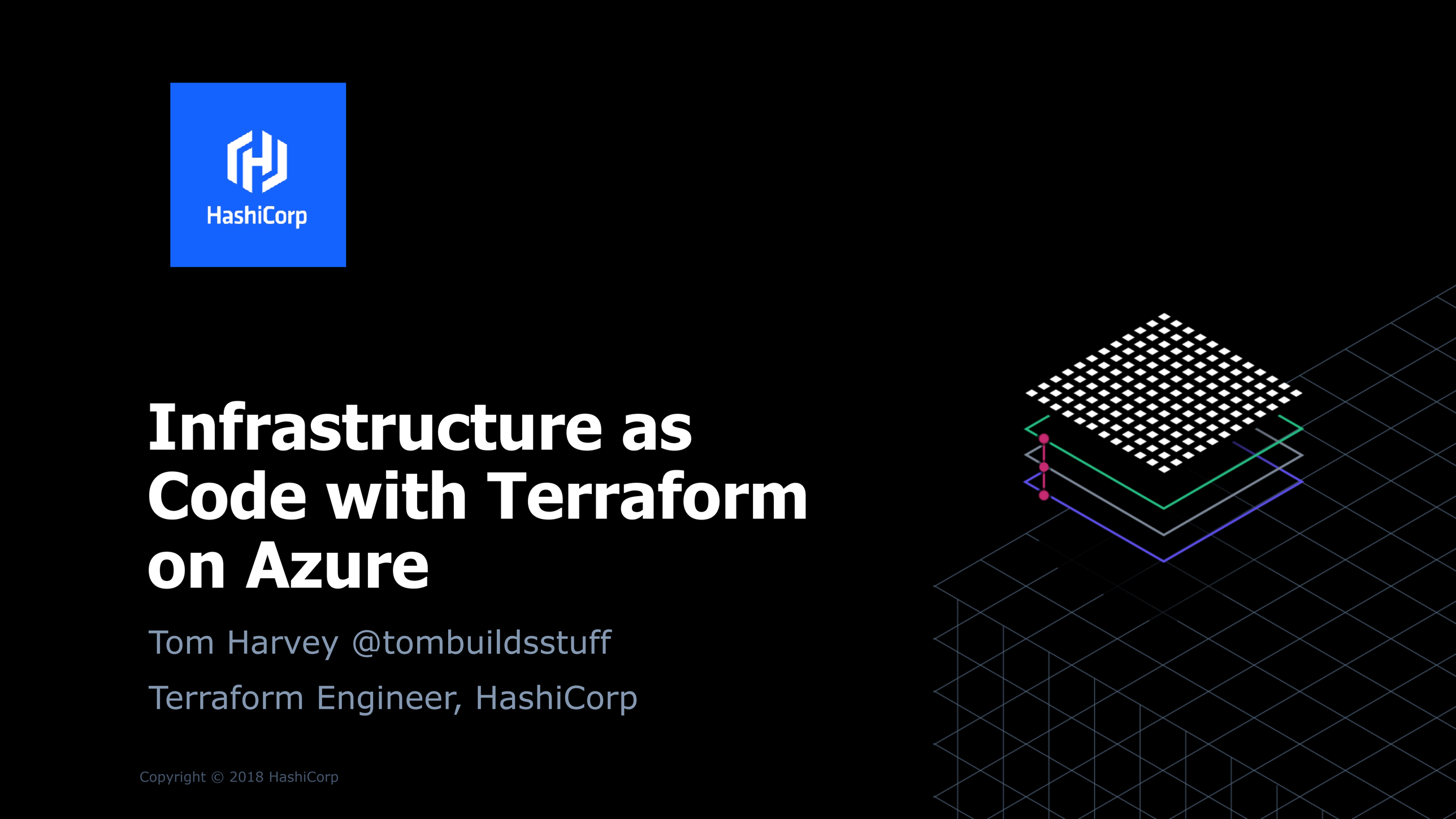
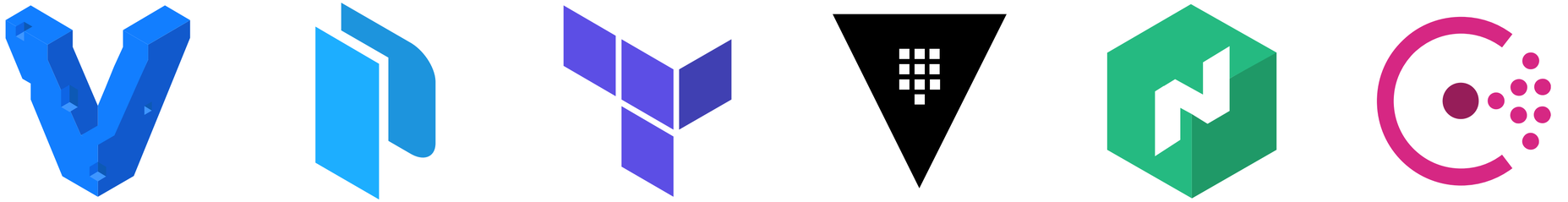


Infrastructure as Code with Terraform on Azure

Tom Harvey @tombuildsstuff

Terraform Engineer, HashiCorp



Infrastructure as Code

What is Infrastructure as Code?

Infrastructure as Code

- Allows you to declaratively describe your infrastructure in code

Infrastructure as Code

- Allows you to declaratively describe your infrastructure in code
- This means software development techniques (such as Code Review/Continuous Integration) can be applied to your infrastructure

Infrastructure as Code

- Allows you to declaratively describe your infrastructure in code
- This means software development techniques (such as Code Review/Continuous Integration) can be applied to your infrastructure
- Side-benefits:
  - allows your infrastructure to be spun up in a different region

Infrastructure as Code

- Allows you to declaratively describe your infrastructure in code
- This means software development techniques (such as Code Review/Continuous Integration) can be applied to your infrastructure
- Side-benefits:
  - allows your infrastructure to be spun up in a different region
  - your infrastructure is documented

Infrastructure as Code

- Allows you to declaratively describe your infrastructure in code
- This means software development techniques (such as Code Review/Continuous Integration) can be applied to your infrastructure
- Side-benefits:
  - allows your infrastructure to be spun up in a different region
  - your infrastructure is documented
  - ensure your infrastructure meets any policy requirements

Infrastructure as Code

- Allows you to declaratively describe your infrastructure in code
- This means software development techniques (such as Code Review/Continuous Integration) can be applied to your infrastructure
- Side-benefits:
  - allows your infrastructure to be spun up in a different region
  - your infrastructure is documented
  - ensure your infrastructure meets any policy requirements
- Alternatives: ARM Templates / CloudFormation

Infrastructure as Code with Terraform

Applying Infrastructure as Code with Terraform

Infrastructure as Code with Terraform

HCL: HashiCorp Configuration Language

- Terraform uses a DSL known as HCL (HashiCorp Configuration Language)

Example

HCL: HashiCorp Configuration Language

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "Australia South"
}

resource "azurerm_virtual_network" "test" {
  name                = "example-network"
  resource_group_name = "${azurerm_resource_group.test.name}"
  location            = "${azurerm_resource_group.test.location}"
  address_space       = ["10.0.0.0/16"]
}
```

Infrastructure as Code with Terraform

HCL: HashiCorp Configuration Language

- Terraform uses a DSL known as HCL (HashiCorp Configuration Language)
- HCL provides a common language which can be used to provision resources across any Provider

Infrastructure as Code with Terraform

Providers

- Alicloud
- AWS
- Azure
- Azure Stack
- Bitbucket
- Circonus
- Cloudflare
- CloudScale.ch
- CloudStack
- Cobbler
- Datadog
- DigitalOcean
- DNSMadeEasy
- DNSimple
- Dyn
- Fastly
- FlexibleEngine
- GitHub
- Gitlab
- Google Cloud
- Grafana
- Heroku
- Hetzner Cloud
- HuaweiCloud
- Icinga2
- Ignition
- InfluxDB
- Kubernetes
- Librato
- Netlify
- New Relic
- NS1
- 1&1
- OpenStack
- OpenTelekom Cloud
- Oracle Cloud
- OVH
- Packet
- PagerDuty
- Palo Alto Networks
- PowerDNS
- ProfitBricks
- Rancher
- RightScale
- RunScope
- Scaleway
- SoftLayer
- StatusCake
- Spotinst
- Terraform Enterprise
- TLS
- Triton
- Vault
- vCloud Director
- NSX-T
- vSphere

Infrastructure as Code with Terraform

Cloud Providers

- Alicloud
- AWS
- Azure
- Azure Stack
- Bitbucket
- Circonus
- Cloudflare
- CloudScale.ch
- CloudStack
- Cobbler
- Datadog
- DigitalOcean
- DNSMadeEasy
- DNSimple
- Dyn
- Fastly
- FlexibleEngine
- GitHub
- Gitlab
- Google Cloud
- Grafana
- Heroku
- Hetzner Cloud
- HuaweiCloud
- Icinga2
- Ignition
- InfluxDB
- Kubernetes
- Librato
- Netlify
- New Relic
- NS1
- 1&1
- OpenStack
- OpenTelekom Cloud
- Oracle Cloud
- OVH
- Packet
- PagerDuty
- Palo Alto Networks
- PowerDNS
- ProfitBricks
- Rancher
- RightScale
- RunScope
- Scaleway
- SoftLayer
- StatusCake
- Spotinst
- Terraform Enterprise
- TLS
- Triton
- Vault
- vCloud Director
- NSX-T
- vSphere

Infrastructure as Code with Terraform

DNS Providers

- Alicloud
- AWS
- Azure
- Azure Stack
- Bitbucket
- Circonus
- Cloudflare
- CloudScale.ch
- CloudStack
- Cobbler
- Datadog
- DigitalOcean
- DNSMadeEasy
- DNSimple
- Dyn
- Fastly
- FlexibleEngine
- GitHub
- Gitlab
- Google Cloud
- Grafana
- Heroku
- Hetzner Cloud
- HuaweiCloud
- Icinga2
- Ignition
- InfluxDB
- Kubernetes
- Librato
- Netlify
- New Relic
- NS1
- 1&1
- OpenStack
- OpenTelekom Cloud
- Oracle Cloud
- OVH
- Packet
- PagerDuty
- Palo Alto Networks
- PowerDNS
- ProfitBricks
- Rancher
- RightScale
- RunScope
- Scaleway
- SoftLayer
- StatusCake
- Spotinst
- Terraform Enterprise
- TLS
- Triton
- Vault
- vCloud Director
- NSX-T
- vSphere

Infrastructure as Code with Terraform

SaaS Services

- Alicloud
- AWS
- Azure
- Azure Stack
- Bitbucket
- Circonus
- Cloudflare
- CloudScale.ch
- CloudStack
- Cobbler
- Datadog
- DigitalOcean
- DNSMadeEasy
- DNSimple
- Dyn
- Fastly
- FlexibleEngine
- GitHub
- Gitlab
- Google Cloud
- Grafana
- Heroku
- Hetzner Cloud
- HuaweiCloud
- Icinga2
- Ignition
- InfluxDB
- Kubernetes
- Librato
- Netlify
- New Relic
- NS1
- 1&1
- OpenStack
- OpenTelekom Cloud
- Oracle Cloud
- OVH
- Packet
- PagerDuty
- Palo Alto Networks
- PowerDNS
- ProfitBricks
- Rancher
- RightScale
- RunScope
- Scaleway
- SoftLayer
- StatusCake
- Spotinst
- Terraform Enterprise
- TLS
- Triton
- Vault
- vCloud Director
- NSX-T
- vSphere

Infrastructure as Code with Terraform

On Premise Providers

- Alicloud
- AWS
- Azure
- Azure Stack
- Bitbucket
- Circonus
- Cloudflare
- CloudScale.ch
- CloudStack
- Cobbler
- Datadog
- DigitalOcean
- DNSMadeEasy
- DNSimple
- Dyn
- Fastly
- FlexibleEngine
- GitHub
- Gitlab
- Google Cloud
- Grafana
- Heroku
- Hetzner Cloud
- HuaweiCloud
- Icinga2
- Ignition
- InfluxDB
- Kubernetes
- Librato
- Netlify
- New Relic
- NS1
- 1&1
- OpenStack
- OpenTelekom Cloud
- Oracle Cloud
- OVH
- Packet
- PagerDuty
- Palo Alto Networks
- PowerDNS
- ProfitBricks
- Rancher
- RightScale
- RunScope
- Scaleway
- SoftLayer
- StatusCake
- Spotinst
- Terraform Enterprise
- TLS
- Triton
- Vault
- vCloud Director
- NSX-T
- vSphere

Infrastructure as Code with Terraform

HCL: HashiCorp Configuration Language

- Terraform uses a DSL known as HCL (HashiCorp Configuration Language)
- HCL provides a common language which can be used to provision resources across any Provider
- Terraform has support for over 80 HashiCorp supported Providers, 80+ community Providers and it's possible to build your own too

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "Australia South"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "Australia South"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "Australia South"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "Australia South"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "Australia South"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "Australia South"
}
```

Terraform

Listing the commands

```
$ terraform
```

Terraform

Listing the commands

```
$ terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    # ...
    destroy            Destroy Terraform-managed infrastructure
    # ...
    init               Initialize a Terraform working directory
    # ...
    plan               Generate and show an execution plan
```

Terraform Init

Downloads any required providers and module sources

```
$ terraform init
```

Terraform Init

Downloads any required providers and module sources

```
$ terraform init

Initializing provider plugins...
- Checking for available provider plugins on https://
releases.hashicorp.com...
- Downloading plugin for provider "azurerm" (1.15.0)...

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to
see
any changes that are required for your infrastructure. All Terraform
commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget,
other
commands will detect it and remind you to do so if necessary.
```

Terraform Plan

Determines what changes need to be performed

```
$ terraform plan
```

Terraform Plan

Determines what changes need to be performed

```
$ terraform plan

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + azurerm_resource_group.test
      id:       <computed>
      location: "australiasouth"
      name:     "example-resources"
      tags.%:   <computed>


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

Terraform Apply

Determines what changes need to be performed, then makes them

```
$ terraform apply
```

Terraform Apply

Determines what changes need to be performed, then makes them

```
$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + azurerm_resource_group.test
      id:       <computed>
      location: "australiasouth"
      name:     "example-resources"
      tags.%:   <computed>


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

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

Terraform Apply

Determines what changes need to be performed, then makes them

```
# ...

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

azurerm_resource_group.test: Creating...
  location: "" => "australiasouth"
  name:     "" => "example-resources"
  tags.%:   "" => "<computed>"
azurerm_resource_group.test: Creation complete after 4s (ID: /
subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/
example-resources)

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

Terraform Plan

Determines what changes need to be performed

```
$ terraform plan
```

Terraform Plan

Determines what changes need to be performed

```
$ terraform plan

Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not
be
persisted to local or remote state storage.

azurerm_resource_group.test: Refreshing state... (ID: /subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example-resources)

------------------------------------------------------------------------

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

Terraform Destroy

Deletes any resources that Terraform manages

```
$ terraform destroy
```

Terraform Destroy

Deletes any resources that Terraform manages

```
$ terraform destroy
azurerm_resource_group.test: Refreshing state... (ID: /subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example-resources)

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  - azurerm_resource_group.test


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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value:
```

Terraform Destroy

Deletes any resources that Terraform manages

```
# ...

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

azurerm_resource_group.test: Destroying... (ID: /subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example-resources)
azurerm_resource_group.test: Still destroying... (ID: /subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example-resources, 10s
elapsed)
# ...
azurerm_resource_group.test: Still destroying... (ID: /subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example-resources, 50s
elapsed)
azurerm_resource_group.test: Destruction complete after 57s

Destroy complete! Resources: 1 destroyed.
```

Demo

Provisioning a Resource Group and Virtual Network in Azure

Summary

- **terraform init**
  - Downloads any providers/modules being used

Summary

- **terraform init**
  - Downloads any providers/modules being used

- **terraform plan**
  - Determines which changes need to be applied

Summary

- **terraform init**
  - Downloads any providers/modules being used
- **terraform plan**
  - Determines which changes need to be applied
- **terraform apply**
  - Makes the changes specified in the Plan

Summary

- **terraform init**
  - Downloads any providers/modules being used
- **terraform plan**
  - Determines which changes need to be applied
- **terraform apply**
  - Makes the changes specified in the Plan
- **terraform destroy**
  - Destroys any resources provisioned via Terraform

Working with your existing infrastructure

Provisioned via some other means

Working with your existing infrastructure

- Terraform supports two methods of working with your existing infrastructure:
  - **Data Sources** - reference information about the resources

Data Source

Retrieve information about an existing Resource

```
data "azurerm_virtual_network" "test" {
  name                = "existing-network"
  resource_group_name = "existing-resources"
}
```

Data Source

Retrieve information about an existing Resource

```
data "azurerm_virtual_network" "test" {
  name                = "existing-network"
  resource_group_name = "existing-resources"
}
```

Data Source

Retrieve information about an existing Resource

```
data "azurerm_virtual_network" "test" {
  name                = "existing-network"
  resource_group_name = "existing-resources"
}

output "address_space" {
  value = "${data.azurerm_virtual_network.test.address_space}"
}
```

Working with your existing infrastructure

- Terraform supports two methods of working with your existing infrastructure:
  - **Data Sources** - reference information about the resources
  - **Importing** - bringing the resource under Terraform's control

Importing

Importing your existing resources to Terraform

```
$ terraform import azurerm_resource_group.test
/subscriptions/
00000000-0000-0000-0000-000000000000/
resourceGroups/example
```

Importing

Importing your existing resources to Terraform

```
$ terraform import azurerm_resource_group.test /subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example

azurerm_resource_group.test: Importing from ID "/subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example"...
azurerm_resource_group.test: Import complete!
  Imported azurerm_resource_group (ID: /subscriptions/
00000000-0000-0000-0000-000000000000/resourceGroups/example)
azurerm_resource_group.test: Refreshing state... (ID: /
subscriptions/00000000-0000-0000-0000-000000000000/
resourceGroups/example)

Import successful!

The resources that were imported are shown above. These resources
are now in
your Terraform state and will henceforth be managed by Terraform.
```

Importing

Importing your existing resources to Terraform

Import

Resource Groups can be imported using the `resource id`, e.g.

```
terraform import azurerm_resource_group.mygroup /subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myresourcegroup
```

Demo

Working with your existing infrastructure

Modules

Creating reusable modules with Terraform

Modules

Reusing your Terraform Configurations

- In Terraform - a module is a directory which contains *.tf files
- A Terraform Module can have Inputs (Variables) and Outputs - and is a self-contained unit
- Modules can be sourced from the file system, a git repository or a Module Registry

Modules

Reusing your Terraform Configurations

```
# from the local file system
module "example" {
  source = "./modules/example"
}
```

Modules

Reusing your Terraform Configurations

```
# from the local file system
module "example" {
  source = "./modules/example"
}

# from a Git repository (a specific Tag)
module "example" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}
```

Modules

Reusing your Terraform Configurations

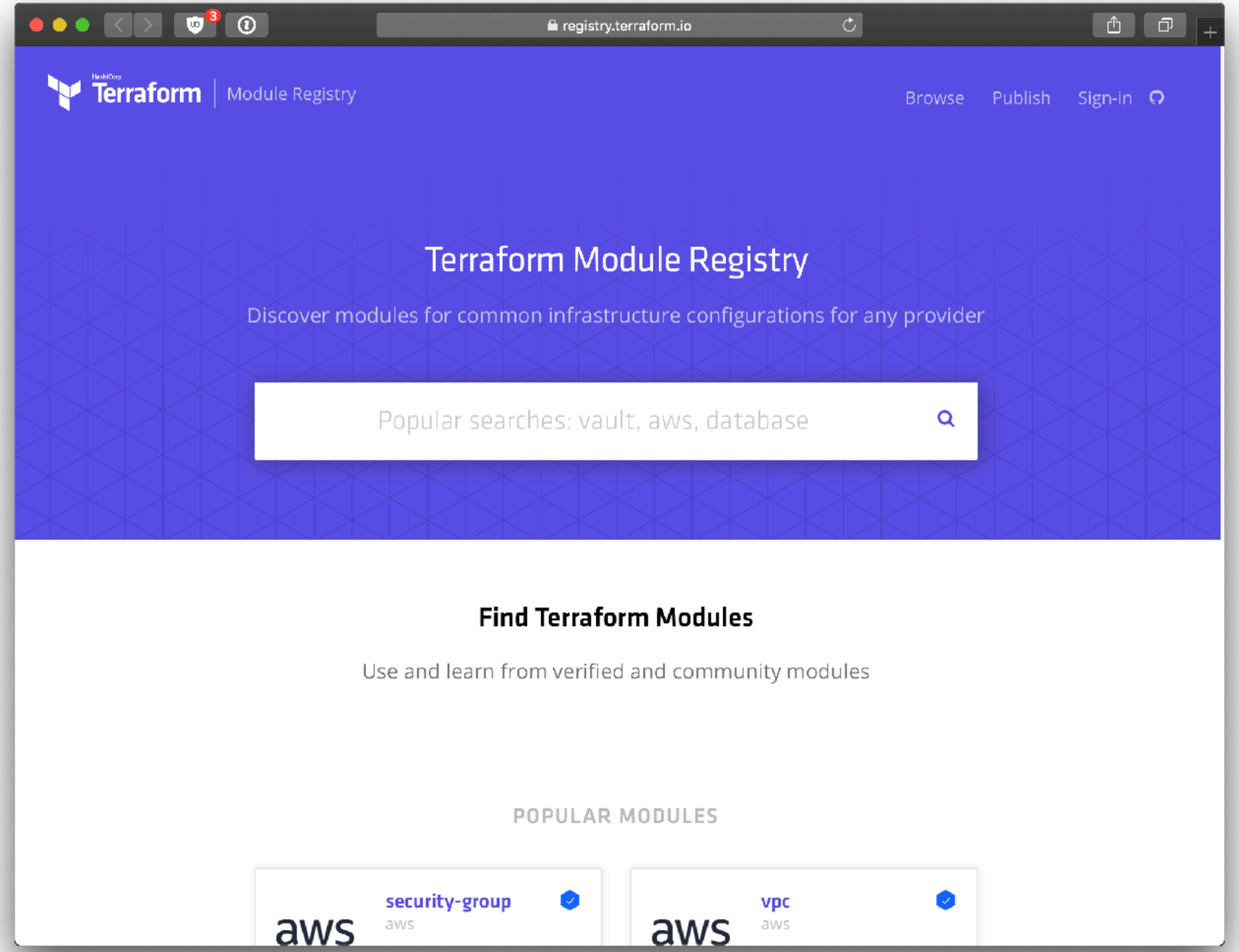
```
# from the local file system
module "example" {
  source = "./modules/example"
}

# from a Git repository (a specific Tag)
module "example" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}

# from a Module Registry
module "consul" {
  source = "hashicorp/consul/aws"
  version = "0.1.0"
}
```

Module Registry

Pre-created Terraform Modules



Demo

Reusable code with Modules in Terraform

Summary

- Terraform's DSL (HCL) allows for a common language to be used to provision any infrastructure on any provider

Summary

- Terraform's DSL (HCL) allows for a common language to be used to provision any infrastructure on any provider

- Terraform's Plan command will show you which changes it's going to make to your infrastructure to ensure the configuration defined in code matches the state of the world; which can then be made using Terraform's Apply command

Summary

- Terraform's DSL (HCL) allows for a common language to be used to provision any infrastructure on any provider

- Terraform's Plan command will show you which changes it's going to make to your infrastructure to ensure the configuration defined in code matches the state of the world; which can then be made using Terraform's Apply command

- When you're done - Terraform Destroy allows you to tear down any resources Terraform's provisioned

Summary

- Terraform's DSL (HCL) allows for a common language to be used to provision any infrastructure on any provider
- Terraform's Plan command will show you which changes it's going to make to your infrastructure to ensure the configuration defined in code matches the state of the world; which can then be made using Terraform's Apply command
- When you're done - Terraform Destroy allows you to tear down any resources Terraform's provisioned
- Terraform Configurations can be shared in Modules allowing code reuse

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com