**Terraform cheat sheet**

**CLI commands**

* Initialize Terraform: terraform init
* Create a plan: terraform plan
* Apply the plan: terraform apply
* Destroy the infrastructure: terraform destroy
* To import an existing infrastructure: terraform import [resource\_name] [resource\_id]
* To see the state of the infrastructure: terraform show
* To list all the resources that Terraform is managing: terraform state list
* To get detailed information about a specific resource: terraform state show [resource\_name]
* To validate the syntax of the Terraform files: terraform validate
* To see a list of all the available Terraform providers: terraform providers
* To see the version of Terraform currently installed: terraform version
* To override a variable value from the command line: terraform apply -var '[variable\_name]=[value]

**TF file language**

* To create a new Terraform module: terraform module new [module\_name]
* To use a module in the Terraform configuration: module "[module\_name]" {}
* To pass variables to a module: module "[module\_name]" { [variable\_name] = [variable\_value] }
* To use a different version of a provider: provider "[provider\_name]" { version = "[version]" }
* To create and edit a Terraform variable file: terraform.tfvars
* To reference a variable in the Terraform configuration: var.[variable\_name]

**Worth noting**

* Before running the terraform apply command, check the plan using terraform plan.
* terraform destroy will delete all the resources defined in the current configuration.
* The terraform import command can only import resources that are already created and exist in the target provider.
* When importing, you’ll need to generate the config file of the resource you import.
* terraform state commands inspect the current state of the infrastructure managed by Terraform.
* Before initializing Terraform, remember to specify the provider and backend configuration in the main.tf file.
* The terraform validate command is used to validate the syntax of the Terraform files; it’s a good practice to run it before applying any change.
* terraform.tfvars is used to store sensitive data such as credentials, and it’s better not to include it in the version control.

**Advanced Terraform tips and tricks**

* To save the current state of the infrastructure to a file: terraform state pull > [file\_name].tfstate
* To load the state from a file: terraform state push [file\_name].tfstate
* To refresh the state and update it with the current infrastructure: terraform refresh
* To generate an execution plan and save it to a file: terraform plan -out [file\_name].plan
* To apply an execution plan from a file: terraform apply [file\_name].plan
* To see the current state: terraform show
* To return a detailed exit code to determine if changes exist between the current infrastructure and the current plan: terraform plan -detailed-exitcode
  + 0 = no changes; succeeded with empty diff
  + 1 = Error with plan
  + 2 = Succeeded, with changes to be made (non-empty diff)
* To generate a visual representation of either a configuration or execution plan: terraform graph | dot -Tpng > graph.png

**Terraform best practices**

* Keep the Terraform files in version control
* Use modules to organize the Terraform files
* Use variables to make the Terraform files more reusable
* Use terraform state commands to inspect and manage the state of the infrastructure
* Use Terraform workspaces to manage different environments (e.g. dev, staging, production)
* Use Terraform’s built-in validation checks, such as terraform validate, to catch errors early
* Use a separate backend configuration to store the state remotely to avoid loss and conflicts.
* Use terraform plan -detailed-exitcode to check for changes before applying.
* Use the terraform import command to import existing resources and keep track of them in the state.
* Use terraform taint to mark a specific resource for recreation.
  + This command is useful for operational resources that are experiencing internal problems not managed by terraform.
  + For example, a VM machine may be operational, but the software running on the instance has crashed
* While terraform state list is excellent for seeing all infrastructure that exists, the list will be large and will be difficult to parse out information
  + Ensure that you filter by resource or by module to get back helpful information
* Use terraform graph | dot -Tpng > graph.png to visualize configuration or execution plans

**Learn more about Terraform, infrastructure as code, and CI/CD**

In conclusion, this cheat sheet is a valuable tool for developers and DevOps professionals looking to take control of their infrastructure with Terraform. It covers basic tasks such as creating plans, applying them, and managing state, as well as advanced tips and tricks and best practices to help you make the most of this powerful tool.

However, this cheat sheet is not exhaustive, and there will always be more to learn about Terraform and infrastructure management. For more information, I recommend checking out the following resources:

* The [Terraform documentation](https://terraform-docs.io/)
* [The CI/CD pipeline: A developer’s guide](https://www.architect.io/blog/2022-10-24/cicd-pipeline-guide/)
* [Get started with the Terraform Kubernetes provider](https://www.architect.io/blog/2021-02-17/terraform-kubernetes-tutorial/)

These resources provide in-depth information and guidance on using Terraform and other infrastructure management tools and best practices for CI/CD. By following the tips and tricks outlined in this cheat sheet and diving deeper into these resources, you’ll be well on mastering Terraform and automating your infrastructure.