Infrastructure-as-Code

PXL DIGITAL

An introduction

PXL DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Goals

De student

- kan de term Infrastructure-as-Code en de kenmerken ervan beschrijven
- kan de vier tooling categorieën beschrijven en onderscheiden.
- kan mutable en immutable infrastructure beschrijven en het verschil tussen beide uitleggen.
- kan imperative en declarative code beschrijven en het verschil tussen beide uitleggen.
- kan de verschillende IaC Stack Deployment benaderingen beschrijven en onderscheiden in twee groepen.
- kan de voor- en nadelen van de twee soorten van IaC Stack Deployment benaderingen beargumenteren.
- kan populaire tools (i.e. Puppet, Chef, Ansible, Terraform) linken en verklaren met het corresponderende Tool type, Infrastructuur, Architectuur, Aanpak en Gebruikte manifest taal.

The "Cloud Age" of IT

- Where all the cool kids are now "DevOps".
- You're no longer doing "scripting," just "coding".
- Apps are decoupled from physical hardware.
- Routine provisioning and maintenance can be delegated to software systems, freeing humans from drudgery and tedium.
- Changes can be made in minutes, if not seconds.
- Change management can exploit this speed and provide better reliability and faster release cycles.

"Infrastructure-as-Code" appears...

Why infrastructure-as-code matters: a short story.

You are starting a
new project

I know, I’ll use Ruby on Rails!

```
> gem install rails
```

```
> gem install rails
Fetching: i18n-0.7.0.gem (100%)
Fetching: json-1.8.3.gem (100%)
Building native extensions. This could take a while...
ERROR: Error installing rails:
ERROR: Failed to build gem native extension.

        /usr/bin/ruby1.9.1 extconf.rb
creating Makefile

make
sh: 1: make: not found
```

Ah, I just need to install make

```
> sudo apt-get install make
...
Success!
```

```
> gem install rails
```

```
> gem install rails
Fetching: nokogiri-1.6.7.2.gem (100%)
Building native extensions. This could take a while...
ERROR: Error installing rails:
ERROR: Failed to build gem native extension.

        /usr/bin/ruby1.9.1 extconf.rb
checking if the C compiler accepts... yes
Building nokogiri using packaged libraries.
Using mini_portile version 2.0.0.rc2
checking for gzdopen() in -lz... no
zlib is missing; necessary for building libxml2
*** extconf.rb failed ***
```

Hmm. Time to visit StackOverflow.

```
> sudo apt-get install zlib1g-dev
...
Success!
```

```
> gem install rails
```

```
> gem install rails
Building native extensions. This could take a while...
ERROR: Error installing rails:
ERROR: Failed to build gem native extension.

        /usr/bin/ruby1.9.1 extconf.rb
checking if the C compiler accepts... yes
Building nokogiri using packaged libraries.
Using mini_portile version 2.0.0.rc2
checking for gzdopen() in -lz... yes
checking for iconv... yes

Extracting libxml2-2.9.2.tar.gz into tmp/x86_64-pc-linux-
gnu/ports/libxml2/2.9.2... OK
*** extconf.rb failed ***
```

y u never install correctly?

(Spend 2 hours trying random StackOverflow suggestions)

```
> gem install rails
```

```
> gem install rails
...
Success!
```

Finally!

```
> rails new my-project
> cd my-project
> rails start
```

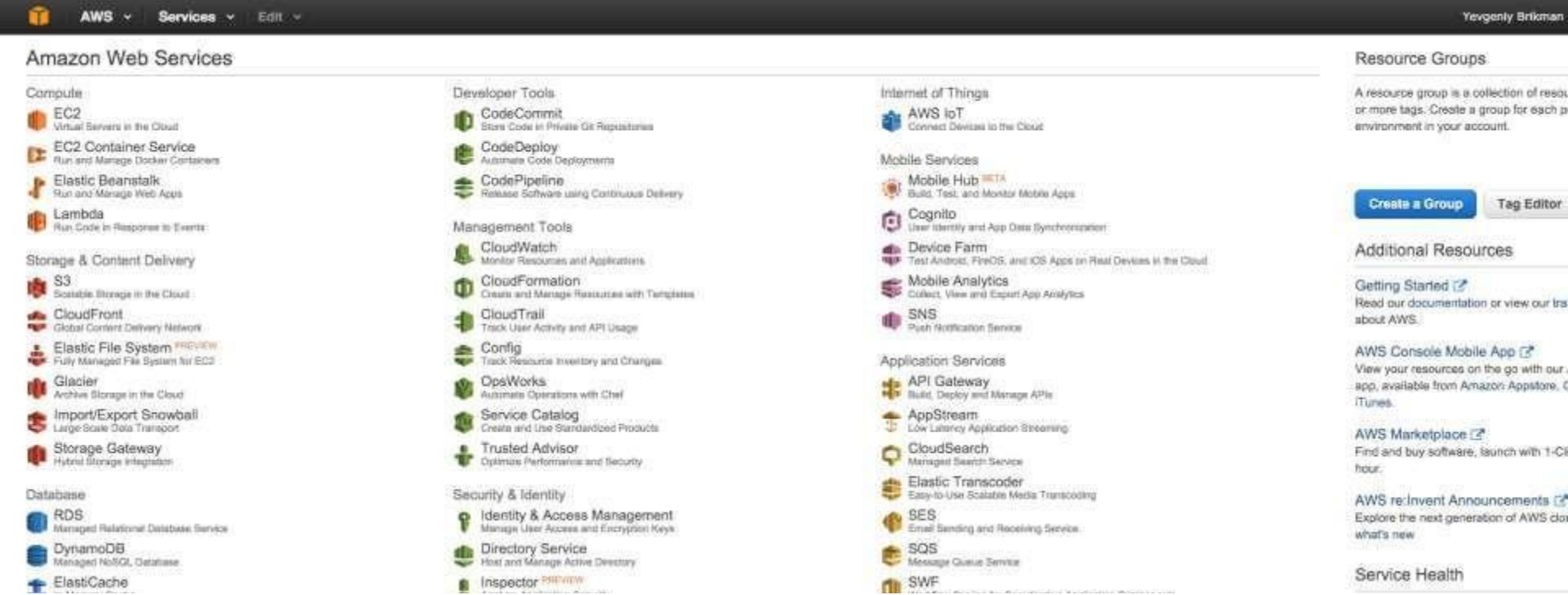
```
> rails new my-project
> cd my-project
> rails start

/source/my-project/bin/spring:11:in `<top (required)>':
undefined method `path_separator' for Gem:Module
(NoMethodError)
  from bin/rails:3:in `load'
  from bin/rails:3:in `<main>'
```


FFFFFFF
FFFFFFF
FFFFFFF
FFFFF
FFFFF
FFFFF
UUUU
UUUU
UUUU-

Eventually, you get it working

Now you have to deploy your Rails app in production



You use the AWS Console to deploy an EC2 instance

```
> ssh ec2-user@ec2-12-34-56-78.compute-1.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|
      ___|
[ec2-user@ip-172-31-61-204 ~]$ gem install rails
```

```
> ssh ec2-user@ec2-12-34-56-78.compute-1.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

[ec2-user@ip-172-31-61-204 ~]$ gem install rails
ERROR: Error installing rails:
ERROR: Failed to build gem native extension.

        /usr/bin/ruby1.9.1 extconf.rb
```


FFFFFFF
FFFFFFF
FFFFFFF
FFFFF
FFFFF
FFFFF
UUUU
UUUU
UUUU-

Eventually you get it working

Critical Ruby On Rails Issue Threatens 240,000 Websites

Bug allows attackers to execute arbitrary code on any version of Ruby published in the last six years.

All versions of the open source Ruby on Rails Web application framework released in the past six years have a critical vulnerability that an attacker could exploit to execute arbitrary code, steal information from databases and crash servers. As a result, all Ruby users should immediately upgrade to a newly released, patched version of the software.

That warning was sounded Tuesday in a Google Groups post made by Aaron Patterson, a key Ruby programmer. "Due to the critical nature of this vulnerability, and the fact that portions of it have been disclosed publicly, all users running an affected release should either upgrade or use one of the work arounds immediately," he wrote. The patched versions of Ruby on Rails (RoR) are 3.2.11, 3.1.10, 3.0.19 and 2.3.15.

As a result, more than 240,000 websites that use Ruby on Rails Web applications are at risk of being exploited by attackers. High-profile websites

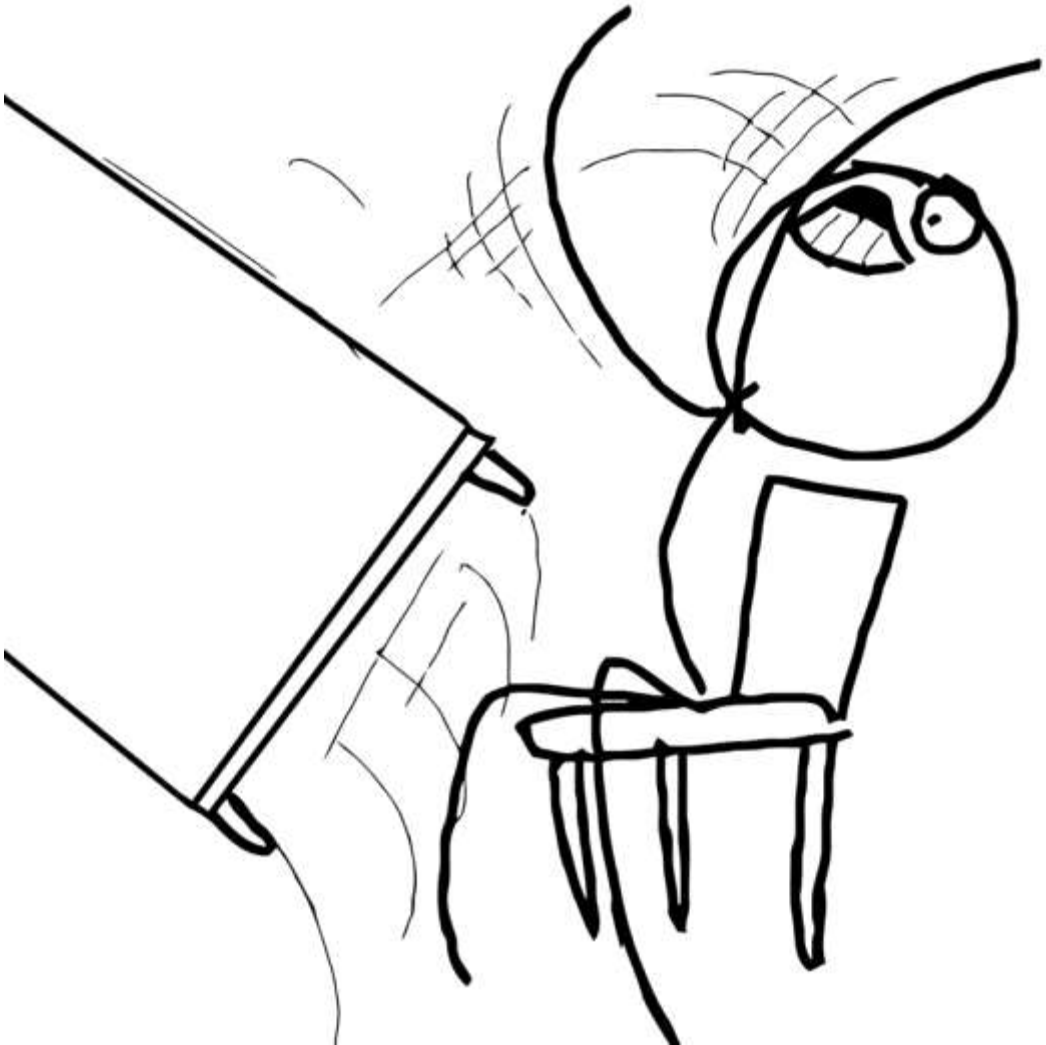
Now you urgently have to update all your Rails installs

```
> bundle update rails
```

```
> bundle update rails
Building native extensions. This could take a while...
ERROR: Error installing rails:
ERROR: Failed to build gem native extension.

        /usr/bin/ruby1.9.1 extconf.rb
checking if the C compiler accepts... yes
Building nokogiri using packaged libraries.
Using mini_portile version 2.0.0.rc2
checking for gzdopen() in -lz... yes
checking for iconv... yes

Extracting libxml2-2.9.2.tar.gz into tmp/x86_64-pc-linux-
gnu/ports/libxml2/2.9.2... OK
*** extconf.rb failed ***
```

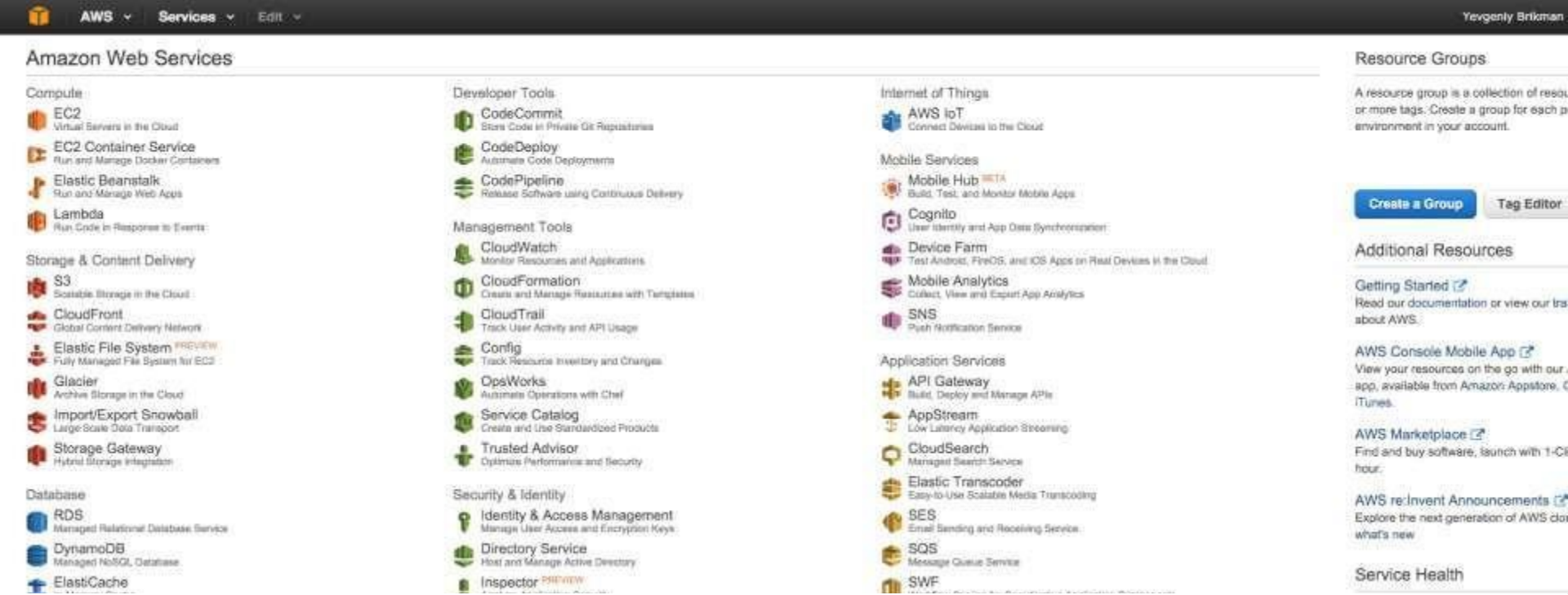


The problem isn’t Rails

```
> ssh ec2-user@ec2-12-34-56-78.compute-1.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|
      ___|
[ec2-user@ip-172-31-61-204 ~]$ geminstall rails
```

The problem is that you're configuring servers manually

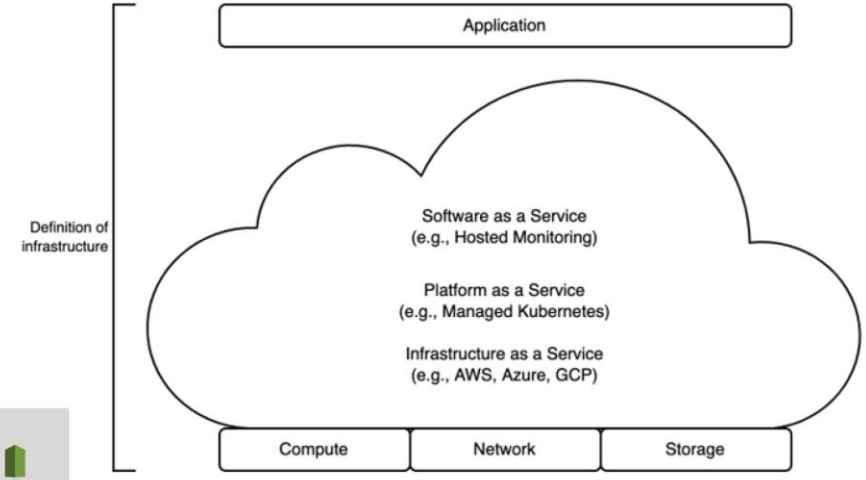


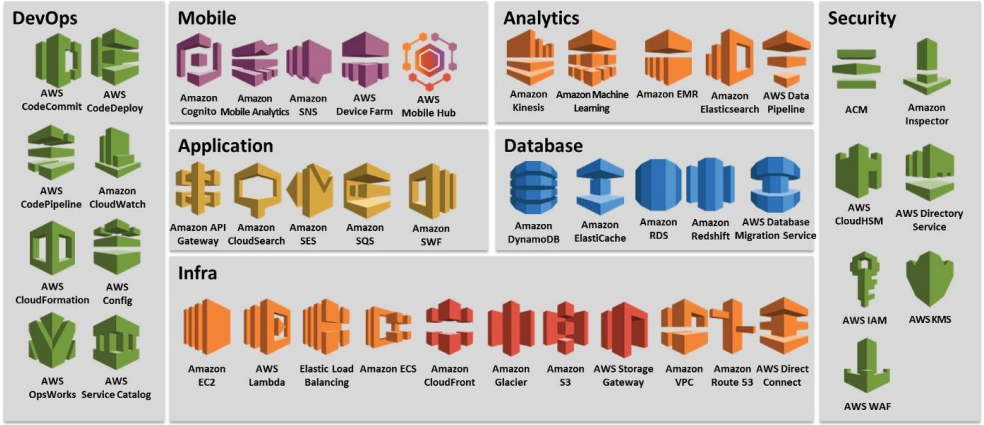
And that you're deploying infrastructure manually

A better alternative: infrastructure- as-code

What is infrastructure?

Last century, this was a data center in the basement with servers and racks.





Infrastructure for an application today can even include queues on a public cloud, containers of running applications, serverless functions for additional processing, or monitoring services to check system health.

PXL DIGITAL

What Is Infrastructure as (from) Code?

- Infrastructure-as-Code (IaC) is an approach to infrastructure automation based on practices from software development.
- The emphasis is on consistent, repeatable routines for the deployment and modification of systems and their configuration.
- Changes are made to definitions and then deployed to systems through unattended processes that include thorough validation.

```
resource "google_compute_instance" "vm_instance" {
 name = "terraform-instance"
 machine_type = "f1-micro"
 initial_node_count = “3”
 boot_disk {
    initialize_params {
        image = "debian-cloud/debian-9"
        }
 }
 network_interface {
    network = google_compute_network.vpc_network.name
    access_config {
    }
 }
 }
```

TerraForm voorbeeld

Tooling Categories

- Ad-Hoc scripts
- Configuration management Tools
- Server Templating Tools
- Server Provisioning Tools

PXL DIGITAL

Ad-Hoc Scripts

```
# Function createLog()
#  Create a log for output from this script to file.
func_createLog(){
    #Redirect output to logfile
    dateNow=$(date '+%Y-%m-%d_%H-%M-%S')
    exec > >(tee -i $DIR_LOG/DeployLog_$dateNow.log)
    exec 2>&1
}
```

```
# Function update Live
#  copy code from build dir to live
func_updateLive(){
    source=$dir_git"/www/"
    destination=$dir_live

    echo "source : "$source
    echo "dest   : "$destination

    rsync -rv --progress --stats \
        $source $destination

    func_checkCmdStatus "rsync"
}
```

```
# Function checkCmdStatus()
#  check the status of previously executed command
func_checkCmdStatus(){
    if [ $? -eq 0 ]
    then
        echo "--OK $1"
    else
        echo "Failure: command failed $1" >&2
        echo "Exiting!!!" >&2
        func_pingGithub false;
        exit 1
    fi
}
```

```
# Function init()
#  loads config file with specifc env variables
#  env='local', 'test', 'stage', 'prod'
func_init(){
    configFile='config'
    if [ -f "$configFile" ]; then
        source $configFile;

        #enable logging
        func_createLog;
        echo $ENV

        #Load settings
        env=$ENV;
        gitrepo=$GIT_REPO;
        gitbranch=$GIT_BRANCH
        gitssh=$GIT_SSH

        dir_git=$DIR_GIT

        dir_build=$DIR_BUILD
        dir_live=$DIR_LIVE

    else
        echo "Config file not found [$configFile]" >&2
        echo "Exiting!!!" >&2
        exit 1
    fi
}
```

PXL DIGITAL

Configuration Management Tools

- **Chef, Puppet, Ansible, and SaltStack** are all *configuration management* tools, designed to install and manage software on existing servers.
- **Coding conventions** - Consistent and predictable structure, file layout, clearly named parameters, secrets management, etc.
- **Idempotent Code** - Running the same code repeatedly while producing the same result.
- **Distribution** - Unlike ad hoc scripts, CM tools are specifically designed for managing large numbers of remote servers.

puppet CHEF ANSIBLE SALTSTACK

```yaml
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

Ansible example

Server Templating Tools

- Increasingly popular are server templating tools such as **Docker**, **Packer** and **Vagrant**.
- Create an image of a server that creates a completely self-contained "snapshot" of the operating system, software, files, and all other relevant details.
- And move on to the next deployment step in your pipeline
- Server templating is an important part of the shift to an **immutable** infrastructure.

```
# syntax=docker/dockerfile:1
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Dockerfile example

docker

HashiCorp Packer

HashiCorp Vagrant

Server (or "resource") Provisioning Tools

- Provisioning tools such as **Terraform**, **Azure Resource Manager Templates**, **AWS CloudFormation** or **Google Cloud Deployment Manager** are responsible for building servers.
- You can use these tools not only to build servers, but also other resources such as databases, load balancers, firewall settings, storage, etc.
- Multi-Platform
- And so much more...

```
resource "google_compute_instance" "vm_instance" {
 name = "terraform-instance"
 machine_type = "f1-micro"
 initial_node_count = “3”
 boot_disk {
    initialize_params {
        image = "debian-cloud/debian-9"
        }
 }
 network_interface {
    network = google_compute_network.vpc_network.name
    access_config {
    }
 }
 }
```

TerraForm example

HashiCorp Terraform

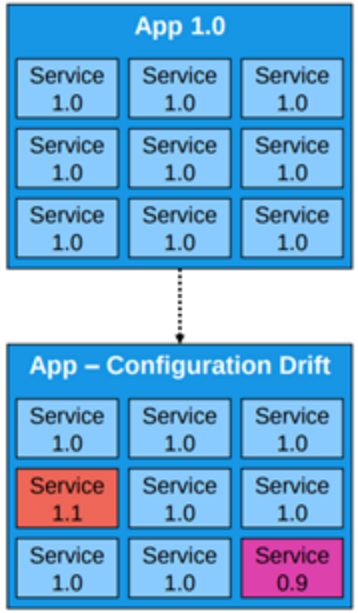
CloudFormation

PXL DIGITAL

Mutable & Immutable Infrastructure

Mutable Infrastructure

- **You can update your mutable infrastructure on the spot, without recreation or restart.**
- Infrastructure will be constantly updated, patched and tuned to meet the changing needs for which it exists.
- CM tools such as Chef or Puppet typically belong to a mutable infrastructure paradigm.
- Over time, as you apply more and more updates, each server builds a unique history of changes.
- As a result, each service is a little different from all the others, which leads to *configuration drift* and can end up with difficult to find and repurpose bugs.

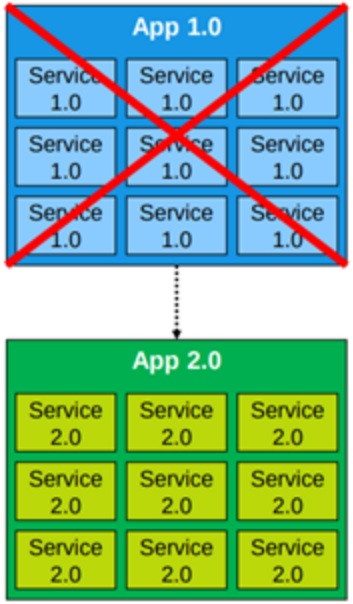


PXL DIGITAL

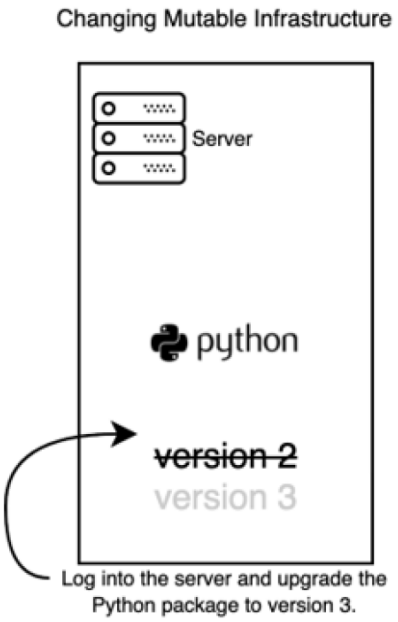
Mutable & Immutable Infrastructure

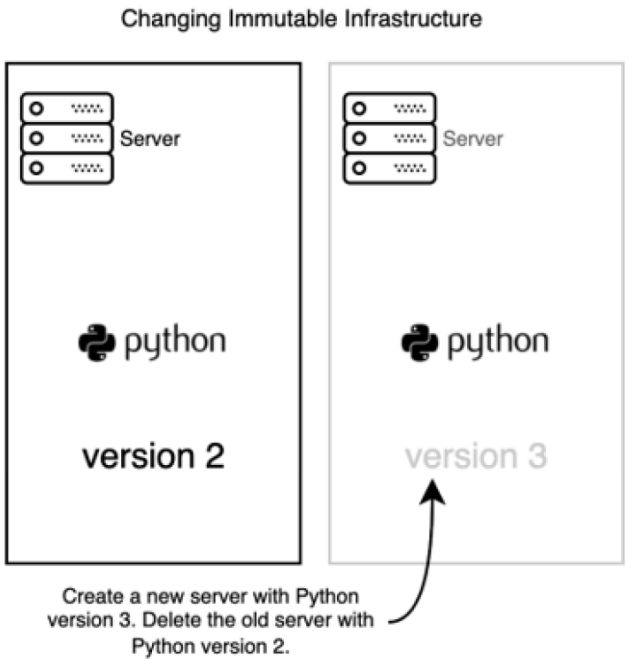
Immutable Infrastructure

- **You must create a new resource for each change to the infrastructure configuration.**
- You do not change the resource after you create it.
- Using Terraform to deploy machine images created with Docker or Packer results in "changes" that are an implementation of an entirely new app version.
- Reduces the chance of configuration drift bugs and makes it easier to know exactly what software is running on each server.
- Automated testing is more effective because an immutable image that passes your tests in the test environment is likely to behave exactly as it does in the production environment.
- Blue / Green



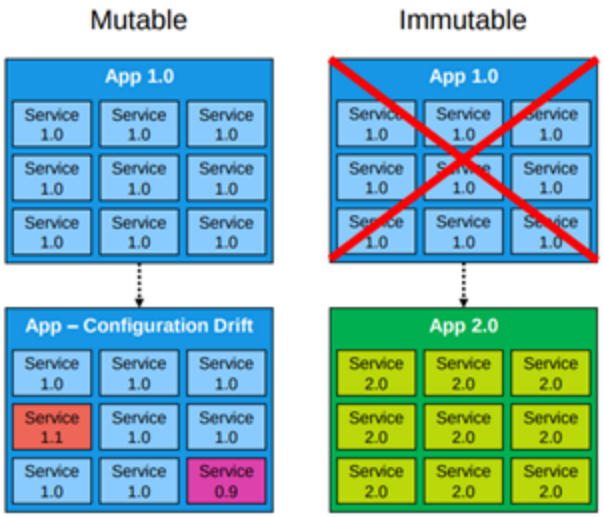
Mutable & Immutable Infrastructure - example





Mutable vs Immutable Infrastructure

- The Pets and Cattle debate.
- Best solution depends on the use case.
- With the mutable approach, the whole team must always be aware of the infrastructure "history."
- The immutable approach is better for stateless applications.
- Immutable does not allow deviations or changes. It is what it is.



PXL DIGITAL

Imperative Code vs. Declarative Code

Imperative (procedural):
Defines specific commands that need to be executed in the appropriate order to end with the desired conclusion.

AKA "The How"

5

My house is #9 and will be on the right

1

Leave the house

2

Get in the the car

3

Drive straight on Morty Blvd. for 3km

4

Turn right on Rick street and drive for 5 blocks

PXL DIGITAL

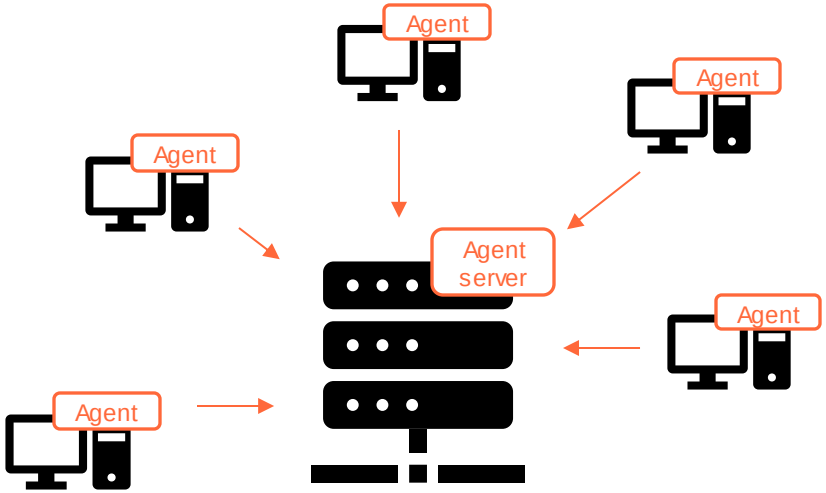
Imperative Code vs. Declarative Code

Declarative (functional):
Defines the desired state and the system executes what needs to happen to achieve that desired state.

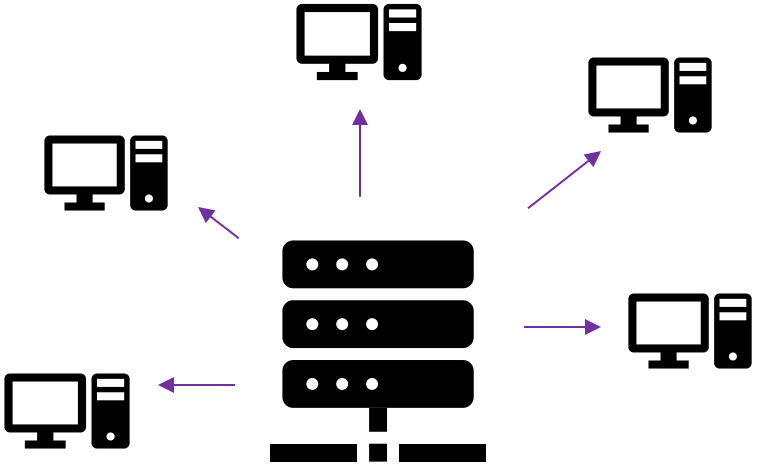
AKA "The What"

My address is: 9 Rick Street, Jacksonville, FL 32218, USA

IaC Stack Deployment Approach



Pull



Push

PXL DIGITAL

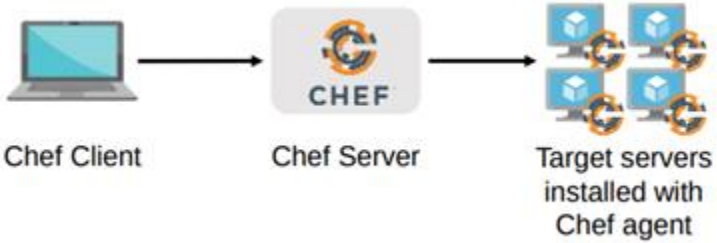
Pull / Master / Agent

Pros:

- Central place where you can manage the status of your infrastructure.
- Some tools provide a web interface for the master server.
  - Puppet Enterprise (PE) console
  - Chef management console
- Scalability

Cons:

- Requires a daemon to be installed on all machines and a central authority to be setup.
- Extra infrastructure
- Maintenance
- Security
- Difficult to manage – moving parts, availability, etc



PXL DIGITAL

Push / Masterless / Agentless

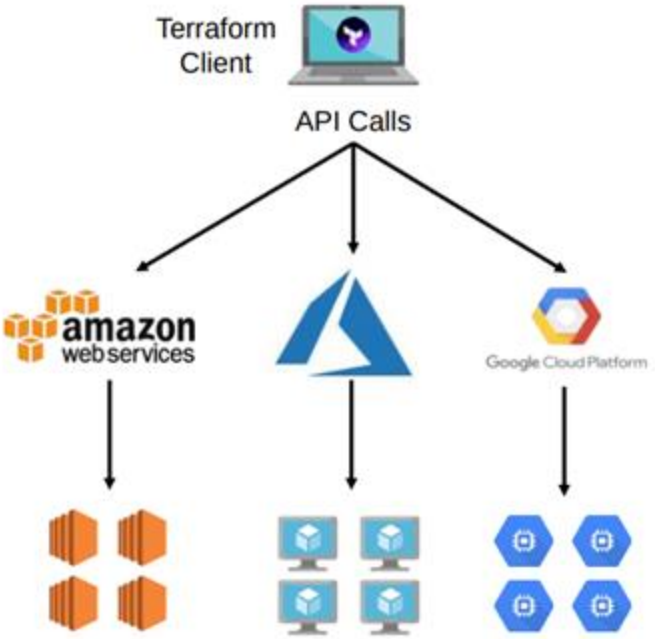
- A client contacts the nodes and sends updates as they are needed.
- When a change is made to the infrastructure, each node is alerted of this and then run the changes.

Pros :

Simple to manage and setup

Cons :

No central control plane



Puppet

- Type: Configuration Management
- Infrastructure: Mutable
- Architecture: Pull
- Approach: Declarative
- Language: DSL & Ruby

```
define apache::vhost ( $port, $docroot, $template='apache/vhosts.erb') {
file { "/etc/apache2/sites-available/$name":
 content => template($template),
      owner => 'root',
      group => 'wheel',
      mode => "644', }
 }
```

Example of usage

```
node 'www' {
include apache
apache::vhost { 'www-second':
     port => 80,
     docroot => '/var/www/www-second',
     template => 'apache/www_vhosts',
     }
}
```

(Progress) Chef

- Type: Configuration Management
- Infrastructure: Mutable
- Architecture: Pull
- Approach: Declarative & Imperative
- Language: DSL & Ruby

```ruby
package 'apache2' do
  case node['platform']
  when 'centos', 'redhat', 'fedora', 'suse'
    package_name 'httpd'
  when 'debian', 'ubuntu'
    package_name 'apache2'
  when 'arch'
    package_name 'apache'
  end
  action :install
end
```

Ansible

ANSIBLE

- Type: Configuration Management
- Infrastructure: Mutable
- Architecture: Push
- Approach: Declarative & Imperative
- Language: YAML

```yaml
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

SaltStack

SALTSTACK

- Type: Configuration Management
- Infrastructure: Mutable
- Architecture: Push & Pull
- Approach: Declarative & Imperative
- Language: YAML

websetup.sls

```
websetup:
   pkg:
      - installed
      - pkgs:
         - apache2
         - php5
         - php5-mysql
```

```
root@saltmaster:/home/vagrant# salt 'minion2' state.sls websetup
```

Terraform

HashiCorp Terraform

- Type: **Provisioning**
- Infrastructure: Immutable
- Architecture: Push
- Approach: Declarative
- Language: HCL (HashiCorp Configuration Language)

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  required_version = ">= 1.2.0"
}

provider "aws" {
  region  = "us-west-2"
}

resource "aws_instance" "app_server" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleAppServerInstance"
  }
}
```

Who’s who?

Tool	Tool Type	Infrastructure	Architecture	Approach	Manifest Written Language
puppet	Configuration Management	Mutable	Pull	Declarative	Domain Specific Language (DSL) & Embedded Ruby (ERB)
CHEF	Configuration Management	Mutable	Pull	Declarative & Imperative	Ruby
ANSIBLE	Configuration Management	Mutable	Push	Declarative & Imperative	YAML
SALTSTACK	Configuration Management	Mutable	Push & Pull	Declarative & Imperative	YAML
Terraform	Provisioning	Immutable	Push	Declarative	HashiCorp Configuration Language (HCL)

PXL DIGITAL