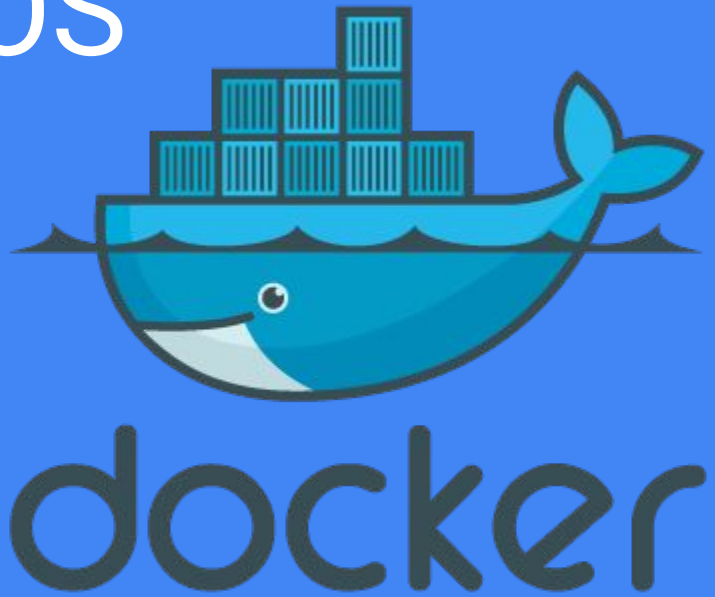
Docker & CoreOS

DevOps, Synergy, Buzzwords oh my!



What is Docker?

Open
Source
Containerization
Engine

- Docker site

How are Docker containers different from traditional VMs?

- Shared kernel (host machine)
- Lighter than VMs
- Process-level isolation

Docker Layered File System

Another Union File System(AUFS).

- AUFS is an implementation of union mounting.
- **Union mounting** is a way of combining multiple directories into one that appears to contain their combined contents

- TL;DR make multiple directories look like they are mounted in one directory
- *One of the reasons Docker is so lightweight is because of these layers. When you change a Docker image—for example, update an application to a new version— a new layer gets built. Thus, rather than replacing the whole image or entirely rebuilding, as you may do with a virtual machine, only that layer is added or updated - Docker official documentation*.

Docker Engine

- Daemon for managing Docker
  - API interactions occur here
  - Can expose daemon to the network for remote administrations.

Why would I want to use Docker?

- If you don’t do anything in kernel land, it’s much faster to spin up these containers.

- Deploy redundant web applications or services
- Microservice architecture with less resources than not using containers

- Keep your process in a safe space.

Use cases for Docker

- *“It worked on my machine.”*
  - Bullshit! It works in microcontainers on AWS now!
- *“AH I wanna try this new framework without ruining my current dev environment”*
  - Pop lock and dock it to avoid polluting your every-day workspace.
- Building for scale and availability
- *Microservices architecture and high availability*

Docker Registry

- Store/distribute docker images
- You own all your images

- Like Gitlab

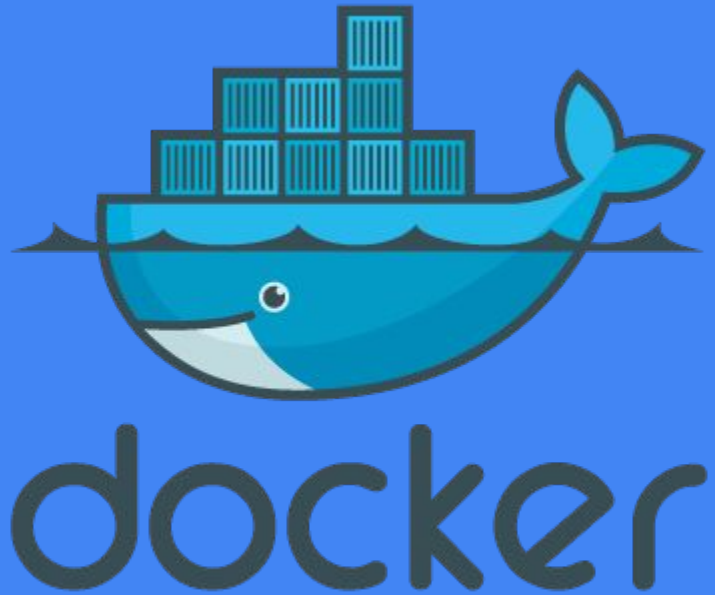
Docker Hub

- Cloud-based registry
- Collaborate with others
- Things can be public or private

- Like Github

Docker Demo

0-100%



How can I manage all my containers?

Containing your containers in an agile way

CoreOS

Fuck it, we wrote an operating system for containers

CoreOS Services

- Etcd
- Systemd
- Fleet

Etcd

Key => value

Key => value

Key => value

Key => value

RESTApi

SystemD

Continued Integration

- Treat dockers like services
- Continuously integrate things
  - Github
  - AWS
  - APIs

Fleet

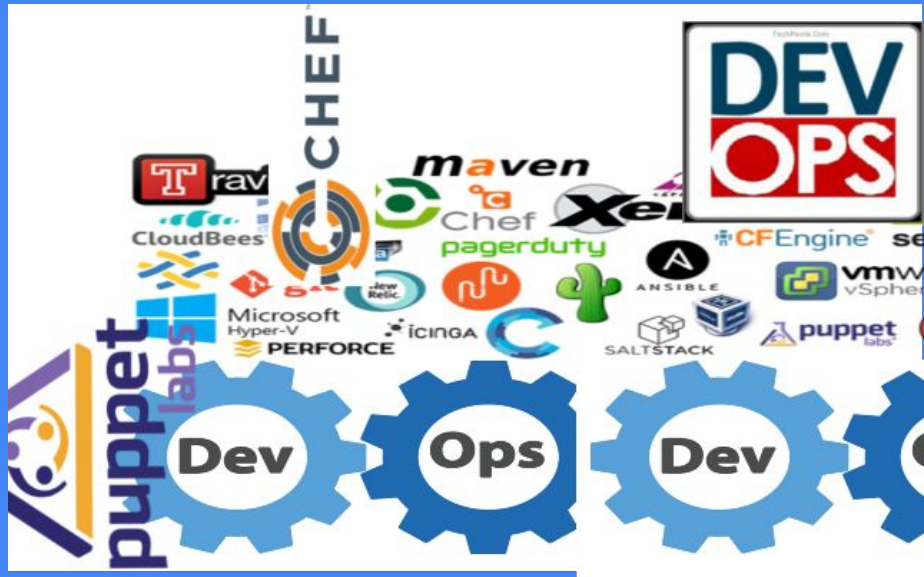
Fleet

```
smakam14@jungle1:~$ fleetctl --endpoint=http://172.17.8.101:2379  list-machines
MACHINE         IP              METADATA
12d30fe3...     172.17.8.103    -
91ab09f8...     172.17.8.101    -
949a5786...     172.17.8.102    -
```

Terraforming a path for your devops train of enlightenment

Automate everything.

choochoo



My qualifications

I read this:

51 Best DevOps Tools for #DevOps Engineers

On 02.18.15, In Cloud Computing, DevOps, by Drue Placette

DevOps, or the collaboration between development and operations teams, is an important component of companies today. Developing and implementing a DevOps culture helps to focus IT results and to save time and money as the gap between developers and IT operations teams closes. Just as the term and culture are new, so are many of the best DevOps tools these DevOps engineers use to do their jobs efficiently and productively.

To help your DevOps process, we have searched for the best tools for DevOps engineers. To make the cut, these tools must include relevant and useful features, support several languages and operating systems, and be known for their reliability and security. We have covered all of the bases with tools for logging, configuration management, security, monitoring, and automation. Please note, we have listed our 51 picks for the best DevOps tools for DevOps engineers in no particular order.

What is “DevOps”?

- Creating automation tools to automate all your work
- So you can create more tools to automate more stuff
  - That no one knew they needed automated until you showed them your tool
-
- Or “automating work so you can focus on the hard stuff”

Automation Service

DevOps Buzzword Bingo

B	I	N	G	O
Netflix	Firefighting	Puppet	Flow	Deming
ITIL	Community	Software-defining	Open Source	Continuous Delivery
Infrastructure as Code	Speed	ScriptRock	Chef	Facebook
Code	Collaboration	Automation	Adapt	Awesome
Kanban	Etsy	Agile	Revolution	Culture

A little backstory

- Ben and I run a bunch of stuff in AWS
- We’re busy
- Devops can take care of boring stuff so we can do fun stuff
- Therefore, I have taken a ride on the devops train of enlightenment

Some DevOps tools we currently use

- **SaltStack** - configuration management and mass deployment/upgrades
- **Nagios/AWS CloudWatch** - monitoring everything
- **OSSEC/Google Grr** - security monitoring and logging
- **Splunk** (soon to be ELK) - Log aggregation and analysis
- **Gitlab** - version control for custom tools
- **Modern Honey Network** - Honeypot management framework

And now, ***Terraform***

Terraform

Automate full stack deployments. Never do any real work. *Take the devops train to enlightenment.*

What is Terraform?

- Allows you to automate deployments of “hardware”.
- Works with multiple infrastructure platforms
- **Version control** for your infrastructure
  - Developers can’t have all the fun!

How Terraform is the most devops tool ever

- Terraform is basically a wrapper for API endpoints.
- You write an API for your service, expose the endpoint, and then write a Terraform ‘provider module’.
- Terraform wins.

Providers

- Providers are basically anything you can manage with Terraform.
- **Providers are responsible** for making Terraform work by providing APIs.
- Terraform supports three groups of providers:
  - **IaaS** (i.e. AWS, DigitalOcean, GCE, OpenStack)
  - **PaaS** (i.e. Heroku, CloudFoundry)
  - **SaaS** (i.e. Atlas, DNSimple, CloudFlare)

- Atlas
- AWS
- Azure
- Chef
- CloudFlare
- CloudStack
- Consul
- Datadog
- DigitalOcean
- DNSMadeEasy
- DNSimple
- Docker
- Dyn
- Google Cloud
- Heroku
- Mailgun
- MySQL
- OpenStack
- Packet
- PostgreSQL
- PowerDNS
- Rundeck
- StatusCake
- Template
- TLS
- Vsphere
- vCloud
- Terraform

Terraform Files

provider-variables.tf

```
variable "provider" {
    default = {
        access_key = "my-access-key"
        secret_key = "my-secret-key"
        region     = "us-east-1"
    }
}

variable "test3" {
        default = {
                ami = "ami-408c7f28"
                instance_type = "t1.micro"
                environment_name = "test_env"
        }
}
```

provider-config.tf

```
provider "aws" {
    access_key = "${var.provider.access_key}"
    secret_key = "${var.provider.secret_key}"
    region     = "${var.provider.region}"
}

resource "aws_instance" "test3" {
        ami = "${var.test3.ami}"
        instance_type = "${var.test3.instance_type}"
        tags = {
                Name = "${environment_name}-magic"
        }
}
```

Execute your changes

~$ terraform plan # shows you what's going to happen

~$ terraform apply # tells terraform to apply the changes

~$ terraform show # shows you what's happened

“Configuration management” with Terraform

- Terraform uses “provisioners” to do configuration management
- Supported provisioners:
  - **Chef** - causes a chef client to be installed/run
  - **Connection** - Defines how Terraform can connect to a host (ssh, WinRM, etc)
  - **File** - copy files or directories from Terraform server to new host
  - **Local-exec** - runs a command on the Terraform server after a resource is created
  - **Remote-exec** - runs a command on the resource after it’s been configured
  - **Null_resource** - Configure provisioners that have custom triggers

Install vim with Terraform (remote-exec)

```
provisioner "remote-exec" {
        inline = [
                "sudo apt-get update",
                "sudo apt-get install vim --yes"
        ]
}
```

This is “bad”. Have it install Salt/Ansible/whatever instead.

“Version Control” with Terraform

- “Technically” you need Atlas, HashiCorp’s magical stack
- Atlas let’s you visually make and collaborate on infrastructure changes
  - And of course it costs money
- We just use git.
  - Make a new branch for every major change in infrastructure
  - Since Terraform just needs files to make changes, this works pretty well

How are we using Terraform?

- Rotating IP addresses
- “Actual” one click deployments
- Having a “backup” of infrastructure outside of EC2