

Introduction to Terraform: IaC

101.1 What is Terraform?

101.2 Key Features of Terraform

101.3 Pros and Cons

101.4 Providers

101.5 Basic of Infrastructure

101.6 Getting Started

Samsung Electronics, Staff Engineer

Global Service Reliability Engineer

Jay Kim

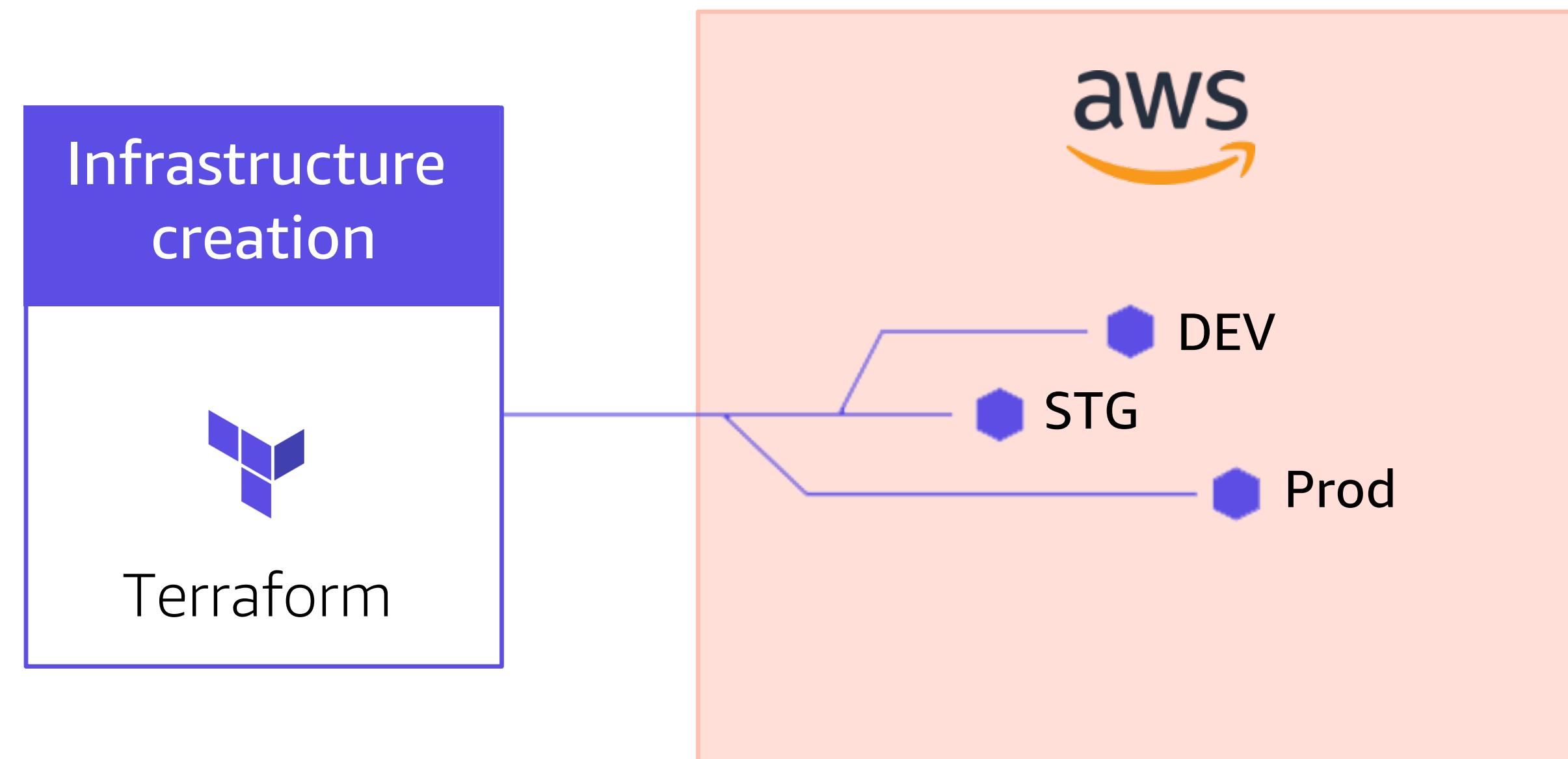
101.1 What is Terraform?



Use Infrastructure as Code(IaC) to provision and manage any cloud, infrastructure, or service



Use Infrastructure as Code(IaC) to provision and manage any cloud, infrastructure, or service





Use Infrastructure as Code(IaC) to provision and manage any cloud, infrastructure, or service

The screenshot shows the AWS Management Console interface. At the top, there's a dark header bar with the AWS logo, a '서비스' (Services) dropdown, a '리소스 그룹' (Resource Groups) dropdown, a bell icon for notifications, and a user email 'rpkim.jay@gmail.com'. Below the header, on the left, is a sidebar with links like '내역' (History), 'EC2', '콘솔 홈' (Console Home), 'IAM', 'DynamoDB', 'S3', 'ECR', and 'EKS'. The main content area has a search bar with Korean placeholder text '이름 또는 기능(예: EC2, S3 또는 VM, 스토리지)으로 서비스를 찾습니다.' (Search by name or function (e.g., EC2, S3 or VM, storage)). To the right of the search bar are four service categories: '컴퓨팅' (Computing) with EC2, Lightsail, ECR, ECS, EKS, Lambda, and Batch; '고객 지원' (Customer Support) with AWS IQ, 지원 (Support), Managed Services, and Amazon Managed Blockchain; '분석' (Analytics) with Athena, EMR, CloudSearch, Elasticsearch Service, Kinesis, QuickSight, and Data Pipeline; and '블록체인' (Blockchain) with Amazon Managed Blockchain.

컴퓨팅	고객 지원	분석
EC2	AWS IQ	Athena
Lightsail	지원	EMR
ECR	Managed Services	CloudSearch
ECS		Elasticsearch Service
EKS		Kinesis
Lambda	Amazon Managed Blockchain	QuickSight
Batch		Data Pipeline



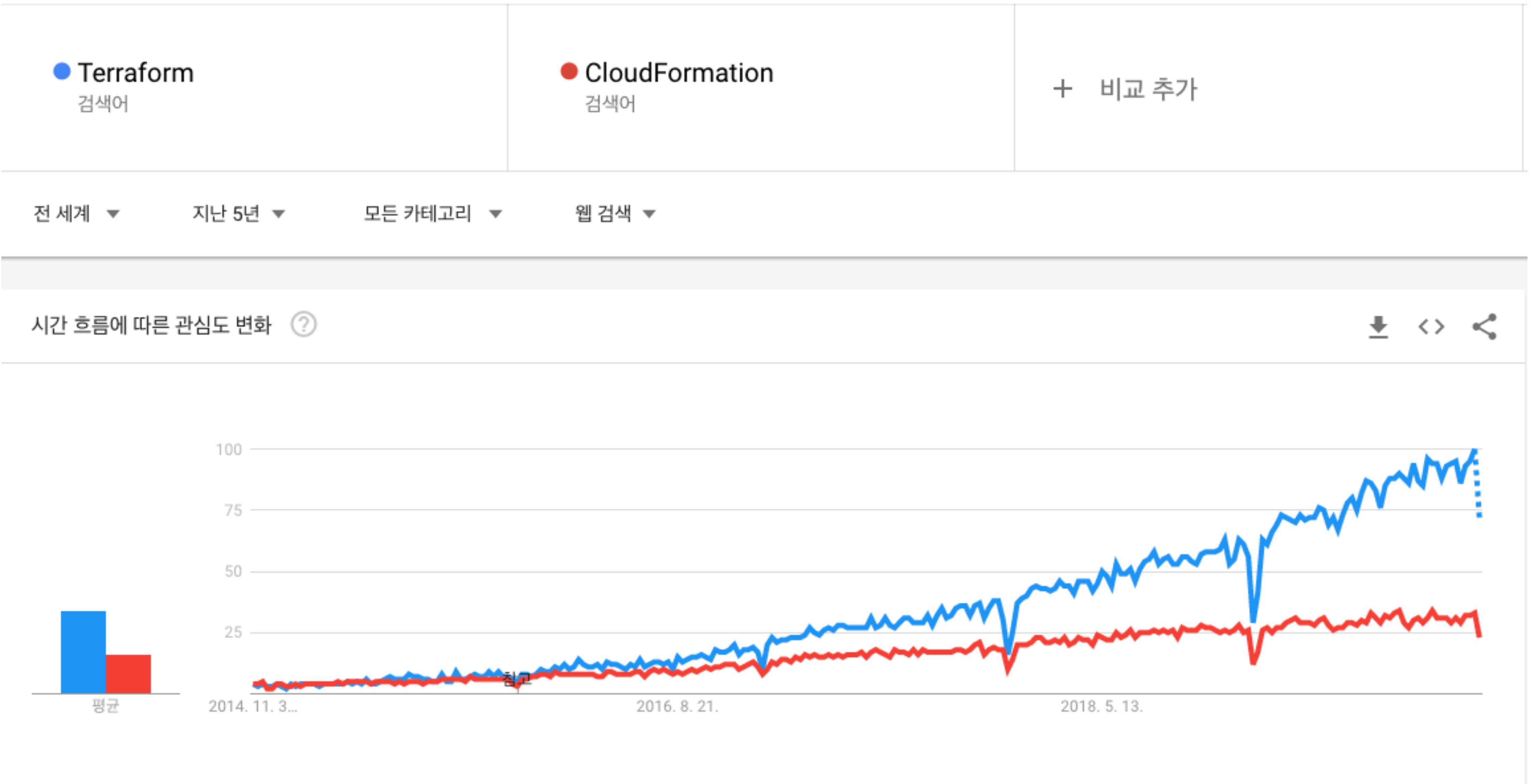
Use Infrastructure as Code(IaC) to provision and manage any cloud, infrastructure, or service

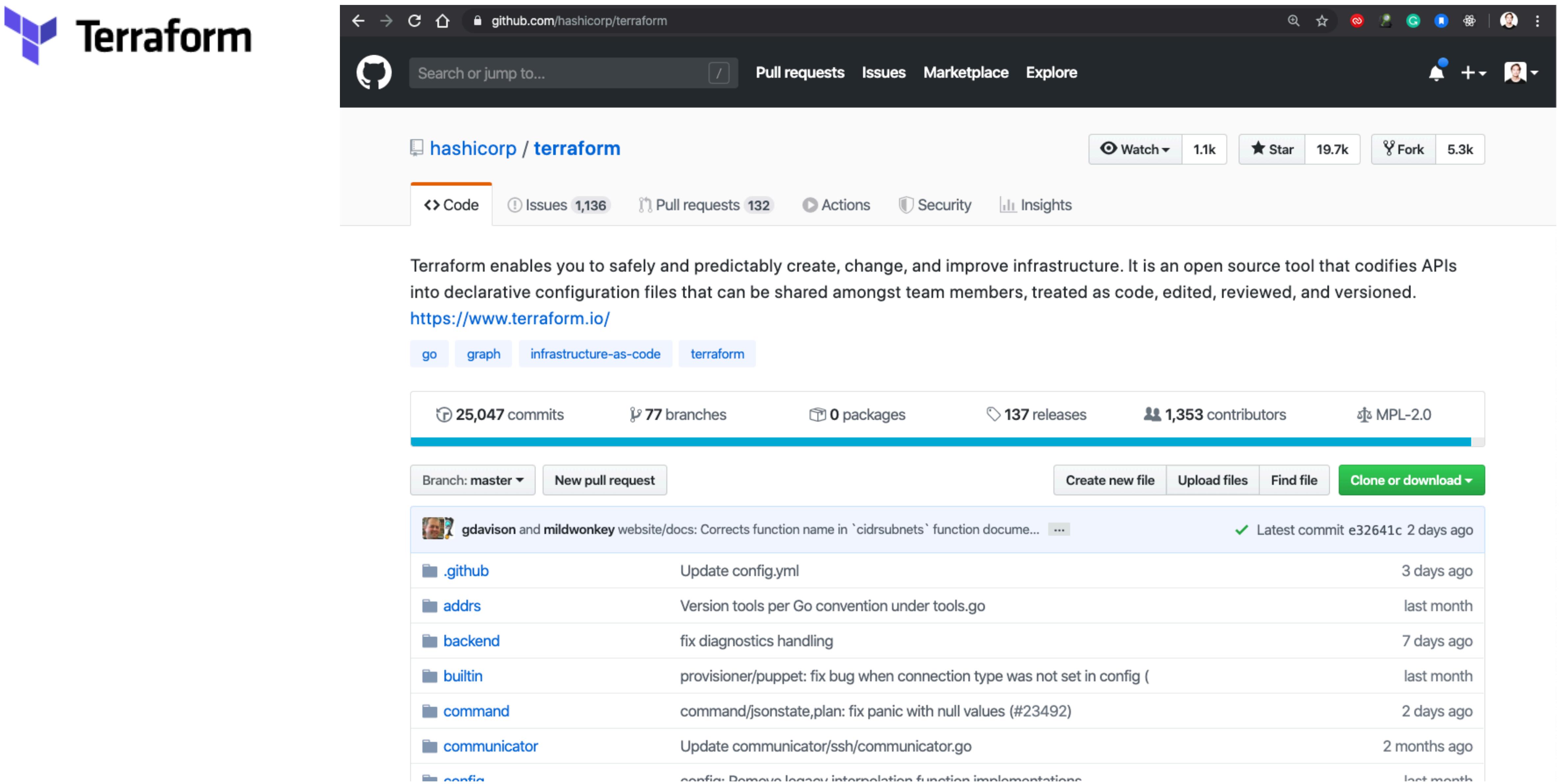
```
resource "aws_vpc" "vpc" {
    cidr_block          = "${var.cidr_block}"                                HCL
    enable_dns_hostnames = true
    enable_dns_support   = true

    tags = {
        Name      = "${var.name}"
        terraform = "true"
    }
}

resource "aws_subnet" "public" {
    count                  = "${length(var.public_subnets)}"
    map_public_ip_on_launch = true
    vpc_id                 = "${aws_vpc.vpc.id}"
    availability_zone       = "${element(var.zones, count.index)}"
    cidr_block              = "${element(var.public_subnets, count.index)}"
```

(Hashicorp Configuration Language (HCL))





The screenshot shows the GitHub repository page for Terraform, maintained by hashicorp. The page includes the Terraform logo, a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. Key statistics are displayed: 25,047 commits, 77 branches, 0 packages, 137 releases, 1,353 contributors, and an MPL-2.0 license. A recent commit from gdavison and mildwonkey is highlighted. The repository is open-source, with 1.1k stars and 5.3k forks.

Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.

<https://www.terraform.io/>

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

Commit	Message	Date
 gdavison and mildwonkey	website/docs: Corrects function name in `cidrsubnets` function docume...	Latest commit e32641c 2 days ago
 .github	Update config.yml	3 days ago
 addrs	Version tools per Go convention under tools.go	last month
 backend	fix diagnostics handling	7 days ago
 builtin	provisioner/puppet: fix bug when connection type was not set in config (last month
 command	command/jsonstate,plan: fix panic with null values (#23492)	2 days ago
 communicator	Update communicator/ssh/communicator.go	2 months ago
 config	config: Remove legacy interpolation function implementations	last month



CLOUD CLI

The screenshot shows the Terraform Cloud interface. At the top, there are tabs for 'Cloud' and 'CLI'. Below the tabs, there's a navigation bar with links for 'Dashboard', 'Workspaces', 'Modules', 'Settings', 'Documentation', and 'Status'. A breadcrumb trail indicates the current workspace and run details. The main area shows a run titled 'aws-vpc' in 'PLANNING' status, labeled 'Queued manually in Terraform Cloud'. It lists three steps: 'Plan queued', 'Apply pending', and a note about cancelability. At the bottom, there are 'Cancel Run' and 'Add Comment' buttons.

CLOUD CLI

The screenshot shows the Terraform CLI terminal. It displays the output of the 'terraform init' command. The output includes a note about provider version constraints, the command itself, a success message in green, and instructions for working with Terraform. The terminal prompt 'terraformUser\$' is at the bottom.

```
corresponding provider blocks in configuration, with the constraint strings suggested below.

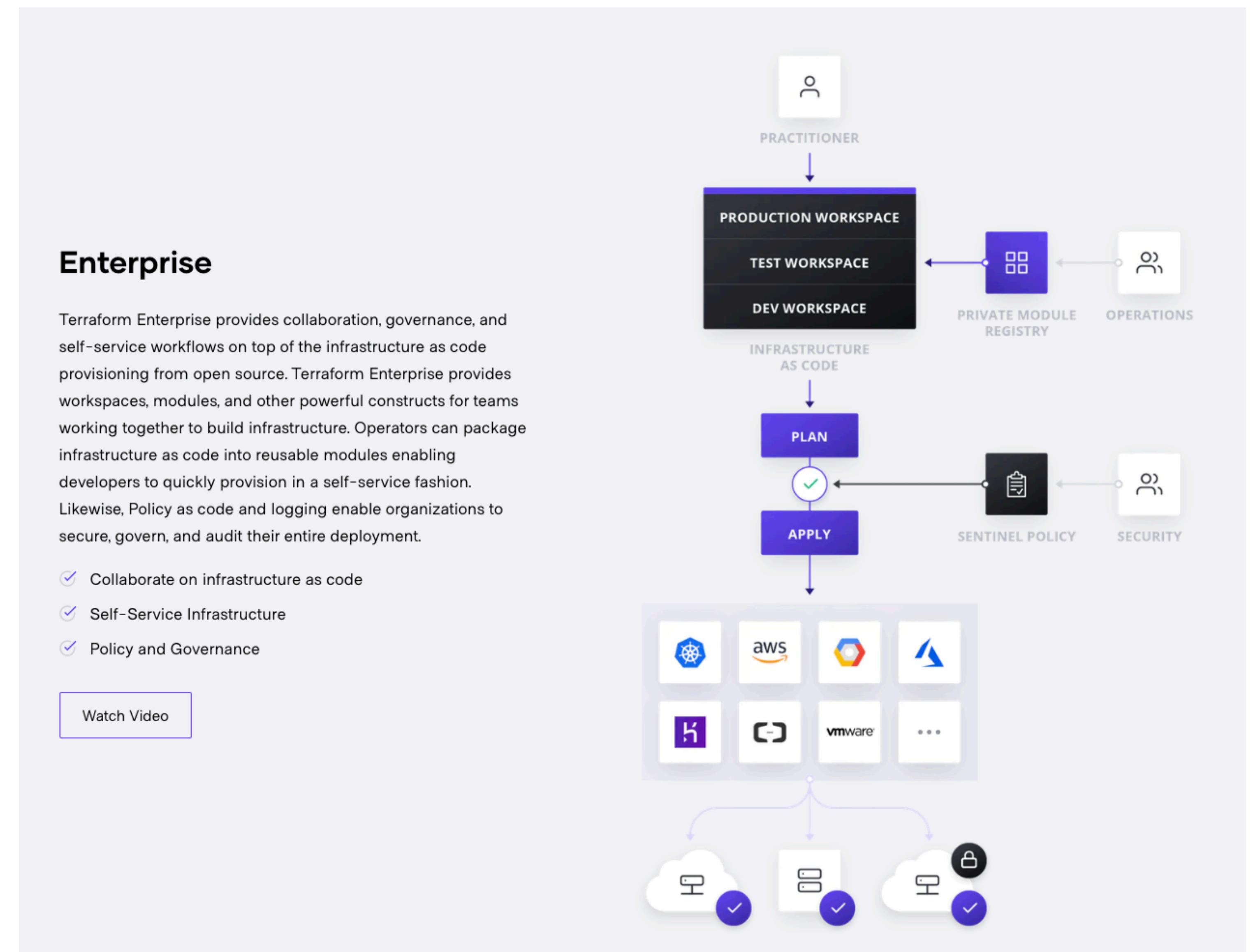
* provider.aws: version = "~> 2.23"

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.

terraformUser$
```









Mitchell Hashimoto(26)
co-founder and CEO

Armon Dadgar(24)
co-founder and CTO



Mitchell Hashimoto(26)
co-founder and CEO

Armon Dadgar(24)
co-founder and CTO

PRODUCTS

-  Terraform
-  Vault
-  Consul
-  Nomad
-  Vagrant
-  Packer



JUNE 6, 2013

Automation Obsessed

Someone asked me recently to share some previous work I had done. As I looked back at my personal history, and reflected on what I've done successfully, it was clear to me that I'm definitely working on things I'm meant to be working on. Automation has been my passion since the very beginning.

I started programming because I wanted to cheat Neopets. I played Neopets daily, and I was frustrated that I would do the same things daily for hours for the same measly amount of Neopoints (the virtual currency used by Neopets). I learned Visual Basic with the goal of writing a program to do those tasks for me. I succeeded at this, and was soon making my measly

PRODUCTS

- Terraform
- Vault
- Consul
- Nomad
- Vagrant
- Packer



Mitchell Hashimoto(26)

co-founder and CEO

Recently, I started HashiCorp. And while it isn't publicly clear what I'm up to yet, and I'm not quite ready to talk about everything I'm up to, you can be sure of one thing: I'm busy automating. More specifically, I'm developing software to automate everything, and giving it to you.

PRODUCTS

-  Terraform
-  Vault
-  Consul
-  Nomad
-  Vagrant
-  Packer

Terraform is, the result of
“Engineering Thinking”

Do Coding For Everything
(Automation)

101.2 Key Features of Terraform

Write declarative configuration files

- Collaborate and share configurations
- Evolve and version your infrastructure
- Automate provisioning

Define infrastructure as code to manage the full lifecycle — create new resources, manage existing ones, and destroy those no longer needed.

```
variable "base_network_cidr" {  
  default = "10.0.0.0/8"  
}  
  
resource "google_compute_network" "example" {  
  name          = "test-network"  
  auto_create_subnetworks = false  
}  
  
resource "google_compute_subnetwork" "example" {  
  count = 4  
  
  name          = "test-subnetwork"  
  ip_cidr_range = cidrsubnet(var.base_network_cidr, 4, count.index)  
  region        = "us-central1"  
  network       = google_compute_network.example.self_link  
}
```

```
• • •  
Terraform will perform the following actions:  
  
# kubernetes_pod.example will be updated in-place  
~ resource "kubernetes_pod" "test" {  
    id = "default/terraform-test"  
  
    metadata {  
        generation      = 0  
        labels         = {  
            "app" = "MyApp"  
        }  
        name           = "terraform-test"  
        namespace      = "default"  
        resource_version = "650"  
        self_link       = "/api/v1/namespaces/default/pods/terraform-test"  
        uid             = "5130ef35-7c09-11e9-be7c-080027f59de6"  
    }  
  
~ spec {
```

Plan and predict changes

- Clearly mapped resource dependencies
- Separation of plan and apply
- Consistent, repeatable workflow

Terraform provides an elegant user experience for operators to safely and predictably make changes to infrastructure.

Create reproducible infrastructure

- Reproducible production, staging, and development environments
- Shared modules for common infrastructure patterns
- Combine multiple providers consistently

Terraform makes it easy to re-use configurations for similar infrastructure, helping you avoid mistakes and save time.

[View All Providers](#)

Microsoft
Azure



aws



Google Cloud Platform

vmware®

Alibaba Cloud

ORACLE®



DATADOG



200+ more

Modules

JUMP TO SECTION ▾

Note: This page is about Terraform 0.12 and later. For Terraform 0.11 and earlier, see [0.11 Configuration](#)

Language: [Modules](#).

A *module* is a container for multiple resources that are used together.

Every Terraform configuration has at least one module, known as its *root module*, which consists of the resources defined in the `.tf` files in the main working directory.

A module can call other modules, which lets you include the child module's resources into the configuration in a concise way. Modules can also be called multiple times, either within the same configuration or in separate configurations, allowing resource configurations to be packaged and re-used.

This page describes how to call one module from another. Other pages in this section of the documentation describe the different elements that make up modules, and there is further information about how modules can be used, created, and published in [the dedicated *Modules* section](#).

Write declarative configuration files

- Collaborate and share configurations
- Evolve and version your infrastructure
- Automate provisioning

Define infrastructure as code to manage the lifecycle — create new resources, manage ones, and destroy those no longer needed.

Modules

JUMP TO SECTION ▾

Note: This page is about Terraform 0.12 and later. For Terra

Language: [Modules](#).

A *module* is a container for multiple resources that are used together.

Every Terraform configuration has at least one module, known as the root module. It contains the top-level resources defined in the `.tf` files in the main working directory.

A module can call other modules, which lets you include the same configuration in multiple places in a concise way. Modules can also be called multiple times, either from different configurations or from within a single configuration, allowing resource configurations to be packaged and reused.

This page describes how to call one module from another. Other pages describe the different elements that make up modules, and the process for creating and publishing them. Modules can also be used, created, and published in [the dedicated Modules section](#).

Create reproducible infrastructure

- Reproducible production, staging, and development environments
- Shared modules for common infrastructure patterns
- Combine multiple providers consistently

Terraform makes it easy to re-use configuration across multiple projects and environments to build similar infrastructure, helping you avoid mistakes and save time.

[View All Providers](#)

These features are very similar with 

Declarative

 makes it painless to create interactive UIs. Design simple views for each state in your application, and  will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in  without rewriting existing code.

 can also render on the server using Node and power mobile apps using 

These features are very similar with **React**

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

React can also render on the server using Node and power mobile apps using [React Native](#).

Terraform is an infrastructure provisioning tool
but the concepts aren't only for SysOps Engineer.

*.examples of declarative

Using a Jenkinsfile

This section builds on the information covered in [Getting started with Pipeline](#) and introduces more useful steps, common patterns, and demonstrates some non-trivial [Jenkinsfile](#) examples.

Creating a [Jenkinsfile](#), which is checked into source control ^[1], provides a number of immediate benefits:

- Code review/iteration on the Pipeline
- Audit trail for the Pipeline
- Single source of truth ^[2] for the Pipeline, which can be viewed and edited by multiple members of the project.

Pipeline supports [two syntaxes](#), Declarative (introduced in Pipeline 2.5) and Scripted Pipeline. Both of which [support building continuous delivery pipelines](#). Both may be used to define a Pipeline in either the web UI or with a [Jenkinsfile](#), though it's generally considered a best practice to create a [Jenkinsfile](#) and check the file into the source control repository.

Using Helm and Kustomize to Build More [Declarative Kubernetes Workloads](#)

[Argo CD - Declarative GitOps CD for Kubernetes](#)

101.2 Key Features of Terraform

Write declarative configuration files

- Collaborate and share configurations
- Evolve and version your infrastructure
- Automate provisioning

Define infrastructure as code to manage the full lifecycle — create new resources, manage existing ones, and destroy those no longer needed.

```
variable "base_network_cidr" {  
  default = "10.0.0.0/8"  
}  
  
resource "google_compute_network" "example" {  
  name          = "test-network"  
  auto_create_subnetworks = false  
}  
  
resource "google_compute_subnetwork" "example" {  
  count = 4  
  
  name          = "test-subnetwork"  
  ip_cidr_range = cidrsubnet(var.base_network_cidr, 4, count.index)  
  region        = "us-central1"  
  network       = google_compute_network.example.self_link  
}
```

```
• • •  
Terraform will perform the following actions:  
  
# kubernetes_pod.example will be updated in-place  
~ resource "kubernetes_pod" "test" {  
    id = "default/terraform-test"  
  
    metadata {  
        generation      = 0  
        labels         = {  
            "app" = "MyApp"  
        }  
        name           = "terraform-test"  
        namespace      = "default"  
        resource_version = "650"  
        self_link       = "/api/v1/namespaces/default/pods/terraform-test"  
        uid             = "5130ef35-7c09-11e9-be7c-080027f59de6"  
    }  
  
~ spec {
```

Plan and predict changes

- Clearly mapped resource dependencies
- Separation of plan and apply
- Consistent, repeatable workflow

Terraform provides an elegant user experience for operators to safely and predictably make changes to infrastructure.

Create reproducible infrastructure

- Reproducible production, staging, and development environments
- Shared modules for common infrastructure patterns
- Combine multiple providers consistently

Terraform makes it easy to re-use configurations for similar infrastructure, helping you avoid mistakes and save time.

[View All Providers](#)

Microsoft
Azure



aws



Google Cloud Platform

vmware®

Alibaba Cloud

ORACLE®



DATADOG



200+ more

Modules

JUMP TO SECTION ▾

Note: This page is about Terraform 0.12 and later. For Terraform 0.11 and earlier, see [0.11 Configuration](#)

Language: [Modules](#).

A *module* is a container for multiple resources that are used together.

Every Terraform configuration has at least one module, known as its *root module*, which consists of the resources defined in the `.tf` files in the main working directory.

A module can call other modules, which lets you include the child module's resources into the configuration in a concise way. Modules can also be called multiple times, either within the same configuration or in separate configurations, allowing resource configurations to be packaged and re-used.

This page describes how to call one module from another. Other pages in this section of the documentation describe the different elements that make up modules, and there is further information about how modules can be used, created, and published in [the dedicated *Modules* section](#).

Write declarative configuration files

- Collaborate and share configurations
- Evolve and version your infrastructure
- Automate provisioning

Define infrastructure as code to manage the lifecycle — create new resources, manage ones, and destroy those no longer needed.

Modules

JUMP TO SECTION ▾

Note: This page is about Terraform 0.12 and later. For Terra

Language: [Modules](#).

A *module* is a container for multiple resources that are used to

Every Terraform configuration has at least one module, known as the root module. It contains the top-level resources defined in the `.tf` files in the main working directory.

A module can call other modules, which lets you include the logic of multiple configurations in a concise way. Modules can also be called multiple times, either from different configurations or from other configurations, allowing resource configurations to be packaged and re-used.

This page describes how to call one module from another. Other pages describe the different elements that make up modules, and the process for creating and publishing them. Modules can also be used, created, and published in the dedicated [Modules section](#).

Create reproducible infrastructure

- Reproducible production, staging, and development environments
- Shared modules for common infrastructure patterns
- Combine multiple providers consistently

Terraform makes it easy to re-use configuration across multiple projects and environments to build similar infrastructure, helping you avoid mistakes and save time.

[View All Providers](#)

These features are very similar with 

Declarative

 makes it painless to create interactive UIs. Design simple views for each state in your application, and  will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in  without rewriting existing code.

 can also render on the server using Node and power mobile apps using 

These features are very similar with **React**

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

React can also render on the server using Node and power mobile apps using [React Native](#).

Terraform is an infrastructure provisioning tool
but the concepts aren't only for SysOps Engineer.

*.examples of declarative

Using a Jenkinsfile

This section builds on the information covered in [Getting started with Pipeline](#) and introduces more useful steps, common patterns, and demonstrates some non-trivial [Jenkinsfile](#) examples.

Creating a [Jenkinsfile](#), which is checked into source control ^[1], provides a number of immediate benefits:

- Code review/iteration on the Pipeline
- Audit trail for the Pipeline
- Single source of truth ^[2] for the Pipeline, which can be viewed and edited by multiple members of the project.

Pipeline supports [two syntaxes](#), [Declarative](#) (introduced in Pipeline 2.5) and [Scripted Pipeline](#). Both of which support building continuous delivery pipelines. Both may be used to define a Pipeline in either the web UI or with a [Jenkinsfile](#), though it's generally considered a best practice to create a [Jenkinsfile](#) and check the file into the source control repository.

Using Helm and Kustomize to Build More [Declarative](#) Kubernetes Workloads

[Argo CD - Declarative GitOps CD for Kubernetes](#)

101.3 Pros and Cons

Pros	Cons
Problem Solving w/Engineer's way	High Entry Barrier (Especially, non-coder)
Reusable Resources	Everyone Should do Terraform
Productivity	Difficult to design
Reduce Human Error	Difficult to train

101.4 Providers

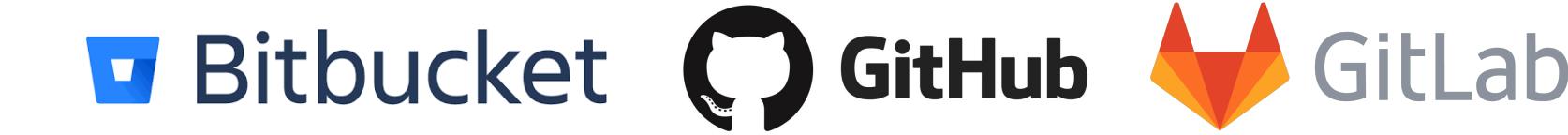
major cloud



major cloud



VCS



Infrastructure Software



Monitoring & System Management



SignalFx

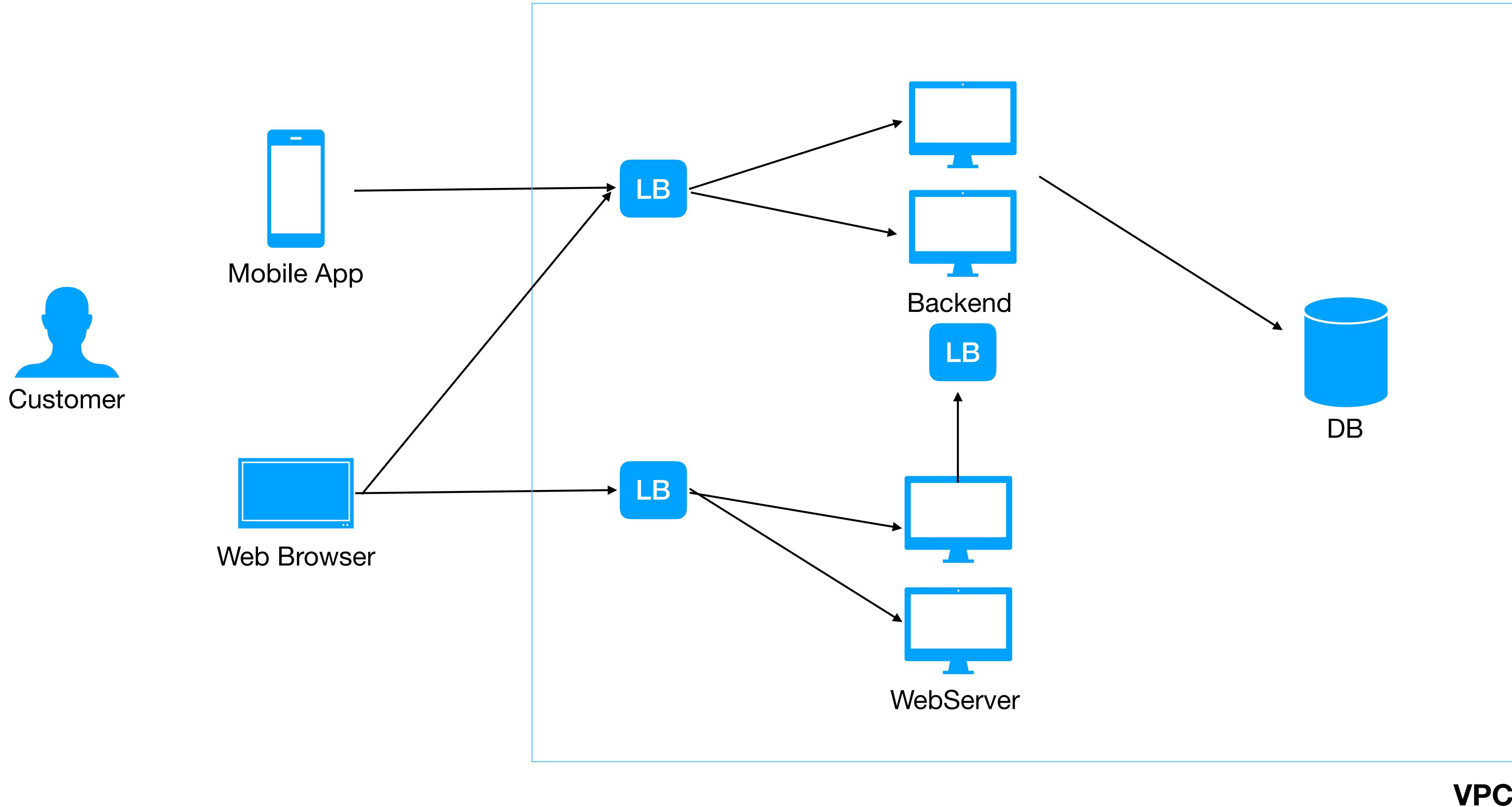
Community

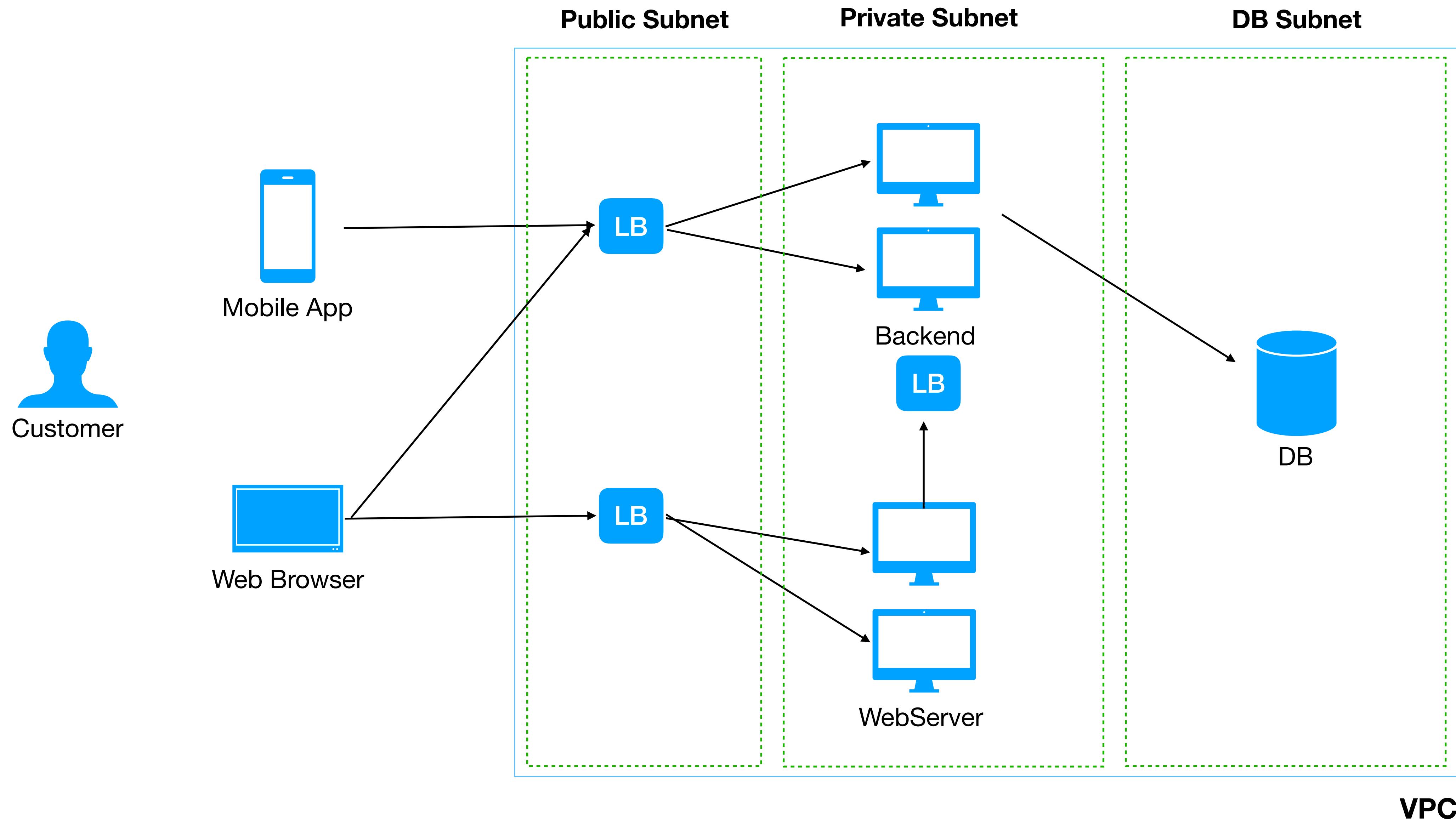
- 1Password
- Abiquo
- Active Directory - adlerrobert
- Active Directory - GSLabDev
- Aiven
- AlienVault
- AnsibleVault
- Apigee
- Artifactory
- Auth
- Citrix ADC
- Cloud Foundry
- CloudAMQP
- CloudKarafka
- CloudMQTT
- CloudPassage Halo
- CodeClimate
- Confidant
- Confluent Cloud
- Foreman
- Gandi
- Generic Rest API
- Git
- GitHub Code Owners
- GitHub File
- GitInfo
- Glue
- GoCD
- Google Calendar
- Jira (Extended)
- JumpCloud
- Kafka
- Kafka Connect
- Keboola
- Keycloak
- Keyring
- Kibana
- Kong
- OpenvCloud
- oVirt
- Pass
- PHPiPAM
- Pingdom
- Pivotal Tracker
- Proxmox
- Puppet CA
- PuppetDB
- Purestorage Flasharray
- Sewan
- Shell
- Smartronix
- Snowflake
- snowflakedb
- sops
- Spinnaker
- SQL
- Stateful
- Auth0
- Automic Continuous Delivery
- AVI
- Aviatrix
- AWX
- Azure Devops
- Bitbucket Server
- Centreon
- Checkly
- Cherry Servers
- Databricks
- Dead Man's Snitch
- Digital Rebar
- Docker Machine
- Drone
- Dropbox
- Duo Security
- EfficientIP
- Elastic Cloud Enterprise (ECE)
- Elasticsearch
- Google G Suite
- GorillaStack
- Hiera
- HPE OneView
- HTTP File Upload
- IBM Cloud
- IIJ GIO
- Infoblox
- InsightOPS
- Jira
- Manifold
- Matchbox
- MongoDB Atlas
- Nagios XI
- Name
- Nelson
- NetApp
- NSX-V
- Okta
- QingCloud
- Qiniu
- Redshift
- RKE
- Rollbar
- SakuraCloud
- SCVMM
- Sendgrid
- Sensu
- Sentry
- Uptimerobot
- Vaulted
- Venafi
- vRealize Automation
- Vultr
- Wavefront
- Win DNS
- XML

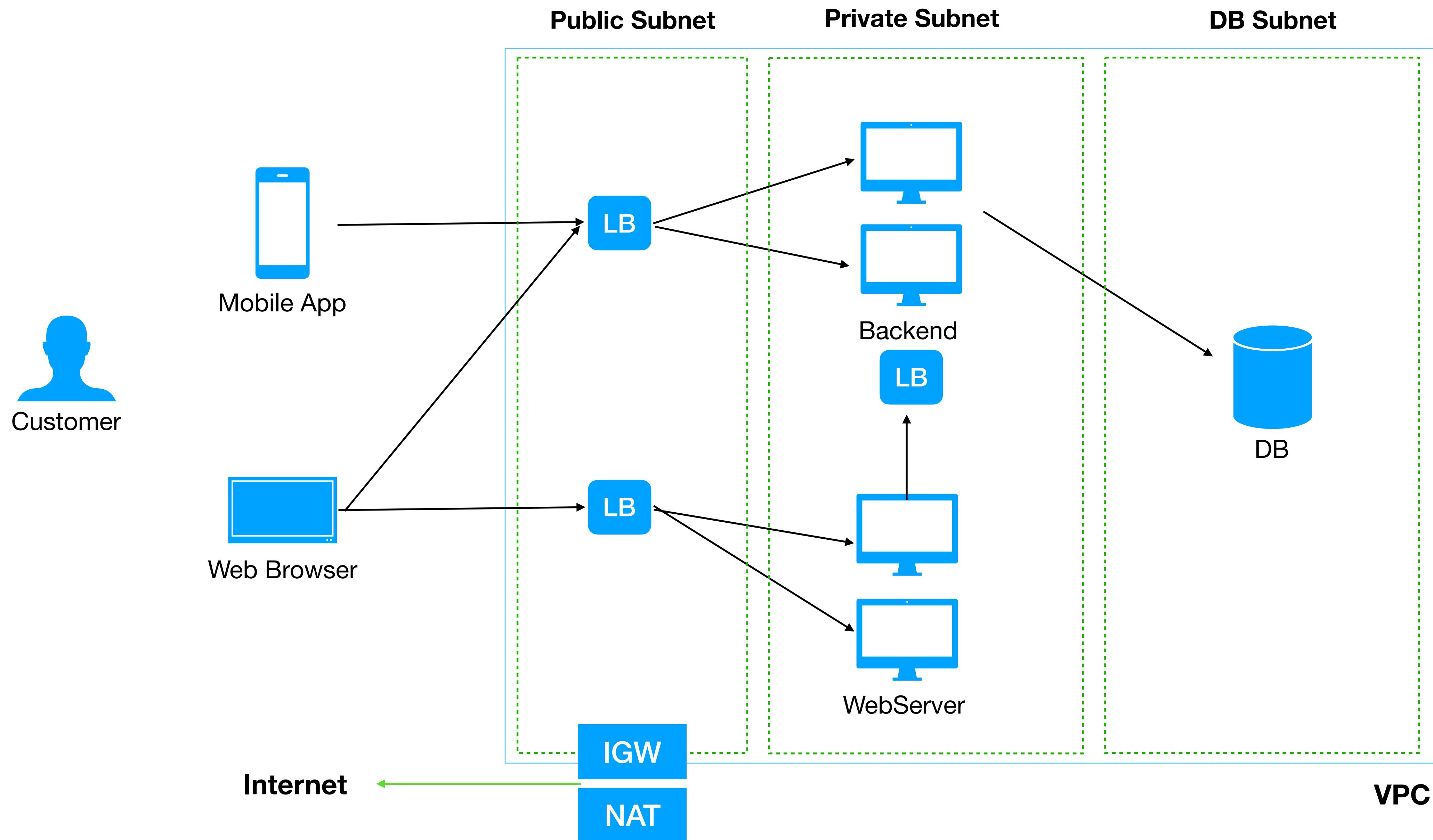
Code For Everything

Not only for the infrastructure

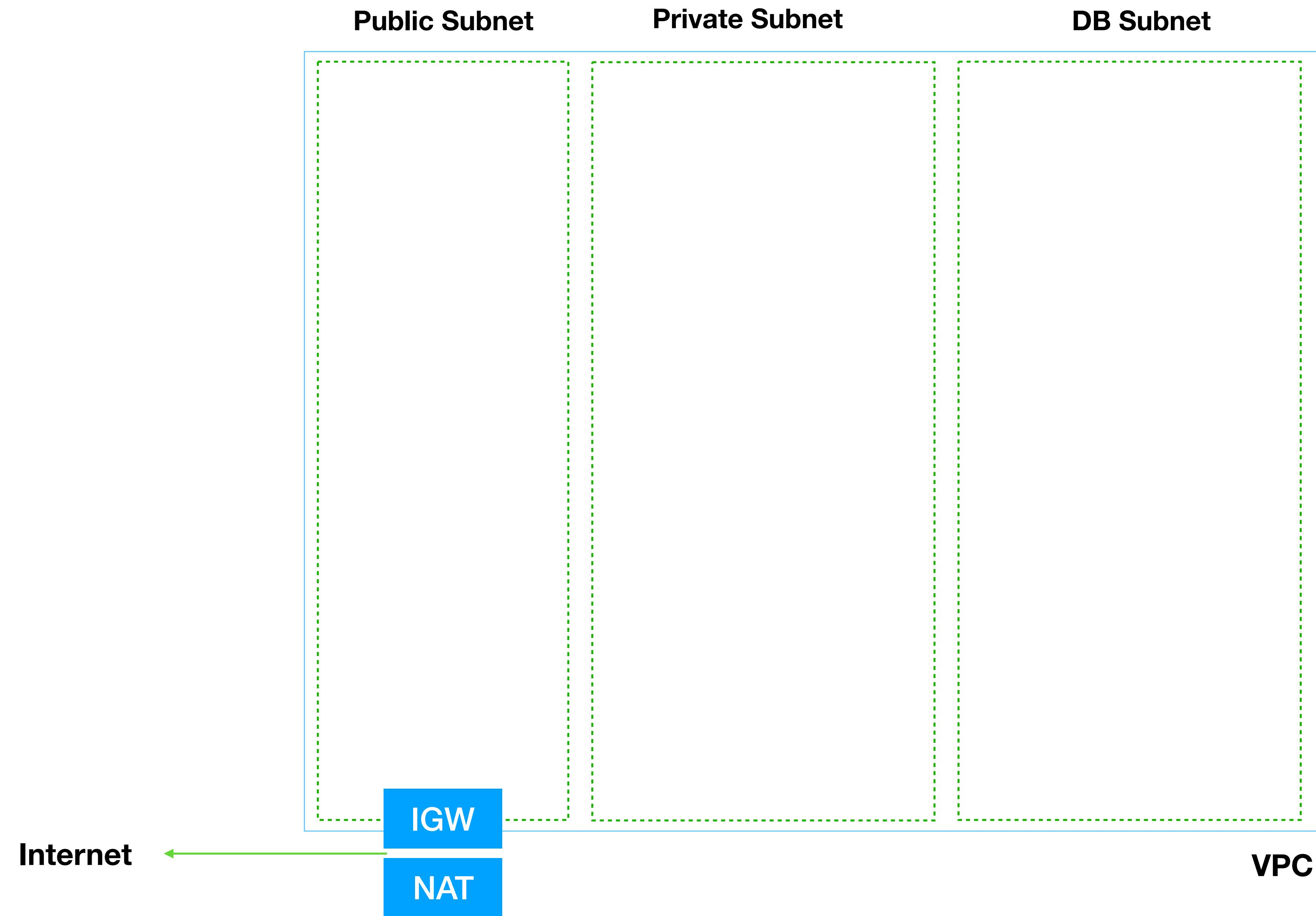
101.5 Basic Infrastructure







101.6 Getting Started

DEMO

Terraform Install

<https://www.terraform.io/downloads.html>

Download Terraform

JUMP TO SECTION ▾

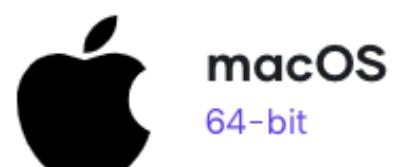
Below are the available downloads for the latest version of Terraform (0.12.16). Please download the proper package for your operating system and architecture.

Terraform is distributed as a single binary. Install Terraform by unzipping it and moving it to a directory included in your system's [PATH](#).

You can find the [SHA256 checksums](#) for Terraform 0.12.16 online and you can [verify the checksums](#) signature file which has been signed using HashiCorp's GPG key. You can also [download older versions](#) of Terraform from the releases service.

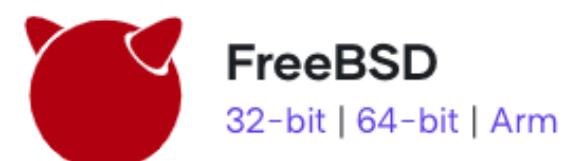
Check out the [v0.12.16 CHangelog](#) for information on the latest release.

Note: When you upgrade to Terraform 0.12, your existing Terraform configurations might need syntax updates. You can make most of these updates automatically with the `terraform 0.12upgrade` command; for more information, see [Upgrading to Terraform 0.12](#).



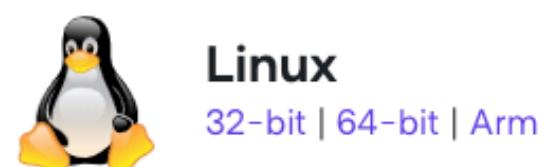
macOS

64-bit



FreeBSD

32-bit | 64-bit | Arm



Linux

32-bit | 64-bit | Arm

After download the package,

```
$ mv terraform terraform_v0.12.0
```

```
$ mv terraform_v0.12.0 /usr/local/bin/terraform_v0.12.0
```

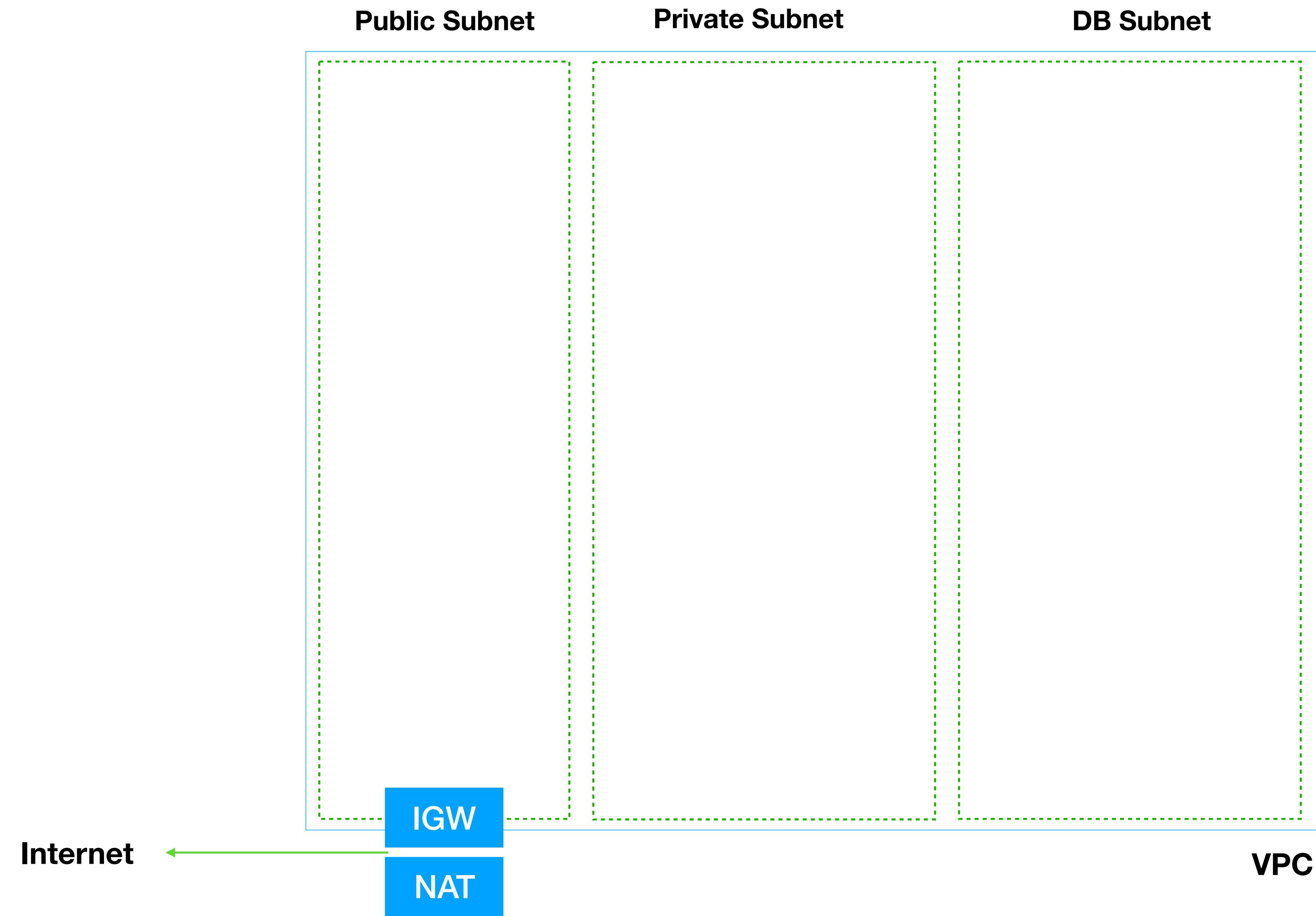
```
$ ln -s /usr/local/bin/terraform_v0.12.0 /usr/local/bin/tf
```

HCL(Hashicorp Configuration Language)

1. Providers
2. Resources
3. Input Variables. (TBD)
4. Output Values. (TBD)
5. Local Values
6. Modules. (TBD)
7. Data Sources (TBD)
8. Functions(Interpolation). (TBD)
9. State (TBD)
10. Backend (TBD)

Commands

1. Init
2. plan (w/target)
3. apply
4. destroy

DEMO

DEMO

1. AWS Programmable Key Creation

AWS Admin Console -> User -> Security Credentials -> Access Keys -> Create access key

2. AWS Profile Setup

\$ vi ~/.aws/credentials

```
[dev]
aws_access_key_id={{ access_key }}
aws_secret_access_key={{ secret_access_key }}
```

DEMO

3. TF Demo Folder Creation
4. aws.tf file creation
5. Search “terraform was provider” in Google

[Provider: AWS - Terraform by HashiCorp](#)

[https://www.terraform.io › docs › providers › aws ▾](https://www.terraform.io/docs/providers/aws)

Example Usage

```
# Configure the AWS Provider
provider "aws" {
    version = "~> 2.0"
    region  = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
    cidr_block = "10.0.0.0/16"
}
```

DEMO

6. Copy & Paste

7. Add profile = "dev"

```
aws.tf+  
1 provider "aws" {  
2   version = "~> 2.0"  
3   region  = "us-east-1"  
4   profile  = "dev"  
5 }  
-
```

Example Usage

```
# Configure the AWS Provider  
provider "aws" {  
  version = "~> 2.0"  
  region  = "us-east-1"  
}  
  
# Create a VPC  
resource "aws_vpc" "example" {  
  cidr_block = "10.0.0.0/16"  
}
```

DEMO

8. “tf init”

```
→ tf-demo tf init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.40.0...

Terraform has been successfully initialized!

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

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

DEMO

9. Search “terraform aws vpc” in google

VPC - AWS: aws_vpc - Terraform by HashiCorp

[https://www.terraform.io › docs › providers › aws › vpc](https://www.terraform.io/docs/providers/aws/vpc)

cidr_block - (Required) The CIDR block for the VPC. instance_tenancy - (Optional) option for instances launched into the VPC; enable_dns_support ...

» · » Example Usage; » · » Argument Reference; » · » Attributes Reference; »

Example Usage

Basic usage:

```
resource "aws_vpc" "main" {  
    cidr_block = "10.0.0.0/16"  
}
```

Basic usage with tags:

```
resource "aws_vpc" "main" {  
    cidr_block      = "10.0.0.0/16"  
    instance_tenancy = "dedicated"  
  
    tags = {  
        Name = "main"  
    }  
}
```

DEMO

9. Search “terraform aws vpc” in google

VPC - AWS: aws_vpc - Terraform by HashiCorp

<https://www.terraform.io/docs/providers/aws/vpc>

cidr_block - (Required) The CIDR block for the **VPC**. **instance_tenancy** - (Optional) option for instances launched into the **VPC**; **enable_dns_support** ...

» · » Example Usage; » · » Argument Reference; » · » Attributes Reference; »

Argument Reference

The following arguments are supported:

- `cidr_block` - (Required) The CIDR block for the VPC.
 - `instance_tenancy` - (Optional) A tenancy option for instances launched into the VPC. Valid values are `host` or `local`.
Default is `local`.
 - `enable_dns_support` - (Optional) A boolean flag to enable/disable DNS support for the VPC. Only valid in regions and accounts that support EC2 Classic. See the [ClassicLink documentation](#) for more information.
Default is `false`.
 - `enable_classiclink` - (Optional) A boolean flag to enable/disable EC2 ClassicLink support for the VPC. Only valid in regions and accounts that support EC2 ClassicLink.
Default is `false`.
 - `enable_classiclink_dns_support` - (Optional) A boolean flag to enable DNS support for EC2 ClassicLink for the VPC. Only valid in regions and accounts that support EC2 ClassicLink.
Default is `false`.
 - `assign_generated_ipv6_cidr_block` - (Optional) Requests an Amazon-assigned IPv6 CIDR block for the VPC. You cannot specify the range of IP addresses for the block.
Default is `false`.
 - `tags` - (Optional) A mapping of tags to assign to the resource.

DEMO

9. Search “terraform aws vpc” in google

[VPC - AWS: aws_vpc - Terraform by HashiCorp](#)

<https://www.terraform.io/docs/providers/aws/vpc>

cidr_block - (Required) The CIDR block for the VPC. instance_tenancy - (Optional) option for instances launched into the VPC; enable_dns_support ...

» » Example Usage; » » Argument Reference; » » Attributes Reference; »

Attributes Reference

In addition to all arguments above, the following attribute

- `arn` - Amazon Resource Name (ARN) of VPC
- `id` - The ID of the VPC
- `cidr_block` - The CIDR block of the VPC
- `instance_tenancy` - Tenancy of instances spin up w
- `enable_dns_support` - Whether or not the VPC has
- `enable_dns_hostnames` - Whether or not the VPC ha
- `enable_classiclink` - Whether or not the VPC has
- `main_route_table_id` - The ID of the main route tab change a VPC's main route table by using an `aws_m`
- `default_network_acl_id` - The ID of the network AC
- `default_security_group_id` - The ID of the security
- `default_route_table_id` - The ID of the route table
- `ipv6_association_id` - The association ID for the IF

DEMO

10. create “vpc.tf” file

```
vpc.tf+
1 resource "aws_vpc" "main" {
2   cidr_block      = "10.0.0.0/16"
3   instance_tenancy = "dedicated"
4
5   tags = [
6     Name = "main"
7   ]
8 }
```

DEMO

11. Execute “tf plan”

```
→ tf-demo tf plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not
persisted to local or remote state storage.

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

Terraform will perform the following actions:

# aws_vpc.main will be created
+ resource "aws_vpc" "main" {
    + arn                                = (known after apply)
    + assign_generated_ipv6_cidr_block = false
    + cidr_block                          = "10.0.0.0/16"
    + default_network_acl_id            = (known after apply)
    + default_route_table_id           = (known after apply)
    + default_security_group_id        = (known after apply)
    + dhcp_options_id                  = (known after apply)
    + enable_classiclink              = (known after apply)
    + enable_classiclink_dns_support = (known after apply)
    + enable_dns_hostnames            = (known after apply)
    + enable_dns_support              = true
    + id                               = (known after apply)
    + instance_tenancy                 = "dedicated"
```

DEMO

11. Execute “tf 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
```

12. Checking “VPC Creation” in AWS Console

The screenshot shows the AWS VPC service page. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, and user information 'jay-.kim@samsung.com @ srep' and 'Tokyo'. Below the navigation, there's a sidebar with 'Dashboard', 'VPCs' (selected), 'Private Cloud', and 'AWS Lambda'. The main content area has a 'Create VPC' button and an 'Actions' dropdown. A search bar says 'Filter by tags and attributes or search by keyword'. A table lists one VPC entry:

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP options set	Main Route table
main	vpc-0a4607449cc79910f	available	10.0.0.0/16	-	dopt-8b7b28ec	rtb-090019fb8

DEMO

13. Change “Name” Tag Value & “tf plan” & “tf apply”
14. Checking the “VPC Name Change” on AWS Console
15. Add “local” variable

```
vpc.tf+  
1 locals {  
2   name = "Terraform Testing"  
3   cidr_block      = "10.0.0.0/16"  
4   app_name        = "suite"  
5 }  
6  
7 resource "aws_vpc" "main" {  
8   cidr_block      = local.cidr_block  
9   instance_tenancy = "dedicated"  
10  
11   tags = [  
12     Name = "${local.app_name} ${local.name}"  
13   ]  
14 }
```

DEMO

16. Search “terraform aws subnet”

About 86,800 results (0.36 seconds)

[AWS subnet - AWS: aws_subnet - Terraform by HashiCorp](#)

[https://www.terraform.io › docs › providers › aws › subnet](https://www.terraform.io/docs/providers/aws/subnet.html) ▾

Provides an VPC **subnet** resource. ... AWS and HashiCorp are working together to reduce the amount of time required for resource deletion and updates can be ...

» » Subnets In Secondary ... » Argument Reference; »

You've visited this page many times. Last visit: 11/26/19

17. Add “aws_subnet” “private”, “public”, “db” with cidrs

```
vpc.tf+
locals {
  name = "Terraform Testing"
  cidr_block      = "10.0.0.0/16"
  app_name        = "suite"
  tag_name        = "${local.app_name} ${local.name}"
}
```

DEMO

17. Add “aws_subnet” “private”, “public”, “db” with cidrs

```
vpc.tf+
locals {
  name = "Terraform Testing"
  cidr_block      = "10.0.0.0/16"
  app_name        = "suite"
  tag_name        = "${local.app_name} ${local.name}"
}
```

*** Interpolation Warning**

Terraform Version < 0.12.x

`${aws_vpc.main.id}`

Terraform Version > 0.12.x

`aws_vpc.main.id`

18.plan / apply

```
18 resource "aws_subnet" "private" {
19   vpc_id      = aws_vpc.main.id
20   cidr_block  = "10.0.1.0/24"
21
22   tags = {
23     Name = local.tag_name
24   }
25 }
26
27 resource "aws_subnet" "public" {
28   vpc_id      = aws_vpc.main.id
29   cidr_block  = "10.0.2.0/24"
30
31   tags = {
32     Name = local.tag_name
33   }
34 }
35
36 resource "aws_subnet" "db" {
37   vpc_id      = aws_vpc.main.id
38   cidr_block  = "10.0.3.0/24"
39
40   tags = {
41     Name = local.tag_name
42   }
43 }
```

DEMO

19. igw, route table, nat, nat_eip

```
45 resource "aws_internet_gateway" "igw" {
46   vpc_id = aws_vpc.main.id
47
48   tags = {
49     Name = local.tag_name
50   }
51 }
52
53
54 resource "aws_eip" "nat_eips" {
55   vpc    = "true"
56 }
57
58
59 resource "aws_nat_gateway" "nats" {
60   subnet_id      = aws_subnet.public.id
61   allocation_id = aws_eip.nat_eips.id
62 }
```

DEMO

19. igw, route table, nat, nat_eip

```

63
64 resource "aws_route_table" "private" {
65   vpc_id = aws_vpc.main.id
66
67   tags = {
68     Name = "${local.app_name} private"
69   }
70 }
71
72 resource "aws_route_table" "public" {
73   vpc_id = aws_vpc.main.id
74
75   tags = {
76     Name = "${local.app_name} public"
77   }
78 }
79
80 resource "aws_route_table" "db" {
81   vpc_id = aws_vpc.main.id
82
83   tags = {
84     Name = "${local.app_name} db"
85   }
86 }
87

```

```

88 resource "aws_route" "public" {
89   destination_cidr_block = "0.0.0.0/0"
90   gateway_id              = aws_internet_gateway.igw.id
91   route_table_id           = aws_route_table.public.id
92 }
93
94 resource "aws_route" "private" {
95   destination_cidr_block = "0.0.0.0/0"
96   nat_gateway_id          = aws_nat_gateway.nats.id
97   route_table_id           = aws_route_table.private.id
98 }
99
100 resource "aws_route" "db" {
101  destination_cidr_block = "0.0.0.0/0"
102  nat_gateway_id          = aws_nat_gateway.nats.id
103  route_table_id           = aws_route_table.db.id
104 }
105
106 resource "aws_route_table_association" "public" {
107   route_table_id = aws_route_table.public.id
108   subnet_id      = aws_subnet.public.id
109 }
110
111 resource "aws_route_table_association" "private" {
112   route_table_id = aws_route_table.private.id
113   subnet_id      = aws_subnet.private.id
114 }
115
116 resource "aws_route_table_association" "db" {
117   route_table_id = aws_route_table.db.id
118   subnet_id      = aws_subnet.db.id
119 }

```