CONTINUOUSLY DELIVERING INFRASTRUCTURE USING TERRAFORM AND PACKER

Hello!

I AM ANTON BABENKO

I enjoy AWS, DevOps, solutions architecture & web-development.

github.com/antonbabenko

linkedin.com/in/antonbabenko

0.
AGENDA

0.
AGENDA

1. State of things
2. Basics of Terraform and Packer
    - Getting started demo
3. More advanced concepts in Terraform
    - Practice
4. Working as a team
    - CI/CD pipeline with Terraform and Packer
    - Practice
5. Resources

1.

STATE OF THINGS

Tools for AWS & Infrastructure as code

AVAILABLE TOOLS

AWS CloudFormation, Google Deployment Manager

Puppet, Chef, Ansible, Salt…

AWS API, libraries (Boto, Fog)

Terraform & Packer by HashiCorp

Packer

- Tool for creating identical machine images for multiple platforms from a single source configuration.
- Packer is lightweight, runs on every major operating system, and is highly performant, creating machine images for multiple platforms in parallel.

HASHICORP

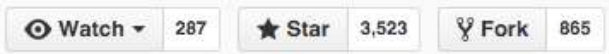
www.packer.io

TERRAFORM

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently.

www.terraform.io

TERRAFORM FACTS (2015)



Version: 0.6.8 (released 2.12.2015)

Open-source, written in Golang.

Very active development:

- CHANGELOG.md (ca. 1 release per month)
- GitHub Issues (ca. 5-15 issues resolving daily)
- Growing community (IRC, Mailing list, Stack Overflow)

TERRAFORM FACTS (2017)

Latest version: 0.9.4 (released 26.4.2017)

Open-source, written in Golang.

Very active development:

- CHANGELOG.md (ca. 3 releases per month)
- GitHub Issues (10+ issues resolving daily)
- Growing community (IRC, Mailing list, Stack Overflow, Slack channels, Gitter, etc)

TERRAFORM VS CLOUDFORMATION

Year 2015	**CloudFormation**	**Terraform**
Configuration format	JSON	HCL/JSON
State management	No	Yes
Execution control	No	Yes!
Logical comparisons	Yes	Limited
Supports iterations	No	Yes
Manage already created resources	No	Yes (hard)
Providers supported	Only AWS	20+ (incl. AWS, GCE, Azure)

Year 2017	**CloudFormation**	**Terraform**
Configuration format	**YAML**/JSON	HCL/JSON
State management	Kind of	Yes
Execution control	**Yes**	Yes!
Logical comparisons	Yes	**Yes**
Supports iterations	**Yes**	Yes
Manage already created resources	No	**Yes!**
Providers supported	Only AWS	60+ (incl. AWS, GCE, Azure)

AWS SPECIFICS

	CloudFormation (2015)	Terraform 0.6.8 (2015)	Terraform 0.9.4 (2017)
AWS resource types	121	103	280
Resource properties and operations completeness	90%	Work in progress	Work in progress :)
Handle failures	Optional rollback	Fix it & retry	Exit faster. Fix it & retry
Contribute?	No	Yes!	Yes!

2.
TERRAFORM

Commands

TERRAFORM COMMANDS

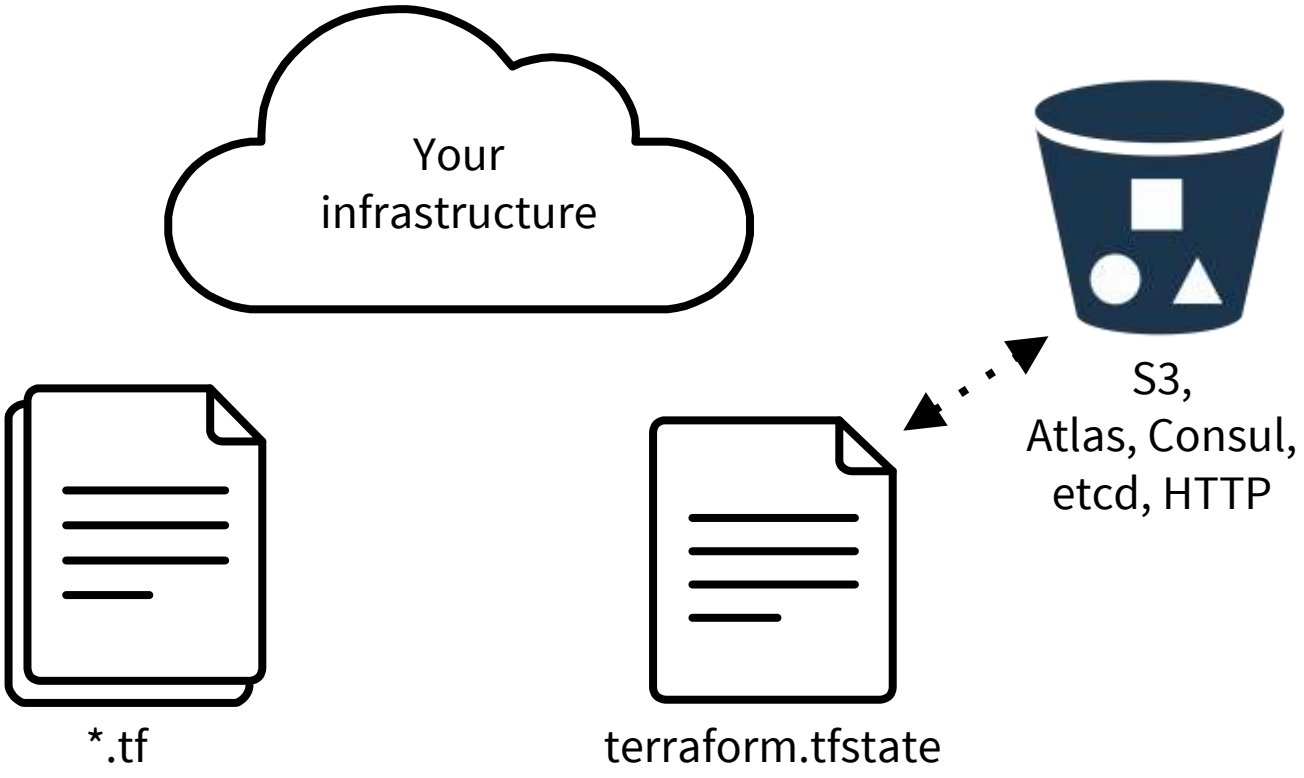
```
$ terraform
Usage: terraform [--version] [--help] <command> [args]

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Environment management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the configuration
    graph              Create a visual graph of Terraform resources
    import             Import existing infrastructure into Terraform
    init               Initialize a new or existing Terraform configuration
    output             Read an output from a state file
    plan               Generate and show an execution plan
    push               Upload this Terraform module to Atlas to run
    refresh            Update local state file against real resources
    show               Inspect Terraform state or plan
    taint              Manually mark a resource for recreation
    untaint            Manually unmark a resource as tainted
    validate           Validates the Terraform files
    version            Prints the Terraform version

All other commands:
    debug              Debug output management (experimental)
    force-unlock       Manually unlock the terraform state
    state              Advanced state management
```

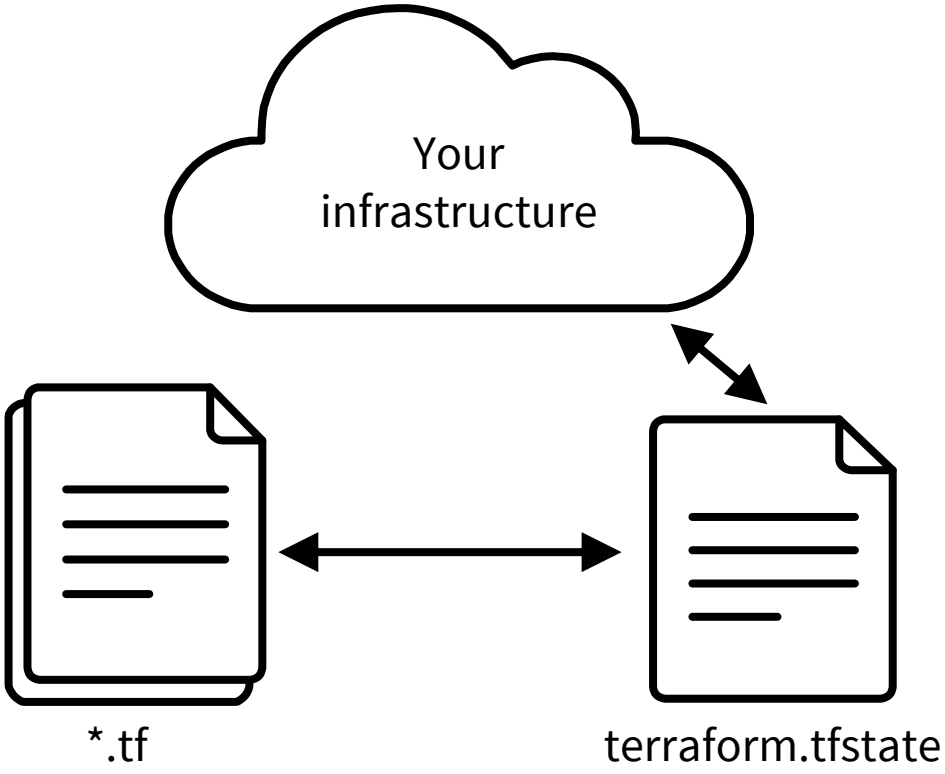
TERRAFORM INIT

Initialize a new or existing Terraform environment by creating initial files, loading any remote state, downloading modules, etc.



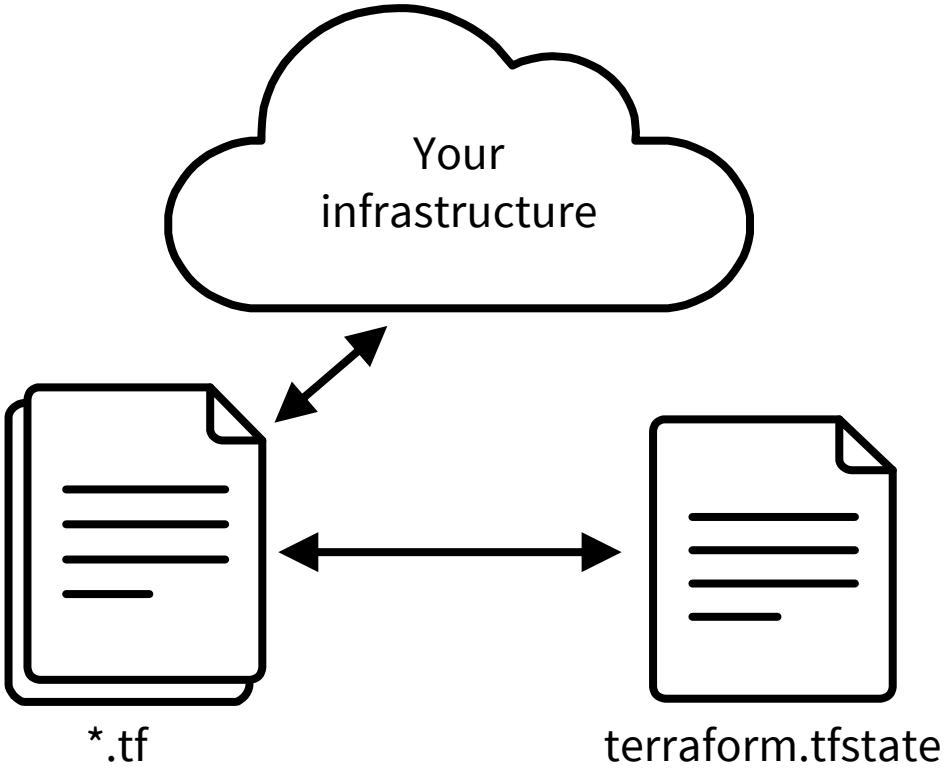
TERRAFORM PLAN

Generates an execution plan for Terraform



TERRAFORM APPLY

Builds or changes infrastructure according to Terraform configuration files



TERRAFORM etc

```
# Draw dependency graph (require “graphviz”)
terraform graph -draw-cycles | dot -Tpng -o graph.png

# Show help
terraform --help
```

TERRAFORM & PACKER DEMO1

Code inside **{terraform,packer}/demo1**:

https://github.com/antonbabenko/cd-terraform-demo

3.
TERRAFORM

More advanced...

TERRAFORM AHEAD

Variables

Modules

States

Backends

Data sources, providers, provisioners

Conditions

TERRAFORM - MODULES

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

```
module "network_security" {
  source   = "git::git@github.com:myself/tf_modules.git//modules/network/security?ref=v1.0.0"

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

Links:

https://github.com/terraform-community-modules/

Lots of github repositories (588)

TERRAFORM - VARIABLES

Terraform != programming language

Types: string, number, boolean, list, map

Interpolation functions: length, element, file …

```
variable "iam_users" {
 description = "List of IAM users to create"
 type = "list"
}

resource "aws_iam_user" "users" {
 count = "${length(var.iam_users)}"
 name = "${element(var.iam_users, count.index)}"
}
```

Links:

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

TERRAFORM - RESOURCES

```
resource "aws_autoscaling_group" "application" {
 name = "${var.name}"
 launch_configuration = "${aws_launch_configuration.application.name}"
 vpc_zone_identifier = ["${module.public_subnet.subnet_ids}"]

 depends_on = ["module.s3_artifacts"]

 tag {
   key = "Name"
   value = "${var.name}"
   propagate_at_launch = true
 }

 lifecycle {
   create_before_destroy = true
   ignore_changes = ["desired_capacity"]
 }
}
```

Links:

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

TERRAFORM - DATA SOURCES

```
data "aws_ami" "ami" {
 most_recent = true

 filter {
   name = "name"
   values = ["ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-*"]
 }

 owners = ["099720109477"] // Canonical
}

resource "aws_launch_configuration" "application" {
 image_id = "${data.aws_ami.ami.image_id}"
}
```

Links:

https://www.terraform.io/docs/configuration/data-sources.html

TERRAFORM - OUTPUTS

```
output "application_name" {
 value = "${var.name}"
}

output "vpc_id" {
 value = "${module.vpc.vpc_id}"
}
```

Links:

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

TERRAFORM - STATES & BACKENDS

Terraform keeps state of managed infrastructure and configuration in “terraform.tfstate”.

```
terraform {
 backend "s3" {
   bucket     = "my-tf-states"
   key        = "staging/eu-west-1/shared"
   region     = "eu-west-1"
   lock_table = "terraform_locks"
 }
}
```

Links:

https://www.terraform.io/docs/state/index.html

https://www.terraform.io/docs/backends/index.html

TERRAFORM - REMOTE STATES

```
data "terraform_remote_state" "shared" {
 backend = "s3"

 config {
   bucket  = "my-tf-states"
   region  = "eu-west-1"
   key     = "staging/eu-west-1/shared"
   encrypt = true
 }
}

output "vpc_id" {
 value = "${data.terraform_remote_state.shared.vpc_id}"
}
```

Links:

https://www.terraform.io/docs/providers/terraform/d/remote_state.html

TERRAFORM - CONDITIONS

```
# Example: If ... then
resource "foo" "bar"  {
 count = "${var.enable_ssl}" # true => 1, false => 0
}
```

```
# Example: If not ... then
resource "foo" "bar"  {
 count = "${1-var.enable_ssl}" # true => 1, false => 0
}
```

```
module "application" {
 is_feature = "${replace(replace(terraform.env, "/^[^(feature)].*/", "false"), "/^feature.*/", "true")}"
}
```

Links:

https://blog.gruntwork.io/terraform-tips-tricks-loops-if-statements-and-gotchas-f739bbae55f9

TERRAFORM DEMO2

Code inside **terraform/demo2** :

https://github.com/antonbabenko/cd-terraform-demo

4.
TERRAFORM

Working as a team...

TERRAFORM HOW?

- How to structure your configs?

  Reduce radius blast

    Size matters a lot

    Structure based on teams (infrastructure team-members = network; developers = modules owners)

    Separate repositories for modules and infrastructure

    Infrastructure *can* share same repository as application

- How to continuously **test** infrastructure using Terraform?

  Validate, plan, env

    Test modules independently, include working examples and README

    Test Kitchen, Inspec, Serverspec…

    Full run with smaller (yet, sane!) values

TERRAFORM WORK FLOW

Init, plan, *apply, apply*, plan, apply…

Executors:

- Single developer
- Multiple developers
    - Requires remote backend configuration (locks for lengthy operations)
- CI system

Notes:

- MFA?
- Module versioning is important
- Group code by both - region and environment (staging, prod)

TERRAFORM WORK FLOW

Init, plan, *apply, apply*, plan, apply…

Open a Pull request:

Validation (terraform validate)

Optionally: Create new ephemeral (short-lived) Terraform environment (“terraform env new feature-branch”), run automated tests (kitchen-terraform, for example) and destroy it after

Run plan and display output for review (git comment)

Branch merged into master:

Terraform apply to staging

Optionally: terragrunt apply-all

Branch tagged (release):

Terraform apply to production

TERRAFORM - EXAMPLE 1 (pseudo)

- Developer commits application code
- CI system:
    - Run tests, builds artifact
        - Packer: Bake AMI
        - Terraform: Plan and apply with just created AMI id to create deployment
        - Run integration, performance tests
        - Deploy to staging

TERRAFORM - EXAMPLE 1 - feature

- Developer commits application code to a **feature branch** name **feature-123**
- CI system:
  - Run tests, builds artifact using Packer
  - Run Packer: Bake AMI and tag it with **branch=feature-123**
  - Run Terraform:
    - Plan the infrastructure for **test** environment, where AMI id lookup is using data source **ami** by tag **branch=feature-123**
    - Optionally, save plan to a file, prompt git user in UI, post comment to github PR
    - Apply the plan
  - Run integration, performance tests
  - Deploy to staging

TERRAFORM DEPLOYMENTS

Rolling deployments

- Using provider's mechanisms:
  - ECS (or other scheduler)
  - CloudFormation
- Using custom mechanisms:
  - DIY scripts combined with '-target' arguments

Blue-green deployments

- No provider's mechanisms for this
- DIY

5. TERRAFORM RESOURCES

TERRAFORM RESOURCES

Books and blog posts:

- Getting Started with Terraform by Kirill Shirinkin
- Terraform: Up and Running: Writing Infrastructure as Code by Yevgeniy Brikman
- Infrastructure as Code: Managing Servers in the Cloud by Kief Morris
- Using Pipelines to Manage Environments with Infrastructure as Code by Kief Morris

Tools:

- https://github.com/gruntwork-io/terragrunt
- https://github.com/dtan4/terraforming
- https://github.com/coinbase/terraform-landscape
- https://github.com/newcontext-oss/kitchen-terraform
- https://github.com/kvz/json2hcl

Other relevant repositories:

THANK YOU!

All code from this talk:

https://github.com/antonbabenko/cd-terraform-demo