Troubleshoot Terraform

14 MIN | PRODUCTS USED Terraform

This tutorial also appears in: Associate Tutorials and State.

The primary method for interacting with Terraform is the HashiCorp Configuration Language (HCL). When Terraform encounters an error in your configuration, it will report an error including line numbers and the type of issue found in the configuration.

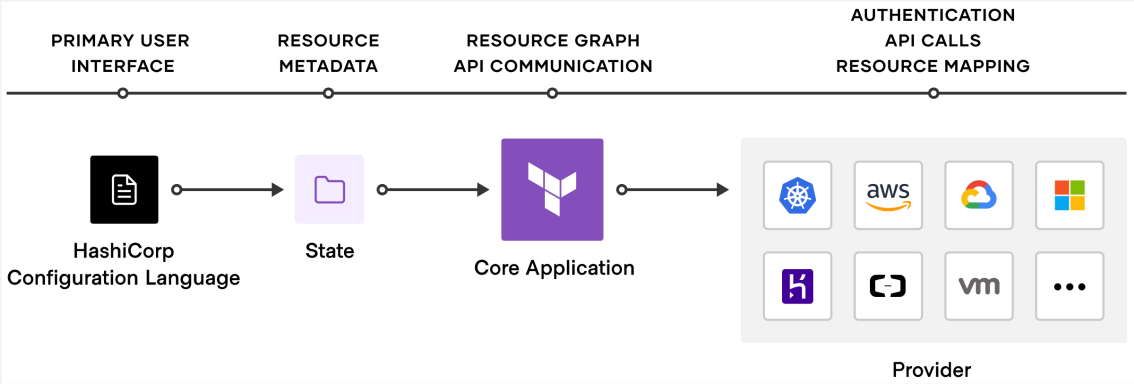
In this tutorial, you will clone a repository with a broken Terraform configuration to deploy an EC2 instance and underlying networking. This configuration contains intentional errors to introduce you to troubleshooting the configuration language.

Prerequisites

For this tutorial, you will need:

- Terraform 0.15.2+ installed locally
- an AWS account with credentials configured for Terraform

Review the Terraform troubleshooting model



There are four potential types of issues that you could experience with Terraform: language, state, core, and provider errors. Starting from the type of error closest to the user:

1. **Language errors**: The primary interface for Terraform is the HashiCorp Configuration Language (HCL), a declarative configuration language. The Terraform core application interprets the configuration language. When Terraform encounters a syntax error in your configuration, it prints out the line numbers and an explanation of the error.

2. **State errors**: The Terraform state file stores information on provisioned resources. It maps resources to your configuration and tracks all associated metadata. If state is out of sync, Terraform may destroy or change your existing resources. After you rule out configuration errors, review your state. Ensure your configuration is in sync by refreshing, importing, or replacing resources.

3. **Core errors**: The Terraform core application contains all the logic for operations. It interprets your configuration, manages your state file, constructs the resource dependency graph, and communicates with provider plugins. Errors produced at this level may be a bug. Later in this tutorial, you will learn best practices for opening a GitHub issue for the core development team.

4. **Provider errors**: The provider plugins handle authentication, API calls, and mapping resources to services. Later in this tutorial, you will learn best practices for opening a GitHub issue for the provider development team.

Clone the GitHub example repository

Clone the example GitHub repository.

```
$ git clone https://github.com/hashicorp/learn-terraform
```

Copy

Change into the repository directory.

```
$ cd learn-terraform-troubleshooting
```

Copy

Open the `main.tf` file in your file editor.

This configuration includes intentional problems and you may find some issues immediately, but do not edit your configuration yet. In the next section, you will run the format command to identify errors.

Format the configuration

The format command scans the current directory for configuration files and rewrites your Terraform configuration files to the recommended format.

In your terminal, run the `terraform fmt` command. This command returns two errors informing you of an invalid character and an invalid expression. Both errors occur on line 46.

```
$ terraform fmt
Error: Invalid character

  on main.tf line 46, in resource "aws_instance" "web_app":
 46:     Name = $var.name-learn

This character is not used within the language.
```

Copy

```
Error: Invalid expression

  on main.tf line 46, in resource "aws_instance" "web_app":
  46:     Name = $var.name-learn

Expected the start of an expression, but found an invalid expressic
```

Terraform found problems that it could not parse, and output errors so you can fix the configuration manually.

Correct a variable interpolation error

In `main.tf`, find the `aws_instance.web_app` resource's `tags` attribute on line 46. The `Name` tag is incorrectly formatted. It is trying to append a string to the `name` input variable without the necessary interpolation syntax. Update the `Name` attribute with the correct syntax.

```
##...
   tags = {
-     Name = $var.name-learn
+     Name = "${var.name}-learn"
   }
 }
##...
```

Run `terraform fmt` again to ensure your variable name meets the formatting requirements.

```
$ terraform fmt
```

Copy

You resolved the invalid character and expression errors, and they don't appear again. Terraform parses `"${var.name}-learn"` as your variable name in the interpolation shorthand with the hardcoded `-learn` string appended to form a custom value.

Validate your configuration

`terraform fmt` only parses your HCL for interpolation errors or malformed resource definitions, which is why you should use `terraform validate` after formatting your configuration to check your configuration in the context of the providers' expectations.

Initialize your Terraform directory to download the providers that your configuration requires.

```
$ terraform init
```

Copy

Run `terraform validate` in your terminal. The output contains a cycle error that highlights a mutual dependency between two security group resources.

```
$ terraform validate
```

Copy

```
Error: Cycle: aws_security_group.sg_ping, aws_security_group.sg_80
```

Cycle errors are instances of circular logic in the Terraform dependency tree. Terraform analyzes the dependencies between resources in your infrastructure configuration to determine the order to perform your operations.

In the next section, you will correct this dependency graph error.

Correct a cycle error

Your `aws_security_group` resources reference one another in their `security_groups` attributes. AWS cannot create the security groups because their configurations each reference the other group, which would not exist yet.

Remove the mutually dependent security group rules in your configuration, leaving the two group resources without ingress attributes.

```
 resource "aws_security_group" "sg_ping" {
   name = "Allow Ping"

-  ingress {
-    from_port       = -1
-    to_port         = -1
-    protocol        = "icmp"
-    security_groups = [aws_security_group.sg_8080.id]
-  }
 }

 resource "aws_security_group" "sg_8080" {
   name = "Allow 8080"

-  ingress {
-    from_port       = 8080
-    to_port         = 8080
-    protocol        = "tcp"
-    security_groups = [aws_security_group.sg_ping.id]
-  }
 }
```

Instead of including the rules in the `aws_security_group` configuration, use the `aws_security_group_rule` resource and reference the security group IDs instead. This avoids a cycle error because the provider will have AWS create both of the `aws_security_group` resources first, without interdependent rules. It will create the rules next, and attach the rules to the groups last.

Add the new, independent rule resource configurations to `main.tf` .

```
resource "aws_security_group_rule" "allow_ping" {
    type = "ingress"
    from_port = -1
```

Copy

```
    to_port = -1
    protocol = "icmp"
    security_group_id = aws_security_group.sg_ping.id
    source_security_group_id = aws_security_group.sg_8080.id
}

resource "aws_security_group_rule" "allow_8080" {
    type = "ingress"
    from_port = 80
    to_port = 80
    protocol = "tcp"
    security_group_id = aws_security_group.sg_8080.id
    source_security_group_id = aws_security_group.sg_ping.id
}
```

Terraform does not continue validating once it catches an error. Run the `terraform validate` command to catch new errors: an invalid reference a the `for_each` attribute because of a splat expression (`*`) in its value.

```
$ terraform validate                                                    Copy
Error: Invalid reference

  on main.tf line 44, in resource "aws_instance" "web_app":
  44:   for_each              = aws_security_group.*.id


A reference to a resource type must be followed by at least one att
access, specifying the resource name.


Error: Invalid "each" attribute

  on main.tf line 47, in resource "aws_instance" "web_app":
  47:   vpc_security_group_ids = [each.id]


The "each" object does not have an attribute named "id". The suppor
attributes are each.key and each.value, the current key and value p
```

```
"for_each" attribute set.
```

The `each` attribute in the `vpc_security_group_ids` cannot return the IDs because of the `for_each` error above it. Terraform did not return any security group IDs, so the `each` object is invalid.

Review the construction of the instance resource.

```
resource "aws_instance" "web_app" {
  for_each               = aws_security_group.*.id
  ami                    = data.aws_ami.ubuntu.id
  instance_type          = "t2.micro"
  vpc_security_group_ids = [each.id]
  user_data              = <<-EOF
              #!/bin/bash
              echo "Hello, World" > index.html
              nohup busybox httpd -f -p 8080 &
              EOF
  tags = {
    Name = "${var.name}-learn"
  }
}
```

Terraform cannot automatically convert types without additional functions.

In the next section, you will correct this expression and `for_each` error.

Correct a `for_each` error

Terraform's `for_each` attribute allows you to create a set of similar resources based on the criteria you define.

In this example, you need to create a set of similar instances, each assigned to a different security group. Terraform cannot parse `aws_security_group.*.id` in this attribute because the splat expression

`(*)` only interpolates list types, while the `for_each` attribute is reserved for map types. A local value can return a map type.

In `main.tf`, on line 44, replace the value of the `for_each` attribute with a local value. On line 47, replace the `vpc_security_group_ids` value with the value from the `for_each` attribute. Finally, update the `tags` attribute to give each instance a unique name.

```
resource "aws_instance" "web_app" {
-  for_each               = aws_security_group.*.id
+  for_each               = local.security_groups
   ami                    = data.aws_ami.ubuntu.id
   instance_type          = "t2.micro"
-  vpc_security_group_ids = [each.id]
+  vpc_security_group_ids = [each.value]
   user_data              = <<-EOF
              #!/bin/bash
              echo "Hello, World" > index.html
              nohup busybox httpd -f -p 8080 &
              EOF
   tags = {
-    Name = "${var.name}-learn"
+    Name = "${var.name}-learn-${each.key}"
   }
 }
}
```

Define the local value in your `main.tf` file. This converts the list of security groups to a map.

```
locals {                                                    Copy
  security_groups = {
    sg_ping   = aws_security_group.sg_ping.id,
    sg_8080   = aws_security_group.sg_8080.id,
  }
}
```

Format your configuration.

```
$ terraform fmt
main.tf
```

Copy

Your next `terraform validate` operation will produce errors output errors because of the `for_each` value you corrected. Your outputs do not capture the multiple instances in the `aws_instance.web_app` resources.

Validate your configuration to return the output errors.

```
$ terraform validate
```

Copy

```
Error: Missing resource instance key

  on outputs.tf line 3, in output "instance_id":
   3:   value       = aws_instance.web_app.id

Because aws_instance.web_app has "for_each" set, its attributes mu
accessed on specific instances.

For example, to correlate with indices of a referring resource, us
    aws_instance.web_app[each.key]


Error: Missing resource instance key

  on outputs.tf line 8, in output "instance_public_ip":
   8:   value       = aws_instance.web_app.public_ip

Because aws_instance.web_app has "for_each" set, its attributes mu
accessed on specific instances.

For example, to correlate with indices of a referring resource, us
    aws_instance.web_app[each.key]


Error: Missing resource instance key
```

```
  on outputs.tf line 13, in output "instance_name":
  13:   value       = aws_instance.web_app.tags

Because aws_instance.web_app has "for_each" set, its attributes mu
accessed on specific instances.

For example, to correlate with indices of a referring resource, us
    aws_instance.web_app[each.key]
```

In the next section, you will correct these errors by implementing a `for` expression to define outputs with lists of your instance IDs, IP addresses, and names.

Correct your outputs to return all values

To correct your outputs, you need the `for` expression to capture the elements of the multiple resources.

The `for` expression captures all of the elements of `aws_instance.web_app` in a temporary variable called `instance`. Then, Terraform returns all of the specified values of the `instance` elements. In this example, `instance.id`, `instance.public_ip`, and `instance.tags.Name` return every matching key value for each instance you created.

Open `outputs.tf` and update the output values with the `for` expression.

```
output "instance_id" {
  description = "ID of the EC2 instance"
-   value       = aws_instance.web_app.id
+   value       = [for instance in aws_instance.web_app: instance.i
 }

output "instance_public_ip" {
```

```
   description = "Public IP address of the EC2 instance"
-   value       = aws_instance.web_app.public_ip
+   value       = [for instance in aws_instance.web_app: instance.p
 }

output "instance_name" {
   description = "Tags of the EC2 instance"
-  value       = aws_instance.web_app.tags
+  value       = [for instance in aws_instance.web_app: instance.t
}
```

Format your configuration.

```
$ terraform fmt
outputs.tf
```

Copy

Validate your configuration

```
$ terraform validate
Success! The configuration is valid.
```

Copy

Apply your changes

Now that you have corrected the Terraform configuration, run `terraform apply` to create your resources. Enter `yes` when prompted to confirm your changes.

```
$ terraform apply
## ...

aws_security_group.sg_ping: Creating...
aws_security_group.sg_8080: Creating...
aws_security_group.sg_8080: Creation complete after 2s [id=sg-0873a
aws_security_group.sg_ping: Creation complete after 2s [id=sg-031ca
aws_security_group_rule.allow_8080: Creating...
```

Copy

```
aws_security_group_rule.allow_ping: Creating...
aws_instance.web_app["sg_ping"]: Creating...
aws_instance.web_app["sg_8080"]: Creating...
aws_security_group_rule.allow_ping: Creation complete after 6s [id=
aws_security_group_rule.allow_8080: Creation complete after 6s [id=
aws_instance.web_app["sg_ping"]: Still creating... [10s elapsed]
aws_instance.web_app["sg_8080"]: Still creating... [10s elapsed]
aws_instance.web_app["sg_8080"]: Still creating... [20s elapsed]
aws_instance.web_app["sg_ping"]: Still creating... [20s elapsed]
aws_instance.web_app["sg_ping"]: Creation complete after 24s [id=i-
aws_instance.web_app["sg_8080"]: Creation complete after 24s [id=i-

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

Outputs:

instance_id = [
  "i-0a679d5acef2b8e01",
  "i-02bd8b86bfa93ae96",
]
instance_name = [
  "terraform-learn-sg_8080",
  "terraform-learn-sg_ping",
]
instance_public_ip = [
  "13.58.116.31",
  "3.131.82.166",
]
```

Bug reporting best practices

You may experience errors due to provider or application issues. Once you eliminate the possibility of language misconfiguration, version mismatching, or state discrepancies, consider bringing your issue to the core Terraform team or Terraform provider community as a bug report.

To provide the development team or the community working on your issue with full context, here are some best practices for opening up a

GitHub issue.

Confirm versioning

Confirm the versions of the providers you are using and the version of Terraform you have in your environment. To confirm your provider and Terraform versions, run the version command.

```
$ terraform version
Terraform v0.15-beta2
+ provider registry.terraform.io/hashicorp/aws v3.33.0
```

Copy

You can also validate that you are using the correct and most recent versions of your providers before reporting a bug. If your lock file specifies an older version, consider updating your providers and attempting to run your operation again.

Enable Terraform logging

Terraform 0.15 allows you to generate logs from the Terraform provider and the core application separately. The Terraform development team needs the core logs for your attempted operation to troubleshoot core-related errors. To enable core logging, set the `TF_LOG_CORE` environment variable to the appropriate log level. For bug reports, you should use the `TRACE` level.

```
$ export TF_LOG_CORE=TRACE
```

Copy

`TRACE` provides the highest level of logging and contains all the information the development teams need. There are other logging levels, but are typically reserved for developers looking for specific information.

You can also generate provider logs by setting the `TF_LOG_PROVIDER` environment variable. By including these in your bug reports, the provider development team can reproduce and debug provider specific errors.

```
$ export TF_LOG_PROVIDER=TRACE
```

Copy

Once you have configured your logging, set the path for your error logs as an environment variable. If your `TF_LOG_CORE` or `TF_LOG_PROVIDER` environment variables are enabled, the `TF_LOG_PATH` variable will create the specified file and append logs generated by Terraform.

```
$ export TF_LOG_PATH=logs.txt
```

Copy

To generate an example of the core and provider logs, run a `terraform refresh` operation.

```
$ terraform refresh
```

Copy

Open and review the `logs.txt` file. This log file contains both `provider.terraform-provider-aws` and Terraform core logging.

```
$ cat logs.txt
##...
2021-04-02T15:36:08.477-0400 [DEBUG] provider.terraform-provider-aw
2021-04-02T15:36:08.477-0400 [DEBUG] provider: using plugin: versic
2021-04-02T15:36:08.525-0400 [TRACE] provider.stdio: waiting for st
2021-04-02T15:36:08.525-0400 [TRACE] GRPCProvider: GetProviderSchem
2021-04-02T15:36:08.593-0400 [TRACE] GRPCProvider: Close
2021-04-02T15:36:08.596-0400 [DEBUG] provider.stdio: received EOF,
2021-04-02T15:36:08.598-0400 [DEBUG] provider: plugin process exite
2021-04-02T15:36:08.598-0400 [DEBUG] provider: plugin exited
2021-04-02T15:36:08.598-0400 [TRACE] terraform.NewContext: complete
2021-04-02T15:36:08.599-0400 [TRACE] backend/local: finished buildi
2021-04-02T15:36:08.599-0400 [TRACE] backend/local: requesting inte
```

Copy

```
2021-04-02T15:36:08.599-0400 [TRACE] Context.Input: Prompting for p
2021-04-02T15:36:08.599-0400 [TRACE] Context.Input: Provider provi
2021-04-02T15:36:08.600-0400 [TRACE] Context.Input: Input for provi
2021-04-02T15:36:08.600-0400 [TRACE] backend/local: running validat
2021-04-02T15:36:08.600-0400 [INFO]  terraform: building graph: Gra
2021-04-02T15:36:08.600-0400 [TRACE] Executing graph transform *ter
##...
```

To remove a log stream, unset the environment variable you do not need. Unset Terraform core logging. When you re-run Terraform, Terraform will only log provider specific operations. When you close your terminal session, all environment variables unset.

```
export TF_LOG_CORE=
```

Copy

Open a ticket

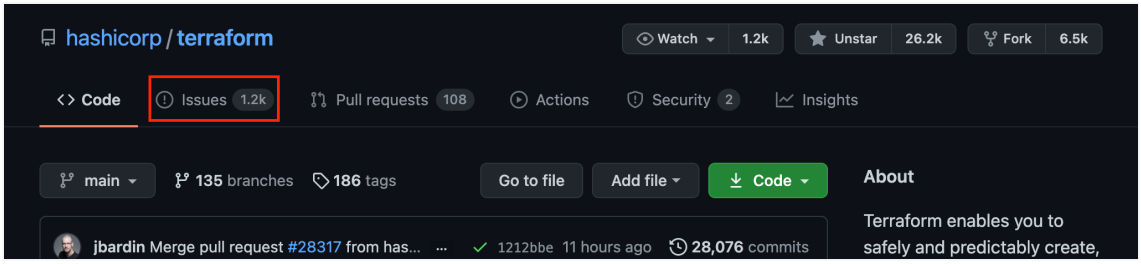
If you would like input from the community before submitting your issue to the repository, consider submitting your issue as a forum topic in the HashiCorp Discuss forum.

To create a bug review issue, you must determine which log stream contains your error. In your `logs.txt` file, find the final error message and trace it back to the source. It should contain `provider-terraform-<PROVIDER-NAME>` if it is a provider issue.

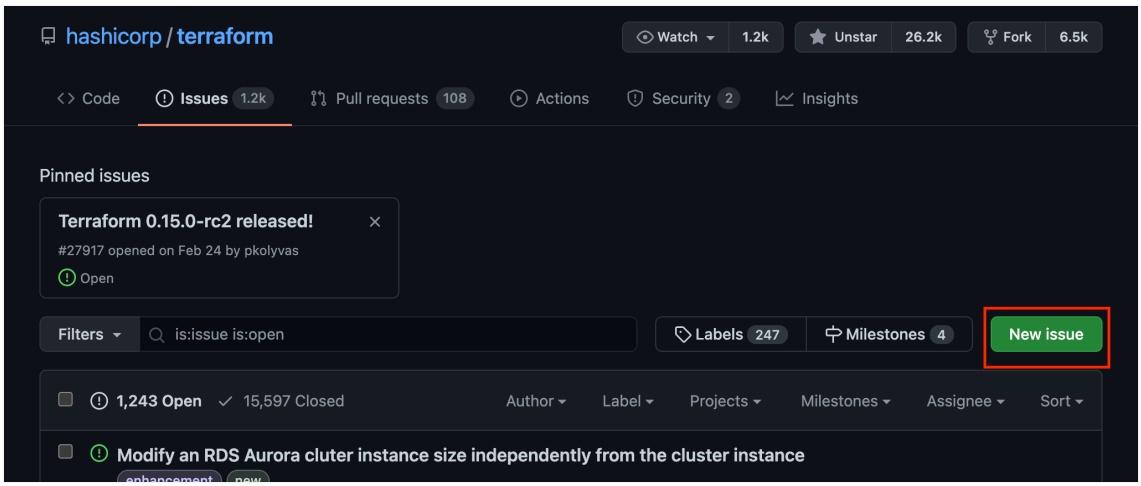
When you determine where your error originated, navigate to the Terraform core GitHub repository or search the provider registry for your provider's GitHub repository.

Some providers may have different suggestions for opening issues, but the Terraform core repository has a ticket template you should follow to provide the team with the information they need.

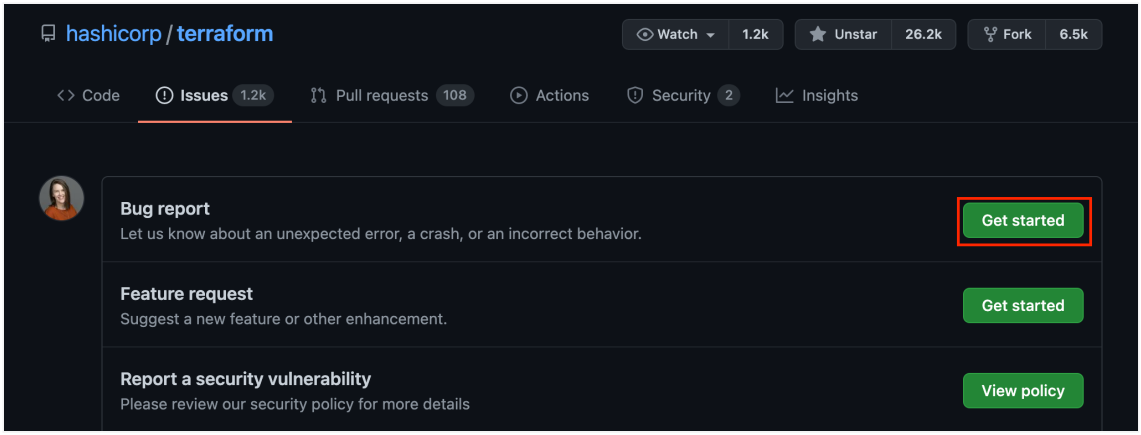
First, navigate to the Terraform GitHub repository and choose "Issues" from the top tabs.



Choose "New Issue".



Select "Get started" with a bug report.



Familiarize yourself with the code of conduct.

Using the Terraform core template, fill in the information you collected and note the expected behavior. When you finish filling out the template, select "Submit New Issue," and the team will review your issue.

Clean up resources

Destroy the resources you created. Respond `yes` to the prompt to confirm.

```
$ terraform destroy
```

Copy

Next steps

In this tutorial, you learned how to troubleshoot Terraform by correcting broken configuration for an EC2 instance and security groups. You corrected a cycle error, a variable interpolation error, and a looping error by formatting and validating your configuration. You also learned how to enable logging, and the best practices for reporting issues to the Terraform and provider teams on GitHub.

For more information on state and recommended practices, review the following tutorials:

- Learn Terraform Import
- Manage resources with the Terraform CLI
- Debugging on the Terraform Docs site

Was this tutorial helpful?

Yes No

BACK TO COLLECTION

stdin: is not a tty