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
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
  - \( \rightarrow \) 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

Parties Involved:
- IT
- Legal
- Purchasing
- Security/Compliance
- IT
- Platform Group
- End-User

Traditional Data Center Server Deployment:
- Request physical hardware from IT department
- Wait
- Deploy OS and Software
- Patch to Latest Version
- Tweak & Configure
- Deploy to Production

Eucalyptus Server Deployment:
- Deploy Eucalyptus Machine Image
- Patch to Latest Version
- Tweak & Configure
- Bundle into Eucalyptus Image Catalog
- Deploy to Production

Scale with demand
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

Static website
- nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB
- postgresql + pgv8 + v8

Queue
- Redis + redis-sentinel

Analytics DB
- hadoop + hive + thrift + OpenJDK

Web frontend
- Ruby + Rails + sass + Unicorn

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

Development VM

QA server

Customer Data Center

Public Cloud

Disaster recovery

Production Servers

Production Cluster

Can I migrate smoothly and quickly?

Do services and apps interact appropriately?

Multiplicity of Stacks

Multiplicity of hardware environments
Results in N X N compatibility nightmare

|----------------|----------------|-----------|--------------------|----------------|--------------|----------------------|------------------|
Cargo Transport Pre-1960

- Multiplicity of Goods
- Multiplicity of methods for transporting/storing
- Do I worry about how goods interact (e.g., coffee beans next to spices)?
- Can I transport quickly and smoothly (e.g., from boat to train to truck)
Also an NxN Matrix

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Barrels" /></td>
<td><img src="image2.png" alt="Cargo Containers" /></td>
<td><img src="image3.png" alt="Car" /></td>
<td><img src="image4.png" alt="Barrel" /></td>
<td><img src="image5.png" alt="Piano" /></td>
<td><img src="image6.png" alt="Rocks" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image1.png" alt="Barrels" /></td>
<td><img src="image2.png" alt="Cargo Containers" /></td>
<td><img src="image3.png" alt="Car" /></td>
<td><img src="image4.png" alt="Barrel" /></td>
<td><img src="image5.png" alt="Piano" /></td>
<td><img src="image6.png" alt="Rocks" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image1.png" alt="Barrels" /></td>
<td><img src="image2.png" alt="Cargo Containers" /></td>
<td><img src="image3.png" alt="Car" /></td>
<td><img src="image4.png" alt="Barrel" /></td>
<td><img src="image5.png" alt="Piano" /></td>
<td><img src="image6.png" alt="Rocks" /></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image1.png" alt="Barrels" /></td>
<td><img src="image2.png" alt="Cargo Containers" /></td>
<td><img src="image3.png" alt="Car" /></td>
<td><img src="image4.png" alt="Barrel" /></td>
<td><img src="image5.png" alt="Piano" /></td>
<td><img src="image6.png" alt="Rocks" /></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[![Train](image7.png)](image7.png) [![Train](image7.png)](image7.png) [![Forklift](image8.png)](image8.png) [![Crane](image9.png)](image9.png) [![Ship](image10.png)](image10.png) [![Tank](image11.png)](image11.png) [![Truck](image12.png)](image12.png)
Solution: Intermodal Shipping Container

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another.
This eliminated the NXN problem...
Docker is a shipping container system for code

An engine that enables any payload to be encapsulated as a lightweight, portable, self-contained unit.

...that can be manipulated using standard operations and run consistently on virtually any hardware platform.

Multiplicity of Stacks

Do services and apps interact appropriately?

Can I migrate smoothly and quickly?

---

Static website
User DB
Web frontend
Queue
Analytics DB

Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor’s laptop
Docker solves the NXN problem
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.

Containers

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
                    └── 5dc74cffe471 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

Container Development / Release Pipeline

1. Version Control
2. Continuous Integration
3. Container Registry
4. 
5. Test Deployment
6. QA Deployment
7. Staging Deployment
8. Load Balancer
9. Prod Deployment

Link to Blog: Continuous Integration with Docker
Ref.arch. according to eggs unimedia

Docker Build Pipeline

Link to Presentation: Locally it worked! Virtualizing Docker
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 market share

- Cloud Foundry: 42%
- Amazon Web Services: 32%
- Microsoft Azure: 31%
- Oracle (Java/Database): 19%
- Google App Engine: 16%
- Force.com (Salesforce.com): 16%
- AliCloud: 9%
- Kubernetes: 6%
- Tencent Weiyun: 4%
- Red Hat OpenShift: 3%
- CloudBees: 3%
- Baidu Cloud: 2%
- Apcera: 1%
- Tsuru: 1%
- None: 4%
Cloud Foundry market share
Cloud Foundry market share

- Kubernetes: 31% (Production), 10% (Dev/QA), 6% (Proof of Concept)
- CloudFoundry: 17% (Production), 1% (Dev/QA), 3% (Proof of Concept)
- OpenShift: 13% (Production), 7% (Dev/QA), 5% (Proof of Concept)
- Mesos: 13% (Production), 5% (Dev/QA), 1% (Proof of Concept)
- Cloudify: 4% (Production), 1% (Dev/QA), 2% (Proof of Concept)
- Docker Swarm: 6% (Production), 4% (Dev/QA), 2% (Proof of Concept)
- Other: 20% (Production), 1% (Dev/QA), 2% (Proof of Concept)

Figure 6.1 n=203 shows all of 2016
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
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
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

demo-time.example.com

CF ROUTER

Blue
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:**

  ```json
  VCAP_SERVICES=
  {   "mongodb": [     {       "name": "db-for-spring-music",
       "label": "mongodb",
       "tags": [        "mongodb"
       ],
       "plan": "standard",
       "credentials": {        "uri": "mongodb://mongo_username:mongo_pass@192.168.3.12:27017,192.168.3.11:27017,192.168.3.10:27017/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.