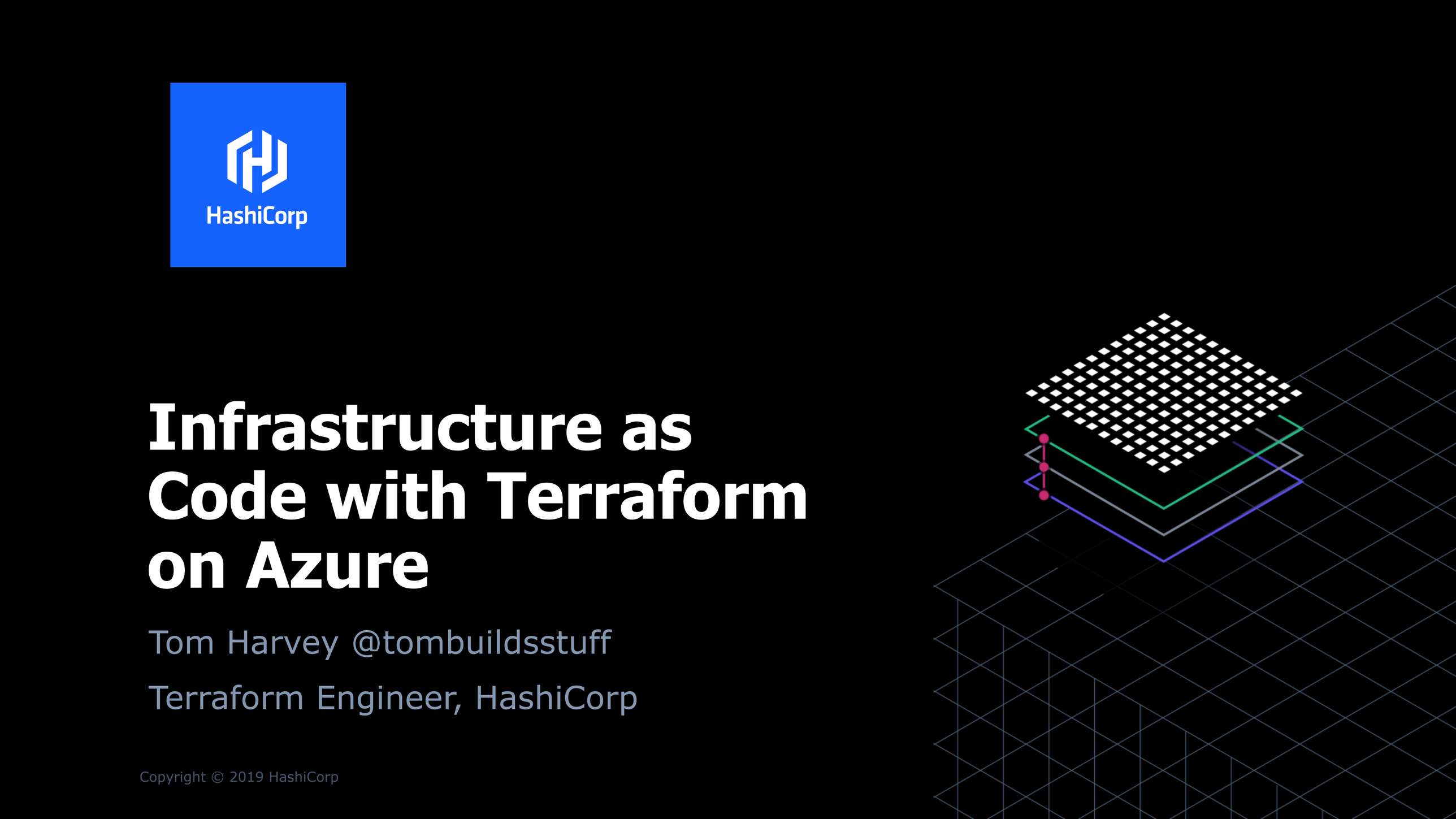
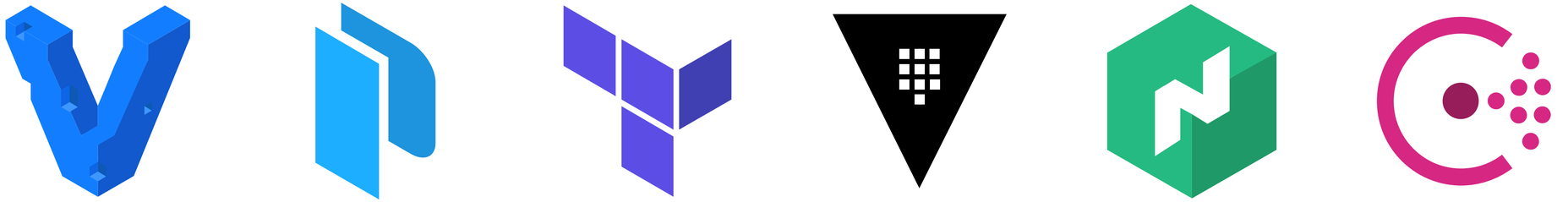


Infrastructure as Code with Terraform on Azure

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 = "West Europe"
}

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 AD
- 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 AD
- 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 AD
- 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 AD
- 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 AD
- 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 120 HashiCorp supported Providers, 100+ 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 = "West Europe"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "West Europe"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "West Europe"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = “West Europe”
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "West Europe"
}
```

Example

Provisioning a Resource Group in Azure with Terraform

```
resource "azurerm_resource_group" "test" {
  name     = "example-resources"
  location = "West Europe"
}
```

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 Plan

Determines what changes need to be performed

```
$ terraform plan
```

Terraform Apply

Determines what changes need to be performed, then makes them

```
$ terraform apply
```

Terraform Destroy

Deletes any resources that Terraform manages

```
$ terraform destroy
```

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.

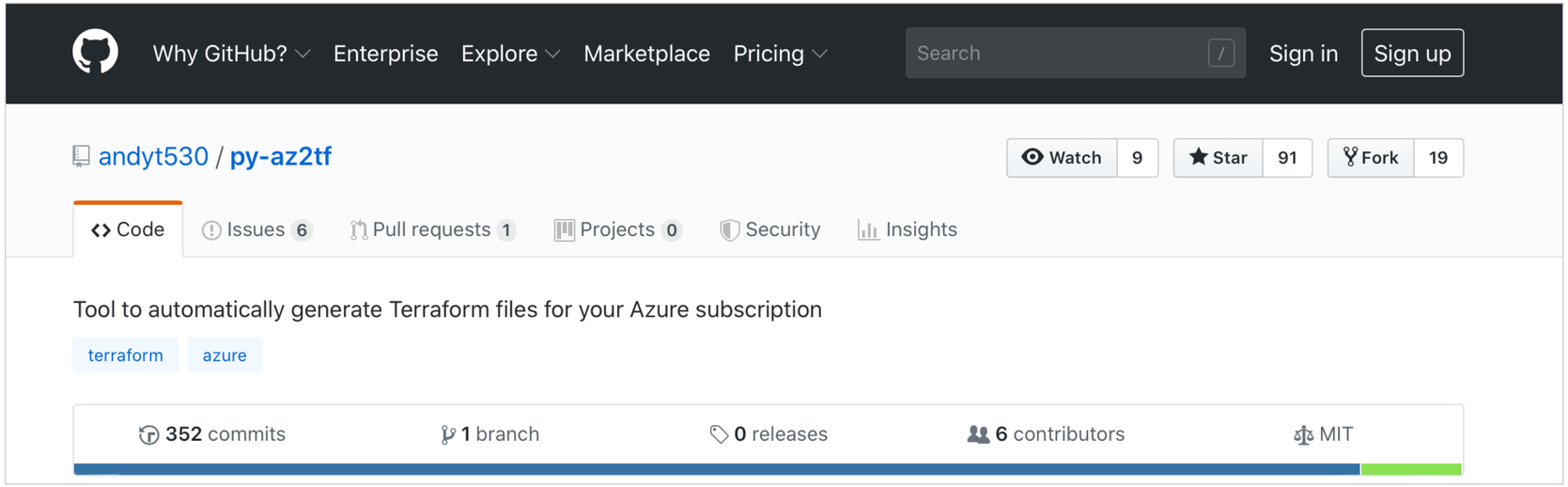
```
terraform import azurerm_resource_group.mygroup /subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myresourcegroup
```

Demo

Working with your existing infrastructure

Bulk Importing

Import & generate code for a Resource Group or a Subscription



github.com/andyt530/py-az2tf

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

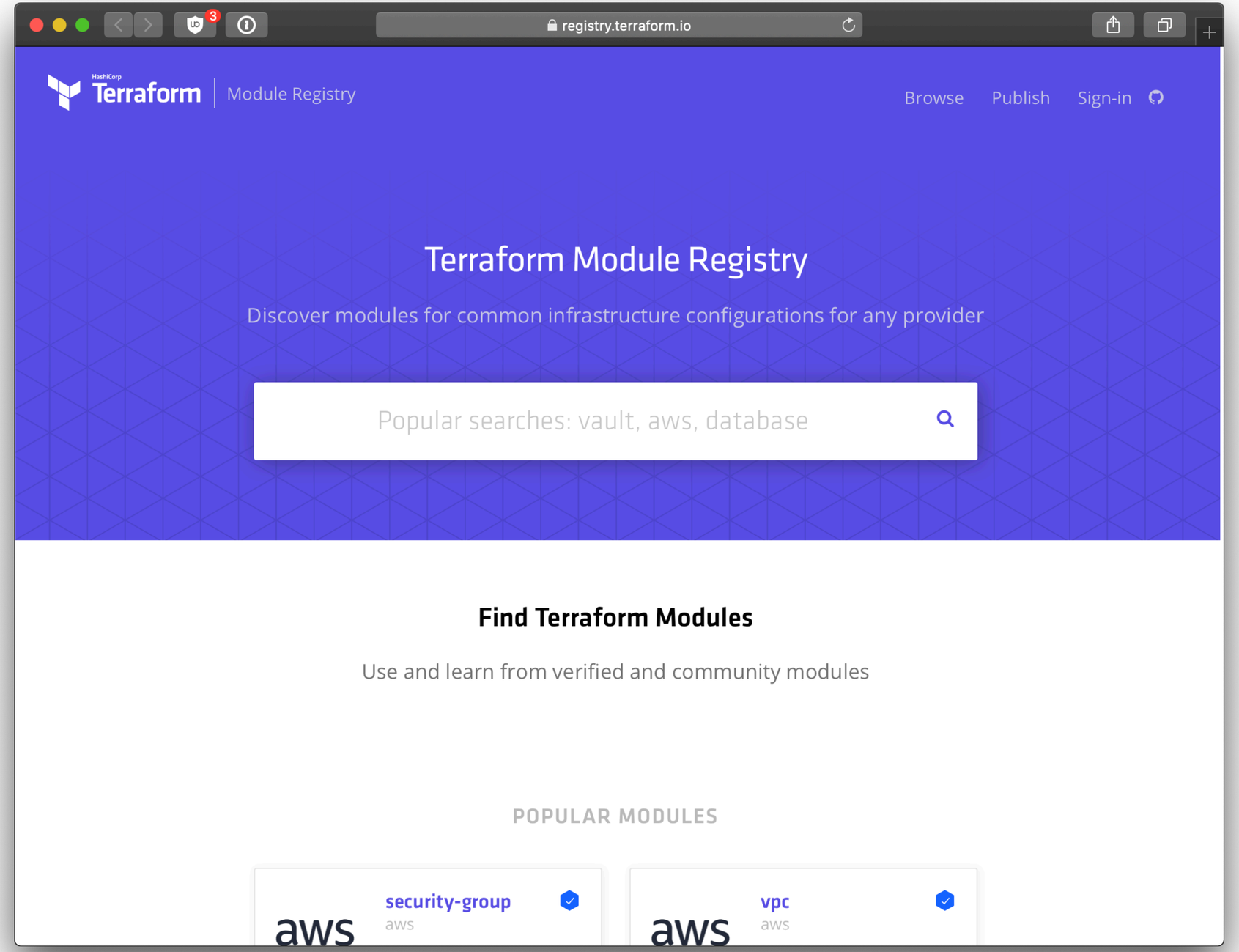
Reusing your Terraform Configurations

```
# 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"
}
```

Module Registry

Pre-created Terraform Modules



Demo

Reusable code with Modules in Terraform

Terraform’s Statefile

Storing your State file

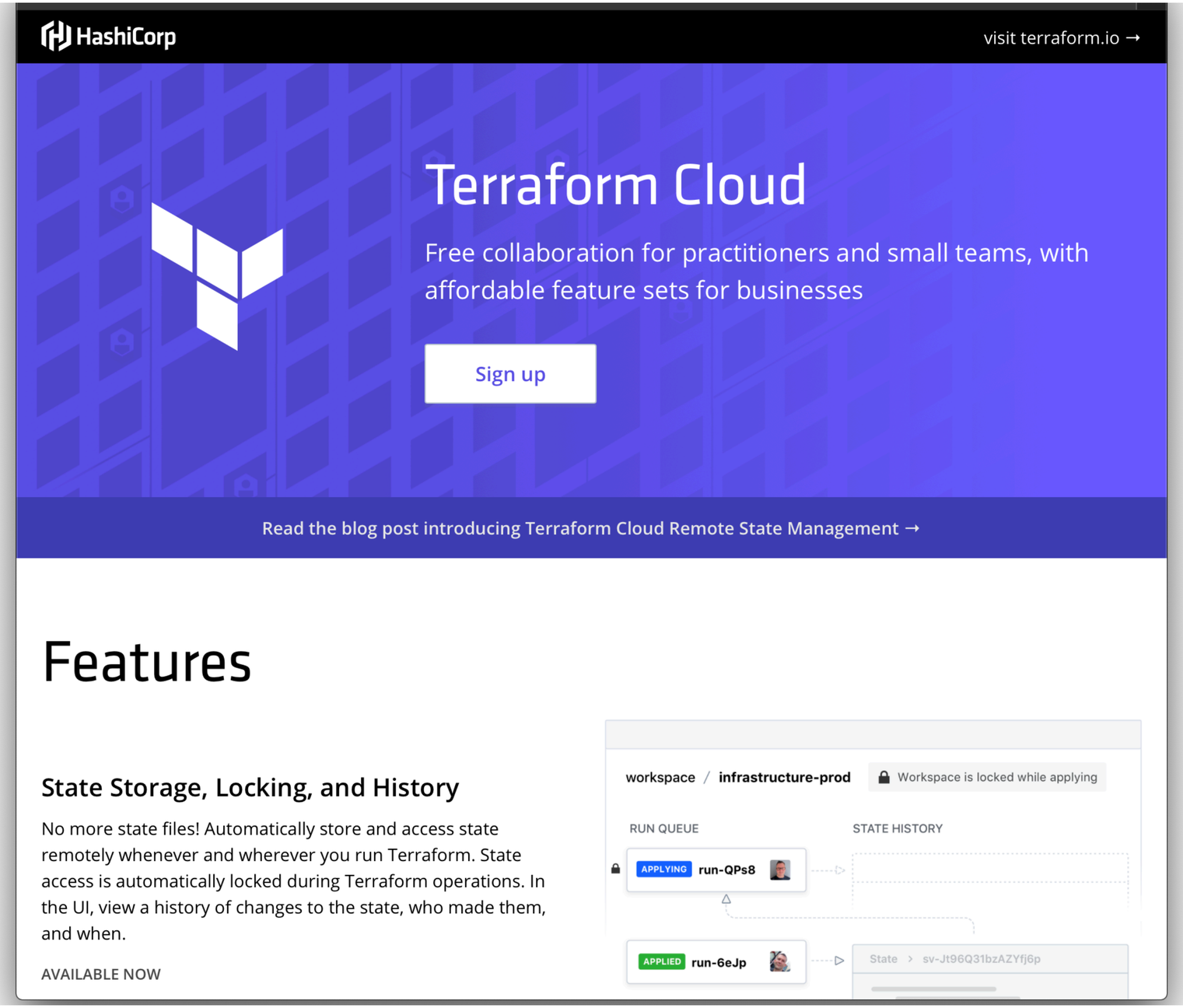
The Statefile

Terraform’s Statefile

```
{
  "version": 3,
  "terraform_version": “0.12.14",
  "serial": 1,
  "lineage": "1205acf8-bdb3-1353-e4d9-65093e16e7ef",
  "modules": [
    {
      "path": [
        "root"
      ],
      "outputs": {},
      "resources": {
        "azurerm_resource_group.test": {
          "type": "azurerm_resource_group",
          "depends_on": [],
          "primary": {
```

Terraform Cloud

Stores your Terraform State files for you



Terraform Cloud

Stores your Terraform Statefile in Terraform Cloud

```
terraform {
  backend "remote" {
   organization = "terraform-azure"

    workspaces {
      name = "example-app"
    }
  }
}
```

AzureRM Backend

Stores your Terraform Statefile in Blob Storage

```
terraform {
  backend "azurerm" {
    storage_account_name = "abcd1234"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"
  }
}
```

What's supported?

Which resources does Terraform support?

What's supported in Terraform?

- AKS / Kubernetes
- App Service
- Application Gateway
- Application Insights
- API Management
- AutoScale Setting
- Automation
- AzureAD
- Azure Monitor
- Batch
- CDN
- Cognitive Services
- Container Instance
- Container Registry
- CosmosDB
- Data Lake Analytics
- Data Lake Store
- DataBricks
- Dev Test Labs
- DevSpaces
- DNS
- EventGrid
- EventHub
- Express Route
- Firewall
- Function Apps
- HDInsight
- Healthcare
- IoTHub
- Key Vault
- Kusto
- Load Balancers
- Local Network Gateway
- Log Analytics / OMS
- Logic Apps
- Management Group
- Management Locks
- Media Services
- MySQL
- Networks
- Notification Hubs
- Policy
- PostgreSQL
- Recovery Services
- Redis
- Relay
- SQL Azure
- Scheduler Job
- Search
- Security Center
- Service Fabric
- ServiceBus
- Storage
- VM Scale Sets
- Virtual Machines

Blob Storage

Uploading files to Blob Storage from Terraform

```
resource "azurerm_resource_group" "test" { ... }
resource "azurerm_storage_account" "test" { ... }
resource "azurerm_storage_container" "test" { ... }

resource "azurerm_storage_blob" "source" {
  name = "local-file.zip"
  resource_group_name    = azurerm_resource_group.test.name
  storage_account_name   = azurerm_storage_account.test.name
  storage_container_name = azurerm_storage_container.test.name
  type                   = "block"
  source                 = "local-file.zip"
}
```

Load Balancer

Defining Load Balancers in a modular fashion

```
resource "azurerm_resource_group" "test" { ... }
resource "azurerm_public_ip" "test" { ... }

resource "azurerm_lb" "test" {
  name                = "example-loadbalancer"
  location            = azurerm_resource_group.test.location
  resource_group_name = azurerm_resource_group.test.name

  frontend_ip_configuration {
    name                 = "first"
    public_ip_address_id = azurerm_public_ip.test.id
  }
}

resource "azurerm_lb_probe" "test" {
  name                = "example-probe"
  location            = azurerm_resource_group.test.location
  resource_group_name = azurerm_resource_group.test.name
  loadbalancer_id     = azurerm_lb.test.id
  port                = 22
}
```

Other Resources

What about something that's not natively supported?

What about something that's not natively supported?

- Terraform has a resource for deploying ARM Templates in the form of the `azurerm_template_deployment` resource.
- This allows resources which aren't supported by Terraform to be provisioned by Terraform
- Integration extends to passing variables in both directions:
  - Variables from Terraform can be passed into an ARM Template
  - Outputs from an ARM Template can be accessed in Terraform
- Downsides: Terraform's not able to diff resources deployed via an ARM Template

Template Deployments

Deploying an ARM Template in Terraform

```
resource "azurerm_resource_group" "test" { ... }

resource "azurerm_template_deployment" "test" {
  name                = "example-deployment"
  resource_group_name = azurerm_resource_group.test.name
  deployment_mode     = "Complete"

  template_body = <<DEPLOY
{
  "$schema": "https://schema.management.azure.com/schemas/
2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  # ...
}
DEPLOY
}
```

Template Deployments

Deploying an ARM Template in Terraform

```
resource "azurerm_resource_group" "test" { ... }

resource "azurerm_template_deployment" "test" {
  name                = "example-deployment"
  resource_group_name = azurerm_resource_group.test.name
  template_body       = file("template.json")
  deployment_mode     = "Complete"
}
```

Template Deployments

Deploying an ARM Template in Terraform

```
resource "azurerm_resource_group" "test" { ... }

resource "azurerm_template_deployment" "test" {
  name                = "example-deployment"
  resource_group_name = azurerm_resource_group.test.name
  template_body       = file("template.json")
  deployment_mode     = "Complete"
}
```

Other options?

- Terraform supports Provisioners - which can run any action locally or remotely (e.g. via SSHing/WinRM into a VM)
- These allow integration with external scripts or tooling, such as the Azure CLI / PowerShell
- Again: Terraform can't perform diffing of resources provisioned via this method

Provisioners

Calling the Azure CLI using a Provisioner

```
resource "null_resource" "provision-shiny-thing" {
  provisioner "local-exec" {
    command = "az ..."
  }
}
```

Azure Provider 2.0

What’s coming over the next few months

Azure Provider 2.0

Announcement / details available on Github

terraform-providers / terraform-provider-azurerm

Watch 160 | Star 1.2k | Fork 1.2k

Code | Issues 643 | Pull requests 55 | Security | Insights

Announcement: Upcoming Changes in Version 2.0 of the Azure Provider #2807

New issue

Open **tombuildsstuff** opened this issue on 30 Jan · 24 comments

tombuildsstuff commented on 30 Jan • edited — Contributor

Terraform initially shipped support for the AzureRM Provider back in December 2015. Since then we've added support for 191 Resources, 58 Data Sources and have launched a couple of related Providers in the form of the Azure Active Directory Provider and the Azure Stack Provider.

Version 2.0 of the AzureRM Provider will be a **Major Release** - in that it will include some larger-scale changes not seen in a regular release. A summary of these changes is outlined below - however a full breakdown will be available on the Terraform Website after the release of v1.22.

Summary

- Existing Resources will be required to be imported
- Custom Timeouts will be available on Resources - this will allow you to specify a custom timeout for provisioning the resource in your Terraform Configuration using the `timeouts` block.
- New resources for Virtual Machines and Virtual Machine Scale Sets
- Removing Fields, Data Sources and Resources which have been deprecated

A brief summary of each item can be found below - more details will be available in the Azure Provider 2.0 upgrade guide on the Terraform Website once v1.22 has been released.

Existing Resources will be required to be Imported

Terraform allows for existing resources which have been created outside of Terraform to be Imported into Terraform's State. Once a resource is imported into the state, it's possible for Terraform to track changes and manage this resource. The Azure Provider allows Importing existing

Assignees: tombuildsstuff

Labels: breaking-change

Projects: None yet

Milestone: v2.0.0

13 participants

What’s coming over the next few months?

Azure Provider 2.0

- Custom Timeouts

Azure Provider 2.0

Custom Timeouts

- At the moment all API calls are hard-limited to six hours
- v2.0 of the Azure Provider will allow you to specify the timeout on resources

Azure Provider 2.0

Custom Timeouts

```
resource "azurerm_resource_group" "test" {
  name     = "example-resource-group"
  location = "West Europe"
}
```

Azure Provider 2.0

Custom Timeouts

```
resource "azurerm_resource_group" "test" {
  name     = "example-resource-group"
  location = "West Europe"

  timeouts {
    create = "10m"
    delete = "30m"
  }
}
```

What’s coming over the next few months?

Azure Provider 2.0

- Custom Timeouts
- Requiring Imports

Azure Provider 2.0

Requiring Imports

- As seen earlier - Azure using the `name` as the Resource ID and having API’s which are Upserts means it’s possible to intentionally import an existing resource into Terraform

Azure Provider 2.0

Requiring Imports

- As seen earlier - Azure using the `name` as the Resource ID and having API's which are Upserts means it's possible to intentionally import an existing resource into Terraform
- However some API's require that a separate Update API is called when the resource is being changed, as such users can see unhelpful error messages

Azure Provider 2.0

Requiring Imports

- As seen earlier - Azure using the `name` as the Resource ID and having API's which are Upserts means it's possible to intentionally import an existing resource into Terraform
- However some API's require that a separate Update API is called when the resource is being changed, as such users can see unhelpful error messages
- To work around this we're going to check for an existing resource with the same name as each resource is created, and then require that these resources are Imported

Azure Provider 2.0

Requiring Imports

- As seen earlier - Azure using the `name` as the Resource ID and having API's which are Upserts means it's possible to intentionally import an existing resource into Terraform
- However some API's require that a separate Update API is called when the resource is being changed, as such users can see unhelpful error messages
- To work around this we're going to check for an existing resource with the same name as each resource is created, and then require that these resources are Imported
- You can opt into this behaviour from v1.22 onwards, enhancements to come but will be fully available in 2.0

What's coming over the next few months?

Azure Provider 2.0

- Custom Timeouts
- Requiring Imports
- New VM / VM Scale Set Resources

Azure Provider 2.0

New Virtual Machine / VM Scale Set Resources

```
resource "azurerm_linux_virtual_machine_scale_set" "test" {
  name                = "example-vmss"
  resource_group_name = "example-resources"
  location            = "West Europe"
  sku                 = "Standard_F2"
  instances           = 3
  admin_username      = "adminuser"

  os_disk {
    storage_account_type = "Standard_LRS"
    caching              = "ReadWrite"
  }

  ssh_key {
    public_key = file("~/.ssh/id_rsa.pub")
  }

  ...
}
```

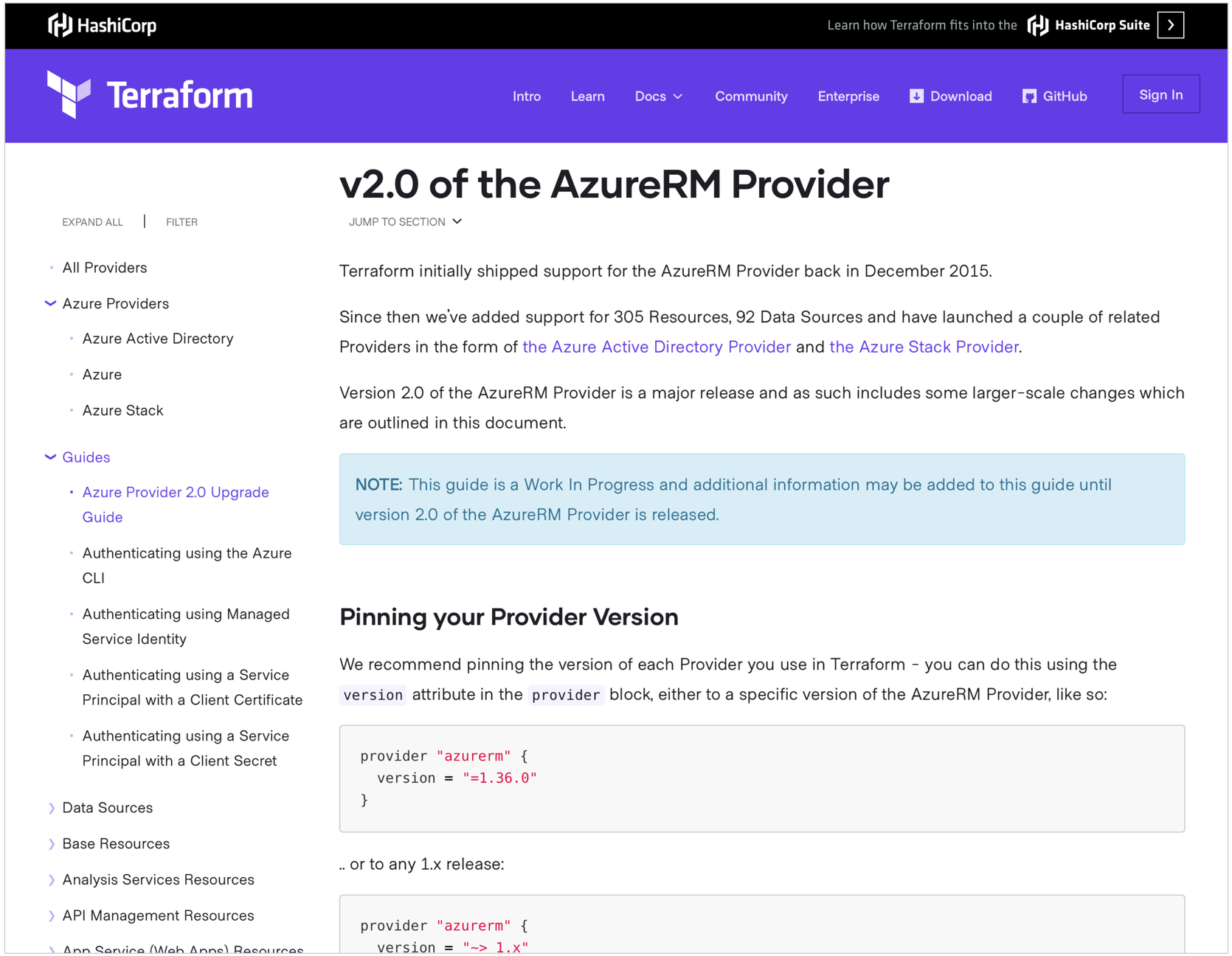
What's coming over the next few months?

Azure Provider 2.0

- Custom Timeouts
- Requiring Imports
- New VM / VM Scale Set Resources
- Removing deprecated fields/resources

Azure Provider 2.0

Complete guide available on terraform.io



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

Resources

- Slides / Demo’s: github.com/tombuildsstuff/buildstuff2019-tfazure
- Azure -> Terraform (az2tf): github.com/andyt530/py-az2tf
- Terraform Docs/Cloud: terraform.io

Thank you.

HashiCorp

www.hashicorp.com hello@hashicorp.com