ChatGPT

One-Day Terraform Preparation Plan

Introduction

To succeed in the technical assessment for a Backend/Cloud developer position at Climatise, you need to become comfortable with the basics of **Terraform**—an Infrastructure-as-Code tool. The fundamentals include the configuration structure, working with providers, variables, resources and modules, and key CLI commands. This schedule is designed for roughly 8–10 hours of intensive study and practice and aims to give you enough confidence to handle theoretical questions like `--auto-approve`.

Study tips:

- Find a quiet spot and take short breaks each hour to stay fresh.
- Create a separate practice project (for example, `~/terraform-practice`) so you can immediately try everything you read about.
- Perform tasks by hand—active practice leads to better retention.
- All links to the official documentation are included for quick access and reinforcement.

Setup and Preparation (08:00–08:30)

1. **Install Terraform and prepare your environment**
2. Download the latest Terraform release from HashiCorp (via `apt`, `brew` or by downloading a binary).
3. Create a working directory, for example `~/terraform-practice`, and `cd` into it.
4. Understand that Terraform keeps a hidden `.terraform` directory for plugins and state. **Initialization (`terraform init`) is required before any operations**—it downloads providers and modules, creates or updates `.terraform` and is idempotent [1].
5. Run `terraform init` in an empty directory to see how the basic structure is created.
6. **Install AWS CLI (optional)**
   If you plan to experiment with AWS resources, install and configure AWS CLI (`aws configure`) using a test account.
   For local experiments you can use providers like `random` or `null_resource` so you don't spend money.

Terraform Basics (08:30–10:30)

1. Providers

- **Understand what a provider is:** Terraform uses provider plugins to interact with real cloud services or APIs. Providers define the available resource and data types [2]. Plugins are distributed separately and hosted on the Terraform Registry [3].

- **The `provider` block:** this declaratively configures how Terraform connects to a service, specifying region, credentials and so on. Provider configuration should only appear in the root module—child modules inherit the configuration [4].
- **Declaring required providers:** inside the top-level `terraform` block you add a `required_providers` block specifying the source and version of the plugin. Then you configure the provider in the root module [5].
- **Practice:** create a `main.tf` file with a `terraform { required_providers {...} }` block and a `provider "aws"` (or `random` / `azurerm`), then run `terraform init` again—you'll see Terraform downloading the plugins.

2. Resources

- **The `resource` block:** represents a piece of infrastructure (an EC2 instance, S3 bucket, IAM role, etc.). The block has a resource type and name, and contains provider-specific arguments [6]. Example:

```
resource "aws_s3_bucket" "example" {
  bucket = var.bucket_name
  acl    = "private"
}
```

* **Meta-arguments:** resources have universal meta-arguments such as `count` (to create multiple instances), `for_each` (to create resources from a list or map), `depends_on` (to explicitly set dependencies) and `lifecycle` (for example `create_before_destroy`, `prevent_destroy`) [7]. These control lifecycle and creation order.
*Practice:** create one or two resources (e.g., `random_pet` or `null_resource`) and experiment with `count` and `depends_on`.

3. Variables and Outputs

- **Input variables (`variable` block):** parameterize modules and configurations. Variables are declared in a module and values are passed via the CLI, variable files or environment variables. Arguments include `type` (data type), `default`, `description`, `validation` and `sensitive` [8]. A variable without a `default` is required.
- **Passing values:** values can be supplied via `terraform apply -var='region=us-east-1'`, a `terraform.tfvars` file, environment variables (`TF_VAR_region`) or in HCP Terraform.

- **Practice:** declare a variable `bucket_name` of type `string` with a description. Pass it via `-var` and via a `.tfvars` file.

- **The `output` block:** returns values after applying a configuration. Outputs are like return values; they appear in CLI output, HCP Terraform UI, and allow you to pass information between modules or external tools [9]. Example:

```
output "bucket_name" {
  description = "Name of the S3 bucket"
  value       = aws_s3_bucket.example.id
}
```

When calling a child module, output values expose attributes to the parent module, and root module outputs are shown after `apply` [10].
*Practice:** add an `output` for the resource name and view the result after `terraform apply`.

4. Modules

- **What is a module:** a collection of configuration files in a folder. The folder you run Terraform from is the **root module**. Within it you can call **child modules** using the `module` block [11]. Modules help you reuse code, standardize infrastructure and quickly provision repeating sets of resources [12].
- **Module sources:** local folders, Git repositories, S3 buckets, or Terraform Registry [13].
- **Practice:** move the S3 bucket definition into a separate folder as a module and call it from the root module:

```
module "s3_bucket" {
  source      = "./modules/s3_bucket"
  bucket_name = "my-terraform-bucket"
}
```

Terraform Commands (11:00–12:00)

Learn the primary CLI commands and experiment with them:

Command	Purpose and key options
`terraform init`	Initializes the directory: downloads providers, prepares `.terraform` and can be re-initialized; run before any operations [1].
`terraform plan`	Creates an **execution plan**: reads the current state, compares it with the configuration and proposes a set of actions [14]. This command does not change anything; it lets you verify the expected changes [15]. Options include `-out=FILE` (save the plan), `-destroy` (create a destroy plan), `-refresh-only` and others [16].
`terraform apply`	Executes a plan: creates/updates/destroys resources and updates the state [17]. If you run it without a plan file, it automatically creates a new plan and asks for confirmation. The `-auto-approve` option skips the confirmation prompt—useful for automation, but you must be confident that no one else is modifying the infrastructure [18].
`terraform destroy`	Removes all objects managed by the configuration. It is an alias for `terraform apply -destroy` [19]. Use this to clean up temporary infrastructure.
`terraform fmt`	Formats `.tf` files to a consistent style.
`terraform validate`	Checks the syntax and consistency of the configuration to catch errors early.

Practice:

1. Create a simple resource (e.g., `random_pet`), then:
2. Run `terraform plan` and study the output: Terraform shows what will be created, changed or destroyed.
3. Save the plan to a file (`-out=plan.out`), then apply it using `terraform apply plan.out`.
4. Try `terraform apply -auto-approve` to see that Terraform skips confirmation.
5. Run `terraform destroy` to remove resources; observe how `plan -destroy` creates a pre-destruction plan [20] .

Working with Strings, Variables and Conditionals (12:00–13:00)

Questions may involve HCL syntax, so:

- Practice declaring **maps and lists** and access elements (`var.list[0]`, `var.map["key"]`).
- Practice **conditional expressions** and functions (`coalesce`, `join`, `length`).
- Understand **local values** (`locals { ... }`), which let you define helper variables within a module.

For reinforcement, follow the official guides: Customize Terraform Configuration with Variables and Output Data From Terraform.

Lunch Break (13:00–14:00)

Take a proper break—have lunch or go for a walk.

Modules and a Bit about State (14:00–15:30)

1. **Create your own modules.** Move configuration from the root module into a separate folder and call it via `module`. Practice passing variables into a module and reading output values. Use module versioning (e.g., Git tags).
2. **Learn about state:** Terraform stores state in a `terraform.tfstate` file. The state tracks what has been created and helps Terraform correctly plan changes [17] . Understand that manual changes to infrastructure can lead to drift.
3. **Intro to backends:** a local backend stores state on disk; a remote backend (e.g., S3 + DynamoDB, or HCP Terraform) stores state remotely, allowing team collaboration.

Review Commands and Prep for Test Questions (15:30–16:30)

- Create a **cheat sheet** of commands and flags. Memorize the `--auto-approve` flag: it tells Terraform to apply a plan without asking for confirmation [18] . Review flags like `-destroy`, `-refresh-only`, `-out`.
- Familiarize yourself with other useful `plan` and `apply` options, like `-replace` (replace a resource) and `-target` (focus on a specific resource) [21] , though they are less likely to be needed at the basic level.
- Take some online Terraform quizzes (search for *Terraform CLI quiz*). This helps you get used to the question format.
- Write a couple of small configurations for different scenarios: creating an S3 bucket, an EC2 instance, a Lambda function (if you know AWS) or local resources.

- If time allows, go through the official Terraform Getting Started tutorial—it explains each step and includes hands-on examples.

Final Review and Notes (16:30–17:00)

- Summarize what commands you used and which concepts were challenging.
- Re-read the job requirements: **AWS (Lambda, RDS, DynamoDB, S3, SNS/SQS), Terraform, GraphQL** and consider which resources you, as a backend developer, will need to describe.
- Remember that the assessment also contains **SQL** and **Python algorithm** questions (e.g., the peak-counting problem). Plan to spend the evening practising SQL and algorithm problems so you can confidently handle all sections.

Additional Resources

- **Official Terraform documentation:**
- CLI initialization and working directories [1].
- The `plan` command and its modes [22].
- `apply` and the `--auto-approve` flag [18].
- `destroy` as an alias for `apply -destroy` [19].
- The `provider` block [23], `required_providers` [5], `variable` [8], `output` [9] and `resource` [24] blocks.
- Overview of modules [11].
- **Practical courses and videos:** watch short videos (e.g., “Terraform in 30 minutes”) to visualise the process.
- **Cheat sheet:** download a handy Terraform CLI cheat sheet (for example from spacelift.io or env0) and keep it nearby.

Conclusion

By following this plan you will cover the key concepts of Terraform in one day—from project initialization to modules and major CLI commands. Building your own small project will reinforce what you learn and help you confidently answer questions like `terraform apply --auto-approve` during the assessment. Good luck with your preparation!

---

[1] Initialize the Terraform working directory | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/cli/init

[2] Providers - Configuration Language | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/providers

[3] [4] [23] Provider block reference for the Terraform configuration language | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/block/provider

[5] Provider Requirements - Configuration Language | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/providers/requirements

[6] Configure a resource | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/resources/configure

[7] [24] resource block reference | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/block/resource

[8] variable block reference for the Terraform configuration language | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/block/variable

[9] Output block reference for the Terraform configuration language | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/block/output

[10] Use outputs to expose module data | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/values/outputs

[11] [12] [13] Modules overview | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/modules

[14] [15] [16] [21] [22] terraform plan command reference | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/cli/commands/plan

[17] Create and manage resources overview | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/language/resources

[18] terraform apply command reference | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/cli/commands/apply

[19] [20] terraform destroy command reference | Terraform | HashiCorp Developer
https://developer.hashicorp.com/terraform/cli/commands/destroy