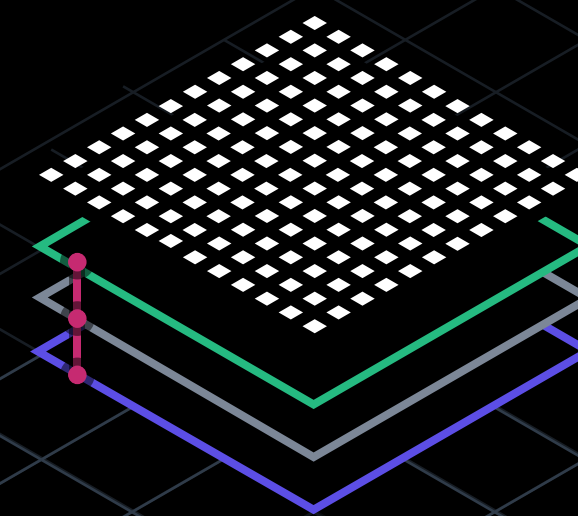




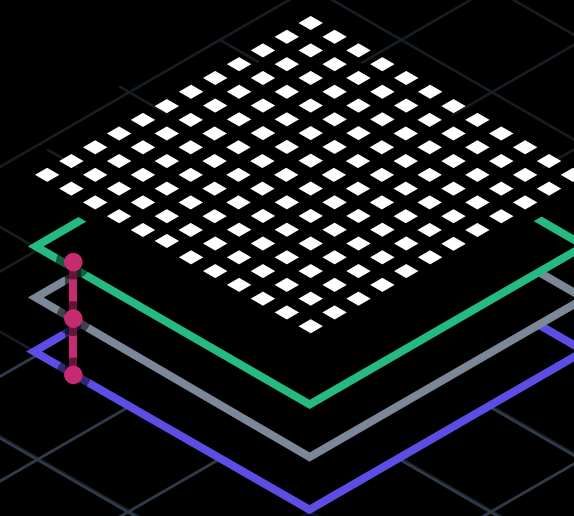
TAMING MODERN CLOUD ENVIRONMENTS WITH TERRAFORM





Nic Jackson

Developer Advocate, HashiCorp





HashiCorp

Terraform

Adgenda



- Why you have a problem
- Quick Terraform intro
- Collaborating with Terraform
 - Remote State
 - Modules
 - GitFlow
 - CI/CD
 - Managing provider secrets



Why you have a problem!

Why you have a problem



AWS Reference Architectures

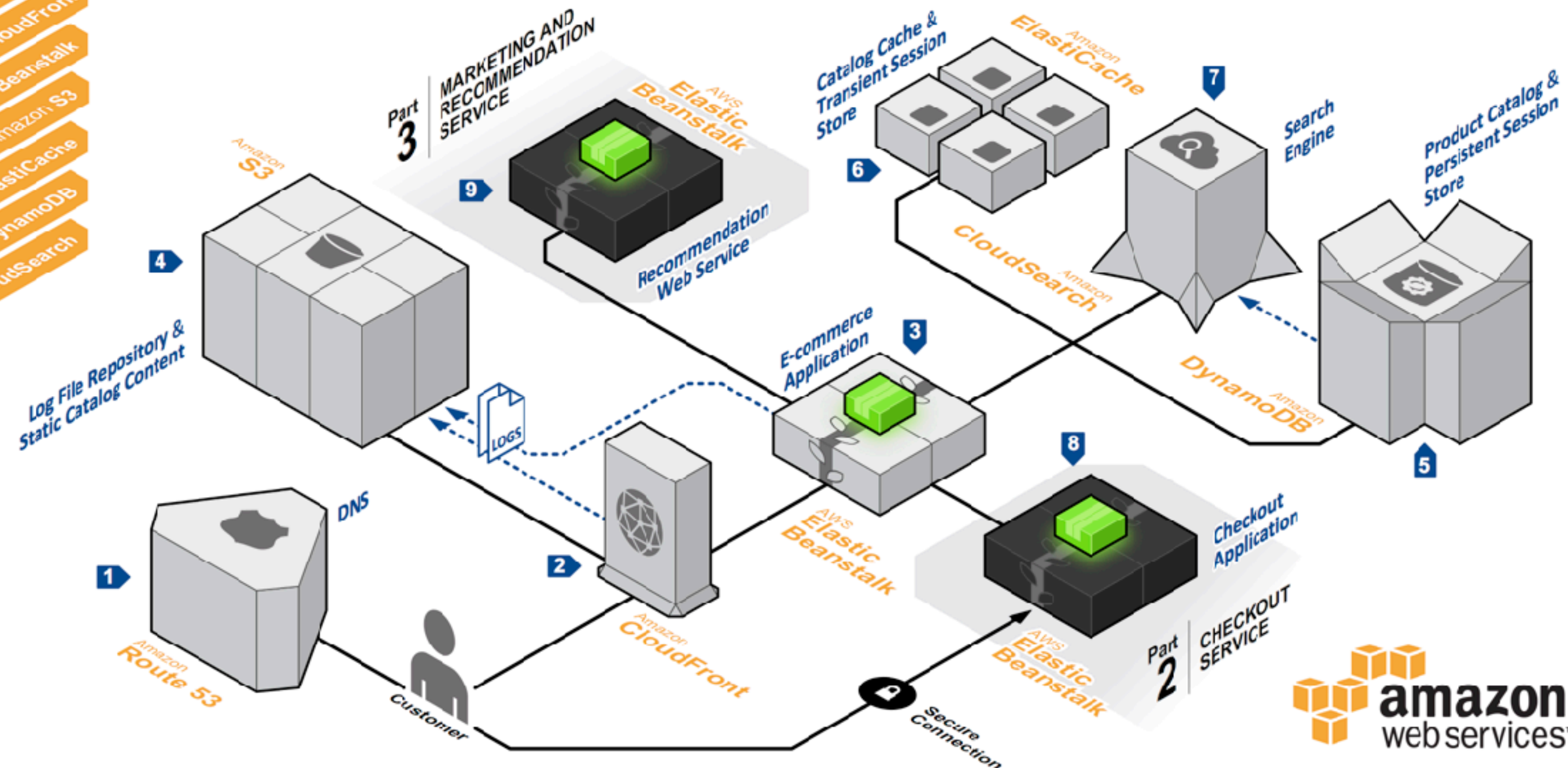
- Amazon Route 53
- Amazon CloudFront
- AWS Elastic Beanstalk
- Amazon S3
- Amazon ElastiCache
- Amazon DynamoDB
- Amazon CloudSearch

E-COMMERCE WEB SITE

PART 1: WEB FRONT-END

With Amazon Web Services, you can build a highly available e-commerce website with a flexible product catalog that scales with your business. Maintaining an e-commerce website with a large product catalog and global customer base can be challenging. The catalog should be searchable, and individual product pages should contain a rich information set that includes, for example, images, a PDF manual, and customer reviews.

Customers want to find the products they are interested in quickly, and they expect pages to load quickly. Worldwide customers want to be able to make purchases at any time, so the website should be highly available. Meeting these challenges becomes harder as your catalog and customer base grow. With the tools that AWS provides, you can build a compelling, scalable website with a searchable product catalog that is accessible with very low latency.



System Overview

- 1** DNS requests to the e-commerce website are handled by **Amazon Route 53**, a highly available Domain Name System (DNS) service.
- 2** **Amazon CloudFront** is a content distribution network (CDN) with edge locations around the globe. It can cache static and streaming content and deliver dynamic content with low latency from locations close to the customer.
- 3** The e-commerce application is deployed by **AWS Elastic Beanstalk**, which automatically handles the details of capacity provisioning, load balancing, auto scaling, and application health monitoring.

- 4** **Amazon Simple Storage Service (Amazon S3)** stores all static catalog content, such as product images, manuals, and videos, as well as all log files and clickstream information from Amazon CloudFront and the e-commerce application.
- 5** **Amazon DynamoDB** is a fully-managed, high performance, NoSQL database service that is easy to set up, operate, and scale. It is used both as a session store for persistent session data, such as the shopping cart, and as the product database. Because DynamoDB does not have a schema, we have a great deal of flexibility in adding new product categories and attributes to the catalog.

- 6** **Amazon ElastiCache** is used as a session store for volatile data and as a caching layer for the product catalog to reduce I/O (and cost) on DynamoDB.
- 7** Product catalog data is loaded into **Amazon CloudSearch**, a fully managed search service that provides fast and highly scalable search functionality.
- 8** When customers check out their products, they are redirected to an SSL-encrypted checkout service.
- 9** A marketing and recommendation service consumes log data stored on Amazon S3 to provide the customer with product recommendations.

Why you have a problem



- Infrastructure as a service (e.g. AWS, Azure, GCP, DigitalOcean)
- Platform as a service (e.g. Heroku)

Why you have a problem



- Many organizations start with PaaS
- Fast default to IaaS as organization grows
- PaaS is amazing, but limitations do not scale well with application complexity

What is Terraform?

What is Terraform? - Write



- Define infrastructure as code to increase operator productivity and transparency
- Configuration can be stored in version control, shared, and collaborated on by teams
- If it can be codified, it can be automated

What is Terraform? - Plan



- Understand how a minor change could have cascading effects across infrastructure before executing the change
- Plans show operators what would happen, applies execute changes
- Single workflow across all infrastructure providers (AWS, GCP, Azure, OpenStack, VMware, and more)

What is Terraform? - Apply



- Use the same configuration in multiple places to reduce mistakes and save time
- Provision identical staging, QA, and production environments
- Effortlessly combine high-level system providers
- Dependency resolution means things happen in the right order

Simple Terraform config

```
terraform.tf

provider "aws" {}

data "aws_availability_zones" "available" {}

resource "aws_vpc" "main" {
  cidr_block = "${var.cidr_block}"
}
```


Collaborating with Terraform

Collaborating with Terraform - 4 phases of collaboration



- Manual
- Semi-automated
- Infrastructure as Code
- Collaborative Infrastructure as Code



Terraform

Remote

State

Collaborating with Terraform - Remote state



- Backends are responsible for storing state and providing an API for [state locking](#). State locking is optional.
- Backends determine where state is stored
- 10 officially supported backends
- Allows state to be stored remotely
- Keeps sensitive information off disk

Remote state backends



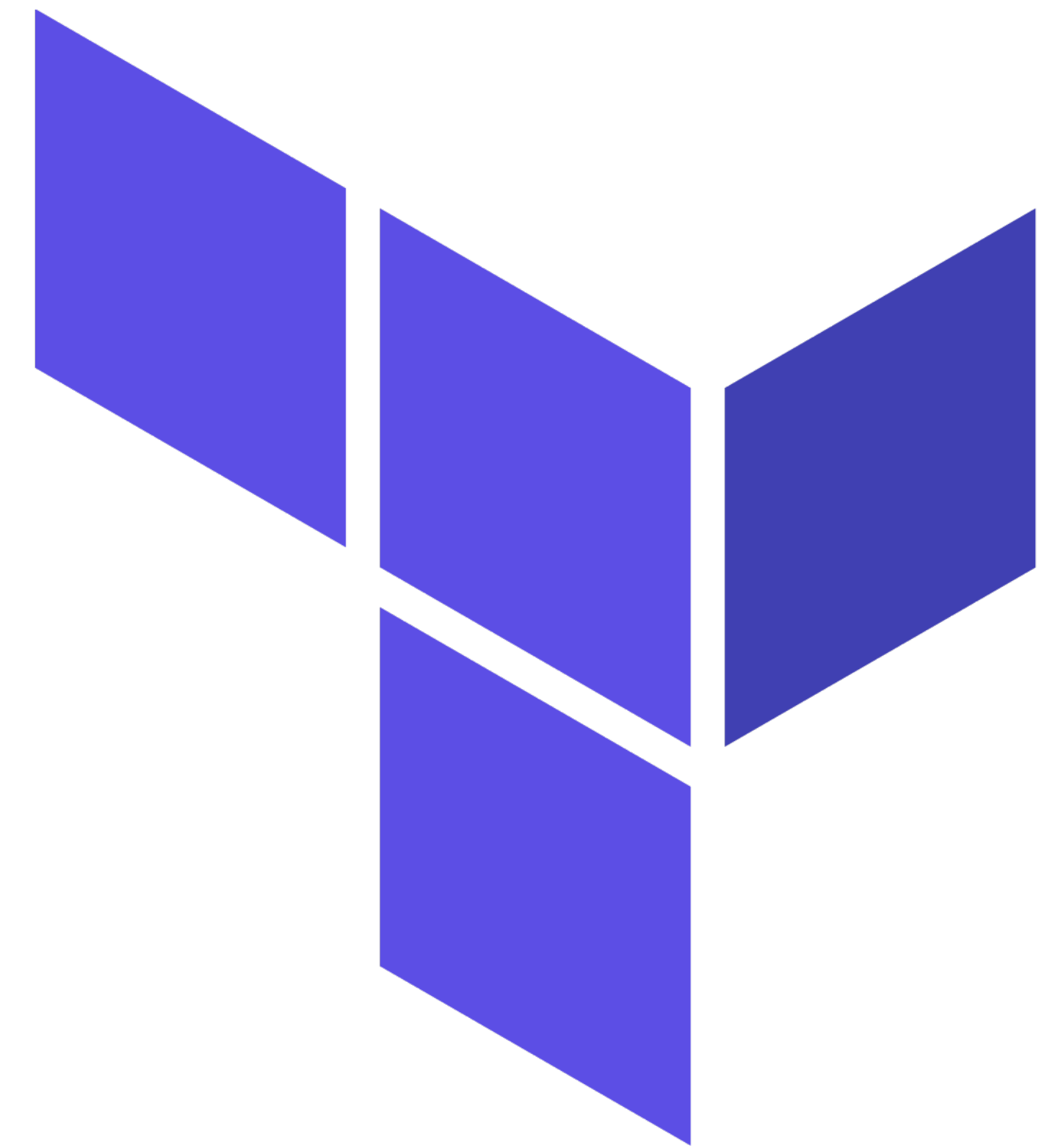
- artifactory - **not** locking
- azurerm - locking
- consul - locking
- etcd - **not** locking
- gcs - locking
- http - **optional** locking
- manta - locking
- s3 - locking
- swift - **not** locking
- terraform enterprise - **not** locking



Terraform remote state

```
terraform {  
  backend "s3" {  
    bucket = "tfremotestatenic"  
    key    = "cfmngmnt"  
    region = "eu-west-1"  
  }  
}
```

Introduction to Modules



Terraform

Introduction to Modules - What is a module?



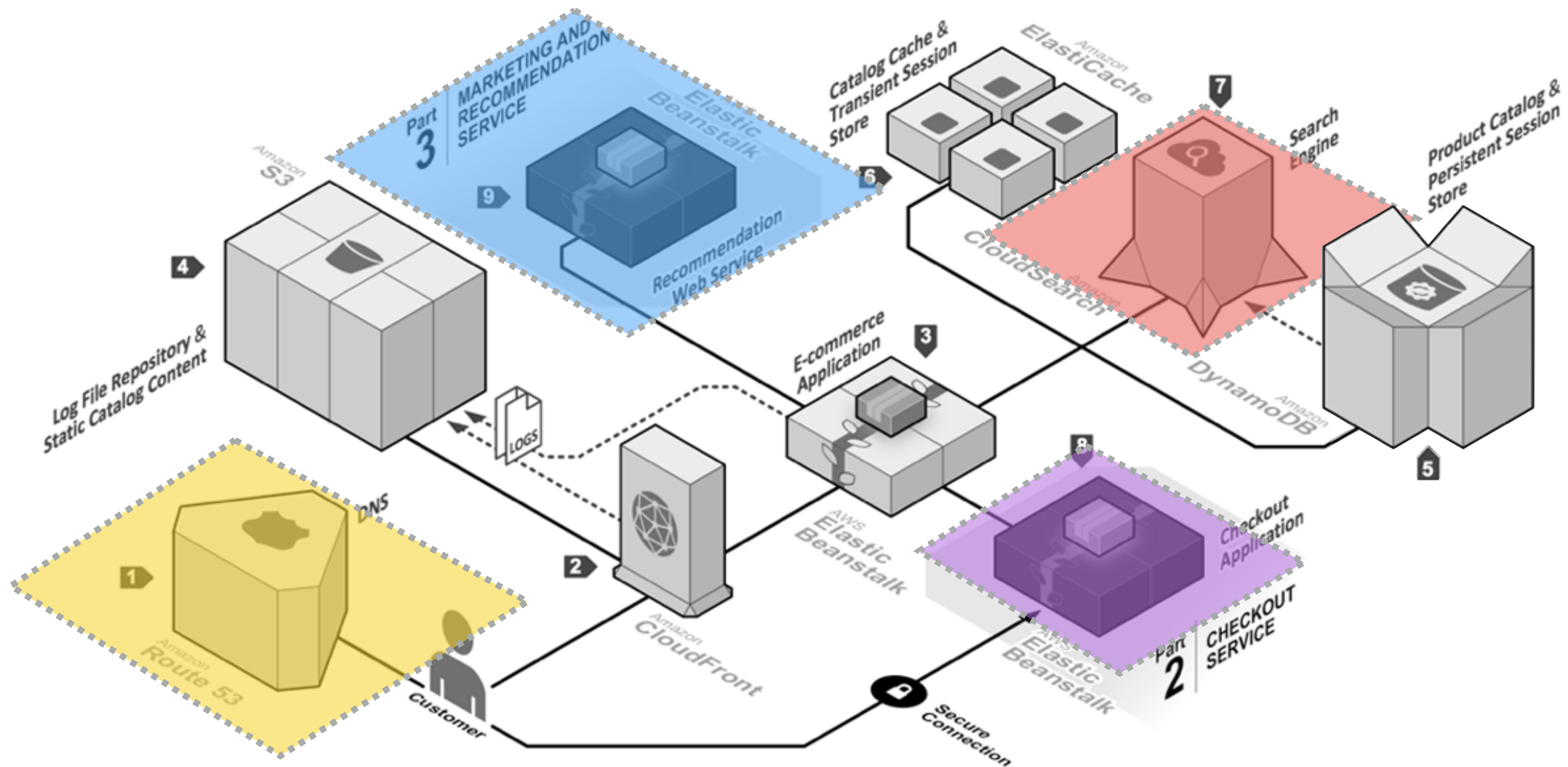
- A module is simply a blueprint for your application
- A module is a high level abstraction
- A module is your own PaaS
- Modules can be shared in Github or the Terraform Module registry

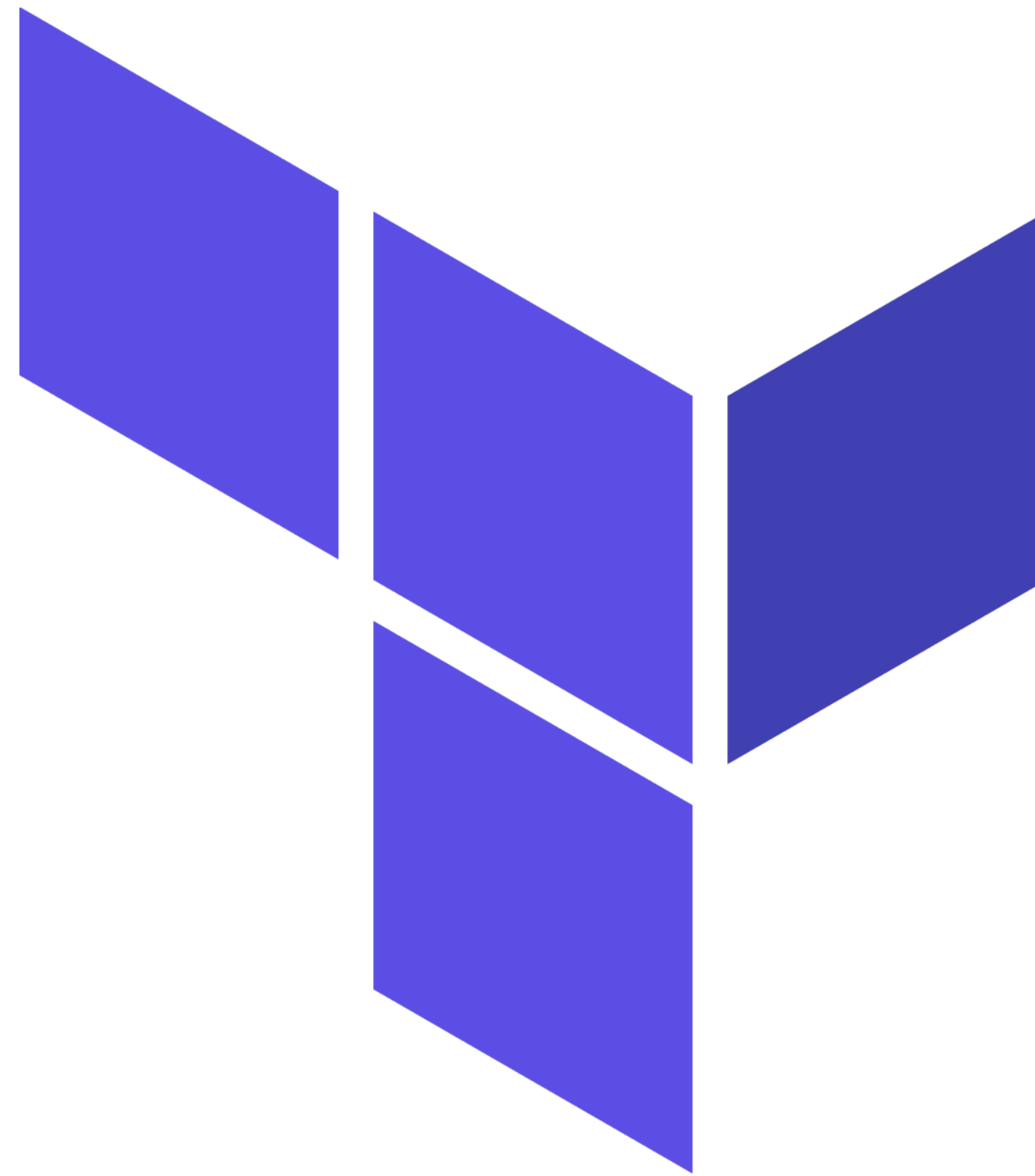
Introduction to Modules - What is a module?



- Modules abstract complexity
- Module perform a single task and have a single purpose
- Build once, use many

Introduction to Modules - Reference Architecture





Terraform

Introduction to Modules - Versioning modules



- Ability to use Git tags
- Move immutable versions across environments

Module source from Git

```
terraform.tf

provider "aws" {}

module "vpc" "default" {
  source = "git@github.com:hashicorp/example.git//subdir"

  cidr_block = "${var.cidr_block}"
}
```

Module source from Git with branch reference

```
terraform.tf

provider "aws" {}

module "vpc" "default" {
  source = "git@github.com:hashicorp/example.git//subdir?ref=hotfix"

  cidr_block = "${var.cidr_block}"
}
```

Introduction to Modules - Modules can contain modules

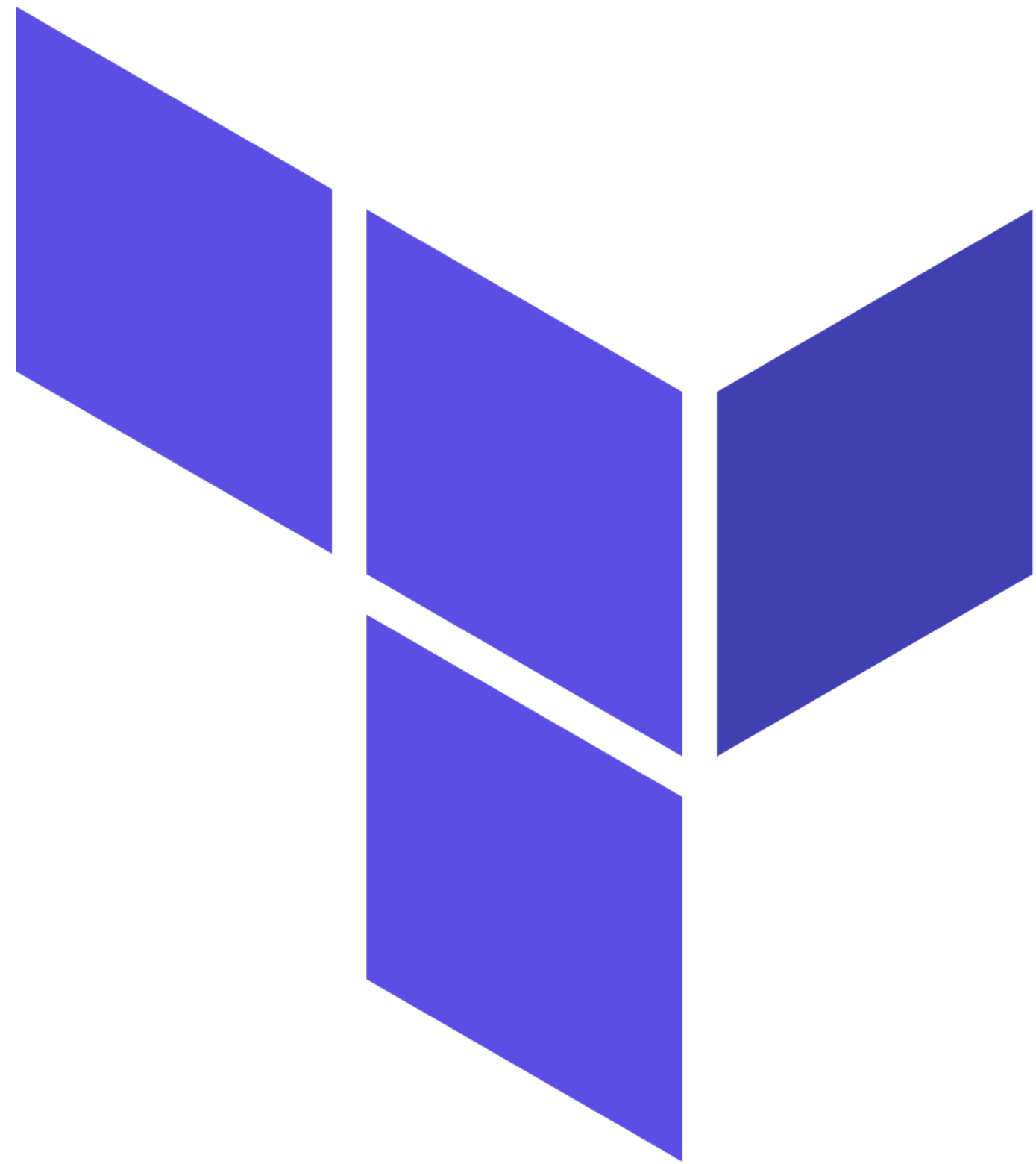


- Modules can be composed of sub-modules
- Sub-modules can be composed of sub-sub-modules
- Sub-sub-modules can be composed of sub-sub-sub-modules
- ...

Introduction to Modules - Multi-Cloud



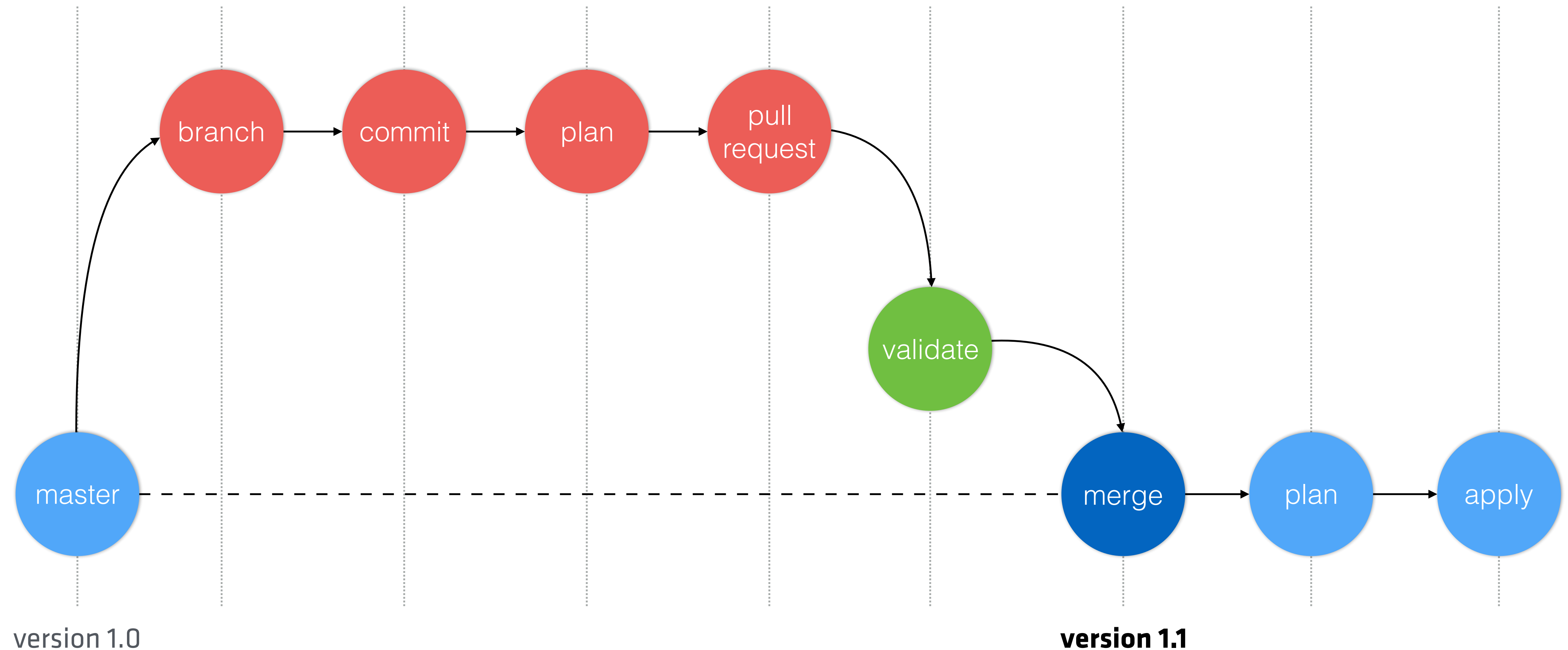
- It will never be possible to write a common abstraction at a resource level for an instance in different cloud providers
- Modeling infrastructure on a high level such as K8s cluster with modules allows abstraction and hides implementation



Terraform

Terraform GitFlow and CI

Collaborating with Terraform - Git Flow



Collaborating with Terraform - GitHub



The screenshot shows the GitHub interface for the repository 'nicholasjackson / terraform-config-management-camp'. The top navigation bar includes links for Features, Business, Explore, Marketplace, and Pricing, along with a search bar and 'Sign in' or 'Sign up' buttons. The repository header shows the name, a 'Watch' button (0), a 'Star' button (0), and a 'Fork' button (0). Below this, tabs for 'Code', 'Issues' (0), 'Pull requests' (1), 'Projects' (0), and 'Insights' are visible. The 'Pull requests' tab is active, showing a search bar with 'is:pr is:open', 'Labels', 'Milestones', and a 'New pull request' button. The list of pull requests shows one open pull request titled 'update CIDR block' with a green checkmark, opened 4 hours ago by 'nicholasjackson'. A 'ProTip!' message suggests adding 'no:assignee' to see everything that's not assigned. The footer includes copyright information for GitHub, Inc. and various links like Terms, Privacy, Security, Status, Help, Contact GitHub, API, Training, Shop, Blog, and About.

Collaborating with Terraform - Pull Requests



update CIDR block #1 Edit

[Open](#) nicholasjackson wants to merge 2 commits into master from updates

Conversation 0 Commits 2 Files changed 2 +7 -1

nicholasjackson commented 4 hours ago Owner

No description provided.

nicholasjackson added some commits 4 hours ago

- update CIDR block verified ✓ a3c64d5
- Added variable for CIDR block verified ✓ f5b1853

Add more commits by pushing to the updates branch on nicholasjackson/terraform-config-management-camp.

All checks have passed Hide all checks
1 successful check

- ci/circleci: plan** — Your tests passed on CircleCI! Details
- This branch has no conflicts with the base branch**
Merging can be performed automatically.

[Merge pull request](#) You can also open this in GitHub Desktop or view command line instructions.

Write **Preview** AA **B** **I** Quote Code Link Image Table Checklist Undo Redo Close

Leave a comment

Attach files by dragging & dropping or [selecting them](#).

Styling with Markdown is supported

[Close pull request](#) [Comment](#)

Reviewers Settings
No reviews—request one

Assignees Settings
No one—assign yourself

Labels Settings
None yet

Projects Settings
None yet

Milestone Settings
No milestone

Notifications
[Unsubscribe](#)
You're receiving notifications because you authored the thread.

1 participant
nicholasjackson

[Lock conversation](#)

Collaborating with Terraform - Automated builds



update CIDR block #1 Edit

Open nicholasjackson wants to merge 2 commits into master from updates

Conversation 0 Commits 2 Files changed 2 +7 -1

nicholasjackson commented 4 hours ago Owner Reviewers No reviews—request one

No description

Commit history

- nicholasjackson
- updates
- Added

Add more commits

Share All checks have passed 1 successful check Hide all checks

ci/circleci: plan — Your tests passed on CircleCI! Details

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also open this in GitHub Desktop or view command line instructions.

Write Preview

Leave a comment








Attach files by dragging & dropping or selecting them.

Styling with Markdown is supported Close pull request Comment



Collaborating with Terraform - Build workflow





Workflows » nicholasjackson » terraform-config-management-camp » master » 86a0997e-e089-412f-8276-9ebf26b32b73

<div> SUCCEEDED</div> <div> Rerun </div>	<div>master / plan-and-apply</div> <div> first draft circle config</div>	<div> 4 hrs ago</div> <div> 01:13</div> <div> 3b92006</div>
--	---	--

2 jobs in this workflow

 plan  00:33

 apply  00:33

Collaborating with Terraform - Plan



```
0 (00:33)
mkdir plan 00:00

Terraform plan 00:22

$ #!/bin/bash -eo pipefail
  export VAULT_TOKEN=$(vault login -token-only -method=github token=${GITHUB_TOKEN})
  envconsul -secret aws/creds/aws_readonly -once ./run_terraform.sh plan -out-plan/plan.out

The token was not stored in token helper. Set the VAULT_TOKEN environment
variable or pass the token below with each request to Vault.

2018/02/01 09:32:32.561279 Looking at vault aws/creds/aws_readonly

Initializing the backend...

Successfully configured the backend 's3"! Terraform will automatically
use this backend unless the backend configuration changes.

Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "aws" (1.8.3)...

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 = '~> 1.8'

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_vpc.main: Refreshing state... (ID: vpc-7c719707)

-----

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

Collaborating with Terraform - Apply



0 (00:33) +

TEST

Spin up Environment 00:07

Checkout code 00:00

Attaching Workspace 00:00

Terraform apply 00:24

```
$ #!/bin/bash -eo pipefail
export VAULT_TOKEN=$(vault login -token-only -method=github token=${GITHUB_TOKEN})
envconsul -secret aws/creds/aws_write -once ./run_terraform.sh apply /tmp/workspace/plan/plan.out

The token was not stored in token helper. Set the VAULT_TOKEN environment
variable or pass the token below with each request to Vault.

2018/02/01 09:33:11.482927 looking at vault aws/creds/aws_write

Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.

Installing provider plugins...
Checking for available provider versions (https://releases.hashicorp.com/).
Installing provider aws (2.20.0).

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 = "~> 1.8"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

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

ONLY PRODUCTION BRANCH / ENFORCE TAGS

Public Service Announcement!



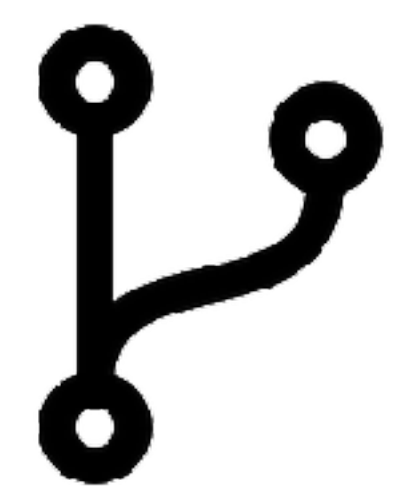
- **DO NOT** EXPOSE Vault / CI WITH PUBLIC GitHub
- THIS IS A SIMPLIFIED EXAMPLE FOR DEMONSTRATION ONLY



Collaborating with Terraform - Fork me



- 1) Fork the below repository
- 2) Create a new branch
- 3) Refactor into a module**
- 4) Pull request



[https://github.com/nicholasjackson/
terraform-cfgmgmtcamp](https://github.com/nicholasjackson/terraform-cfgmgmtcamp)

Managing Credentials With Vault



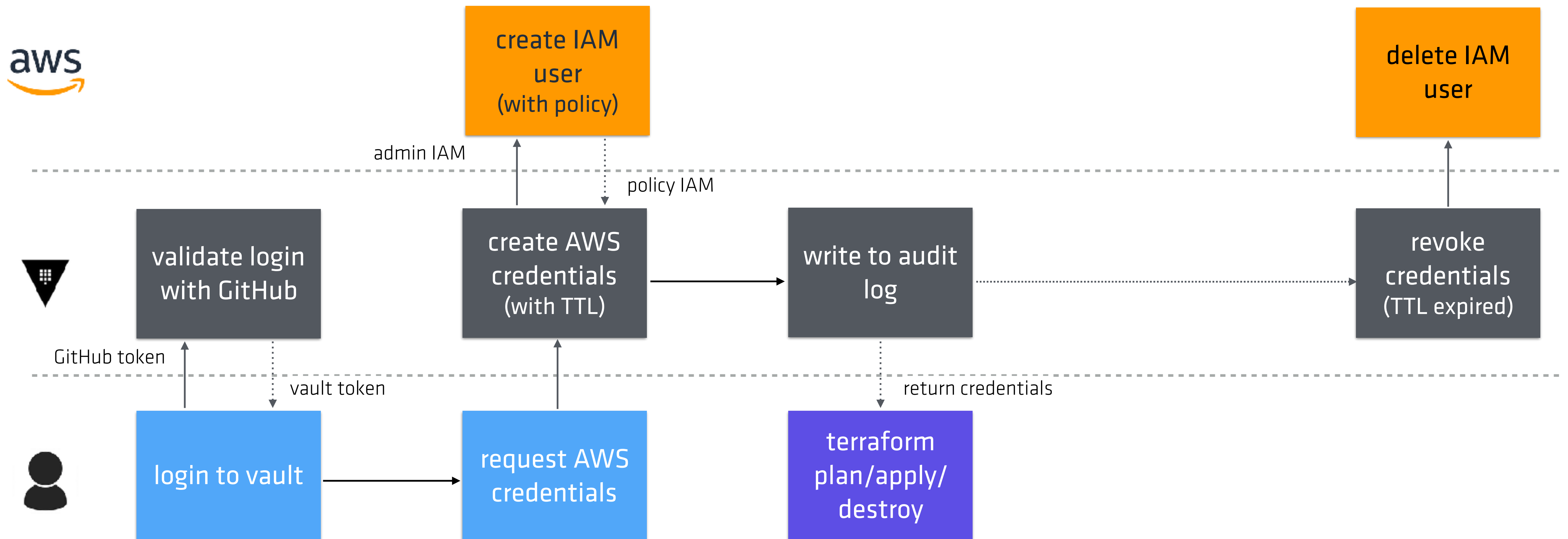
Terraform

Managing Credentials with Vault - Why?

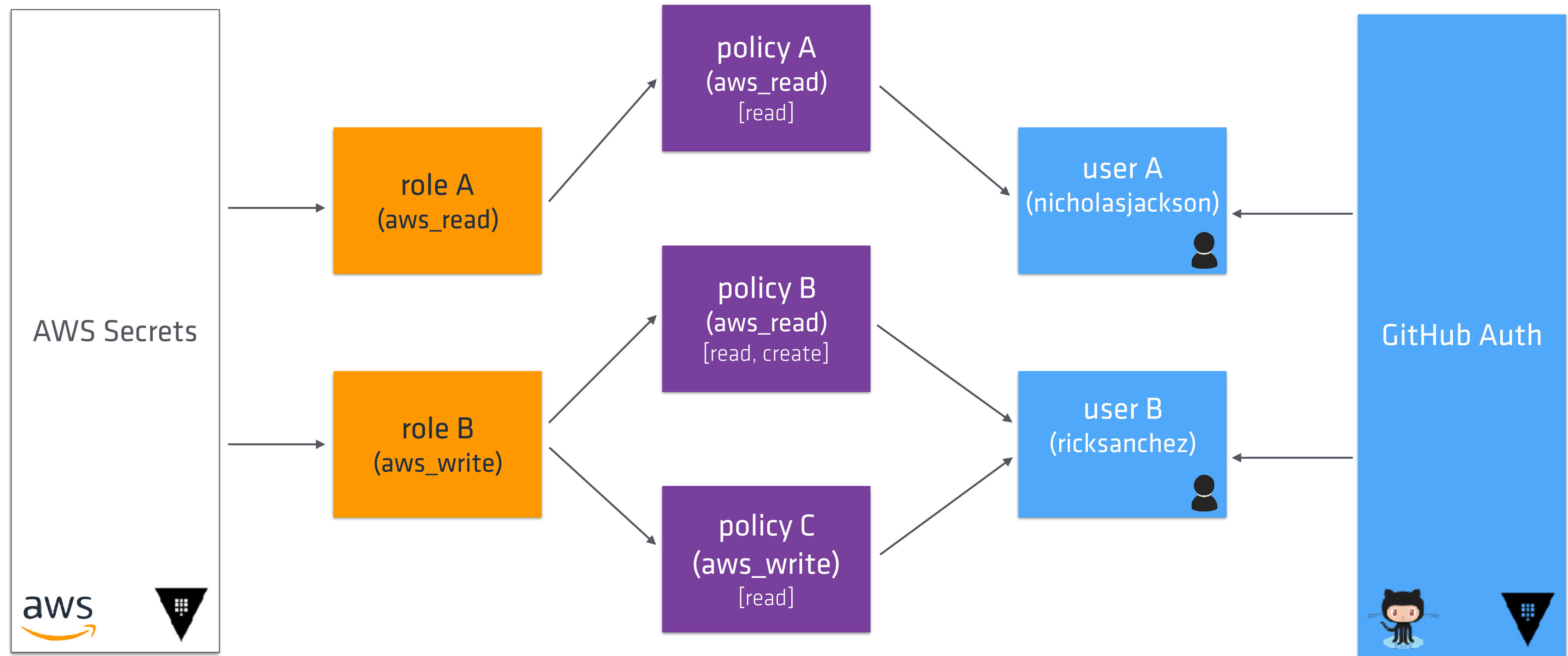


- Separating read / write credentials
- TTL on credentials with automatic revoke
- Fine grained control over access
- Audit log
- Reduce secret sprawl
- Developers / operators **DO NOT** require explicit credentials

Managing Credentials with Vault - Process



Managing Credentials with Vault - hierarchy



Configure AWS secret engine

```
Terminal

# enable the vault secrets engine for AWS
$ vault secrets enable aws
Success! Enabled the aws secrets engine at: aws/

# add IAM credentials which have the capability to manage IAM credentials
$ vault write aws/config/root \
  access_key=AKIAJWVN5Z4F0FT7NLNA \
  secret_key=R4nm063hgMVo4BTT5x0s5nHLeLXA6lar7ZJ3Nt0i \
  region=us-east-1
```

Configure GitHub auth

```
Terminal

# enable the vault secrets engine for AWS
$ vault auth enable github

# set the auth endpoint to a GitHub organization
$ vault write auth/github/config organization=hashicorp
```

Create a read-only AWS IAM policy

```
terraform.tf

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::tfremotestatenic"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": ["arn:aws:s3:::tfremotestatenic/*"]
    },
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "elasticloadbalancing:Describe*",
      "Resource": "*"
    },
    ...
  ]
}
```

Create a write AWS IAM policy

```
terraform.tf

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::tfremotestatenic"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject"
      ],
      "Resource": ["arn:aws:s3:::tfremotestatenic/*"]
    },
    {
      "Action": "ec2:*",
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "elasticloadbalancing:*",
      "Resource": "*"
    },
  ],
}
```

Configure roles

```
Terminal

# create the read only role
$ vault write aws/roles/aws_readonly policy=@aws_read_policy.json
Success! Data written to: aws/roles/aws_readonly

# create the write role
$ vault write aws/roles/aws_write policy=@aws_write_policy.json
Success! Data written to: aws/roles/aws_write
```


Create a read-only Vault policy

```
terraform.tf

path "aws/creds/aws_readonly" {
  capabilities = ["read"] # "create", "read", "update", "delete", "list"
}
```

Create a write Vault policy

```
path "aws/creds/aws_write" {  
  capabilities = ["read"]  
}
```

Add the policy to Vault

```
Terminal

# Add the read only policy to vault
$ vault write sys/policy/aws_readonly policy=@aws_readonly.hcl
Success! Data written to: sys/policy/aws_readonly

# Add the write policy to vault
$ vault write sys/policy/aws_write policy=@aws_write.hcl
Success! Data written to: sys/policy/aws_write
```

Assign the policy to the GitHub user

```
Terminal

# Assign the policy to the GitHub user
$ vault write auth/github/map/users/nicholasjackson
value=aws_readonly,aws_write
Success! Data written to: auth/github/map/users/nicholasjackson

# Or assign the policy to a GitHub team
$ vault write auth/github/map/teams/hashicorp value=aws_readonly,aws_write
Success! Data written to: auth/github/map/teams/hashicorp
```

Login with GitHub obtain IAM credentials

```
Terminal

# login to vault with GitHub token
$ vault login -token-only -method=github token=${GITHUB_TOKEN}
The token was not stored in token helper. Set the VAULT_TOKEN environment
variable or pass the token below with each request to Vault.

ebb39457-cdb8-41ae-c227-9e0245837873

# use vault token to obtain AWS credentials
$ VAULT_TOKEN=ebb39457-cdb8-41ae-c227-9e0245837873 vault read aws/creds/
aws_readonly
Key                               Value
---                               -
lease_id                         aws/creds/aws_readonly/0bb798a3-c4fd-0609-6bcd-
f34673156c8a
lease_duration                   15m
lease_renewable                  true
access_key                      AKIAJMWSOATIG3VQJCNA
secret_key                      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
security_token                  <nil>
```

We can also
do this with
a module

```
provider "vault" {  
  address = "http://pi01.local.demo.gs:8200"  
}  
  
module "vault" {  
  source = "./vault"  
  
  github_org      = "hashicorp"  
  
  aws_access_key_id      = "xxxxxxxxxxxxxxxx"  
  aws_access_key_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"  
  aws_region            = "us-east-1"  
}
```

Putting it all together

THANKS FOR LISTENING

 @sheriffjackson, nic@hashicorp.com

FURTHER INFO:

- <https://www.terraform.io>
- <https://www.vaultproject.io>
- <https://registry.terraform.io>

EXAMPLE PROJECT:

- <https://github.com/nicholasjackson/terraform-cfgmgmtcamp>

AGENDA

14:55 - CoreOS baremetal provisioning with Terraform, Ignition and Matchbox

★ Rafael Porres Molina

15:35 - Break

15:55 - Introduction to provisioning basic infrastructure on Google Cloud Platform with Terraform

★ Stein Inge Morisbak

16:35 - Terraform for fun and profit

★ Bram Vogelaar & Julien Pivottto

17:15 - Terraboard

★ Raphael Pinson