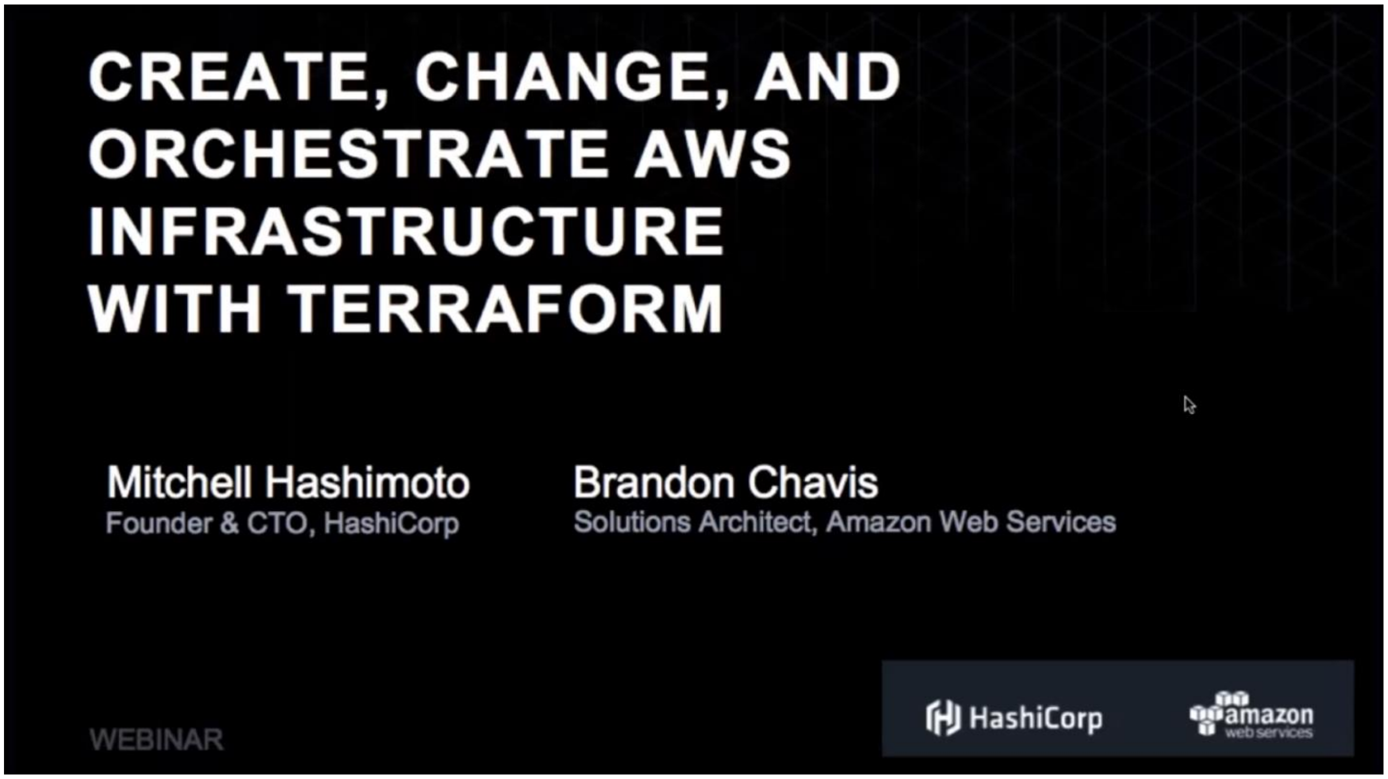


CREATE, CHANGE, AND ORCHESTRATE AWS INFRASTRUCTURE WITH TERRAFORM

Mitchell Hashimoto
Founder & CTO, HashiCorp

Brandon Chavis
Solutions Architect, Amazon Web Services

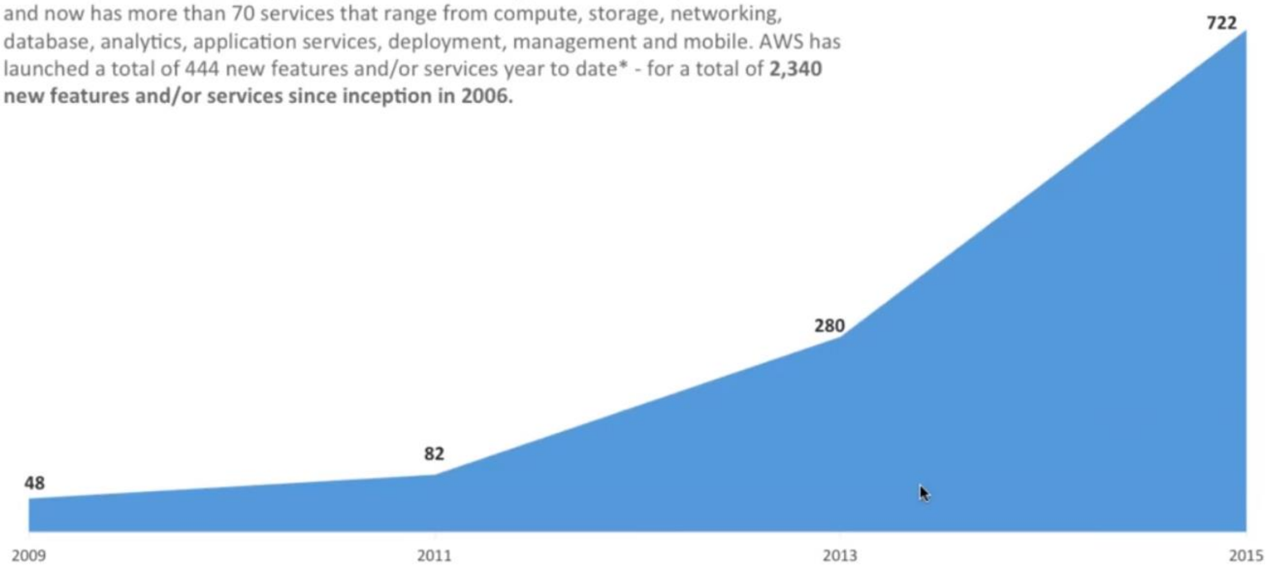
WEBINAR

HashiCorp

amazon web services

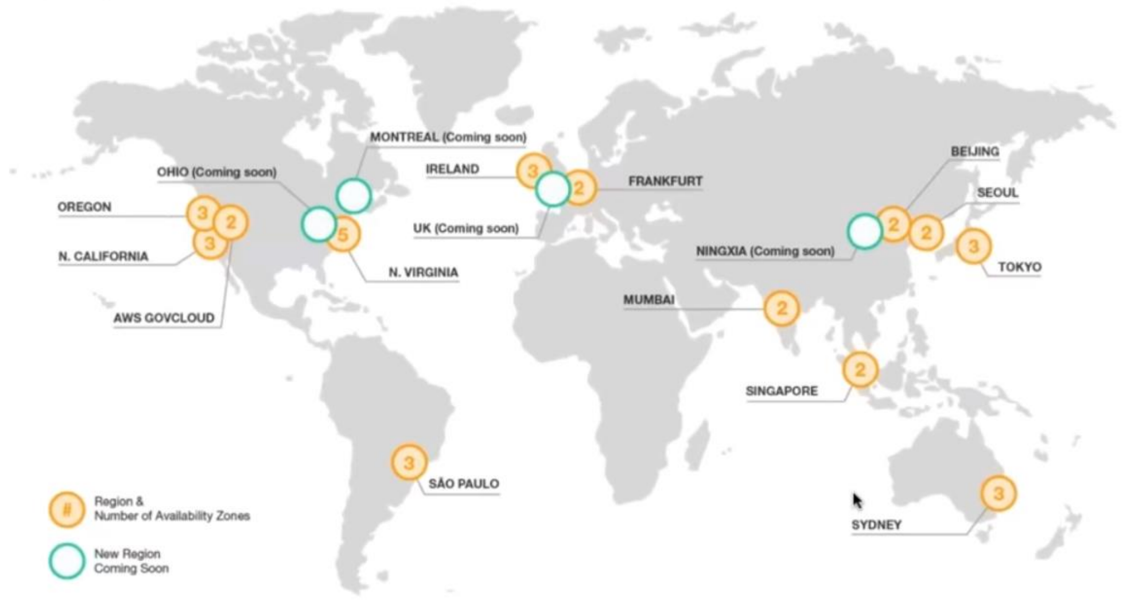
AWS' History of Innovation

AWS has been continually expanding its' services to support virtually any cloud workload and now has more than 70 services that range from compute, storage, networking, database, analytics, application services, deployment, management and mobile. AWS has launched a total of 444 new features and/or services year to date* - for a total of **2,340 new features and/or services since inception in 2006.**

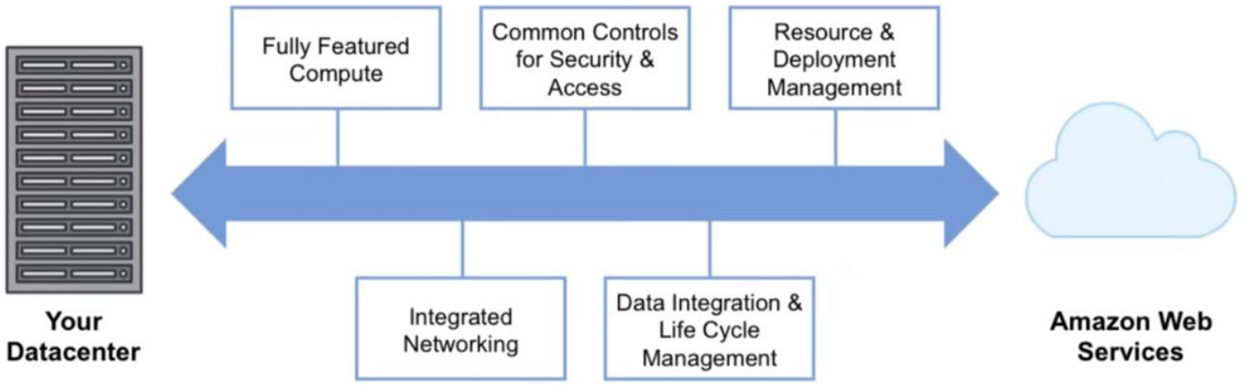


*As of 11 July 2016

Deploy faster wherever you like

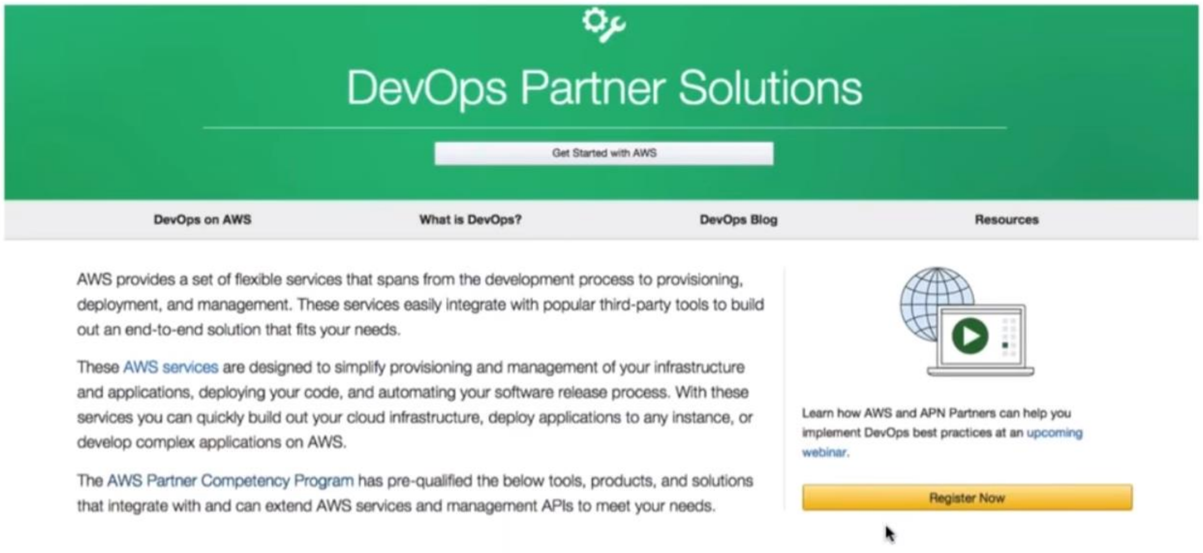


Deploy however you like


Fully Featured Compute
Common Controls for Security & Access
Resource & Deployment Management
Your Datacenter
Integrated Networking
Data Integration & Life Cycle Management
Amazon Web Services

Flexible hybrid options

HashiCorp can help



AWS provides a set of flexible services that spans from the development process to provisioning, deployment, and management. These services easily integrate with popular third-party tools to build out an end-to-end solution that fits your needs.

These AWS services are designed to simplify provisioning and management of your infrastructure and applications, deploying your code, and automating your software release process. With these services you can quickly build out your cloud infrastructure, deploy applications to any instance, or develop complex applications on AWS.

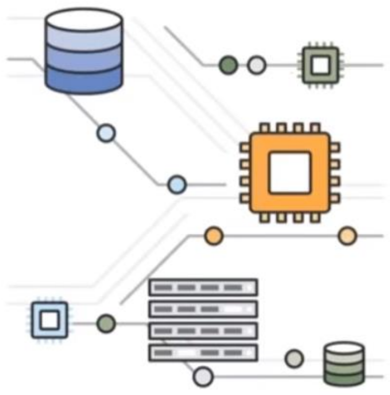
The AWS Partner Competency Program has pre-qualified the below tools, products, and solutions that integrate with and can extend AWS services and management APIs to meet your needs.

Learn how AWS and APN Partners can help you implement DevOps best practices at an upcoming webinar.

https://aws.amazon.com/devops/partner-solutions/

HashiCorp can help

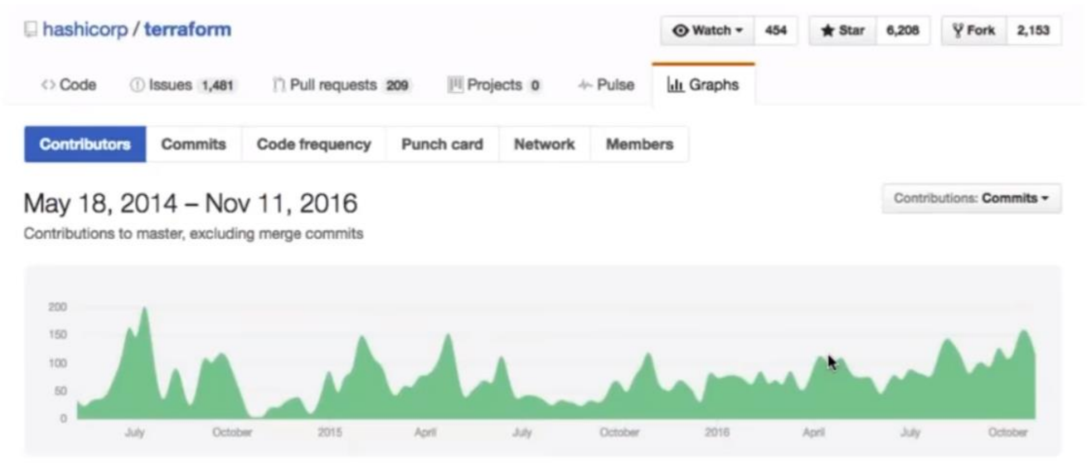
- HashiCorp is a DevOps Competency Partner: Vetted Solutions that meet AWS best practices
- AWS Customers use HashiCorp tools to provision, secure, and run infrastructure both in AWS and on-premise



amazon web services | Partner Network

HashiCorp + AWS Community

- Terraform development happens on Github
- Keep up with AWS: New features added by both the HashiCorp team and AWS Community



May 18, 2014 – Nov 11, 2016

Contributions to master, excluding merge commits

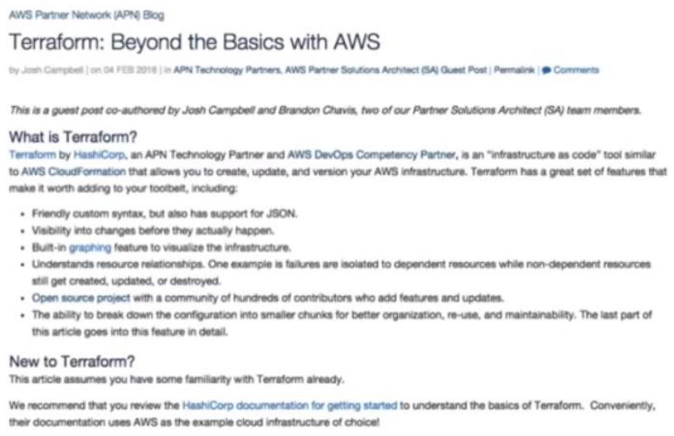
HashiCorp + AWS collaboration

- **Vagrant** - Feb 2013 (vagrant-aws plugin released)
- **Packer** - June 2013 (AMI builder)
- **Terraform** - July 2014 (support for many AWS resources)
- **Otto** - Sept 2015 (creates supporting AWS infrastructure during "deploy" step)
- **Vault** - Sept 2015 (AWS secret backend that uses IAM)
- **Tell us what to build!**

https://www.vaultproject.io/docs/auth/aws-ec2.html

HashiCorp + AWS collaboration

- Terraform on AWS Blog post:
  - Handling credentials, using IAM roles
  - Logically separating your TF modules to correlate to your architecture
  - Using Variables and Modules



AWS Partner Network (APN) Blog

Terraform: Beyond the Basics with AWS

by Josh Campbell | on 04 FEB 2016 | in APN Technology Partners, AWS Partner Solutions Architect (SA) Guest Post | Permalink | Comments

This is a guest post co-authored by Josh Campbell and Brandon Chavis, two of our Partner Solutions Architect (SA) team members.

What is Terraform?

Terraform by HashiCorp, an APN Technology Partner and AWS DevOps Competency Partner, is an "infrastructure as code" tool similar to AWS CloudFormation that allows you to create, update, and version your AWS infrastructure. Terraform has a great set of features that make it worth adding to your toolbelt, including:

- Friendly custom syntax, but also has support for JSON.
- Visibility into changes before they actually happen.
- Built-in graphing feature to visualize the infrastructure.
- Understands resource relationships. One example is failures are isolated to dependent resources while non-dependent resources still get created, updated, or destroyed.
- Open source project with a community of hundreds of contributors who add features and updates.
- The ability to break down the configuration into smaller chunks for better organization, re-use, and maintainability. The last part of this article goes into this feature in detail.

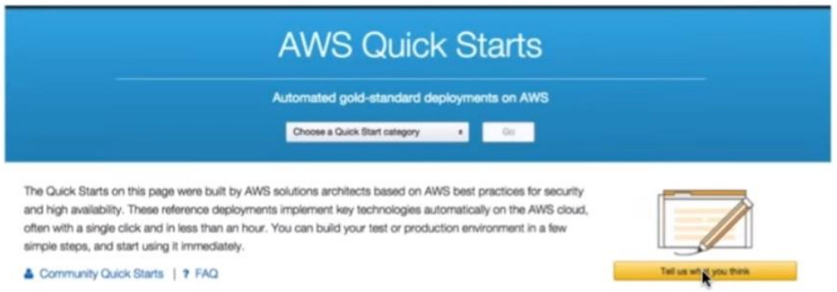
New to Terraform?

This article assumes you have some familiarity with Terraform already.

We recommend that you review the HashiCorp documentation for getting started to understand the basics of Terraform. Conveniently, their documentation uses AWS as the example cloud infrastructure of choice!

New!

- AWS Quick Starts for Vault and Consul: https://aws.amazon.com/quickstart/
- Open source reference deployments of partner products on AWS
- Jointly developed between AWS and HashiCorp



AWS Quick Starts

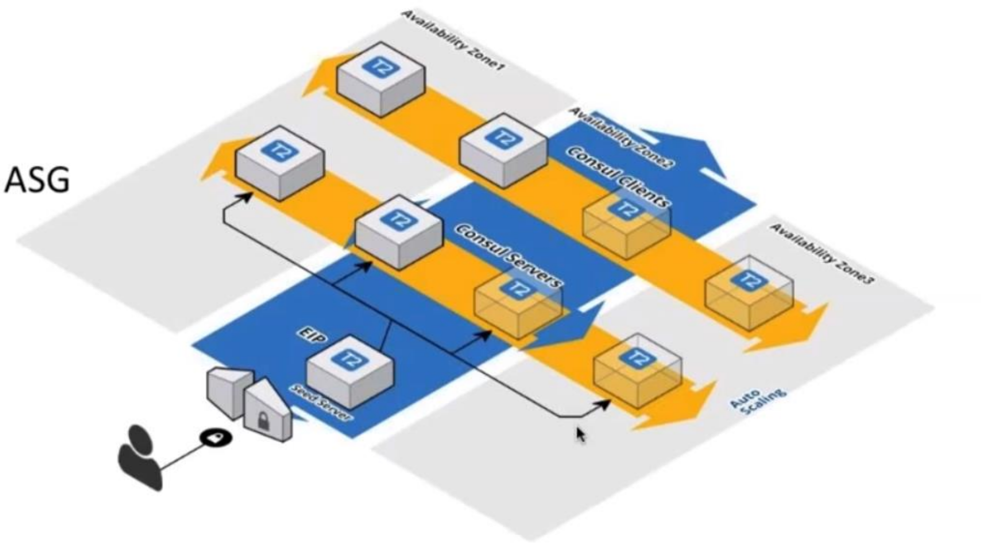
Automated gold-standard deployments on AWS

The Quick Starts on this page were built by AWS solutions architects based on AWS best practices for security and high availability. These reference deployments implement key technologies automatically on the AWS cloud, often with a single click and in less than an hour. You can build your test or production environment in a few simple steps, and start using it immediately.

Community Quick Starts | ? FAQ

Consul Quick Start

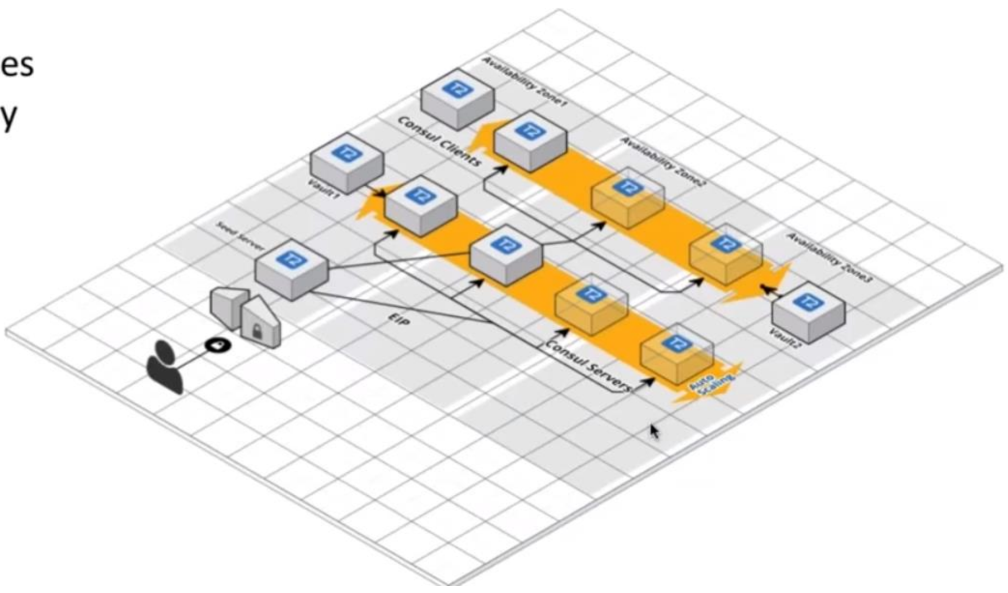
- Deploys:
- VPC
- Consul Servers
- Consul Web UI
- Consul Clients in an ASG
- DNSMasq

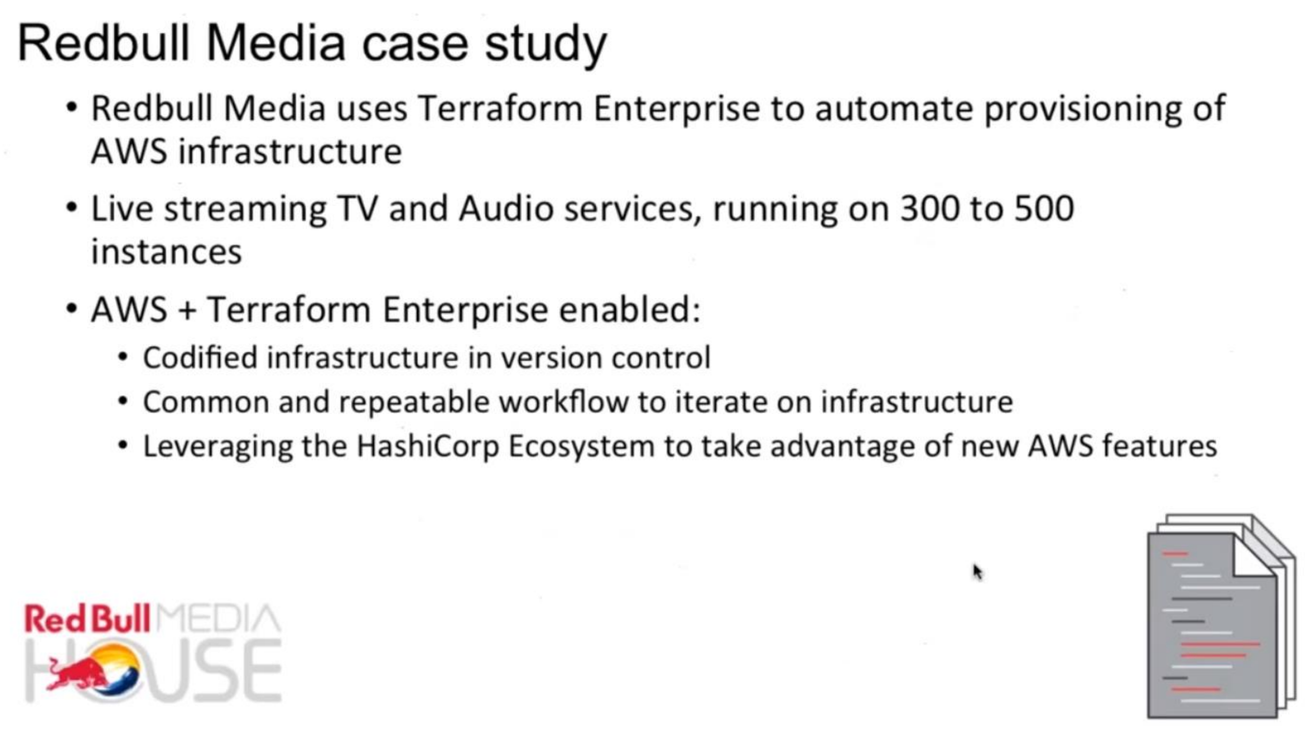


This can be deployed into your existing VPC in a public or private subnet

Vault Quick Start

- Deploys:
- Two Vault instances
- EC2 Auto Recovery
- Logging
- Monitoring





Redbull Media case study

- Redbull Media uses Terraform Enterprise to automate provisioning of AWS infrastructure
- Live streaming TV and Audio services, running on 300 to 500 instances
- AWS + Terraform Enterprise enabled:
  - Codified infrastructure in version control
  - Common and repeatable workflow to iterate on infrastructure
  - Leveraging the HashiCorp Ecosystem to take advantage of new AWS features

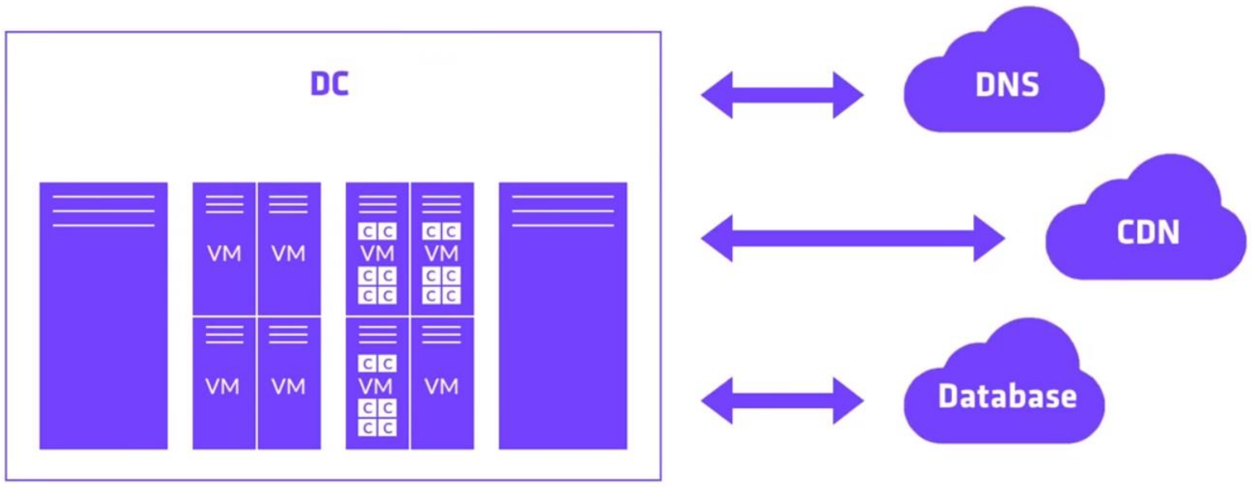
Red Bull MEDIA HOUSE

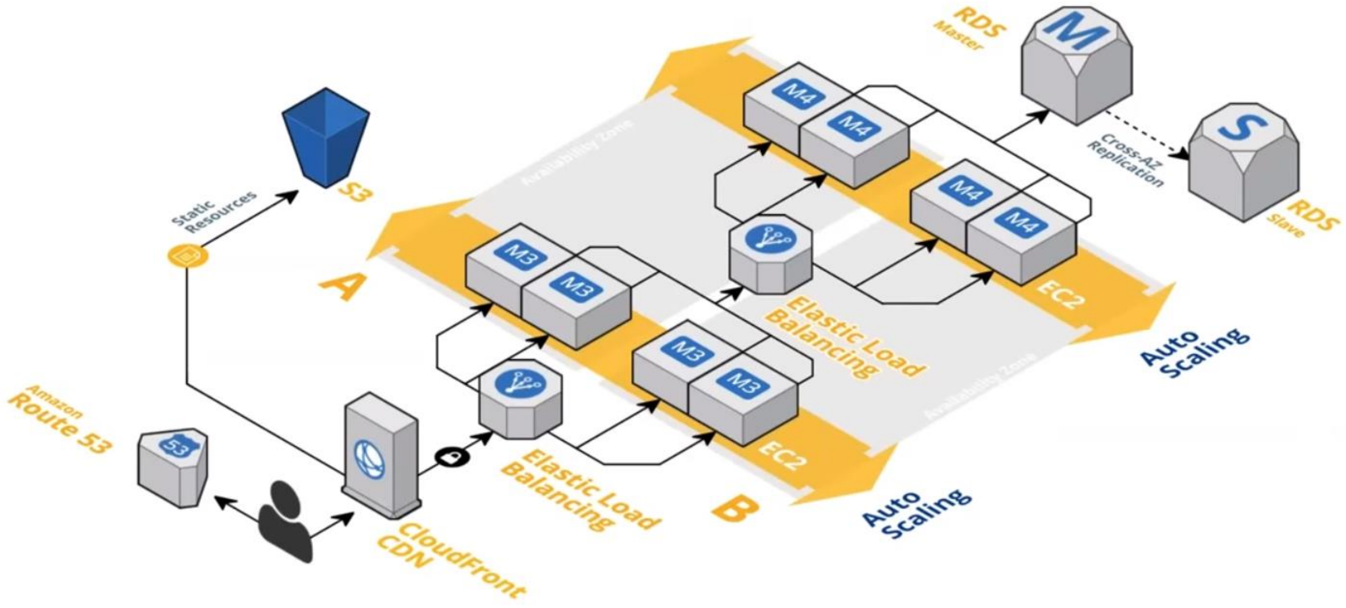
Terraform

HashiCorp

THE PROBLEM

RISING DATACENTER COMPLEXITY





HashiCorp

The Problem

"Datacenter" is a complex, multi-provider problem

Minimum infrastructure for deployment is high

Manual creation is too time intensive

TERRAFORM

Terraform's Goals

Unify the view of resources using infrastructure as code

Support the modern data center (IaaS, PaaS, SaaS)

Expose a way to safely and predictably change infrastructure

Provide a workflow that is technology agnostic

Manage anything with an API

Terraform vs. Other Tools

Provides a high-level description of infrastructure (IaC)

Allows for composition and combination

Supports parallel management of resources (graph, fast)

Separates planning from execution (dry-run)

State of Terraform

Open Source!

First Release July 28, 2014 (~2, 2.5 years old)

Over 700 contributors

Over 6,200 GitHub stars

One release every ~2 weeks

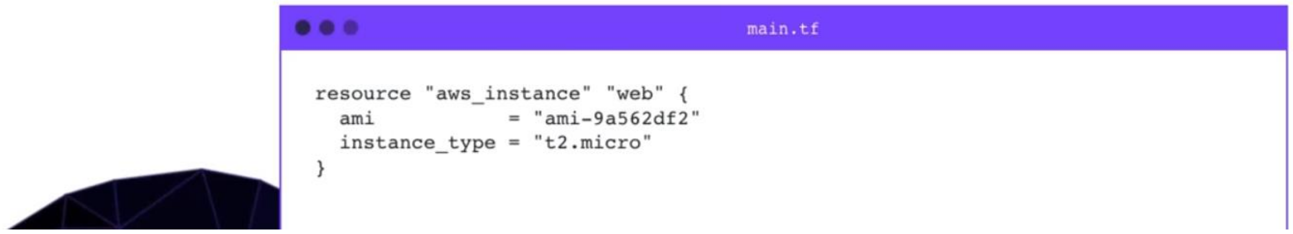
The Power of Community

Average time between AWS announcement and pull request: ~30 minutes

Prioritized releases (outside of normal ~2 week release cycle) when new features are released.

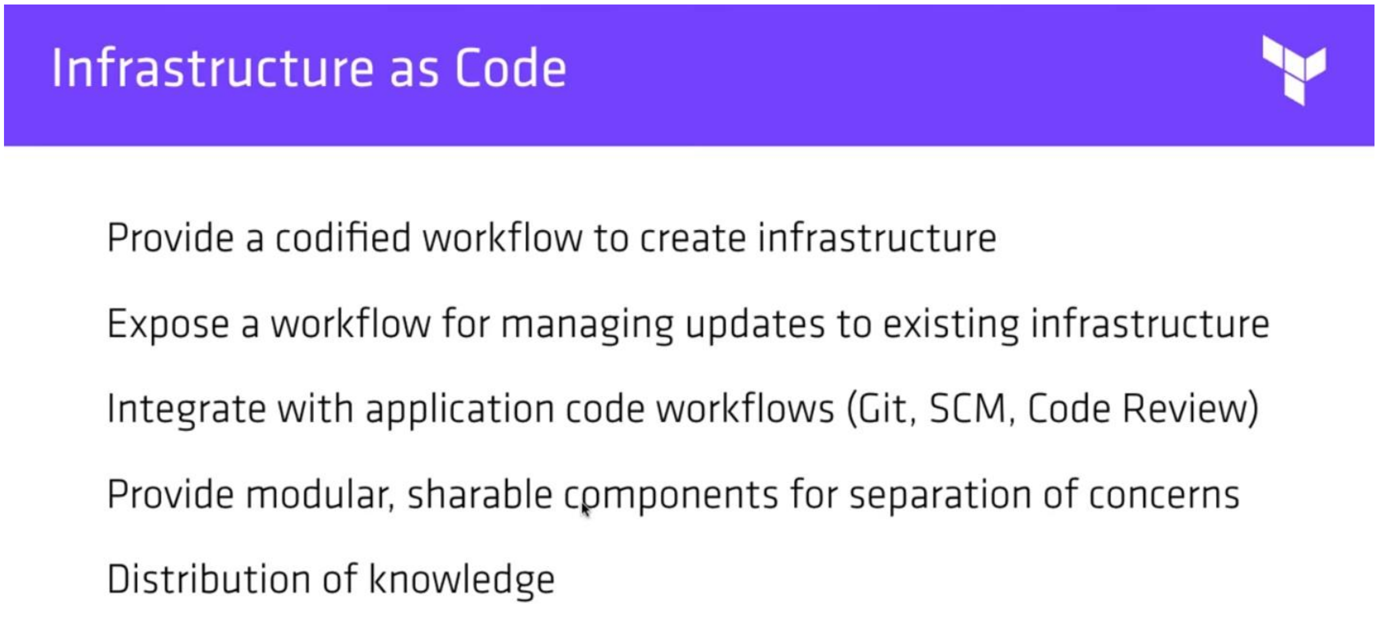
THE BASICS

Let us see some examples of Terraform code for managing resources



```
resource "aws_instance" "web" {
  ami           = "ami-9a562df2"
  instance_type = "t2.micro"
}
```

This code creates and manages an AWS EC2 instance


Infrastructure as Code

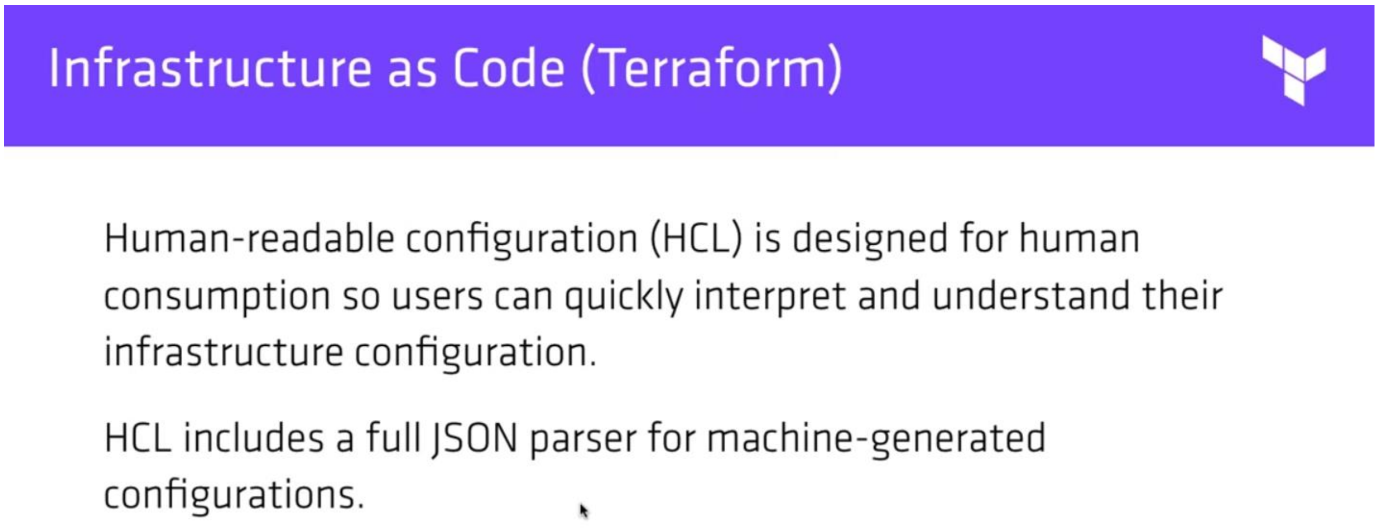
Provide a codified workflow to create infrastructure

Expose a workflow for managing updates to existing infrastructure

Integrate with application code workflows (Git, SCM, Code Review)

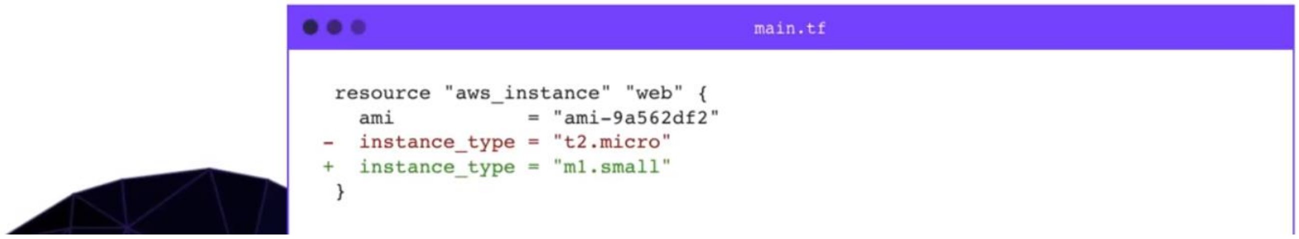
Provide modular, sharable components for separation of concerns

Distribution of knowledge


Infrastructure as Code (Terraform)

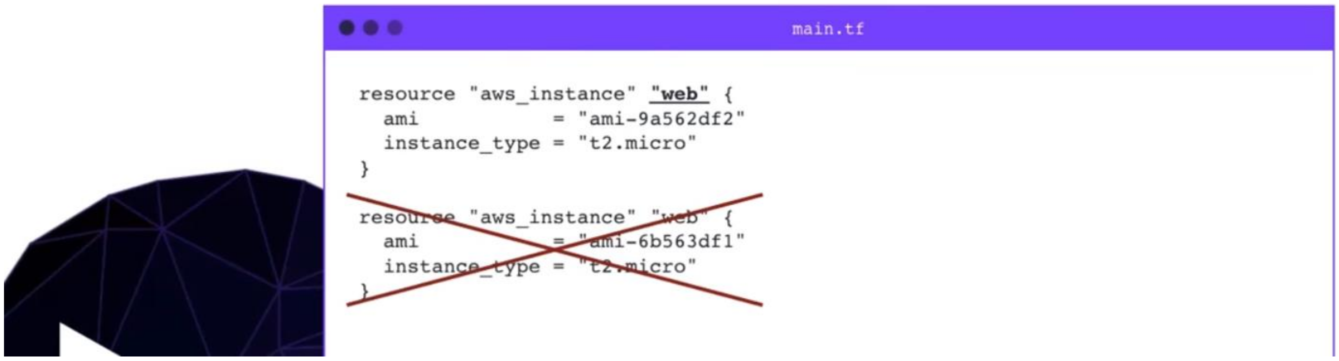
Human-readable configuration (HCL) is designed for human consumption so users can quickly interpret and understand their infrastructure configuration.

HCL includes a full JSON parser for machine-generated configurations.



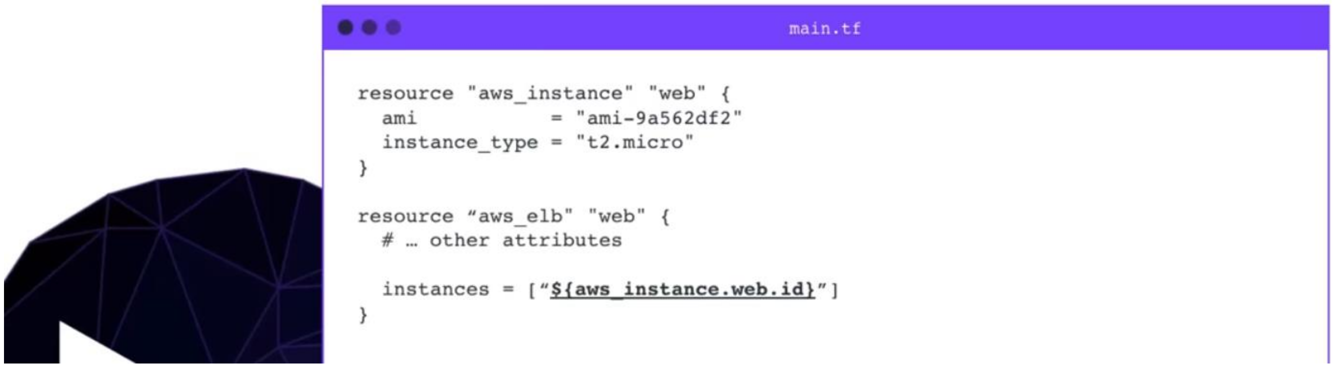
```
main.tf

 resource "aws_instance" "web" {
   ami           = "ami-9a562df2"
-  instance_type = "t2.micro"
+  instance_type = "m1.small"
 }
```

Someone can make and commit a change in GIT as above, terraform only represents the end state that you want



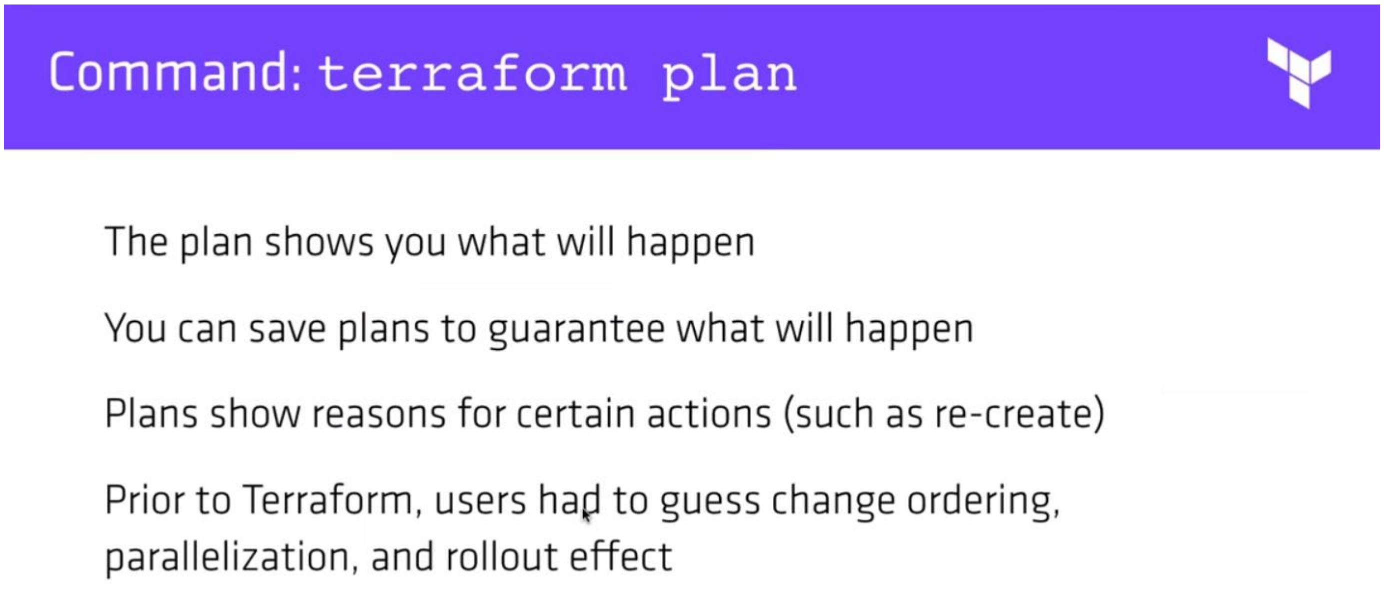
```
main.tf

resource "aws_instance" "web" {
  ami           = "ami-9a562df2"
  instance_type = "t2.micro"
}

resource "aws_instance" "web" {
  ami           = "ami-6b563df1"
  instance_type = "t2.micro"
}
```

Your created resources need to have unique names like web, web1, etc.



```
main.tf

resource "aws_instance" "web" {
  ami           = "ami-9a562df2"
  instance_type = "t2.micro"
}

resource "aws_elb" "web" {
  # … other attributes

  instances = ["${aws_instance.web.id}"]
}
```

You can also represent other instances using their id as above



Command: terraform plan

The plan shows you what will happen

You can save plans to guarantee what will happen

Plans show reasons for certain actions (such as re-create)

Prior to Terraform, users had to guess change ordering, parallelization, and rollout effect

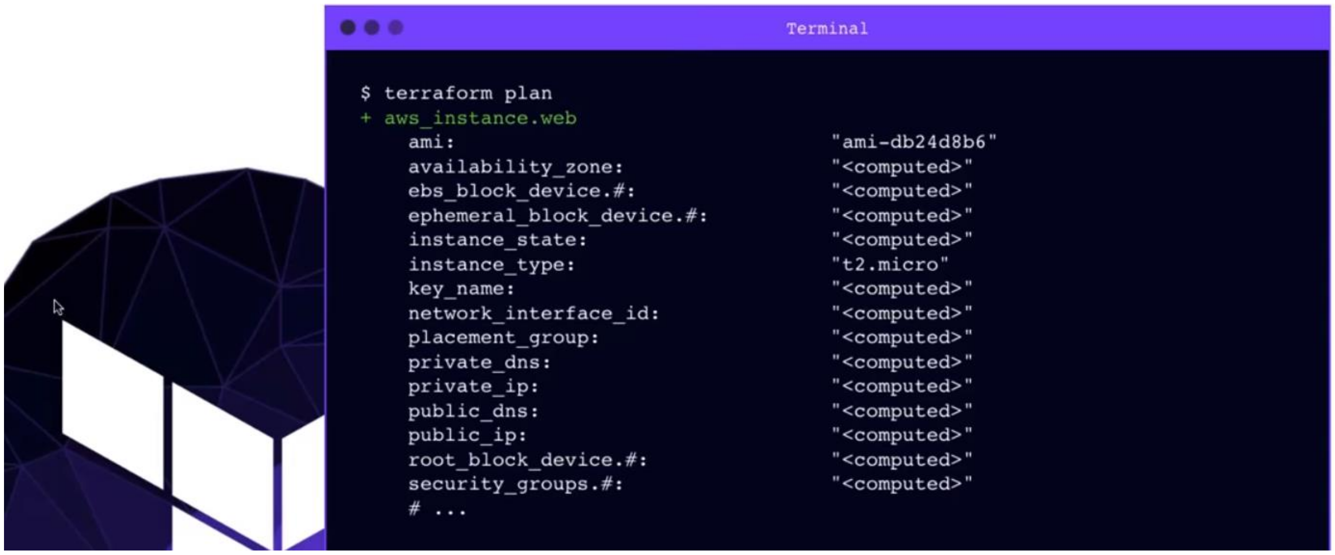
Command: `terraform plan`

+ indicates a resource will be created

- indicates a resource will be destroyed

~ indicates a resource will be updated in-place

-/+ indicates a resources will be destroyed and re-created


Terminal

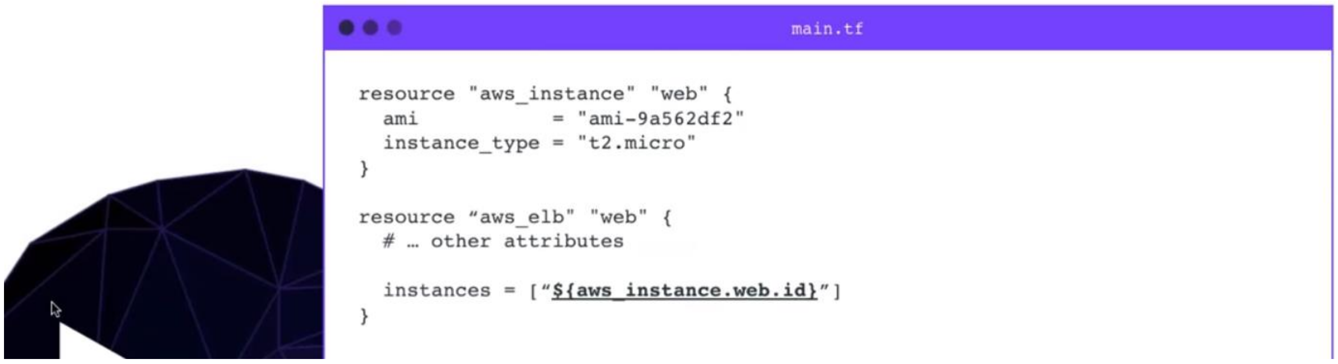
```
$ terraform plan
+ aws_instance.web
    ami:                                  "ami-db24d8b6"
    availability_zone:                    "<computed>"
    ebs_block_device.#:                   "<computed>"
    ephemeral_block_device.#:             "<computed>"
    instance_state:                       "<computed>"
    instance_type:                        "t2.micro"
    key_name:                             "<computed>"
    network_interface_id:                 "<computed>"
    placement_group:                      "<computed>"
    private_dns:                          "<computed>"
    private_ip:                           "<computed>"
    public_dns:                           "<computed>"
    public_ip:                            "<computed>"
    root_block_device.#:                  "<computed>"
    security_groups.#:                    "<computed>"
    # ...
```

Command: `terraform apply`

Execute changes to reach desired state

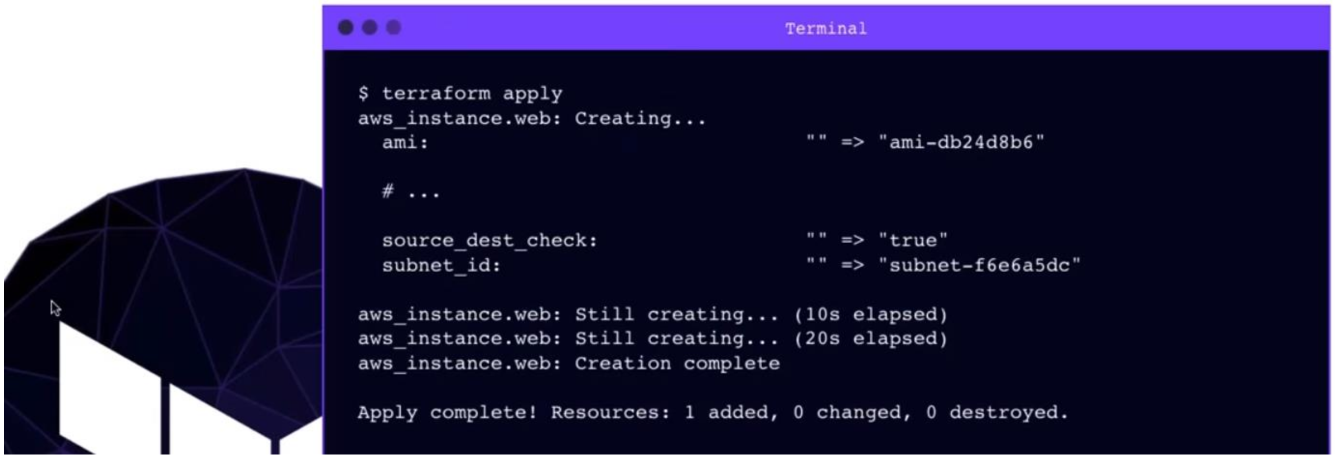
Parallelizes changes when possible

Handles and recovers transient errors safely

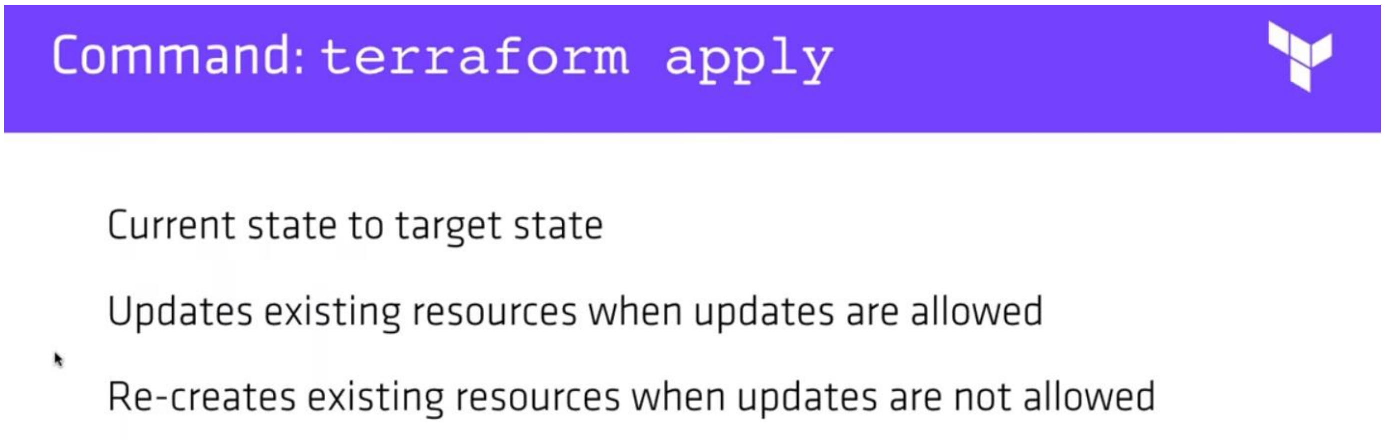

main.tf

```
resource "aws_instance" "web" {
  ami           = "ami-9a562df2"
  instance_type = "t2.micro"
}

resource “aws_elb” "web" {
  # … other attributes

  instances = [“${aws_instance.web.id}”]
}
```

Terraform knows that the aws_instance must be created before the aws_elb in the above case.


Terminal

```
$ terraform apply
aws_instance.web: Creating...
  ami:                               "" => "ami-db24d8b6"

  # ...

  source_dest_check:                 "" => "true"
  subnet_id:                         "" => "subnet-f6e6a5dc"

aws_instance.web: Still creating... (10s elapsed)
aws_instance.web: Still creating... (20s elapsed)
aws_instance.web: Creation complete

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

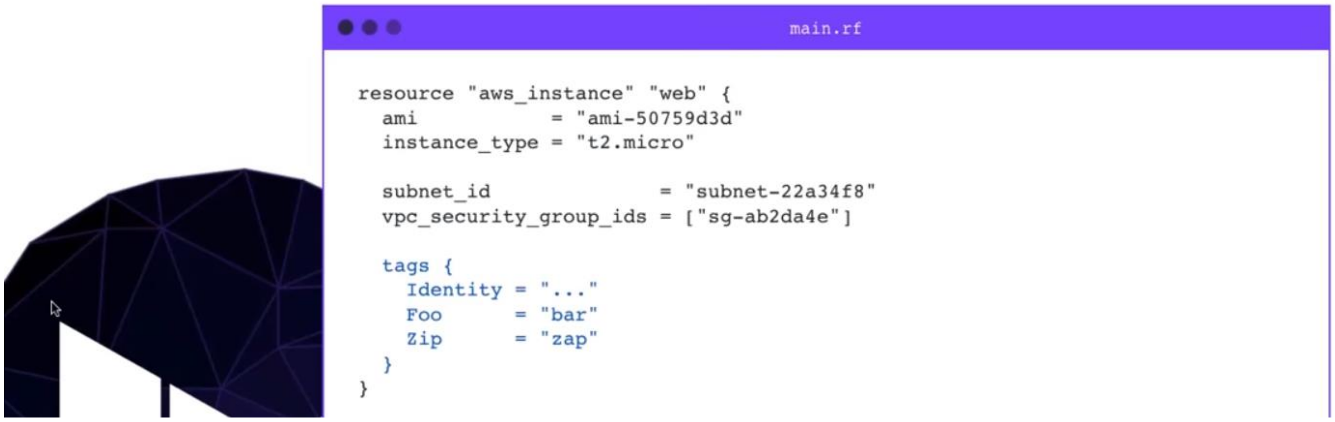


Command: `terraform apply`

Current state to target state

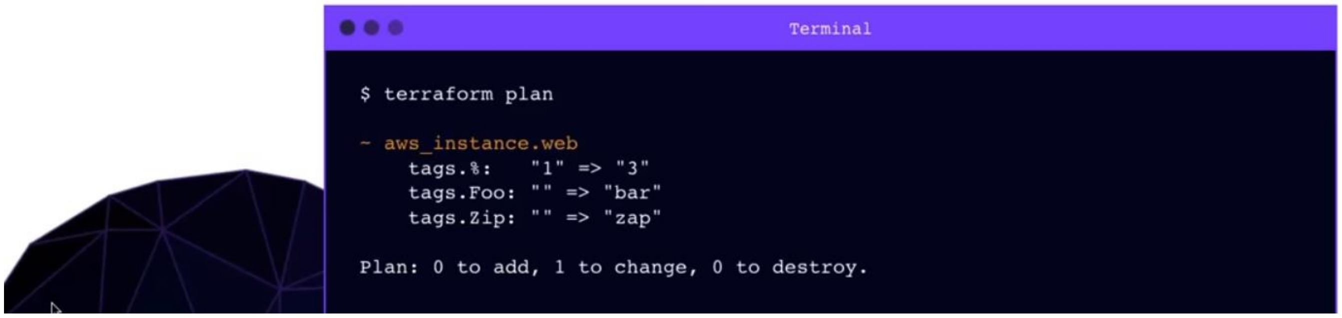
Updates existing resources when updates are allowed

Re-creates existing resources when updates are not allowed

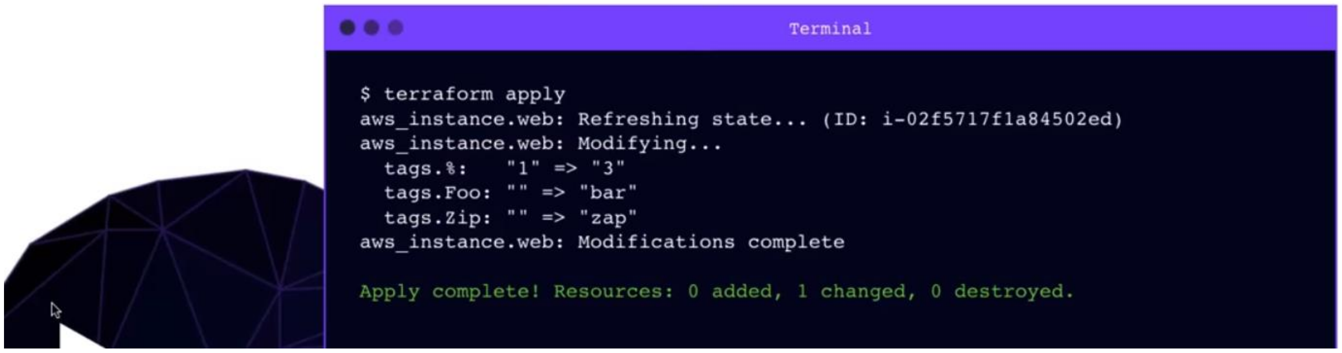

main.rf

```
resource "aws_instance" "web" {
  ami           = "ami-50759d3d"
  instance_type = "t2.micro"

  subnet_id              = "subnet-22a34f8"
  vpc_security_group_ids = ["sg-ab2da4e"]

  tags {
    Identity = "..."
    Foo      = "bar"
    Zip      = "zap"
  }
}
```

We can add some change like the blue text that do not require deleting and creating any new instance as below



```
$ terraform plan

~ aws_instance.web
    tags.%:   "1" => "3"
    tags.Foo: "" => "bar"
    tags.Zip: "" => "zap"

Plan: 0 to add, 1 to change, 0 to destroy.
```



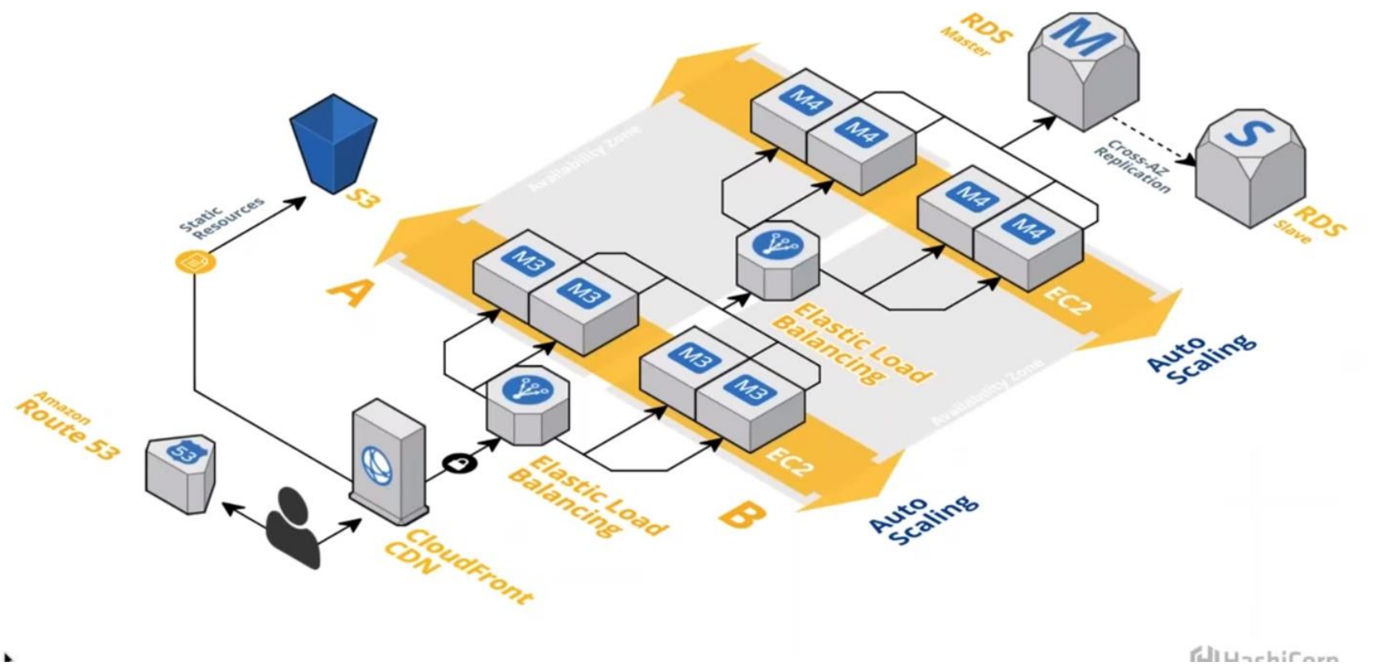
```
$ terraform apply
aws_instance.web: Refreshing state... (ID: i-02f5717f1a84502ed)
aws_instance.web: Modifying...
  tags.%:   "1" => "3"
  tags.Foo: "" => "bar"
  tags.Zip: "" => "zap"
aws_instance.web: Modifications complete

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

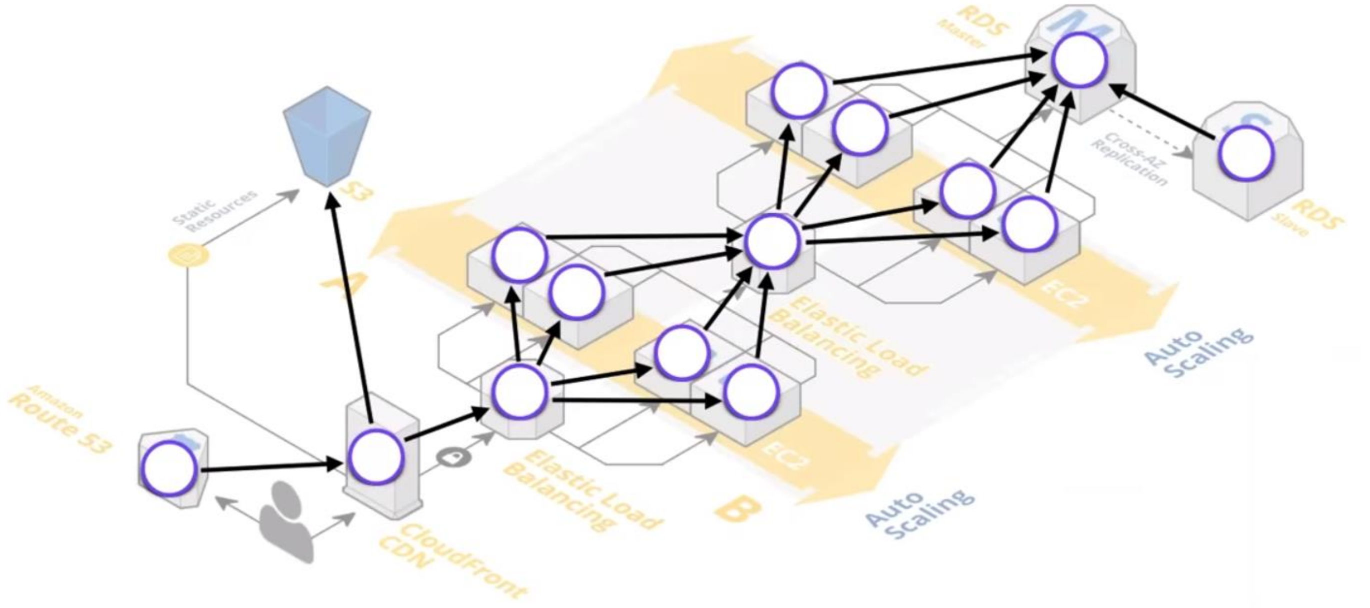
Command: `terraform destroy`

Destroys running infrastructure

Does not touch infrastructure not managed by Terraform

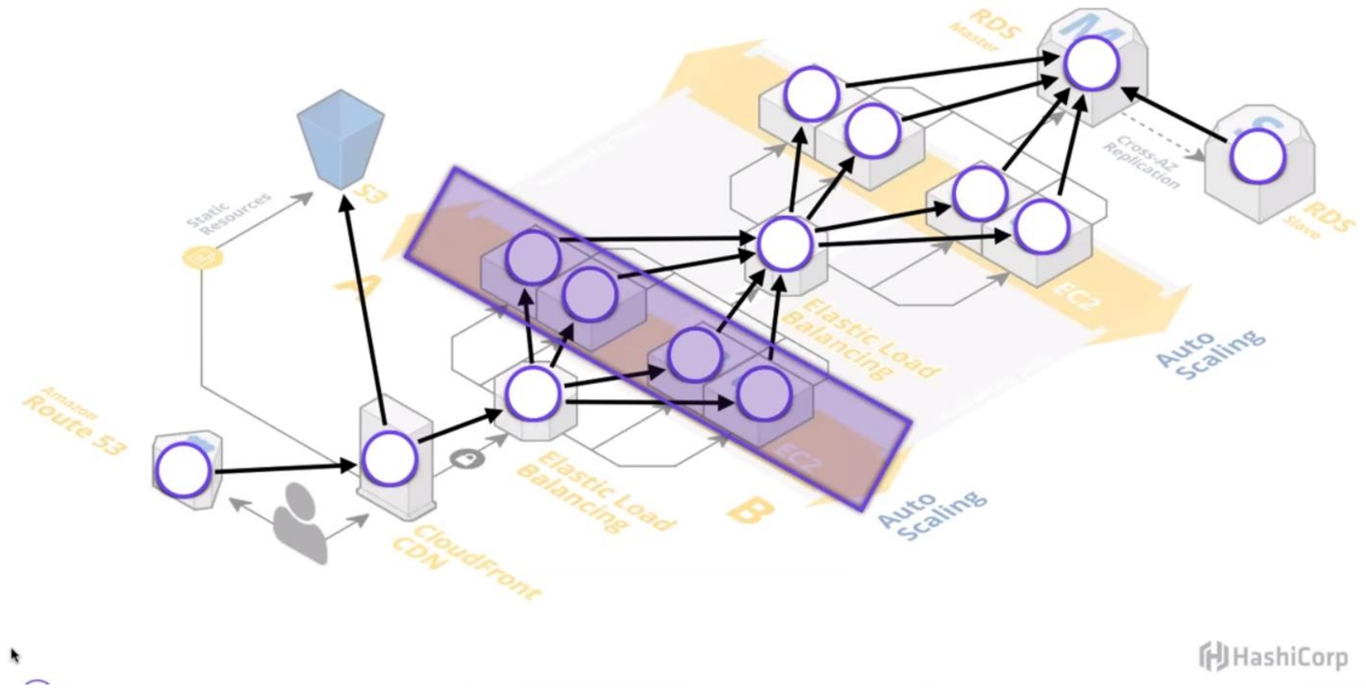
MODELING RESOURCES



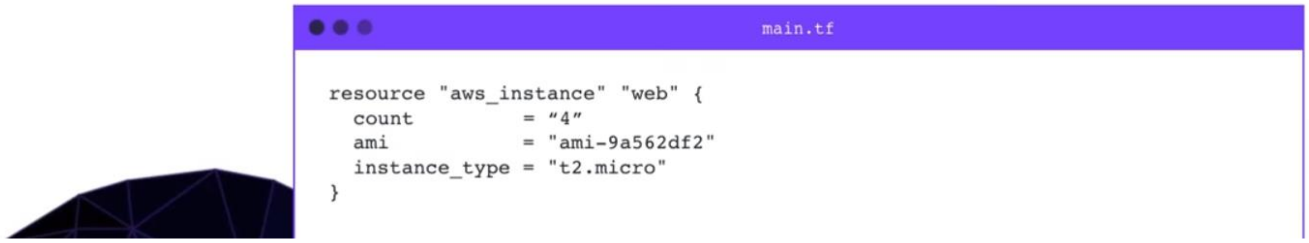


○ = **Resources**

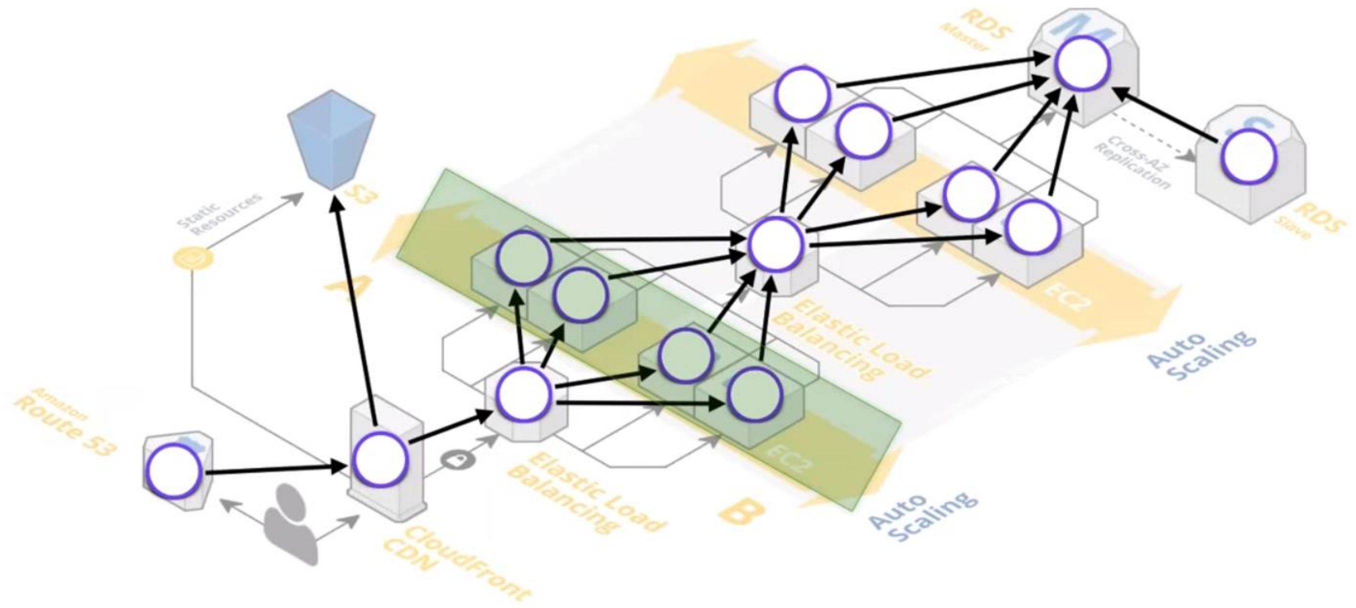
↗ = **Interpolations**



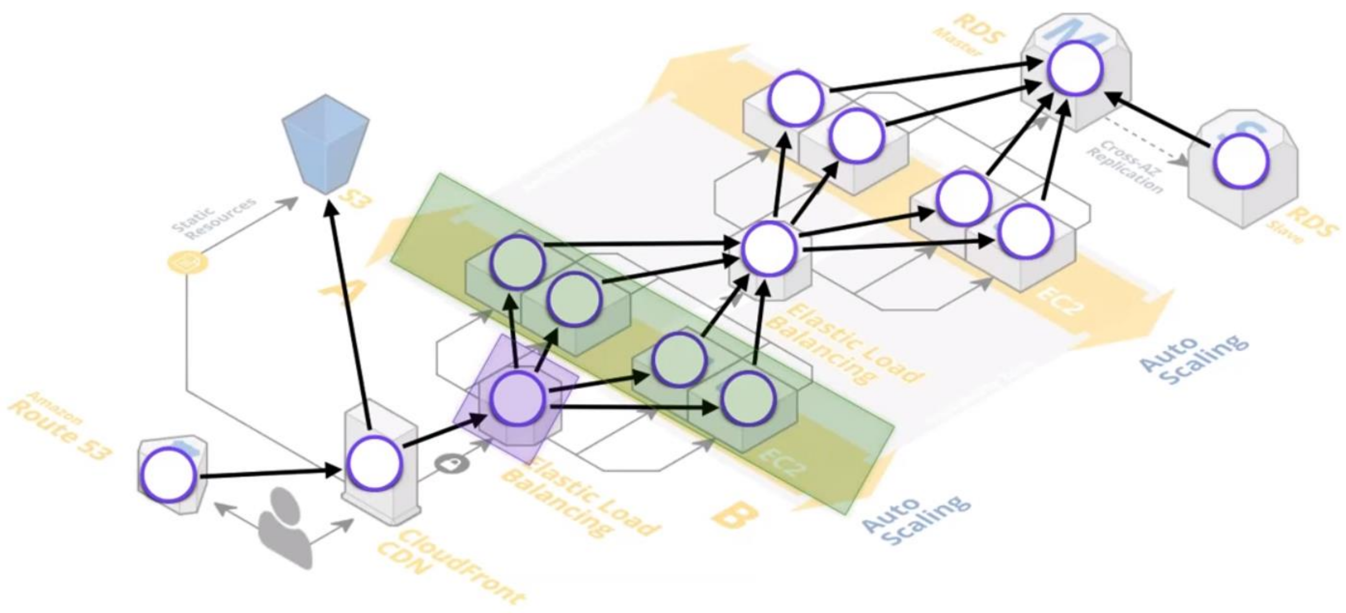
We can start by automating the web servers tier portion of the infrastructure above



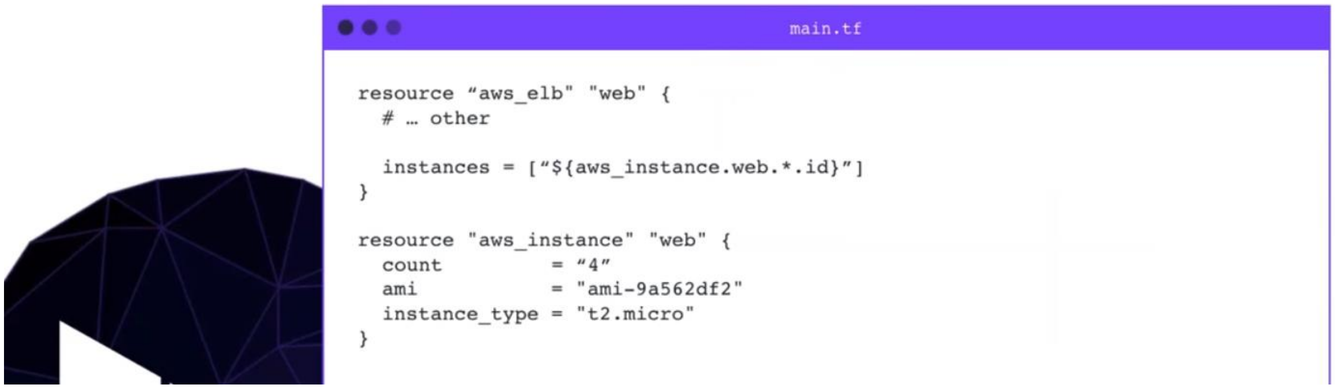
```
resource "aws_instance" "web" {
  count         = "4"
  ami           = "ami-9a562df2"
  instance_type = "t2.micro"
}
```



We have this portion created

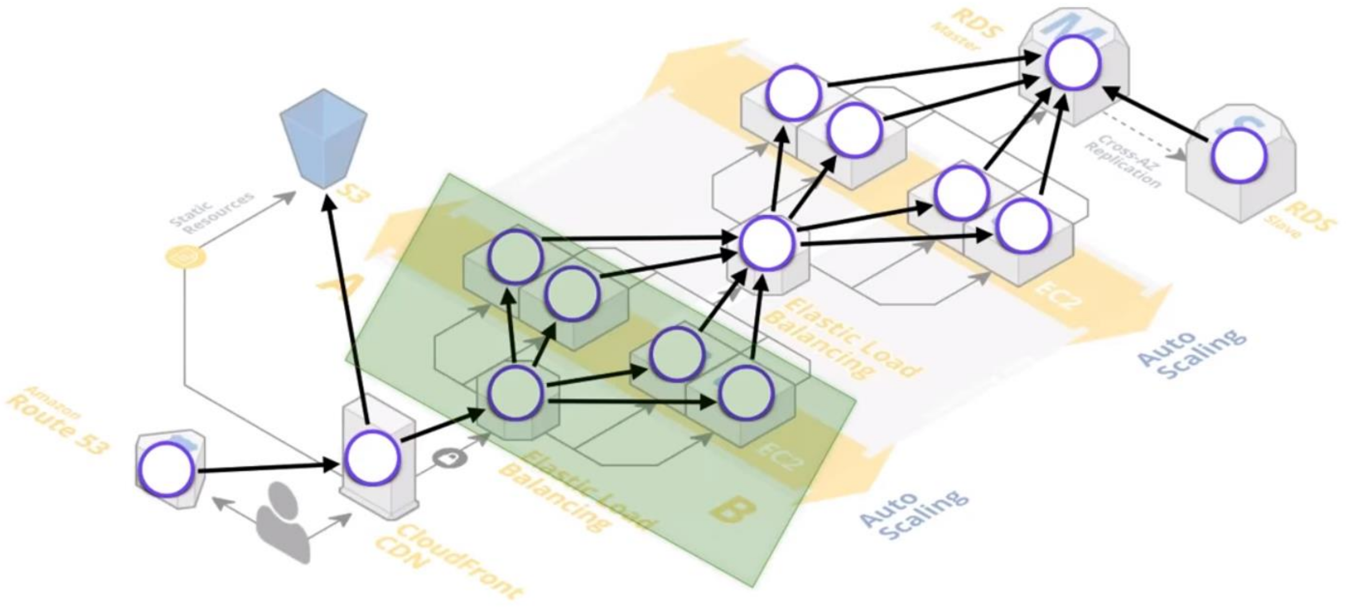


We can now create the load balancer and a ELB over the 4 web servers downstream next



main.tf

```
resource "aws_elb" "web" {
  # … other

  instances = ["${aws_instance.web.*.id}"]
}

resource "aws_instance" "web" {
  count         = "4"
  ami           = "ami-9a562df2"
  instance_type = "t2.micro"
}
```



We now have the ELB and the web servers automated and managed with Terraform now

Modeling Resources

Rinse, Repeat

You don’t need to Terraform 100% of your Infra

NEXT STEPS

Getting More Advanced

Variable, Outputs (Parameterization)

Interpolation Functions

Modules

Remote State

Next Steps

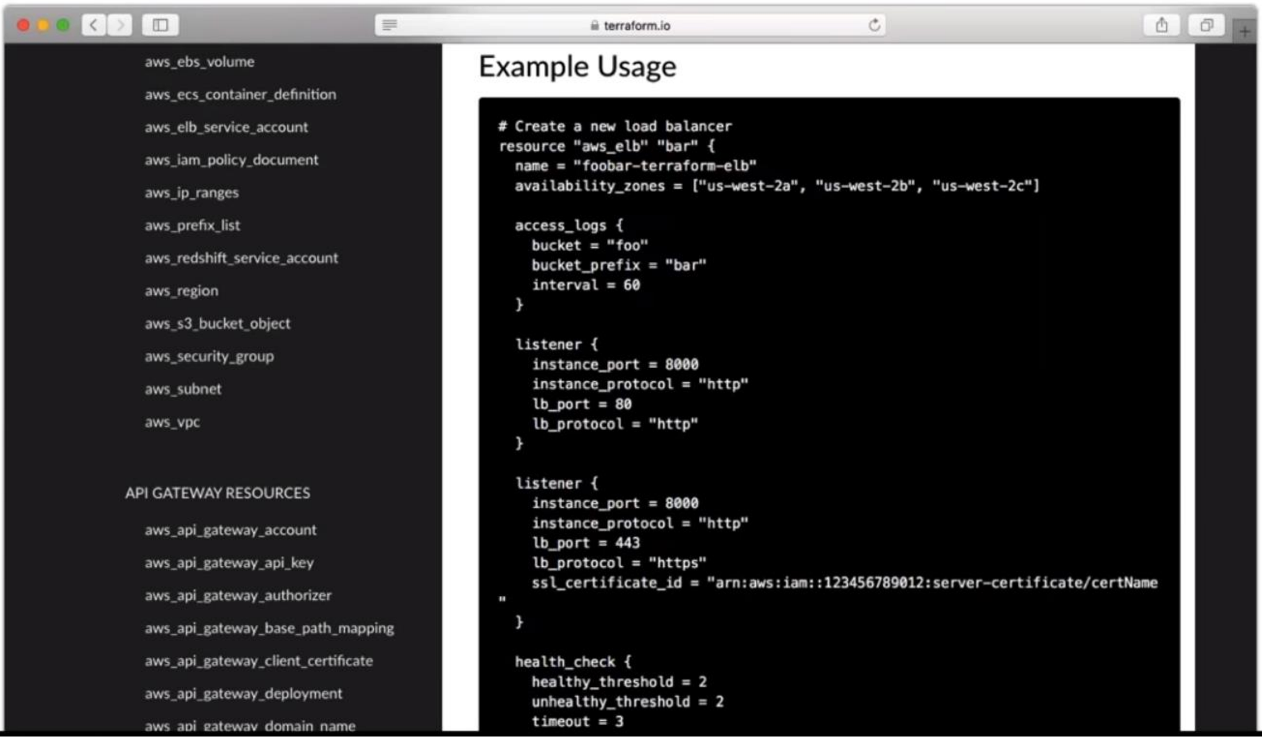
terraform.io

github.com/terraform-community-modules

Google: Lots of blog posts for examples

HashiCorp Events + Training

terraformbook.com (community)



Example Usage

```
# Create a new load balancer
resource "aws_elb" "bar" {
  name = "foobar-terraform-elb"
  availability_zones = ["us-west-2a", "us-west-2b", "us-west-2c"]

  access_logs {
    bucket = "foo"
    bucket_prefix = "bar"
    interval = 60
  }

  listener {
    instance_port = 8000
    instance_protocol = "http"
    lb_port = 80
    lb_protocol = "http"
  }

  listener {
    instance_port = 8000
    instance_protocol = "http"
    lb_port = 443
    lb_protocol = "https"
    ssl_certificate_id = "arn:aws:iam::123456789012:server-certificate/certName
"
  }

  health_check {
    healthy_threshold = 2
    unhealthy_threshold = 2
    timeout = 3
```

CREATE, CHANGE, AND ORCHESTRATE AWS INFRASTRUCTURE WITH TERRAFORM

Q&A

WEBINAR

HashiCorp amazon web services