The boring registry

Opinionated terraform modules, secure information exchange and a registry I November 2020

TIER

Daniel & Alexander

Daniel Ciaglia - Infrastructure Lead

#changemobilityforgood utilising the best infrastructure landscape and tooling
#aws #kubernetes #hashicorp

Alexander Hellbom - Senior Devops Engineer

TIER

TIER as a company

We are here to **lead the way** towards **seamless and sustainable mobility**.

Together with public and private organizations, **we are rethinking urban transportation** and helping **create** a **clean, sustainable** and **better connected tomorrow** with cities **free from pollution and congestion**.

TIER in numbers

- **3** major compute environments
- **~80** nodes in production EKS
- PostgreSQL **Auroras**, **RabbitMQ**, some **λ**, **Redshift**...
- **80+** Developers

- **65k** vehicle (scooters + mopeds)
- **30+ Million** rides
- **80** Cities, **10** countries
- **900+** employees

TIER

Infrastructure as Code

No exceptions!

Policy: no manual creation or change of AWS or Kubernetes resources.

Never! Nowhere!

Solution:

- Global use of **terraform**
- **One** infrastructure repository **per team**
- **Terraform as CI** with final approval of administrative team*
- **Sandbox** account with **Administrator permissions** for **everybody****

*Bottleneck, subject to change
**Cleanup of sandbox is a different topic

Show me your source

Team infrastructure project

<team-infra-repository>

```
├── production/
│   ├── aws.tf
│   ├── okta.tf
│   ├── kubernetes.tf
│   ├── vault.tf
│   ├── example-service-foo/
│   │   ├── eu-central-1
│   │   │   └── main.tf
│   │   ├── eu-west-3
│   │   └── main.tf
│   ├── example-service-bar/
│   └── main.tf
│
├── staging/
│   ├── example-service-foo
│   └── example-service-bar
│
└── sandbox/
```

<production/main.tf>

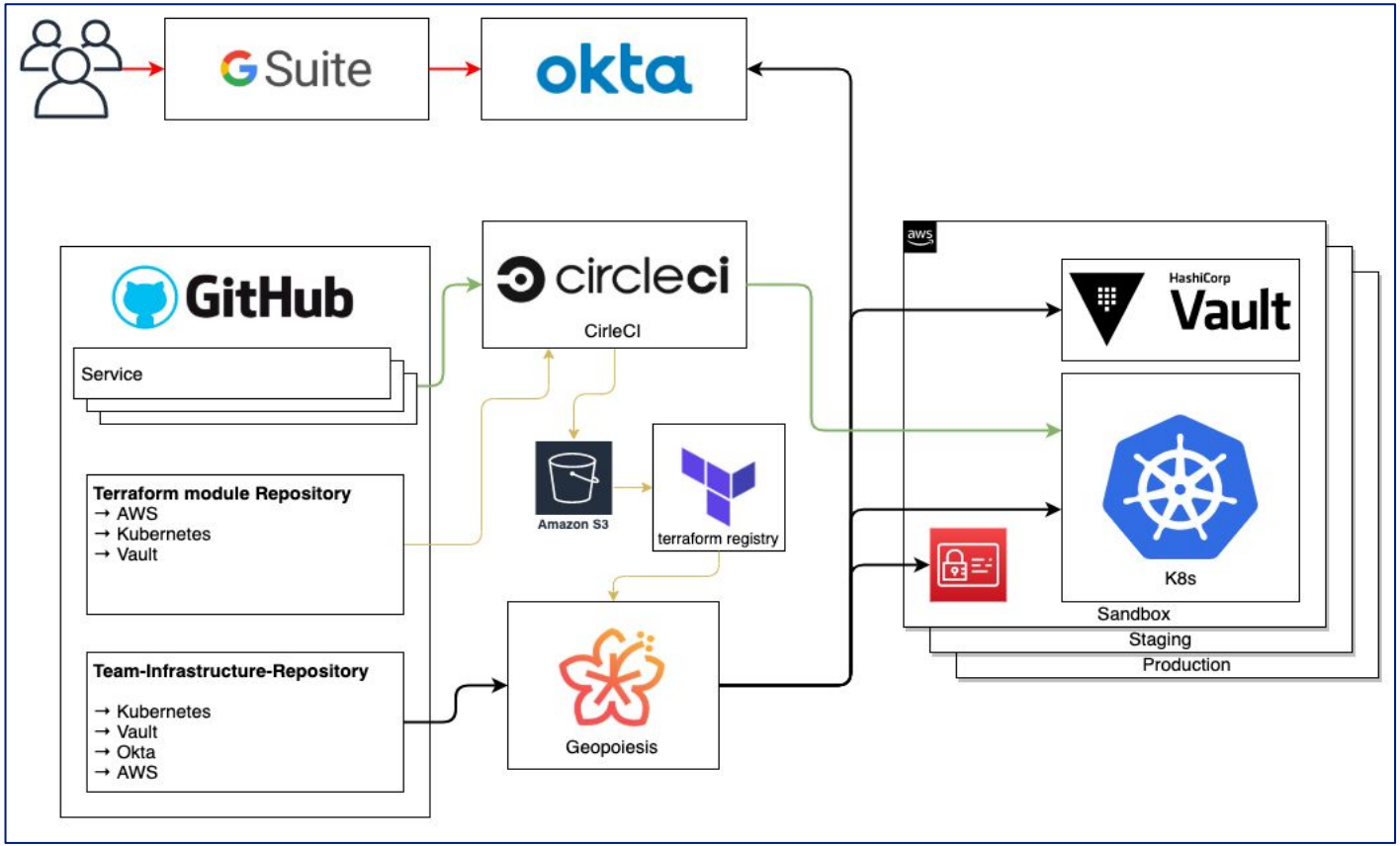
```
module "example-service-foo" {
  source      = "./example-service-foo"
  environment = "production"
  team        = "ring-0"
  service     = "example-service-foo"
  region      = "global"
}
```

<production/example-service-foo/main.tf>

```
module "example-service-foo-eu-central-1" {
  source      = "./eu-central-1"
  service     = var.service
  region      = "eu-central-1"
  team        = var.team
  environment = var.environment
}
```

The moving parts

TIER tooling



TF-CIX

Terra**f**orm **C**onfiguration **I**ntere**X**change

Problem

- Shared information between stacks
- ⚠️ Sensitive Data in State*

Solution

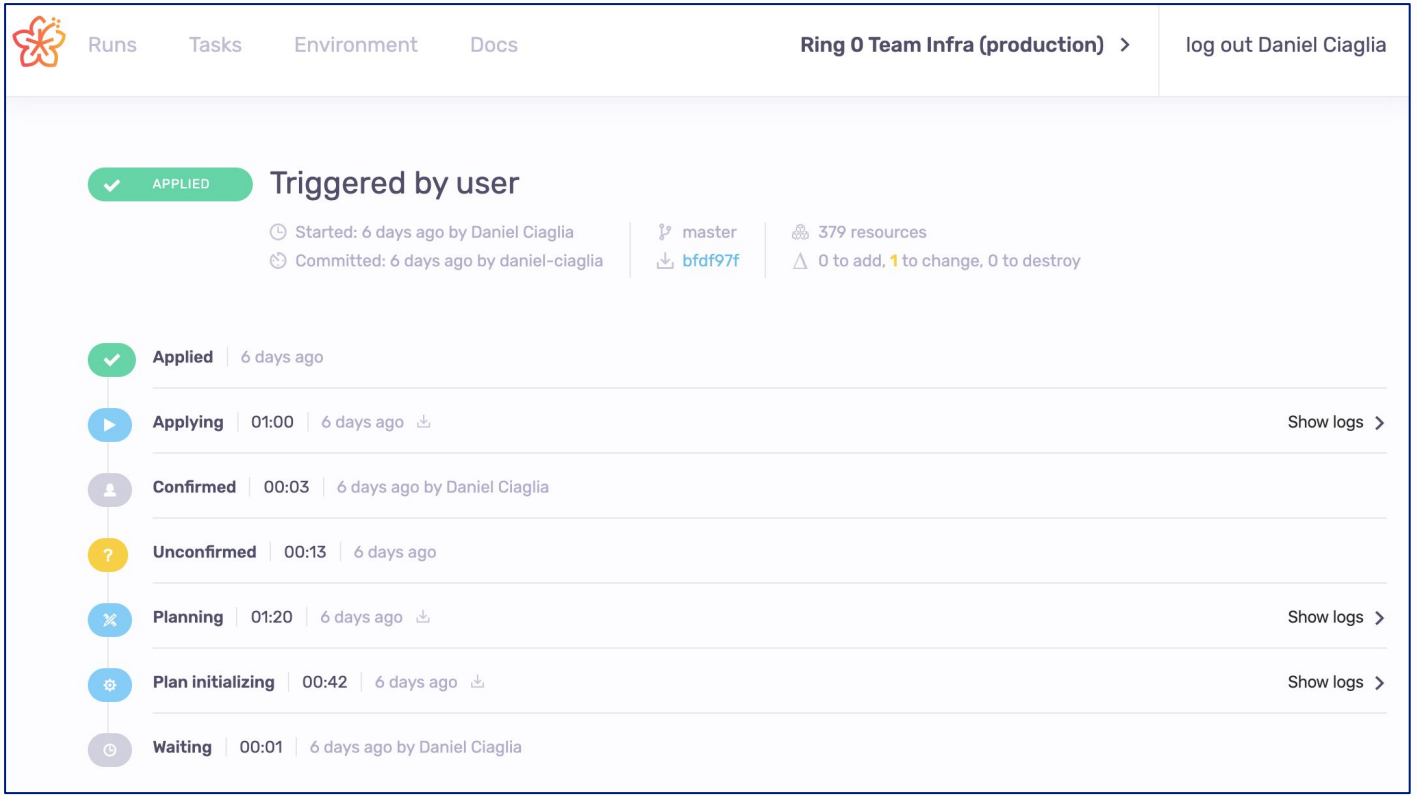
- Source stack writes SSM parameters
  - In a defined structure
- TF-CIX modules read and expose the data

TIER

* https://www.terraform.io/docs/state/sensitive-data.html

Compile the Infrastructure

Terraform CI



TIER

Share the knowledge

TF-CIX - share it

eg. **ring-0/core-infra**
- Creates VPC
  - Subnets
- Creates Kubernetes Cluster
  - Endpoint
  - Node group Security Groups
  - OIDC provider

eg. **teamFoo/serviceBar**
- Needs a RDS instance
- Needs a Kubernetes namespace
- Needs a Vault configuration

```
$> cat ring-0/core-infra/tf-cix.tf

locals {
  ssm_prefix = "/tf-cix/${var.metadata["service"]}/${var.region}"
}

resource "aws_ssm_parameter" "main_db_networks" {
  description = "The ID of the _db_networks_"
  name        = "${local.ssm_prefix}/main-vpc/db_networks"
  tags        = local.aws_tags
  type        = "String"
  value       = jsonencode(var.network_config.db_networks)
  overwrite   = true
}

resource "aws_ssm_parameter" "aaa" { }
resource "aws_ssm_parameter" "bbb" { }
resource "aws_ssm_parameter" "ccc" { }
resource "aws_ssm_parameter" "ddd" { }


----
Parameter: /tf-cix/core-infra/eu-central-1/main-vpc/db_networks
Value:     ["10.0.56.0/21","10.0.120.0/21","10.0.184.0/21"]
```

Share the knowledge

TF-CIX - behind the scenes

```
$> cat TF-CIX/core-infra/main.tf

data "aws_ssm_parameter" "vpc_id" {
  name = "/tf-cix/${var.service}/${var.region}/${var.name}/id"
}

data "aws_ssm_parameter" "aaa" { }
data "aws_ssm_parameter" "bbb" { }


$> cat TF-CIX/core-infra/outputs.tf

output "vpc_id" {
  description = "The ID of the _main-vpc_"
  value       = data.aws_ssm_parameter.vpc_id.value
}

output "aaa" { }
output "bbb" { }
```

Share the knowledge

TF-CIX - consume it

```
$> cat teamFoo/serviceBar/data.tf

module "core-data" {
  source  = "github.com/TierMobility/terraform-registry.git//TF-CIX/core-infra?ref=core-infra_1.0.1"
  service = "core-infra"
  region  = "eu-central-1"
  name    = "main-vpc"
}

$> cat teamFoo/serviceBar/service.tf

module "db" {
  source  = "your.terraform.registry.local/tier/postgresql/aws"
  version = "~> 1.0"

  name                        = "db"
  subnets                     = module.core-data.vpc_aux_networks
  instance_type               = "db.t3.micro"
  engine_version              = "11.8"
  vpc_id                      = module.core-data.vpc_id
  allowed_security_groups     = [module.core-data.eks-worker-security-group-id]
  vault_enabled               = true
  [...]
}
```

Shared modules

- Central Git repo + tags
- Opinionated for
  - Security
  - Naming
  - Conformity

TIER

Show me your source

Opinionated modules

eg. **aws/S3**
- Versioned
- Server Side Encryption (at-rest)
- ACL = ”Private”
- Block Public Access

eg. **aws/RDS**
- Multi-AZ
- Force SSL (in-transit)
- Storage Encrypted (at-rest)
- Performance Insights
- Deletion Protection
- dynamic secrets via Vault // **you want that!**

eg. **vault/Service**
- Authentication via ServiceAccount
- Proper Access policies
- Token TTL

What we gain additionally

- Common naming schema
- Common tagging schema
  👉 great for separating the bill
- Distinction on regional and global objects
  👉 eg. S3 bucket names, IAM objects
- High Code quality
- Active developer participation

Show more source

Module repository

```
# Usage
module "main-s3" {
  source   = "github.com/TierMobility/terraform-registry.git//aws/S3"
  name     = "terraform-registry"
  metadata = local.metadata
}

locals {
  metadata = {
    environment = var.environment
    region      = var.region
    service     = var.service
    team        = var.team
  }
}
```

```
$> git tag | grep s3
s3_1.0.0
s3_1.1.0
s3_2.0.0
s3_2.1.0
s3_2.1.1
```

```
.
├── TF-CIX
│   ├── core-infra
│   ├── rabbitmq
│   ├── rds
│   └── ring-0-team-infra
├── aws
│   ├── DocumentDB
│   ├── DynamoDB
│   ├── ECR
│   ├── Elasticache
│   ├── Elasticsearch
│ [...]
│   ├── RDS
│   │   ├── Aurora
│   │   ├── Postgresql
│   │   └── Postgresql-replica
│   ├── S3
│   └── Transitgateway
├── kubernetes
│   ├── alb-ingress
│   ├── aws-auth
│   ├── aws-node
│   ├── cert-manager
│ [...]
│   └── vertical-pod-autoscaler
└── vault
    ├── Org
    ├── Service
    └── Team
```

The Problem

- Since we have a monorepo of our Terraform Modules, **each source reference includes a new copy of the entire git repo**
- **Tricky to release new features** or bug fixes
- If we stick to branches, we run into the **risk of accidentally introducing breaking changes** and it becomes hard to maintain
- We do not believe in long-lived branches :)

Facing the problem

Know your numbers

```
$> terraform init

$> grep “source = \"github.com:“ | wc -l

   56

$> find .terraform/modules -type f | wc -l

   20937

$> du -hc -d 0 .terraform/modules

   167M  total
```

```
$> git tag | sed -e 's/_.*//g' | uniq -c

   1 alb-ingress
   1 aws-auth
   4 bastion
   5 cert-manager
   4 cluster-autoscaler
   2 core-infra
   4 dashboard
   7 datadog-agent
   2 documentdb
   1 dynamodb
   4 ecr
   6 external-dns
   9 fluentbit
   2 kms
   1 metrics-server
   4 nginx-ingress
   1 node-termination-handler
  11 postgres
   3 prometheus-adapter
   2 prometheus-blackbox-exporter
  21 prometheus-operator
   5 rabbitmq-service-access
  11 rabbitmq
   1 redis
   5 s3
   2 thanos
  26 vault
   1 vpa
```

The Goals

- **Semantic versioning** for our Terraform Modules
- **No moving parts**
  - No extra dependencies
  - No UI
  - Not the entire Registry API
- **Extensible**
  - multiple storages for your cloud
    - AWS
    - GCP
    - Azure
    - Filesystem / In-memory
- **Secure by default**
  - Protect access from the outside
  - Source access is managed elsewhere

The Solution

- Terraform Registry API

  `GET /v1/modules/{namespace}/{name}/{provider}/versions`

  `GET /v1/modules/{namespace}/{name}/{provider}/{version}/download`

- Storage backend with a clear path structure
  - The storage backend knows where to look for modules and module versions
- Example structure

  `namespace=tier/name=s3/provider=aws/version=1.0.0`

```
$> tree .
.
└── namespace=tier
    ├── name=documentdb
    │   └── provider=aws
    │       └── version=1.0.0
    │           └── tier-documentdb-aws-1.0.0.tar.gz
    └── name=dynamodb
        └── provider=aws
            ├── version=1.0.0
            │   └── tier-dynamodb-aws-1.0.0.tar.gz
            └── version=1.0.1
                └── tier-dynamodb-aws-1.0.1.tar.gz
```

But how does it work?

Multi-purpose binary

```
$> boring-registry help

USAGE
  boring-registry [flags] <subcommand> [flags] [<arg>...]

SUBCOMMANDS
  server  run the registry api server
  upload  uploads terraform modules to a registry

FLAGS
  -debug false     log debug output
  -no-color false  disable color output
  -s3-bucket ...   s3 bucket to use for the S3 registry
  -s3-prefix ...   s3 prefix to use for the S3 registry
  -s3-region ...   s3 region to use for the S3 registry
  -type s3         registry type

VERSION
boring-registry v0.1.0
```

But how does it work?

Multi-purpose binary - server

```
$> boring-registry server help

USAGE
  server [flags]

Run the registry API server.

The server command expects some configuration, such as which registry type to use.
The default registry type is "s3" and is currently the only registry type available.
For more options see the available options below.

EXAMPLE USAGE

boring-registry server -type=s3 -s3-bucket=my-bucket

FLAGS
  -api-key ...                    comma-delimited list of api keys
  -debug false                    log debug output
  -listen-address :5601           listen address for the registry api
  -no-color false                 disable color output
  -s3-bucket ...                  s3 bucket to use for the S3 registry
  -s3-prefix ...                  s3 prefix to use for the S3 registry
  -s3-region ...                  s3 region to use for the S3 registry
  -telemetry-listen-address :7801 listen address for telemetry
  -type s3                        registry type

VERSION
boring-registry v0.1.0
```

But how does it work?

As easy as 1-2-3...4-5

```
$> aws s3api create-bucket --bucket example
$> aws cp --recursive . s3://example
```

```
$> export BORING_REGISTRY_API_KEY="very-secure-token"
$> export BORING_REGISTRY_S3_BUCKET="example"
$> boring-registry server
```

```
$> cat ~/.terraformrc

  credentials "boring-registry" {
    token = "very-secure-token"
  }
```

```
$> cat main.tf
  module "main-s3" {
    source = "boring-registry/tier/s3/aws"
    version = "~> 2"
  }
```

Steps:

1. Create a S3 bucket
2. Upload Module Archives
3. Deploy boring-registry
4. Add credentials to your Terraform config
5. terraform init/plan/apply

```
$> terraform init
[...]
Downloading boring-registry/tier/s3/aws 2.0.6 for foo-services.foo-services-eu-central-1.main-s3...
```

Prepare the data

- Write Terraform modules
- Add a *boring-registry.hcl* to each
- Put the repo in CI
- Process the modules

TIER

But how does it work?

Multi-purpose binary - upload

```
$> boring-registry upload help

USAGE
  upload [flags] <dir>

Upload modules to a registry.

The upload command expects some configuration, such as which registry type to use and which local directory to work in.
The default registry type is "s3" and is currently the only registry type available.
For more options see the available options below.

EXAMPLE USAGE

boring-registry upload -type=s3 -s3-bucket=my-bucket terraform/modules


FLAGS
  -debug false     log debug output
  -no-color false  disable color output
  -s3-bucket ...   s3 bucket to use for the S3 registry
  -s3-prefix ...   s3 prefix to use for the S3 registry
  -s3-region ...   s3 region to use for the S3 registry
  -type s3         registry type

VERSION
boring-registry v0.1.0
```

But how does it really work?

Where the magic happens

```
$> cat s3/boring-registry.hcl

metadata {
  namespace = "tier"
  name      = "s3"
  provider  = "aws"
  version   = "2.0.0"
}
```

The **upload mode** does

- Walk the repo
- Look for *`boring-registry.hcl`*
- Generate and upload the archives

```
$> export BORING_REGISTRY_S3_BUCKET=”example”
$> boring-registry upload .

[…]
Successfully uploaded module:
example.s3-eu-central-1.amazonaws.com/namespace=tier/name=s3/provider=aws/version=2.0.0/tier-s3-aws-2.0.0.tar.gz

Skipping already uploaded module:
example.s3-eu-central-1.amazonaws.com/namespace=tier/name=ecr/provider=aws/version=1.0.1/tier-ecr-aws-1.0.1.tar.gz
Skipping already uploaded module:
example.s3-eu-central-1.amazonaws.com/namespace=tier/name=redis/provider=aws/version=2.0.1/tier-redis-aws-2.0.1.tar.gz
```

Star me
Fork me

- TierMobility/boring-registry
- It has documentation
- It has a helm chart
- It’s Golang
- It’s HCL

TIER

Missing pieces

The code needs you!

- Auto-migration from git to registry
- Registry login protocol
- Different object storage backend
- Metrics
- Grafana Dashboard

- 😔 Consumer stacks have to be triggered to use newer module versions

TIER

Last bits

Tooling involved

- **Terraform** - https://www.terraform.io/
- **Geopoiesis** - https://spacelift.io/
- **Boring registry** - https://github.com/TierMobility/boring-registry
- **Module Registry Protocol** - https://www.terraform.io/docs/internals/module-registry-protocol.html
- **Login Protocol** - https://www.terraform.io/docs/internals/login-protocol.html

Summary

- Separate stacks
- Share selected information securely
- Use terraform modules for opinionated settings
- Use them centrally managed and versioned
- Use a boring registry to deliver

Like what you've seen?

We are looking for **DevOps** and **SRE** personalities to join us!

Learn more and apply:

https://tier-mobility-jobs.personio.de/

TIER

BE BOLD.

alexander.hellbom@tier.app // https://www.linkedin.com/in/alexanderhellbom/
daniel.ciaglia@tier.app // https://www.linkedin.com/in/danielciaglia