

# Introduction to Terraform

# Why Terraform?

- ◆ Terraform is an open source Infrastructure as Code (IaC) developed by Hashicorp to automate the provisioning of resources across cloud providers
- ◆ All cloud providers have some form of IaC available
  - AWS has CloudFormation
  - Azure has Azure Resource Manager (ARM), etc.
- ◆ Terraform works across *all* cloud providers (AWS, Azure, Google, AliBaba, etc.)
  - It provides a common vendor agnostic syntax and application structure for all cloud vendors
  - Most organizations nowadays are working with multiple cloud vendors
  - Having a common tool and in-house expertise in the tool is highly cost-efficient and operationally effective

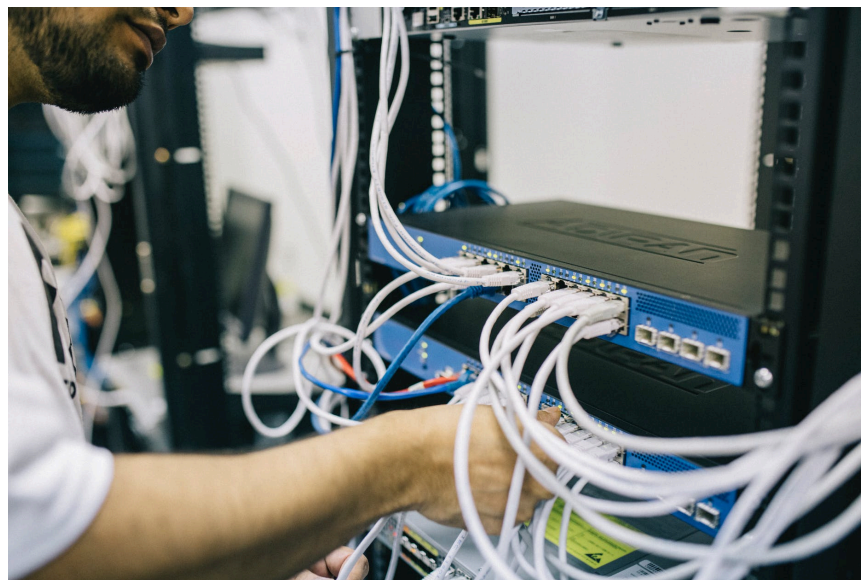


# What We Will Discuss

- ◆ How we got to where we are: the rise of virtualization, the cloud and IaC
- ◆ What exactly *Infrastructure as Code* is
- ◆ The benefits of IaC and how it relates to DevOps
- ◆ What Terraform does and how it works
- ◆ How Terraform compares to other IaC tools like Ansible and Puppet

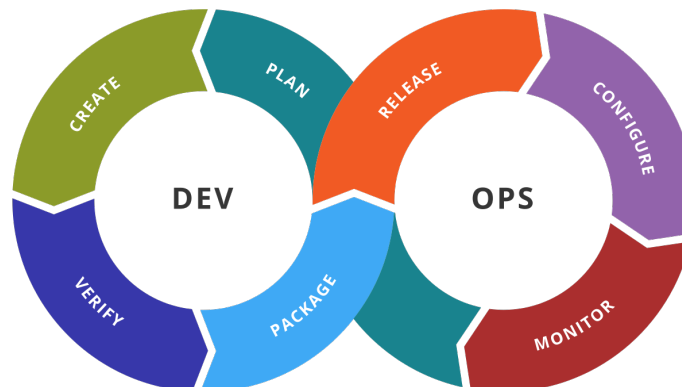
# On Premises Physical Infrastructure

- ◆ On premises (on-prem) physical infrastructure refers to:
  - Physical computers, servers and other devices
  - All associated hardware: cables, routers, switches, disks, etc.
  - Often organized into computer centers or server farms
- ◆ Servers are provisioned with the software needed by the users
  - Hardware and installed software collectively is *infrastructure*
- ◆ The *Dev* team works with software, the *Ops* team with infrastructure
  - The two groups have different tools, cultures and processes



# The Rise of DevOps

- ◆ Virtualization was a game changer
  - On-prem physical servers can be replaced by *Virtual Machines*
  - This still requires physical hosts (for example VMWare)
  - Some organizations use a mix of physical hosts and virtual hosts
- ◆ Cloud computing eliminates the need for any on-prem physical hardware
  - All the infrastructure is virtual
  - The cloud provider handles all the hardware
- ◆ DevOps is the merging of the roles of Dev and Ops
  - Developers write code, Ops now write code
  - DevOps integrates the two roles through common tools
  - Enables *continuous integration & continuous deployment* (CI/CD)



# With DevOps

- ◆ Nordstrom
  - number of features it delivered per month +100%
  - reduce defects by 50%
  - reduce lead times by 60%
- ◆ HP's LaserJet Firmware
  - time its developers spent on developing new features went from 5% to 40%
  - overall development costs were reduced by 40%
- ◆ Four core values in the DevOps movement
  - culture, automation, measurement, and sharing (CAMS)
- ◆ DevOps is now a **lifestyle**
- ◆ In research - ResearchOps is next

# ResearchOps

- ◆ IBM paper
  - "ResearchOps: The case for DevOps in scientific applications"
  - Real-life projects at the IBM Research Brazil Lab
  - May just as well apply to other research institutions
  - [Paper link](#)

# Defining Infrastructure as Code?

- ◆ In a cloud or virtual environment, infrastructure *is* code
  - Code is run to create virtual resources
  - IaaS refers to virtual networks of virtual devices
- ◆ The code we write to create and manage this virtual infrastructure **is** source code
  - Therefore, it can be managed using the tools developers use for managing source code
  - IaC code can be developed using the same types of methods programmers use for their source code



# Types of IaC Tools

- ◆ There are five broad categories of IAC tools:
  - Ad hoc scripts - tools to manage infrastructure on the fly
  - Configuration management tools - install and maintain software on physical and virtual hosts
  - Server templating tools and containerization tools
  - Orchestration tools
  - Provisioning tools

# Ad Hoc Scripts

- ◆ Procedural code written to manage a virtual resource
- ◆ Example: AWS CLI commands to create a security group and EC2 instance.

```
(base) [rod@exgnosis Terraform]$ aws ec2 create-security-group --group-name demo1 --description "Ad hoc" --profile dev1
{
  "GroupId": "sg-05653d27658bf1427"
}
(base) [rod@exgnosis Terraform]$ aws ec2 run-instances --image-id ami-048f6ed62451373d9 --count 1 --instance-type t2.micro --key-name TerraformKeys --security-group-ids sg-05653d27658bf1427 --profile dev1
{
  "Groups": [],
  "Instances": [
    {
      "AmiLaunchIndex": 0,
      "ImageId": "ami-048f6ed62451373d9",
      "InstanceId": "i-0fe60e9131a349f14",
      "InstanceType": "t2.micro",
      "KeyName": "TerraformKeys",
      "LaunchTime": "2021-05-03T22:43:47+00:00",
      "Monitoring": {
        "State": "disabled"
      },
      "Placement": {
        "AvailabilityZone": "us-east-1e",
        "GroupName": "",
        "Tenancy": "default"
      },
      "PrivateDnsName": "ip-172-31-62-218.ec2.internal",
      "PrivateIpAddress": "172.31.62.218",
      "ProductCodes": [],
      "PublicDnsName": "",
      "State": {
        "Code": 0,
        "Name": "pending"
      },
      "StateTransitionReason": "",
      "SubnetId": "subnet-85fac3bb",
    }
  ]
}
```

# Configuration Management Tools

- ◆ CM tools manage software installation and configuration on both virtual and physical hosts
  - Most popular tools: Chef, Puppet, Ansible, and SaltStack
- ◆ Example of an Ansible script (called a playbook)

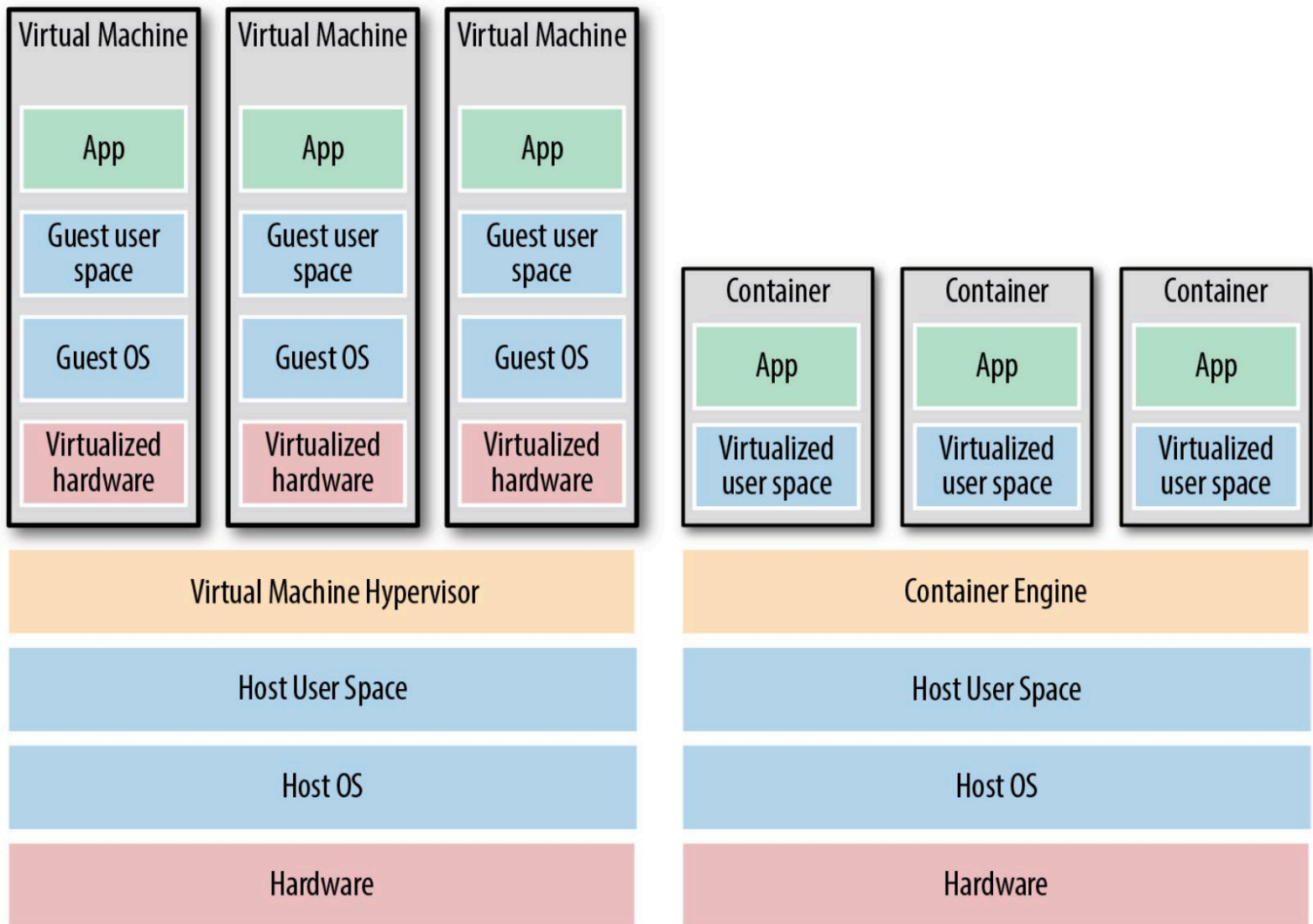
```
---  
# Play1 - WebServer related tasks  
- name: Play Web - Create apache directories and username in web servers  
  hosts: webservers  
  become: yes  
  become_user: root  
  tasks:  
    - name: create username apacheadm  
      user:  
        name: apacheadm  
        group: users,admin  
        shell: /bin/bash  
        home: /home/weblogic  
  
    - name: install httpd  
      yum:  
        name: httpd  
        state: installed
```

# Server Templating Tools

- ◆ Tools to handle packaging, configuration and deployment of VMs and containers

```
1  #Dockerfile vars
2  alpver=3.12
3  kctlver=1.18.0
4
5  #vars
6  IMAGENAME=my_kubectl
7  REPO=my.registry
8  IMAGEFULLNAME=${REPO}/${IMAGENAME}:${KUBECTL_VERSION}
9
10 .PHONY: help build push all
11
12 help:
13     @echo "Makefile arguments:"
14     @echo ""
15     @echo "alpver - Alpine Version"
16     @echo "kctlver - kubectl version"
17     @echo ""
18     @echo "Makefile commands:"
19     @echo "build"
20     @echo "push"
21     @echo "all"
22
23 .DEFAULT_GOAL := all
24
25 build:
26     @docker build --pull --build-arg ALP_VER=${alpver} --build-arg KCTL_VER=${kctlver}
27
28 push:
29     @docker push ${IMAGEFULLNAME}
30
31 all: build push
```

# Images and Containers



# Virtual Machines vs Containers

## ◆ Virtual machines

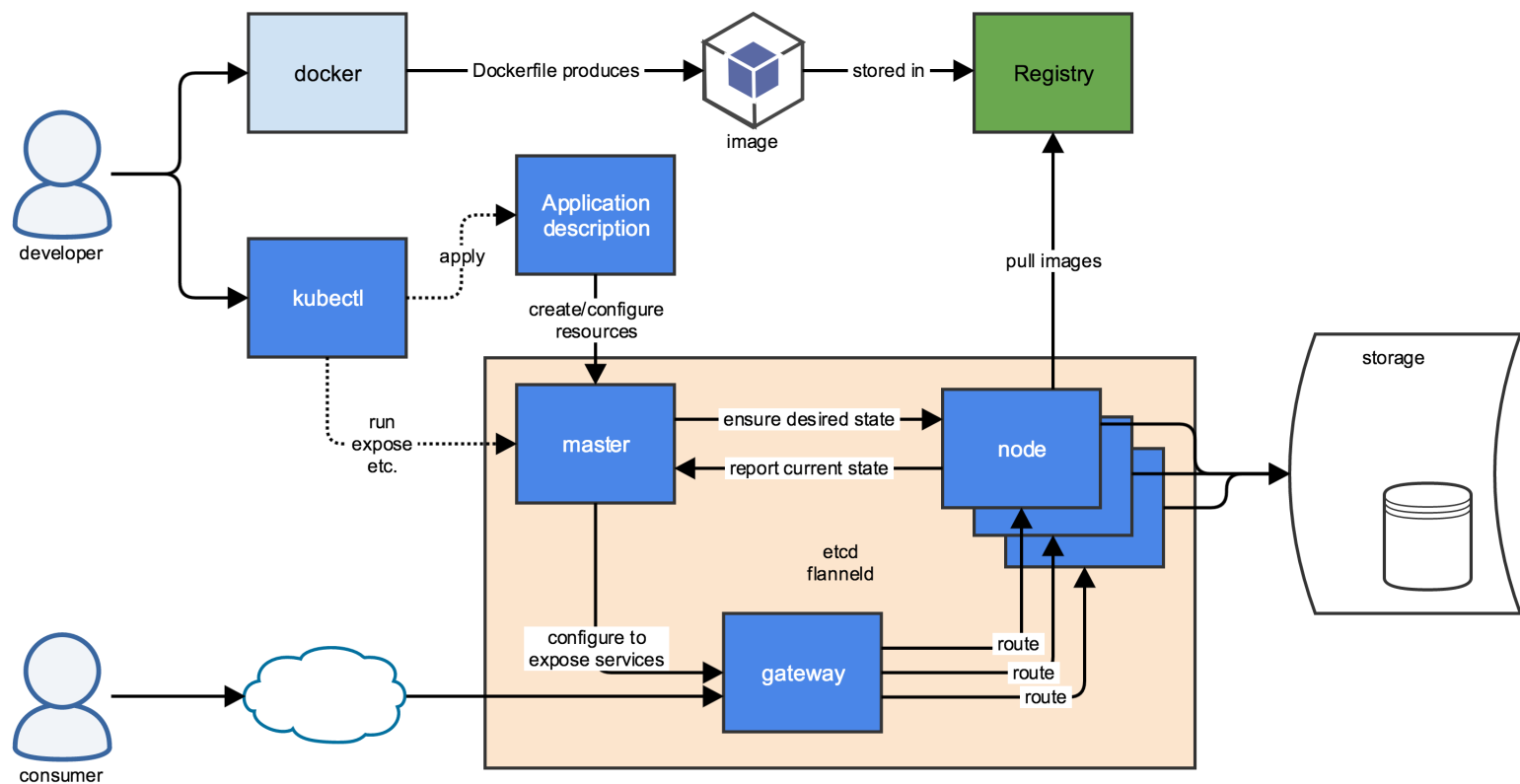
- A virtual machine (VM) emulates an entire computer system, including the hardware. You run a hypervisor, such as VMWare, VirtualBox, or Parallels, to virtualize (i.e., simulate) the underlying CPU, memory, hard drive, and networking.
- Benefit: complete isolation
- Drawback: waste of resources
- You can define VM images as code using tools such as Packer and Vagrant.

## ◆ Containers

- A container emulates the user space of an OS. You run a container engine, such as Docker, CoreOS rkt, or cri-o, to create isolated processes, memory, mount points, and networking.
- Benefit: you run on top of the container engine can see only its own user space
- Benefit: of the containers running on a single server share, milliseconds boot time

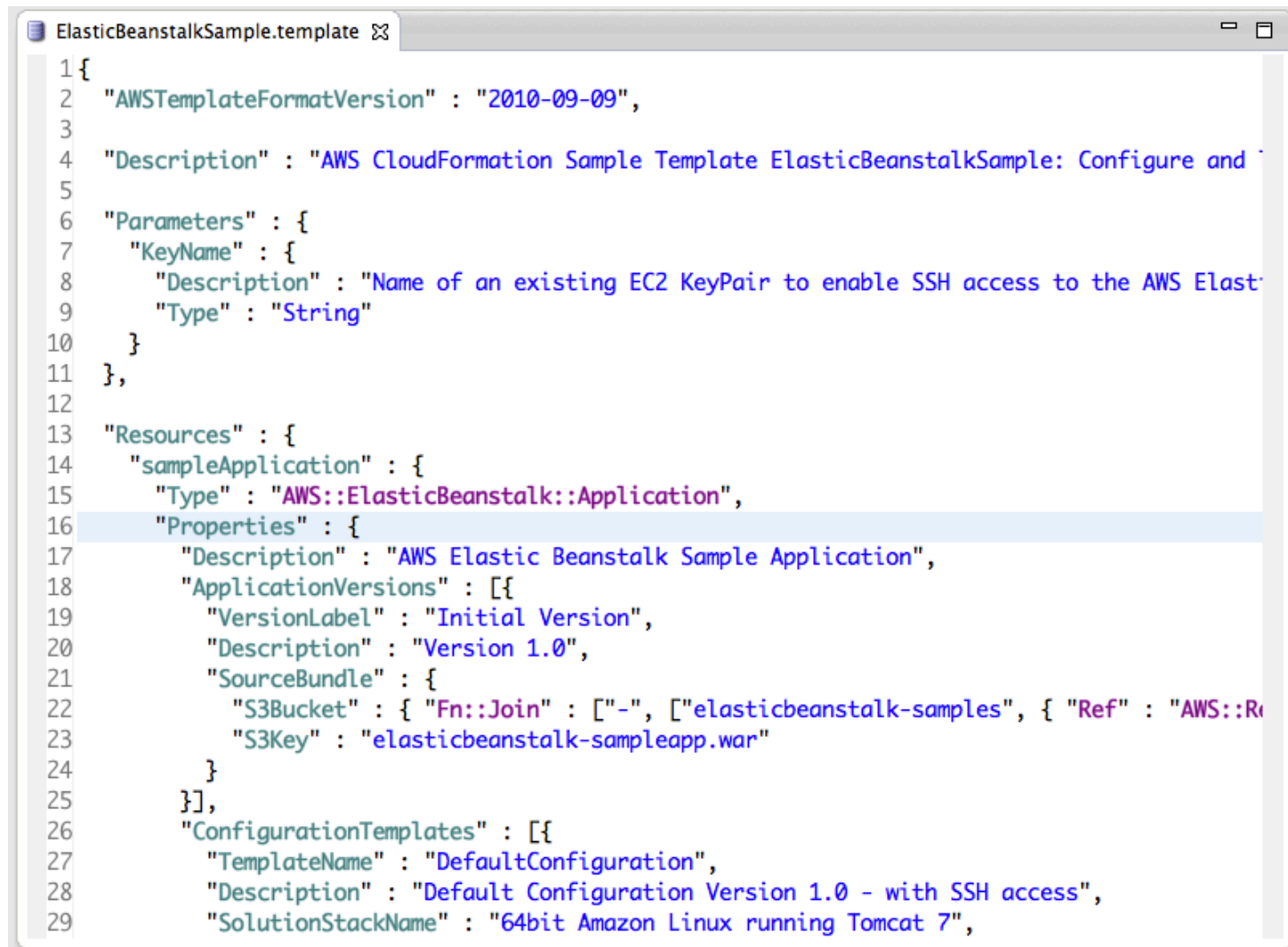
# Orchestration Tools

- ◆ Designed run and coordinate multiple tasks (usually containers) running on multiple hosts
  - Need to keep the overall distributed processing synchronized and robust
  - These are often thought of as "swarms" of hosts/containers that need to be orchestrated



# Provisioning Tools

- ◆ Refers to creating virtual resources from some form of text template, usually JSon



```
1 {
2   "AWSTemplateFormatVersion" : "2010-09-09",
3
4   "Description" : "AWS CloudFormation Sample Template ElasticBeanstalkSample: Configure and
5
6   "Parameters" : {
7     "KeyName" : {
8       "Description" : "Name of an existing EC2 KeyPair to enable SSH access to the AWS Elast
9       "Type" : "String"
10    }
11  },
12
13  "Resources" : {
14    "sampleApplication" : {
15      "Type" : "AWS::ElasticBeanstalk::Application",
16      "Properties" : {
17        "Description" : "AWS Elastic Beanstalk Sample Application",
18        "ApplicationVersions" : [{
19          "VersionLabel" : "Initial Version",
20          "Description" : "Version 1.0",
21          "SourceBundle" : {
22            "S3Bucket" : { "Fn::Join" : ["-", ["elasticbeanstalk-samples", { "Ref" : "AWS::R
23            "S3Key" : "elasticbeanstalk-sampleapp.war"
24          }
25        }],
26        "ConfigurationTemplates" : [{
27          "TemplateName" : "DefaultConfiguration",
28          "Description" : "Default Configuration Version 1.0 - with SSH access",
29          "SolutionStackName" : "64bit Amazon Linux running Tomcat 7",
```



# Why Infrastructure as Code?

- ◆ Speed and simplicity
  - Entire deployments can be set up or torn down by running a script
- ◆ Configuration consistency
  - Identical copies of configurations can be created for testing or development
- ◆ Minimization of risk
  - Reduces human procedural errors
  - Allows for testing, reviews and other quality measures
- ◆ Increased efficiency in software development
  - Infrastructure is not a bottleneck
  - Resources are available as needed
- ◆ Cost savings
  - Grunt work is automated so people can do the important stuff

# The Benefits of Infrastructure as Code

- ◆ Self-service
  - Infrastructure deployment with scripts does not rely on an administrator
- ◆ Speed and safety
  - Infrastructure is deployment and updated faster and with fewer errors
- ◆ Documentation
  - The IaC source files *are* infrastructure documentation
- ◆ Version control
  - Previous deployments can be maintained in source control for regression or audit need, or to satisfy regulatory requirements
- ◆ Validation
  - For every single change, code reviews and dynamic testing can be performed
- ◆ Reuse
  - New infrastructure deployments can be derived quickly from previous deployments

# How Terraform Works

- ◆ Terraform is an open source tool created by HashiCorp
  - Written in the Go programming language
- ◆ A single utility called `_terraform_` is downloaded to a local machine
- ◆ The local *terraform* utility
- ◆ Identifies the plug-ins needed to interact with a cloud provider
- ◆ Downloads and installs the required plugins
- ◆ The *terraform* utility and the plug-ins make API calls on your behalf to one or more providers
  - AWS
  - Azure
  - Google Cloud
  - DigitalOcean
  - OpenStack, and more



# Terraform and Other Tools

	Source	Cloud	Type	Infrastructure	Language	Agent	Master	Community	Maturity
Chef	Open	All	Config Mgmt	Mutable	Procedural	Yes	Yes	Large	High
Puppet	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	High
Ansible	Open	All	Config Mgmt	Mutable	Procedural	No	No	Huge	Medium
SaltStack	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	Medium
CloudFormation	Closed	AWS	Provisioning	Immutable	Declarative	No	No	Small	Medium
Heat	Open	All	Provisioning	Immutable	Declarative	No	No	Small	Low
Terraform	Open	All	Provisioning	Immutable	Declarative	No	No	Huge	Low

# Lab 1-1

- ◆ In Lab 1-1 you will configure and test your terraform set-up
- ◆ The steps in the lab are:
  - Confirm or set up your AWS account ID for the course
  - Install the AWS command line interface
  - Install your AWS developer credentials on your local machine
  - Install the terraform executable
  - Write and run the "Hello World" example code