puppet

Provisioning vs Configuration Management Deployment vs Orchestration

A rose by any other name...

FOSDEM 2018

Peter Souter

Technical Account Manager | Puppet

@petersouter

Who am I?

@petersouter

petems
IRC/Slack/GitHub

Technical Account Manager

7 years using Puppet

3 years @ Puppet Inc

Work with customers on their holistic Puppet Program

Help customers get the best use of Puppet

Evangelise and work with the community

A word of apology

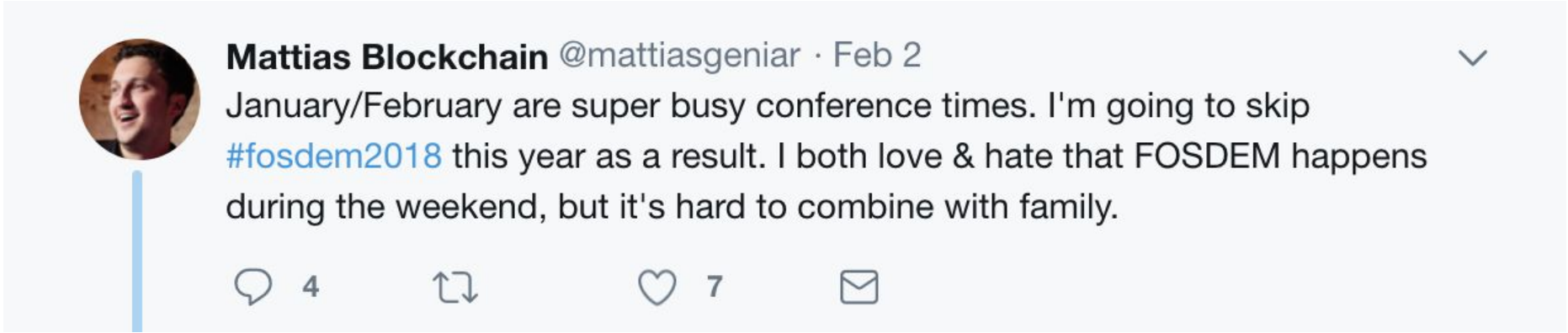
I’ve been distracted as of late...

I've been busy with a non-interruptible process...

DevOps is about empathy...

https://twitter.com/mattiasgeniar/status/582694171691106304

Babies also make it hard to attend confs :(



https://twitter.com/mattiasgeniar/status/959370407924129792

Oooh Meta!

https://twitter.com/PeterSouter/status/959784974785314821

Why am I talking about this?

Well, it keeps coming up so I ended up doing a talk...

But let's talk about names and definitions!

And hope I can make it through this in spite of my lack of sleep...

Naming things is hard

And meanings evolve over time...

But 4 terms seem to have stuck

Provisioning, Configuration Management, Deployment and Orchestration

And these terms mainly fit under one umbrella:

Infrastructure as Code

“Infrastructure as Code [allows you to treat] configuration files as software code. These files can be used to produce a set of artifacts, namely the compute, storage, network, and application services that comprise an operating environment. **Infrastructure as Code eliminates configuration drift, through automation, thereby increasing the speed and agility of infrastructure deployments.**”

- **Infrastructure as Code (July 2017), AWS Whitepaper, https://d0.awsstatic.com/whitepapers/DevOps/infrastructure-as-code.pdf**

We’re here to talk about where everything fits

The history of each area, the differences and where the lines blur

Let's start at the earliest part of the lifecycle: Provisioning

Creating the systems that you'll need to manage later on

Let's talk about the history of provisioning

You don’t know where you're going without seeing where you’ve been...

Before cloud based provisioning, we had bare metal and boot provisioning

Cobbler, Kickstart, PXE booting scripts etc.

Fog is probably the genesis of cloud based provisioning

“When Wes started fog it was a Ruby library for AWS **(which didn't offer an SDK at the time)**. The abstractions that were implemented were portable and abstract enough to be expanded to other providers like Rackspace.”

- https://github.com/fog/fog/wiki/fog-design-document

Fog allowed abstraction: EC2

```
require 'rubygems'
require 'fog'

# create a connection
connection = Fog::Compute.new({
  :provider                 => 'AWS',
  :aws_access_key_id        => YOUR_AWS_ACCESS_KEY_ID,
  :aws_secret_access_key    => YOUR_AWS_SECRET_ACCESS_KEY
})

server = connection.servers.create
```

Fog allowed abstraction: GCE

```
require 'rubygems'
require 'fog'

# create a connection
connection = Fog::Compute.new({
  :provider => 'google',
  :google_project => GOOGLE_PROJECT_ID,
  :google_client_email => GOOGLE_SERVICE_EMAIL,
  :google_key_location => GOOGLE_P12_KEY_LOCATION,
})

server = connection.servers.create
```

Fog allowed abstraction: Rackspace (RIP)

```ruby
require 'rubygems'
require 'fog'

# create a connection
connection = Fog::Compute.new({
  :provider                 => 'Rackspace',
  :rackspace_username       => RACKSPACE_USERNAME,
  :rackspace_api_key        => RACKSPACE_API_KEY
})

server = connection.servers.create
```

Terraform took this concept and ran with it

```
resource "aws_instance" "web" {
  ami = "ami-12345678"
  instance_type = "t1.micro"
  tags {
    Name = "HelloWorld"
  }
  security_groups = [ "${aws_security_group.my_security_group.id}" ]
}
```

One of the big problems here: Drift

APIs can be deprecated, products discontinued and workarounds can become tightly coupled to the tool

As someone who maintains a cloud provisioning tool, I've had 1st hand experience...

I maintain Tugboat, a DigitalOcean Ruby CLI app

Tugboat - A Provisioning CLI for DigitalOcean

```
┌─petersouter@Peters-MacBook-Pro /Users/petersouter  ‹2.3.0›
└─$ tugboat droplets
petersouter-blog-and-things (ip: 46.101.86.69, status: active, region: lon1, size: 1gb, id: 32142097)
```

https://github.com/petems/tugboat/

DigitalOcean 1.0 Deprecation

API v1 has reached end-of-life. All subsequent requests to API v1 will respond with HTTP 410 (Gone) and an error message:

```
# HTTP/1.1 410 (Gone)
{
  "status": "ERROR",
  "error_message": "API v1 has reached end-of-life. Please use API v2.",
  "message": "API v1 has reached end-of-life. Please use API v2."
}
```

https://developers.digitalocean.com/documentation/v1/

Bit of a scramble to go fix it...

API v2 Support #102

Closed **wearhere** opened this issue on 5 Jul 2014 · 12 comments

wearhere commented on 5 Jul 2014 — Contributor

Do you have plans to support Digital Ocean's v2 API? It seems impossible currently to authorize tugboat as it still requires API and client keys vs. v2's new personal access tokens.

blom commented on 6 Jul 2014 — Collaborator

Support for version 2 is planned. In the meantime, keys for version 1 can be managed here:

https://cloud.digitalocean.com/api_access

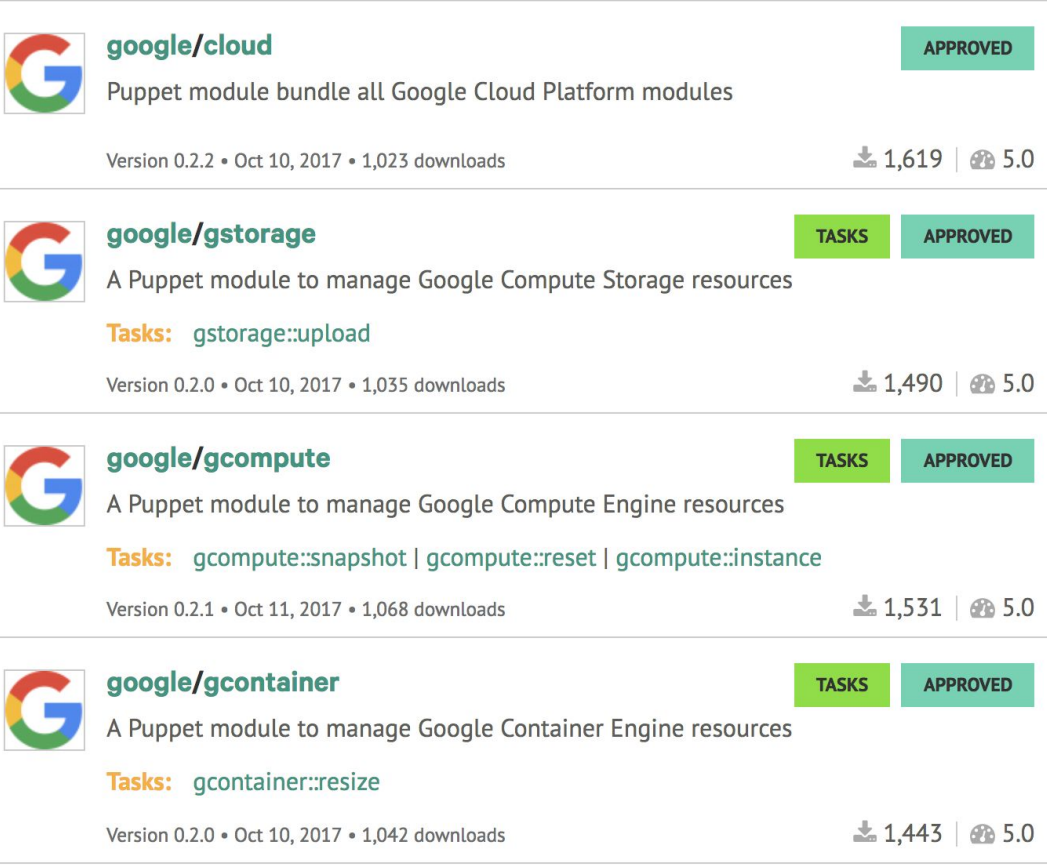
wearhere commented on 7 Jul 2014 — Contributor

Oh thanks! I didn't see that you could still get to the v1 API page from the top of the v2 page.

Workaround: Have the cloud vendor maintain the tool

Avoids support ambiguity and allows development by the specialists

https://forge.puppet.com/modules?utf-8=✓&sort=rank&q=google



Terraform Provider Development Program

The Terraform Provider Development Program allows vendors to build Terraform providers that are officially approved and tested by HashiCorp and listed on the official Terraform website. The program is intended to be largely self-serve, with links to information sources, clearly defined steps, and checkpoints.

Building your own provider? If you're building your own provider and aren't interested in having HashiCorp officially approve and regularly test the provider, refer to the Writing Custom Providers guide.

https://www.terraform.io/guides/terraform-provider-development-program.html

Better Workaround: Cloud vendor creates provisioning framework!

AWS has Cloudformation and Boto

First bit of confusion: Config Management tools often have adjacent provisioning tools

Whilst most config management tools are more for after provisioning, most have a way of creating instances

Puppet: AWS Module

```
ec2_instance { 'name-of-instance':
  ensure            => running,
  region            => 'us-east-1',
  availability_zone => 'us-east-1a',
  image_id          => 'ami-123456',
  instance_type     => 't2.micro',
  key_name          => 'name-of-existing-key',
  subnet            => 'name-of-subnet',
  security_groups   => ['name-of-security-group'],
  tags              => {
    tag_name => 'value',
  },
}
```

https://github.com/puppetlabs/puppetlabs-aws

Chef: Knife

```
knife ec2 server create
  --image ami-ce7b6fba
  --flavor m1.small
  --region eu-west-1
  --server-connect-attribute private_ip_address
  --ssh-gateway user@gateway.ec2.example.com
  --ssh-user ubuntu
  --identity-file ~/.ssh/clarkdave.pem
  --subnet subnet-8d034be5
  --environment production
  --node-name web1
  --run-list 'role[base],role[web_server]'
```

https://github.com/chef/knife-ec2

Ansible: AWS Playbook

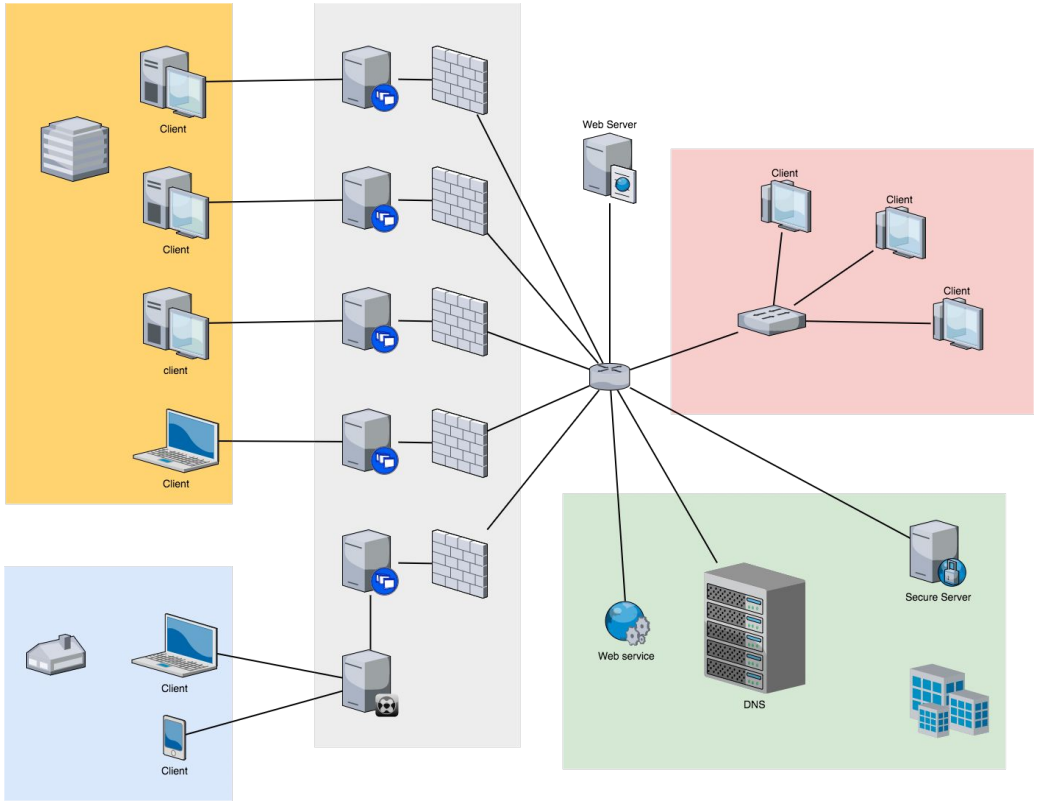
```
- ec2:
    key_name: mykey
    instance_type: t2.micro
    image: ami-123456
    wait: yes
    group: webserver
    count: 3
    vpc_subnet_id: subnet-29e63245
    assign_public_ip: yes
```

http://docs.ansible.com/ansible/latest/ec2_module.html

There is a benefit of using the same tool for config management and provisioning

Allows consolidation of information in one place and allows one tool to be the single source of truth

Ok, so we've provisioned our resources...



How do we actually make them useful?

How do we install and configure them?

No problem, I have a shell script!

A sentence that still makes me involuntary shudder...

Problem: Shell scripts are hard to scale

Hard to read, difficult to maintain, fiddly issues across operating systems...

If only there was a better way...

Some sort of language for **configuration management...**

Let's talk about the history of config management!

Warning: Biases may be present! ;)

An extremely revisionist and broad overview of Config management history:

- First Mark Burgess made CFEngine, arguably the first big open-source config management tool
- Luke Kaines got frustrated with CFEngine, and made Puppet
- Adam Jacobs got frustrated with Puppet, and made Chef
- Michael Dehaan got frustrated with both, and wanted something more “ad-hoc”, so made Ansible based on his experience with Puppet and Cobbler

Each tool has its own pluses and minuses

And obviously I’m biased to say Puppet is best ;)

Puppet

```
class profile::base {

  include ::profile::base::ssh_pub_keys
  include ::profile::base::ssh_hardening
  include ::profile::base::ntp
  include ::profile::base::swap
  include ::profile::base::tcpwrappers
  include ::profile::base::sysctl
  include ::profile::base::rkhunter

  # OS Specific configuration
  include "::profile::base::osfamily::${::osfamily}"

}
```

Chef

```
name "base5"
description "Contains run list of roles which are safe to run on all RHEL5 and RHEL6 servers."
"ntp" => {
  "servers" => [
    "192.0.2.247",
    "192.0.2.193",
    "192.0.2.194"
  ]
},
"resolver" => {
  "search" => "example.com",
  "nameservers" => [
    "192.0.2.30",
    "192.0.2.36",
    "192.0.2.33"
  ],
}
```

Ansible

```
# Apply baseline configuration for RHEL 7 webservers
- name: apply baseline configs
  hosts: testing
  remote_user: root
  roles:
    - accounts
    - packages
    - ssh
    - ntp
    - webserver
    - selinux
  post_tasks:
    - name: restart system
      shell: sleep 2 && systemctl reboot
      async: 1000
      poll: 0
```

Whatever tool you pick, the principles are the same:

A way of treating configuration like code and the benefits that come with that

So, regardless of the tool you pick there's a lot of advantages to using config management

Reduce cost and time per release

Pre-existing code for often available

Potential for sharing and reuse

Share within your organisation or with the public

Single Source of Truth

Your code repository becomes your one place to configuration

Less arguments about semantics

Agreed upon DSL means closer collaboration between practitioners

Ok, our systems are configured...

How do we get our code we've developed on to them?

Ok, so first line-blurring situation...

Does provisioning include **initial** configuration and config management?

Getting some baseline into your provisioning is fine

- In the UK this is called “belt and braces”
- There’s probably a baseline set that should be applied to account for if your config management tool fails
- As part of the baseline, configure and install your config management tool

userdata

```
#!/bin/bash
DEBIAN_FRONTEND=noninteractive apt-get -y update
DEBIAN_FRONTEND=noninteractive apt-get -y upgrade
DEBIAN_FRONTEND=noninteractive apt-get install -y ufw
echo "Welcome to my computer!" >> /etc/motd
ufw default deny incoming
## Allow outgoing
ufw default allow outgoing
## Allow ssh
ufw allow ssh
## Enable ufw
ufw --force enable

# Install Puppet
sudo apt-get install -y puppet
```

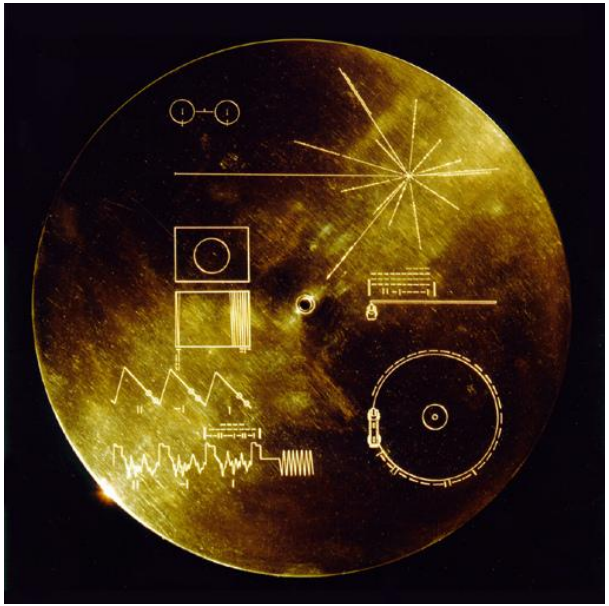
cloud-init

```
#cloud-config
apt:
  primary:
    - arches: [default]
      search:
        - http://local-mirror.mydomain
        - http://archive.ubuntu.com
    ssh_authorized_keys:
      - ssh-rsa
    AAAAB3NzaC1yc2EAAAABIwAAAGEA3FSyQwBI6Z+nCSjUUk8EEAnnkhXlukKoUPND/RRClWz2s5TCzIkd3Ou5+Cyz71X0XmazM3l5WgeErvtIwQMy
    T1KjNoMhoJMrJnWqQPOt5Q8zWd9qG7PBl9+eiH5qV7NZ mykey@host
      - ssh-rsa
    AAAAB3NzaC1yc2EAAAABIwAAAQEA3I7VUf2l5gSn5uavROsc5HRDpZdQueUq5ozemNSj8T7enqKHOEaFoU2VoPgGEWC9RyzSQVeyD6s7APMcE82E
    tmW4skVEgEGSbDc1pvxzxtchBj78hJP6Cf5TCMFSXw+Fz5rF1dR23QDbN1mkHs7adr8GW4kSWqU7Q7NDwfIrJJtO7Hi42GyXtvEONHbiRPOe8stq
    Uly7MvUoN+5kfjBM8Qqpfl2+FNhTYWpMfYdPUnE7u536WqzFmsaqJctz3gBxH9Ex7dFtrxR4qiqEr9Qtlu3xGn7Bw07/+i1D+ey3ONkZLN+LQ714
    cgj8fRS4Hj29SCmXp5Kt5/82cD/VN3NtHw== smoser@brickies
```

Cloudformation

```
Server:
        Type: AWS::EC2::Instance
        Metadata:
            AWS::CloudFormation::Init:
                configSets:
                    default:
                        - install_base
                install_base:
                    packages:
                        yum:
                        # Example packages:
                            bash-completion: []
                            telnet: []
                            wget: []
                    files:
                        # These files are needed for CloudFormation::Init to work
                        /etc/cfn/cfn-hup.conf:
                            content: !Sub |
                                [main]
                                stack=${AWS::StackId}
                                region=${AWS::Region}
                                interval=1
```

There’s also golden image considerations



https://flic.kr/p/86Bw8A

- Pre-baked initial configuration
- Config management tool pre-installed
- Vendor specific (qcow, AMI etc)
- However, easy for drift to occur, hard to maintain
- Balance between OS vendor standards and org standards

Ok, our systems are configured...

How do we get our code we've developed on to them?

Deployment

Getting the work we've done onto the systems in question

Deployment has come a long way

Who remembers doing something like this?

The “good” old days...

```
$ ssh prod-server.example.com
$ cd /var/webapps/django_app/
$ git pull
$ python manage.py migrate --noinput
$ service apache2 restart
```

Anyone seen Mr Robot?

https://gifs.com/gif/mr-robot-configuring-nginx-web-server-oQ7p8L

Maybe you tried to script it a bit...

```
APPNAME="django_app"
USER="django_user"
ROOT="/var/webapps/django_app/"
su - $USER <<EOF
cd $ROOT
# Update repository
echo " - Getting latest version..."
git pull -q >/dev/null
# Run django scripts
echo " - Running scripts..."
echo "   - Migrating database..."
python manage.py migrate --noinput >/dev/null
echo "   - Collecting static files..."
python manage.py collectstatic --noinput >/dev/null
echo "   - Compiling translations..."
python manage.py compilemessages >/dev/null
service apache2 restart
echo "Done deploying $APPNAME."
```

Hopefully, this was frustrating enough that you switched to a deployment tool

Every language and framework has their own one...

Capistrano (Ruby)

```ruby
require "bundler/capistrano"

server "prod.acmecom.com", :web, :app, :db, primary: true

set :application, "example"
set :user, "deployer"
set :deploy_to, "/home/#{user}/apps/#{application}"
set :deploy_via, :remote_cache
set :use_sudo, false
set :port, "3030"

set :scm, "git"
set :repository, "git@github.com:acmecom/#{application}.git"
set :branch, "master"
```

Fabric (Python)

```
import os
from contextlib import contextmanager
from fabric.api import cd, env, prefix, run, sudo, task


PROJECT_NAME = 'YOUR PROJECT NAME'
PROJECT_ROOT = '/var/www/%s' % PROJECT_NAME
VENV_DIR = os.path.join(PROJECT_ROOT, '.venv')
REPO = 'git@bitbucket.org:monmar/%s.git' % PROJECT_NAME

env.hosts = []

@task
def production():
    env.hosts = ['user@production-server']
    env.environment = 'production'

def clean():
    """Cleans Python bytecode"""
    sudo('find . -name \'*.py?\' -exec rm -rf {} \;')
```

Deployer (PHP)

```php
<?php
server('production-web', '<your server url>')
    ->path('<path to the project on your server>')
    ->user('<user on the server>')
    ->pubKey();
stage('master', ['production-web'], ['branch' => 'master']);
task('deploy:checkout', function () {
    runLocally('git checkout '.get('branch', '').' 2> /dev/null');
})->desc('checking out git branch');
task('deploy:app_down', function () {
    run('php artisan down');
    run('php artisan cache:clear');
})->desc('bringing app down');
task('deploy:pull_changes', function () {
    run('git pull origin '.get('branch', '').' 2> /dev/null');
    run('php artisan cache:clear');
})->desc('pull changes on server');
```

For me, the true measure of a deployment tool is it's rollback or failure management

The ability to roll back is one of the distinct advantages over a manual process

Generally, the deployment is more Dev owned

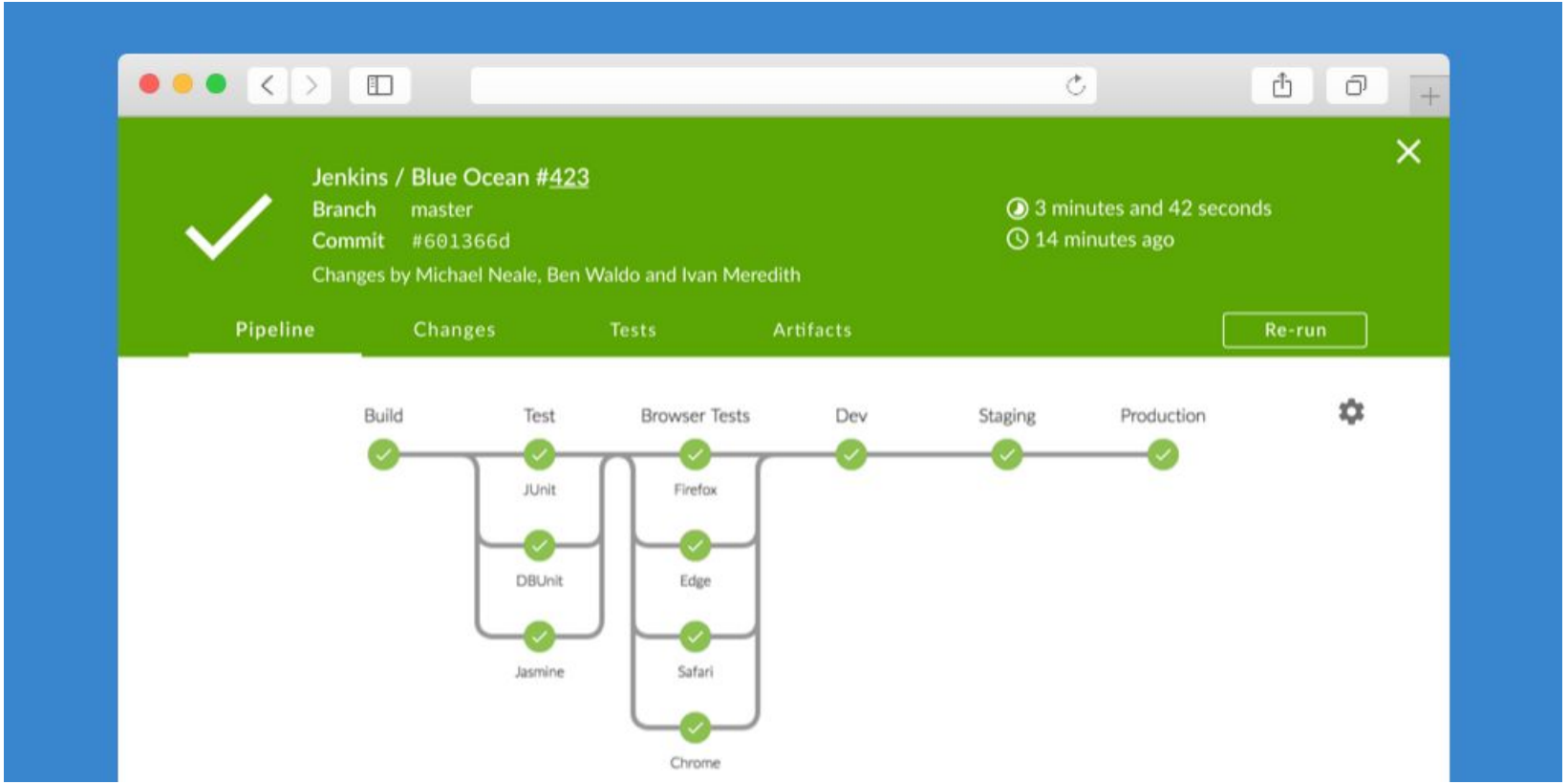
And config management is more ops owned

Leading to some collaboration (DevOps ahoy!)

Hopefully, this deployment is gated

Or do all your developers have SSH access to prod servers?

Jenkins Blue Ocean



Next line blurring moment

CI tools like Jenkins can technically do all 4 of the terms we used

CI Tools often end up being the main interface for the 4 terms we're talking about

Jenkins ends up being the gatekeeper, often pulling the strings of your infrastructure of code process

Next line blurring moment

Sometimes your config management is your deployment tool...

Package release as deployment

```
package { "acme-cool-app":
  ensure          => installed,
} ~>
file { "/etc/acme-cool-app.conf" :
  owner           => "root",
  group           => "root",
  mode            => "0644",
} ~>
service { "acme-cool-app":
  ensure          => running,
}
```

Lastly, we have orchestration

Something that doesn’t neatly fit into any other area...

The term orchestration is also a line-blurred area

As sometimes it’s used to encapsulate the entire “scope” of managing multiple machines and tools

What's different between orchestration and config management?

- Information gathering and systems intelligence
- One-off tasks that don't fit into a state-based workflow
- Triggering config management runs within a time-window
- Emergency "oh god do it now we need to fix it" situations

Ansible is an interesting edge case

It feels more “orchestration-y” than the other config management tools

Ansible at it’s most basic usage

```
ansible -a ‘ps aux | grep apache’ -m shell proxies
```

Q: What is the most common orchestration tool?

I’m pretty sure 99% of the room have used it at least once

SSH!

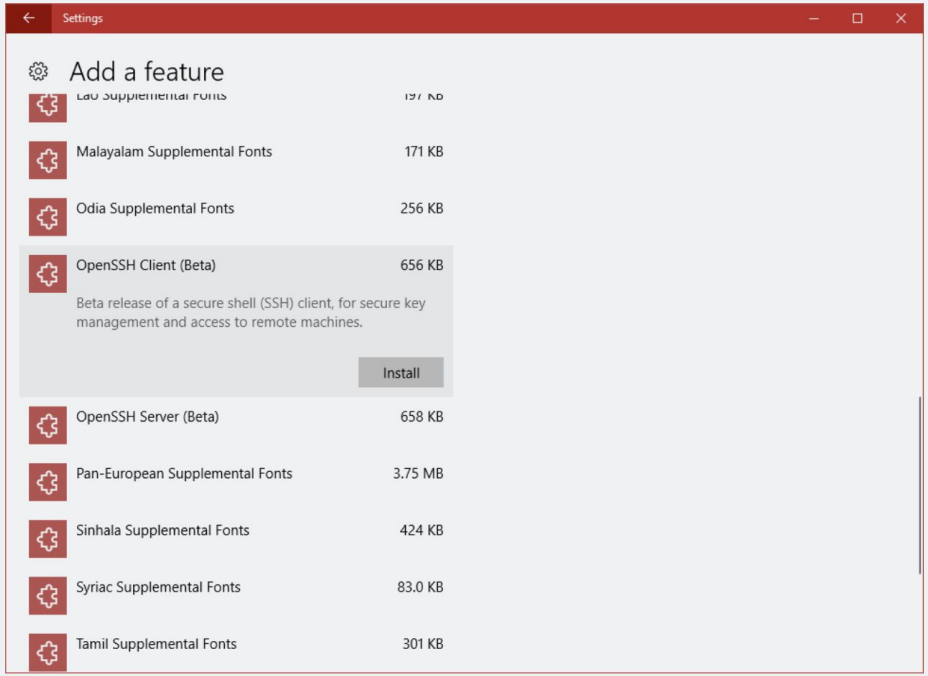
Gathering basic info with SSH

```
$ ssh admin@app001.example.com 'ps -aux'
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         2  0.0  0.0      0     0 ?        S     2017   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     2017   8:08 [ksoftirqd/0]
root         7  0.0  0.0      0     0 ?        S     2017   0:00 [migration/0]
root         8  0.0  0.0      0     0 ?        S     2017   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        R     2017  13:32 [rcu_sched]
```

SSH is the lingua franca for server communication

Even Microsoft is getting in on the act!

In 2015, Microsoft announced its intent to bring OpenSSH, the widely used implementation of the secure shell (ssh) protocol used for remote system access and administration throughout the UNIX world, natively to Windows. Without too many people noticing, it turns out that the company has now done this. The Windows 10 Fall Creators Update adds a couple of optional features, with both client and server now available for installation (via Serve The Home).



Enlarge

https://arstechnica.com/gadgets/2017/12/microsoft-quietly-snuck-an-ssh-client-and-server-into-the-latest-windows-10/

PSSH is a good starting point for orchestrating with SSH

```
$ pssh -i -h serverlist.txt -l deploy 'uname -r'
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
Password:
[1] 09:51:06 [SUCCESS] 192.168.5.88:22
3.10.0-123.el7.x86_64
[2] 09:51:06 [SUCCESS] 192.168.4.11:22
3.10.0-514.16.1.el7.x86_64
```

But obviously, we want to get more advanced

That’s where the orchestration tools come in!

Next blurred line, most Config Management tools have a separate orchestration app or tool

Which makes sense, as not everything can be done with config management as earlier discussed

Bolt ‘Vanilla’ (Puppet)

```
bolt command run "echo 'hello world'"
```

https://puppet.com/products/puppet-bolt

Bolt Plans (Puppet)

```
plan profile::base (
  String[1]                           $load_balancer,
  Variant[String[1], Array[String[1]]]  $frontends,
  Variant[String[1], Array[String[1]]]  $backends,
) {

  run_task('mymodule::lb_remove', $load_balancer, frontends => $frontends)
  run_task('mymodule::update_frontend_app', $frontends, version => '1.2.3')
  run_task('mymodule::lb_add', $load_balancer, frontends => $frontends)
}
```

```
bolt plan run mymodule::myplan --params \
'{"load_balancer":"lb.myorg.com","frontends":["kermit.myorg.com","gonzo.myorg.com"],"backends":["waldorf.
myorg.com","statler.myorg.com"]}'
```

Bolt Tasks implementation

```ruby
def apt_get(action)
  cmd_string = "apt-get #{action}"
  cmd_string << ' -y' if  action == 'upgrade'
  stdout, stderr, status = Open3.capture3(cmd_string)
  raise Puppet::Error, stderr if status != 0
  { status: stdout.strip }
end

params = JSON.parse(STDIN.read)
action = params['action']

begin
  result = apt_get(action)
  puts result.to_json
  exit 0
rescue Puppet::Error => e
  puts({ status: 'failure', error: e.message }.to_json)
  exit 1
end
```

```json
{
  "description": "Allows you to perform apt functions",
  "input_method": "stdin",
  "parameters": {
    "action": {
      "description": "Action to perform",
      "type": "Enum[update, upgrade]"
    }
  }
}
```

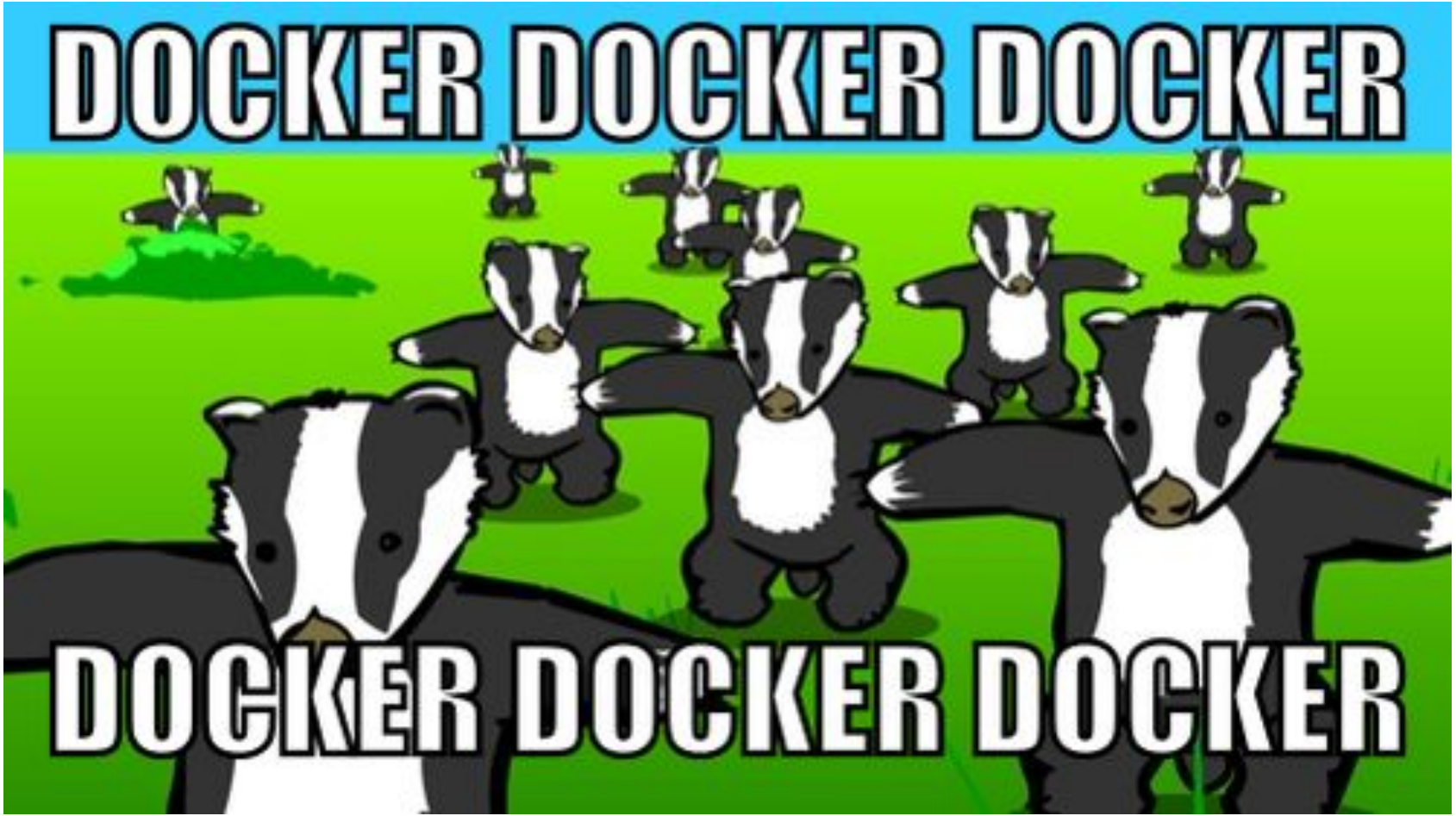
Knife (Chef)

```
# cookbooks/example1/recipes/default.rb
file '/tmp/my.txt' do
  content "hello"
end
```

```
chef exec knife zero converge  "name:mynode1" --ssh-user root
--ssh-password root --ssh-port 22 --attribute knife_zero.host
--override-runlist recipe_name
```

Ok, but there’s one thing we’ve forgotten...

One huge paradigm shift that it’s impossible to ignore!


DOCKER DOCKER DOCKER
DOCKER DOCKER DOCKER

Or more accurately: Containers Containers Containers

Where does it leave all the things we talked about?

Firstly: Provisioning is the least affected

You still need to create the servers to run the containers

Especially if you’re using a PaaS container platform

eg. EKS vs GKE vs AKS

But do we need config management or deployment if we have containers?

Why would we need to setup packages and config files, or deploy artifacts if we're delivering container images?

The real answer is, it depends!

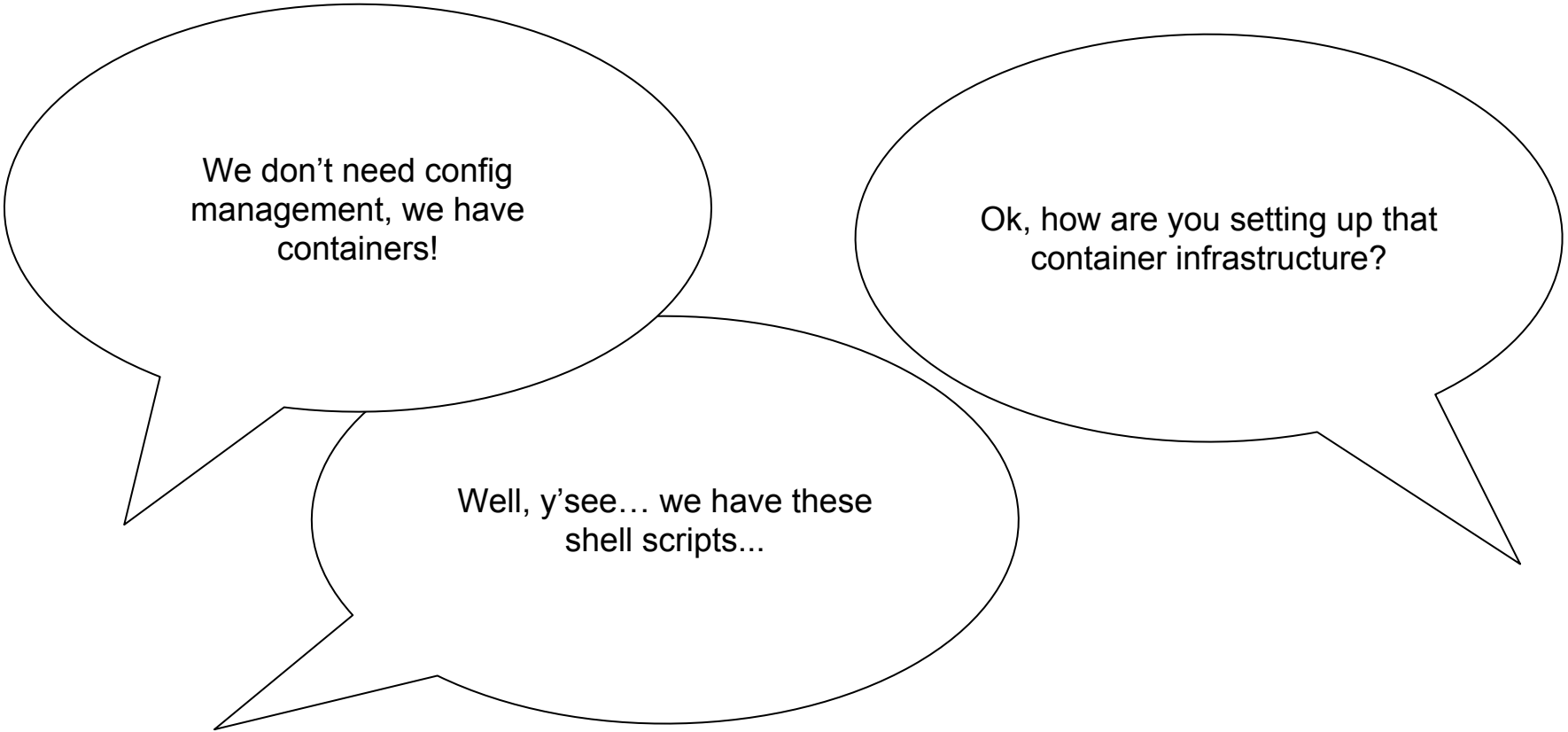
“The tired old debate of “Containers vs. Configuration Management” is rooted in fallacy. We’ve never tried to solve all of our problems with one technology before (You wouldn’t pull down your Java libraries with Puppet, or keep your load balancer config in Maven Central), so why do we have to start now?

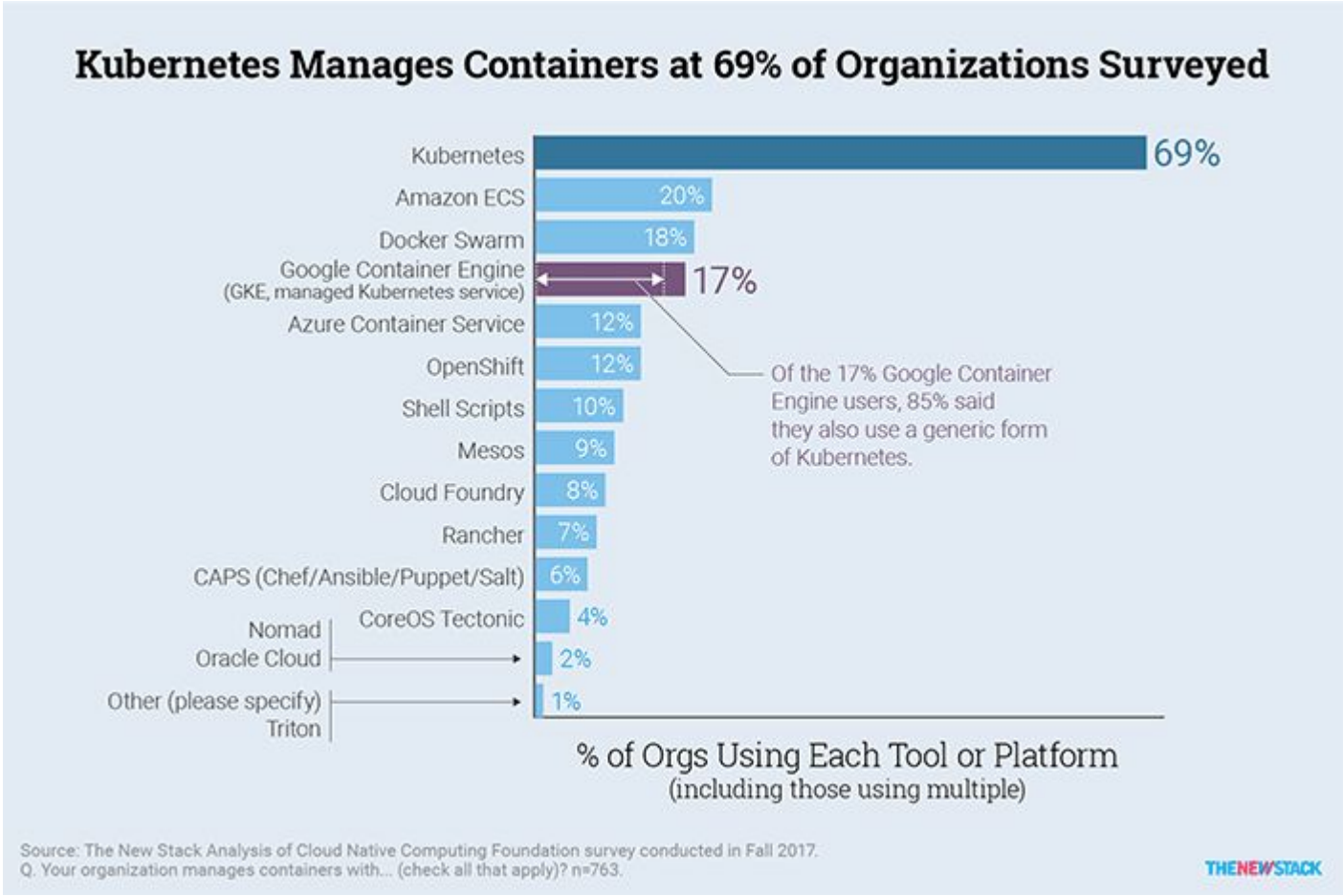
I think there is definitely room for system containers, application containers, and configuration management to coexist. Hopefully more work will be done to make these two great technologies play nicely together.”

- **Containers, Configuration Management, and The Right Tool for the Right Job, Ben Schwartz http://txt.fliglio.com/2015/07/containers-and-cfg-mgmt/**

There is still a place for config management in a container world

A conversation that makes me sad




Kubernetes Manages Containers at 69% of Organizations Surveyed
Kubernetes
69%
Amazon ECS
20%
Docker Swarm
18%
Google Container Engine
(GKE, managed Kubernetes service)
17%
Azure Container Service
12%
OpenShift
12%
Shell Scripts
10%
Mesos
9%
Cloud Foundry
8%
Rancher
7%
CAPS (Chef/Ansible/Puppet/Salt)
6%
CoreOS Tectonic
4%
Nomad
Oracle Cloud
2%
Other (please specify)
Triton
1%
Of the 17% Google Container Engine users, 85% said they also use a generic form of Kubernetes.
% of Orgs Using Each Tool or Platform
(including those using multiple)
Source: The New Stack Analysis of Cloud Native Computing Foundation survey conducted in Fall 2017.
Q. Your organization manages containers with... (check all that apply)? n=763.
THENEWSTACK

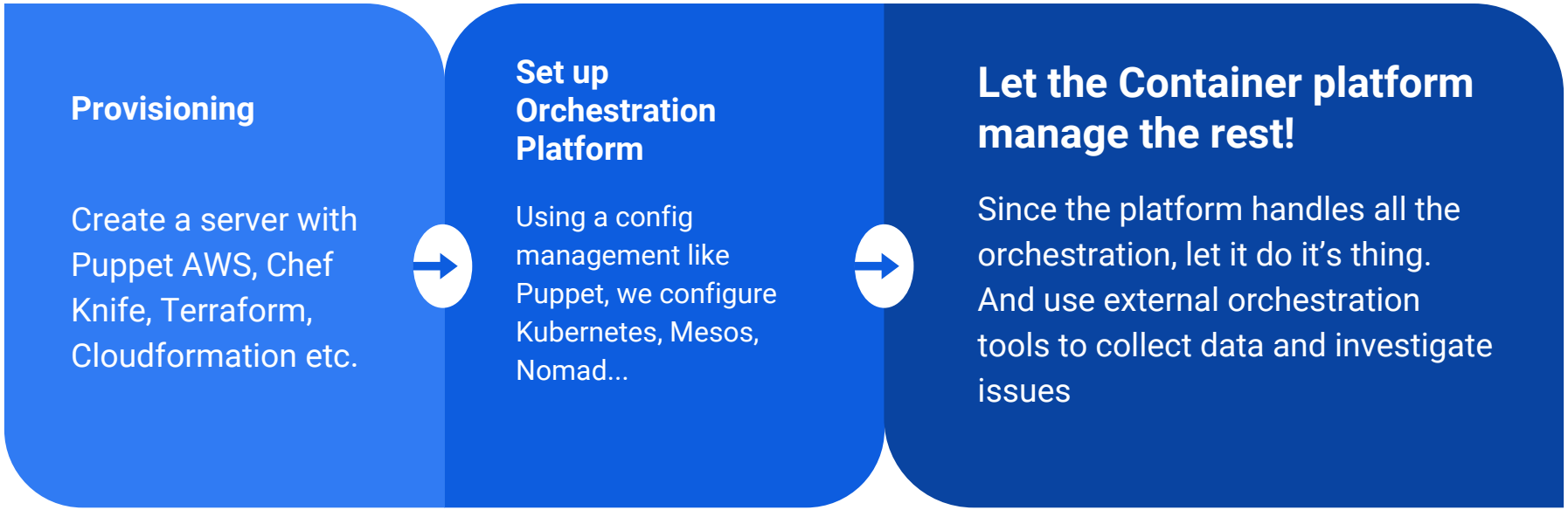
The host environment is not a container!

It still needs configuring and maintaining!

Can we please not reinvent the wheel?

We spent all this time automating all this stuff, let's use it for good!

One of the deployment models we've seen



Building containers with config management?

```
puppet docker build manifests/init.pp --image-name puppet/es --expose 9200 --cmd /docker-entrypoint.sh
```

https://github.com/puppetlabs/puppetlabs-image_build

Building containers with config management?

Managing Containers with Chef

Chef is a powerful automation platform for managing and configuring containers. Instead of roll-your-own management solutions, use Chef to build an industrial-strength workflow that combines the power of Chef with the convenience of containers.

GET CHEF REQUEST A DEMO

https://web.archive.org/web/20150709013111/https://www.chef.io/solutions/containers/

Ok, so let's summarize what we've covered

We've talked about a lot of things so let's recap

Provisioning is the creation of systems

But has some cross-over with it's bootstrapping to config management

Config management is managing state on those systems

But has some cross-over with deployment, especially when deploying applications as packages

Orchestration are actions performed at scale on your systems

And generally things that don’t fit into a state based model

Containers have changed the way that all 4 of these will be performed in the future

But I don’t think it has “replaced” all of these forms. It’s shifted things but not fully obsoleted things

Q&A

Want to know more?

- **What's Deployment Versus Provisioning Versus Orchestration?**
- **http://codefol.io/posts/deployment-versus-provisioning-versus-orchestration**
- **Why Configuration Management and Provisioning are Different** https://www.thoughtworks.com/insights/blog/why-configuration-management-and-provisioning-are-different
- **Configuration Management is an Antipattern** https://hackernoon.com/configuration-management-is-an-antipattern-e677e34be64c
  - NB: I disagree here, but it's a good example of the argument and makes good points
- **The Role of Configuration Management in a Containerized World** https://www.infoq.com/news/2015/12/containers-vs-config-mgmt