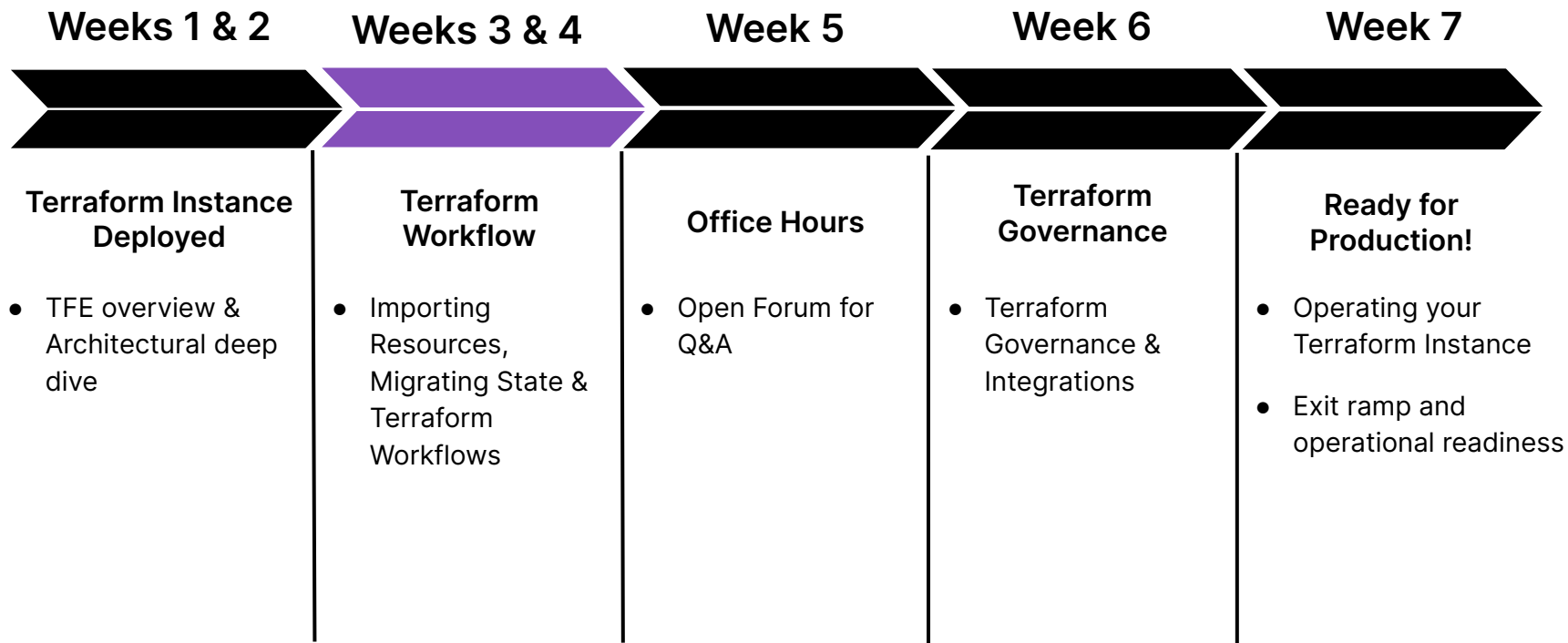


Importing Resources & Migrating State into Terraform Enterprise

TFE Path to Production



Agenda

Terraform Basics 01

Importing Existing Infrastructure 02

Migrating from OSS to Enterprise 03



01

Terraform Basics



Infrastructure as Code

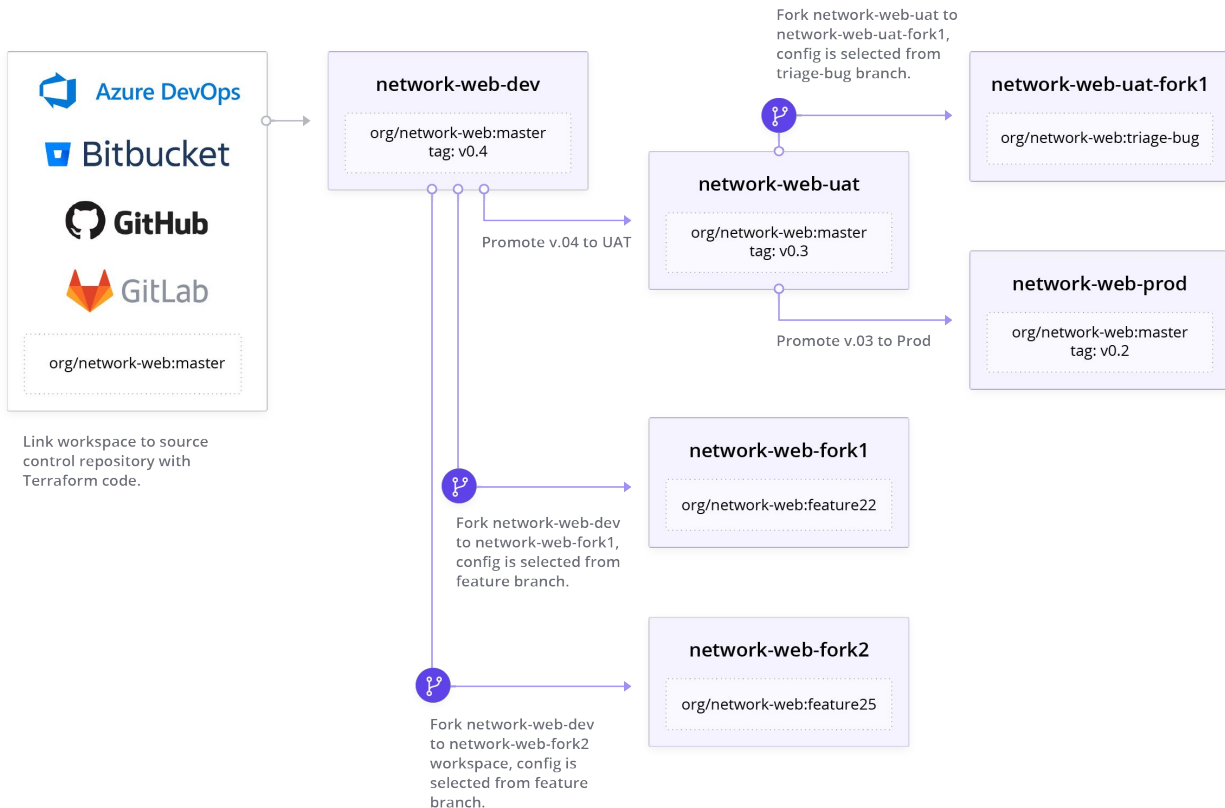
Using HashiCorp Configuration Language (HCL) infrastructure and services from any provider can be provisioned in a codified, secure, and automated fashion

- HCL is human readable and machine executable
- HCL is a declarative, Turing-Complete language
- Used to automate, version, and collaborate on infrastructure

```
resource "google_compute_instance" "svr" {  
  name          = "server"  
  machine_type  = "g1-small"  
  zone          = "us-central1-a"  
  disk {  
    image = "ubuntu-1404-trusty-v20160114e"  
  }  
}  
  
resource "dnsimple_zone_record" "hello" {  
  zone_name = "example.com"  
  name      = "server"  
  value     =  
google_compute_instance.svr.network_interface.0.  
address  
  type      = "A"  
}
```

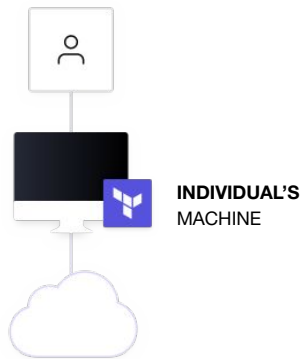
Benefits of Infrastructure as Code

- Versioning
- Collaboration
- Promotion
- Forking



Ways to Interact with Terraform

CLI Terraform CLI



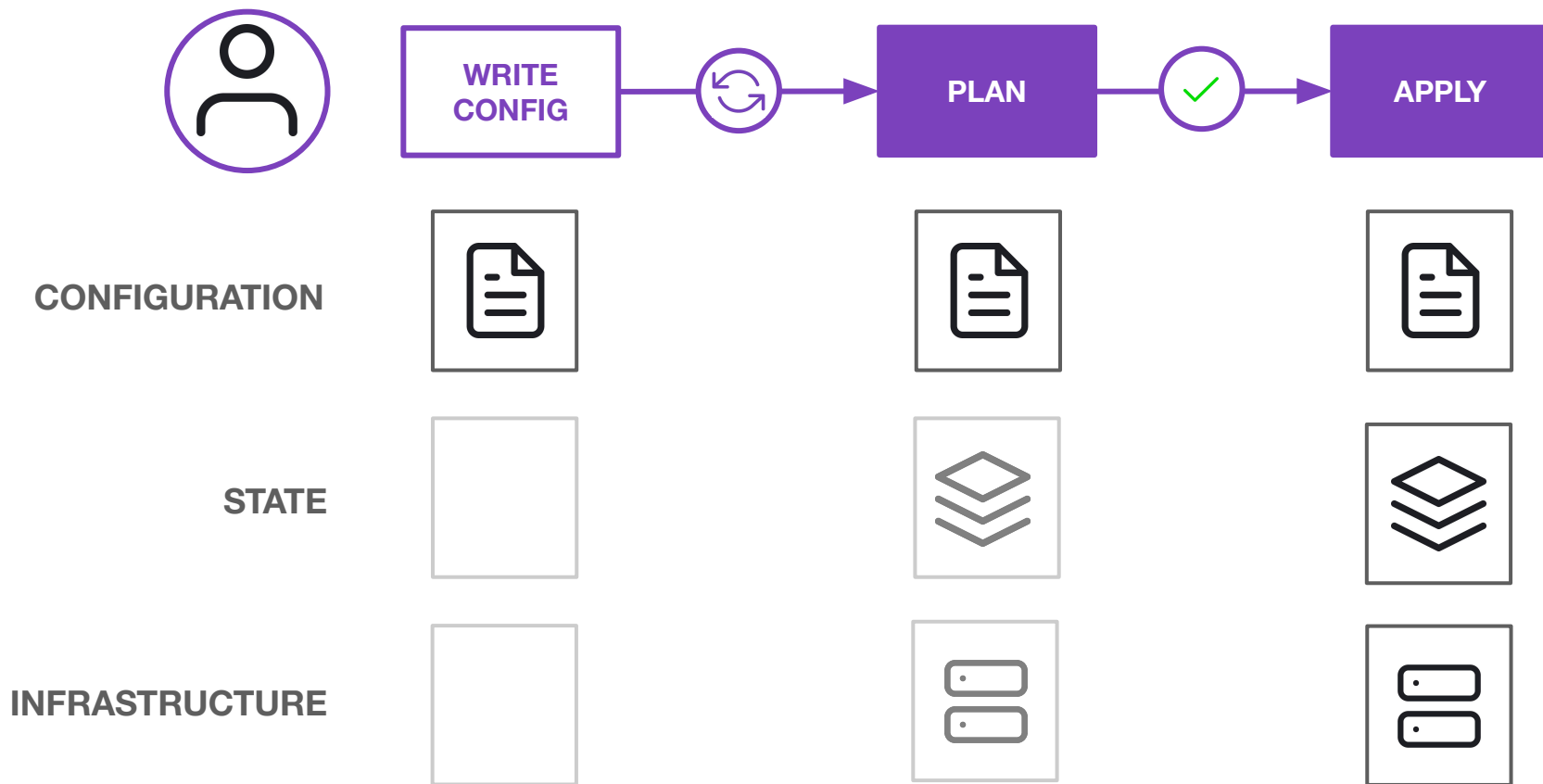
- No requirements for collaboration
- No requirements for centralized reusable configs
- No policy or governance requirements

Enterprise / Cloud Self-Hosted / SaaS



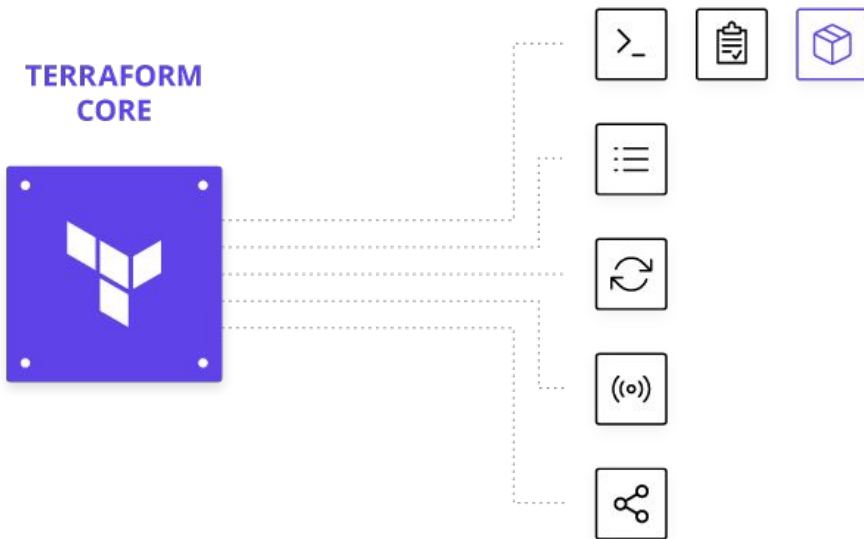
- Air gapped infrastructure and applications
- Data sovereignty requirements
- Regulatory compliance requirements
- Stringent reliability and availability requirements

Terraform Workflow



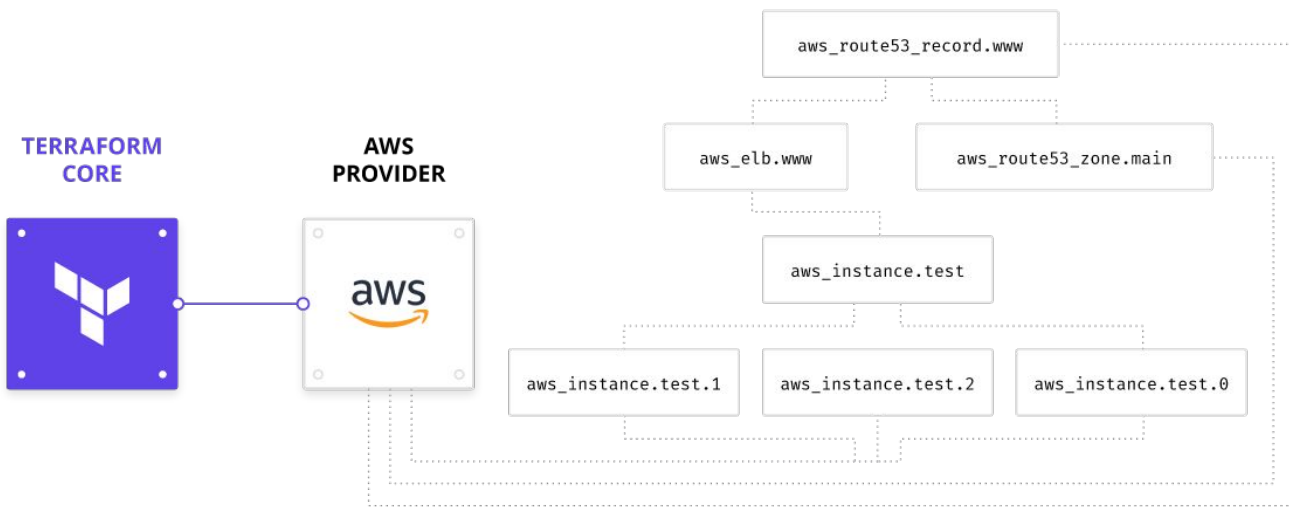
Terraform Core Engine

- The “engine” driving Terraform Enterprise
- Loads providers as needed
- OSS [hosted on Github](#)
- Responsible for:
 - Reading and Interpolating configuration files and modules
 - State Management
 - Executing plan
 - Communicating with providers
 - Constructing resource graphs



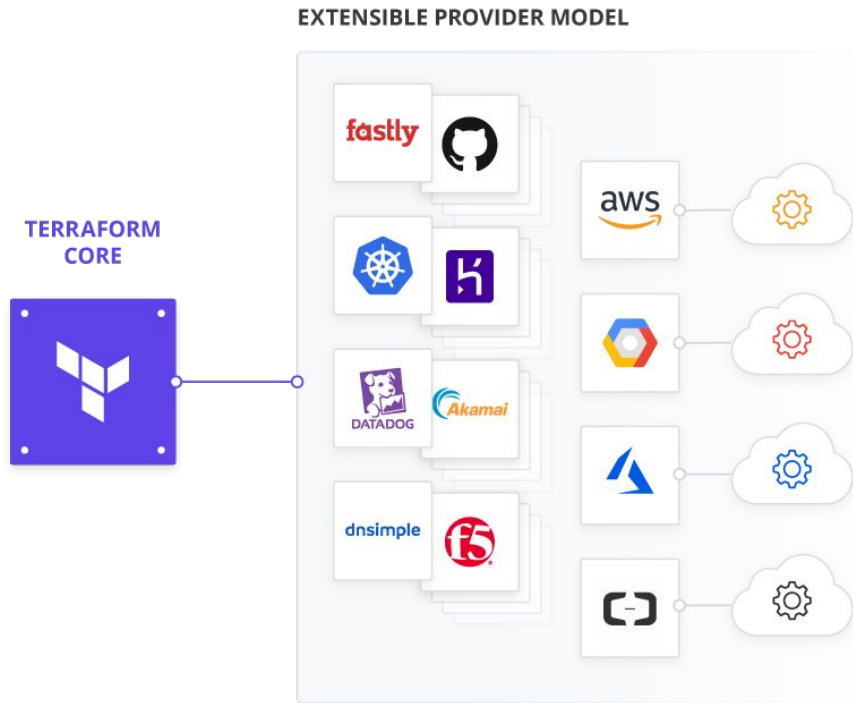
Resource Graph

- Safely provision and change infrastructure
- See planned infrastructure changes before execution
- No need to manually coordinate dependent resources



Provider Plugins

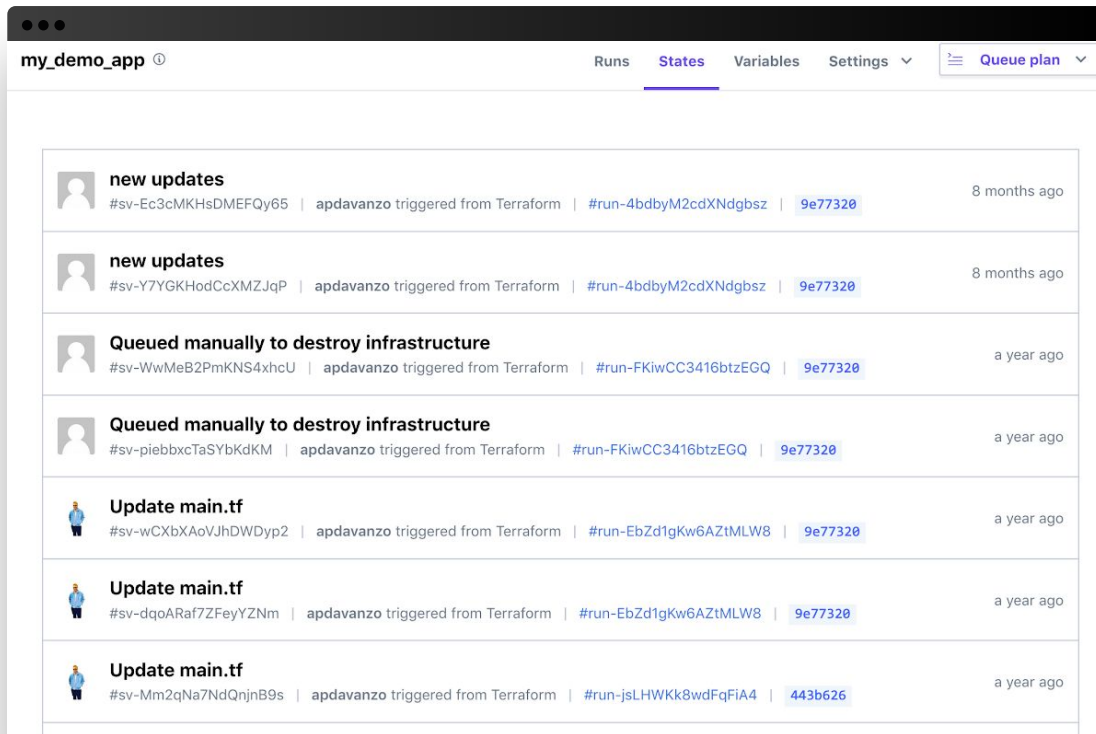
- Expose implementation for specific services
- Offer extensible layer for 'Core' to learn how to talk to anything with an API without any upgrades
- Responsible for:
 - Initializing libraries for API calls
 - Authenticating with the provider
 - Defining resources that map to services
 - Executing commands or scripts for designated resources



Infrastructure State

State is Terraform's understanding of an infrastructure

- Used by Terraform to provide an understanding of resources under its control
- TFE can use it to show where previous runs have made changes to infrastructure
- TFE provides remote state management which encrypts the state file



Managing Existing Infrastructure with TFE



Migrate existing
state from
Terraform OSS to
Terraform
Enterprise

**Is the
infrastructure
managed by
Terraform?**



Import
unmanaged cloud
resources into
Terraform
Enterprise

02

Import Unmanaged Cloud Resources

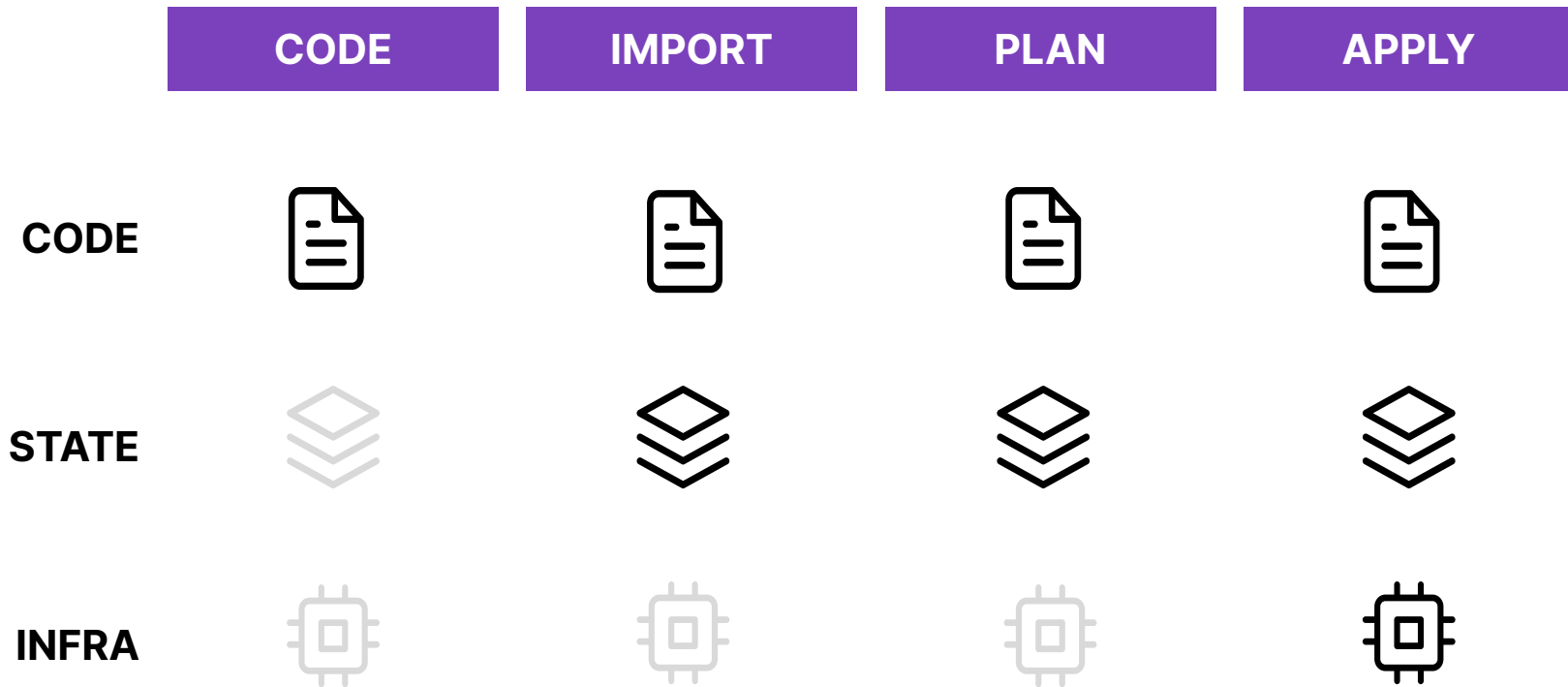


Terraform Import

- Terraform is able to import existing infrastructure
- Importing takes resources created by some other method and bring it under Terraform management
- This is a great way to slowly transition infrastructure to Terraform



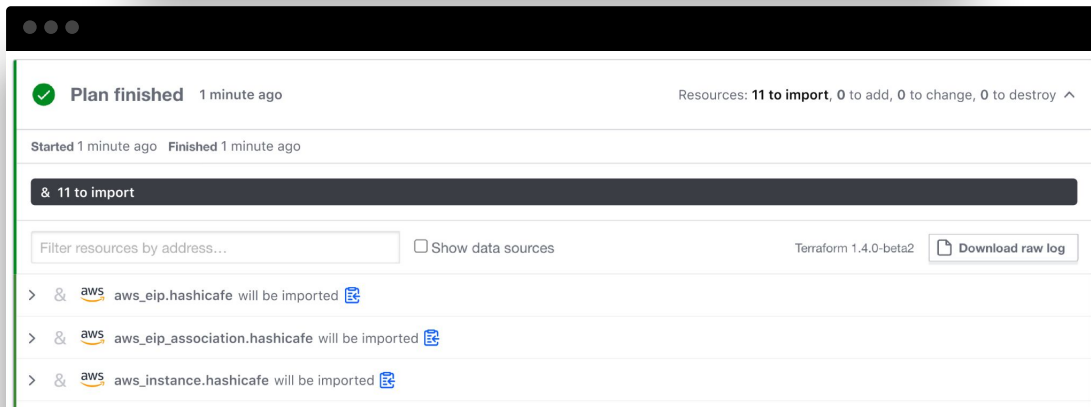
Terraform Import Workflow



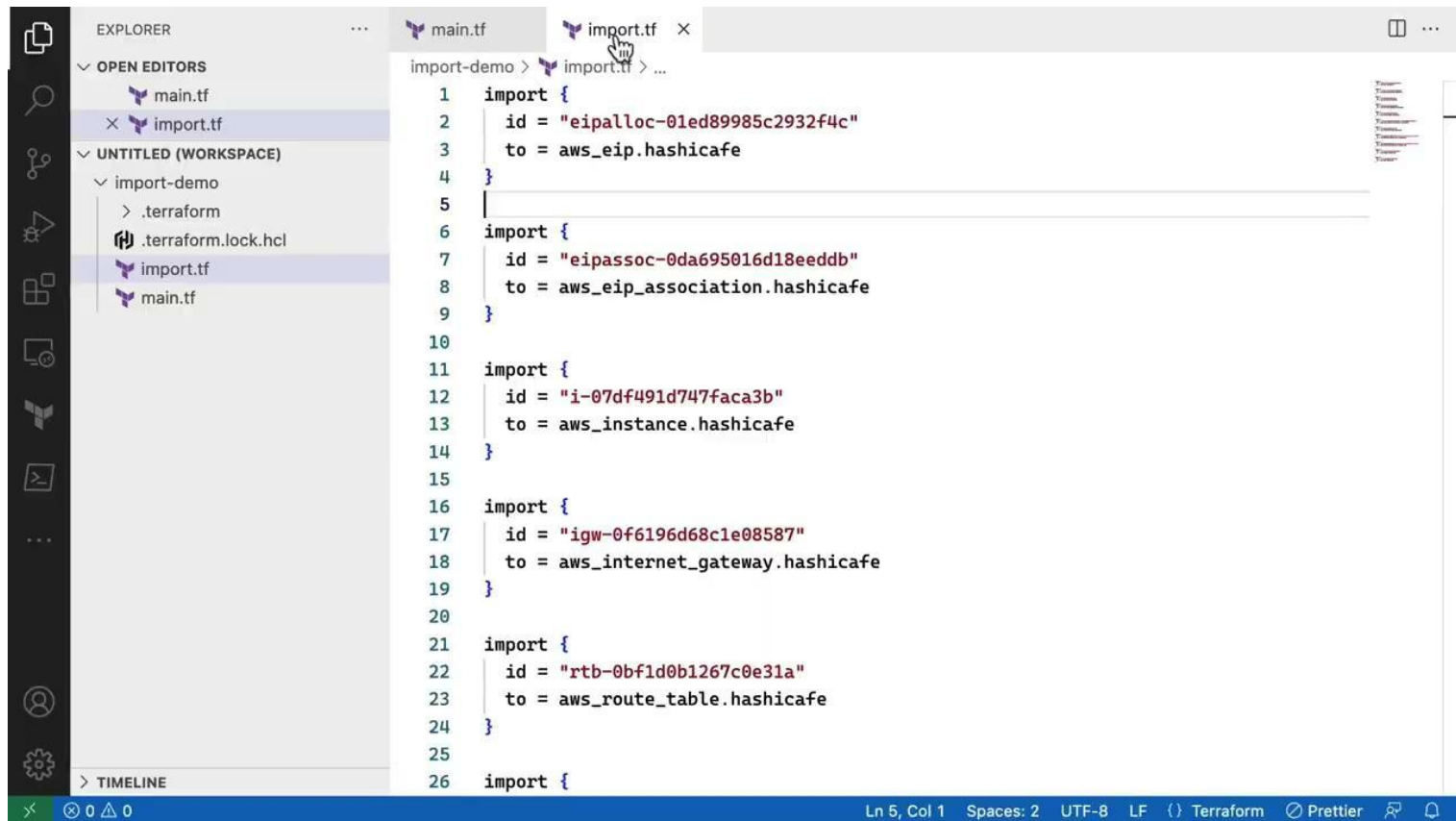
Config-driven Import

- Codify import operations in bulk
- Safer onboarding: review planned import operations before adding to state
- **Auto-generated code for resources**

```
import {  
  id = "i-087f3718ec77cfecb"  
  to = aws_instance.app  
}
```



Config-driven Import



The screenshot shows the Visual Studio Code interface with the Explorer and Open Editors views. The Explorer view on the left shows a workspace with an 'import-demo' folder containing '.terraform', '.terraform.lock.hcl', 'import.tf', and 'main.tf'. The Open Editors view shows 'main.tf' and 'import.tf' open. The 'import.tf' file is selected, and its content is displayed in the editor. The code defines five Terraform import blocks for various AWS resources, each with a specific ID and a target resource name.

```
import-demo > import.tf > ...
1  import {
2    id = "eipalloc-01ed89985c2932f4c"
3    to = aws_eip.hashicafe
4  }
5
6  import {
7    id = "eipassoc-0da695016d18eeddb"
8    to = aws_eip_association.hashicafe
9  }
10
11 import {
12   id = "i-07df491d747faca3b"
13   to = aws_instance.hashicafe
14 }
15
16 import {
17   id = "igw-0f6196d68c1e08587"
18   to = aws_internet_gateway.hashicafe
19 }
20
21 import {
22   id = "rtb-0bf1d0b1267c0e31a"
23   to = aws_route_table.hashicafe
24 }
25
26 import {
```

Ln 5, Col 1 Spaces: 2 UTF-8 LF () Terraform Prettier





Importing Prerequisites

Before you can import existing infrastructure into Terraform Enterprise you must have completed the following:

- Installed [Terraform CLI](#) locally
- Deployed Terraform Enterprise
- [Tutorial](#)





Steps

1. Write or generate Terraform code that matches your infrastructure
2. Import infrastructure into your Terraform state file using **terraform import**
3. Review plan output from **terraform plan** to ensure the configuration matches expected state
4. Apply the configuration to update your Terraform state by running **terraform apply**

Step 1

Write Terraform code that matches your infrastructure

```
[main.tf]

provider "aws" {
  region = "eu-west-1"
}

resource "aws_vpc" "testvpc" {
  cidr_block = "10.0.0.0/16"
}

output "vpcid" {
  value = aws_vpc.testvpc.id
}
```

Step 2

Initialize Terraform with **terraform init**

```
$ terraform init
```

```
Initializing the backend...
```

```
Initializing provider plugins...
```

- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.68.0...
- Installed hashicorp/aws v3.68.0 (signed by HashiCorp)

Terraform has created a lock file `.terraform.lock.hcl` to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

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.



Step 3

Import infrastructure into your Terraform state file using **terraform import**

```
$ terraform import aws_vpc.testvpc "vpc-aabbccdd"
```

```
aws_vpc.testvpc: Importing from ID "vpc-aabbccdd"...
```

```
aws_vpc.testvpc: Import prepared!
```

```
  Prepared aws_vpc for import
```

```
aws_vpc.testvpc: Refreshing state... [id=vpc-aabbccdd]
```

```
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.
```



Step 4

Review the plan output from **terraform plan** to ensure the configuration matches expected state

```
$ terraform plan
```

```
aws_vpc.testvpc: Refreshing state... [id=vpc-aabbccdd]
```

```
Changes to Outputs:
```

```
  + vpcid = "vpc-aabbccdd"
```

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

Note: You didn't use the `-out` option to save this plan, so Terraform can't guarantee to take exactly these actions if you run `"terraform apply"` now.



Step 5

Apply the configuration to update your Terraform state by running **terraform apply**

```
$ terraform apply
```

```
aws_vpc.testvpc: Refreshing state... [id=vpc-aabbccdd]
```

```
Changes to Outputs:
```

```
  + vpcid = "vpc-aabbccdd"
```

```
You can apply this plan to save these new output values to the Terraform state, without  
changing any real infrastructure.
```

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

```
Outputs:
```

```
vpcid = "vpc-aabbccdd"
```

03

Migrating Existing State



Prerequisites

Before you can migrate Terraform OSS state into Terraform Enterprise you must have completed the following:

- Installed [Terraform CLI](#) locally
- Deployed Terraform Enterprise
- Optionally: Acquired an API token for Terraform Enterprise



Migration Steps

Before

- Take a backup!
- Ensure that you have initialized your existing state
- Create and configure the Workspace in Terraform Enterprise



During

- Login to Terraform Enterprise and generate an API Token
- Add the Terraform remote backend
- Reinitialize Terraform and confirm state migration



After

- Verify that the state has been migrated to the workspace
- Move old state (to another backup)
- Trigger a remote run within Terraform Enterprise
- Check everything worked as expected

Before

Review the existing Terraform code

```
[main.tf]

terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = ">= 3.68.0"
    }
  }
  required_version = ">= 1.0.7"
}

provider "aws" {
  region = "eu-west-1"
}

resource "aws_vpc" "testvpc" {
  cidr_block = "10.0.0.0/16"
}

output "vpcid" {
  value = aws_vpc.testvpc.id
}
```

Before

Create and configure the workspace to include AWS credentials

The screenshot shows the 'hashicat-aws' workspace page in Terraform Cloud. The breadcrumb trail is 'hashicorp-jennawong / Projects & workspaces / hashicat-aws / Variables'. The workspace details show 0 resources, Terraform version 1.2.5, and it was updated 10 months ago. It is currently 'Unlocked'. An 'Actions' dropdown menu is visible.

Variables

Terraform uses all [Terraform](#) and [Environment](#) variables for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any `*.auto.tfvars` files in the configuration. You may want to use the Terraform Cloud Provider or the variables API to add multiple variables at once.

Sensitive variables

[Sensitive](#) variables are never shown in the UI or API, and can't be edited. They may appear in Terraform logs if your configuration is designed to output them. To change a sensitive variable, delete and replace it.

Workspace variables (0)

Variables defined within a workspace always overwrite variables from variable sets that have the same type and the same key. Learn more about variable set [precedence](#).

Key	Value	Category
There are no variables added.		

Select variable category

☒ **Terraform variable**
These variables should match the declarations in your configuration. Click the HCL box to use interpolation or set a non-string value.

☐ **Environment variable**
These variables are available in the Terraform runtime environment.

Key	Value	
<input type="text" value="key"/>	<input type="text" value="value"/>	<input type="checkbox"/> HCL <input type="checkbox"/> Sensitive
<p>Description (Optional)</p> <input type="text" value="description (optional)"/>		



Step 1

Login to
Terraform
Enterprise and
generate an API
Token

```
$ terraform login
```

```
Terraform will request an API token for tfe.mycompany.com using your browser.
```

```
If login is successful, Terraform will store the token in plain text in  
the following file for use by subsequent commands:  
    /home/demouser/.terraform.d/credentials.tfrc.json
```

```
Do you want to proceed?
```

```
Only 'yes' will be accepted to confirm.
```

```
Enter a value: yes
```

```
-----  
Terraform must now open a web browser to the tokens page for tfe.mycompany.com.
```

```
If a browser does not open this automatically, open the following URL to proceed:  
    https://tfe.mycompany.com/app/settings/tokens?source=terraform-login
```

```
-----  
Generate a token using your browser, and copy-paste it into this prompt.
```

```
Terraform will store the token in plain text in the following file  
for use by subsequent commands:  
    /home/demouser/.terraform.d/credentials.tfrc.json
```

```
Token for tfe.mycompany.com:
```

```
Enter a value:
```

```
Retrieved token for user demouser
```

Step 2

Add the
Terraform Remote
Backend

```
[main.tf]

terraform {
  backend "remote" {
    hostname = "app.terraform.io"
    organization = "company"
    workspaces {
      name = "my-app-prod"
    }
  }
}
```


Step 3

Reinitialize
Terraform and
confirm state
migration

```
$ terraform init
```

```
Initializing Terraform Cloud...
```

```
Do you wish to proceed?
```

```
As part of migrating to Terraform Cloud, Terraform can optionally copy your  
current workspace state to the configured Terraform Cloud workspace.
```

```
Answer "yes" to copy the latest state snapshot to the configured  
Terraform Cloud workspace.
```

```
Answer "no" to ignore the existing state and just activate the configured  
Terraform Cloud workspace with its existing state, if any.
```

```
Should Terraform migrate your existing state?
```

```
Enter a value:
```

After

Verify the workspace exists and that the state file has been uploaded

hashicorp-jennawong / Projects & workspaces / hashicat-azure / States / sv-eTk9ndeyesHHxxNv

hashicat-azure

ID: ws-9kPMaTETXQtzHEP

No workspace description available. [Add](#) workspace description.

🔒 Unlocked Actions ▾

Resources 9 **Terraform version** 1.1.4 **Updated** a year ago

New state #sv-eTk9ndeyesHHxxNv
jennawong triggered from Terraform a year ago

Download Newer

filter Apply [Learn more about filtering JSON data.](#) Expand Full screen

```
1 {
2   "version": 4,
3   "terraform_version": "1.1.4",
4   "serial": 0,
5   "lineage": "7aa4d8d5-e4bf-ccee-812c-9a0dc3b54521",
6   "outputs": {},
7   "resources": [
8     {
9       "mode": "managed",
10      "type": "azurerm_network_interface",
11      "name": "catapp-nic",
12      "provider": "provider[\"registry.terraform.io/hashicorp/azurerm\"]",
13      "instances": []
14    },
15    {
16      "mode": "managed",
17      "type": "azurerm_network_interface_security_group_association",
```

[Advanced](#)

Trigger a remote run within Terraform Enterprise

```
$ mv terraform.tfstate terraform.tfstate.local
```

```
$ terraform apply
```

Running apply in the remote backend. Output will stream here. Pressing Ctrl-C will cancel the remote apply if it's still pending. If the apply started it will stop streaming the logs, but will not stop the apply running remotely.

Preparing the remote apply...

To view this run in a browser, visit:

<https://tfe.mycompany.com/app/myorganization/state-migration/runs/run-64iLttGfK5eSLJ3F>

Waiting for the plan to start...

```
Terraform v1.0.11
on linux_amd64
Configuring remote state backend...
Initializing Terraform configuration...
aws_vpc.testvpc: Refreshing state... [id=vpc-01feeac01bbb6af51]
```

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply":

```
# aws_vpc.testvpc has been changed
~ resource "aws_vpc" "testvpc" {
    id                = "vpc-01feeac01bbb6af51"
+ tags              = {}
  # (15 unchanged attributes hidden)
}
```

Unless you have made equivalent changes to your configuration, or ignored the relevant attributes using `ignore_changes`, the following plan may include actions to undo or respond to these changes.

No changes. Your infrastructure matches the configuration.

Verify a remote run has been triggered by the CLI

hashicorp-jennawong / Projects & workspaces / hashicat-aws / Runs

hashicat-aws

ID: ws-58Trc2hhBJ5rbWse

No workspace description available. [Add workspace description.](#)

Unlocked


Resources
0

Terraform version
1.2.5

Updated
10 months ago

Actions

Current Run

 **Triggered via CLI** CURRENT


#run-1TGAnREM9SCCTpEW | destroy run | jennawong triggered via CLI

✓ Applied
10 months ago

Run List


All 5 Needs Attention 0 Errored 2 Running 0 On Hold 0 Success 2

Q Status Operation Source

 **Triggered via CLI** CURRENT

#run-1TGAnREM9SCCTpEW | destroy run | jennawong triggered via CLI

✓ Applied
10 months ago

 **Triggered via CLI**

#run-VixX8quT9sKBxsSM | jennawong triggered via CLI

✓ Applied
10 months ago

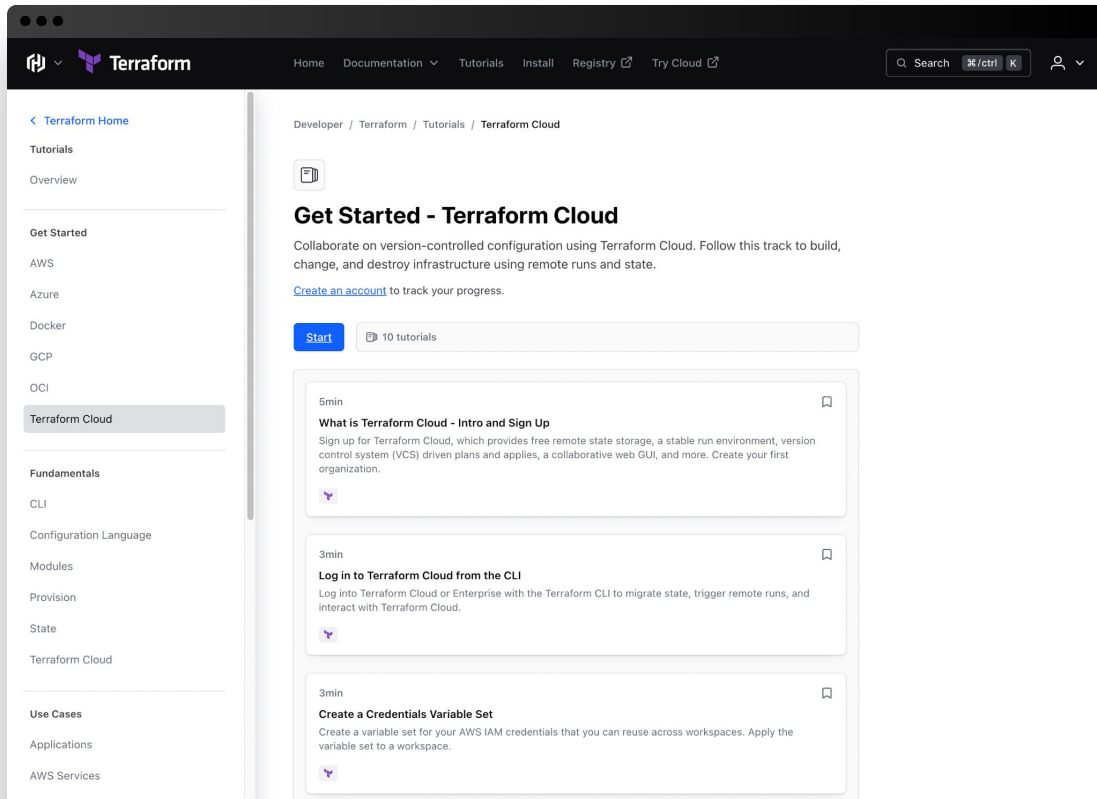
38 © HASHICORP

Next Steps

Tutorials

<https://developer.hashicorp.com/terraform/tutorials>

Step-by-step guides to accelerate deployment of Terraform Cloud



Additional Resources

- [HCL Reference](#)
- [Terraform Resource Graph](#)
- [Migrating Terraform OSS to Terraform Enterprise](#)
- [Importing Existing Infrastructure into Terraform Enterprise](#)
- [Terraform Import](#)
- [Config-driven Import](#)
- [Tutorial using Config-driven Import](#)
- Community Tools for importing resources*
 - [aws2tf](#)
 - [Terraformer](#)

* Community projects are **not maintained, supported or endorsed** by HashiCorp.

Need Additional Help?

Customer Success

Contact our Customer Success Management team with any questions. We will help coordinate the right resources for you to get your questions answered.

customer.success@hashicorp.com

Technical Support

Something not working quite right? Engage with HashiCorp Technical Support by opening a ticket for your issue at:

support.hashicorp.com.

Discuss

Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers

discuss.hashicorp.com



Upcoming Webinars



Terraform Workflow Management

Deep dive into best practices around run workflows, workspaces, variables, modules, and Git repo structure



Office Hours

Bring your questions to Office Hours!



Terraform Governance

Learn best practices and guidance for implementing key TFE features like Cloud Agents, RBAC, Sentinel, and run triggers and notifications

Action Items

- Share to customer.success@hashicorp.com
 - Authorized technical contacts for support
 - Stakeholders contact information (name and email addresses)
- Take an inventory of your current resources and begin your migration planning
- Select version control system(s)



Q&A





Thank you

customer.success@hashicorp.com

www.hashicorp.com/customer-success