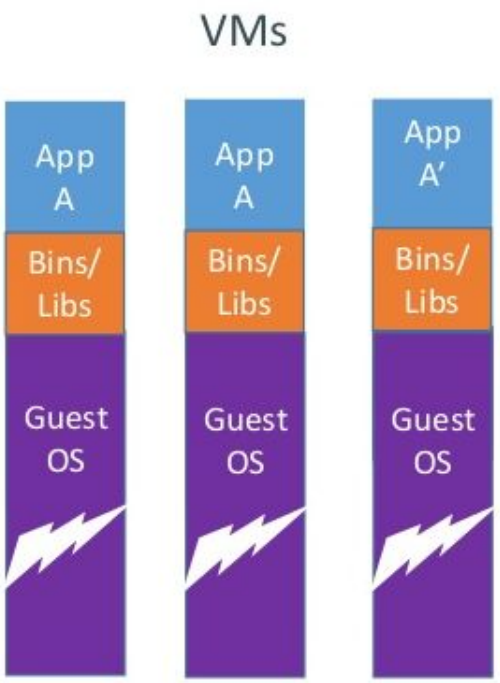
Docker: Build once, run everywhere

1. Prepare your development environment
2. Deploy it directly to production servers
   (no need to rebuild your app)

… this concept is known from Java

https://en.wikipedia.org/wiki/Write_once,_run_anywhere

Virtual Machines vs. Containers



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Original App

(No OS to take up space, resources, or require restart)

Copy of App

No OS. Can Share bins/libs

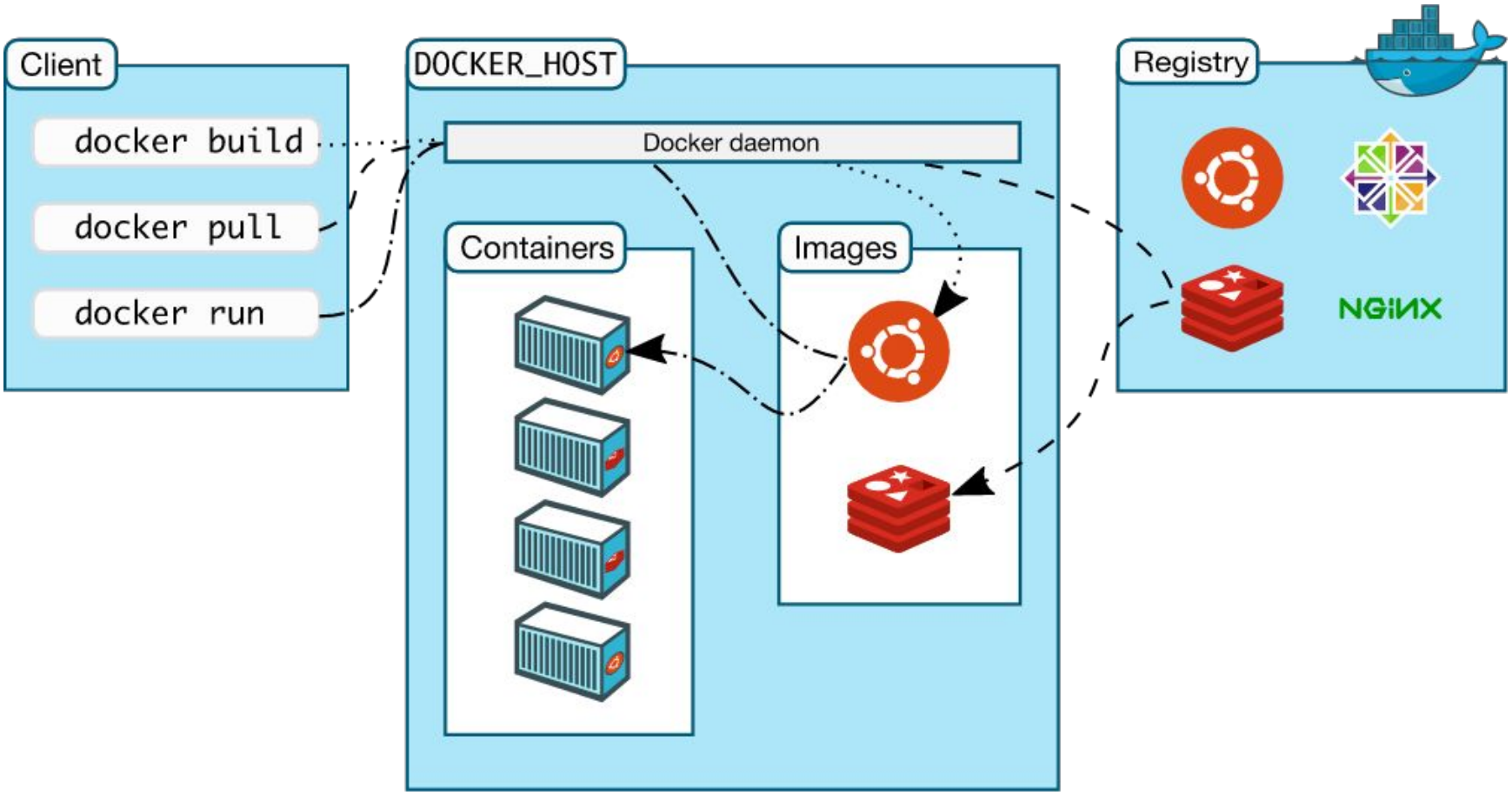
Modified App

Union file system allows us to only save the diffs Between container A and container A'

Docker layers in action

```
docker images --tree
Warning: '--tree' is deprecated, it will be removed soon. See usage.
└─511136ea3c5a Virtual Size: 0 B Tags: scratch:latest
  └─59e359cb35ef Virtual Size: 85.18 MB
    └─e8d37d9e3476 Virtual Size: 85.18 MB Tags: debian:wheezy
      └─c58b36b8f285 Virtual Size: 85.18 MB
        └─90ea6e05b074 Virtual Size: 118.6 MB
          └─5dc74cffc471 Virtual Size: 118.6 MB Tags: vim:latest
```

Docker's architecture



Source: https://docs.docker.com/engine/introduction/understanding-docker/

Docker Hub

Cloud-based registry service for building and shipping application or service containers.

- **Image Repositories**
- **Automated Builds**
- **Webhooks**

https://hub.docker.com/

Docker Summary

- Container platform
  - uses cgroups and namespaces through libcontainer
- Unique features
  - shipping format
  - layered structure
  - central repository of images
- Keywords
  - image
  - instance
  - volume
  - open port
- Examples: https://github.com/sameersbn

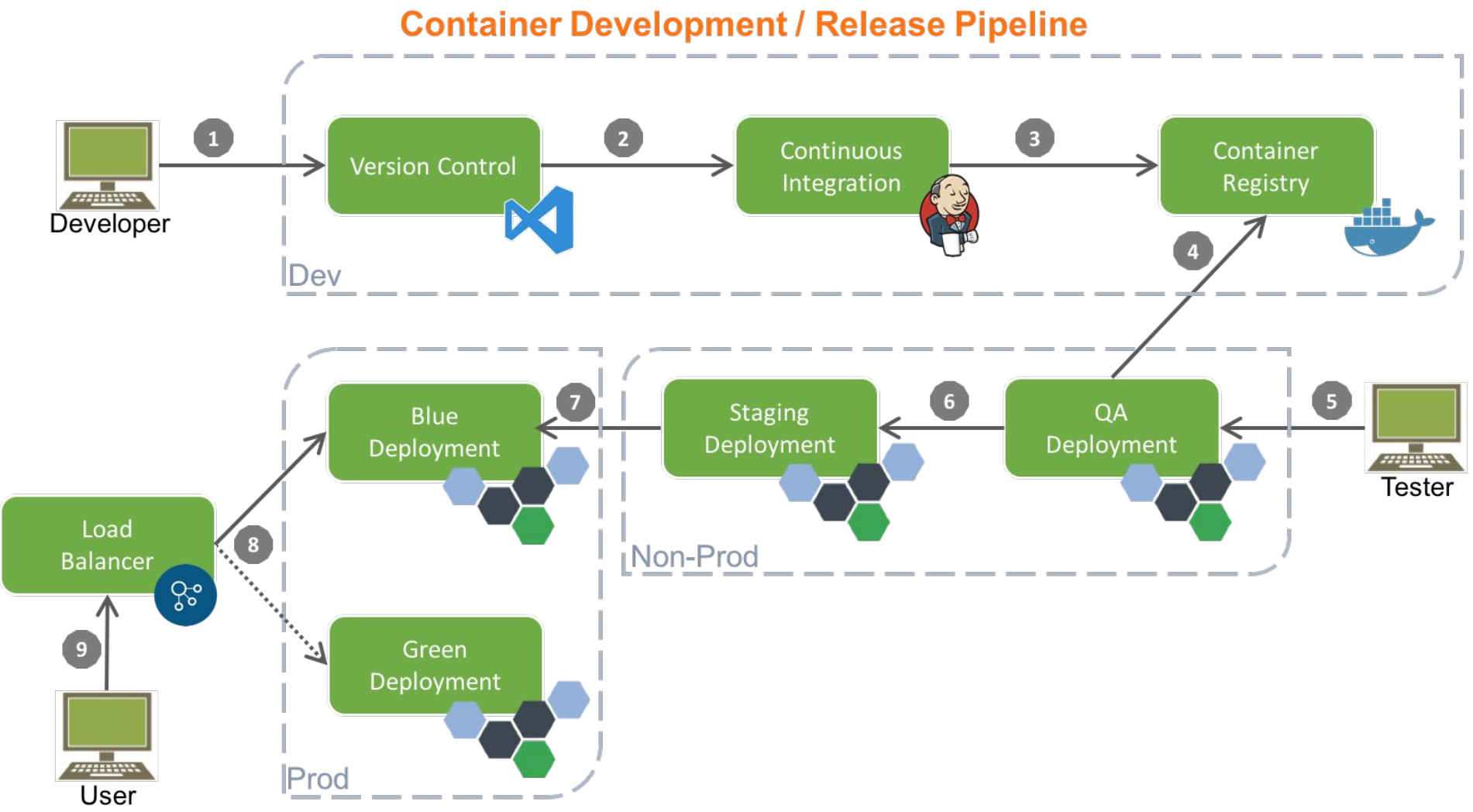
Docker critique

- We already have shipping formats
  - deb? rpm? OVF? tgz is inside OCI anyway.
- Why layers anyway?
  - Memory reduction not necessary - we have KSM
  - Driver trouble
    - overlays: incompatible kernel implementations
      - aufs -> overlayfs -> overlayfs2
    - btrfs: “too many references”, crashed fs with du
    - device-mapper thin provisioning: wastes space
- Central repository = a loaded gun
  - 2015 survey: Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities

The gap between Docker and PaaS

- CI for consistent building of images
- Image repository
- Network security
- Host OS patching
- Load Balancing and Scaling
- Databases and other persistence services
- Logging and monitoring
- Service discovery
- Orchestration of container relationships
- Application updates and redeployment

Ref.arch. according to Robert Greiner

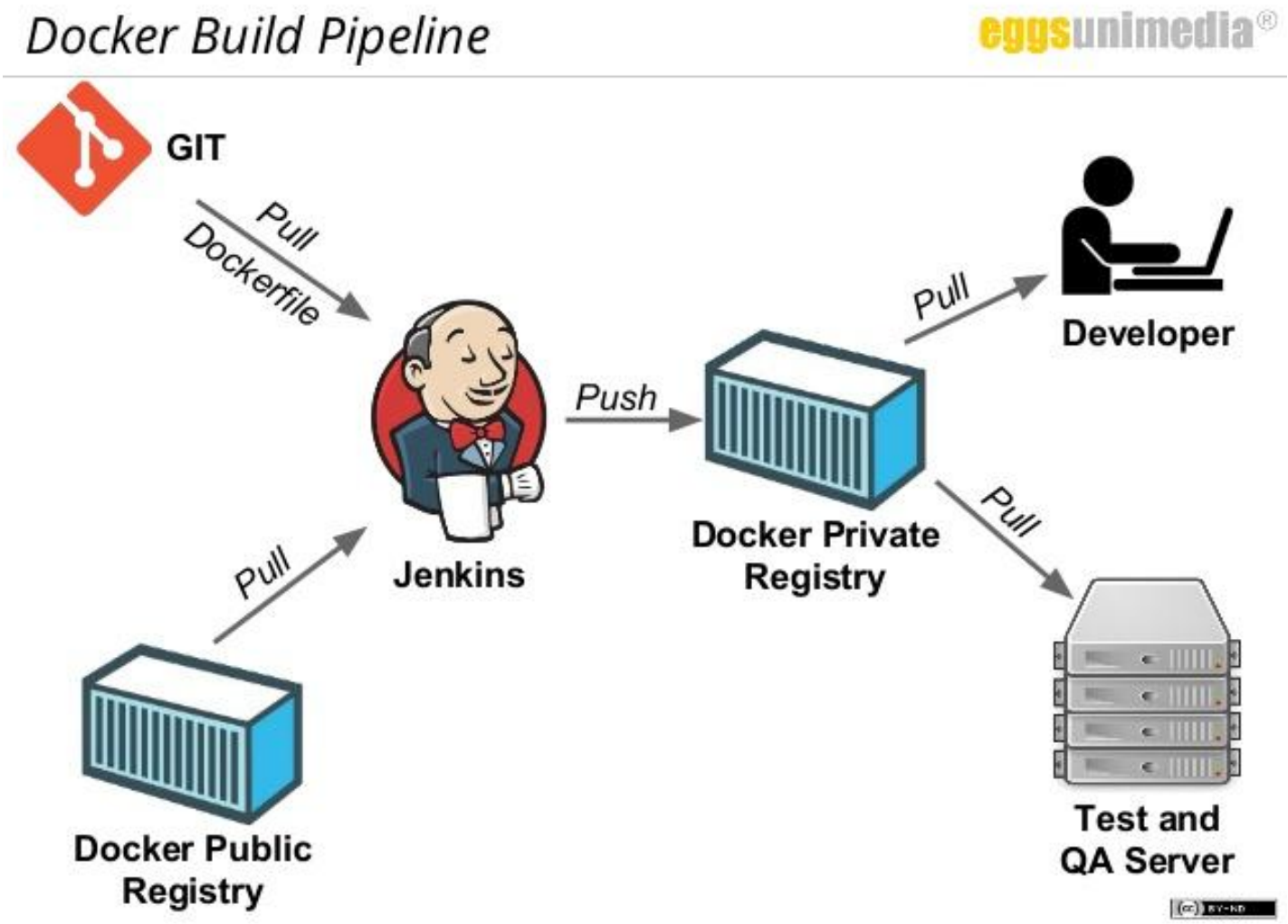


Sonatype

Link to Blog: Continuous Integration with Docker

http://bit.ly/2aeA1io

Ref.arch. according to eggs unimedia



Sonatype

Link to Presentation: Locally it worked! Virtualizing Docker

http://bit.ly/2au62ra

Cloud Foundry

- Container technology not related to Docker
  - “Warden” also uses cgroups and namespaces
- No layers and central repository
- Application is a first-class concept
  - the container is an implementation detail
  - built by language-specific buildpack at staging time
- Provides ready-made Services
  - MySQL, Postgres, Mongo, Redis, Riak, RabbitMQ
- Load balancing and scaling built in
- Can run Docker containers as well
  - volumes and TCP load balancers already available
  - virtual networking in the making

CLOUD FOUNDRY

PLATFORM RUNTIME CAPABILITIES

PROGRAMMING LANGUAGES

CLIs & IDE INTEGRATIONS

ROUTING

CONTAINER MANAGEMENT

CONFIGURATION

SERVICE CATALOG

LOGGING & METRICS

MESSAGING

APPLICATION SERVICES

DATABASES

OBJECT STORAGE

TESTING

CONTINUOUS INTEGRATION & DELIVERY

USER PROVIDED

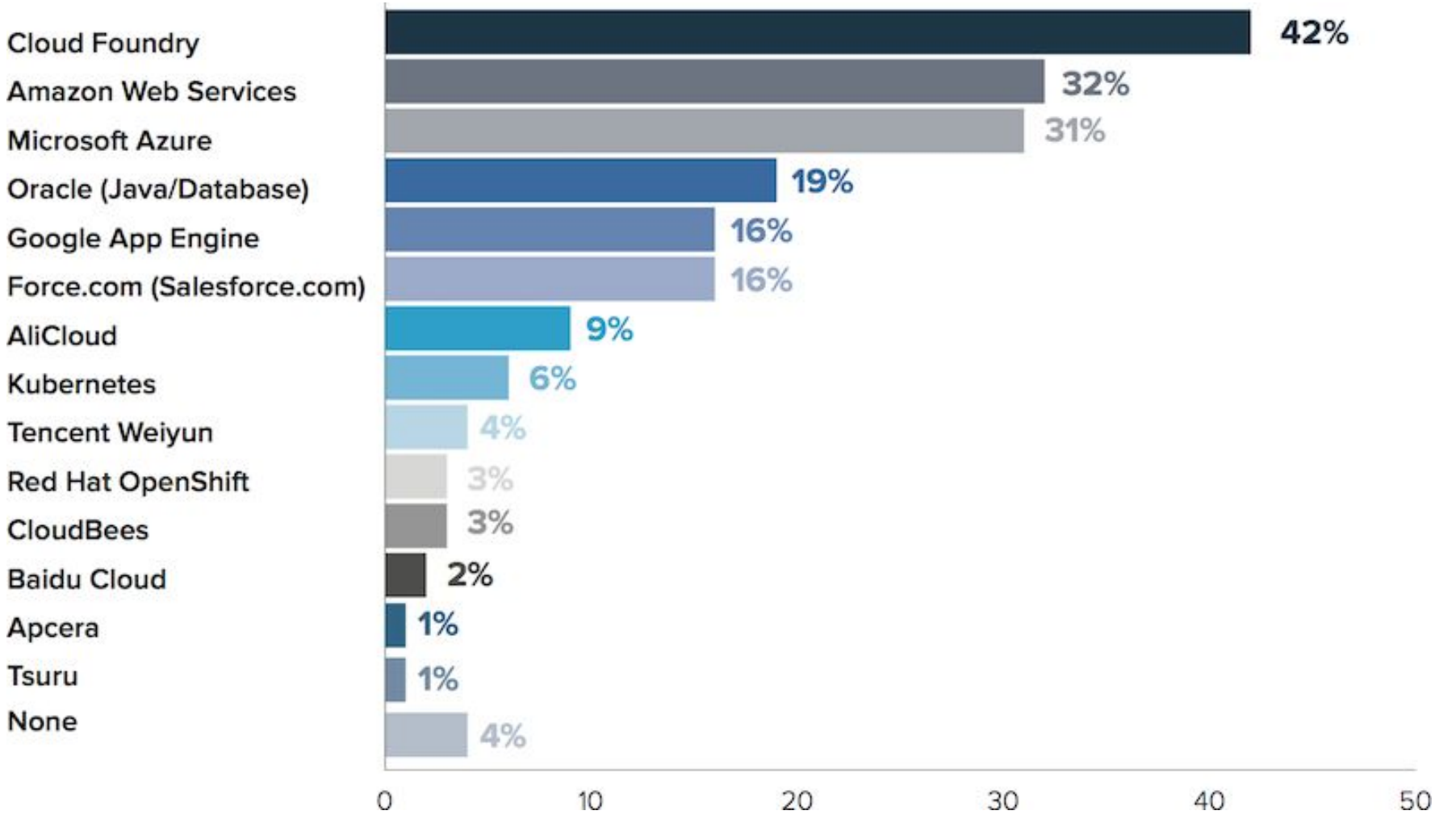
OPERATIONS

ZERO DOWNTIME | FAILURE & RECOVERY | SCALING | SECURITY & PATCHING | PLATFORM UPGRADES

INFRASTRUCTURE

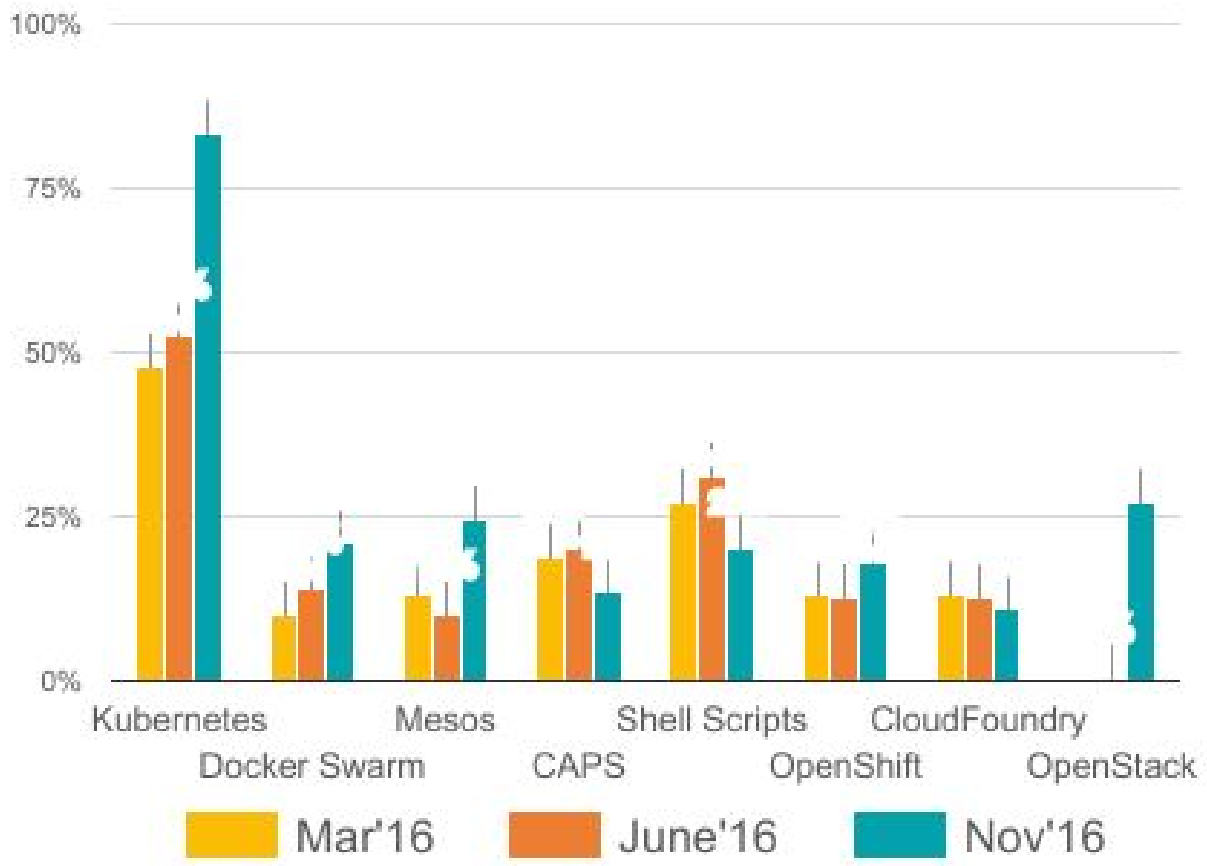
OPENSTACK | VMWARE | AWS | AZURE | GOOGLE | OPENSTACK | IBM | EMC BAREMETAL

Cloud Foundry market share



Cloud Foundry foundation

Cloud Foundry market share



Kurbernetes foundation

Cloud Foundry market share

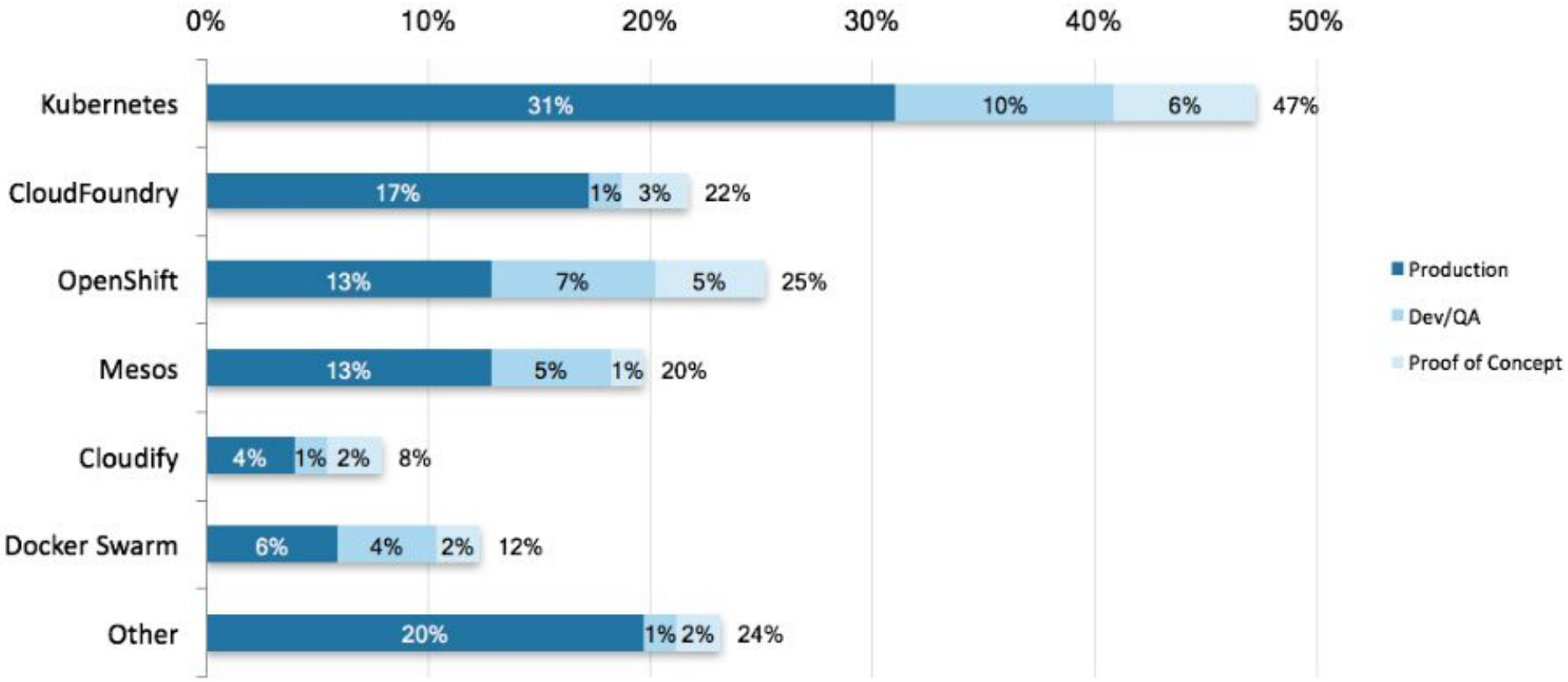


Figure 6.1 n=203 shows all of 2016

OpenStack foundation

History in comparison with Kubernetes

- CF is here since 2011
- Kubernetes 2014
- OpenShift also 2011, but was rewritten from scratch based on Kubernetes
- CF has a history of continual evolution
  - originally by VMware
  - 2013 transferred to daughter company Pivotal
  - 2014 Cloud Foundry Foundation established
    - open-source governance

All dates in this presentation are from Wikipedia

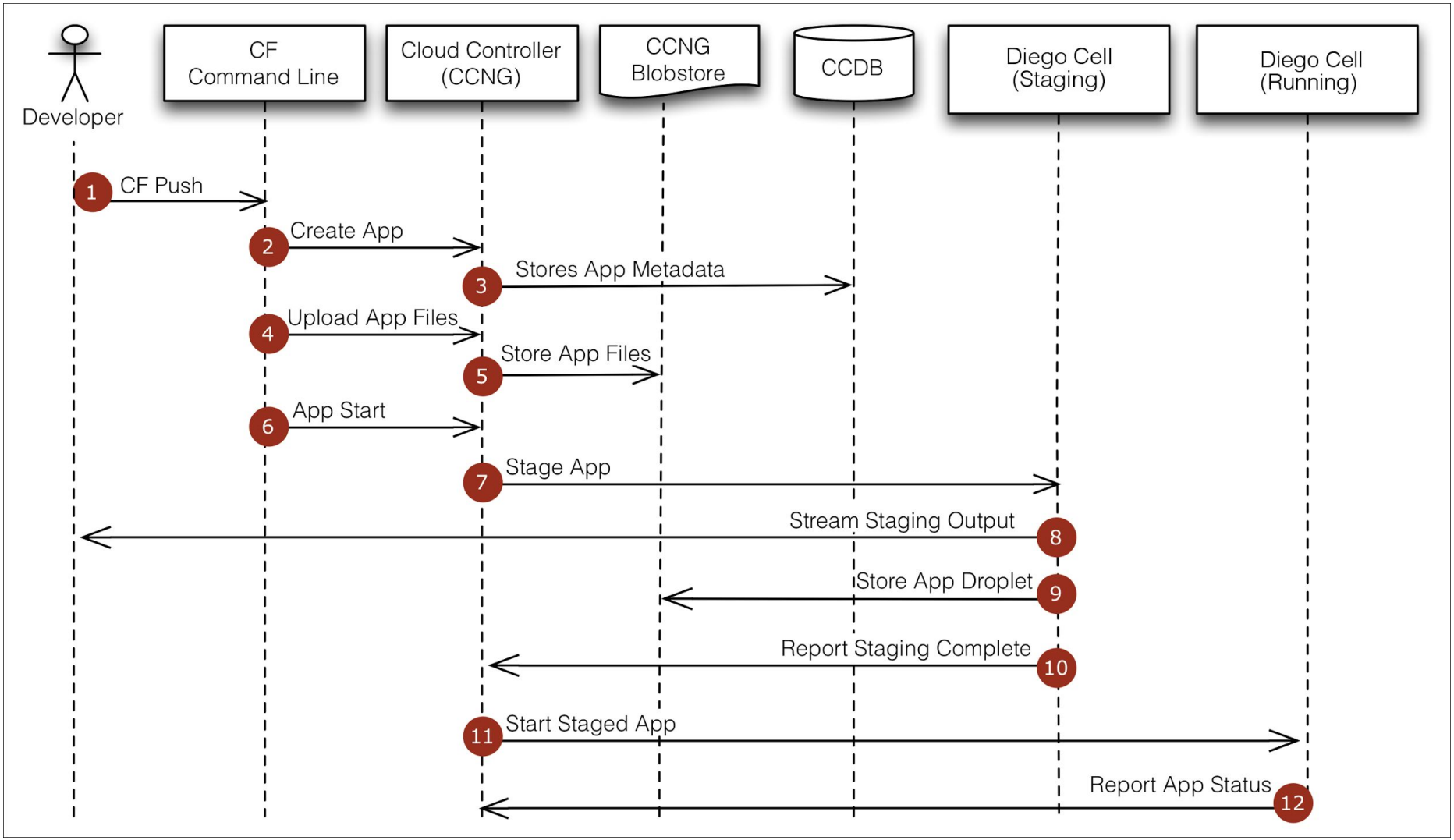
Application deployment

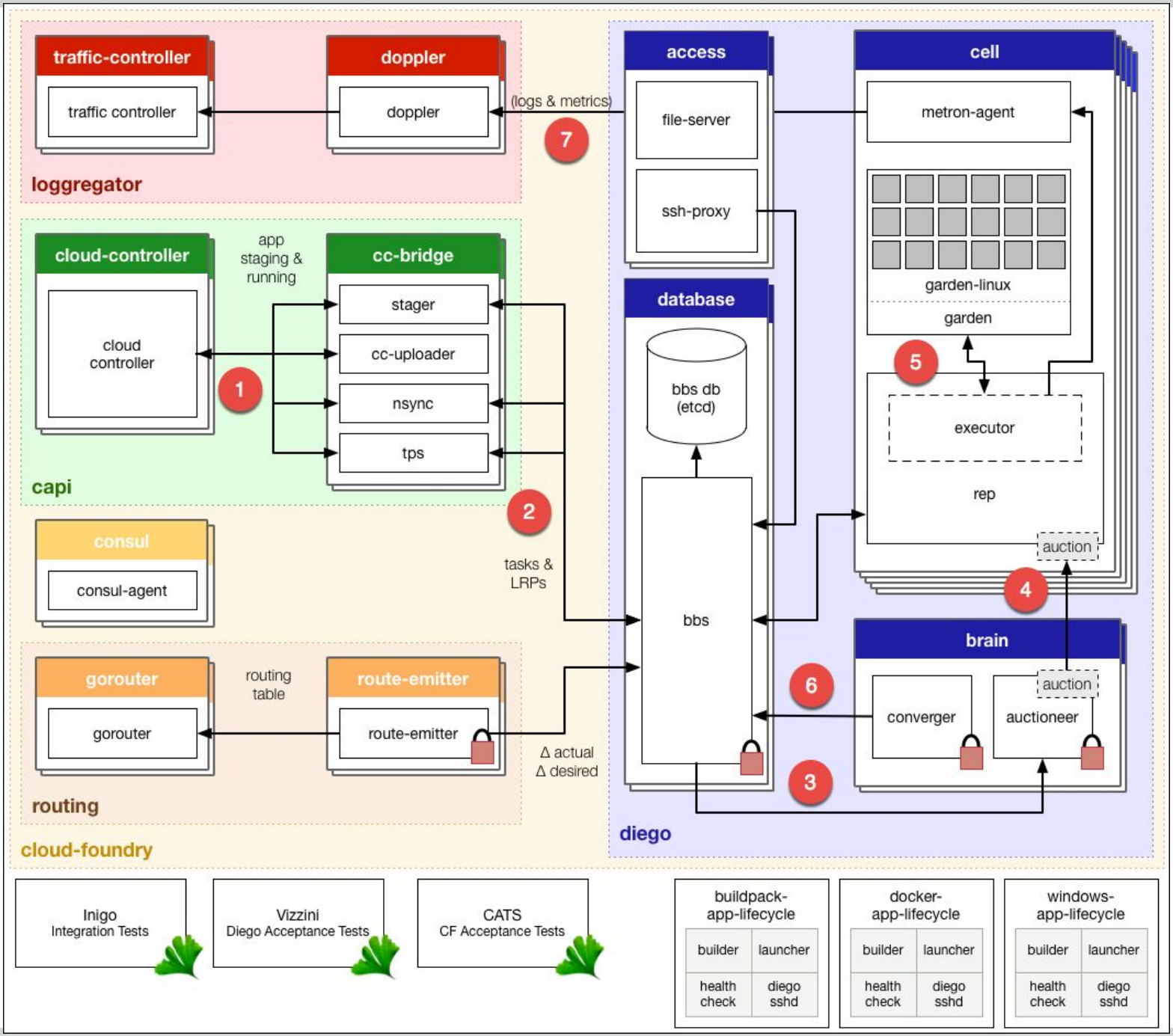
- Process starts with magic words “cf push”
  - Uploads and stores app files
  - Examines and stores app metadata
  - Buildpack runs and creates a “droplet” of the app
  - Selects an appropriate Diego cell
  - Starts the app
  - Optionally creates a route to the app
  - Optionally configures service connections

Stacks, Buildpacks, and the rest

- Stack is a base file system
  - “cflinuxfs2” is based on Ubuntu 14
- Buildpack packages the app and its dependencies
- Droplet is a container image
- Droplets are stored in the Blobstore
- Diego cell is the machine running containers
- Warden/Garden is the container technology

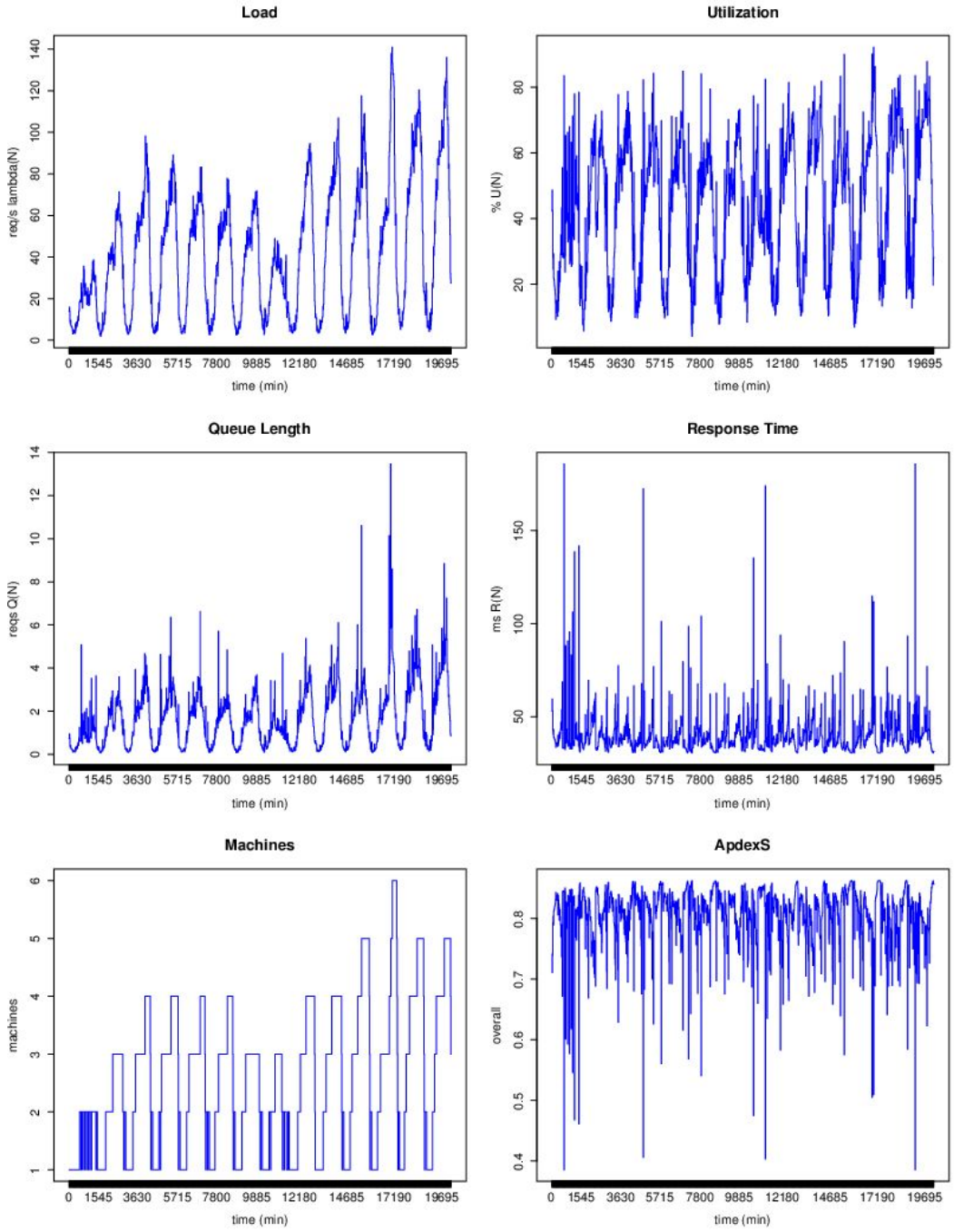
- If the standard buildpacks are not enough, you can write your own
- See what is already available in the community


Developer
CF Command Line
Cloud Controller (CCNG)
CCNG Blobstore
CCDB
Diego Cell (Staging)
Diego Cell (Running)
1 CF Push
2 Create App
3 Stores App Metadata
4 Upload App Files
5 Store App Files
6 App Start
7 Stage App
8 Stream Staging Output
9 Store App Droplet
10 Report Staging Complete
11 Start Staged App
12 Report App Status


traffic-controller
traffic controller
doppler
doppler
(logs & metrics)
7
loggregator
access
file-server
ssh-proxy
cell
metron-agent
garden-linux
garden
cloud-controller
cloud controller
app staging & running
cc-bridge
stager
cc-uploader
nsync
tps
1
capi
database
bbs db (etcd)
5
executor
rep
auction
2
tasks & LRPs
consul
consul-agent
bbs
4
brain
converger
auctioneer
auction
6
gorouter
gorouter
routing table
route-emitter
route-emitter
Δ actual
Δ desired
routing
3
diego
cloud-foundry
Inigo
Integration Tests
Vizzini
Diego Acceptance Tests
CATS
CF Acceptance Tests
buildpack-app-lifecycle
builder
launcher
health check
diego sshd
docker-app-lifecycle
builder
launcher
health check
diego sshd
windows-app-lifecycle
builder
launcher
health check
diego sshd

Monitoring and Scaling

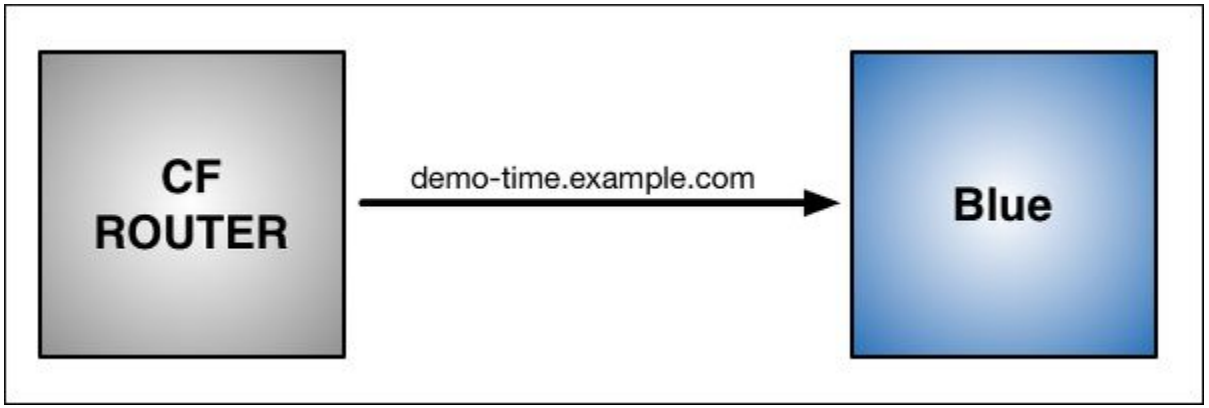
- Open-source version provides APIs to
  - see current CPU, memory and disk usage
  - scale the number of instances horizontally
  - scale the application resource quota vertically
    - restarts the app
- Our version will have autoscaling
  - metrics from API stored in Influxdb
  - user specifies scaling rules
    - like CPU over 70% for 5 minutes
  - autoscaler engine horizontally scales the app
  - all integrated in the Home at Cloud portal


Load
req/s lambda(N)
0 20 40 60 80 100 120 140
0 1545 3630 5715 7800 9885 12180 14685 17190 19695
time (min)
Utilization
% U(N)
20 40 60 80
0 1545 3630 5715 7800 9885 12180 14685 17190 19695
time (min)
Queue Length
reqs Q(N)
0 2 4 6 8 10 12 14
0 1545 3630 5715 7800 9885 12180 14685 17190 19695
time (min)
Response Time
ms R(N)
50 100 150
0 1545 3630 5715 7800 9885 12180 14685 17190 19695
time (min)
Machines
machines
1 2 3 4 5 6
0 1545 3630 5715 7800 9885 12180 14685 17190 19695
time (min)
ApdexS
overall
0.4 0.5 0.6 0.7 0.8
0 1545 3630 5715 7800 9885 12180 14685 17190 19695
time (min)

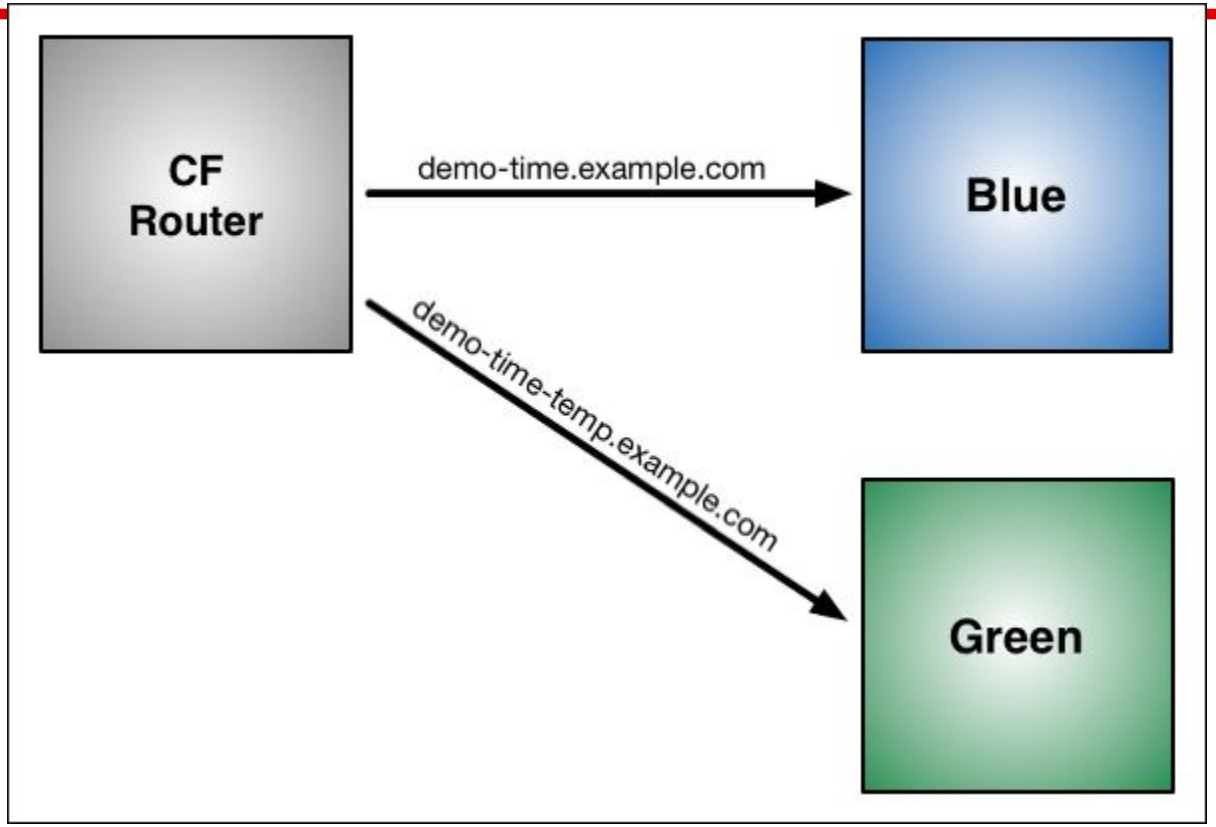
Routing

- Done by CF component gorouter
- Multiple gorouters behind HAProxy
- Can do
  - Shared domain
  - Bring-your-own-domain
  - Domain with path
  - Multiple routes to one app
  - One route to multiple apps
- Recently added component tcprouter

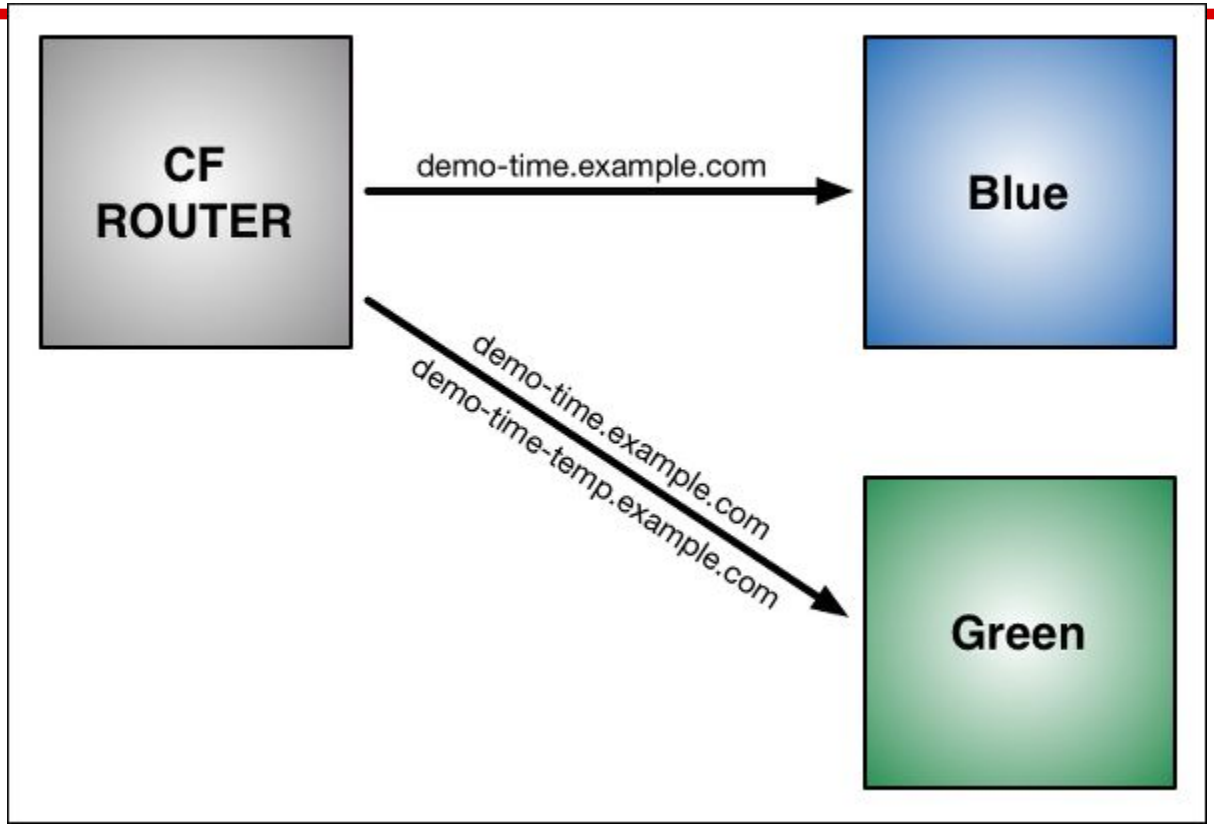
Blue-green deployment



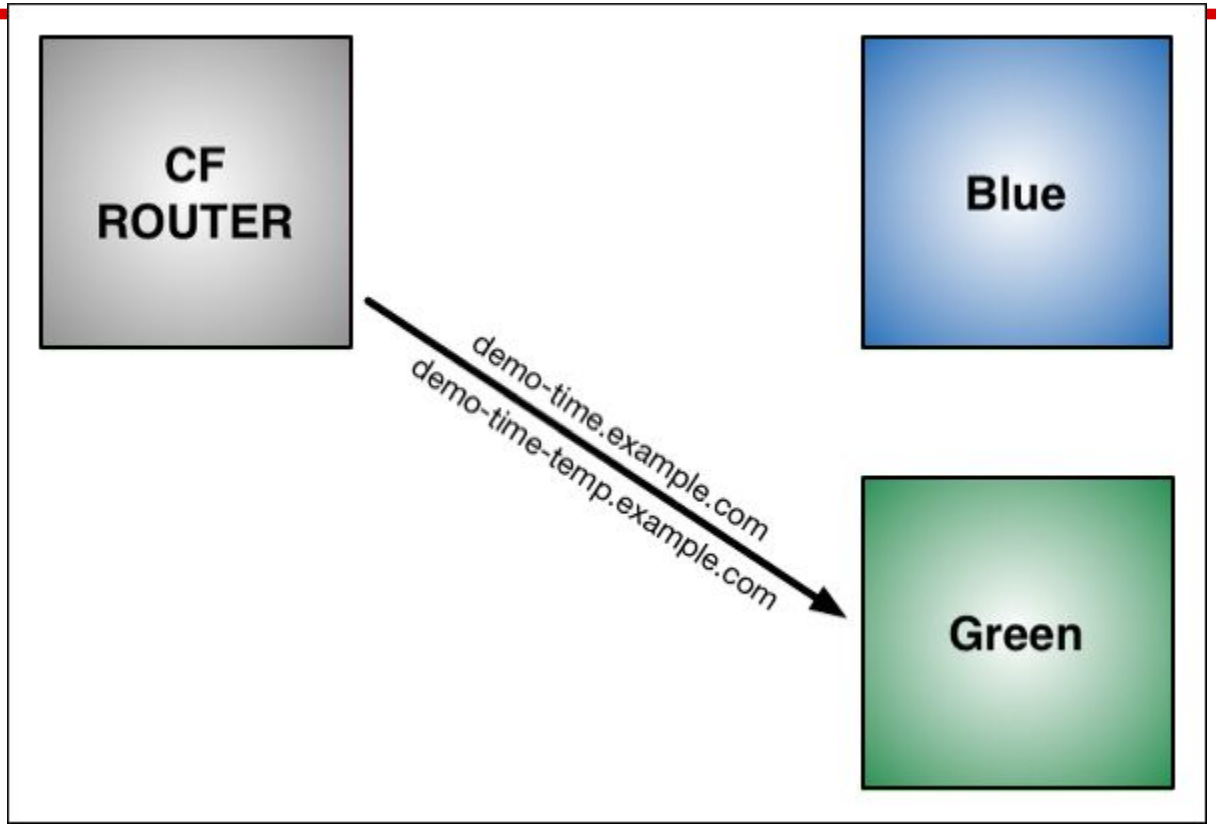
Blue-green deployment



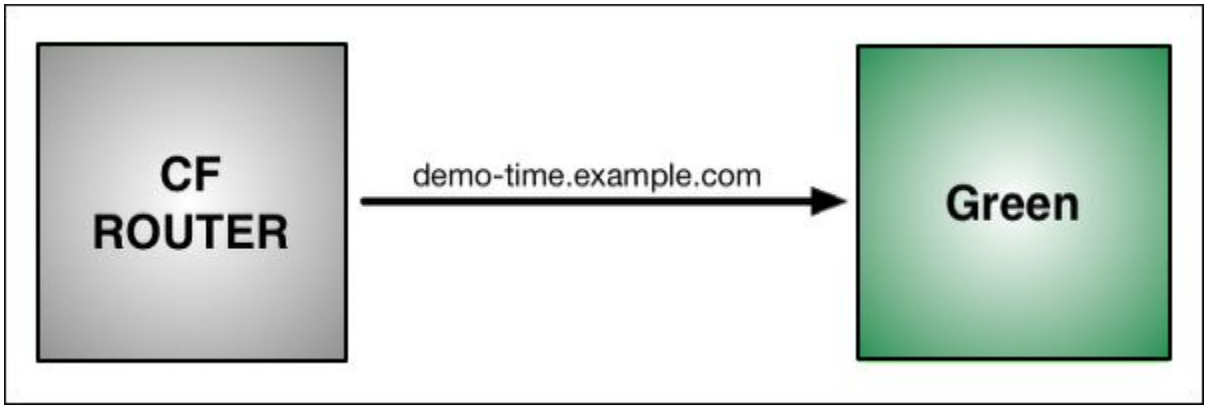
Blue-green deployment



Blue-green deployment



Blue-green deployment



Lyve demo

1. Get the CLI tool “cf” from Github
2. #
```
cf login -a http://api.cftest.homeatcloud.cz -u user -p pass --skip-ssl-validation
```
3. Try and see what you have:

```
cf help # All commands
cf apps # Deployed apps
cf marketplace # Available CF services
cf services # Deployed service instances
cf logs --recent spring-music # logs
cf app spring-music # info
cf ssh spring-music # ssh
```

How did I get the service?

- Create the MongoDB service

```
cf create-service MongoDB standard <instance_name>
cf bind-service <app_name> <instance_name>
```

- The App receives this JSON ENV variable:

```
VCAP_SERVICES=
{  "mongodb": [    {
      "name": "db-for-spring-music",
      "label": "mongodb",
      "tags": [
        "mongodb"
      ],
      "plan": "standard",
      "credentials": {
        "uri":
"mongodb://mongo_username:mongo_pass@192.168.3.12:27017,192.168.3.11:27017,1
92.168.3.10:27017/dbname"
      }    }  ],}
```

And the app?

- Official CF demo app in Java
  - You probably need to have a JDK in your $PATH

```
git clone https://github.com/cloudfoundry-samples/spring-music.git
cd spring-music/
./gradlew assemble

cf push

cf bind-service spring-music <service_instance_name>
cf restart spring-music

# if you see timeouts, they’re due to insufficient entropy on the hosting VM; try
cf push --health-check-type none
# or before restart/restage
cf set-health-check spring-music none
```

Why I could use just cf push

… without arguments?

The app has a manifest.yml file:

```
$ cat manifest.yml
---
applications:
- name: spring-music
  memory: 1G
  random-route: true
  path: build/libs/spring-music.jar
```

I already had a Docker image!

- CF runs those as well
  - Quite new, not as well tested
  - You should get the same ENV variable with service info when your Entrypoint is called
- Only works with images in Dockerhub
  - ..or another public registry, not local uploads (yet?)
- You may try some examples:

```
cf push test-app -o cloudfoundry/test-app
#or
cf push lattice-app -o cloudfoundry/lattice-app
```

Don’t try this at home

- Actually, you can. See microBOSH
- Our beta deployment on OpenStack
- Including admin station and ELK, uses
  - 51 VMs, 65 vCPU, 82 GB RAM, 885 GB local and 1,4 TB persistent storage
- Open Core means a lot of work
  - operations, services, monitoring, logging, accounting
  - autoscaling (bachelor’s thesis)
- Still missing to production
  - SSL, customer portal integration, billing
  - user testing

Questions?

If not:

Write to support@homeatcloud.cz

for beta access to Cloud Foundry at

Offer valid for 2 weeks.
End of beta program will be announced one month in advance.