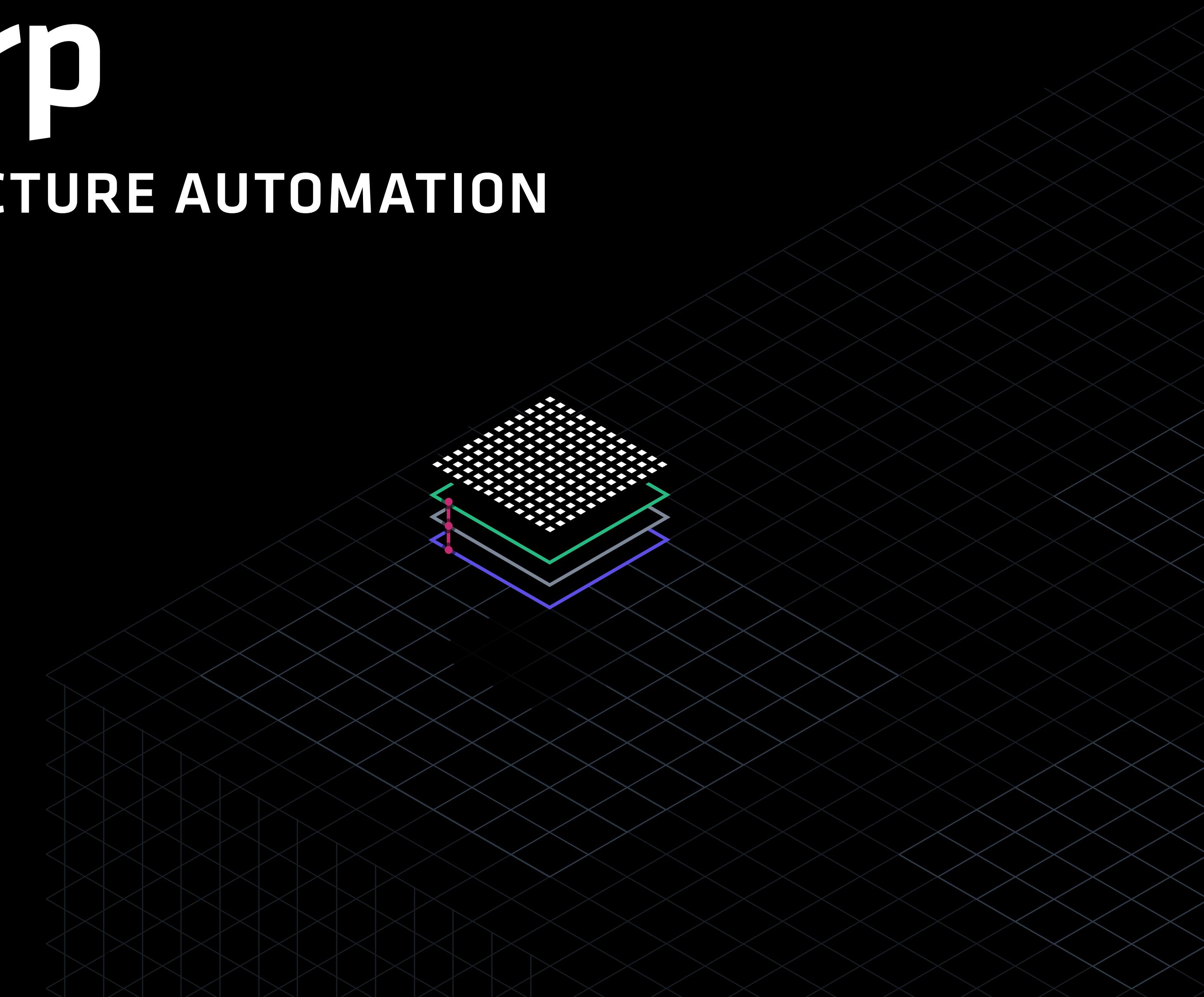




HashiCorp

CLOUD INFRASTRUCTURE AUTOMATION





HashiCorp
Terraform

Agenda

Evolution of Infrastructure

Basic Configuration

Outputs & Variables

Configuration Format and
Auto-Formatting

Modules, Provisioners,
and Providers

Graph Theory

Meta Parameters

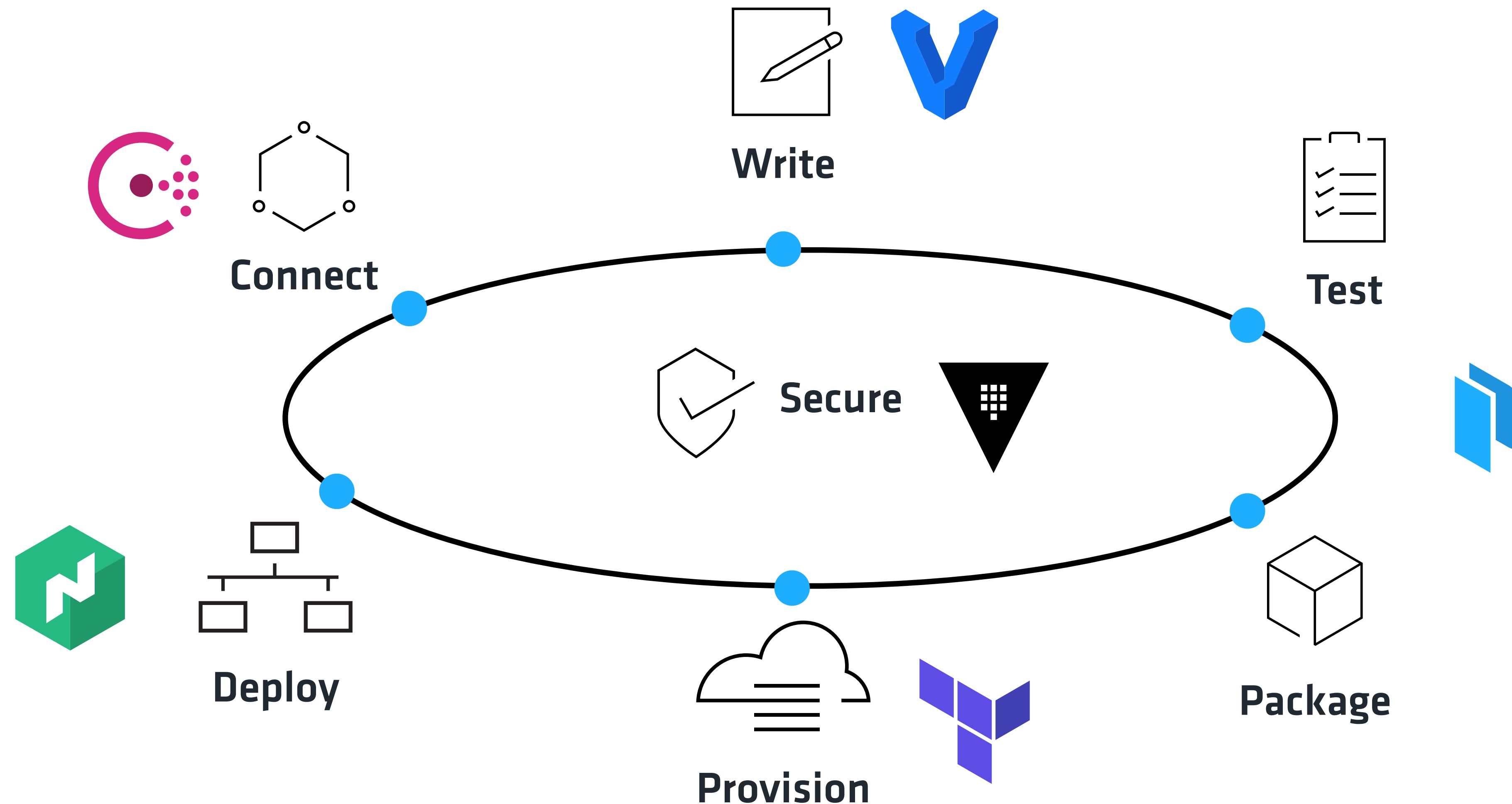
Terraform Enterprise:
Remote State

Terraform Enterprise:
GitHub Deploys

Destroy

Introduction

Application Delivery Lifecycle



Terraform's Goals

Unify the view of resources using infrastructure as code

Support the modern data center (IaaS, PaaS, SaaS)

Expose a way for individuals and teams to safely and predictably change infrastructure

Provide a workflow that is technology agnostic

Manage anything with an API

Terraform vs. Other Tools

Provides a high-level abstraction of infrastructure (IaC)

Allows for composition and combination

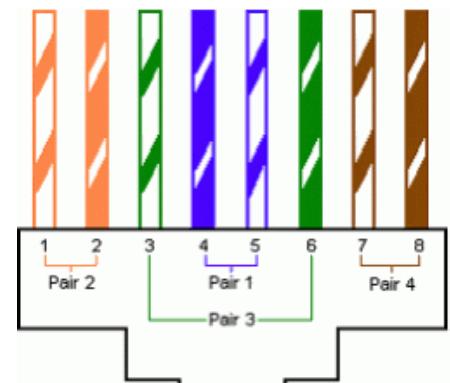
Supports parallel management of resources (graph, fast)

Separates planning from execution (dry-run)

Progression of Infrastructure Management



Maturity



Manual Everything

```
#!/bin/bash
```



Basic Automation



Machine Virtualization



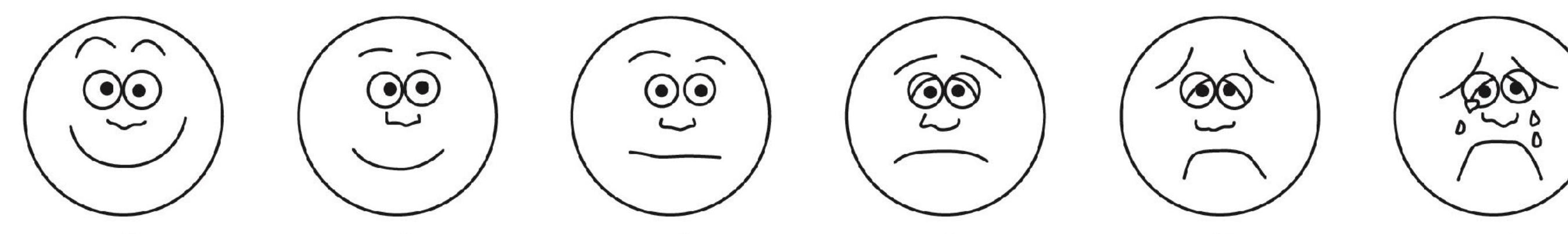
*aaS



Datacenter-as-Computer



Maturity



No
Pain

A Little
Pain

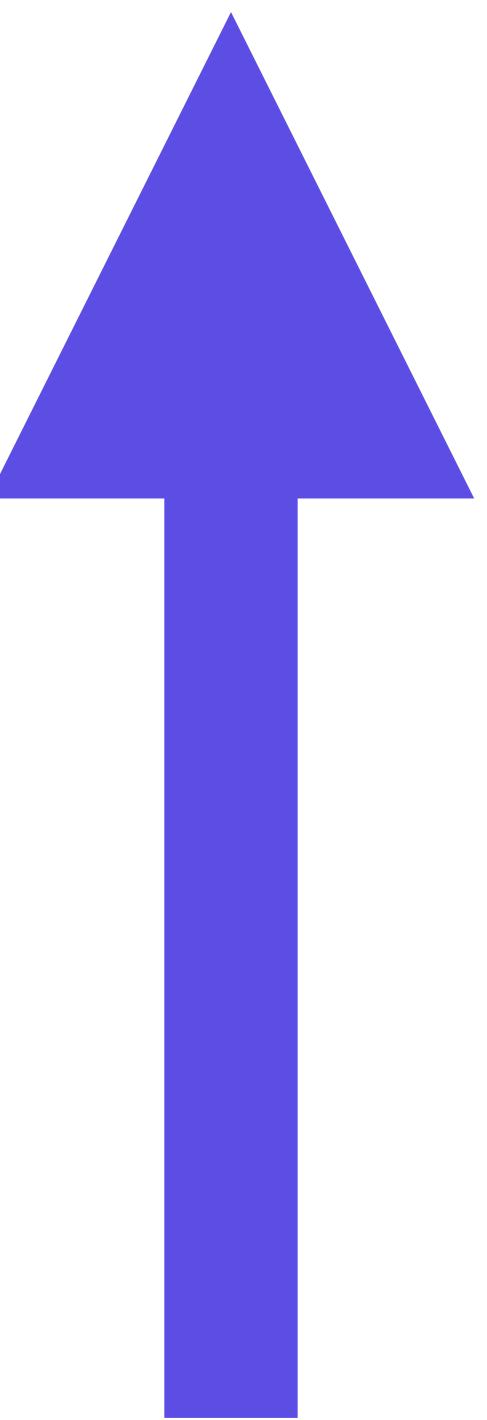
A Little
More Pain

Even More
Pain

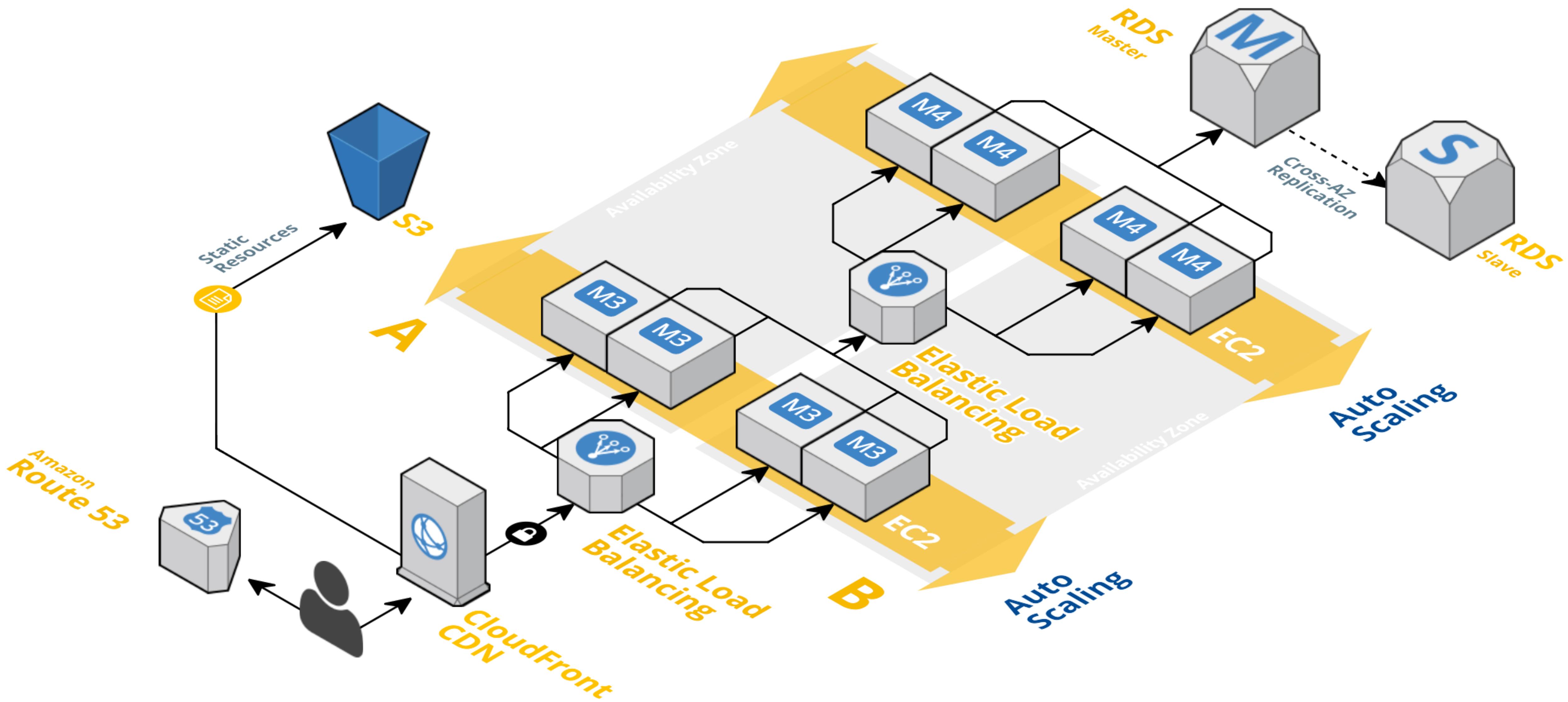
A Whole Lot
Of Pain

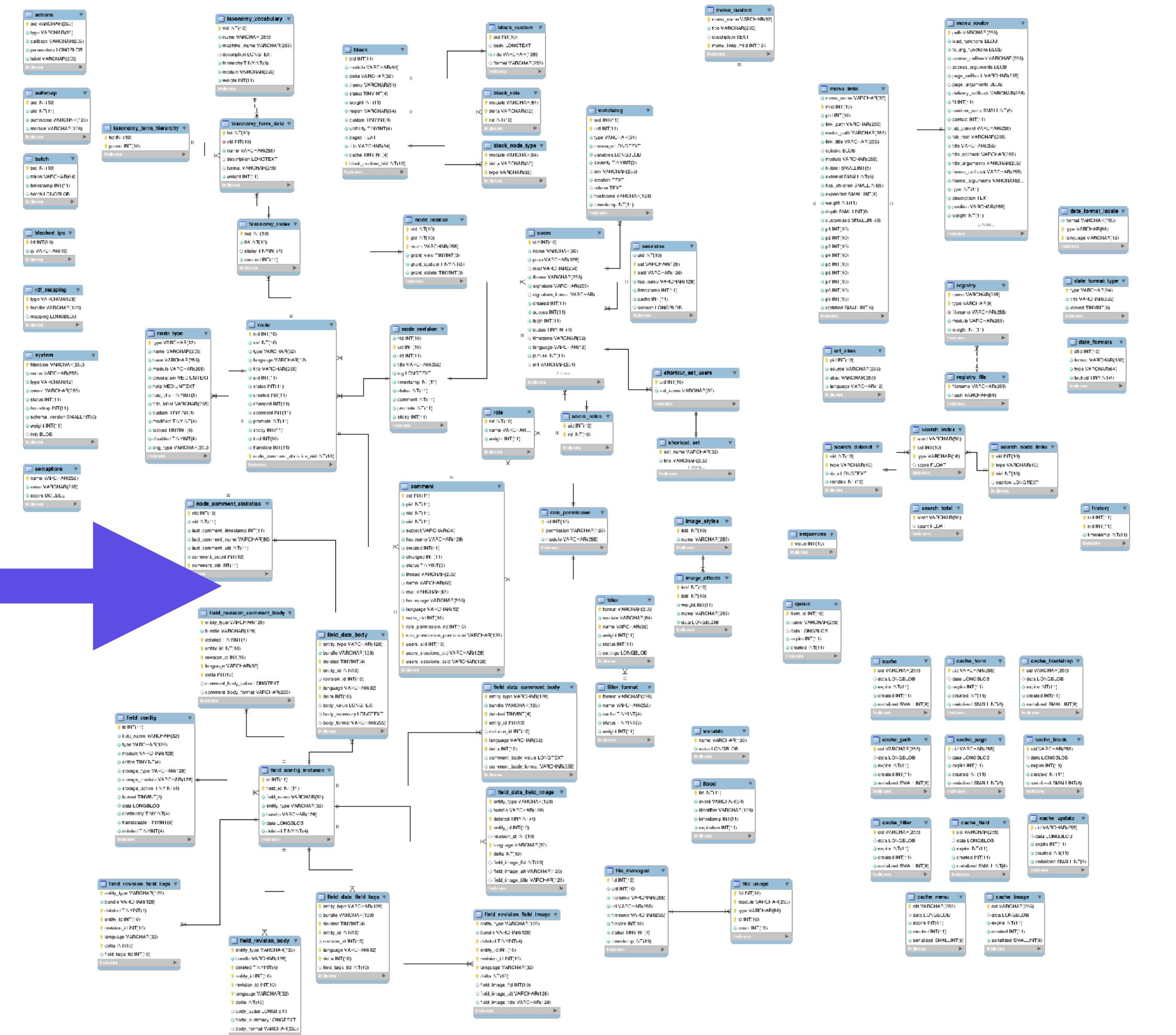
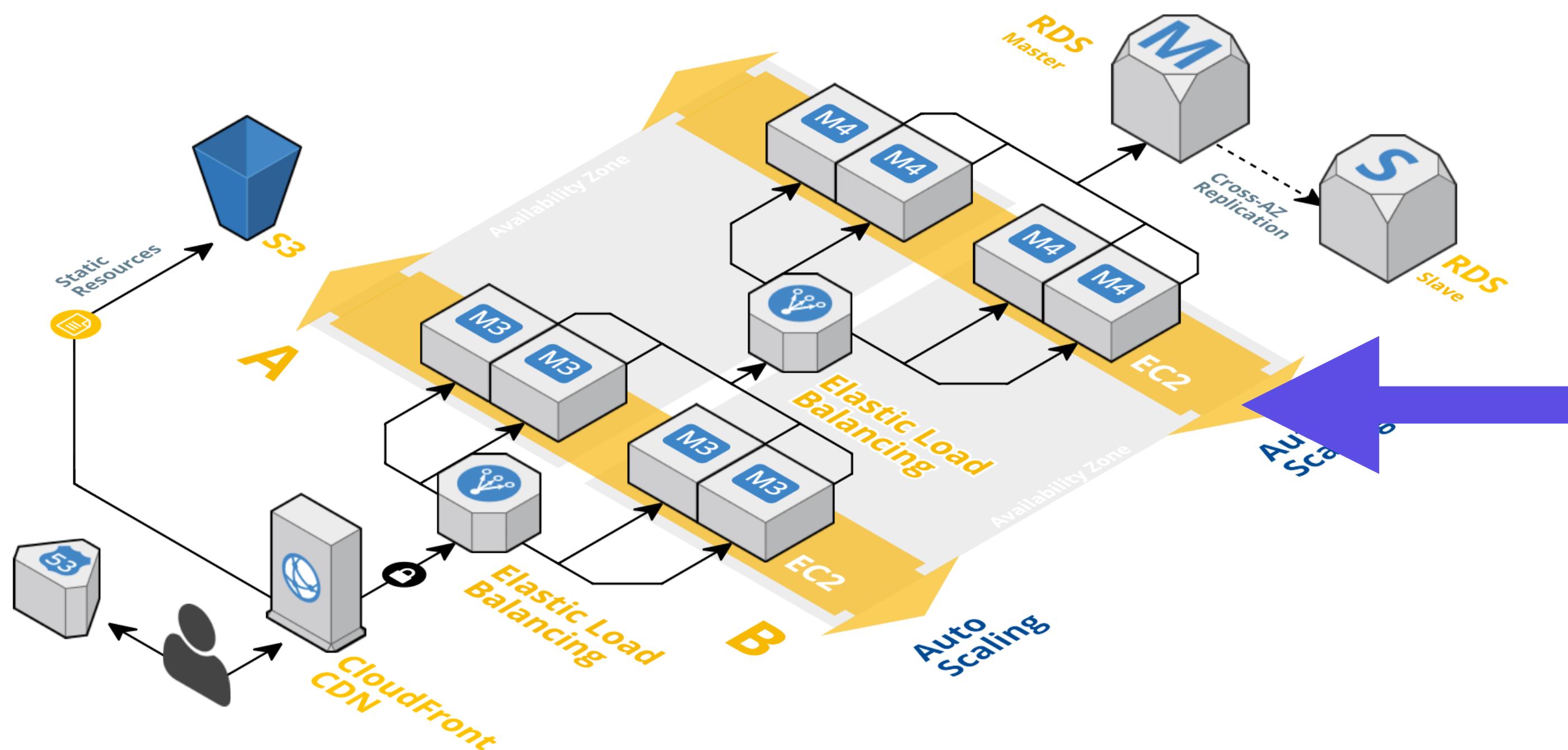
Worst
Pain

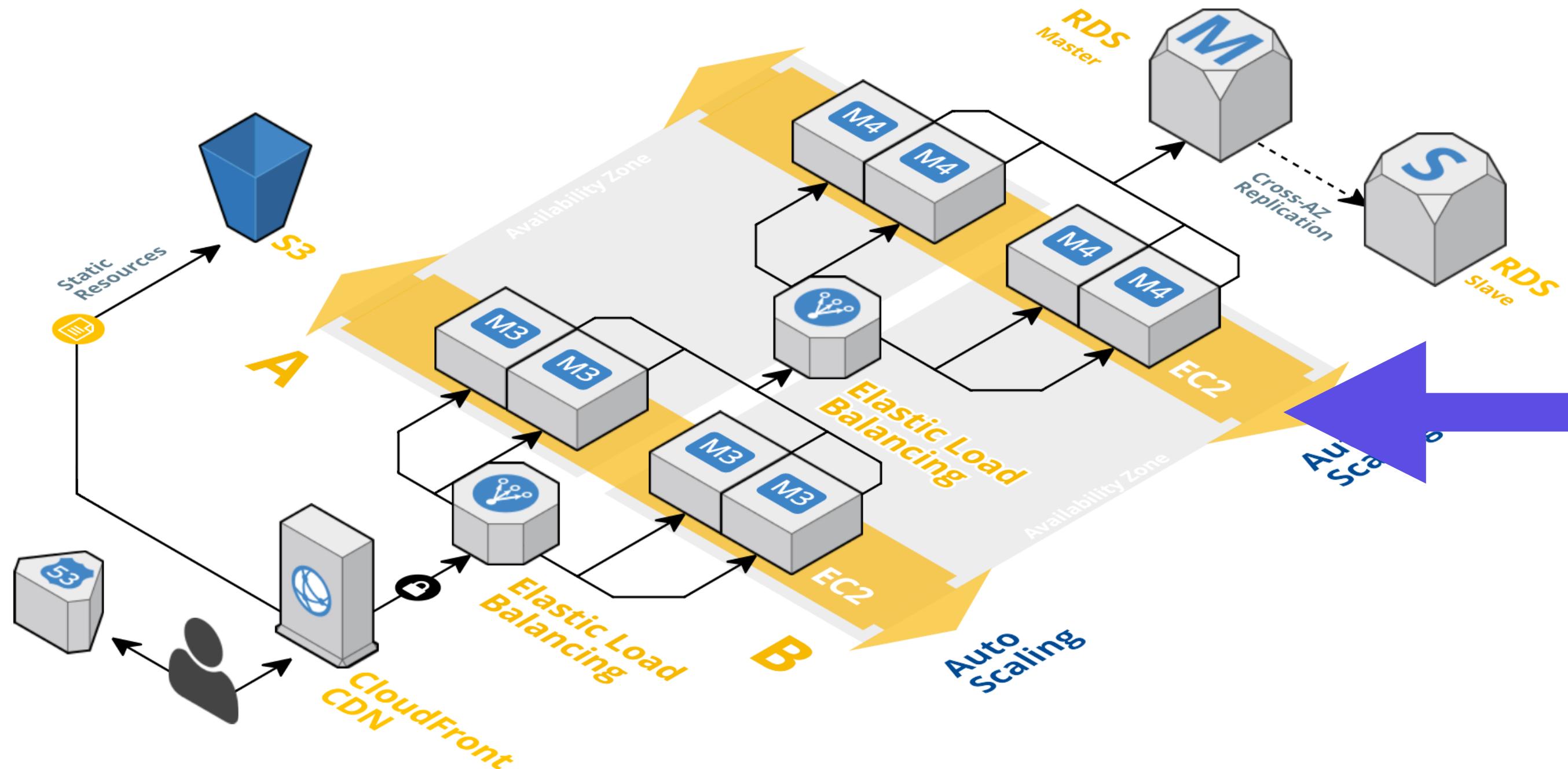
Capability



Complexity



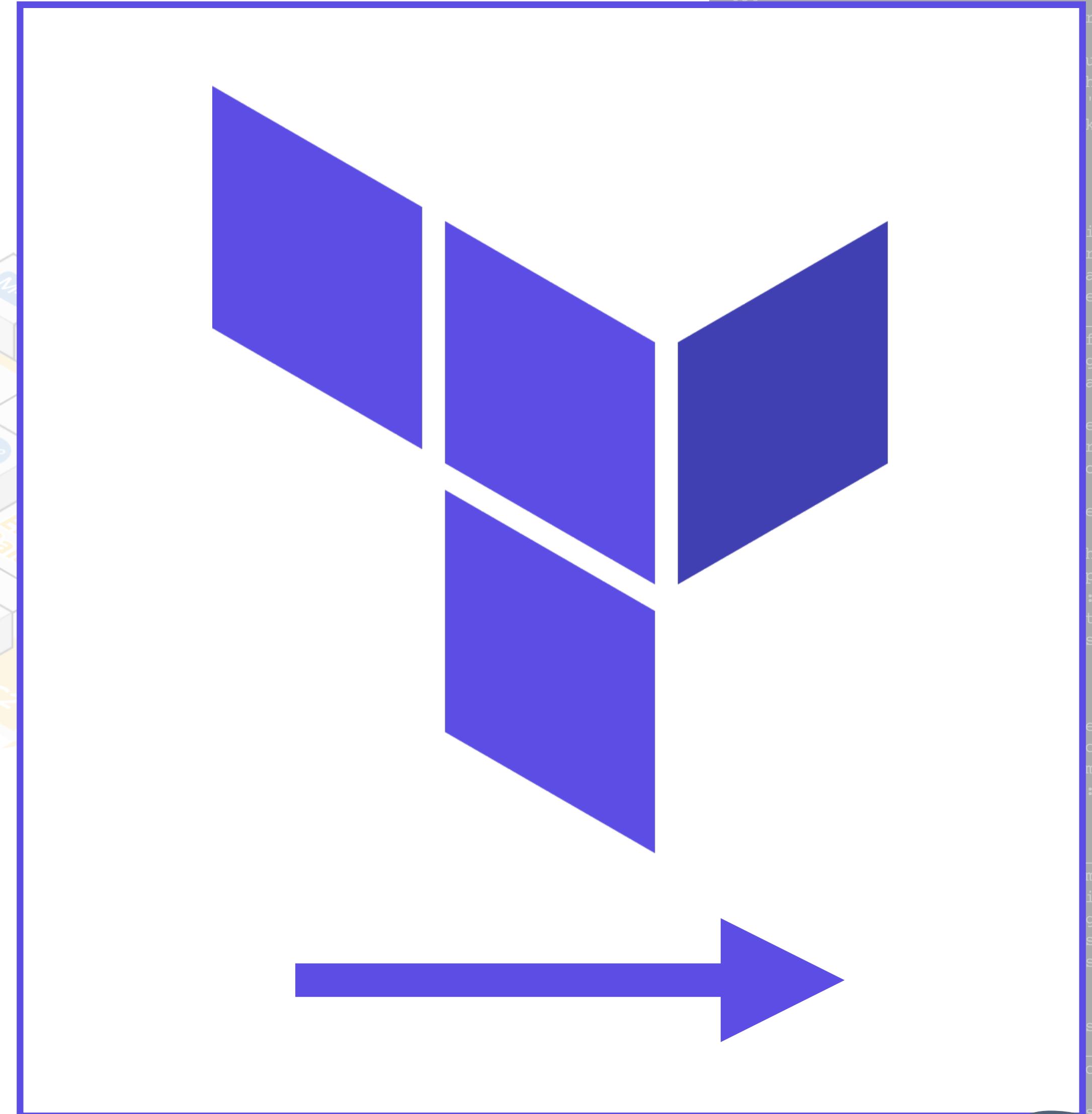
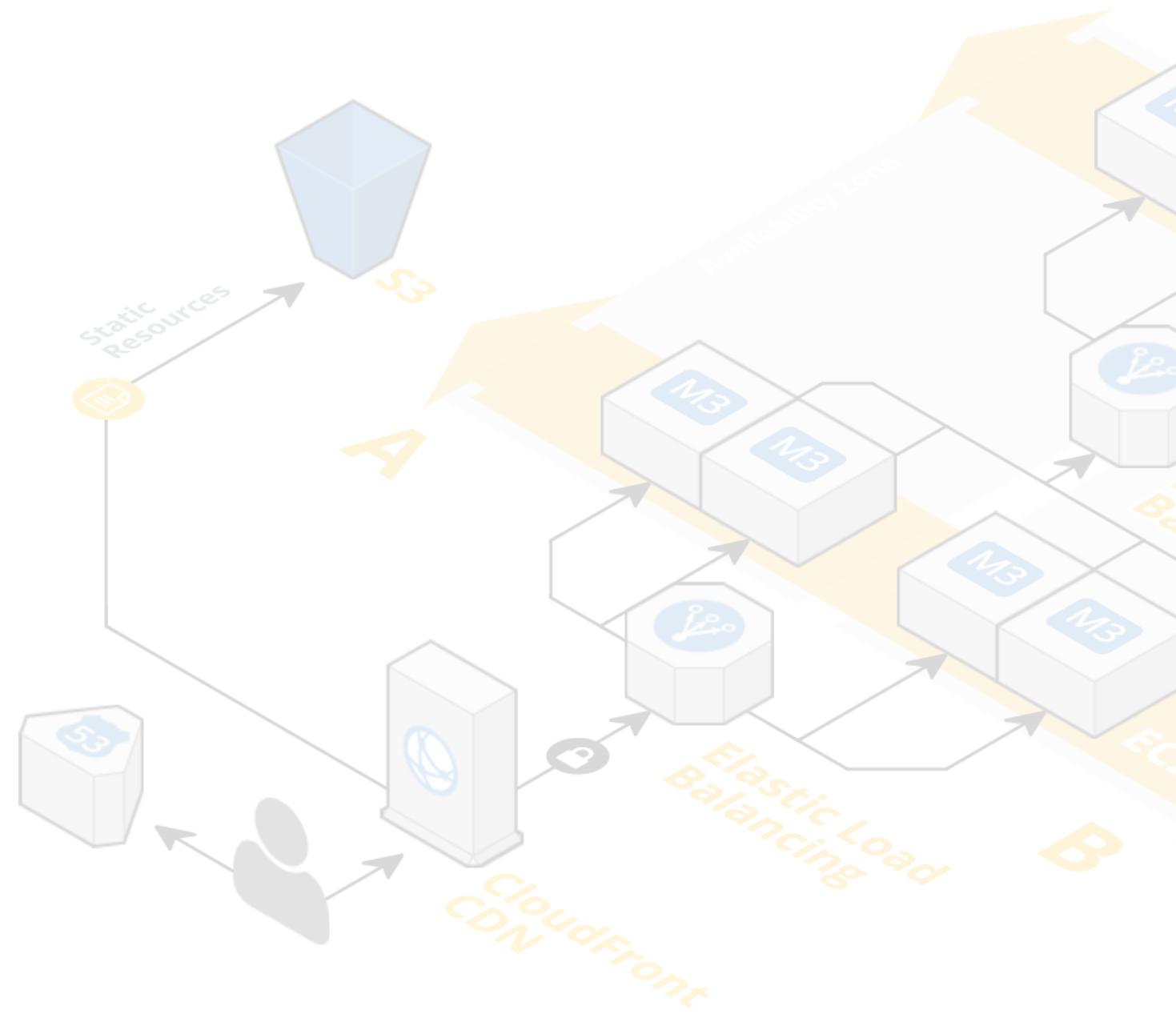




```

475  def update
476    return update_api if api_request?
477
478    if authorized_action(@account, @current_user, :manage_account_settings)
479      respond_to do |format|
480
481        custom_help_links = params[:account].delete :custom_help_links
482        if custom_help_links
483          @account.settings[:custom_help_links] = custom_help_links.select{|k, h| h['state'] != 'deleted'}
484          hash = index_with_hash[1]
485          hash.delete('state')
486          hash.assert_valid_keys ["text", "subtext", "url", "available_to"]
487          hash
488        end
489      end
490
491    params[:account][:turnitin_host] = validated_turnitin_host(params[:account][:turnitin_host])
492    enable_user_notes = params[:account].delete :enable_user_notes
493    allow_sis_import = params[:account].delete :allow_sis_import
494    params[:account].delete :default_user_storage_quota_mb unless @account.root_account? && !@account.grants_right?(@current_user, :manage_storage_quotas)
495    unless @account.grants_right?(@current_user, :manage_storage_quotas)
496      [:storage_quota, :default_storage_quota, :default_storage_quota_mb,
497       :default_user_storage_quota, :default_user_storage_quota_mb,
498       :default_group_storage_quota, :default_group_storage_quota_mb].each { |key| params[:account].set(key, params[:account].delete(key)) }
499    end
500    if params[:account][:services]
501      params[:account][:services].slice(*Account.services_exposed_to_ui_hash(nil, @current_user, @account))
502      @account.set_service_availability(key, value == '1')
503    end
504    params[:account].delete :services
505    if @account.grants_right?(@current_user, :manage_site_settings)
506      # If the setting is present (update is called from 2 different settings forms, one for notifications)
507      if params[:account][:settings] && params[:account][:settings][:outgoing_email_default_name]
508        # If set to default, remove the custom name so it doesn't get saved
509        params[:account][:settings][:outgoing_email_default_name] = '' if params[:account][:settings][:outgoing_email_default_name] == params[:account][:settings][:outgoing_email_default_name]
510      end
511
512      google_docs_domain = params[:account][:settings].try(:delete, :google_docs_domain)
513      if @account.feature_enabled?(:google_docs_domain_restriction) && @account.root_account? && !@account.site_admin?
514        @account.settings[:google_docs_domain] = google_docs_domain.present? ? google_docs_domain : nil
515      end
516
517      @account.enable_user_notes = enable_user_notes if enable_user_notes
518      @account.allow_sis_import = allow_sis_import if allow_sis_import && @account.root_account?
519      if @account.site_admin? && params[:account][:settings]
520        # these shouldn't get set for the site admin account
521        params[:account][:settings].delete(:enable_alerts)
522        params[:account][:settings].delete(:enable_eportfolios)
523      end
524    else
525      # must have :manage_site_settings to update these
526      [ :admins_can_change_passwords,
527        :admins_can_view_notifications,
528        :enable_alerts,
529        :enable_eportfolios,
530        :enable_profiles,
531        :show_scheduler,
532        :global_includes,
533        :gmail_domain
534      ].each do |key|
535        params[:account][:settings].try(:delete, key)
536      end
537    end
538  end

```



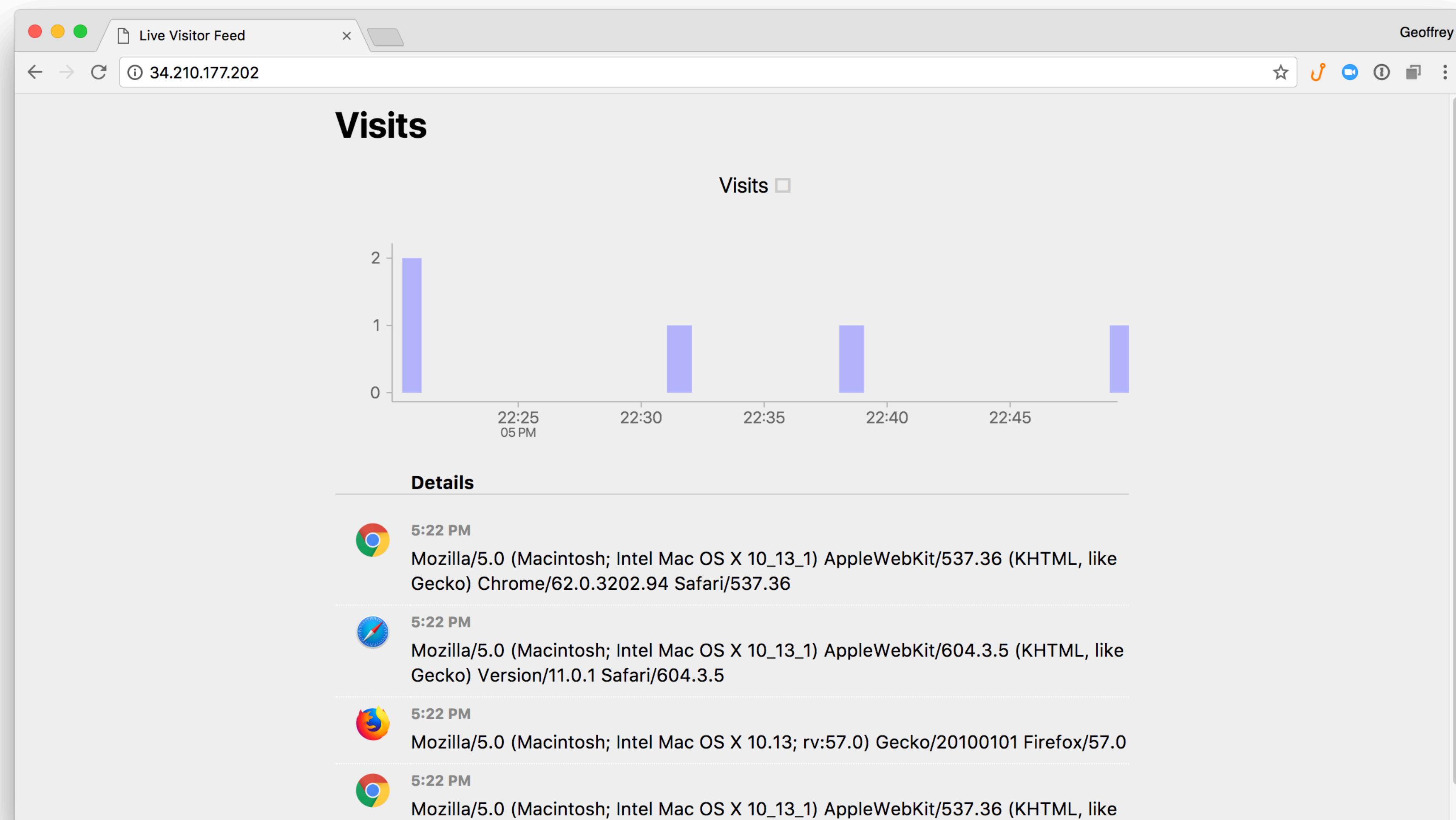
Infrastructure as Code

```

475  def update
476    return update_api if api_request?
477
478    if authorized_action(@account, @current_user, :manage_account_settings)
479      respond_to do |format|
480
rams[:account].delete :custom_help_links
custom_help_links = custom_help_links.select{|k, h| h['state'] != 'delet
hash[1]
')
keys ["text", "subtext", "url", "available_to"]
itin_host] = validated_turnitin_host(params[:account][:turnitin_host])
rams[:account].delete :enable_user_notes
ams[:account].delete :allow_sis_import
e :default_user_storage_quota_mb unless @account.root_account? && !@acco
_right? @current_user, :manage_storage_quotas
fault_storage_quota, :default_storage_quota_mb,
ge_quota, :default_user_storage_quota_mb,
age_quota, :default_group_storage_quota_mb].each { |key| params[:account]
ervices]
rvices].slice(*Account.services_exposed_to_ui_hash(nil, @current_user, @
ce_availability(key, value == '1'))
ete :services
ht?(@current_user, :manage_site_settings)
present (update is called from 2 different settings forms, one for notif
:settings] && params[:account][:settings][:outgoing_email_default_name_o
t, remove the custom name so it doesn't get saved
settings][:outgoing_email_default_name] = '' if params[:account][:settin
params[:account][:settings].try(:delete, :google_docs_domain)
enabled?(:google_docs_domain_restriction) &&
ount? &&
min?
:google_docs_domain] = google_docs_domain.present? ? google_docs_domain
_notes = enable_user_notes if enable_user_notes
import = allow_sis_import if allow_sis_import && @account.root_account?
in? && params[:account][:settings]
get set for the site admin account
settings].delete(:enable_alerts)
settings].delete(:enable_eportfolios)
site_settings to update these
_passwords,
otifications,
s,
:enable_pro
:enable_sched
:bal_in
:go
:do |key,
params[:account][:settings].try(:delete, key)
533
536
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
857
858
859
859
860
861
862
863
864
865
866
867
867
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
957
958
959
959
960
961
962
963
964
965
966
967
967
968
969
969
970
971
972
973
974
975
976
977
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999

```

What You'll Build



EXERCISE

Instructor Demo

Build a server on AWS that runs a single binary web application written in Go.

KNOW IT WORKED IF: You visit the IP address in the web browser and can see a chart with a list of visitors.

EXERCISE

Lab 0: Choose your editor

Duration: 15 minutes

Goal: Make sure you can login to your workstation to run commands and edit files.

Basic Configuration

What You'll Learn

- ☑ Login to your workstation
- ☑ Create a basic Terraform configuration
- ☑ Initialize and apply the configuration (create infrastructure)
- ☑ Change and re-apply the configuration

EXERCISE

Explore Workstation

Use VSCode, SSH, or the web console SSH tool to login to your workstation.

Change directory to `/workstation/terraform` and look inside `main.tf`.

Open Folder

EXPLORER

OPEN EDITORS

X *Untitled-1*

NO FOLDER OPENED

Connected to SSH: 34.2

Open Folder

/workstation/terraform/

OK

Show Local

..

.git/

.terraform/

assets/

getting-started/

keys/

webapp/

PROBLEMS

OUTPUT

TERMINAL

...

1: bash

+ □ ✖ ˄ ×

```
baby-shark@robin-training-env-ant:~ > cd /workstation/terraform
baby-shark@robin-training-env-ant:/workstation/terraform > ls
assets          keys        README.md      terraform.tfstate.backup  webapp
getting-started main.tf  terraform.tfstate  terraform.tfvars
baby-shark@robin-training-env-ant:/workstation/terraform > 
```

```
> cd /workstation/terraform  
  
> nano main.tf # Or use `vim main.tf` if you prefer
```

```
#  
# Your security group ID is:  
#  
#      abcd1234  
#  
# ...  
  
provider "aws" {  
    access_key = "AKIAIZT7VATBGPC3AP3Q"  
    secret_key = "W/txHyHA8SsevaENCsUIS9/KGoIScy2yoqrdfodKz"  
    region     = "us-west-2"  
}  
  
resource "aws_instance" "web" {  
    # ...  
}
```

```
provider "aws" {
    access_key = "AKIAIZT7VATBGPC3AP3Q"
    secret_key = "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"
    region     = "us-west-2"
}

resource "aws_instance" "web" {
    ami           = "ami-0e63f50857fdc1f9f"
    instance_type = "t2.micro"

    subnet_id          = "subnet-0c5f00e7c2fe92579"
    vpc_security_group_ids = ["sg-0e8be760e755de7cc"]

    tags = {
        "Identity" = "<user>-training-env-<animal>"
    }
}
```

Command Line Interface

All interactions with Terraform occur via the CLI

Terraform is a local tool (runs on the current machine)

The terraform ecosystem also includes providers for many cloud services, and a module repository

Hashicorp also has products to help teams manage Terraform: Terraform Cloud and Terraform Enterprise

```
> terraform help
```

Common commands:

apply

console

destroy

env

fmt

get

graph

import

init

output

plan

providers

push

refresh

show

taint

Builds or changes infrastructure

Interactive console for Terraform interpolations

Destroy Terraform-managed infrastructure

Workspace management

Rewrites config files to canonical format

Download and install modules for the configuration

Create a visual graph of Terraform resources

Import existing infrastructure into Terraform

Initialize a Terraform working directory

Read an output from a state file

Generate and show an execution plan

Prints a tree of the providers used in the config

Upload this Terraform module to Atlas to run

Update local state file against real resources

Inspect Terraform state or plan

Manually mark a resource for recreation

Terraform Init

You must run `terraform init` to initialize a new Terraform working directory, and after changing provider configuration

You **do not** need to run `terraform init` before each command

Similar to `git init`

```
> terraform init  
Initializing provider plugins...  
- Checking for available provider plugins on https://releases...  
- Downloading plugin for provider "aws" (2.16)...
```

The following providers do not have any version constraints in configuration, so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking changes, it is recommended to add `version = "..."` constraints to the corresponding provider blocks in configuration, with the constraint strings suggested below.

```
* provider.aws: version = "~> 2.16"
```

Terraform has been successfully initialized!

Command: terraform plan

The plan shows you what will happen

You can save plans to guarantee what will happen

Plans show reasons for certain actions (such as re-create)

Prior to Terraform, users had to guess change ordering, parallelization, and rollout effect

Command: terraform plan

- + resource will be created
- resource will be destroyed
- ~ resource will be updated in-place
- /+ resources will be destroyed and re-created

```
> terraform plan
+ resource "aws_instance" "web" {
    + ami                                = "ami-0e63f50857fdc1f9f"
    + arn                                = (known after apply)
    + availability_zone                  = (known after apply)
    + cpu_core_count                     = (known after apply)
    + get_password_data                 = false
    + instance_type                      = "t2.micro"
    + placement_group                   = (known after apply)
    + private_ip                         = (known after apply)
    + public_dns                          = (known after apply)
    + public_ip                           = (known after apply)
    + security_groups                    = (known after apply)
# ...
Plan: 1 to add, 0 to change, 0 to destroy.
```

Command: terraform apply

Executes changes in order based on the resource graph

Parallelizes changes when possible

Handles and recovers transient errors

Runs plan first unless given a serialized plan file

Command: terraform apply

Updates existing resources when updates are allowed

Re-creates existing resources when updates are not allowed

```
> terraform apply
```

```
# ...
```

```
Plan: 1 to add, 0 to change, 0 to destroy.
```

```
Do you want to perform these actions?
```

```
Terraform will perform the actions described above.
```

```
Only 'yes' will be accepted to approve.
```

```
Enter a value: yes
```

```
aws_instance.web: Creating...
```

```
aws_instance.web: Still creating... [10s elapsed]
```

```
aws_instance.web: Still creating... [20s elapsed]
```

```
aws_instance.web: Still creating... [30s elapsed]
```

```
aws_instance.web: Creation complete after 31s [id=i-0b8bf14603ebdc264]
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Command: terraform show

Displays human-friendly state

Can be used to get on-the-fly information

```
> terraform show  
# aws_instance.web:  
resource "aws_instance" "web" {  
    ami                                = "ami-0e63f50857fdc1f9f"  
    arn                                = "..."  
    associate_public_ip_address        = true  
    availability_zone                  = "us-west-2a"  
    cpu_core_count                     = 1  
    cpu_threads_per_core              = 1  
    disable_api_termination           = false  
    ebs_optimized                      = false  
    get_password_data                 = false  
    id                                 = "i-09c617e1c6d3514a8"  
    # ...  
    public_ip                         = "34.214.210.211"
```

```
$ ping 34.214.210.211
PING 34.214.210.211 (34.214.210.211) 56(84) bytes of
data.

64 bytes from 34.214.210.211: icmp_seq=1 ttl=63
time=0.807 ms
64 bytes from 34.214.210.211: icmp_seq=2 ttl=63
time=0.770 ms
<Ctrl-C>
```

```
resource "aws_instance" "web" {
    ami              = "ami-0e63f50857fdc1f9f"
    instance_type   = "t2.micro"

    subnet_id        = "subnet-0c5f00e7c2fe92579"
    vpc_security_group_ids = [ "sg-0e8be760e755de7cc" ]

    tags = {
        "Identity"  = "..."
        "Name"       = "Moose"
        "Zip"        = "zap"
    }
}
```

```
> terraform plan
aws_instance.web: Refreshing state...
[ id=i-0b8bf14603ebdc264 ]
# ...
# aws_instance.web will be updated in-place
~ resource "aws_instance" "web" {
# ...
~ tags      = {
    "Identity" = "..."
+ "Name"      = "Moose"
+ "Zip"       = "zap"
}
# ...
Plan: 0 to add, 1 to change, 0 to destroy.
```

```
> terraform apply
aws_instance.web: Refreshing state...
[ id=i-0b8bf14603ebdc264 ]
# ...
# aws_instance.web will be updated in-place
~ resource "aws_instance" "web" {
# ...
~ tags      = {
    "Identity" = "..."
+ "Name"      = "Moose"
+ "Zip"       = "zap"
}
# ...
Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

Terraform 0.12

Released in May of 2019

Better expression syntax, type support, iteration constructs, and many other improvements

The command `terraform 0.12upgrade` will upgrade configuration to the new syntax

Use Version Control Tools For Rollback

Terraform only knows about your configuration and the state of the infrastructure it has built.

If you need to rollback, use the capabilities of your version control software to revert to an earlier version of `main.tf`, then run `terraform apply` on it.

What You Learned

- ✓ Login to your workstation
- ✓ Create a basic Terraform configuration
- ✓ Initialize and apply the configuration (create infrastructure)
- ✓ Change and re-apply the configuration

EXERCISE

Lab 1: Lab Setup

Duration: 20 minutes

Goal: Familiarize yourself with your workstation and the Terraform CLI

Outputs

What You'll Learn

- ➊ Print dynamic data with an output
- ➋ Query specific values from the output

Outputs

Outputs define useful values that will be highlighted to the user when Terraform applies: IP addresses, usernames, generated keys

Outputs can be queried using the `terraform output` command

Outputs are a way to easily extract and query information from all of Terraform's collected data

```
resource "aws_instance" "web" {
# ...
}

output "public_ip" {
  value = aws_instance.web.public_ip
}

# Prior to Terraform 0.12 syntax:

output "public_ip" {
  value = "${aws_instance.web.public_ip}"
}
```

```
resource "aws_instance" "web" {
# ...
}

output "public_ip" {
  value = aws_instance.web.public_ip
}

output "public_dns" {
  value = aws_instance.web.public_dns
}
```

```
> terraform refresh  
aws_instance.web: Refreshing state... [id=i-0b8bf14603ebdc264]
```

Outputs:

```
public_dns = ec2-34-222-156-11.us-west-2.compute.amazonaws.com  
public_ip = 34.222.156.11
```

```
> terraform output  
public_dns = ec2-34-222-156-11.us-west-2.compute.amazonaws.com  
public_ip = 34.222.156.11
```

```
> terraform output public_dns  
ec2-34-222-156-11.us-west-2.compute.amazonaws.com
```

```
> terraform output public_dns
ec2-34-222-156-11.us-west-2.compute.amazonaws.com
> ping ${terraform output public_dns)
PING ec2-34-222-156-11.us-west-2.compute.amazonaws.com
(34.222.156.11) 56(84) bytes of data.
64 bytes from ec2-34-222-156-11.us-
west-2.compute.amazonaws.com (34.222.156.11): icmp_seq=1
ttl=63 time=0.440 ms
64 bytes from ec2-34-222-156-11.us-
west-2.compute.amazonaws.com (34.222.156.11): icmp_seq=2
ttl=63 time=0.570 ms
<Ctrl-C>
```

What You Learned

- ☑ Print dynamic data with an output
- ☑ Query specific values from the output

EXERCISE

Lab 2: Outputs

Duration: 10 minutes

Goal: Configure specific outputs and learn to query your configuration for those.

Console

What You'll Learn

- ☑ Interact with live data in the console
- ☑ Use the console from the command line

REPL

A REPL is a Read-Evaluate-Print-Loop

The Terraform console provides an interactive REPL for writing, executing, and querying Terraform configuration

Command: terraform console

The `terraform console` command creates an interactive console for interacting with Terraform

This is useful for experimenting with interpolations and interacting with Terraform state

```
> terraform console
> aws_instance.web.ami
ami-0e63f50857fdc1f9f
> aws_instance.web.public_ip
54.174.40.55
> aws_instance.web.tags
{
  "Identity" = "robin-training-env-ant"
  "Name" = "Moose"
  "Zip" = "zap"
}
> <Ctrl-D>
```

```
> terraform console  
> exit
```

```
> echo "aws_instance.web.ami" | terraform console  
ami-0e63f50857fdc1f9f
```

EXERCISE

Lab 3: REPL

Duration: 10 minutes

Goal: The Read-Evaluate-Print-Loop is useful for understanding interpolation and finding specific information.

Auto-Formatting

Auto-Formatting

Terraform has built-in support for auto-formatting configuration files to match the HCL specification.

There is no need to manually align things.

This helps keep configurations VCS-friendly and reduces bike-shed arguments over formatting styles.

Plugins exist for most major editors.

Command: terraform fmt

The `terraform fmt` command formats configuration files.

It does not prompt for confirmation or

It also reports on syntax errors, if they exist.

EXERCISE

Format Your Code

Run the `terraform fmt` command with no arguments. By default it will choose all Terraform configurations in the current working directory non-recursively.

Open the `main.tf` file and see that it has been formatted.

Command: terraform fmt

Please feel free to use the `terraform fmt` command through this training and at your leisure, but we will not explicitly discuss it anymore.

Variables

What You'll Learn

- ✓ Use variables to pass values to your configuration
- ✓ Refactor existing configuration to use variables
- ✓ Keep sensitive data out of source code control
- ✓ Pass variables to Terraform in several ways

Variables

Work a lot like variables from programming languages

Allow you to remove hard coded values and pass parameters to your configuration

Can help make configuration easier to understand and re-use

Must always have a value. Variables are never optional, but they can have a default value

```
variable "animal" {}
```

```
variable "animal" {  
    type      = "string"  
    default   = "moose"  
    description = "An animal, such as whale, elephant, or  
bear"  
}
```

Variable Types

The most common variable types are `string`, `number`, `list`, and `map`

Other support types include `bool` (true/false), `set`, `object`, and `tuple`

If omitted, the type is inferred from the default value

If neither type nor default is provided, the type is assumed to be `string`

```
variable "access_key" {}
variable "secret_key" {}
variable "region" {
  default = "us-west-2"
}
```

```
variable "access_key" {  
    type = "string"  
}  
  
variable "secret_key" {  
    type = "string"  
}  
  
variable "region" {  
    type     = "string"  
    default = "us-west-2"  
}
```

```
variable "access_key" {  
    type      = "string"  
    description = "AWS access key"  
}  
  
variable "secret_key" {  
    type      = "string"  
    description = "AWS secret access key"  
}  
  
variable "region" {  
    type      = "string"  
    default   = "us-west-2"  
    description = "AWS region"  
}
```

Using Variables

Variables can be referred to by using the syntax
var.<name>

Variables' values can also be interpolated into a string using the syntax “ ***\${var.<name>}*** ”

```
provider "aws" {
    access_key = var.access_key
    secret_key = var.secret_key
    region     = var.region
}

# ...
```

```
> terraform plan
```

```
var.access_key
```

```
Enter a value:
```

```
<Ctrl + C>
```

```
> export TF_VAR_access_key=<access_key>  
  
> terraform plan  
var.secret_key  
Enter a value:  
  
<Ctrl + C>
```

```
access_key = "AKIAIZT7VATBGPC3AP3Q"
secret_key = "W/txHyHeNCsUIS9/KGoIScy2yoqrdfodKz"
# ami = "ami-0e63f50857fdc1f9f"
# subnet_id = "subnet-0c5f00e7c2fe92579"
# identity = "robin-training-env-ant"
# region = "us-west-2"
# vpc_security_group_ids = ["sg-0e8be760e755de7cc"]
```

```
> terraform plan -var secret_key=<secret_key>  
aws_instance.web: Refreshing state... (ID:  
i-02f5717f1a84502ed)
```

```
# ...
```

No changes. Infrastructure is up-to-date.

```
# ...
```

```
> terraform plan -var-file secrets.tfvars
```

Specifying Variable Values

1. Default values
2. Environment variables
3. Saved in variable definition files (`terraform.tfvars`)
4. Using the `-var` or `-var-file` command line option
5. Configured in a Terraform Enterprise or Terraform Cloud workspace
6. Entered via CLI prompts

```
variable "num_webservers" {
    type     = "number"
    default  = 5
}

variable "zones" {
    type     = "list"
    default  = [ "us-east-1a", "us-east-1b" ]
}

variable "image_ids" {
    type     = "map"
    default  = {
        us-east-1 = "image-1234"
        us-west-2 = "image-4567"
    }
}
```

WARNING

Variables & Security

Never put secret values, like passwords or access tokens, in .tf files or other files that are checked into source control.

TIP

Use git secrets Plugin

Install on each repository.

It will warn you if you try to commit code that looks like an AWS secret. (Configurable for others, too.)

<https://github.com/awslabs/git-secrets>

What You Learned

- ✓ Use variables to store sensitive values
- ✓ Refactor existing configuration to use variables
- ✓ Keep sensitive data out of source code control systems
- ✓ Populate variables to Terraform in several ways

EXERCISE

Lab 4: Variables

Duration: 15 minutes

Goal: Remove all hardcoded values in your current configuration and use variables instead.

Configuration Format

Configuration Syntax

Configurations are written in HashiCorp Configuration Language (HCL)

HCL is designed to strike a balance between human-readable and machine-parsable

Terraform can also read configuration in JSON, but we recommend using HCL, except when the configuration is machine-generated

Load Order & Semantics

Configuration can be in a single file or split across multiple files.
Terraform will process all files in the current working directory
which end in `.tf` or `.tf.json`

Sub-folders are not included (non-recursive)

Files are processed in lexicographical (dictionary) order

Any files with a different extension are ignored (e.g. `.jpg`
or `.tf.old`)

Generally, the order in which things are defined doesn't matter.

Load Order & Semantics

Parsed configurations are *appended* to each other, not merged

Resources with the same name are **not** merged (this will produce an error)

Configuration syntax is declarative, so references to other resources do not depend on the order they are defined

EXERCISE

Which Files Does Terraform Use?

Which of the following files would be parsed by Terraform when a command is issued?

```
> tree
.
├── README.md
├── main.tf
├── other.tf
├── gravatar.jpg
└── alphabet
    ├── README.md
    └── main.tf
```

```
> tree  
.  
├── README.md  
├── main.tf  
├── other.tf  
├── gravatar.jpg  
└── alphabet  
    ├── README.md  
    └── main.tf
```

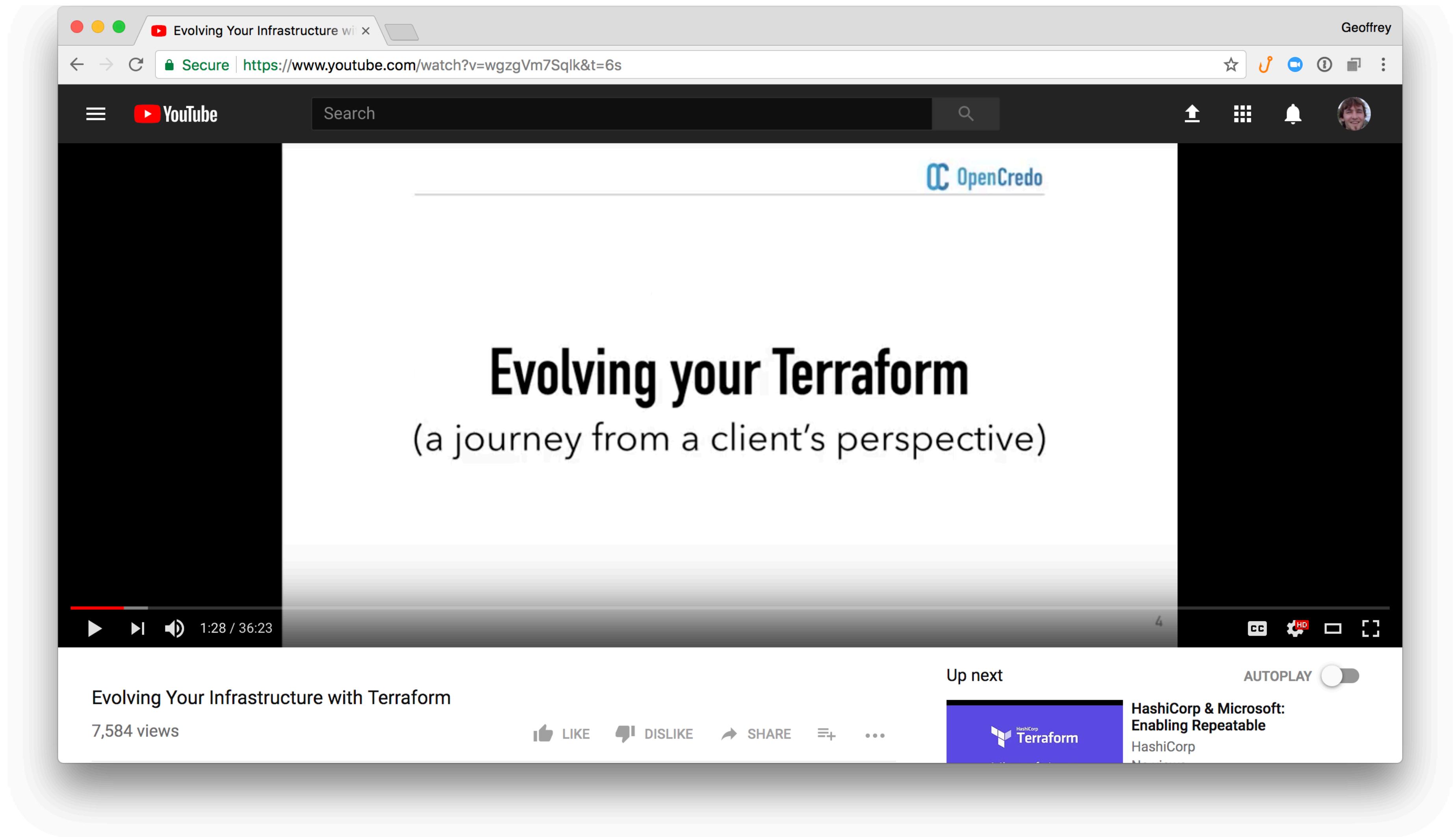
Configuration Organization Patterns

Teams choose to organize their configurations in many different patterns.

```
> ls  
main.tf  
outputs.tf  
variables.tf
```

```
> ls  
instances.tf  
load-balancers.tf  
shared.tf
```

```
> ls  
everything.tf
```



```
resource "aws_instance" "web" {  
    ami              = "ami-0e63f50857fdc1f9f"  
    instance_type   = "t2.micro"  
}
```

Top Level Keywords

provider

variable

resource

output

module

data

terraform

locals

```
resource "aws_instance" "web" {  
    ami              = "ami-0e63f50857fdc1f9f"  
    instance_type   = "t2.micro"  
}
```

```
resource "aws_instance" "web" {  
    ami              = "ami-0e63f50857fdc1f9f"  
    instance_type   = "t2.micro"  
}
```

```
resource "aws_instance" "web" {  
    ami              = "ami-0e63f50857fdc1f9f"  
    instance_type   = "t2.micro"  
}  
  
resource "aws_instance" "web" {  
    ami              = "ami-07669fc90e6e6cc47"  
    instance_type   = "t2.micro"  
}
```

```
resource "aws_instance" "web" {  
    ami              = "ami-0e63f50857fdc1f9f"  
    instance_type   = "t2.micro"  
}  
  
resource "aws_instance" "web-2" {  
    ami              = "ami-07669fc90e6e6cc47"  
    instance_type   = "t2.micro"  
}
```

```
resource "aws_instance" "web" {
    ami              = "ami-0e63f50857fdc1f9f"
    instance_type   = "t2.micro"
}

resource "digitalocean_droplet" "web" {
    image     = "ubuntu-18-04-x64"
    name      = "web"
    size      = "s-1vcpu-1gb"
    region    = "nyc2"
}
```

```
resource "aws_instance" "web" {  
    ami              = "ami-0e63f50857fdc1f9f"  
    instance_type   = "t2.micro"  
}
```

```
resource "aws_instance" "web" {  
    ami              = "ami-0e63f50857fdc1f9f"  
    instance_type   = "t2.micro"  
}
```

```
resource "aws_instance" "web" {
    ami              = "ami-0e63f50857fdc1f9f"
    instance_type   = "t2.micro"
}

resource "dnsimple_record" "web" {
    domain = "hashicorp.com"
    name   = "web"
    ttl    = "3600"
    type   = "A"
    value  = aws_instance.web.public_ip
# Terraform pre-0.12 syntax:
# value  = "${aws_instance.web.public_ip}"
}
```

```
# This is a comment
resource "aws_instance" "web" {
    ami              = "ami-0e63f50857fdc1f9f"
    instance_type   = "t2.micro"
}

/* This is a multiline
comment */
resource "dnsimple_record" "web" {
    # ...
}
```

```
variable "thing" {  
  description = <<EOF  
This is a  
multiline  
string  
EOF  
}
```

Syntax Highlighting

Plugins for Terraform/HCL exist for most major editors, but ruby tends to work best if one does not exist.

Modules

What You'll Learn

- ✓ The Terraform Public Module Registry
- ✓ Use a module from `main.tf`
- ✓ Understand how modules are stored on disk

Modules

Portable Terraform configurations (packages)

Allow separation of concerns and responsibilities
among teams

Parallels: Chef Cookbook, Puppet Module, Ruby gem

Modules

Modules are just Terraform configurations inside a folder - there's nothing special about them.

Complex configurations, team projects, and multi-repository codebases will benefit from modules. Get into the habit of using them wherever it makes sense.

```
> tree my-module
my-module
└── main.tf
```

```
variable "ami_id" {}
variable "instance_type" {}

resource "aws_instance" "db" {
    # ...
}

output "address" {
    value = aws_instance.db.public_dns
}
```

```
● ● ●  
module "my-module" {  
    # Source can be any URL or file path  
    source = "../../my-module"  
}
```

```
module "my-module" {
    # Source can be any URL or file path
    source = "../../my-module"

    ami          = "ami-0e63f50857fdc1f9f"
    instance_type = "m5.large"
}
```

```
module "my-module" {
    # Source can be any URL or file path
    source = "../../my-module"

    ami          = "ami-0e63f50857fdc1f9f"
    instance_type = "m5.large"
}

output "db-address" {
    value = module.my-module.address
}
```

Modules

Variables defined in a module become arguments to module block

Outputs defined in module become attributes from module definition

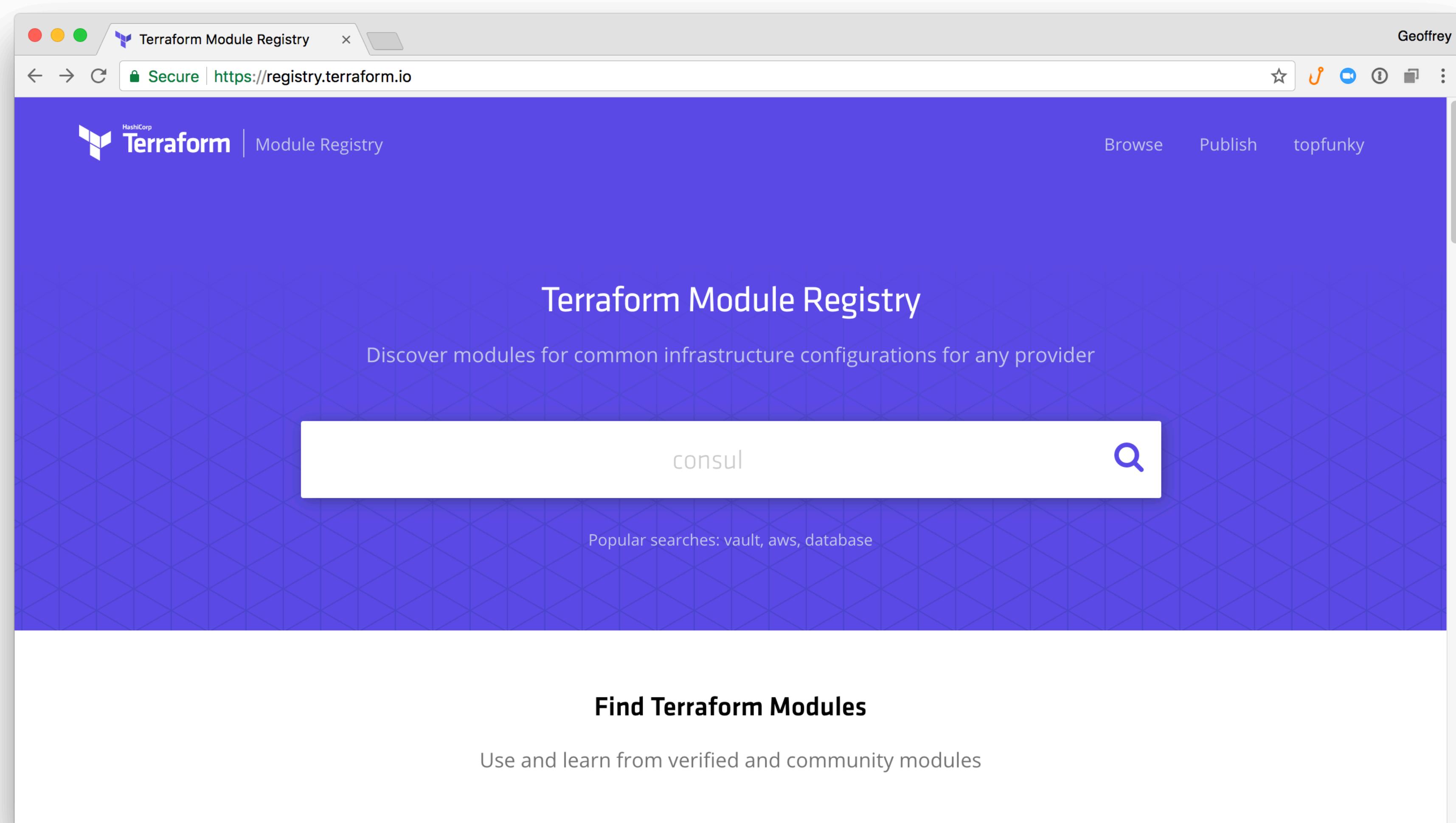
Configuration inside a module is a "black box" from the outside

Individual resources are not accessible outside the module

```
module "my-module" {
    # Source can be any URL or file path
    source = "../../my-module"

    argument_1 = "value"
    argument_2 = "value"
}

output "example" {
    value = module.my-module.aws_instance.db.public_ip
}
```





Terraform | Module Registry

Search



Browse

Publish

Sign-in



dynamic-keys

AWS

Terraform module that dynamically generates a public/private keypair.

Version 1.0.0 ▾

Published April 6, 2018 by mitchellh

Total provisions: 127

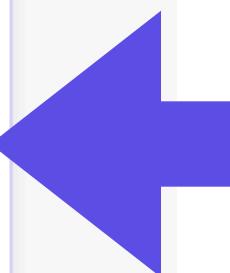
Source: github.com/mitchellh/terraform-aws-dynamic-keys (report an issue)

Provision Instructions

Copy and paste into your Terraform configuration, insert the variables, and run `terraform init`:

```
module "dynamic-keys" {
  source  = "mitchellh/dynamic-keys/aws"
  version = "1.0.0"

  # insert the 1 required variable here
}
```



```
variable "identity" { }

module "keypair" {
    source = "mitchellh/dynamic-keys/aws"
    path   = "${path.root}/keys"
    name   = "${var.identity}-key"
}
```

```
output "key_name" {
  value = aws_key_pair.generated.key_name
}

output "public_key_openssh" {
  value = tls_private_key.generated.public_key_openssh
}

output "private_key_pem" {
  value = tls_private_key.generated.private_key_pem
}

output "public_key_filepath" {
  value = local.public_key_filename
}

output "private_key_filepath" {
  value = local.public_key_filename
}
```

```
> terraform plan
```

```
Failed to load root config module: Error loading modules:  
module example: not found, may need to be downloaded  
using 'terraform get'
```

```
> terraform get
Downloading mitchellh/dynamic-keys/aws 1.0.0 for
keypair...
- keypair in .terraform/modules/keypair/mitchellh-
  terraform-aws-dynamic-keys-37d94b3
```

```
> terraform plan
```

Error: Could not satisfy plugin requirements

```
# ...
```

Error: provider.tls: no suitable version installed
version requirements: "(any version)"
versions installed: none

Error: provider.local: no suitable version installed
version requirements: "(any version)"
versions installed: none

```
> terraform init
```

```
Initializing modules...
```

```
Initializing the backend...
```

```
Initializing provider plugins...
```

```
- Checking for available provider plugins...
```

```
- Downloading plugin for provider "null" (terraform-providers/null) 2.1.2...
```

```
- Downloading plugin for provider "tls" (terraform-providers/tls) 2.0.1...
```

```
- Downloading plugin for provider "local" (terraform-providers/local) 1.3.0...
```

```
# ...
```

Module References Are Stored in `.terraform`

A hidden directory is built at the root of your project

You can use the `tree` command or `ls -l` to see the contents

Note that local modules are symlinked. This explains why changes to module code are available immediately

```
> tree .terraform

.terraform/
└── modules
    ├── 56e6098163e5b2675d14437a2a723a54
    │   └── mitchellh-terraform-aws-dynamic-keys-37d94b3
    │       ├── main.tf
    │       ├── outputs.tf
    │       ├── README.md
    │       └── variables.tf
    └── modules.json
└── plugins
    └── linux_amd64
        ├── lock.json
        ├── terraform-provider-aws_v1.23.0_x4
        ├── terraform-provider-local_v1.1.0_x4
        ├── terraform-provider-null_v1.0.0_x4
        └── terraform-provider-tls_v1.1.0_x4

4 directories, 10 files
```

```
> terraform apply
aws_instance.web: Refreshing state... [id=i-03724b60e0ff853bc]

# ...

# module.keypair.aws_key_pair.generated will be created
+ resource "aws_key_pair" "generated" {
    + fingerprint = (known after apply)
    + id          = (known after apply)
    + key_name    = "robin-training-env-ant-key"
    + public_key  = (known after apply)
}

# ...

Plan: 5 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes
```

What You Learned

- ✓ Import a module from the Terraform Public Module Registry
- ✓ Use a module from `main.tf`
- ✓ Understand how modules are stored on disk

EXERCISE

Lab 5: Modules

Duration: 20 minutes

Goal: Refactor your existing code into a separate module and connect a module from the Public Registry.

Provisioners

What You'll Learn

- ✓ How to create a connection to our server
- ✓ Specify the key for connecting to the server
- ✓ Provision the server with files and commands

WARNING

The Server We Created is Useless

We created a server without any running code.

No useful services are running on it.

In this section, we'll see how to provision or instance.

```
resource "aws_instance" "web" {
    # . . .

    connection = {
        type      = "ssh"
        user      = "ubuntu"
        host      = self.public_ip
        private_key = module.keypair.private_key_pem
    }
}
```

```
variable "identity" { }

module "keypair" {
    source = "mitchellh/dynamic-keys/aws"
    path   = "${path.root}/keys"
    name   = "${var.identity}-key"
}
```

```
resource "aws_instance" "web" {  
#  .  .  .  
  
  key_name = "${var.identity}-key"  
#  .  .  .  
}
```

```
resource "aws_instance" "web" {
    # . . .

    provisioner "file" {
        source      = "assets"
        destination = "/tmp/"
    }

}
```

```
resource "aws_instance" "web" {
    # . . .

    provisioner "remote-exec" {
        inline = [
            "sudo sh /tmp/assets/setup-web.sh"
        ]
    }
}
```

Force Provisioning With the `taint` Command

If we run `apply` after adding provisioners, it won't change the server.

Force it to be destroyed and recreated with the `taint` command:

```
terraform taint aws_instance.web
```

If you're unsure of the correct ID for the resource you want to taint, you can use `terraform show`:

```
> terraform show  
aws_instance.web:  
  id = i-07842f0a26236fb91  
  ami = ami-0def3275  
  . . .
```

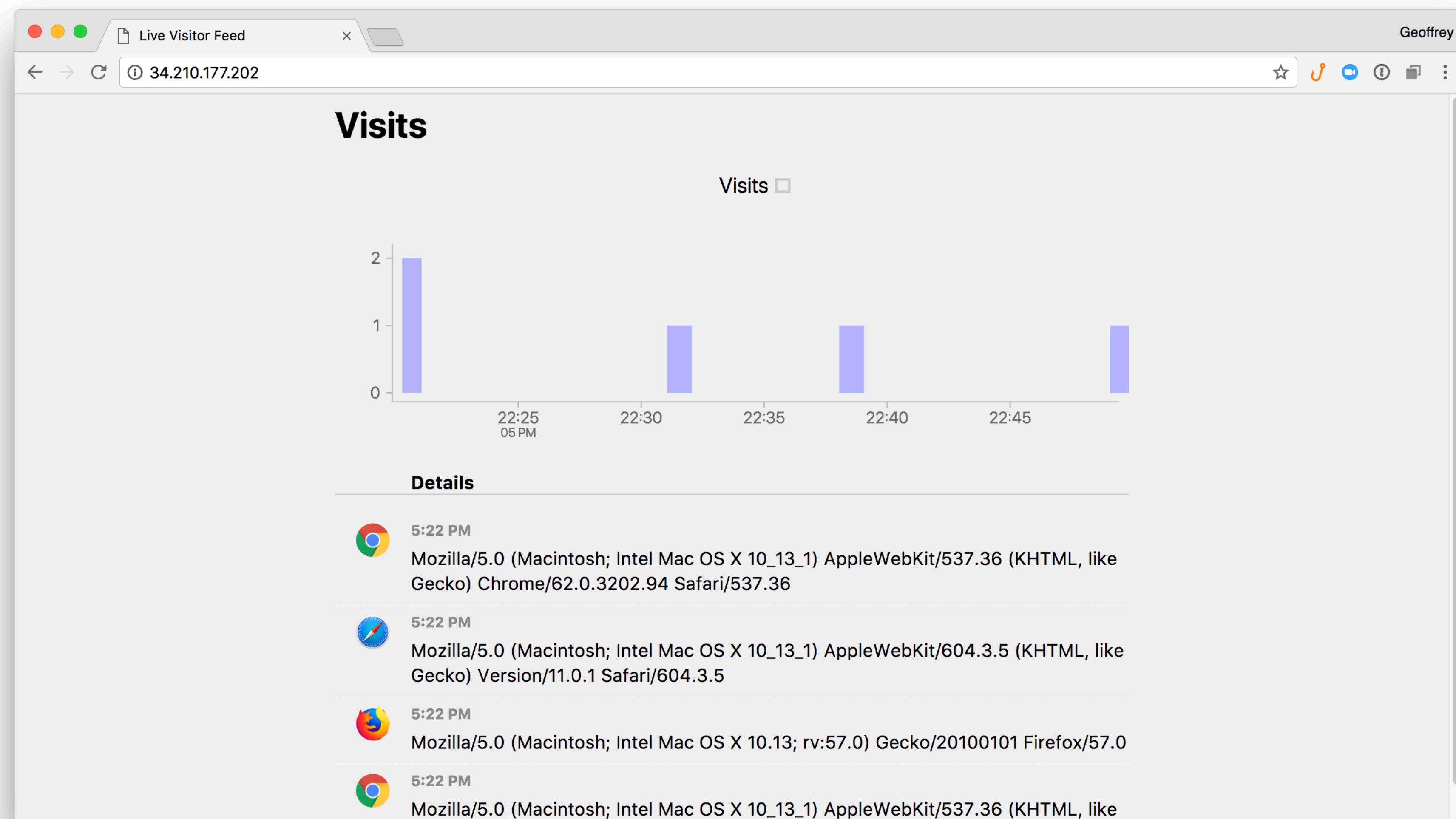
```
> terraform taint aws_instance.web  
Resource instance aws_instance.web has been marked as  
tainted.
```

```
> terraform apply
# ...
Plan: 1 to add, 0 to change, 1 to destroy.
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

# ...

web: Provisioning with 'remote-exec'...
web (remote-exec): Connecting to remote host via SSH...
web (remote-exec):   Host: 35.162.171.124
web (remote-exec):   User: ubuntu
web (remote-exec):   Password: false
web (remote-exec):   Private key: true
web (remote-exec):   SSH Agent: false
web (remote-exec): Connected!
```



Provisioning Best Practices

Good

Automate the creation of infrastructure and initialization of instances with `user_data` or AWS `cloud-init`.

Better

Use `remote-exec` provisioner on a base AMI to run a few commands on instance creation.

Best

Build AMIs with Packer so that minimal configuration is needed.

What You Learned

- ☑ How to create a connection to our server
- ☑ Specify the key for connecting to the server
- ☑ Provision the server with files and commands

EXERCISE

Lab 6: Provisioners

Duration: 10 minutes

Goal: Provisioners allow us to create instances that are ready to use at runtime. Use your SSH key from the last lab to connect to and install our web app on a new instance.

Useful Providers

```
resource "random_id" "server" {
    keepers {
        # Generate a new id each time we switch to a new AMI
        ami_id = var.ami_id
    }
    byte_length = 4
}

output "id" {
    value = random_id.server.hex
}
```

```
variable "animal" {}

data "template_file" "setup" {
    template = file("templates/setup.sh")

    vars {
        animal = var.animal
    }
}

output "template" {
    value = data.template_file.setup.rendered
}
```

```
#!/usr/bin/env sh  
  
echo "Deploying ${animal}"
```

Theory Behind Terraform

What You'll Learn

- ☑ Understand infrastructure as code
- ☑ Understand how graph theory makes Terraform more effective
- ☑ Generate a graph of your infrastructure

Infrastructure as Code

Provide a codified workflow to create infrastructure

Expose a workflow for managing updates to existing infrastructure

Integrate with application code workflows (Git, SCM, Code Review)

Provide modular, sharable components for separation of concerns

Infrastructure as Code (Terraform)

Human-readable configuration (HCL) is designed for human consumption so users can quickly interpret and understand their infrastructure configuration

Terraform can also parse configuration in JSON for machine-generated configurations

Configuration format is very VCS friendly with support for multi-line lists, trailing commas, and auto-formatting

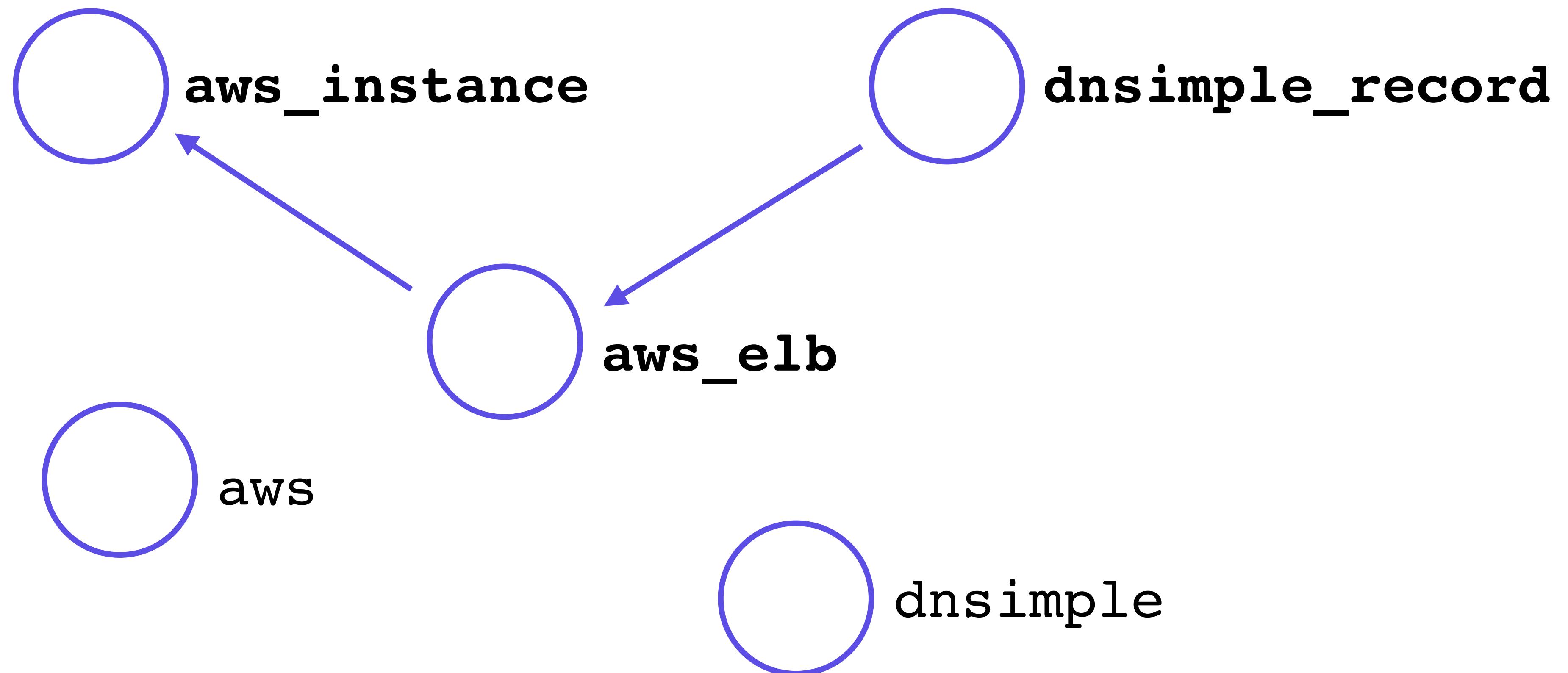
```
resource "aws_instance" "web" {  
    ami                  = "ami-9a562df2"  
-    instance_type = "t2.micro"  
+    instance_type = "m1.small"  
}
```

Infrastructure as Code (Terraform)

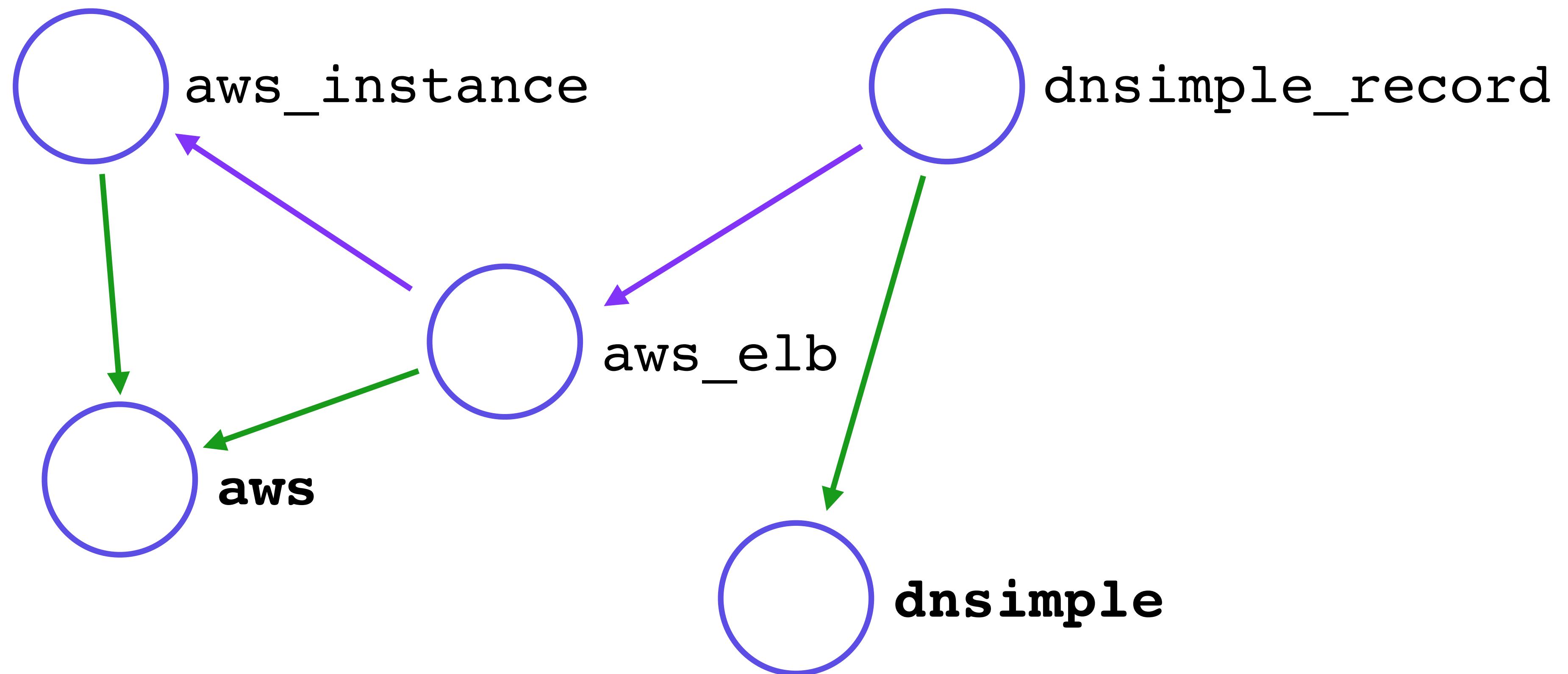
Reference values from other resources, building the implicit dependency graph.

```
resource "aws_instance" "web" {  
    ami              = "ami-0375ca3842950ade6"  
    instance_type   = "t2.micro"  
}  
  
resource "dnsimple_record" "web" {  
    domain  = "hashicorp.com"  
    name    = "web"  
    ttl     = "3600"  
    type    = "A"  
    value   = aws_instance.web.public_ip  
}
```

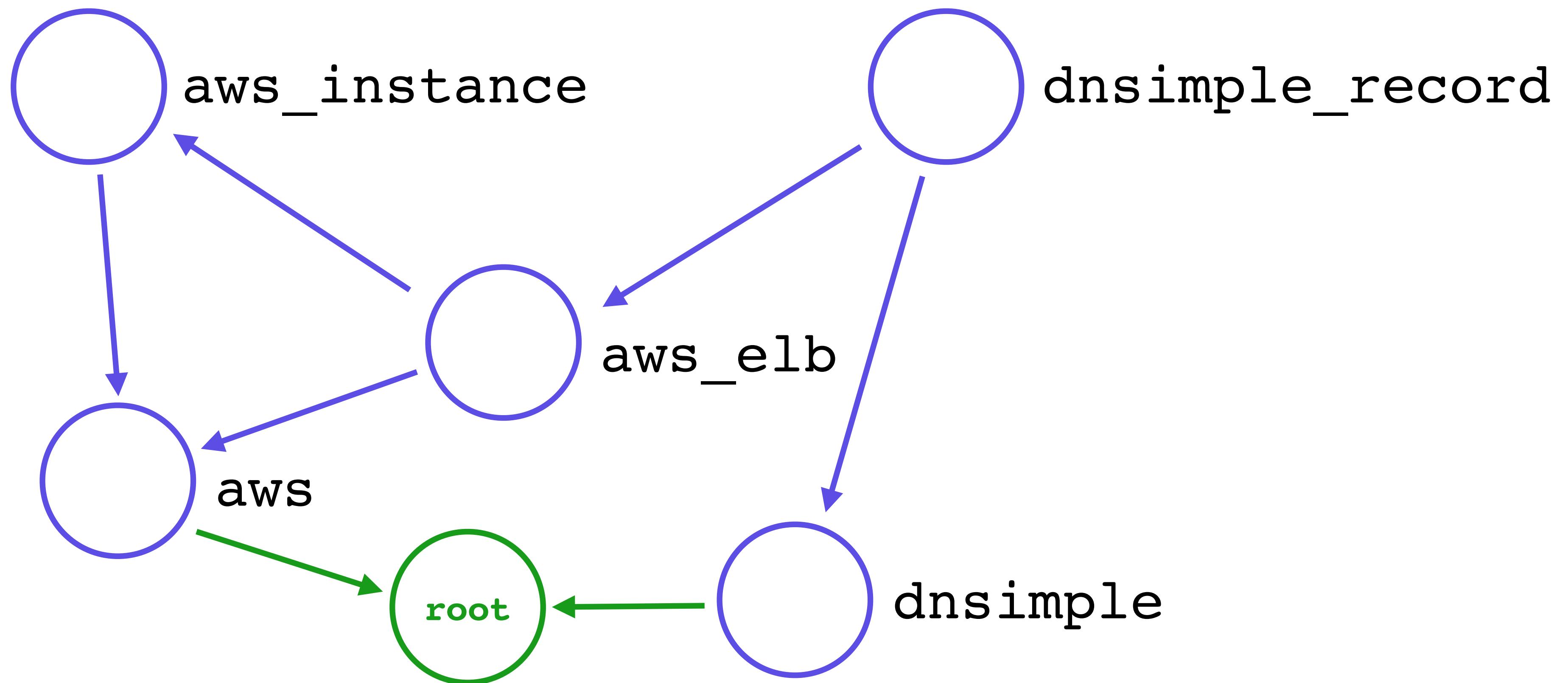
Building the Graph



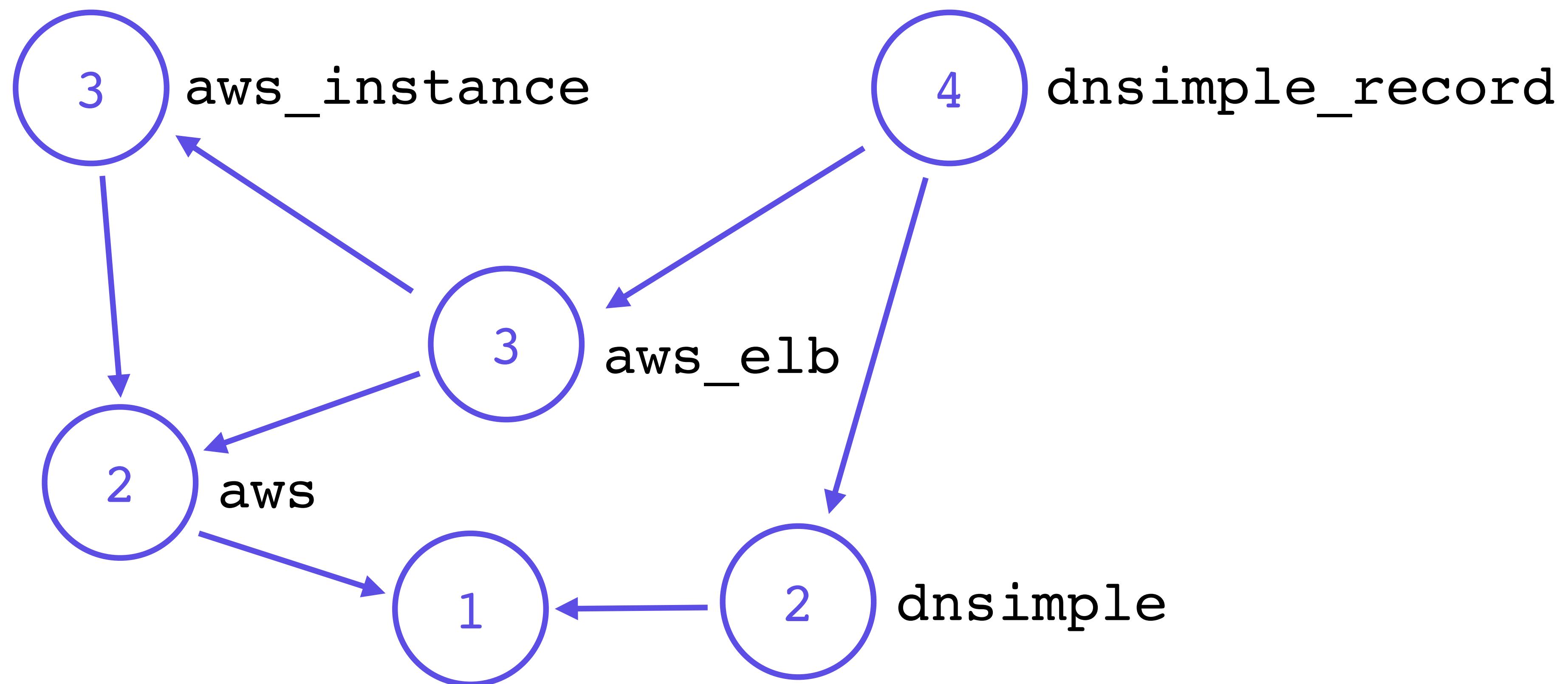
Building the Graph



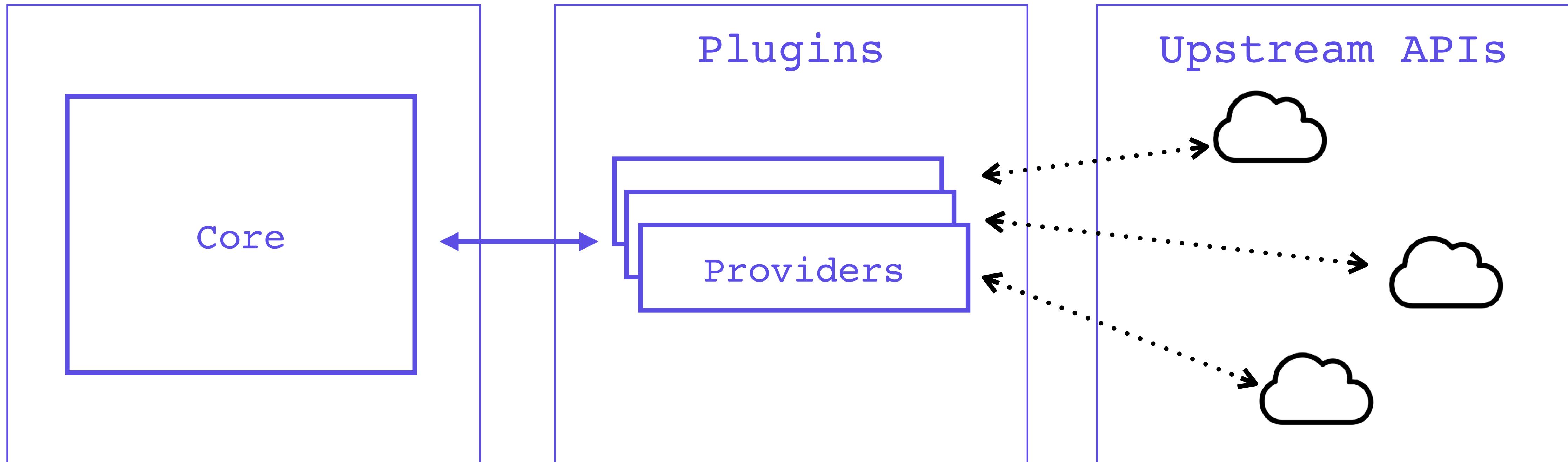
Building the Graph



Walking the Graph



Terraform's Internals: Structure



Terraform's Internals: Core Concepts

Config: Target Reality

State: Current Reality

Diff: {Config - State}

Plan: Presents Diff

Apply: Resolves Diff

Resource Graph

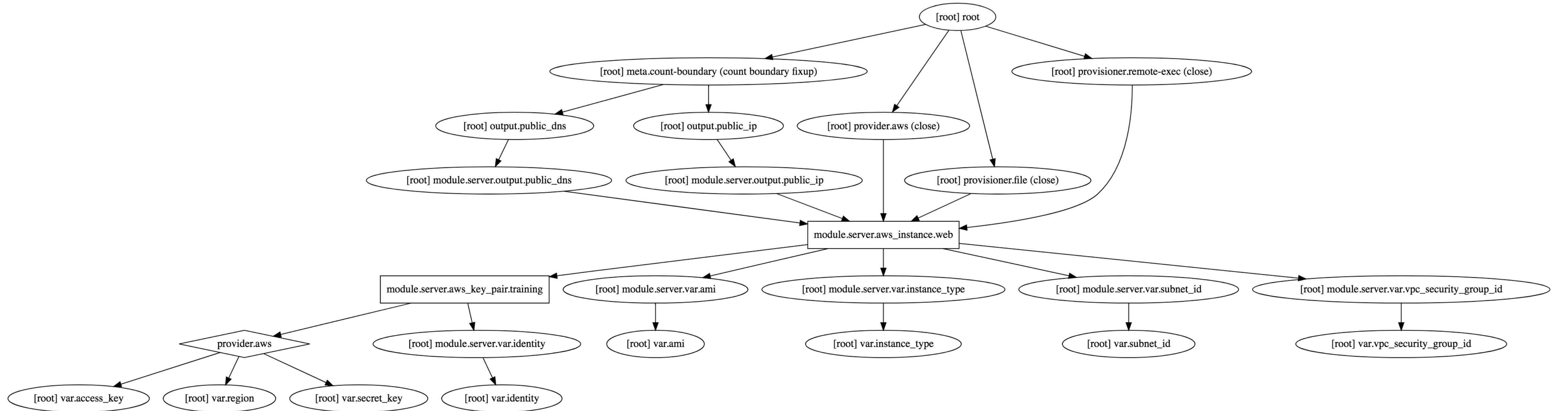
The resource graph is an internal representation of all resources and their dependencies.

A human-readable graph can be generated using the `terraform graph` command.

Can optionally draw cycles (advanced).

```
> terraform graph
digraph {
    compound = "true"
    newrank = "true"
    subgraph "root" {
        "[root] aws_instance.web" [label = "aws_instance.web", shape = "box"]
        "[root] provider.aws" [label = "provider.aws", shape = "diamond"]
        "[root] aws_instance.web" -> "[root] provider.aws"
        "[root] meta.count-boundary (count boundary fixup)" -> "[root] ..."
        "[root] provider.aws (close)" -> "[root] aws_instance.web"
        "[root] root" -> "[root] meta.count-boundary (count boundary fixup)"
        "[root] root" -> "[root] provider.aws (close)"
    }
}
```

terraform graph



Terraform Graph

Useful for visualizing infrastructure and dependencies

Builds upon existing visualization technologies and open formats such as DOT

Can optionally draw cycles (resources that depend on each other in a circle)

EXERCISE

Lab 7: Graphs

Duration: 10 minutes

Goal: Visualize the dependencies in a Terraform configuration with the graph command.

Meta Arguments

What You'll Learn

- ✓ Understand meta arguments
- ✓ Use count to create multiple instances
- ✓ Manage your own metadata

Meta-Arguments

Meta-arguments allow for higher-level control flow and lifecycle management in Terraform

These don't map directly to cloud resources or APIs, but help you control Terraform's actions

Meta-Arguments

Count

The count argument allows for N number of identical resources to be created. This removes the need for iteration with "for" or "while" loops in many cases

Meta-Arguments

Depends On

The `depends_on` argument allows for declaration of explicit dependencies. This is useful where interpolation is not required, but explicit ordering is desired

Meta-Arguments

Provider

The `provider` argument allows for a resource to explicitly use a provider by name or alias. This is most useful when there are multiple providers in a single configuration, such as `aws.west` and `aws.east`

Meta-Arguments

Lifecycle

The `lifecycle` argument allow explicit configuration of resource lifecycle such as preventing destruction or ignoring property changes. This is an advanced option and is not recommended for users who are getting started with Terraform

```
# ...

resource "aws_instance" "web" {
    count          = 2

    ami            = "ami-07669fc90e6e6cc47"
    instance_type = "t2.micro"

# ...
}
```

```
# ...  
  
output "public_ip" {  
    value = aws_instance.web.*.public_ip  
}  
  
output "public_dns" {  
    value = aws_instance.web.*.public_dns  
}
```

EXERCISE

Lab 8: Meta Arguments

Duration: 10 minutes

Goal: Modify the number instances created with count.

```
> terraform apply  
# module.server.aws_instance.web[1] will be created  
+ resource "aws_instance" "web" {  
    + ami:                      "ami-0375ca3842950ade6"  
    + availability_zone:        "<computed>"  
# ...
```

Plan: 1 to add, 0 to change, 0 to destroy.

```
# ...
```

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

```
variable "num_webservers" {
  count  = number
  default = 2
}

# ...

resource "aws_instance" "web" {
  count  = var.num_webservers

  ami              = var.ami
  instance_type   = "t2.micro"

  subnet_id        = var.subnet_id
  vpc_security_group_ids = [var.vpc_security_group_id]

  tags {
    # ...
    "Name" = "web ${count.index+1}/${var.num_webservers}"
  }
}
```

State

What You'll Learn

- ☑ Understand how state is managed
- ☑ Understand where state can be stored
- ☑ Understand Terraform Cloud and Terraform Enterprise

State

Terraform stores the state of your managed infrastructure from the last time Terraform was run.

Terraform uses this state to create plans and make changes to your infrastructure.

It is critical that this state is maintained appropriately so future runs operate as expected.

```
> head terraform.tfstate
{
  "version": 4,
  "terraform_version": "0.12.3",
  "serial": 46,
  "lineage": "b139a29e-9daa-47a9-b00f-33a4462a706b",
  "modules": [
    {
      "path": [
        "root"
      ],
      "outputs": {
        "public_dns": {
          "sensitive": false,
          "type": "list",
        }
      }
    }
  ]
}
```

```
> terraform state list

module.server.aws_instance.web[0]
module.server.aws_instance.web[1]
module.server.aws_key_pair.training

# ...
```

State Resolutions

Configuration	State	Reality	Operation
aws_instance.web			
aws_instance.web	aws_instance.web		
aws_instance.web	aws_instance.web	aws_instance.web	
	aws_instance.web	aws_instance.web	
		aws_instance.web	
aws_instance.web		aws_instance.web	
	aws_instance.web		

State Resolutions

Configuration	State	Reality	Operation
aws_instance.web			create
aws_instance.web	aws_instance.web		create
aws_instance.web	aws_instance.web	aws_instance.web	noop
	aws_instance.web	aws_instance.web	delete
		aws_instance.web	noop
aws_instance.web		aws_instance.web	re-create*
	aws_instance.web		update state

State Import

Terraform only manages previous or imported resources.

Terraform can import state for existing resources. If you have already spun up an instance with the web console, you may need to import it so Terraform knows about it.

This only writes the state file, not the Terraform configuration.

State Import

Critical Thinking

After importing an existing resource, what operation(s) would the terraform plan show?

State Import

Critical Thinking

After importing an existing resource, what operation(s) would the terraform plan show?

Answer

Destroy (state exists, no configuration)

State Locking

If supported, the state backend will "lock" to prevent concurrent modifications which could cause corruption.

Not all backends support locking - Terraform's documentation identifies which backends support this functionality (Terraform Enterprise, Terraform Cloud, S3, GCS, Azure Storage do)

Where is State?

Local State (default)

Stored locally in a JSON format.
(`terraform.tfstate`)

Remote State

Stored on a remote source (Terraform Cloud, Terraform Enterprise, S3, GCS, Azure Storage, etc).

Local State

State is stored locally on one machine

It is generally acceptable for individuals and small teams

No good way to back up or share state

Tends not to scale for larger teams

Requires a more "mono repo" pattern

Remote State

State is on a remote source like S3, Terraform Cloud, Terraform Enterprise, or Consul

Remote storage is responsible for handling merging and locking

Can be queried for information in other Terraform configurations

Removes the risk of losing state to a hard drive crash or other data loss

Sensitive Data in State

State can contain sensitive data depending on the resources used

Sometimes it can contain initial database passwords or other secret data returned by a provider

Some resources support PGP encrypting the values in the state, but this is implemented on a per-resource basis

We recommend never keeping state in source control, for example

Sensitive Data in State

Local state (JSON) is not encrypted

Remote state encryption is backend-specific

State is only held in memory when remote state is used

Example: S3 bucket can be encrypted + IAM + TLS connection

Example: TFE encrypted in transit and rest + full audit log

```
resource "aws_iam_access_key" "lb" {  
    user      = aws_iam_user.lb.name  
    pgp_key  = "keybase:sethvargo"  
}
```

Terraform Cloud

Terraform Ecosystem

Terraform CLI	Terraform Cloud	Terraform Enterprise
Command line tool	SaaS app	On premise/in your cloud
Infrastructure as Code	Free for individuals and teams	Workspace & team management
Open Source	Remote state	Governance and policy features
150+ Providers	Version Control System Integration	Private module registry
Module Registry		

The screenshot shows the Terraform Enterprise web interface. The title bar reads "Terraform Enterprise | geoffrey". The address bar shows a secure connection to "https://atlas.hashicorp.com/app/geoffrey-org/training-lab-dev/states". The top navigation bar includes a user icon labeled "Geoffrey", a search bar, and links for "Documentation" and "Status". Below the navigation is a blue header bar with the "geoffrey-org" workspace name and a "Workspaces" dropdown. The main content area is titled "training-lab-dev" and features a "Queue Plan" button. A horizontal menu bar below the title includes "Latest Run", "Runs", "States" (which is underlined), "Variables", "Settings", "Integrations", and "Access". The main body displays a list of five recent state changes:

New state	Saved by	Time ago
#sv-Q8jFw8VkFGH6uVL7	geoffrey from Terraform	3 days ago
#sv-EiZEpZjAnttcBGh	geoffrey from Terraform	3 days ago
#sv-Lzt4kV8LbNa68fXV	geoffrey from Terraform	4 days ago
#sv-kbJqHaEsaV6QVbgG	geoffrey from Terraform	4 days ago
#sv-as5VdWx1TcL7N4UH	geoffrey from Terraform	4 days ago

At the bottom of the list, there is a URL: <https://atlas.hashicorp.com/app/geoffrey-org/training-lab-dev/states/sv-kbJqHaEsaV6QVbgG>.

Transitioning from Local to Remote

Transitioning is a one-time operation

After configured, Terraform will no longer store local state

The `terraform` stanza declares the configuration alongside resources

```
● ● ●
```

```
terraform {
  backend "remote" {
    hostname = "app.terraform.io"
    organization = "example-company"

# Single workspace:
  workspaces {
    name = "my-app-prod"
  }

# Or use multiple workspaces:
#   workspaces {
#     prefix = "my-app-"
#   }
}
```

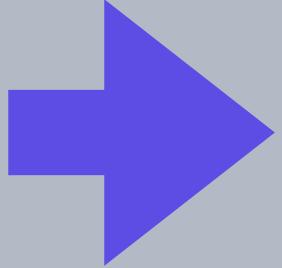
```
data "terraform_remote_state" "vpc" {
    backend = "remote"

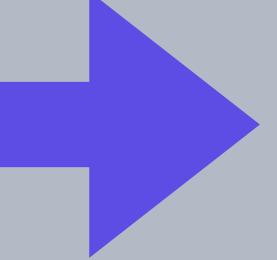
    config {
        organization = "example-org"
        hostname = "app.terraform.io"
    }

    workspaces {
        name = "example-workspace"
    }
}

resource "aws_instance" "example" {
    # ...
    subnet_id = data.terraform_remote_state.vpc.subnet_id
}
```

```
credentials "app.terraform.io" {  
    token = "<TOKEN>"  
}
```





```
data "terraform_remote_state" "vpc" {
  backend = "remote"

  config {
    organization = "example-org"
    hostname = "app.terraform.io"
    access_token = "xxxxxx"
  }
}

resource "aws_instance" "example" {
  # ...
  subnet_id = data.terraform_remote_state.vpc.subnet_id
}
```

Backend Initialization

Remote State backends must be initialized before use

Initialization prepares the remote backend for storage

Initialization will prompt for any missing values and cache them locally in `.terraform` (which should be ignored from source)

Initialization will also optionally migrate state

```
> terraform init
```

Initializing the backend...

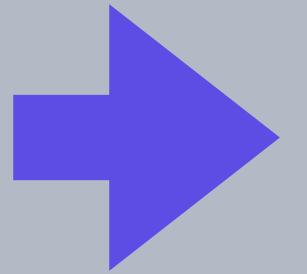
Acquiring state lock. This may take a few moments...

Do you want to copy existing state to the new backend?

Pre-existing state was found while migrating the previous "local" backend to the newly configured "remote" backend. No existing state was found in the newly configured "remote" backend.

Do you want to copy this state to the new "remote" backend? Enter "yes" to copy and "no" to start with an empty state.

Enter a value:



```
Successfully configured the backend "remote"! Terraform  
will automatically use this backend unless the backend  
configuration changes.
```

The screenshot shows the Terraform Enterprise web interface. The browser title bar reads "Terraform Enterprise | geoffrey". The address bar shows a secure connection to "https://atlas.hashicorp.com/app/geoffrey-org/tmp-terraform/states". The top navigation bar includes a logo, the workspace name "geoffrey-org", a "Workspaces" dropdown, and links for "Documentation" and "Status". A user profile icon is also present.

The main content area is titled "tmp-terraform". Below it is a navigation bar with tabs: "Latest Run", "Runs", "States", "Variables", "Settings", "Integrations", and "Access". The "States" tab is currently selected, indicated by a blue underline.

The main content area displays three recent state saves:

- New state #sv-UZsnuGWa6ndRBUX2** - saved by [geoffrey](#) from Terraform a few seconds ago
- New state #sv-jSLo82cs21nzdQ55** - saved by [geoffrey](#) from Terraform 2 minutes ago
- New state #sv-gjjKLvuN3sjn54Sc** - saved by [geoffrey](#) from Terraform 2 minutes ago

At the bottom left is the HashiCorp logo. At the bottom right are links for "Terms", "Privacy", "Security", and "© 2017 HashiCorp, Inc."

Remote State

Terraform will no longer write to the `.tfstate` file

Everything else about your local workflow remains the same

```
> terraform apply  
aws_instance.web.0: Refreshing state... (ID:  
i-09ac7e80c7d33b461)
```

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

Outputs:

```
public_dns = [  
    ec2-54-174-40-55.compute-1.amazonaws.com,  
]  
public_ip = [  
    54.174.40.55  
]
```

EXERCISE

Lab 9: Data Source (Optional)

Duration: 10 minutes

Goal: Use a data source to select the AMI used for your web server

Destroy

What You'll Learn

- Destroy infrastructure that Terraform has created

Command: terraform destroy

Destroys running infrastructure

Does not touch infrastructure not managed by
Terraform

```
> terraform destroy  
aws_instance.web: Refreshing state...  
[id=i-0ec4ffe55abd300a5]
```

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:

- destroy

Terraform will perform the following actions:

```
# aws_instance.web will be destroyed  
- resource "aws_instance" "web" {  
# ...
```

EXERCISE

Lab 10: Destroy

Duration: 5 minutes

Goal: Use the destroy command to destroy your infrastructure

What You'll Learn

- ✓ Create a repository at GitHub
- ✓ Push your code to GitHub
- ✓ Create an organization at Terraform Cloud
- ✓ Connect to GitHub via OAuth
- ✓ Add a workspace that references a GitHub repo
- ✓ Commit changes and create infrastructure

Terraform Cloud

Self-service workflow

Collaboration for teams

Powerful ACLs & auditing

Runs Terraform for you

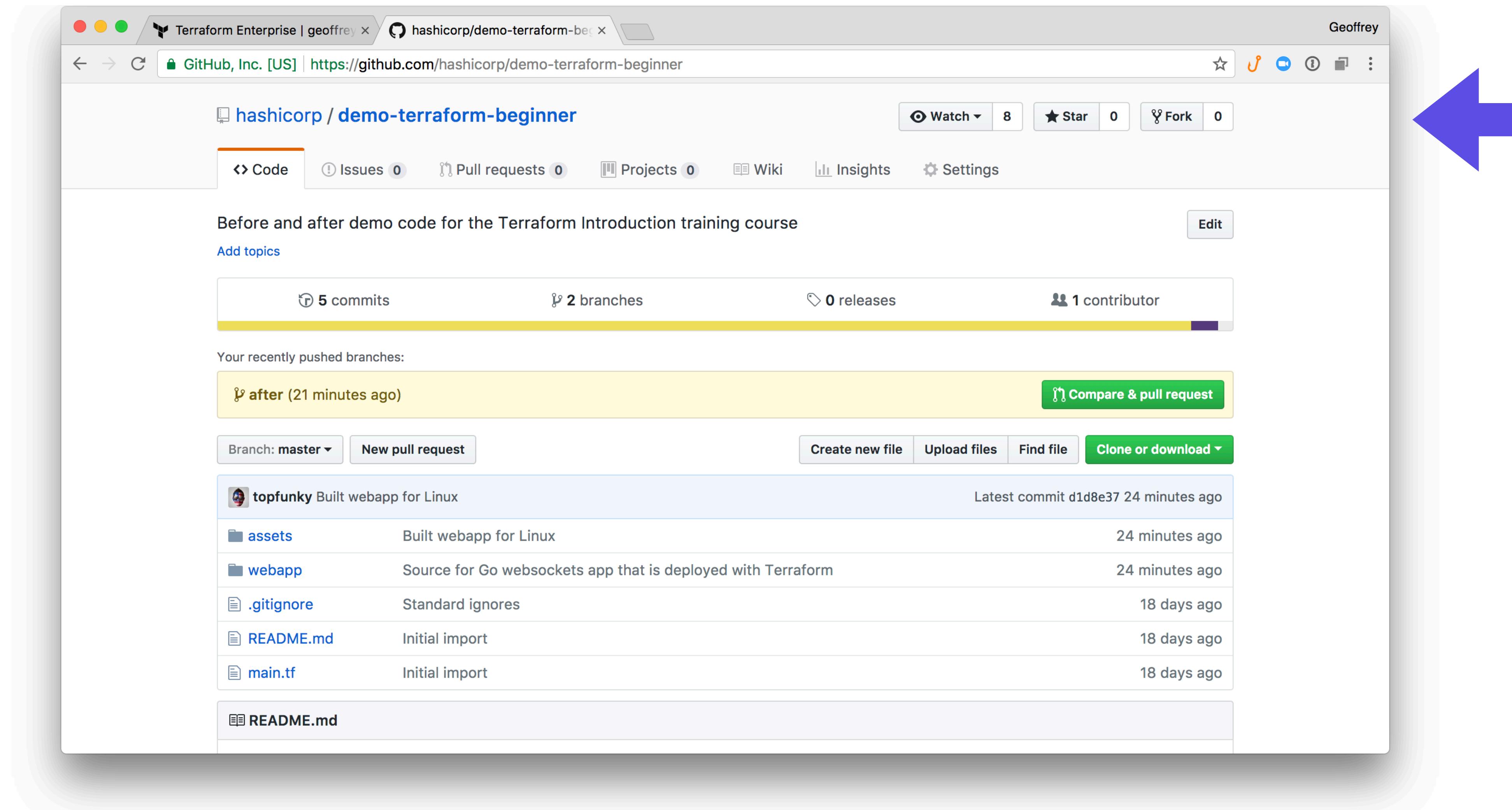
Control Terraform version

Prevent concurrent
changes

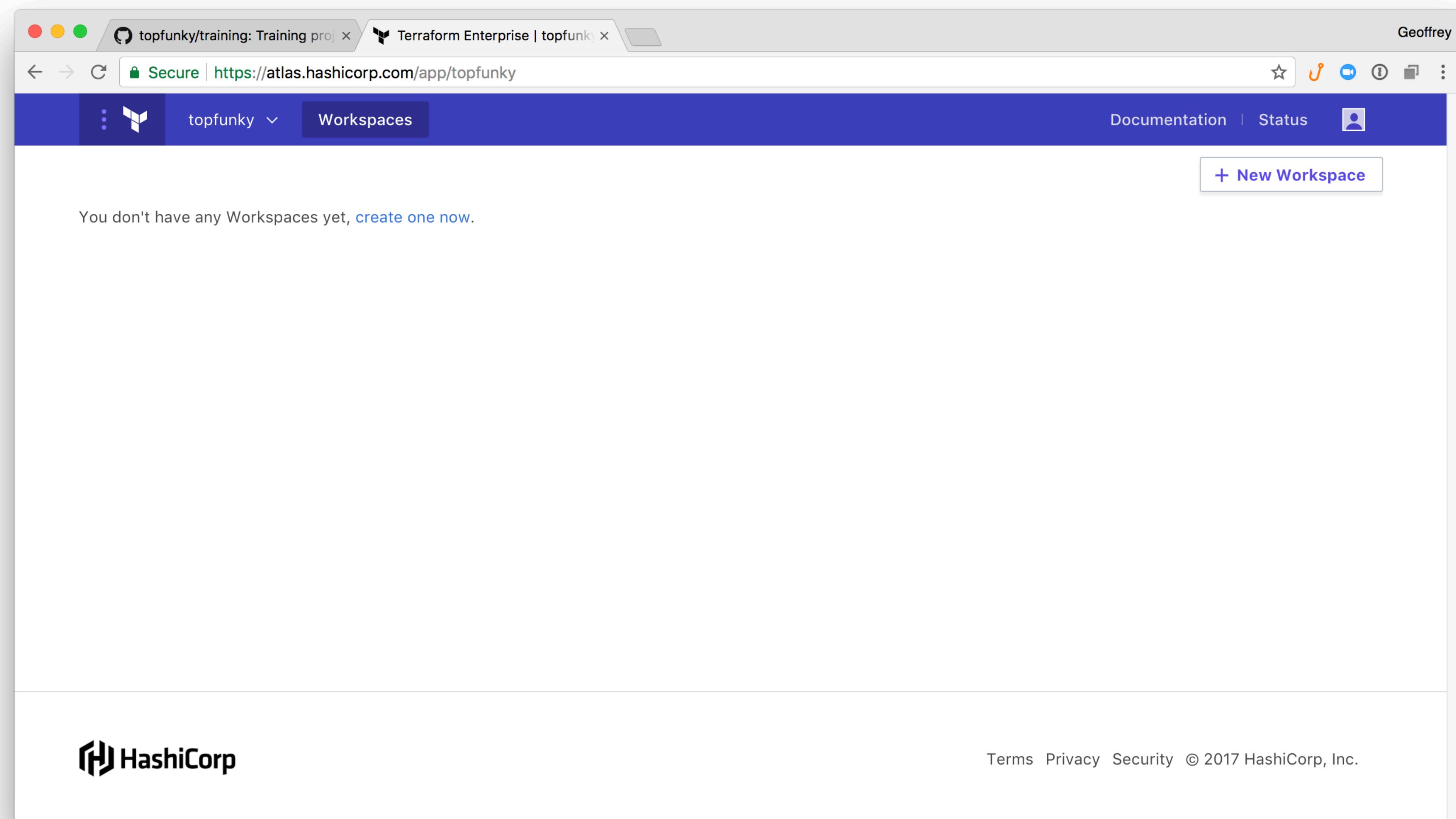
Integrate with SCM

View history of changes

Enforce policies with
Sentinel



The screenshot shows a web browser window titled "Terraform Enterprise | New Org". The URL in the address bar is "Secure | https://atlas.hashicorp.com/app/organizations/new". The page has a blue header with the HashiCorp logo, the organization name "geoffrey-org", and a "Workspaces" button. On the right side of the header are links for "Documentation" and "Status" and a user profile icon. The main content area is titled "New Organization". It contains two input fields: "ORGANIZATION USERNAME" with the value "topfunky" and "EMAIL ADDRESS" with the value "geoffrey+tfe@hashicorp.com". Below the email field is a note: "The organization email is used for any future notifications, such as billing, and the organization avatar, via [gravatar.com](#)". At the bottom left is a blue "Create organization" button. The HashiCorp logo is at the bottom left, and a footer at the bottom right includes links for Terms, Privacy, Security, and a copyright notice: "© 2017 HashiCorp, Inc."



The screenshot shows a web browser window for Terraform Enterprise. The title bar includes tabs for 'topfunky/training: Training proj' and 'Terraform Enterprise | topfunky'. The address bar is secure, showing the URL <https://atlas.hashicorp.com/app/topfunky/workspaces/new>. The user 'Geoffrey' is logged in. The main content is titled 'Create a new Workspace' and states that the workspace will be created under the organization 'topfunky'. A 'WORKSPACE NAME' input field contains 'workspace-name'. Below it, a note says the name must be unique and can include dashes and alphanumeric characters, with a link to 'naming workspaces'. Under 'VCS CONNECTION', it says there are no VCS connections and provides a link to the 'Workspace API'. At the bottom are 'Create Workspace' and 'Cancel' buttons. The footer shows the URL <https://atlas.hashicorp.com/app/topfunky/settings/oauth/add>.

Create a new Workspace

This workspace will be created under the current organization, **topfunky**.

WORKSPACE NAME

workspace-name

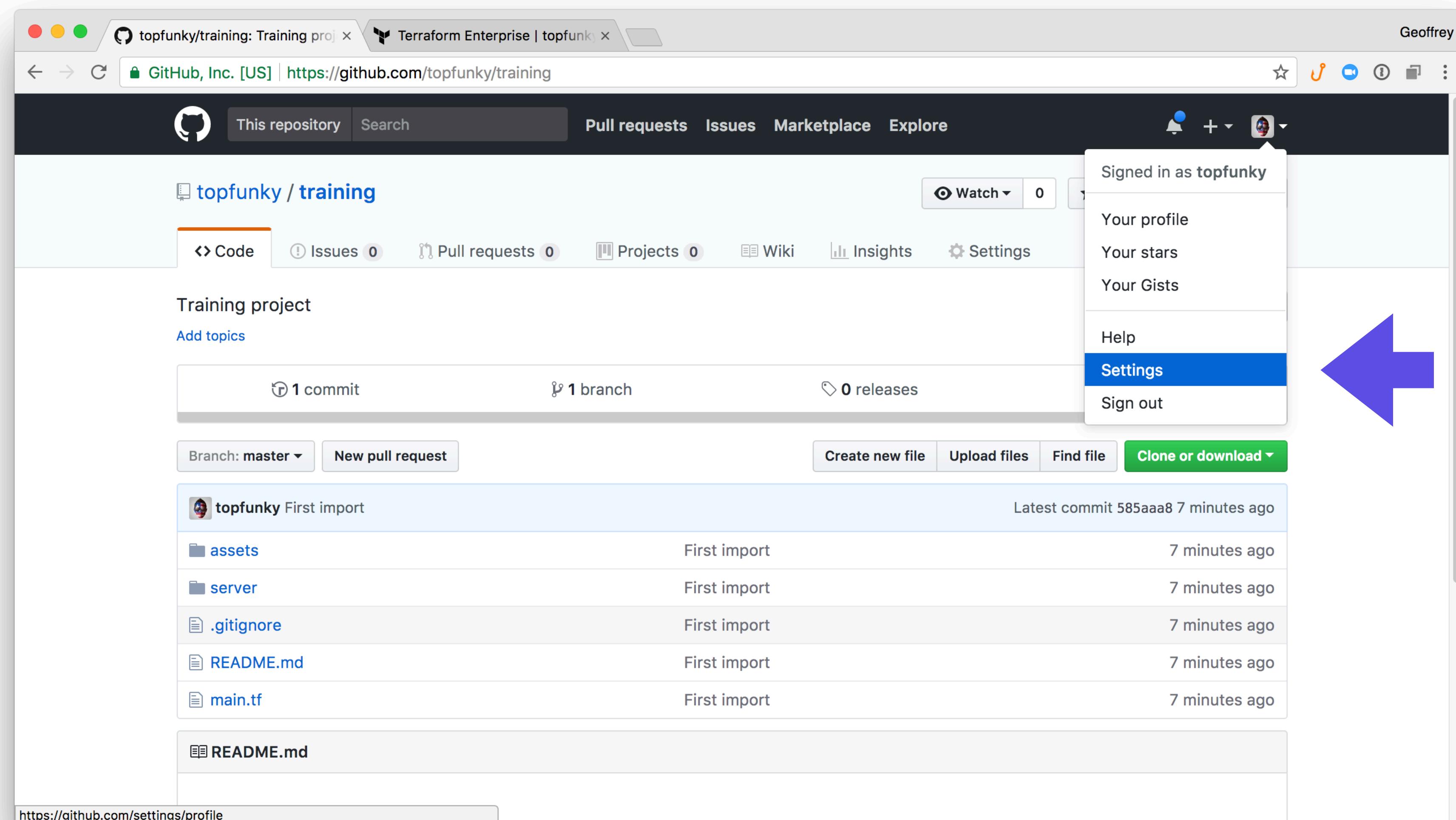
The name of your workspace is unique and used in tools, routing, and UI. Dashes and alphanumeric characters are permitted. Learn more about [naming workspaces](#).

VCS CONNECTION

You have no VCS connections on this organization. Add a VCS connection. To create a workspace without a VCS connection use the [Workspace API](#).

Create Workspace **Cancel**

<https://atlas.hashicorp.com/app/topfunky/settings/oauth/add>



Your Profile Terraform Enterprise | topfunk

GitHub, Inc. [US] | https://github.com/settings/profile

Search GitHub Pull requests Issues Marketplace Explore

Geoffrey

Personal settings

Profile

Account

Emails

Notifications

Billing

SSH and GPG keys

Security

Blocked users

Repositories

Organizations

Saved replies

Applications

Developer settings

Public profile

Name

Geoffrey Grosenbach

Profile picture

Upload new picture

Public email

boss@topfunky.com

You can manage verified email addresses in your [email settings](#).

Bio

Entrepreneur, developer, designer, teacher, athlete. Currently:
Director of Training Programs at @hashicorp. Previously:
PeepCode, Pluralsight

You can @mention other users and organizations to link to them.

URL

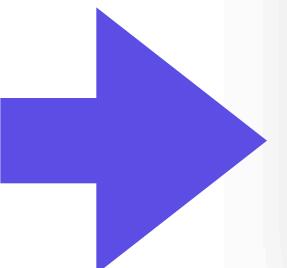
<https://topfunky.com>

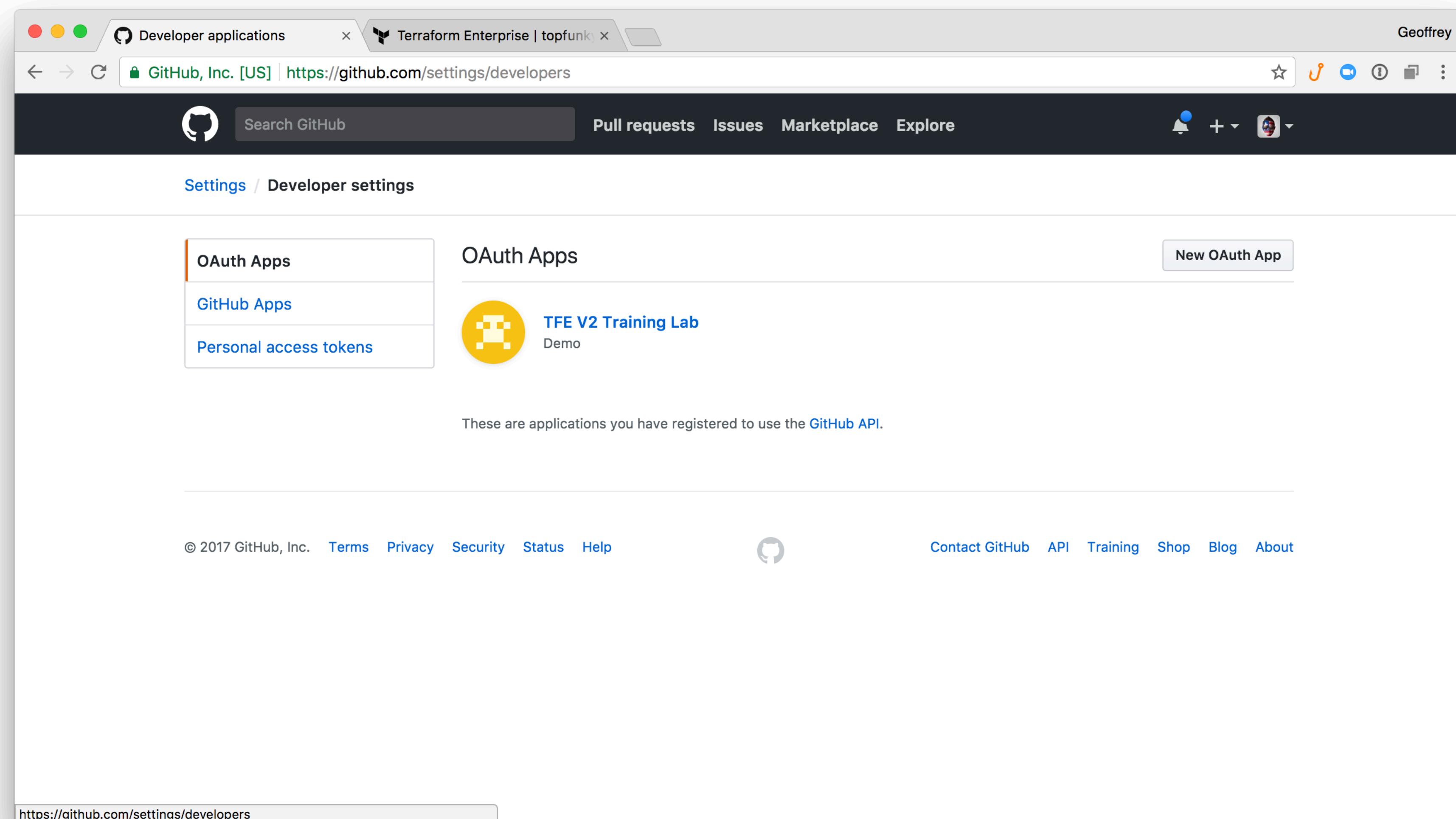
Company

@hashicorp

You can @mention your company's GitHub organization to link it.

<https://github.com/settings/developers>





A screenshot of a Mac OS X desktop showing a web browser window for GitHub. The window title is "Developer applications" and the tab title is "Terraform Enterprise | topfunk". The URL in the address bar is "GitHub, Inc. [US] | https://github.com/settings/developers". The page displays the "Developer settings" section under "Settings". On the left, a sidebar menu has "OAuth Apps" selected, with other options "GitHub Apps" and "Personal access tokens". The main content area is titled "OAuth Apps" and shows a single registered application: "TFE V2 Training Lab" (Demo). A "New OAuth App" button is located in the top right of this section. A large blue arrow points from the right side of the image towards the "New OAuth App" button.

Geoffrey

Developer applications

Terraform Enterprise | topfunk

GitHub, Inc. [US] | https://github.com/settings/developers

Search GitHub

Pull requests Issues Marketplace Explore

Settings / Developer settings

New OAuth App

OAuth Apps

GitHub Apps

Personal access tokens

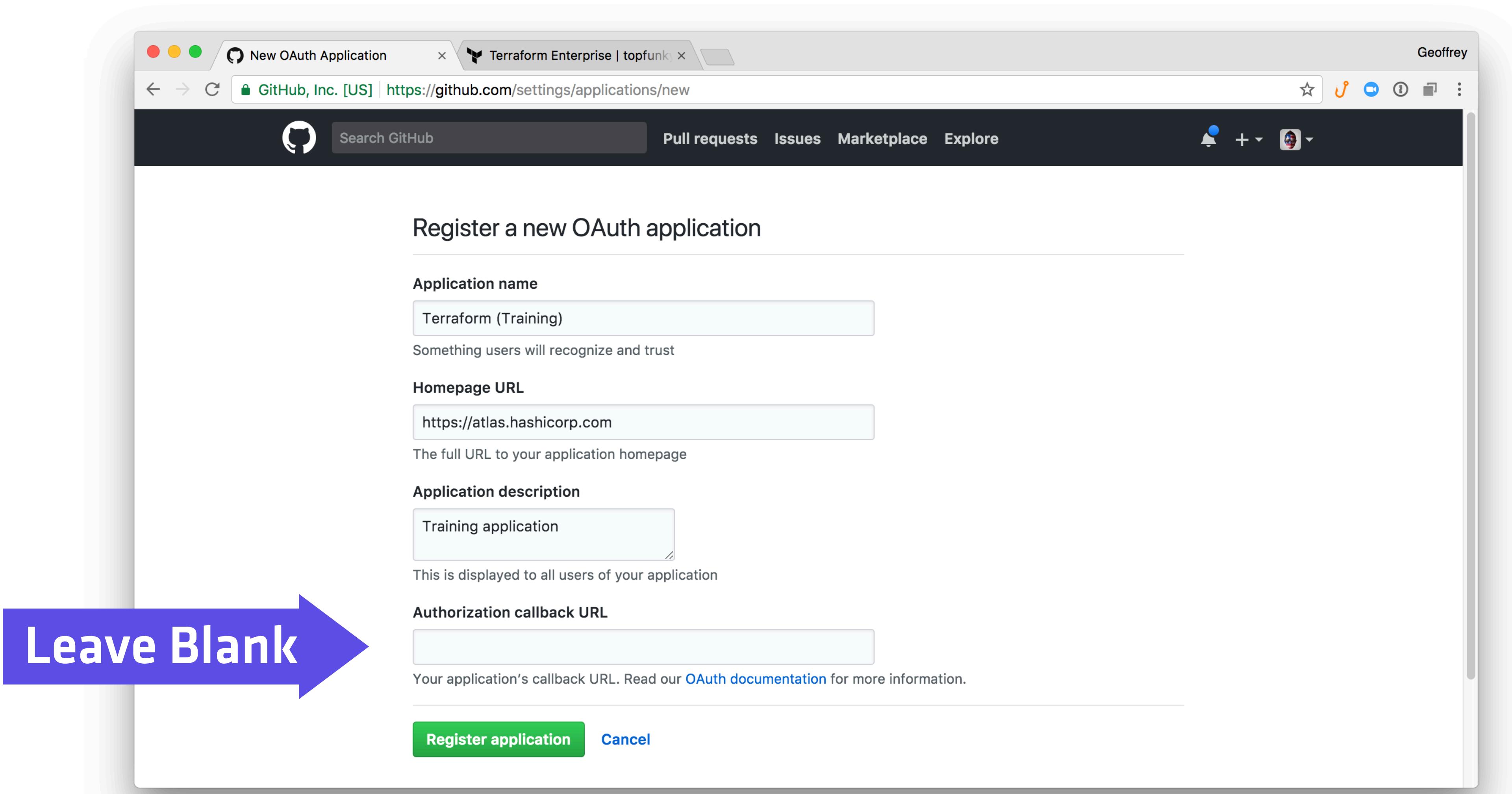
TFE V2 Training Lab
Demo

These are applications you have registered to use the GitHub API.

© 2017 GitHub, Inc. Terms Privacy Security Status Help

Contact GitHub API Training Shop Blog About

https://github.com/settings/developers



New OAuth Application Terraform Enterprise | topfunk

GitHub, Inc. [US] | https://github.com/settings/applications/new

Geoffrey

Search GitHub Pull requests Issues Marketplace Explore

Register a new OAuth application

Application name

Terraform (Training)

Something users will recognize and trust

Homepage URL

https://atlas.hashicorp.com

The full URL to your application homepage

Application description

Training application

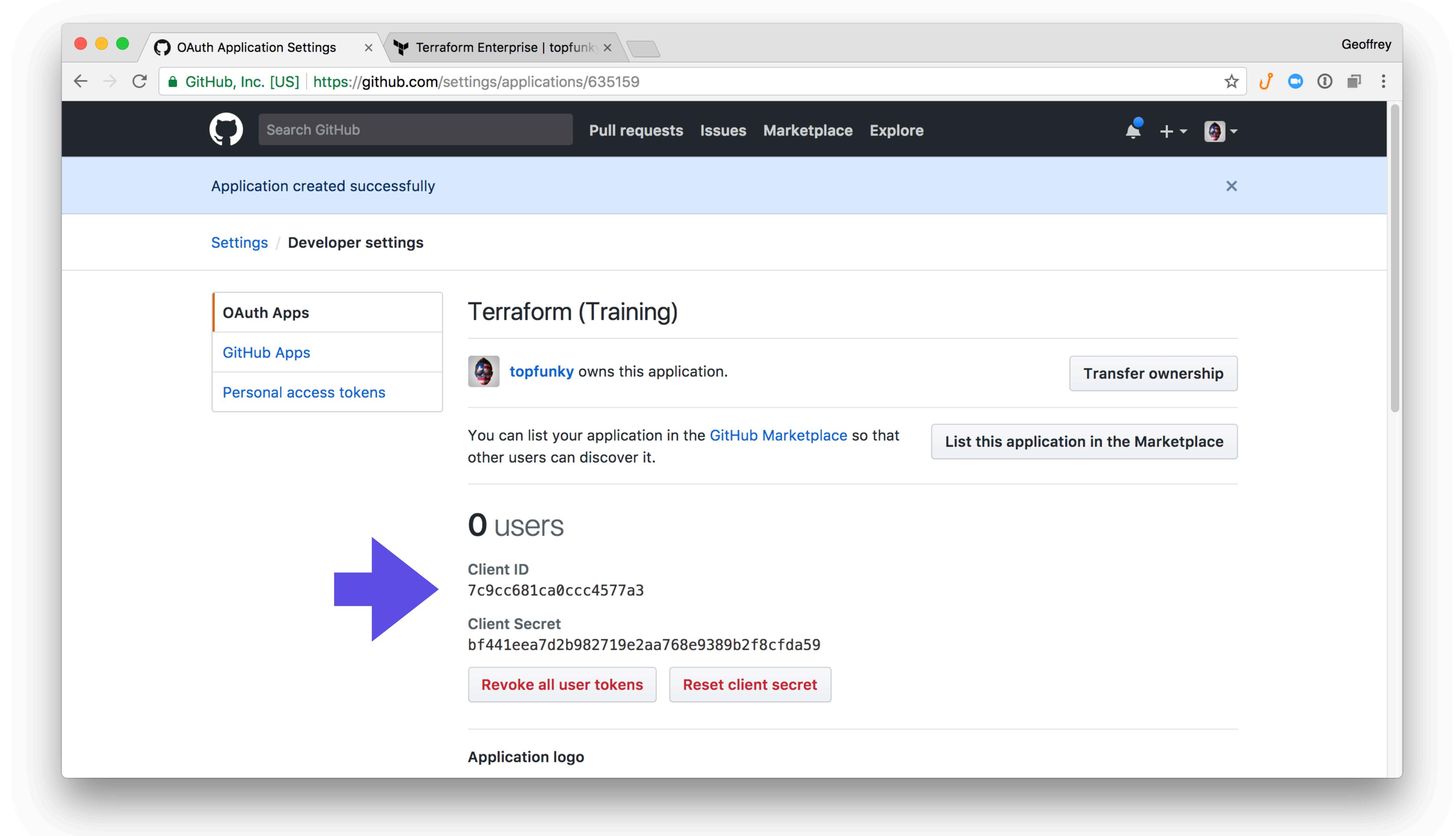
This is displayed to all users of your application

Authorization callback URL

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Register application Cancel

Leave Blank



The screenshot shows the 'Add OAuth Client' page in Terraform Enterprise. The left sidebar lists organization settings like Profile, Teams, OAuth Configuration (selected), API Token, User Sessions, Manage SSH Keys, and Sentinel Policy. The main content area is titled 'Add OAuth Client' and explains the purpose of connecting workspaces to VCS providers. It shows a dropdown for 'VERSION CONTROL SYSTEM (VCS) PROVIDER' set to 'GitHub'. Below it are fields for 'HTTP URL' (https://github.com) and 'API URL' (https://api.github.com). The 'CLIENT ID' field contains '7c9cc681ca0ccc4577a3' and a note to register a new OAuth application on GitHub.com. The 'CLIENT SECRET' field contains 'bf441eea7d2b982719e2aa768e9389b2f8cfda59'.

Geoffrey

Secure | https://atlas.hashicorp.com/app/topfunky/settings/oauth/add

topfunky Workspaces Documentation Status

ORGANIZATION SETTINGS

topfunky ✓

Profile Teams OAuth Configuration API Token User Sessions Manage SSH Keys Sentinel Policy

Add OAuth Client

To connect workspaces to git repositories containing Terraform configurations, Terraform Enterprise needs access to your version control system (VCS) provider. Use this page to configure OAuth authentication with your VCS provider. For more information, please see the Terraform Enterprise documentation on [Configuring Version Control Access](#).

VERSION CONTROL SYSTEM (VCS) PROVIDER

GitHub

HTTP URL

https://github.com

API URL

https://api.github.com

CLIENT ID

7c9cc681ca0ccc4577a3

Register a new OAuth application on [GitHub.com](#) to generate the Client ID and Client Secret.

CLIENT SECRET

bf441eea7d2b982719e2aa768e9389b2f8cfda59

The screenshot shows the 'OAuth Application Settings' page for the organization 'topfunky' in Terraform Enterprise. The left sidebar lists organization settings like Profile, Teams, OAuth Configuration (which is selected), API Token, User Sessions, Manage SSH Keys, and Sentinel Policy. The main content area displays an 'OAuth Clients' section for GitHub. It shows the GitHub logo, callback URL (<https://atlas.hashicorp.com/auth/94dc3315-5f10-42af-8e04-7d95b17dcd8a/callback>), HTTP URL (<https://github.com>), API URL (<https://api.github.com>), and creation date (Dec 12, 2017 17:22:54PM). A large blue arrow points from the right towards the 'Delete Client' button at the bottom right of the client card.

OAuth Clients

Client Type	Callback URL	HTTP URL	API URL	Created
GitHub	https://atlas.hashicorp.com/auth/94dc3315-5f10-42af-8e04-7d95b17dcd8a/callback	https://github.com	https://api.github.com	Dec 12, 2017 17:22:54PM

Connect to GitHub
Connecting to GitHub will take your GitHub user through the OAuth flow to create an authorization token for access to all repositories for this organization. This means that your currently logged in GitHub user token will be used for all GitHub API interactions by any Terraform Enterprise user anywhere within the scope of **topfunky**.

[Connect organization topfunk](#)

[Delete Client](#)

OAuth Application Settings Terraform Enterprise | topfunk Geoffrey

GitHub, Inc. [US] | <https://github.com/settings/applications/635159>

 [Upload new logo](#)
You can also drag and drop a picture from your computer.

Badge background color

The hex value of the badge background color.


Application name

Something users will recognize and trust

Homepage URL

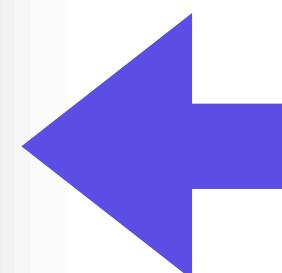
The full URL to your application homepage

Application description

This is displayed to all users of your application

Authorization callback URL

Your application's callback URL. Read our [OAuth documentation](#) for more information.



The screenshot shows the 'OAuth Application Settings' page for the organization 'topfunky' in the Terraform Enterprise interface. The left sidebar lists organization settings like Profile, Teams, OAuth Configuration (which is selected), API Token, User Sessions, Manage SSH Keys, and Sentinel Policy. The main content area displays the 'OAuth Clients' section for GitHub. It shows the GitHub logo, the client name 'GitHub', its callback URL (<https://atlas.hashicorp.com/auth/94dc3315-5f10-42af-8e04-7d95b17dcd8a/callback>), and its HTTP URL (<https://github.com>). The client was created on Dec 12, 2017 at 17:22:54PM. A 'Connect to GitHub' button is present, along with a note explaining the connection process. At the bottom, there are 'Delete Client' and 'Connect organization topfunky' buttons.

Geoffrey

Secure | https://atlas.hashicorp.com/app/topfunky/settings/oauth

topfunky Workspaces Documentation Status

ORGANIZATION SETTINGS

topfunky ✓

Profile Teams OAuth Configuration API Token User Sessions Manage SSH Keys Sentinel Policy

OAuth Clients

GitHub

Callback URL
<https://atlas.hashicorp.com/auth/94dc3315-5f10-42af-8e04-7d95b17dcd8a/callback>

HTTP URL
<https://github.com>

API URL
<https://api.github.com>

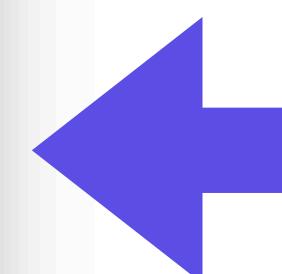
Created
Dec 12, 2017 17:22:54PM

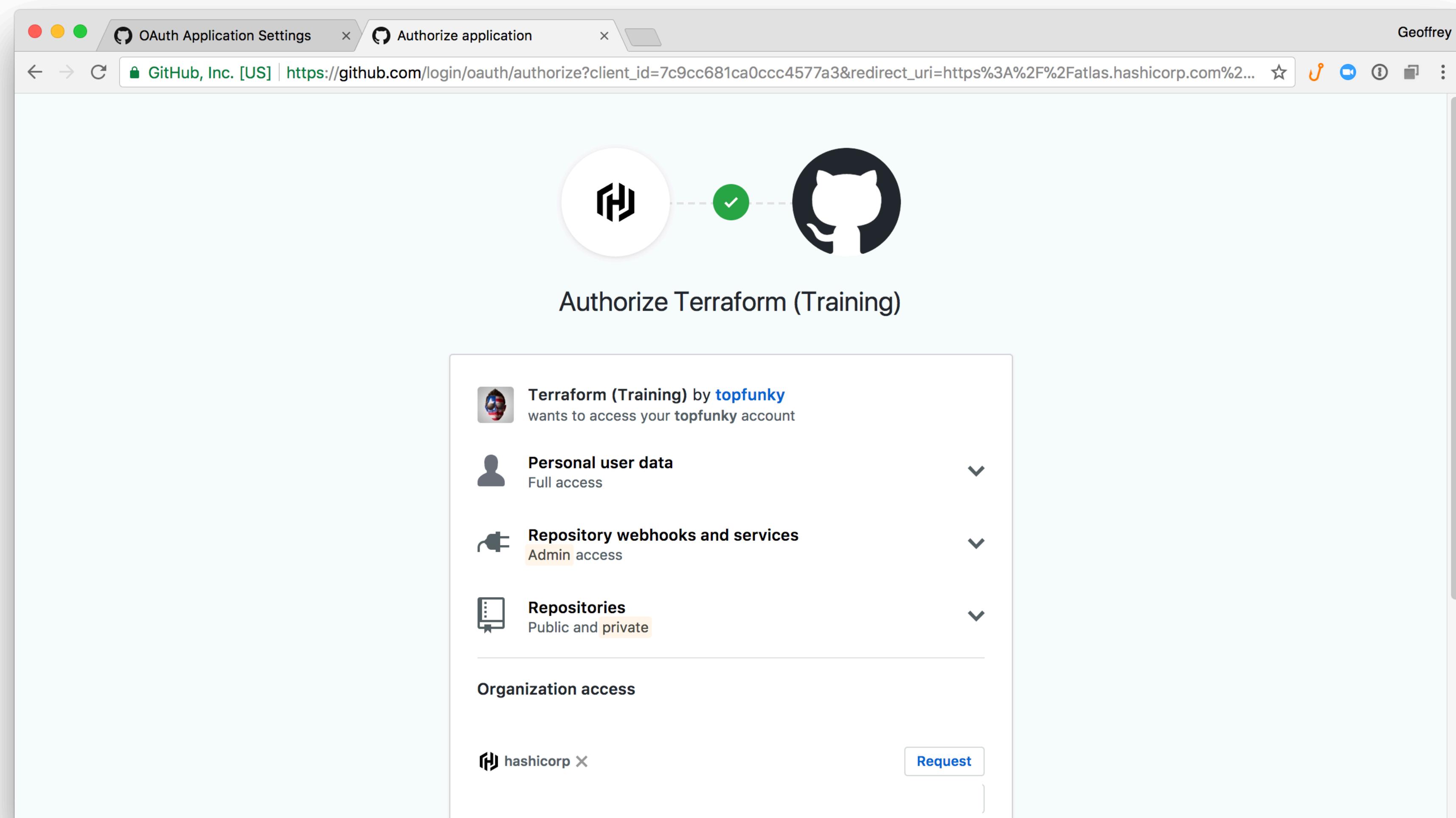
Connect to GitHub

Connecting to GitHub will take your GitHub user through the OAuth flow to create an authorization token for access to all repositories for this organization. This means that your currently logged in GitHub user token will be used for all GitHub API interactions by any Terraform Enterprise user anywhere within the scope of **topfunky**.

Connect organization topfunky

Delete Client





The screenshot shows the 'OAuth Application Settings' page for the 'topfunky' organization in Terraform Enterprise. The left sidebar lists organization settings like Profile, Teams, OAuth Configuration (which is selected), API Token, User Sessions, Manage SSH Keys, and Sentinel Policy. The main content area displays an OAuth Client for GitHub, showing its configuration details: Callback URL (<https://atlas.hashicorp.com/auth/94dc3315-5f10-42af-8e04-7d95b17dcd8a/callback>), HTTP URL (<https://github.com>), and API URL (<https://api.github.com>). It also shows the client was created on Dec 12, 2017 at 17:22:54PM. A note indicates a connection was made via OAuth from a GitHub user named 'topfunky'. At the bottom, there are 'Revoke Connection' and 'Delete Client' buttons.

Geoffrey

Secure | https://atlas.hashicorp.com/app/topfunky/settings/oauth

topfunky Workspaces Documentation Status

ORGANIZATION SETTINGS

topfunky ✓

Profile Teams OAuth Configuration API Token User Sessions Manage SSH Keys Sentinel Policy

OAuth Clients

GitHub

Callback URL
<https://atlas.hashicorp.com/auth/94dc3315-5f10-42af-8e04-7d95b17dcd8a/callback>

HTTP URL
<https://github.com>

API URL
<https://api.github.com>

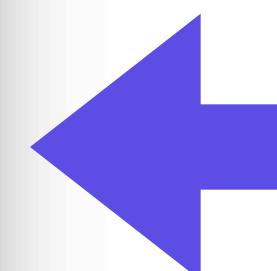
Created
Dec 12, 2017 17:22:54PM

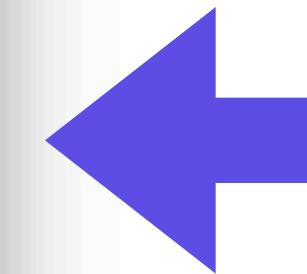
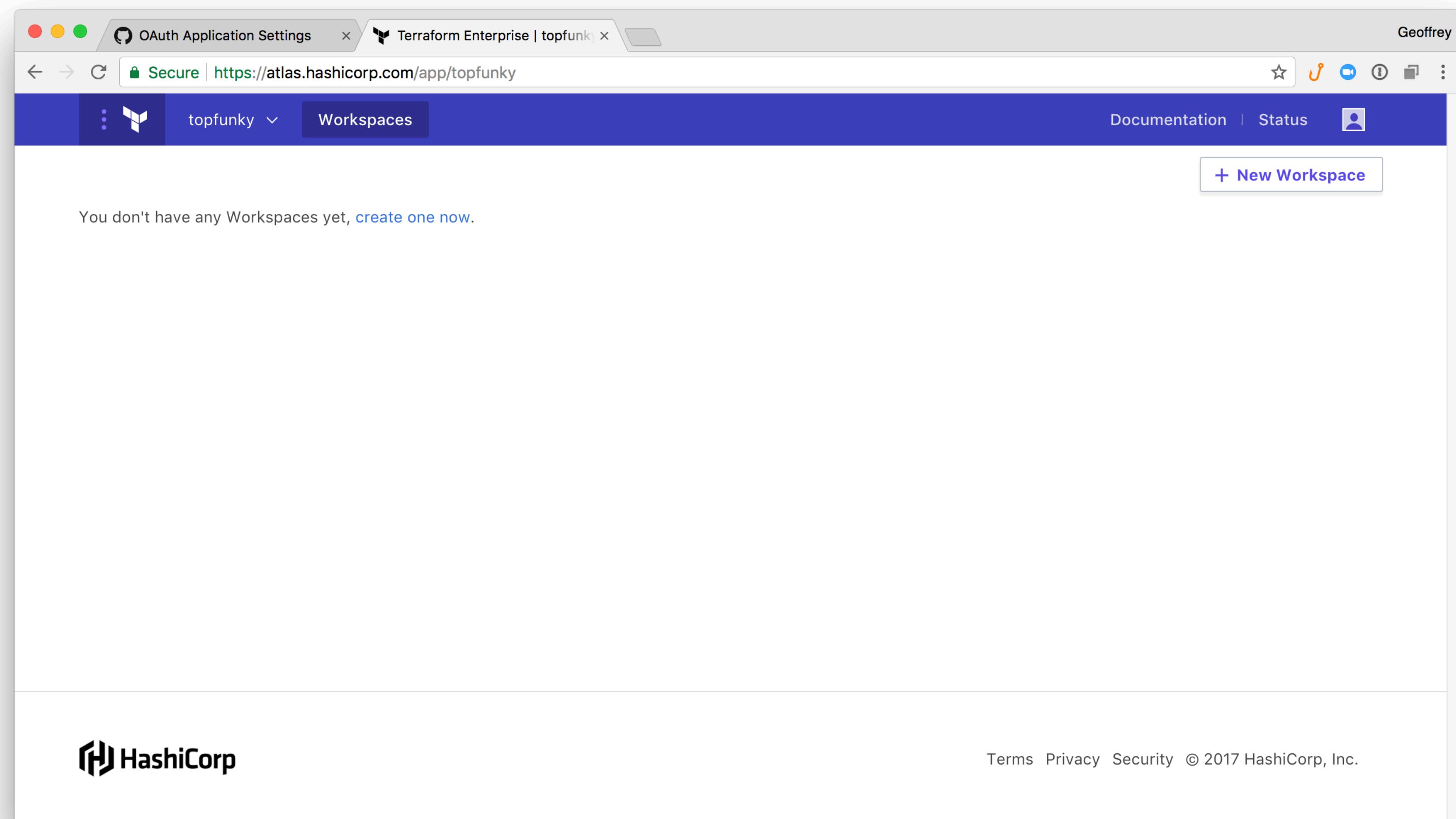
A connection was made on Dec 12, 2017 17:26:01PM by authenticating via OAuth as GitHub user **topfunky**, which assigned an OAuth token for use by all Terraform Enterprise users in the **topfunky** organization.

You can [add a private SSH key](#) to this connection to be used for git clone operations.

Revoke Connection

Delete Client





The screenshot shows a web browser window titled "Terraform Enterprise | topfunk" with the URL "https://atlas.hashicorp.com/app/topfunky/workspaces/new". The page is titled "Create a new Workspace" and displays the following fields:

- WORKSPACE NAME:** A text input field containing "training".

This workspace will be created under the current organization, **topfunky**.
- VCS CONNECTION:** A dropdown menu showing "GitHub".

The name of your workspace is unique and used in tools, routing, and UI. Dashes and alphanumeric characters are permitted. Learn more about [naming workspaces](#).
- REPOSITORY:** A search bar and a dropdown menu showing "topfunky/training".

The screenshot shows a web browser window titled "Terraform Enterprise | topfunky-demo/terraform-intro-demo" with the URL "https://atlas.hashicorp.com/app/topfunky-demo/terraform-intro-demo/integrations". The page displays the "Version Control Integration" configuration for this workspace. A blue box highlights the message: "This workspace is linked to the repository **topfunky-demo/terraform-intro-demo**". Below this, there are four configuration sections: "TERRAFORM WORKING DIRECTORY", "VCS ROOT PATH", "VCS BRANCH", and "SSH KEY". Each section has a description and a text input field. A large blue arrow points from the bottom right towards the "VCS BRANCH" section.

This workspace is linked to the repository **topfunky-demo/terraform-intro-demo**.

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.

VCS ROOT PATH

This path, if specified, will be used as the root of the Terraform execution context and all files outside of this path will be thrown away. The Terraform Working Directory is calculated relative to this subpath.

VCS BRANCH

after

The branch from which to import new versions. This defaults to the value your version control provides as the default branch for this repository.

SSH KEY

The screenshot shows a web browser window for Terraform Enterprise. The title bar reads "OAuth Application Settings" and "Terraform Enterprise | topfunk". The user is identified as "Geoffrey". The URL in the address bar is "Secure | https://atlas.hashicorp.com/app/topfunk/training/runs". The main header has tabs for "Workspaces" (selected), "Documentation", and "Status". Below the header, the workspace name "topfunk" is shown. A "Queue Plan" button is visible on the right. The navigation menu includes "Latest Run", "Runs" (which is selected), "States", "Variables", "Settings", "Integrations", and "Access". A success message "Configuration uploaded successfully" with a green checkmark is displayed. It advises the user to configure variables or queue a plan. Buttons for "Configure Variables" and "Queue Plan" are present. The HashiCorp logo is at the bottom left, and a footer links to Terms, Privacy, Security, and copyright information.

Geoffrey

Secure | https://atlas.hashicorp.com/app/topfunk/training/runs

topfunk

Workspaces

Documentation | Status

Queue Plan

Latest Run Runs States Variables Settings Integrations Access

Configuration uploaded successfully

Your configuration has been uploaded. Next, you probably want to configure variables (such as access keys or configuration values). If your configuration doesn't require variables, you can queue your first plan now.

Configure Variables Queue Plan

HashiCorp

Terms Privacy Security © 2017 HashiCorp, Inc.

The screenshot shows the 'Variables' tab of the Terraform Enterprise interface for the 'training' workspace. The top navigation bar includes tabs for 'Latest Run', 'Runs', 'States', 'Variables' (which is active), 'Settings', 'Integrations', and 'Access'. A 'Queue Plan' button is located in the top right corner. The main content area is divided into three sections: 'Terraform Variables', 'Environment Variables', and 'Personal Environment Variables'. Each section contains a message indicating no variables are set and provides an 'Edit' toggle switch.

Terraform Variables

These variables are set using the `var` option when performing a plan and apply

Environment Variables

These variables are set using `export` in the environment running the plan and apply

Personal Environment Variables

These environment variables will be included in Terraform runs created by you for the current

The screenshot shows the Terraform Enterprise interface for managing variables in a workspace named "training". The "Variables" tab is selected in the navigation bar. A message at the top indicates that variables are set using the `var` option during plan and apply. The "Editing" toggle is turned on. The variables listed are:

Variable Name	Value (Redacted)	HCL	Sensitive
access_key	xxxxxxxxxx	<input type="checkbox"/>	<input checked="" type="checkbox"/> Sensitive
secret_key	xxxxxxxxxx	<input type="checkbox"/>	<input checked="" type="checkbox"/> Sensitive
ami	xxxxx	<input type="checkbox"/>	<input type="checkbox"/> Sensitive
subnet_id	xxxxx	<input type="checkbox"/>	<input type="checkbox"/> Sensitive
vpc_security_group_id	xxxxx	<input type="checkbox"/>	<input type="checkbox"/> Sensitive
identity	xxxxx	<input type="checkbox"/>	<input type="checkbox"/> Sensitive

Terraform Enterprise | geoffrey

Secure | https://atlas.hashicorp.com/app/geoffrey-org/tmp-terraform/variables

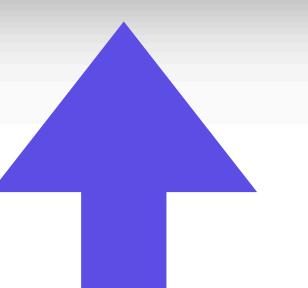
Terraform Variables

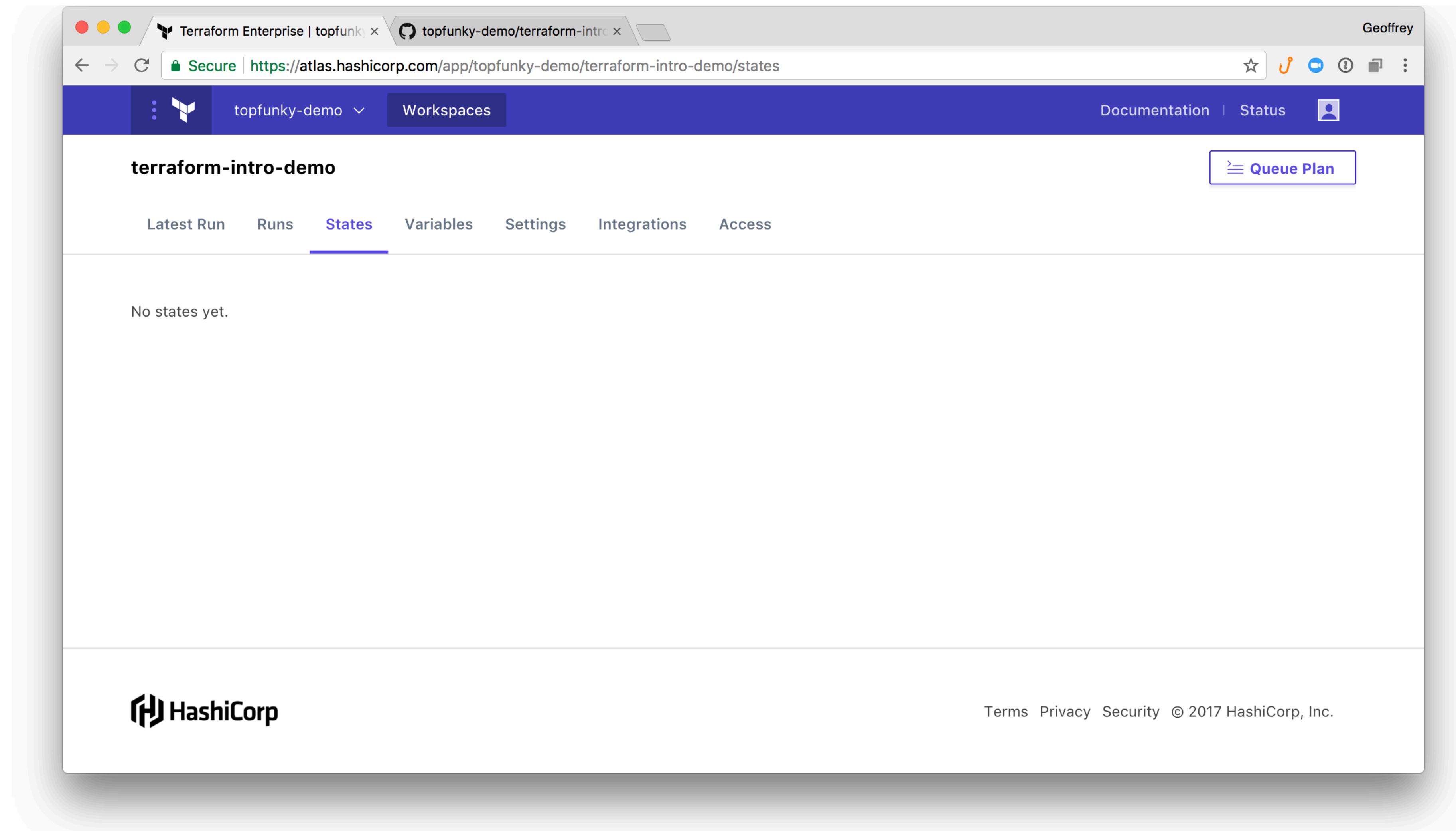
These variables are set using the `var` option when performing a plan and apply

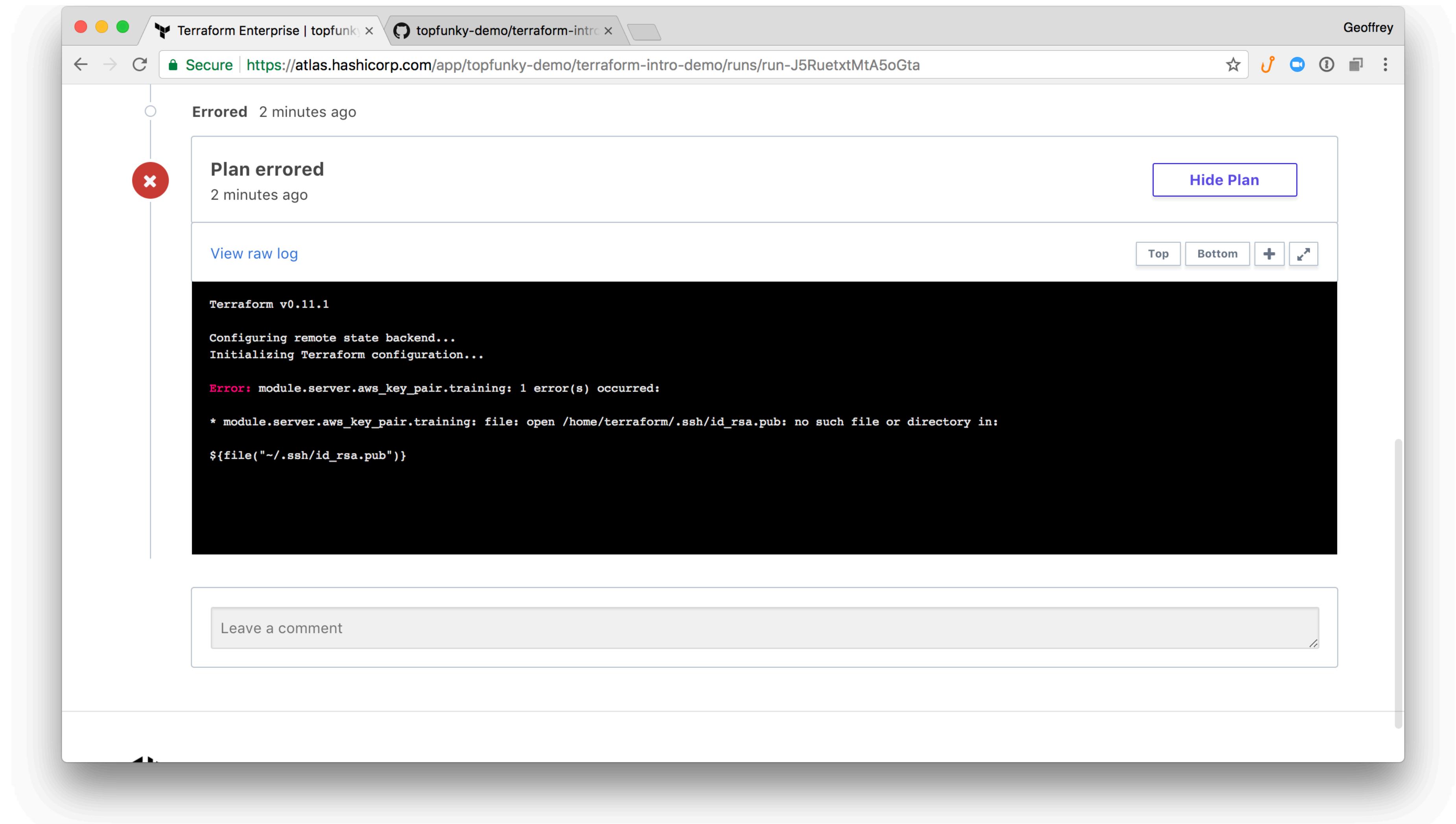
Editing

region	us-west-2	<input type="checkbox"/> HCL	<input type="checkbox"/> Sensitive	
access_key	AKIAIJ4OJYSU5HRXLE5Q	<input type="checkbox"/> HCL	<input checked="" type="checkbox"/> Sensitive	
secret_key	JnN7nRQkJISNZLuYiMBQq0X9krd	<input type="checkbox"/> HCL	<input checked="" type="checkbox"/> Sensitive	
ami	ami-a2e544da	<input type="checkbox"/> HCL	<input type="checkbox"/> Sensitive	
subnet_id	subnet-ae7e2de6	<input type="checkbox"/> HCL	<input type="checkbox"/> Sensitive	
identity	demo-geoffrey-duck	<input type="checkbox"/> HCL	<input type="checkbox"/> Sensitive	
vpc_security_group_id	sg-e5db4799	<input type="checkbox"/> HCL	<input type="checkbox"/> Sensitive	
key	value	<input type="checkbox"/> HCL	<input type="checkbox"/> Sensitive	Add

Save & Plan **Save** **Cancel**







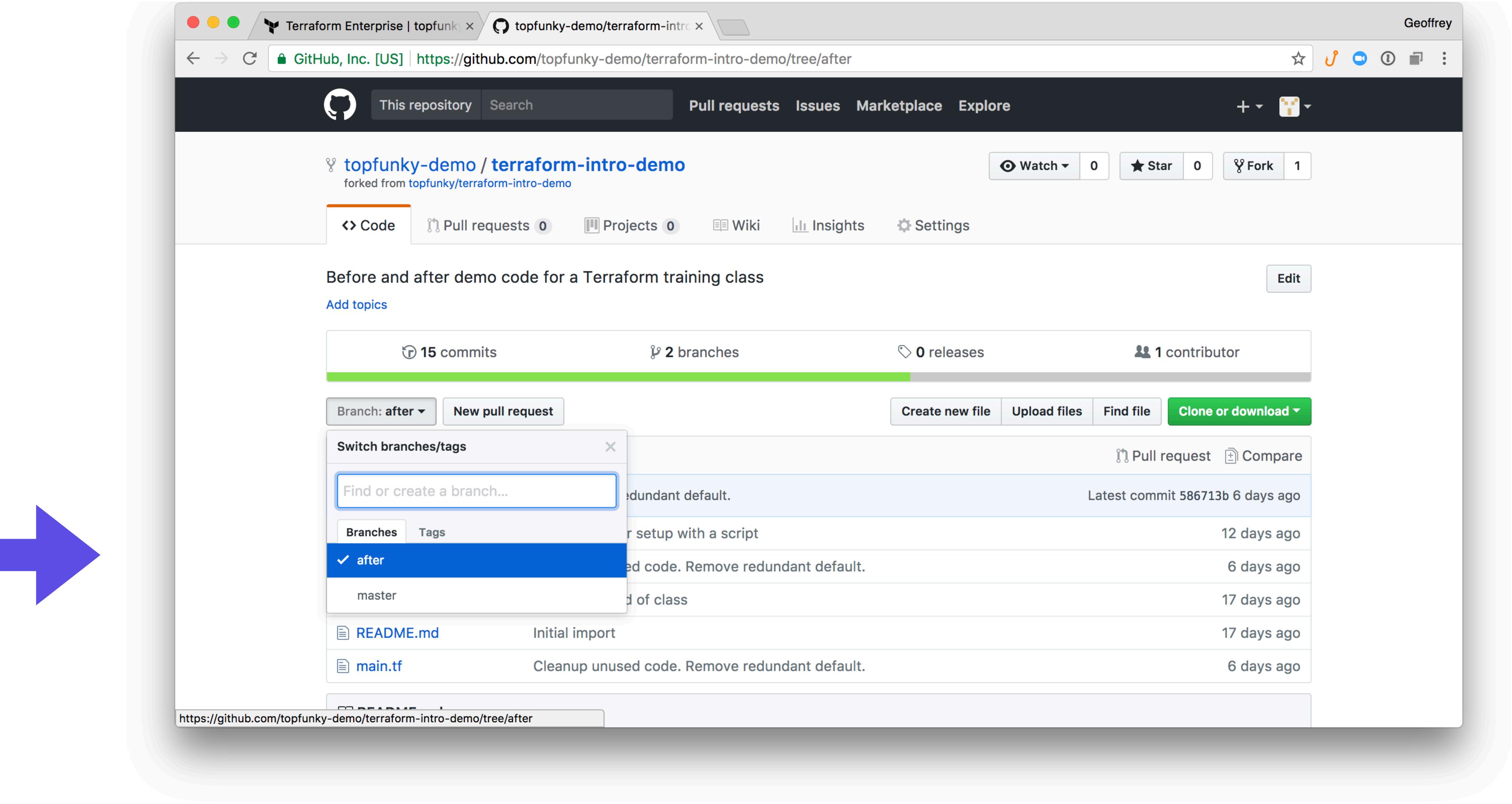
Remove/Repair References to the Local Filesystem

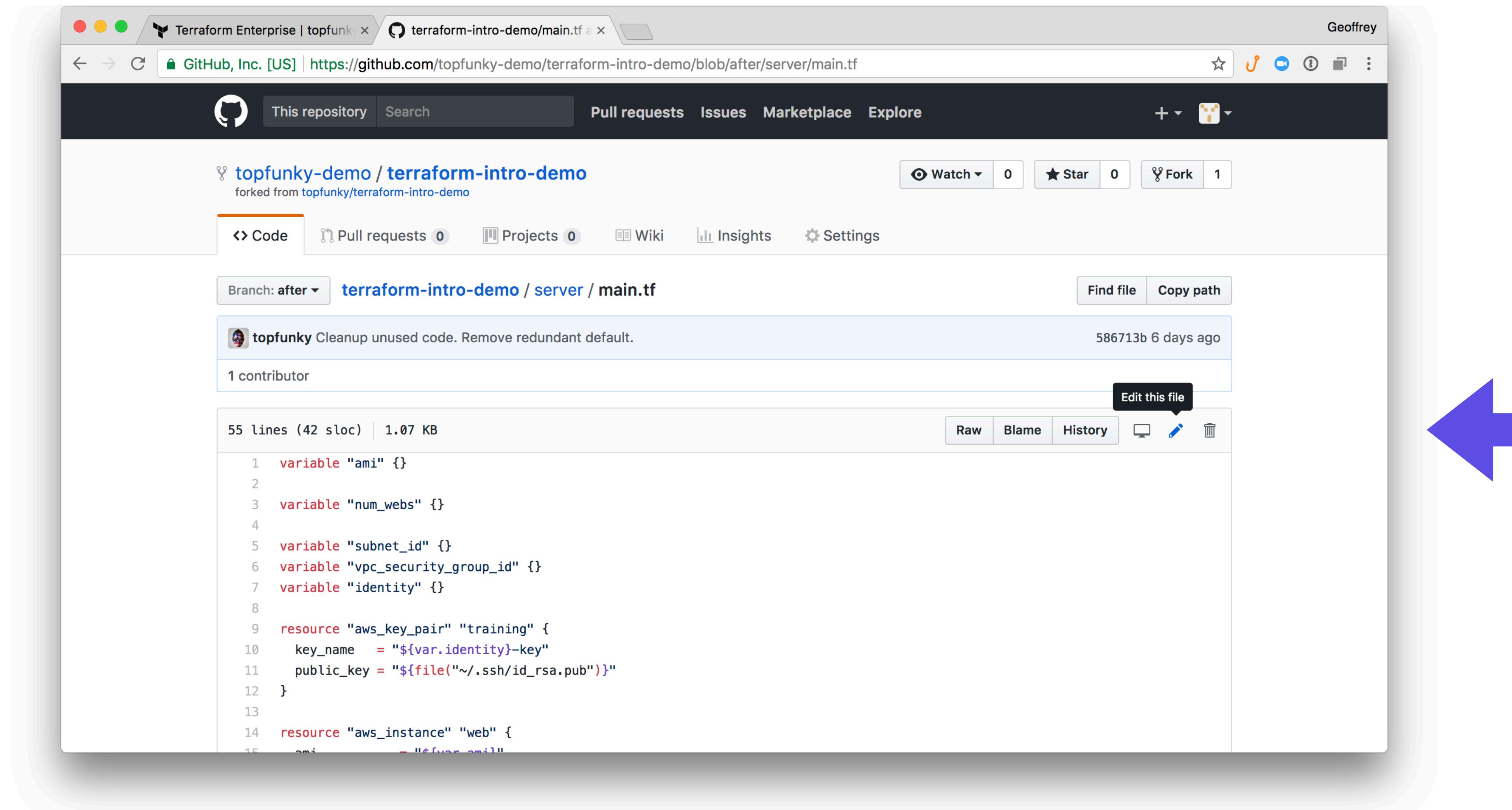
Unlike a CI platform, Terraform Cloud doesn't allow you to execute shell scripts before or after a run.

You're limited to the exact contents of your Git repository plus the variables you've set up.

The SSH keys we used were part of our instance but aren't part of the Terraform environment.

Instead of reading files, use content from variables.





Terraform Enterprise | topfunk... x terraform-intro-demo/main.tf a x Geoffrey

GitHub, Inc. [US] | https://github.com/topfunky-demo/terraform-intro-demo/blob/after/server/main.tf

This repository Search Pull requests Issues Marketplace Explore + ⚙️

topfunky-demo / terraform-intro-demo forked from topfunky/terraform-intro-demo Watch 0 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Insights Settings

Branch: after → terraform-intro-demo / server / main.tf Find file Copy path

topfunky Cleanup unused code. Remove redundant default. 586713b 6 days ago

1 contributor

55 lines (42 sloc) | 1.07 KB Raw Blame History

```
1 variable "ami" {}
2
3 variable "num_webs" {}
4
5 variable "subnet_id" {}
6 variable "vpc_security_group_id" {}
7 variable "identity" {}

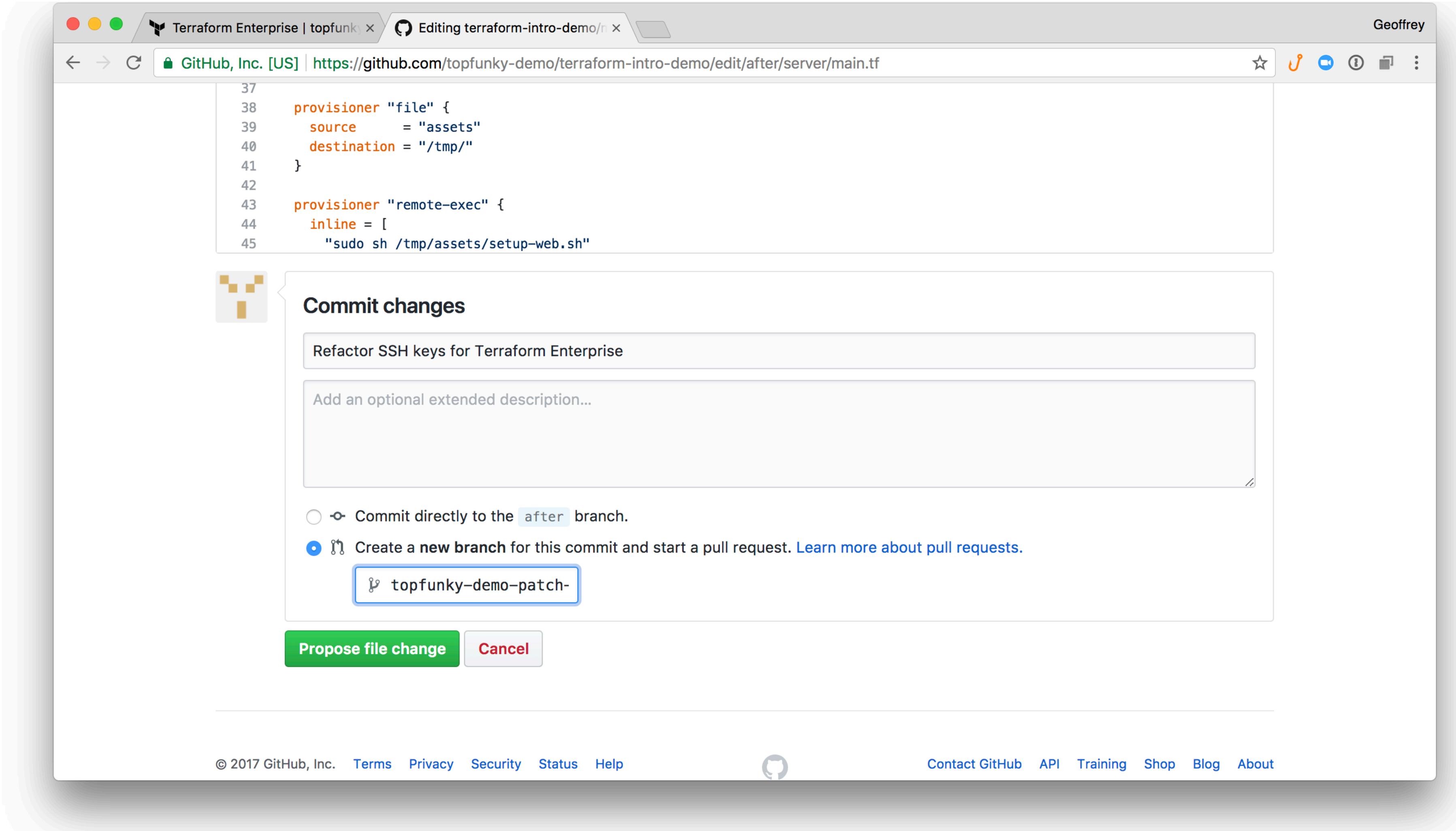
9 resource "aws_key_pair" "training" {
10   key_name    = "${var.identity}-key"
11   public_key  = "${file("~/.ssh/id_rsa.pub")}"
12 }
13
14 resource "aws_instance" "web" {
15   ami           = "ami-0f5f5e0f5f5f5f5f5"
16 }
```

```
variable "public_key" {}
variable "private_key" {}

resource "aws_key_pair" "training" {
    key_name      = "${var.identity}-key"
    public_key    = var.public_key
}

resource "aws_instance" "web" {
    ami           = var.ami
    # . . .

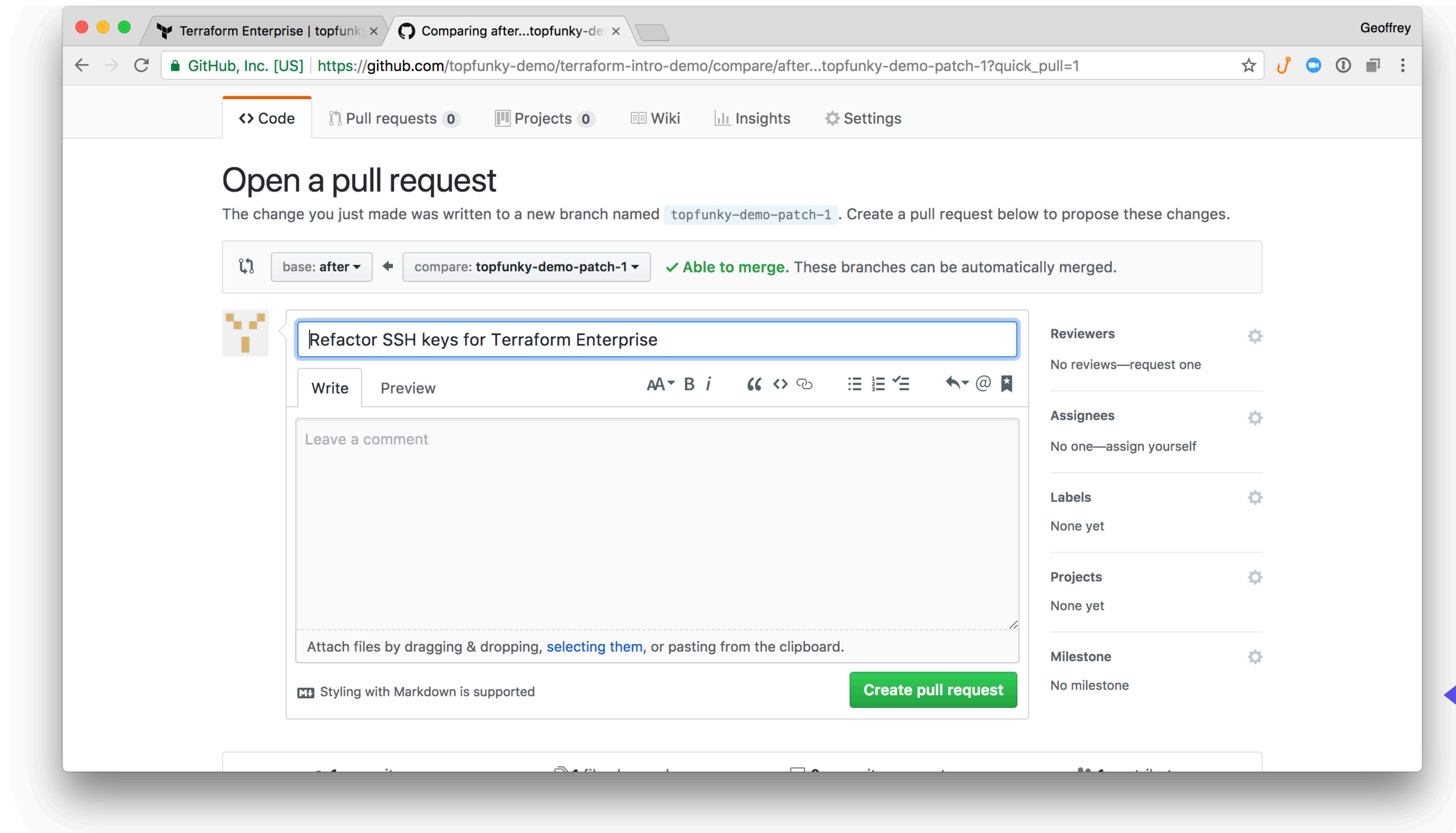
    connection {
        user          = "ubuntu"
        private_key   = var.private_key
    }
}
```

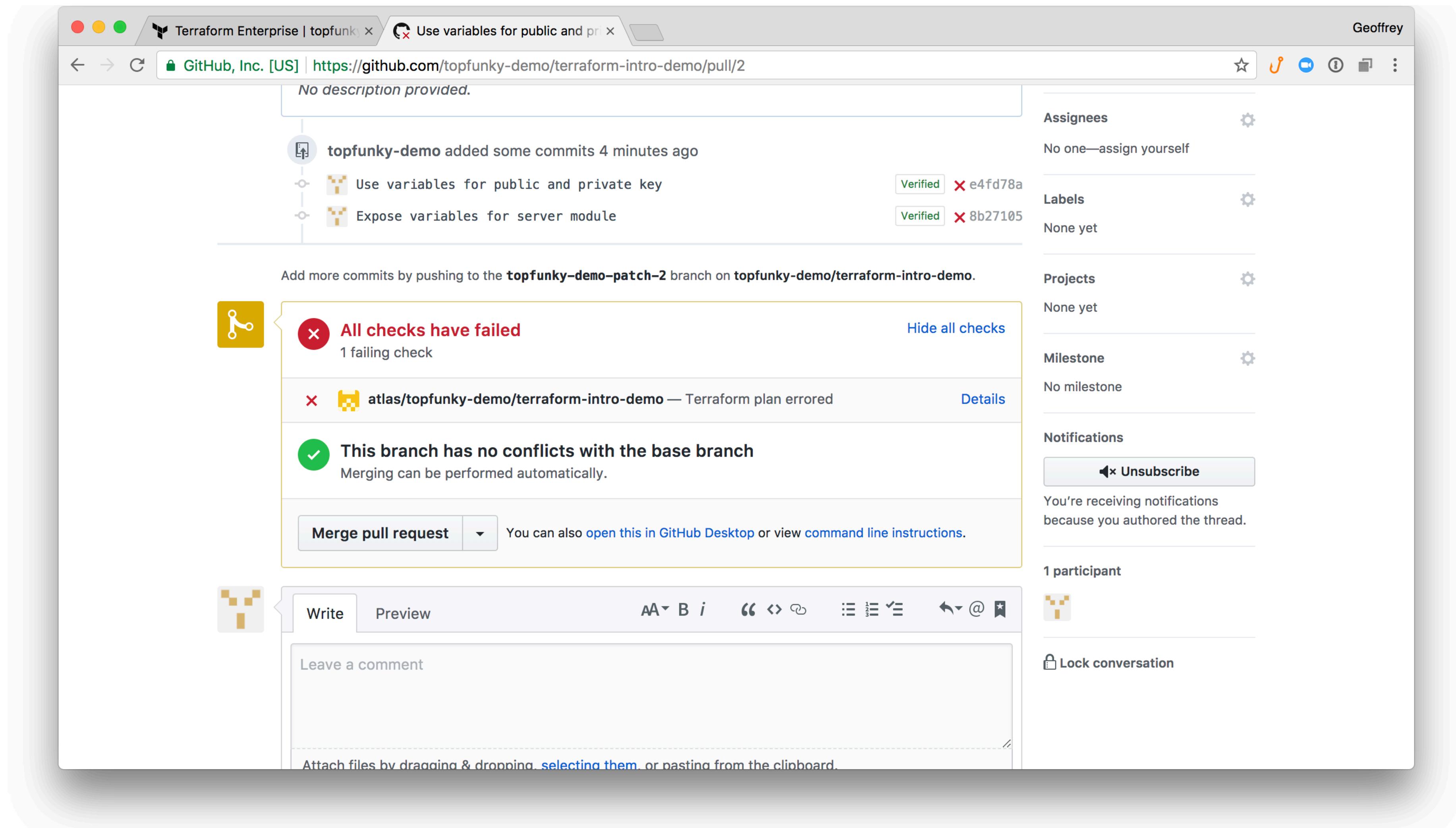


A screenshot of a GitHub commit dialog box overlaid on a browser window showing a Terraform configuration file. The dialog box is titled "Commit changes". It contains a text input field with "Refactor SSH keys for Terraform Enterprise" and a larger text area below it. At the bottom of the dialog, there are two radio button options: "Commit directly to the `after` branch." and "Create a new branch for this commit and start a pull request. Learn more about pull requests." The second option is selected, and the branch name "topfunky-demo-patch-" is entered in the text field. Below the dialog are two buttons: "Propose file change" (green) and "Cancel" (gray). The browser window shows a Terraform configuration file with code snippets:

```
37
38 provisioner "file" {
39   source      = "assets"
40   destination = "/tmp/"
41 }
42
43 provisioner "remote-exec" {
44   inline = [
45     "sudo sh /tmp/assets/setup-web.sh"

```





WARNING

Pull Request Plan Runs Will Not Appear in TFC

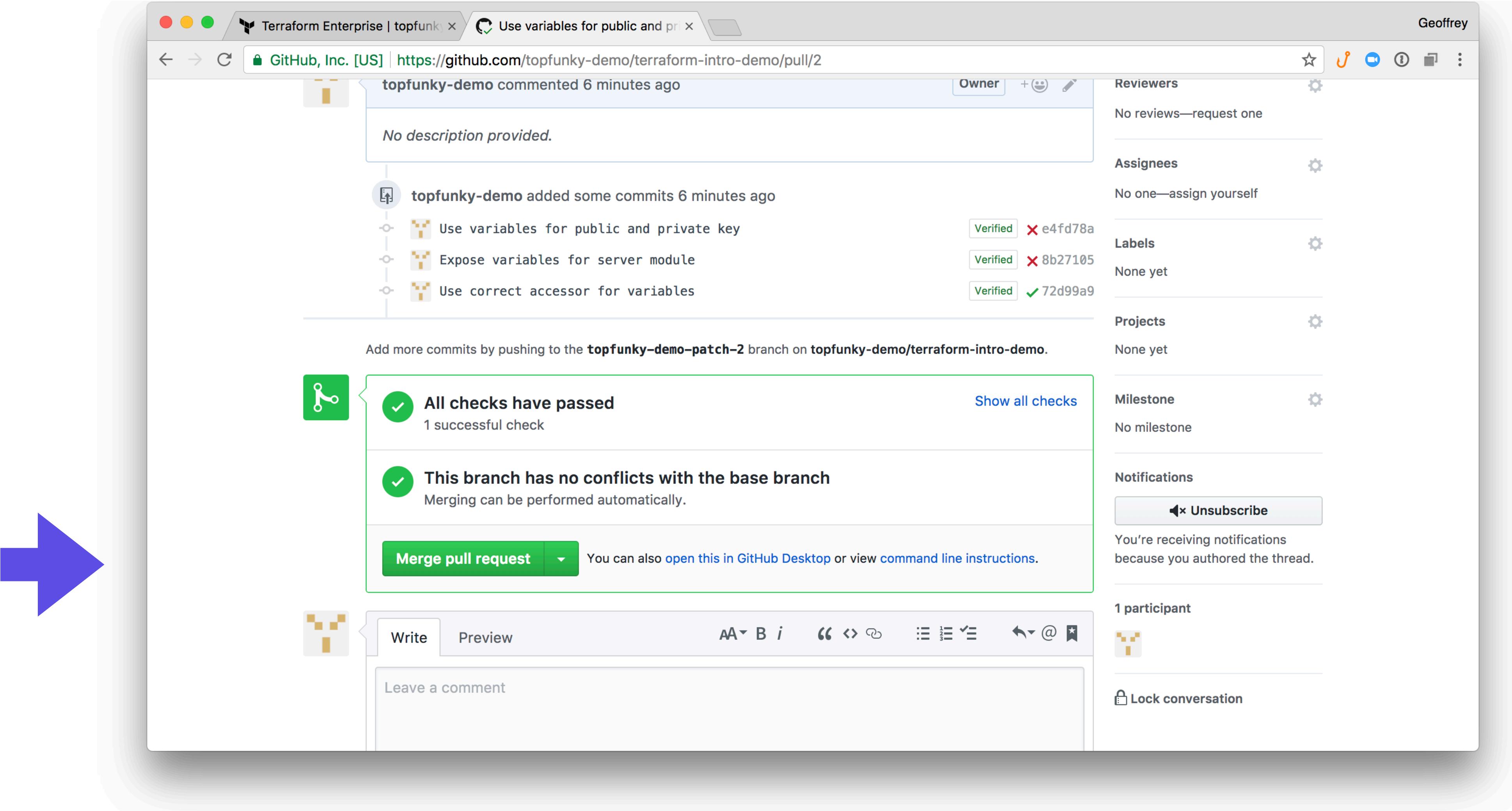
When Terraform Cloud runs a plan against a pull request, the only way to see the result is through the link in the pull request at GitHub.

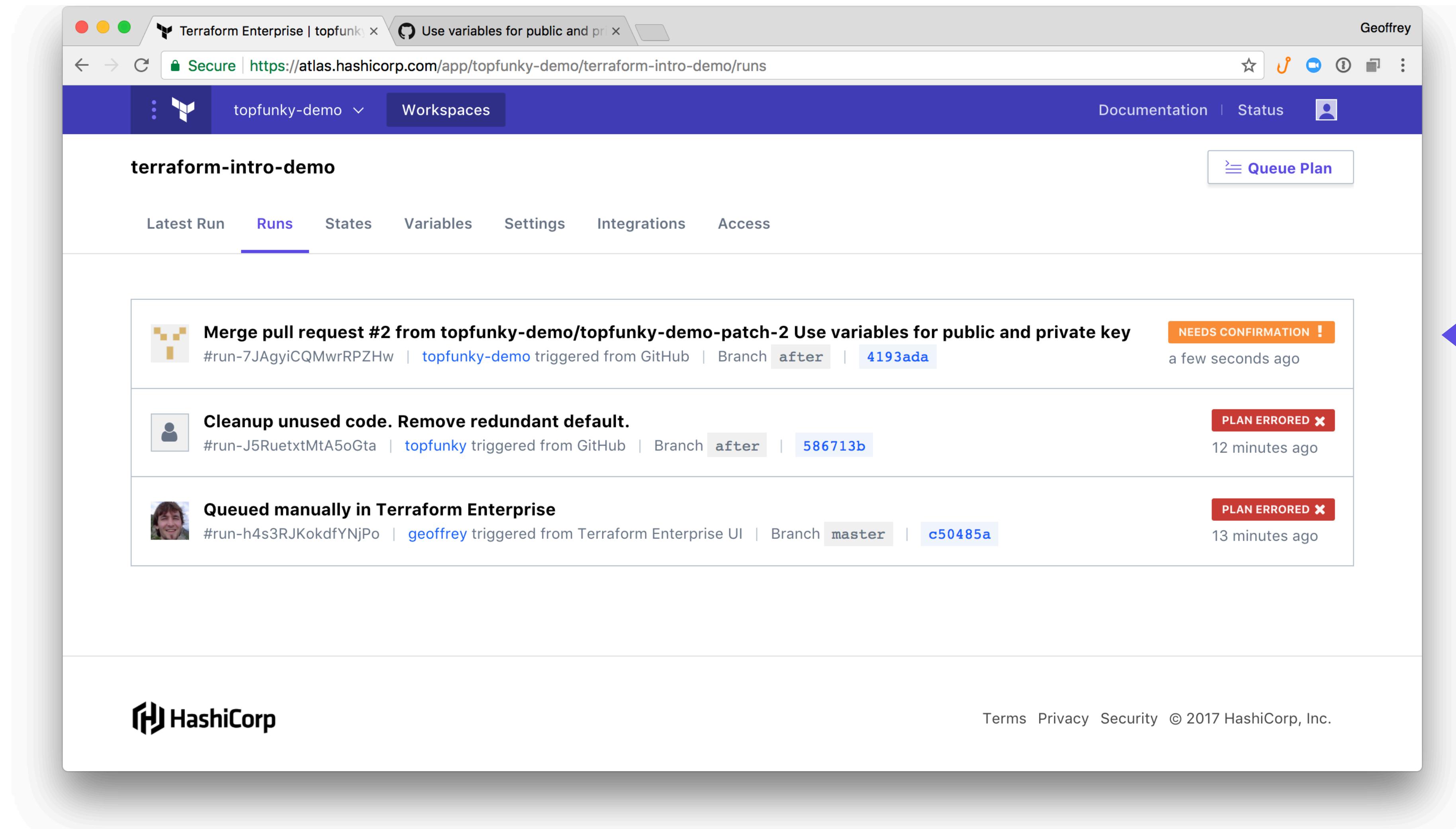
The plan is ephemeral and can not be navigated to in the Terraform Cloud UI.

```
variable "public_key" {}
variable "private_key" {}

module "server" {
  source = "./server"

  num_webs          = var.num_webs
  ami               = var.ami
  subnet_id         = var.subnet_id
  vpc_security_group_id = var.vpc_security_group_id
  identity          = var.identity
  public_key         = var.public_key
  private_key        = var.private_key
}
```





Terraform Enterprise | topfunk... | Use variables for public and pr... Geoffrey

Secure | https://atlas.hashicorp.com/app/topfunky-demo/terraform-intro-demo/runs

topfunky-demo Workspaces Documentation | Status

terraform-intro-demo Queue Plan

Latest Run Runs States Variables Settings Integrations Access

Merge pull request #2 from topfunky-demo/topfunky-demo-patch-2 Use variables for public and private key NEEDS CONFIRMATION ! #run-7JAgyiCQMwrRPZHW | topfunky-demo triggered from GitHub | Branch after | 4193ada a few seconds ago

Cleanup unused code. Remove redundant default. PLAN ERRORED ✘ #run-J5RuetxtMtA5oGta | topfunky triggered from GitHub | Branch after | 586713b 12 minutes ago

Queued manually in Terraform Enterprise PLAN ERRORED ✘ #run-h4s3RJKokdfYNjPo | geoffrey triggered from Terraform Enterprise UI | Branch master | c50485a 13 minutes ago

HashiCorp Terms Privacy Security © 2017 HashiCorp, Inc.

A screenshot of a web browser window titled "Terraform Enterprise | topfunk" showing a run details page. The URL is "Secure | https://atlas.hashicorp.com/app/topfunky-demo/terraform-intro-demo/runs/run-7JAgyiCQMwrRPZHw". The page displays a timeline of events:

- Variables input to this run
- Configuration input to this run
- Run created by GitHub Webhook**

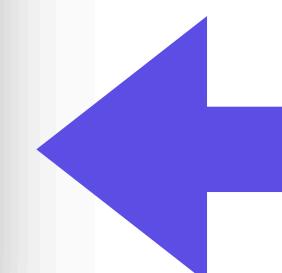
Below this, under the heading **PLAN**, it shows:

- Queued a minute ago
- Started a minute ago

A green circular icon with a checkmark indicates the plan was executed successfully. The message "Executed and saved successfully" is displayed, along with the timestamp "a minute ago". A "View Plan" button is located to the right.

The next section, "State versions output with this plan", shows "No state versions output".

At the bottom, there is a comment input field labeled "Leave a comment" and two buttons: "Confirm & Apply" (in blue) and "Discard Plan".



The screenshot shows a Terraform Enterprise interface for a run titled "topfunky-demo/terraform-intro-demo#sv-G8QTDAv9vuHKzDqA". The status bar indicates the run was queued and started a few seconds ago. A large green checkmark icon is present on the left. The main content area displays a success message: "Executed successfully with changes a few seconds ago". It includes a "View raw log" link and a terminal log window. The log shows the execution of a module, including connecting via SSH and creating an EC2 instance. It also lists outputs: public_dns and public_ip. Below the log is a section for state versions output with this apply, listing two items. A purple arrow points from the bottom right towards the terminal log area.

APPLY

Queued a few seconds ago
Started a few seconds ago

Executed successfully with changes
a few seconds ago

[View raw log](#)

```
module.server.aws_instance.web (remote-exec):  Private key: true
module.server.aws_instance.web (remote-exec):  SSH Agent: false
module.server.aws_instance.web (remote-exec): Connected!
module.server.aws_instance.web: Creation complete after 27s (ID: i-0936d03b68874f0b6)

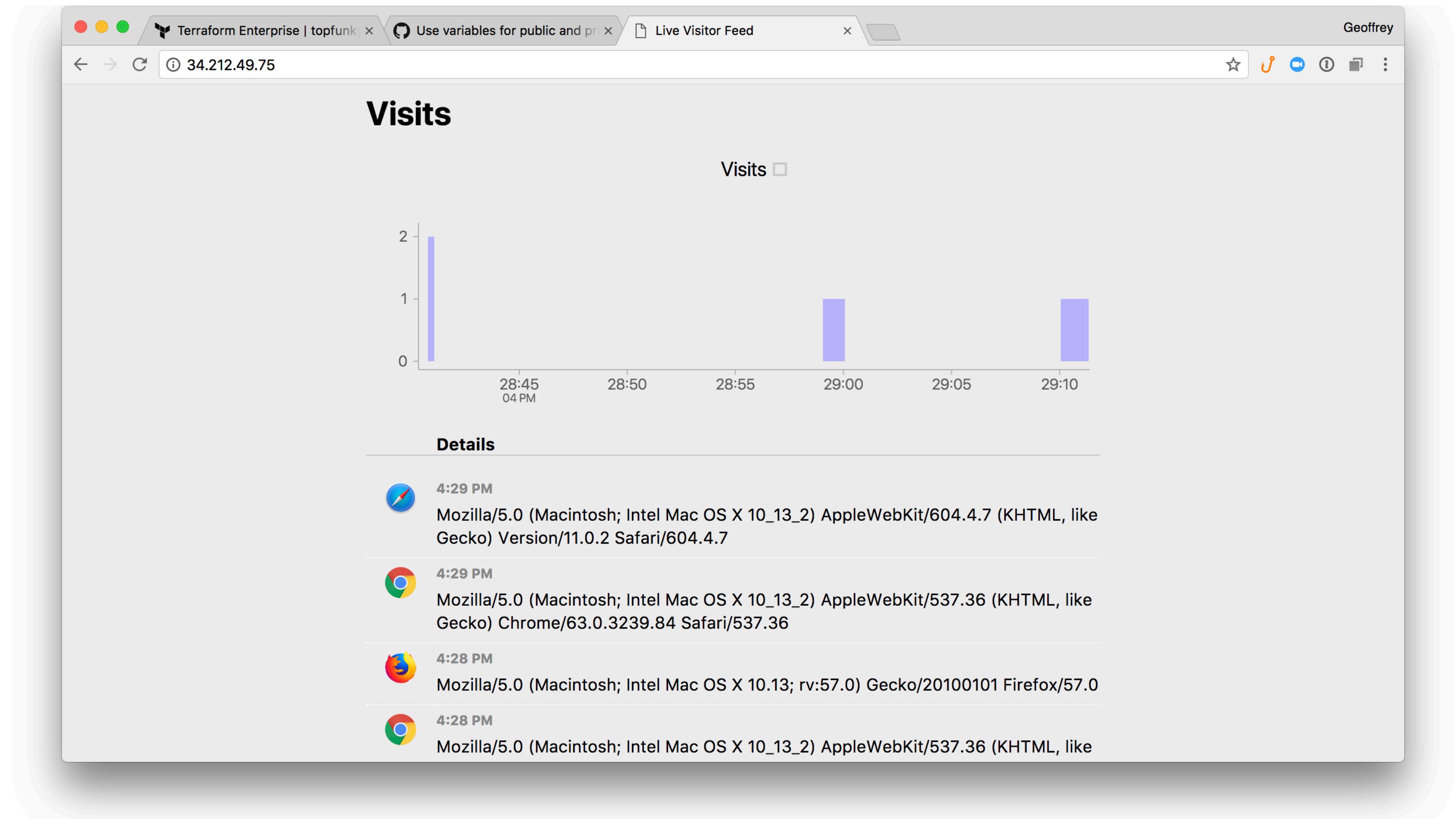
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

public_dns = [
    ec2-34-212-49-75.us-west-2.compute.amazonaws.com
]
public_ip = [
    34.212.49.75
]
```

State versions output with this apply

- topfunky-demo/terraform-intro-demo#sv-G8QTDAv9vuHKzDqA
- topfunky-demo/terraform-intro-demo#sv-LdHR89hU5K9sEmNM



Instructor Discretion Exercises

If time permits, the instructor may suggest extra exercises to complete using Terraform Cloud.

Terraform Cloud: Review

Integrates with SCM and application workflows

Provides a dry-run experience for infrastructure changes

Continuous Integration, but for Infrastructure as Code

What You Learned

- ✓ Create a repository at GitHub
- ✓ Push your code to GitHub
- ✓ Create an organization at Terraform Cloud
- ✓ Connect to GitHub via OAuth
- ✓ Add a workspace that references a GitHub repo
- ✓ Commit changes and create infrastructure

What You'll Learn

- Destroy infrastructure that Terraform has created

Command: terraform destroy

Destroys running infrastructure

Does not touch infrastructure not managed by
Terraform

The screenshot shows a web browser window titled "Terraform Enterprise | topfunk". The URL is "https://atlas.hashicorp.com/app/topfunk/training/settings". The page displays a message: "You can manually unlock this workspace to resume Terraform plans and applies." Below this is a button labeled "Unlock topfunk/training".

Workspace Delete

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 Enterprise, including any variables, settings, and alert history can be deleted.

Queueing a destroy Plan will redirect to a new Plan that will destroy all of the infrastructure managed by Terraform. This requires confirmation before Apply. It is equivalent to running `terraform plan -destroy -out=destroy.tfplan` followed by `terraform apply destroy.tfplan` locally.

Queueing a destroy Plan will be disabled until there is an environment variable set named `CONFIRM_DESTROY` with a value of `1`. You can use the variables page to set it.

[Queue destroy Plan](#) [Delete from Terraform Enterprise](#)

HashiCorp Terms Privacy Security © 2017 HashiCorp, Inc.

The screenshot shows the Terraform Enterprise application window titled "Terraform Enterprise | topfunk". The URL in the address bar is "Secure | https://atlas.hashicorp.com/app/topfunk/training/variables". The main content area displays two sections: "Environment Variables" and "Personal Environment Variables".

Environment Variables

These variables are set using `export` in the environment running the plan and apply

Editing

CONFIRM_DESTROY	1	<input type="checkbox"/> Sensitive	Add
-----------------	---	------------------------------------	-----

Save & Plan Save Cancel

Personal Environment Variables

These environment variables will be included in Terraform runs created by you for the current workspace. They will not apply for other users. They will override all other environment variables.

Edit

There are no variables set.

Destroying in Terraform Cloud

It must be an environment variable, not any other kind of variable

Do not click “Save and Plan.” This will not queue a destroy plan (it will queue a normal plan)

The screenshot shows a web browser window for 'Terraform Enterprise | topfunk'. The title bar includes 'OAuth Application Settings' and 'Secure | https://atlas.hashicorp.com/app/topfunk/training/settings'. The main content area is titled 'Settings' and contains the following sections:

- Auto apply**: Describes automatic application of changes when a Terraform plan is successful.
- Manual apply**: Describes requiring operator confirmation before applying changes.
- TERRAFORM VERSION**: Set to '0.11.1'.

A 'Save settings' button is at the bottom.

The screenshot shows a web browser window titled "Terraform Enterprise | topfunk". The URL in the address bar is <https://atlas.hashicorp.com/app/topfunk/training/settings>. The page content includes a message: "You can manually unlock this workspace to resume Terraform plans and applies." Below this is a button labeled "Unlock topfunk/training". The main section is titled "Workspace Delete" and contains the following text: "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 Enterprise, including any variables, settings, and alert history can be deleted." It also describes the process of queueing a destroy Plan. At the bottom, there are two buttons: "Queue destroy Plan" and "Delete from Terraform Enterprise".

You can manually unlock this workspace to resume Terraform plans and applies.

[Unlock topfunk/training](#)

Workspace Delete

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 Enterprise, including any variables, settings, and alert history can be deleted.

Queueing a destroy Plan will redirect to a new Plan that will destroy all of the infrastructure managed by Terraform. This requires confirmation before Apply. It is equivalent to running `terraform plan -destroy -out=destroy.tfplan` followed by `terraform apply destroy.tfplan` locally.

Queueing a destroy Plan will be disabled until there is an environment variable set named `CONFIRM_DESTROY` with a value of `1`. You can use the variables page to set it.

[Queue destroy Plan](#) [Delete from Terraform Enterprise](#)

HashiCorp

[Terms](#) [Privacy](#) [Security](#) © 2017 HashiCorp, Inc.

Terraform Enterprise | topfunky | topfunky-demo/terraform-intro-demo | Geoffrey

Secure | https://atlas.hashicorp.com/app/topfunky-demo/terraform-intro-demo/runs

topfunky-demo | Workspaces | Documentation | Status |

terraform-intro-demo | Queue Plan

Latest Run | Runs | States | Variables | Settings | Integrations | Access

Queued manually to destroy infrastructure | NEEDS CONFIRMATION !
#run-fU78NLx9UZdsLTYc | geoffrey triggered from Terraform Enterprise UI | Branch after | 4193ada | a few seconds ago

Merge pull request #2 from topfunky-demo/topfunky-demo-patch-2 Use variables for public and private key | APPLIED ✓
#run-7JAgyiCQMwrRPZHW | topfunky-demo triggered from GitHub | Branch after | 4193ada | 7 minutes ago

Cleanup unused code. Remove redundant default. | PLAN ERRORRED ✗
#run-J5RuetxtMtA5oGta | topfunky triggered from GitHub | Branch after | 586713b | 19 minutes ago

Queued manually in Terraform Enterprise | PLAN ERRORRED ✗
#run-h4s3RJKokdfYNjPo | geoffrey triggered from Terraform Enterprise UI | Branch master | c50485a | 21 minutes ago

The screenshot shows a Terraform Enterprise interface for a run titled "topfunky-demo/terraform-intro-demo/run-fU78NLx9UZdsLYc". The status bar indicates "Executed and saved successfully" a minute ago. A "View raw log" link is available. The main content area displays the execution plan:

```
An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
- destroy  
  
Terraform will perform the following actions:  
  
- module.server.aws_instance.web  
  
- module.server.aws_key_pair.training  
  
Plan: 0 to add, 0 to change, 2 to destroy.
```

Below the plan, a section titled "State versions output with this plan" shows "No state versions output".

At the bottom, there is a comment input field labeled "Leave a comment" and two buttons: "Confirm & Apply" (highlighted in blue) and "Discard Plan".

The screenshot shows a Terraform Enterprise interface for a demo run. The title bar indicates the session is for 'Geoffrey' and the specific run is 'topfunky-demo/terraform-intro-demo/run-fU78NLx9UZdsLYc'. The main content area displays the results of a successful 'Apply' operation:

Executed successfully with changes
a few seconds ago

Hide Apply

View raw log

Terraform v0.11.1

```
Initializing plugins and modules...
module.server.aws_instance.web: Destroying... (ID: i-0936d03b68874f0b6)
module.server.aws_instance.web: Still destroying... (ID: i-0936d03b68874f0b6, 10s elapsed)
module.server.aws_instance.web: Still destroying... (ID: i-0936d03b68874f0b6, 20s elapsed)
module.server.aws_instance.web: Destruction complete after 21s
module.server.aws_key_pair.training: Destroying... (ID: demo-geoffrey-duck-key)
module.server.aws_key_pair.training: Destruction complete after 0s

Apply complete! Resources: 0 added, 0 changed, 2 destroyed.
```

State versions output with this apply

- [topfunky-demo/terraform-intro-demo#sv-hu293aoK4Q1P5HVa](#)
- [topfunky-demo/terraform-intro-demo#sv-Mg7iWaNcokm951UD](#)

Leave a comment

Topics for Self-Exploration

State Environments

terraform.io/docs/state/environments.html

Custom Providers

terraform.io/docs/plugins/index.html

Destroy Provisioners

[hashicorp.com/blog/terraform-0-9/#destroy-provisioners](https://www.hashicorp.com/blog/terraform-0-9/#destroy-provisioners)

Q & A