Don’t Be Beat

Use Your Terraform Cheatsheet

Like anything else, to get maximum outcomes it helps to follow best practices, and Terraform is no different. Managing states optimally won’t force you to waste time later on reviewing infrastructure pieces. Good examples of this include, managing state in a remote system, running Terraform from a centralized system where you can implement CI, maintain Terraform in a single code base, but separate the state into logical components through a structured hierarchy. Another golden rule is not to manage the entire system in a huge state, this causes very long-running timelines and makes it vulnerable to human error. If you need to touch specific code, it’s best to manage it from different states.

Initialize work directory

- `terraform init` Initialize directory, pull down providers.
- `terraform init -verify-plugins=false` Initialize directory, do not verify plugins for Hashicorp signature. This can be useful for custom plugins.

Plan

- `terraform plan-outplan.out` Output the deployment plan to plan.out.
- `terraform plan-target=aws_instance.my_ec2 ent` Only use when absolutely necessary and then test against entire resources and not any specific one.

Apply

- `terraform apply -auth-approve` Destroy/cleanup deployment without being prompted for “yes”. This applies to 90% of actions and is only used in automatic systems. When running from your own machine, try to avoid so you can preview changes before applying them.
- `terraform apply -var my_region_variable+us-east-1` Pass a variable via command-line while applying a configuration. Best practice is not to add var through the command line but to add a var file -var-file secret.tfvars. Allows you to keep track of what was changed and track the variables for deployment, as opposed to running a single command that may not have the history of what you’ve used previously.
- `terraform apply -parallelism=5` The number of simultaneous resource operations. Some resources are dependent on others, if you run resources in parallel things tend to break, so when you use parallelism make sure you can and that it won’t break anything.

Destroy

- `terraform destroy` Convenient way to destroy all remote objects. Command accepts most of the options that are accepted by **terraform apply.**
- `terraform plan -destroy` Create a speculative destroy plan to see what the effect of destroying would be. Showing you the changes without executing them.

Workspaces

- `terraform workspace new mynewworkspace` Used to create a new workspace. Know before you start what workspaces are and whether they suit your workflow needs.

State Manipulation

As a general rule, don’t manipulate anything regarding your state. Wherever you’re at in relation to your state, if you find yourself manipulating something this means that something wrong has happened and an underlying problem needs to be fixed.

- `terraform state show aws_instance.my_ec2` Show details stored in Terraform state for the resource.
- `terraform state pull > terraform.tfstate` To download and output Terraform state to a file. Make sure you have some sort of state system, don’t want to manage this on files.
- `terraform state mv aws_iam_role.my_ssm_role module.custom_module` To get into state option, move resources to a different module.
- `terraform state list` List all the resources tracked in the current state file.

Import and Outputs

- `terraform import aws_instance.new_ec2_instance i-abcd1234` Import EC2 instance with id-abcd1234 into the Terraform resource named “new_ec2_instance” of the type “aws_instance”.
- `terraform import aws_instance.new_ec2_instance[0]’i-abcd1234` As above, imports a real world resource into an instance of Terraform resource.
- `terraform apply - parallelism=5` The number of simultaneous resource operations. Some resources are dependent on others, if you run resources in parallel things tend to break, so when you use parallelism make sure you can and that it won’t break anything.
- `terraform output` List all outputs as stated in code.
- `terraform output instance_public_ip` List a specific declared output.
- `terraform output -jason` List all outputs in JSON format. At the end of every run can output some variables. This function lets you interact with outputs.

Taint/Untaint

- `terraform taint aws_instance.my_ec2` Taint resource to be recreated on next apply.
- `terraform untaint aws_instance.my_ec2` Remove taint from a resource.
- `terraform force-unlock LOCK-ID` Force-unlock a locked state file, LOCK_ID is provided when the state file is already locked.

Dependency Graphing

- `terraform graph | dot-Tpng > graph.png` Produces a PNG diagram showing relationship and dependencies between Terraform resources in your configuration/code. May be worth playing around with to get some additional insights.

Console

Tips to test out interpolations

- `echo ‘join(“,”,[“foo”,”bar”])’ | terraform console` Echo an expression into terraform console and see its expected result as output. This can help you to debug and make sure your code is comprehensible in very specific sections of language.

Miscellaneous commands

- `terraform providers` Get information about providers used in current configuration.
- `terraform version` Display Terraform binary version, also warns if a version is old.
- `terraform get -update+true` Download and update modules in the “root” module. Worth experimenting on.

Format and validate code

- `terraform fmt` Format code per HCL canonical standard.
- `terraform validate` Validate code for syntax.
- `terraform validate -backend=false` Validate code skip backend validation.