Terraform Best Practices (workshop)

Anton Babenko
@antonbabenko

June 2019

Anton Babenko

AWS Community Hero / Terraform fanatic since 2015

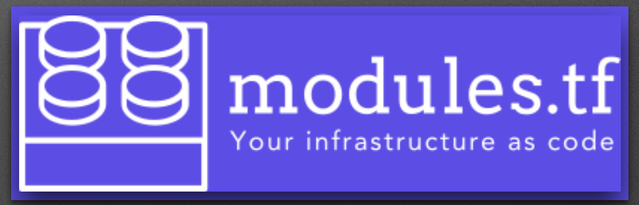
Organiser of HashiCorp UG, AWS UG, DevOps Norway, DevOpsDays Oslo

I 💚 open-source:

- terraform-community-modules + terraform-aws-modules
- antonbabenko/pre-commit-terraform — clean code and documentation
- antonbabenko/tfvars-annotations — update terraform.tfvars using annotations
- antonbabenko/modules.tf-lambda — generate Terraform code from visual diagrams
- antonbabenko/terragrunt-reference-architecture — Terragrunt reference architecture
- www.terraform-best-practices.com
- medium.com/@anton.babenko
- @antonbabenko — Twitter, GitHub, Linkedin

What do I do?

- All-things Terraform + AWS + DevOps
- Consulting
- Workshops
- Trainings
- Mentorship



My interview: https://medium.com/@anton.babenko/my-terraform-aws-journey-hashitimes-interview-73d1b542fcc0

My email: anton@antonbabenko.com

LinkedIn: https://www.linkedin.com/in/antonbabenko

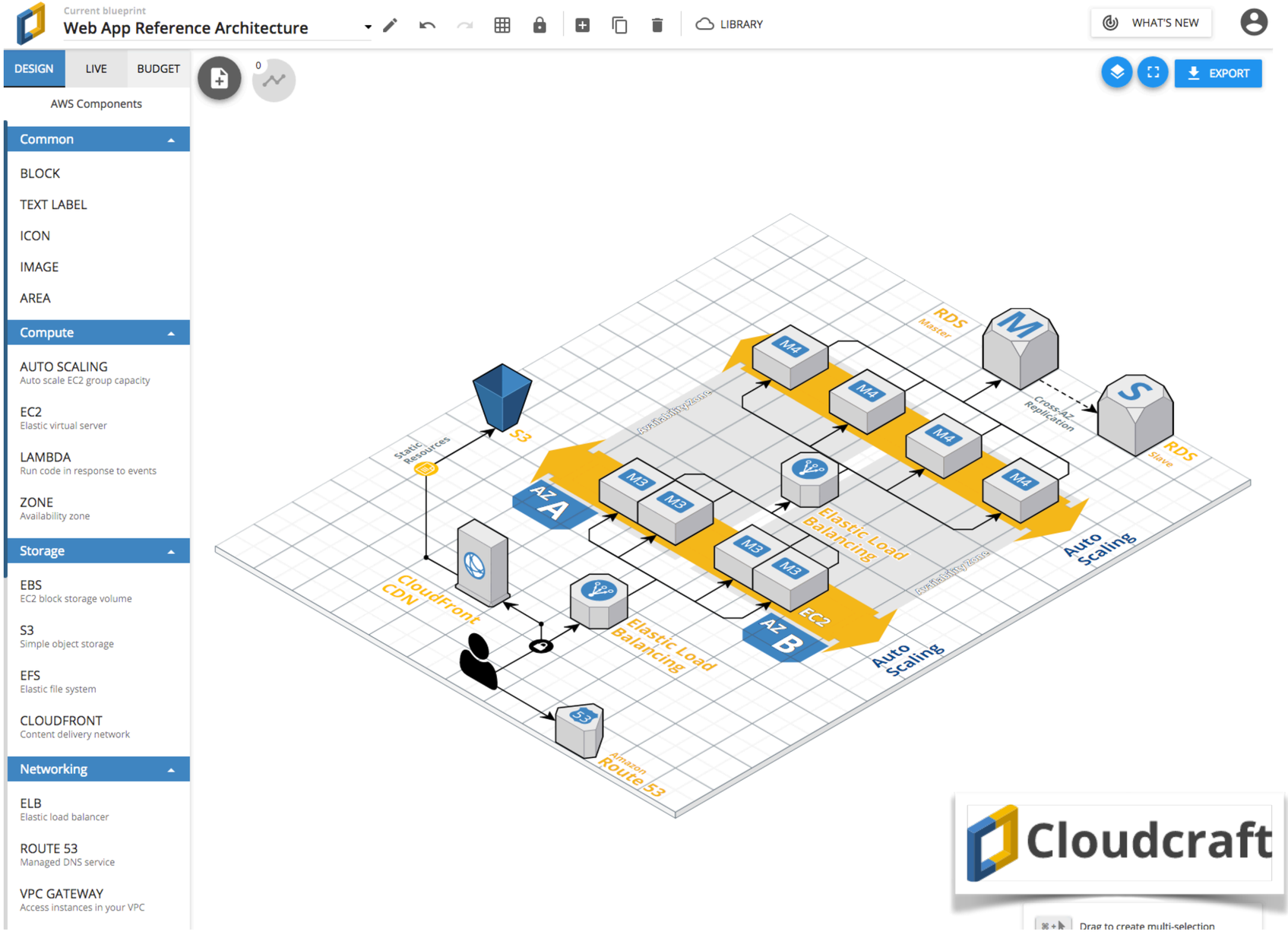
Collection of open-source Terraform AWS modules supported by the community.

More than 4 mil. downloads since September 2017.

(VPC, Autoscaling, RDS, Security Groups, ELB, ALB, Redshift, SNS, SQS, IAM, EKS, ECS...)

github.com/terraform-aws-modules

registry.terraform.io/modules/terraform-aws-modules



Cloudcraft.co — the best way to draw AWS diagrams

cloudcraft.co features

- Manage components in browser (EC2 instances, autoscaling groups, RDS, etc)
- Connect components
- Import live AWS infrastructure
- Calculate the budget
- Share link to a blueprint
- Export as image
- Embed drawing to wiki, Confluence, etc

Infrastructure as code makes DevOps possible

Key benefits:

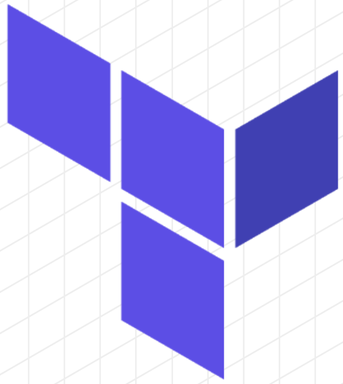
- Treat infrastructure like application code
- Always know what changed
- Validate infrastructure before deployment

https://dzone.com/articles/infrastructure-as-code-the-benefits

Plan

- What is Terraform?
- Modules
- Composition and orchestration
- Workspaces
- What’s new in Terraform 0.12?
- Some of my favourite Terraform best practices
- Practical tasks

@antonbabenko



HashiCorp

Terraform

Write, plan and manage infrastructure as code

www.terraform.io

```hcl
variable "aws_region" {
  description = "Region where resources should be created"
  default     = "eu-west-1"
}

provider "aws" {
  region = "${var.aws_region}"
}

resource "aws_s3_bucket" "this" {
  bucket = "my-bucket-${random_pet.bucket.id}"
}

resource "random_pet" "bucket" {
  keepers = {
    aws_region = "${var.aws_region}"
  }

  length = 1
}

output "this_s3_bucket_id" {
  description = "ID of S3 bucket"
  value       = "${aws_s3_bucket.this.id}"
}
```

```hcl
variable "aws_region" {
  description = "Region where resources should be created"
  default     = "eu-west-1"
}

provider "aws" {
  region = "${var.aws_region}"
}

resource "aws_s3_bucket" "this" {
  bucket = "my-bucket-${random_pet.bucket.id}"
}

resource "random_pet" "bucket" {
  keepers = {
    aws_region = "${var.aws_region}"
  }

  length = 1
}

output "this_s3_bucket_id" {
  description = "ID of S3 bucket"
  value       = "${aws_s3_bucket.this.id}"
}
```

```hcl
variable "aws_region" {
  description = "Region where resources should be created"
  default     = "eu-west-1"
}

provider "aws" {
  region = "${var.aws_region}"
}

resource "aws_s3_bucket" "this" {
  bucket = "my-bucket-${random_pet.bucket.id}"
}

resource "random_pet" "bucket" {
  keepers = {
    aws_region = "${var.aws_region}"
  }

  length = 1
}

output "this_s3_bucket_id" {
  description = "ID of S3 bucket"
  value       = "${aws_s3_bucket.this.id}"
}
```

```
$ terraform init

Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "aws" (1.10.0)...
- Downloading plugin for provider "random" (1.1.0)...

Terraform has been successfully initialized!
```

```
$ terraform apply

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_s3_bucket.this
      id:                   <computed>
      acl:                  "private"
      bucket:               "my-bucket-${random_pet.bucket.id}"

  + random_pet.bucket
      id:                   <computed>
      keepers.%:            "1"
      keepers.aws_region:   "eu-west-1"
      length:               "1"

Plan: 2 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.
```

```
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

random_pet.bucket: Creating...
  keepers.%:           "" => "1"
  keepers.aws_region: "" => "eu-west-1"
  length:              "" => "1"
random_pet.bucket: Creation complete after 0s (ID: seasnail)
aws_s3_bucket.this: Creating...
  acl:                  "" => "private"
  arn:                  "" => "<computed>"
  bucket:               "" => "my-bucket-seasnail"
aws_s3_bucket.this: Creation complete after 6s (ID: my-bucket-seasnail)

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

Outputs:

this_s3_bucket_id = my-bucket-seasnail
```

amazon web services

AWS CloudFormation

Google Cloud

Google Cloud Deployment Manager

Azure

Azure Resource Manager

HashiCorp
Terraform

+ more than 100 providers

HashiCorp
Terraform

Why Terraform and not AWS CloudFormation, Azure ARM, Google Cloud Deployment Manager?

- Terraform manages 100+ providers, has easier syntax (HCL), has native support for modules and remote states, has teamwork related features, is an open-source project.
- 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)

Terraform — universal tool for everything with an API

- GSuite
- Dropbox files and access
- New Relic metrics
- Datadog users and metrics
- Jira issues
- Minecraft, or even order Domino’s pizza
- All Terraform providers

Terraform modules

Modules in Terraform are self-contained packages of Terraform configurations that are managed as a group.

Resource modules

- Create resources in a very flexible configuration
- Open-source

Infrastructure modules

- Consist of resource modules
- Enforce tags and company standards
- Use preprocessors, jsonnet, cookiecutter

Terraform modules: do and don’t

@antonbabenko

Very Frequent Problem:
Terraform modules can’t be re-used, because they are very specific

Providers in modules — evil

```
1 provider "aws" {
2   region = "eu-west-1"
3
4   assume_role { ... }
5 }
```

Exception: logical providers (template, random, local, http, external)

Providers in modules — solution

```
# in root module
terraform {
  required_providers {
    aws = ">= 2.7.0"
  }
}
```

https://www.terraform.io/docs/configuration/terraform.html

```
1  # Vasya likes defaults
2  provider "aws" {
3    region = "eu-west-1"
4  }
5
6  # Petya likes shared credentials file
7  provider "aws" {
8    region = "eu-west-1"
9
10   shared_credentials_file = "~/.aws/my_secrets"
11 }
12
13 # They both lose!
```

Provisioner — evil

```
1 resource "aws_vpc" "this" {
2   cidr_block = "10.10.0.0/16"
3
4   provisioner "local-exec" {
5     command = "aws ec2 ..."
6   }
7 }
```

Avoid provisioner in all resources

Provisioner — evil

```
1 resource "aws_vpc" "this" {
2   cidr_block = "10.10.0.0/16"
3
4   provisioner "local-exec" {
5     command = "aws ec2 ..."
6   }
7 }
```

Avoid provisioner in all resources

Provisioner — evil

```
1 resource "aws_instance" "this" {
2   ami           = "ami-12345678"
3   instance_type = "t3.large"
4
5   provisioner "local-exec" {
6     command = "ansible-playbook ..."
7   }
8 }
```

Avoid provisioner even in EC2 resources

Provisioner — evil

```
1 resource "aws_instance" "this" {
2   ami           = "ami-12345678"
3   instance_type = "t3.large"
4
5   provisioner "local-exec" {
6     command = "ansible-playbook ..."
7   }
8 }
```

Avoid provisioner even in EC2 resources

@antonbabenko

```
# Solution 1
resource "aws_instance" "this" {
  ami           = "ami-12345678"
  instance_type = "t3.large"

  user_data     = "aws s3 cp ... & ansible-playbook ..."
}

# Solution 2 - Autoscaling group
resource "aws_launch_configuration" "this" {
  image_id      = "ami-12345678"
  instance_type = "t3.large"

  user_data     = "aws s3 cp ... & ansible-playbook ..."
}
```

```
1  # Solution 1
2  resource "aws_instance" "this" {
3    ami           = "ami-12345678"
4    instance_type = "t3.large"
5
6    user_data     = "aws s3 cp ... & ansible-playbook ..."
7  }
8
9  # Solution 2 - Autoscaling group
10 resource "aws_launch_configuration" "this" {
11   image_id      = "ami-12345678"
12   instance_type = "t3.large"
13
14   user_data     = "aws s3 cp ... & ansible-playbook ..."
15 }
```

null_resource provisioner — good

```
1 resource "null_resource" "this" {
2   provisioner "local-exec" {
3     command = "aws ec2 ..."
4     when    = "create"
5   }
6
7   depends_on = ["aws_vpc.this"]
8 }
```

Traits of good Terraform modules

- Documentation and examples
- Feature rich
- Sane defaults
- Clean code
- Tests

Read more: http://bit.ly/common-traits-in-terraform-modules

@antonbabenko

How to structure Terraform configurations? How to call them?

Call Terraform modules

- Use Terraform modules, because amount of resources and code is increasing
- How to organize Terraform configurations and invoke them?
- How to orchestrate modules?

All-in-one

Good:

- Declare variables and outputs in fewer places

Bad:

- Large blast radius
- Everything is blocked at once
- Impossible to specify dependencies between modules (depends_on)

```
# terraform.tfvars
region = "eu-west-1"

# main.tf
provider "aws" {
  region = "${var.region}"
}

module "vpc" {}
module "alb" {}
module "application" {}
module "security_group_alb" {}
module "security_group_app" {}
module "security_group_microservice" {}
module "microservice_1" {}
module "microservice_2" {}
module "microservice_X" {}
```

1-in-1

Good:

- Smaller blast radius
- Possible to join invocation
- Easier and faster to work with

Bad:

- Declare variables and outputs in more places

```
.
├── vpc
│   ├── main.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   └── variables.tf
├── alb
│   ├── main.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   └── variables.tf
├── application
│   ├── main.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   └── variables.tf
└── microservice_1
    ├── main.tf
    ├── outputs.tf
    ├── terraform.tfvars
    └── variables.tf
```

Which way do you group your code?

All-in-one or 1-in-1?

```
# terraform.tfvars
region = "eu-west-1"

# main.tf
provider "aws" {
  region = "${var.region}"
}

module "vpc" {}
module "alb" {}
module "application" {}
module "security_group_alb" {}
module "security_group_app" {}
module "security_group_microservice" {}
module "microservice_1" {}
module "microservice_2" {}
module "microservice_X" {}
```

All-in-one

or

```
.
├── vpc
│   ├── main.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   └── variables.tf
├── alb
│   ├── main.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   └── variables.tf
├── application
│   ├── main.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   └── variables.tf
└── microservice_1
    ├── main.tf
    ├── outputs.tf
    ├── terraform.tfvars
    └── variables.tf
```

1-in-1

~~Correct~~

MFA (Most Frequent Answer):

Somewhere in between

All-in-one

- Undefined project scope
- Fast prototyping and initial development phase
- Small number of resources & developers
- Tightly connected resources

1-in-1

- Defined project scope
- Different types of developers involved (!)
- Code reuse is encouraged (across organization and environments)
- Use Terragrunt

What about Terraform workspaces?

Problems with Terraform workspaces

- Terraform Workspaces aren’t infrastructure-as-code friendly. You can’t answer straight from the code:
  - "How many workspaces do you have?"
  - "What infrastructure has been deployed in *workspaceX*?"
  - "What is the difference between *workspaceX* and *workspaceY*?"
- Introducing complexity almost in all cases.

Solution — use re-usable modules instead of workspaces

Terraform 0.12

Will it help?

Terraform 0.12

- HCL2 — simplified syntax
- Loops ("for")
- Dynamic blocks ("for_each")
- Correct conditional operators (… ? … : …)
- Extended types of variables
- Templates in values
- Links between resources are supported (depends_on everywhere)
- Read more — https://www.hashicorp.com/blog/announcing-terraform-0-1-2-beta

Everything will be all right!

Just use 0.12, or not?

Different types of Terraform users

Types of Terraform users

- Terraform developers
- Terraform users (everyone else)

Terraform developers

- Write and support Terraform modules
- Implement company's standards (security, encryption, integrations)
- Maintain reference architectures

Terraform users (everyone)

- Use Terraform modules by specifying correct values
- Domain experts
- May not have "Terraform" in LinkedIn profile

Terraform 0.12 for developers

- DevOps&Terraform developers
- Allow to implement flexible/dynamic/reusable Terraform modules

Terraform 0.12 for users

- Terraform users
  - Like HCL2 lightweight syntax more

@antonbabenko

Some of my favourite Terraform Best Practices

- https://www.terraform.io/
- http://learn.hashicorp.com
- https://www.terraform-best-practices.com/
- + ask me for more

@antonbabenko

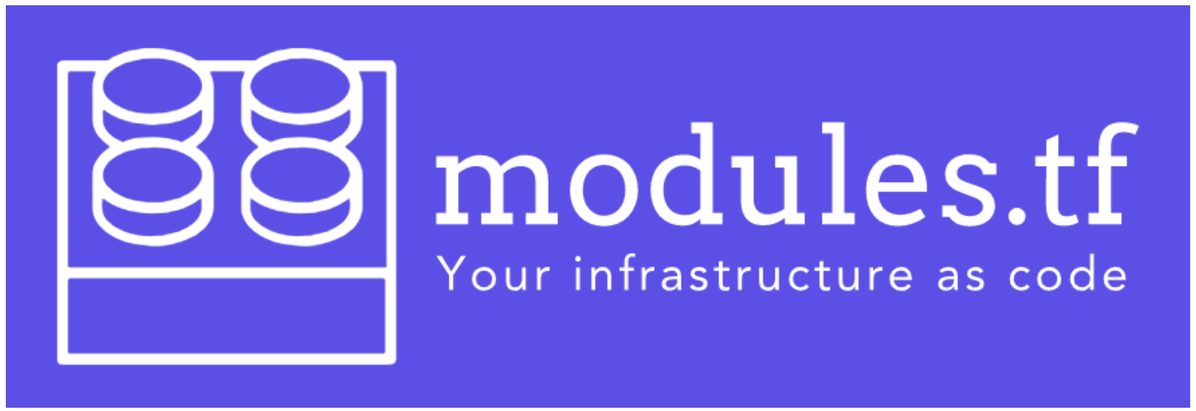
Your turn… What you want to do?

- Build real infrastructure using https://github.com/terraform-aws-modules
- BYOP (Bring Your Own Problem)

Build real infrastructure

- https://github.com/antonbabenko/terraform-best-practices-workshop
- git clone git@github.com:antonbabenko/terraform-best-practices-workshop.git
- Read "Attendee’s checklist" in README.md
- Follow the agenda in README.md

Bonus


modules.tf
Your infrastructure as code

Current blueprint
Web App Reference Architecture

LIBRARY

WHAT'S NEW

DESIGN | LIVE | BUDGET

EXPORT

AWS Components

Common

BLOCK

TEXT LABEL

ICON

IMAGE

AREA

Compute

AUTO SCALING
Auto scale EC2 group capacity

EC2
Elastic virtual server

LAMBDA
Run code in response to events

ZONE
Availability zone

Storage

EBS
EC2 block storage volume

S3
Simple object storage

EFS
Elastic file system

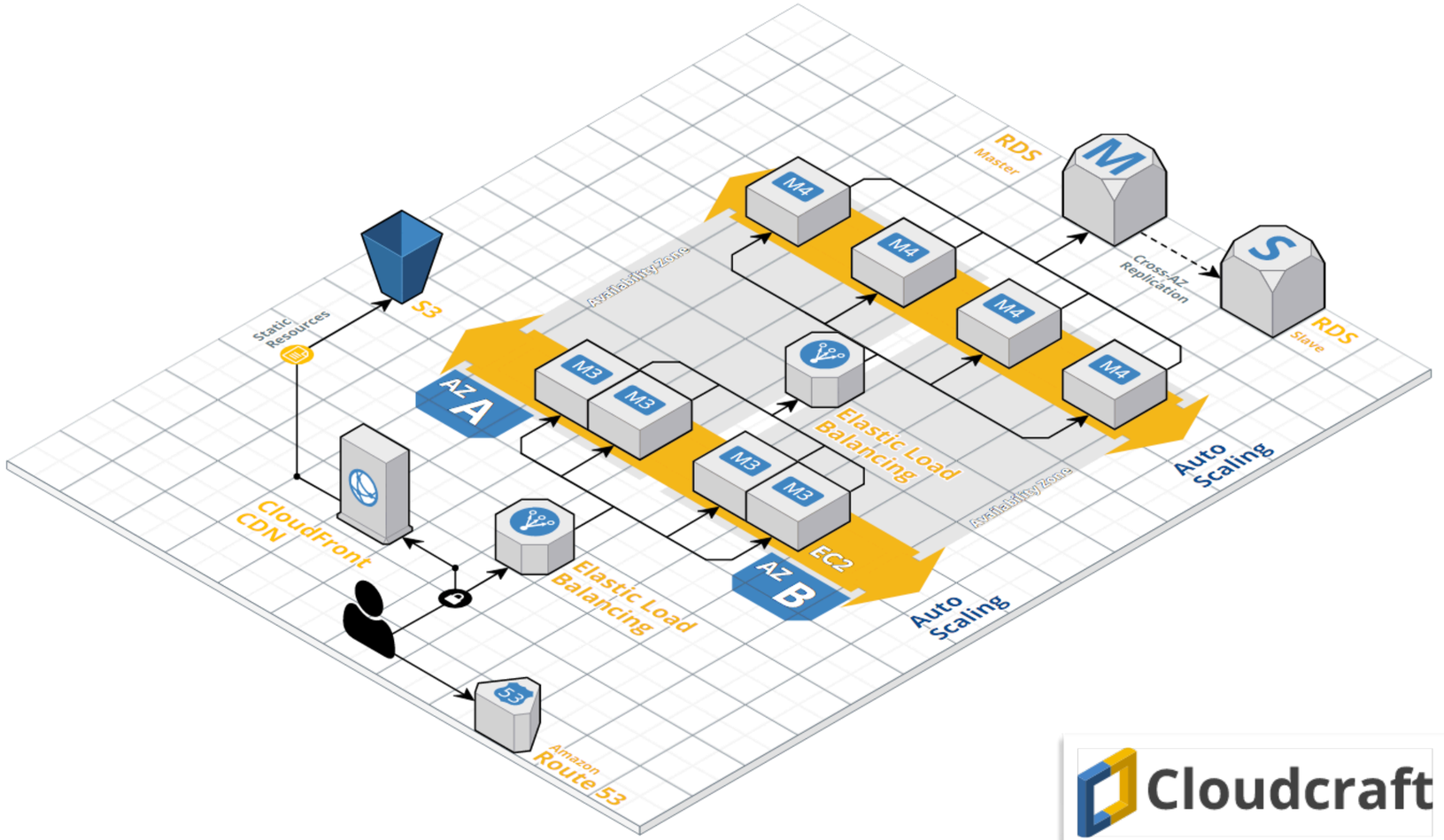
CLOUDFRONT
Content delivery network

Networking

ELB
Elastic load balancer

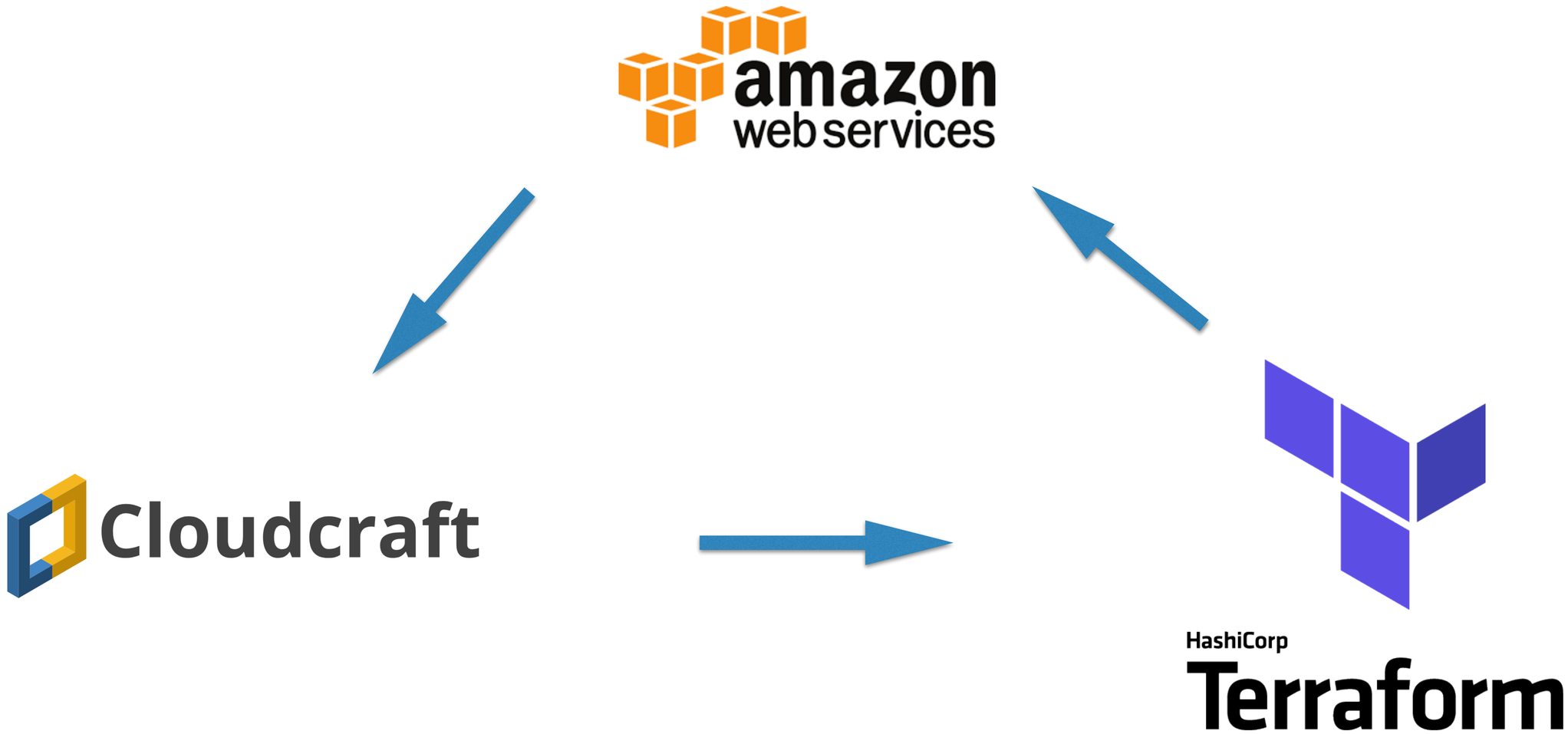
ROUTE 53
Managed DNS service

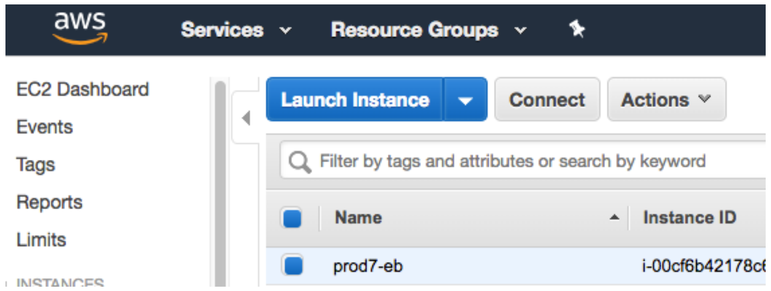
VPC GATEWAY
Access instances in your VPC

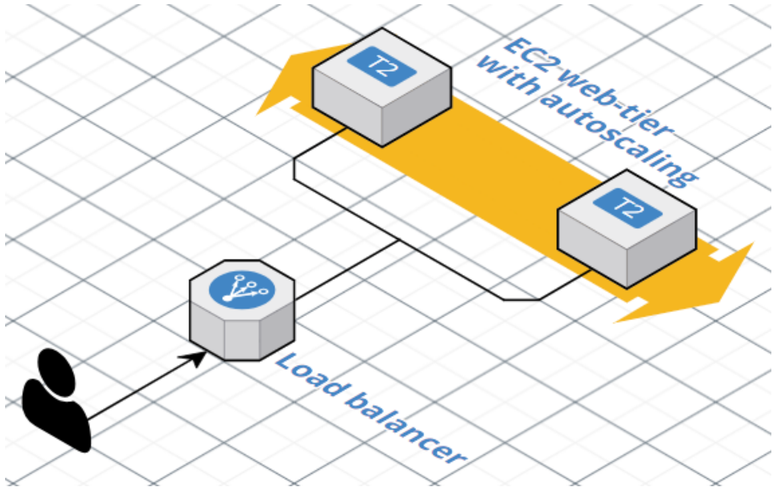


Cloudcraft

Drag to create multi-selection

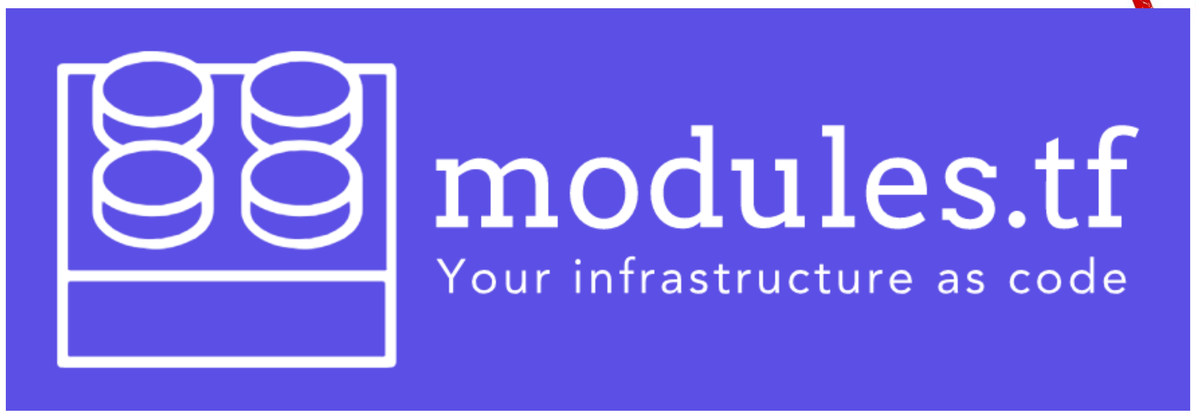

amazon
web services
Cloudcraft
HashiCorp
Terraform






```hcl
variable "aws_region" {
  description = "Region where resources should be created"
  default     = "eu-west-1"
}

provider "aws" {
  region = "${var.aws_region}"
}

resource "aws_s3_bucket" "this" {
  bucket = "my-bucket-${random_pet.bucket.id}"
}

resource "random_pet" "bucket" {
  keepers = {
    aws_region = "${var.aws_region}"
  }

  length = 1
}

output "this_s3_bucket_id" {
  description = "ID of S3 bucket"
  value       = "${aws_s3_bucket.this.id}"
}
```

- ✓ cloudcraft.co — design, plan and visualize
- ✓ terraform-aws-modules — building blocks of AWS infrastructure
- ✓ Terraform — infrastructure as code

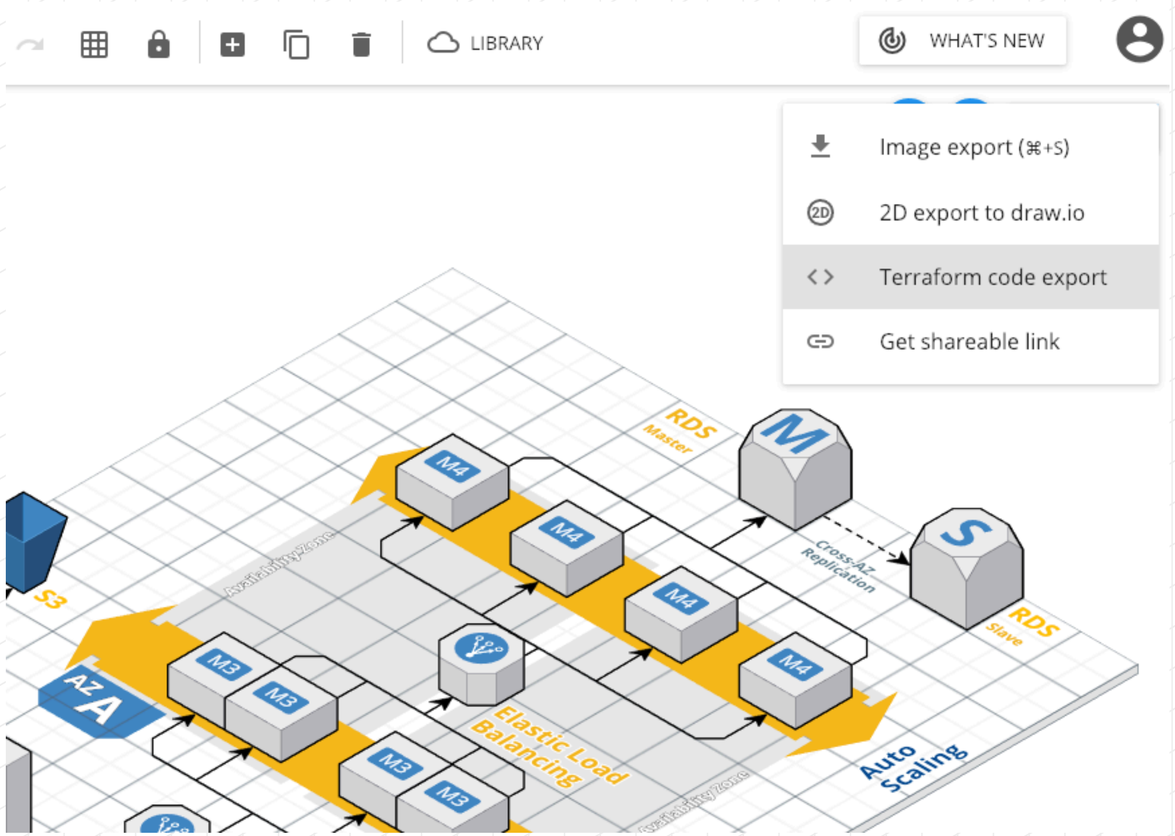


Infrastructure as code generator — from visual diagrams to Terraform

https://github.com/antonbabenko/modules.tf-lambda

Demo video: https://www.youtube.com/watch?v=F1Ax1zfZbiY

Try it yourself!



1. Go to cloudcraft.co
2. Sign up, sign in (free account)
3. Draw your AWS infrastructure
4. Click "Export"
5. Click "Terraform code export"

modules.tf — generated code

- ✓ Potentially ready-to-use Terraform configurations
- ✓ Suits best for bootstrapping
- ✓ Enforces Terraform best-practices
- ✓ Batteries included (terraform-aws-modules, terragrunt, pre-commit)
- ✓ 100% free and open-source (https://github.com/antonbabenko/modules.tf-lambda)
- ✓ Released under MIT license

Thanks!

Questions?

github.com/antonbabenko

twitter.com/antonbabenko