Cloud Deployment: Infrastructure Automation

Infrastructure-as-Code, Containerization

Jürgen Cito
@citostyle
University of Zurich, Switzerland
The Days before Infrastructure Automation
Manual Provisioning for Deployment
“Old School” Server Configuration and Deployment

Order hardware to set up (virtual) servers

Manual execution of steps to achieve server state

Patches are applied to the running server

Error prone, Slow / Time consuming

Requires heavy process (in large organizations)

Further Reading: https://speakerdeck.com/garethr/infrastructure-as-code
Knowledge about existing infrastructure in manual provisioning is usually not documented in any formal manner:

> In operator’s head
(internal knowledge, worst case)

> Documentation in internal wikis
(textual representation, still not great)
Developing Software for the Cloud
(Quick Recap)

Scalability / API-driven Infrastructure at Scale

> New resources (cloud instances) can be spawned at scale over an API

Ephemeral Infrastructure / Cloud Instance Volatility

> Cloud Instances terminate frequently (scale down, faults)

The Need for Infrastructure Automation for Server Provisioning in the Cloud

Scalability / API-driven Infrastructure at Scale
Not feasible to manually configure every new instance

Ephemeral Infrastructure / Cloud Instance Volatility
Manual changes on cloud instances get lost

Infrastructure **provisioning** and application **deployment** in the cloud largely **automated**

Servers are **not** seen as **durable entities**

Any **changes** in infrastructure are **defined in code**

”What happens in our infrastructure is a lot more **obvious**. Everything we do on that level [infrastructure] is **over code** (...) So, I don’t need ask my colleague what he did to get that process running - I just look at the code and maybe the commit history”

Immutable Infrastructure

Existing instances that have already been deployed are never modified but rather replaced with a new (tested) updated instance.

Manual Provisioning

Further Reading: https://martinfowler.com/bliki/ImmutableServer.html
Immutable Infrastructure

Existing instances that have already been deployed are never modified but rather replaced with a new (tested) updated instance

Further Reading: https://martinfowler.com/bliki/ImmutableServer.html
Immutable Infrastructure: Data

If instances are ephemeral, so is the data stored on the instance.

Cloud services generally offer mountable storage and database services.

Further Reading: https://martinfowler.com/bliki/ImmutableServer.html
Infrastructure as Code

Declarative (or sometimes imperative) definition of infrastructure in program code files to manage configuration and automate provisioning.

Sometimes also called “programmable infrastructure”

Allows to leverage software engineering best practices

“Infrastructure as Code (IaC) is the process of managing and provisioning computing infrastructure (processes, bare-metal servers, virtual servers, etc.) and their configuration through machine-processable definition files, rather than physical hardware configuration or the use of interactive configuration tools.”

Infrastructure as Code
Different Abstractions, Different Tools (for IaaS)

Cloud Environment Orchestration:
- Openstack Heat
- CloudFormation
- Terraform
- ...

Creates networks, instances, storage, etc.

Configuration Management:
- Chef
- Puppet
- Ansible
- ...

Provisions an existing server instance
Infrastructure as Code
Example: Chef

```ruby
service "tomcat" do
  service_name "tomcat6"
  case node["platform"]
    when "centos","redhat","fedora"
      supports :restart => true, :status => true
    when "debian","ubuntu"
      supports :restart => true, :reload => true, :status => true
  end
  action [:enable, :start]
end
```
Infrastructure as Code
Example: Puppet

```yaml
# execute 'apt-get update'
extec { 'apt-update':
    command => '/usr/bin/apt-get update'
}

# install apache2 package
package { 'apache2':
    require => Exec['apt-update'],
    ensure  => installed,
}

# install mysql-server package
package { 'mysql-server':
    require => Exec['apt-update'],
    ensure  => installed,
}
```
Infrastructure as Code
Goal: Reach certain system state through executed actions

Example Tasks

<table>
<thead>
<tr>
<th>#</th>
<th>Task</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Install MySQL</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>Set MySQL password</td>
<td>$p_2 = \text{root password}$</td>
</tr>
<tr>
<td>a3</td>
<td>Install Apache &amp; PHP</td>
<td>$p_3 = \text{operating system distribution (e.g., 'debian')}$</td>
</tr>
<tr>
<td>a4</td>
<td>Deploy Application</td>
<td>$p_4 = \text{application context path (e.g., '/myapp')}$</td>
</tr>
</tbody>
</table>

State Graph

Infrastructure as Code
Software Engineering Best Practices

Version Control
> Track infrastructure evolution

Testing & Continuous Integration
> Validation of infrastructure with unit- and integration tests

Code Review
> Infrastructure code inspection reduces potential of errors in configuration

Reuse
> Existing definitions of infrastructure can be used in different contexts
Infrastructure as Code
Testing: Linter (Example: Puppet Lint)

“Puppet Lint will test modules and manifests against the recommended Puppet style guidelines”

$ puppet-lint /etc/puppet/modules
foo/manifests/bar.pp - ERROR: trailing whitespace found on line 1
apache/manifests/server.pp - WARNING: variable not enclosed in {} on line 56...

http://puppet-lint.com/
Infrastructure as Code
Testing: Unit Tests (Example: rspec-puppet)

Assert expectations of actions defined in infrastructure code file — by testing resulting resources

```ruby
it do
  is_expected.to contain_user('luke').only_with(
    'ensure' => 'present',
    'uid'    => '501'
  )
end
...

it { is_expected.to contain_service('keystone').without(
  [{'restart', 'status'}]
)}
```
Infrastructure as Code
Testing: Unit Tests (Example: serverspec)

Assert expectations of actions defined in infrastructure code file — by testing server’s actual state

```ruby
describe service('apache2') do
  it { should be_enabled }
  it { should be_running }
end

describe port(80) do
  it { should be_listening }
end
```
Different Approaches to Infrastructure Automation

“Baking” Images
> Amazon Machine Images (AMI), Packer/Vagrant
> Speeds up provisioning

PaaS Deployment
> Previously packaged in specific format. Has to adhere to guidelines, less overall freedom (Buildpacks in Heroku, CloudFoundry)
> Container Cloud Deployment on the rise
Containers:
Mix of packaging and means for infrastructure automation
Docker Containers

What is Docker?

Docker allows you to package an application with all of its dependencies into a standardized unit for software development.

Containers consist of everything that enables software to run:
> Code
> Runtime
> System Tools
> System Libraries

* https://www.docker.com/what-docker
Technical Overview / Virtual Machines vs Containers

“Lightweight” VM

> Container is an isolated process ("chroot on steroids")
> Own process space
> Own network interface
> Feels like a VM
> Share kernel with the host
> Isolation through cgroups/namespaces

https://www.docker.com/what-is-docker
https://blog.docker.com/2016/03/containers-are-not-vms/
Docker Engine

Centralized runtime environment for containers

Enables portability

Sole dependency for Docker

No Emulation layer (almost no performance impact)

https://www.docker.com/products/docker-engine
Benefits of Docker Containers

Fast instantiation (~1-3 seconds)

Almost native performance

Transparent build process

Smaller Images

Easy to build, share, and publish

* https://www.docker.com/what-docker
Local Docker Workflow

Dockerfile  Docker Image  Docker Container
Terminology

**Dockerfile**
Declarative definition of an environment for producing an image

**Docker Image**
Immutable artifact built from a Dockerfile, has one to many layers.

**Docker Container**
Execution environment - Instantiation/running version of an image (can be parameterized)

**Docker Registry**
Public or private repository that stores allows for distribution of images (Docker Hub - [https://hub.docker.com/](https://hub.docker.com/) or CoreOS Quay - [https://quay.io/](https://quay.io/))
Local Docker Workflow

- **Dockerfile**

  ```
  # Build redis from source
  # Make sure you have the redis source code checked out in
  # the same directory as this Dockerfile
  FROM ubuntu:12.04
  MAINTAINER dockerfiles http://dockerfiles.github.io
  RUN echo "deb http://archive.ubuntu.com/ ubuntu precise main universe" > /etc/apt/sources.list
  RUN apt-get update
  RUN apt-get upgrade -y
  RUN apt-get install -y gcc make g++ build-essential libc6-dev tcl wget
  RUN wget http://download.redis.io/redis-stable.tar.gz -O - | tar -zv
  # RUN tar -zvzf /redis/redis-stable.tar.gz
  RUN (cd /redis-stable && make)
  RUN (cd /redis-stable && make test)
  RUN mkdir -p /redis-data
  VOLUME ["/redis-data"]
  EXPOSE 6379
  ENTRYPOINT ["/redis-stable/src/redis-server"]
  CMD ["--dir", "/redis-data"]
  ```

- **Image**

- **Docker Image**

- **Docker Container**

- **build**

- **run**
Dockerfile

Definition of infrastructure and dependencies of a container through instructions

FROM ubuntu:12.04
MAINTAINER dockerfiles

RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y gcc make g++ build-essential libc6-dev tcl wget
RUN wget http://download.redis.io/redis-stable.tar.gz -O - | tar -xvz
RUN (cd /redis-stable && make)
RUN (cd /redis-stable && make test)
COPY redis.conf /var/www/redis.conf
RUN mkdir -p /redis-data
VOLUME ["/redis-data"]
EXPOSE 6379
ENTRYPOINT ["/redis-stable/src/redis-server"]
CMD ["--dir", "/redis-data"]
Data Volumes

A specially-designated directory within one or more containers that bypasses the Union File System*

Volumes allow you to manage data within containers

> Mount a host directory (dependency to the host filesystem)
> Mount a data volume container (dependency to another container)
> Mount a shared-storage volume (NFS, iSCSI, etc.)

* [https://docs.docker.com/engine/userguide/containers/dockervolumes/](https://docs.docker.com/engine/userguide/containers/dockervolumes/)
Local Docker Workflow

Dockerfile

FROM ubuntu:12.04
MAINTAINER dockerfiles http://dockerfiles.github.io
RUN echo "deb http://archive.ubuntu.com/ ubuntprecise main universe" > /etc/apt/sources.list
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y gcc make g++ build-essential libc6-dev tcl wget
RUN wget http://download.redis.io/redis-stable.tar.gz -O - | tar -xvz
RUN (cd /redis-stable && make)
RUN (cd /redis-stable && make test)
RUN mkdir -p /redis-data
VOLUME ["/redis-data"]
EXPOSE 6379
ENTRYPOINT ["/redis-stable/src/redis-server"]
CMD ["--dir", "/redis-data"]

Docker Image

Docker Container

build ➔ run
Dockerfile —> Image

Definition of infrastructure and dependencies of a container through instructions

docker build -t <imagename> .

Build Context
containing all local dependencies and Dockerfile
Docker Images

# docker images

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>CREATED</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mhart/alpine-node</td>
<td>latest</td>
<td>2a15d8568f75</td>
<td>1 week ago</td>
<td>36.76 MB</td>
</tr>
<tr>
<td>hakyll</td>
<td>latest</td>
<td>d575da1e730c</td>
<td>2 weeks ago</td>
<td>1.487 GB</td>
</tr>
<tr>
<td>redis</td>
<td>alpine</td>
<td>50405530a7e5</td>
<td>4 weeks ago</td>
<td>15.95 MB</td>
</tr>
</tbody>
</table>

# docker rmi hakyll

Untagged: hakyll:latest
Deleted: sha256:3240943c9ea3f72db51...
Deleted: sha256:a3aeefaee0d4b8f61...
Deleted: sha256:16a7ebd378002f1261...

Lists all previously built images

Removes image ‘hakyll’ and all its layers from disk
Local Docker Workflow

# Build redis from source
# Make sure you have the redis source code checked out in
# the same directory as this Dockerfile

FROM ubuntu:12.04
MAINTAINER dockerfiles http://dockerfiles.github.io

RUN echo "deb http://archive.ubuntu.com/
  ubuntu precise main universe" > /etc/apt/
sources.list
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y gcc make g++ build-
essential libc6-dev tcl wget
RUN wget http://download.redis.io/redis-
stable.tar.gz -O - | tar -zv
RUN (cd /redis-stable && make)
RUN (cd /redis-stable && make test)
RUN mkdir -p /redis-data
VOLUME ["/redis-data"]
EXPOSE 6379
ENTRYPOINT ["/redis-stable/src/redis-
server"]
CMD ["--dir", "/redis-data"]
docker run -d --name <containernname> -p 80:5000 <imagename>

Give the container a unique name

Run container in the background (d for daemon)

Port mapping
First the exposed port (80)
Second the port within the container (5000)

Many more possibilities to run containers, see full reference here:
https://docs.docker.com/engine/reference/run/
Container Management

docker ps
List all running containers

docker ps -a
List all containers (also stopped)

docker stop <container>
Stop a running container

docker rm <container>
Remove a stopped container
Container Debugging

# docker inspect <container>
Low-level information on a container or image

# docker exec -ti <container> bash
Start an interactive shell into a running container

# docker run -ti --entrypoint=bash <imagename>
Start image with a different entrypoint
Docker Hub: Public Registry

OFFICIAL REPOSITORY

nginx ⭐
Last pushed: 4 days ago

Repo Info  Tags

Short Description
Official build of Nginx.

Docker Pull Command

docker pull nginx

Full Description

Supported tags and respective

Dockerfile links

- latest, 1, 1.9, 1.9.15 (mainline/jessie/Dockerfile)
- stable, 1.10, 1.10.0 (stable/jessie/Dockerfile)
- mainline-alpine, alpine, 1-alpine, 1.9-alpine, 1.9.15-alpine (mainline/alpine/Dockerfile)
- stable-alpine, 1.10-alpine, 1.10.0-alpine (stable/alpine/Dockerfile)

ImageLayers.io  183 MB / 8 Layers

For more information about this image and its history, please see the relevant manifest file (library/nginx). This image is updated via pull requests to the docker-library/official-images
Docker Hub: Automated/Trusted Build
Improves transparency

https://docs.docker.com/docker-hub/builds/
Cloud Deployment

Infrastructure as Code, Containerization

Summary

Means of automating infrastructure provisioning

Provides formal definition and documentation of infrastructure and configuration

Enables use of software engineering best practices

Encourages reuse of infrastructure code even beyond organisational boundaries (Dockerhub, Chef Recipes, etc.)

Containers also allow a way of packaging application into a standardized unit for deployment in different stages