



Lab: Terraform Cloud Workspaces

A Terraform workspace is a managed unit of infrastructure. Workspaces are the workhorse of Terraform Cloud and build on the Terraform CLI workspace construct. Each uses the same Terraform code to deploy infrastructure and each keeps separate state data for each workspace. Terraform Cloud simply adds more functionality. On your local workstation, the terraform workspace is simply a directory full of terraform code and variables. This code is also ideally stored in a git repository. Terraform Cloud workspaces take on some extra roles. In Terraform Cloud your workspace stores state data, has it's own set variable values and environment variables, and allows for remote operations and logging. Terraform Cloud workspaces also provide access controls, version control integration, API access and policy management.

This lab demonstrates how to utilize workspaces within Terraform Cloud.

- Task 1: Review Terraform Cloud Workspaces User Interface
- Task 2: Create a Terraform Cloud Workspace for DEV deployments
- Task 3: Create a Terraform Cloud Workspace for PROD developments
- Task 4: Assign and set variables per workspace
- Task 5: Change Terraform versions per workspace
- Task 6: Deploy infrastructure using Terraform Cloud workspaces

Task 1: Review Terraform Cloud Workspaces Features

Now that we have our state and variables stored in our Terraform Cloud workspace and have begun to execute remote runs within the workspace, let's look at some of the other features that Terraform Cloud workspaces have to offer.

1.1 Workspace Dashboard

The landing page for a workspace includes an overview of its current state and configuration.



HashiCorp Certified: Terraform Associate

Hands-On Labs



my-aws-app

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

[Overview](#) [Runs](#) [States](#) [Variables](#) [Settings](#) [⌵](#)

Resources: 30 Terraform version: 1.0.11 Updated: a few seconds ago

Unlocked [Actions](#) [⌵](#)

Execution mode: [Remote](#)

Auto apply: [Off](#)

Latest Run [View all runs](#)

Triggered via CLI [✔ Applied](#)

gabe_maentz triggered a run an hour ago via [CLI](#)

Policy checks: [✔ 2 of 2 passed](#) Estimated cost increase: [↑ \\$25.89](#) Plan & apply duration: [Less than a minute](#) Resources changed: [+0 ~1 -0](#) [See details](#)

Resources: [30](#) Outputs: [4](#) [Current as of the most recent state version.](#)

Filter resources [🔍](#)

NAME	PROVIDER	TYPE	MODULE	UPDATED ⌵
available	hashicorp/aws	data.aws_ava...	root	Dec 28 2021

Metrics (last 12 runs)

Average plan duration: [< 1 min](#)

Average apply duration: [2 mins](#)

Total failed runs: [3](#)

Policy check failures: [1](#)

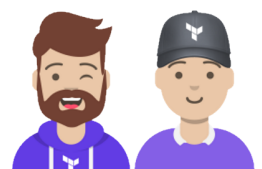
Tags (0)

[Add a tag](#) [⌵](#)

Tags have not been added to this workspace.

Figure 1: Terraform Cloud Workspace - Overview

You quickly find resource count, terraform version, and the time since the last update. You can view the latest information about last run including who performed the run, how the run was triggered, which resources changed, the duration of the run, along with the cost change. For further details you can jump into the he Runs tab to see all historical runs.



HashiCorp Certified: Terraform Associate Hands-On Labs



my-aws-app


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

Resources: 30 Terraform version: 1.0.11 Updated: an hour ago


Overview **Runs** States Variables Settings ▾


🔓 Unlocked **Actions** ▾

Current Run

 **Triggered via CLI** CURRENT ✓ Applied
#run-jNpYaBjyaQuhhca8 | gabe_maentz triggered via CLI
an hour ago

Run List

 **Triggered via CLI** CURRENT ✓ Applied
#run-jNpYaBjyaQuhhca8 | gabe_maentz triggered via CLI
an hour ago

 **Build with Sentinel Policy enforcement** ✓ Planned and finished
#run-VICHJjr7cENxztTB | gabe_maentz triggered via UI
an hour ago


 **Triggered via CLI** ✓ Applied
#run-tEdxuRFQRzSV4HSD | gabe_maentz triggered via CLI
3 hours ago

Figure 2: Terraform Cloud Workspace - Historical Runs

The **Actions** menu allows you to lock the workspace or trigger a new run. The resources and outputs tables display details about the current infrastructure and any outputs configured for the workspace. The sidebar contains metrics and details about workspace settings. It also includes a section for workspace tags. You can manage your workspace's tags here, and your list of tags in your Organization settings. Tags allow you to group and filter your workspaces.

1.2 Remote or Local Runs

As we have shown by using the `remote` backend, Terraform operations like plan or apply can now be run within Terraform Cloud on a hosted agent which saves the results and logs to the workspace. This gives you a standardized working environment for all Terraform runs and possibly more importantly, a single place to go and inspect logs when something goes wrong.





General Settings

ID

ws-3PyarudNZ5dY1h2R [🔗](#)

Name

my-aws-app

Description

Optional

Workspace description

Execution Mode

If you change the execution mode any in progress runs will be discarded.

☒ **Remote**

Your plans and applies occur on Terraform Cloud's infrastructure. You and your team have the ability to review and collaborate on runs within the app.

☐ **Local**

Your plans and applies occur on machines you control. Terraform Cloud is only used to store and synchronize state.

Figure 3: Terraform Cloud Workspace - Execution Mode

You can also choose between remote and local execution on a per workspace basis.

1.3 Terraform Version

When operating in Remote execution mode we can also specify the version of Terraform for the hosted agent to run without having to worry about the Terraform version installed on the local machine.





● **Manual apply**

Require an operator to confirm the result of the Terraform plan before applying. If this workspace is linked to version control, a push to the default branch of the linked repository will only trigger a plan and then wait for confirmation.

Terraform Version

1.0.11



The version of Terraform to use for this workspace. Upon creating this workspace, the latest version was selected and will be used until it is changed manually. It will **not upgrade automatically**.

Terraform Working Directory

The directory that Terraform will execute within. This defaults to the root of your repository and is typically set to a subdirectory matching the environment when multiple environments exist within the same repository.

Remote state sharing

Choose whether this workspace should share state with the entire organization, or only with specific approved workspaces. The `terraform_remote_state` data source relies on state sharing to access workspace outputs.

Figure 4: Terraform Cloud Workspace - Terraform Version

The Terraform core version can be specified on a per workspace basis and can differ between workspaces. Workspaces make it easy to standardize and upgrade when the time comes.

1.4 Team Access

Each workspace can be configured with team access permissions. The Team Access category allows you to define which teams have access to the workspace and what level of access they have. There are pre-canned levels of access, like read, plan, write, and admin, or you can construct your own custom permission sets.



HashiCorp Certified: Terraform Associate Hands-On Labs



my-aws-app

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

Resources
30

Terraform version
1.0.11

Updated
an hour ago

Overview Runs States Variables **Settings** ▾

Unlocked

Actions ▾

Team Access

Add team and permissions



Heads up

Teams with [organization-level permissions](#) can also access this workspace, even if they are not listed on this page or are listed at a lower access level.

NAME	PRIVILEGES
Owners of example-org-6cde13	default

Figure 5: Terraform Cloud Workspace - Team Access





Add Team Permissions

Add a team and assign permissions to this workspace.

1 Select a team

2 Assign permissions

Assign permissions to tfc_getting_started

Assign permissions to the selected team below.

☐ Customize permissions for this team

Read	Assign permissions	
Baseline permissions for reading a workspace		
<input checked="" type="checkbox"/> Read runs	<input checked="" type="checkbox"/> Read variables	<input checked="" type="checkbox"/> Read TF config versions
<input checked="" type="checkbox"/> Read workspace information	<input checked="" type="checkbox"/> Read state	

Plan	Assign permissions	
Read permissions plus the ability to create runs		
<input checked="" type="checkbox"/> All permissions of read	<input checked="" type="checkbox"/> Create runs	

Write	Assign permissions	
Read, plan and write permissions		
<input checked="" type="checkbox"/> All permissions of plan	<input checked="" type="checkbox"/> Can read and write	<input checked="" type="checkbox"/> Approve runs
<input checked="" type="checkbox"/> Lock/unlock workspace		

Admin	Assign permissions	
Full control of the workspace		
<input checked="" type="checkbox"/> All permissions of write	<input checked="" type="checkbox"/> Manage team access	<input checked="" type="checkbox"/> Delete workspace
<input checked="" type="checkbox"/> VCS configuration	<input checked="" type="checkbox"/> Execution mode	<input checked="" type="checkbox"/> Access to state

Figure 6: Terraform Cloud Workspace - Team Access

1.5 Notifications

Moving over to the notifications category, when something happens with the workspace, you can trigger a notification to be sent to an email address, slack channel, or webhook. This is great if you want to be notified when a new plan needs to be approved or an apply went horribly wrong.








Create a Notification

Notifications allow you to send messages to other applications based on Run events.

Destination


Webhook
POST messages to any URL


Email
Send messages to users via Email


Slack
Send messages to a Slack Channel

Name

e.g. My Notification

Webhook URL

https://example.com/...

Token

Encrypted - write only

Used to generate the HMAC on the notification request. [Read more in the documentation](#).

Triggers

Choose the events you want to receive notifications on.

☐ All events

☒ Only certain events

☒ Created

Every time a run is created and enters the "Pending" state.

☒ Planning

When a run acquires the lock and starts to execute.

☒ Needs Attention

Human decision required. When a plan has changes and is not auto-applied, or requires a policy override.

☒ Applying

After a plan is confirmed or auto-applied.

☒ Completed

When the run has completed on a happy path and can't go any further.

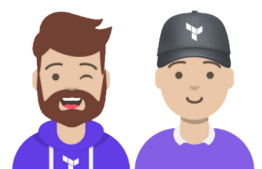
☒ Errored

If the run has terminated early due to error or cancelation.

Create a notification

Cancel

Figure 7: Terraform Cloud Workspace - Notifications





1.6 Workflows: CLI/VCS/API Integration

When you create a new workspace, Terraform Cloud will ask what type of workflow will be used by the workspace. You will be presented with three options, CLI, VCS, and API.


1 Choose Type

2 Connect to VCS

3 Choose a repository


4 Confirm changes

Choose your workflow

 **Version control workflow** Most common


Store your Terraform configuration in a git repository, and trigger runs based on pull requests and merges.

[Learn More](#)

 **CLI-driven workflow**

Trigger remote Terraform runs from your local command line.

[Learn More](#)

 **API-driven workflow**

A more advanced option. Integrate Terraform into a larger pipeline using the Terraform API.

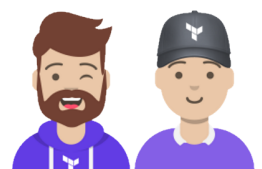
[Learn More](#)

Figure 8: Terraform Cloud Workspace - Notifications

We have been operating within the CLI workflow up to this point in the course, triggering all commands from the command line. You control the Terraform workflow from the CLI with standard Terraform commands like plan and apply.

The VCS or version control system workflow is the most common option, but it also requires that you host the Terraform code in a VCS repository. Events on the repository will trigger workflows on Terraform Cloud. For instance, a commit to the default branch could kick off a plan and apply workflow in Terraform Cloud.

If you need more customized automation and workflows than what is available in Terraform Cloud, you can use an API workflow to integrate Terraform Cloud into a larger automation pipeline. For instance, if you are already using Jenkins or CI/CD Pipelines to automate your IaC deployment, you can hook





Terraform Cloud in to handle the Terraform actions.

Task 2: Create a Terraform Cloud Workspace for DEV deployments

A workspace name has to be all lower case letters, numbers, and dashes. The recommended naming convention from HashiCorp is the team name, the cloud the infrastructure will be deployed in, the application or purpose of the infrastructure, and the environment, whether it's dev, staging, prod, etc. Workspace names are relative to the organization, so they do not have to be globally unique, only unique within the organization.

Suggested workspace naming convention —, e.g. devops-aws-myapp-dev which makes it easier to filter, navigate and performed automated operations against Terraform Cloud workspaces.

Let's create a new development workspace for your app called: `devops-aws-myapp-dev`

1. Select [New Workspace](#) and choose the [CLI-driven workflow](#)
2. Give the workspace a name of `devops-aws-myapp-dev`
3. Select [Create Workspace](#)

Task 3: Create a Terraform Cloud Workspace for PROD deployments

The state data in a workspace can also be shared with other workspaces in the organization as a data source. That's helpful if you decide to refactor your Terraform code into separate workspaces, but you need to pass information between them.

To isolate our production deployment from our development, let's create a new workspace for our production infrastructure named: `devops-aws-myapp-prod`

1. Select [New Workspace](#) and choose the [CLI-driven workflow](#)
2. Give the workspace a name of `devops-aws-myapp-prod`
3. Select [Create Workspace](#)

Task 4: Assign and set variables per workspace

3.1 - Assign Variables to DEV workspace

- Navigate to your Terraform Cloud `devops-aws-myapp-dev` workspace
- Once there, navigate to the [Variables](#) tab.





- In the [Variables](#) tab, you can add variables related to the state file that was previously migrated.
- To do this, first select the + [Add variable](#) button
- Let's add a Terraform variable named `aws_region` with a value of `us-east-1`
- Let's add a second Terraform variable named `vpc_name` with a value of `dev_vpc`
- Let's add a third Terraform variable named `environment` with a value of `dev`
- Choose the [Apply variable set](#) option to apply the `AWS Credentials` variable set

3.2 - Assign Variables to PROD workspace

- Navigate to your Terraform Cloud `devops-aws-myapp-prod` workspace
- Once there, navigate to the [Variables](#) tab.
- In the [Variables](#) tab, you can add variables related to the state file that was previously migrated.
- To do this, first select the + [Add variable](#) button
- Let's add a Terraform variable named `aws_region` with a value of `us-east-1`
- Let's add a second Terraform variable named `vpc_name` with a value of `prod_vpc`
- Let's add a third Terraform variable named `environment` with a value of `prod`
- Choose the [Apply variable set](#) option to apply the `AWS Credentials` variable set

Task 5: Change Terraform versions per workspace

Modify the Terraform version for the development workspace to use `1.1.2` to test if the deployments are compatible with that release and set production workspace at terraform version `1.0.11`

To set the Terraform version for a workspace navigate to [Settings](#) > [General](#) of the workspace and specify the appropriate Terraform core version.

Task 6: Deploy infrastructure using Terraform Cloud workspaces

6.1 - Declare Environment variables

The last step is to now deploy our infrastructure into the appropriate workspace. Now that our infrastructure will be broken up by environment, let's declare a variable in our `variables.tf` named `environment`

`variables.tf`

```
variable "environment" {
```





```
type = string
description = "Infrastructure environment. eg. dev, prod, etc"
default = "test"
}
```

Let's also update a couple of our resource blocks inside the `main.tf` to make use of this new environment variable.

`main.tf`

```
resource "aws_vpc" "vpc" {
  cidr_block = var.vpc_cidr

  tags = {
    Name           = var.vpc_name
    Environment    = var.environment
    Terraform      = "true"
  }

  enable_dns_hostnames = true
}

resource "aws_key_pair" "generated" {
  key_name     = "MyAWSKey${var.environment}"
  public_key   = tls_private_key.generated.public_key_openssh
}
```

6.2 - Deploy infrastructure using appropriate Terraform Cloud workspaces

Since we have Terraform Cloud workspaces that share the same Terraform codebase we will leverage backend partial configuration to select the correct workspace. If you are unfamiliar with Terraform backend partial configuration it is recommended you review the Terraform Backend Configuration lab which explains it in detail.

Create two new files in our working directory called `dev.hcl` and `prod.hcl`

`dev.hcl`

```
workspaces { name="devops-aws-myapp-dev" }
```

`prod.hcl`

```
workspaces { name="devops-aws-myapp-prod" }
```

Initialize and apply configuration in the development workspace.





```
terraform init -backend-config=dev.hcl -reconfigure
terraform plan
terraform apply
```

Initialize and apply configuration in the production workspace.

```
terraform init -backend-config=prod.hcl -reconfigure
terraform plan
terraform apply
```

You can view each of these deployments from either the CLI or Terraform Cloud interface. The infrastructure can be validated by looking at the Terraform Cloud workspace dashboard.

Workspaces 4 total [+ New workspace](#)

WORKSPACE NAME	RUN STATUS	REPO	LATEST CHANGE
devops-aws-myapp-dev	✓ Applied		2 minutes ago
devops-aws-myapp-prod	🔄 Applying		34 minutes ago
getting-started	✓ Applied		a month ago
my-aws-app	✓ Applied		3 hours ago

Figure 9: Terraform Cloud Workspace - Dashboard

6.3 - Destroy infrastructure in appropriate workspace

If you would like to cleanup and destroy your infrastructure to keep costs down, that can be done at the workspace level. Navigate to the [Settings](#) of the workspace and select [Destruction and Deletion](#). That will provide you with the ability to [Queue destroy plan](#) which follows the same workflow as a `terraform destroy`. Be sure not to delete the workspace itself (which is also an option on this same page) as in a future lab we will connect these new workspaces to the VCS workflow trigger runs based on code commits rather than via CLI.



HashiCorp Certified: Terraform Associate

Hands-On Labs



devops-aws-myapp-dev

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

Resources: 30 | Terraform version: 1.1.2 | Updated: 7 minutes ago

Overview | Runs | States | Variables | **Settings** ▾

Unlocked | Actions ▾

Destruction and Deletion

There are two independent steps for destroying this workspace and any infrastructure associated with it. First, any Terraform infrastructure should be destroyed. Second, the workspace in Terraform Cloud, including any variables, settings, and alert history can be deleted.

Destroy infrastructure

☒ **Allow destroy plans**
When enabled, this setting allows a destroy plan to be created and applied. This also applies when using the CLI.

Manually destroy

Queuing a destroy plan will redirect to a new plan that will destroy all of the infrastructure managed by Terraform. It is equivalent to running `terraform plan -destroy -out=destroy.tfplan` followed by `terraform apply destroy.tfplan` locally.

[Queue destroy plan](#) ←

Delete Workspace

Deleting a workspace will remove any variables, settings, alert history, run history, and state related to it. This **will not** remove any infrastructure managed by this workspace. If needed, destroy the infrastructure prior to deleting the workspace.

[Delete from Terraform Cloud](#)

Figure 10: Terraform Cloud Workspace - Dashboard

What belongs in a workspace?

Often times we are asked - What should I put in a workspace? We recommend infrastructure that should be managed together as a unit be placed into the same workspace. Who has to manage it, how often does it change, does it have external dependencies that we can't control. Ask these questions. Think about what happens when you run terraform apply. You should be able to describe what you just built, and what outputs it provides, who this infrastructure is for, and how to utilize it.

A workspace could be: An entire application stack from network on up. Great for dev environments that you want to be completely self-contained. Or it could be a workspace that builds core network infrastructure and nothing else. Maybe the network team has to manage that. They get to be the lords of the network and provide terraform outputs to those who need a VPC or subnet. You can also use workspaces to deploy platform services like Kubernetes.

