Infrastructure as Code

QS Data Science Sharing

22nd July 2021

Content

- What is IAC?
- Some tools for IAC
- Terraform
- Demo

DevOps tasks **before** automation

- Lets say you wrote an application and you want to deploy that on your server.
    - Setup servers
    - Configure networking
    - Create route tables
    - Install software
    - Configure software
    - Install DB
    - …
- Done manually by system administrators

Only for setup

Maintenance?

Multiple environments?

Issues

- Cost
- Scalability and availability
- Inconsistency

Introducing IAC

- **Wikipedia**

  Infrastructure as code is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

- **A simpler definition**

  Infrastructure as code (IaC) means to manage your IT infrastructure using configuration files.

Why would you want to do that?

- Speed
  - On premise set up without IAC – Total: ~11 months + ~3-6 months (CSG clearance)
  - GCC set-up without IAC – Total: ~81 days + ~3-6 months (CSG clearance)
  - GCC set-up with IAC – Total: ~41 days + ~1 month (CSG clearance)
- Consistency
- Security
  - https://www.developer.tech.gov.sg/2021/07/22/security-benefits-iac.html
- Lower cost
- Make software development life cycle more efficient

Some common tools for IAC

HashiCorp Terraform

ANSIBLE

puppet

CloudFormation

CHEF

Broad differences between IAC tools

	Terraform	**Chef**	**AWS CloudFormation**	**Puppet**	**Ansible**
Code	Open source	Open source	Closed source	Open source	Open source
Type	Orchestration	Configuration management	Orchestration	Configuration management	Configuration management
Cloud	All providers	All providers	Limited to AWS only	All providers	All providers
Language	Declarative	Declarative	Declarative	Declarative	Procedural
Infrastructure	Immutable	Mutable	Immutable	Mutable	Mutable
Architecture	Client only	Client-server	Client only	Client-server	Client only

Differences between IAC tools

- Declarative vs Procedural

Declarative

End result:

- 2 servers
- 1 graph database
- 2 security groups

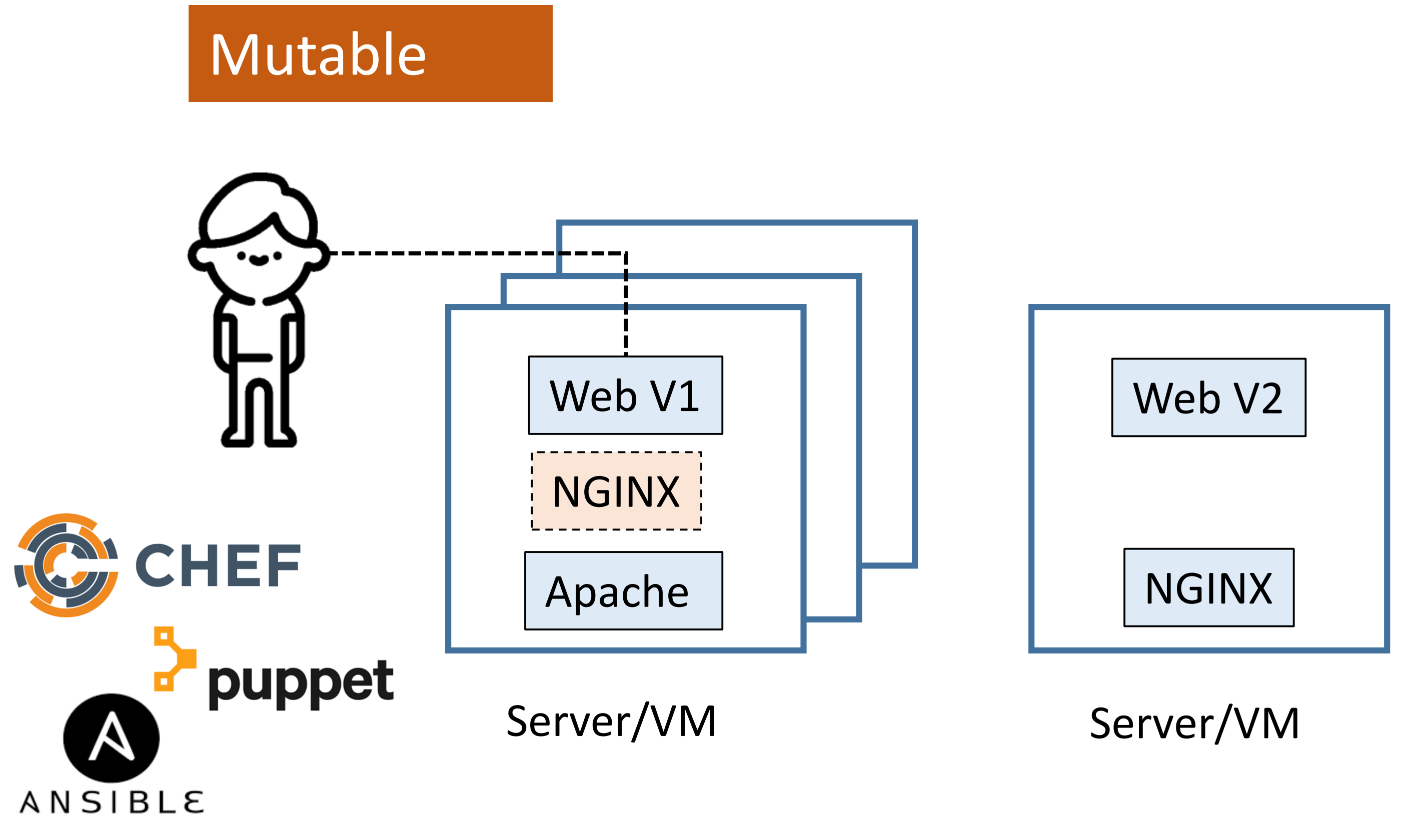
Procedural

Step by step:

1) Create server
2) Add server
3) Make this change
4) …

Differences between IAC tools

- Mutable vs Immutable

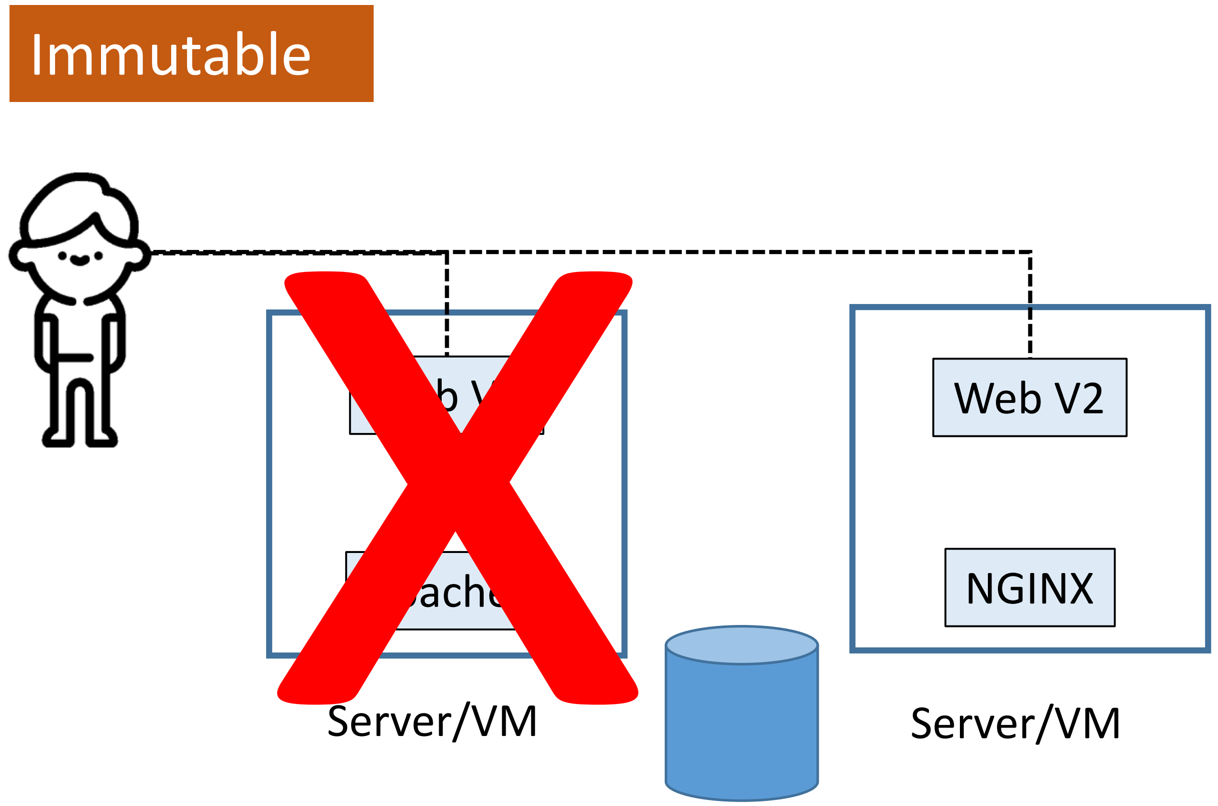


+ Retain previous data without worrying about obtaining new infrastructure
+ Don't need to build servers every time a change occurs

- Updates can fail due to several reasons
- Configuration drift, complexity
- Indiscrete versioning

Differences between IAC tools

- Mutable vs Immutable



+ Discrete versioning
+ Predictability
+ Easy to roll back deployments

- Cannot modify existing servers.
- Need to externalize data storage

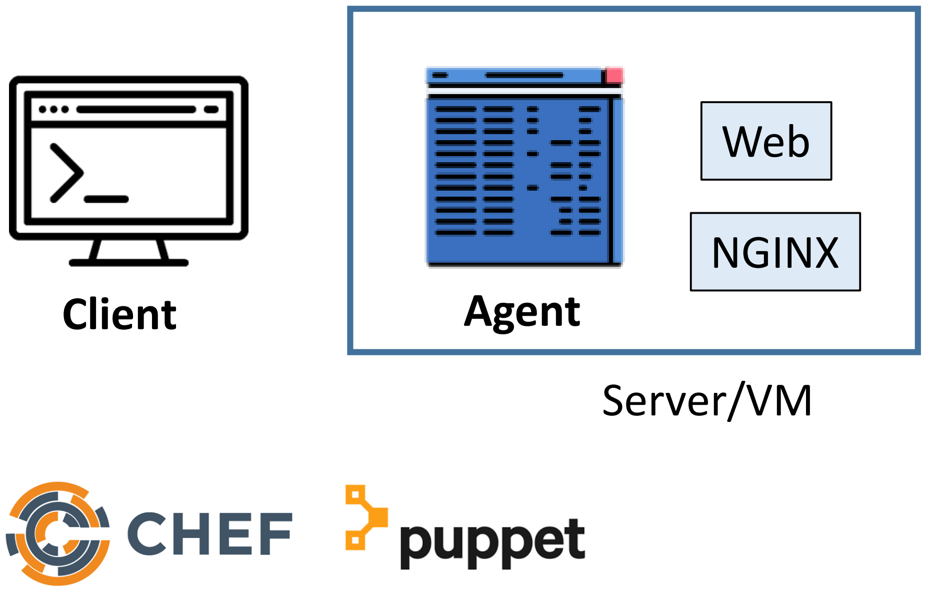
Differences between IAC tools

- Orchestration vs Configuration Management
  - **Orchestration** addresses the requirement to provision environments at a higher level than configuration management. The focus here is on coordinating configuration across complex environments and clusters.
  - **Configuration management** refers to the process of systematically handling changes to a system in a way that it maintains integrity over time. Configure software and systems on infrastructure that has already been provisioned.
  - Configuration orchestration tools do some level of configuration management, and configuration management tools do some level of orchestration.
  - Many times, <u>combination of tools</u> are used – e.g. Terraform + Ansible.

Differences between IAC tools

- Client-Server Architecture vs Client Architecture

Client-Server

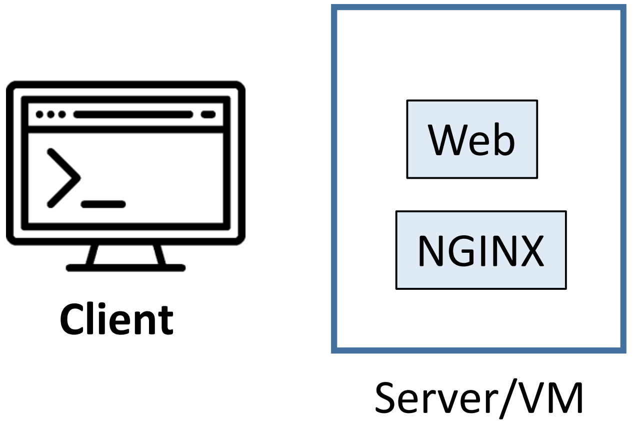


- Have to install and deploy extra software on every server. Also have to maintain, monitor, backups, upgrade etc.
- Increases surface area for attacks
- All of these extra moving parts introduce a large number of new failure modes into your infrastructure

Differences between IAC tools

- Client-Server Architecture vs Client Architecture

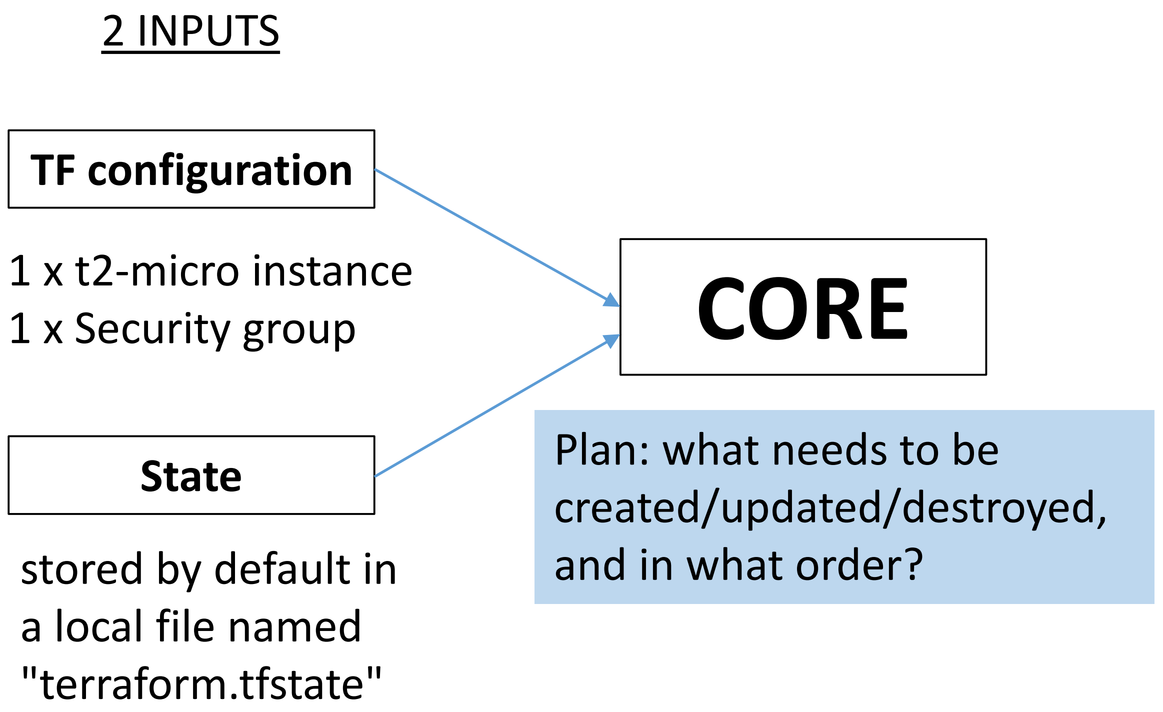
Client



- Ansible client works by connecting directly to servers via SSH.
- Terraform uses cloud provider APIs to provision infrastructure, no new authentication mechanisms.
- Cloud-formation considered client/server, but since AWS handles all the server details, as an end user we only have to think about client code.

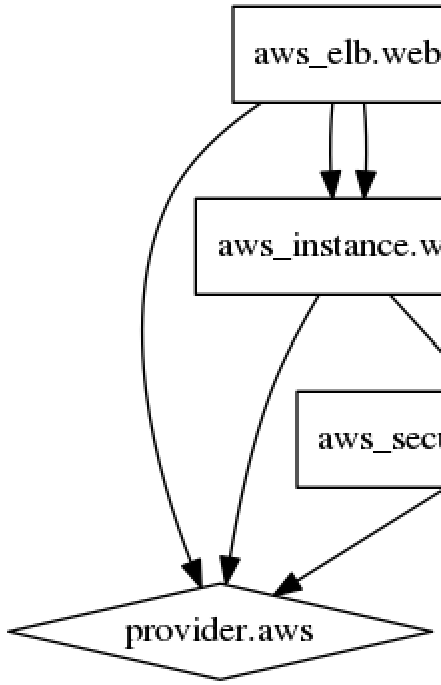
Terraform

- Mainly used for infrastructure provisioning.
- Declarative


2 INPUTS
TF configuration
1 x t2-micro instance
1 x Security group
State
stored by default in a local file named "terraform.tfstate"
CORE
Plan: what needs to be created/updated/destroyed, and in what order?

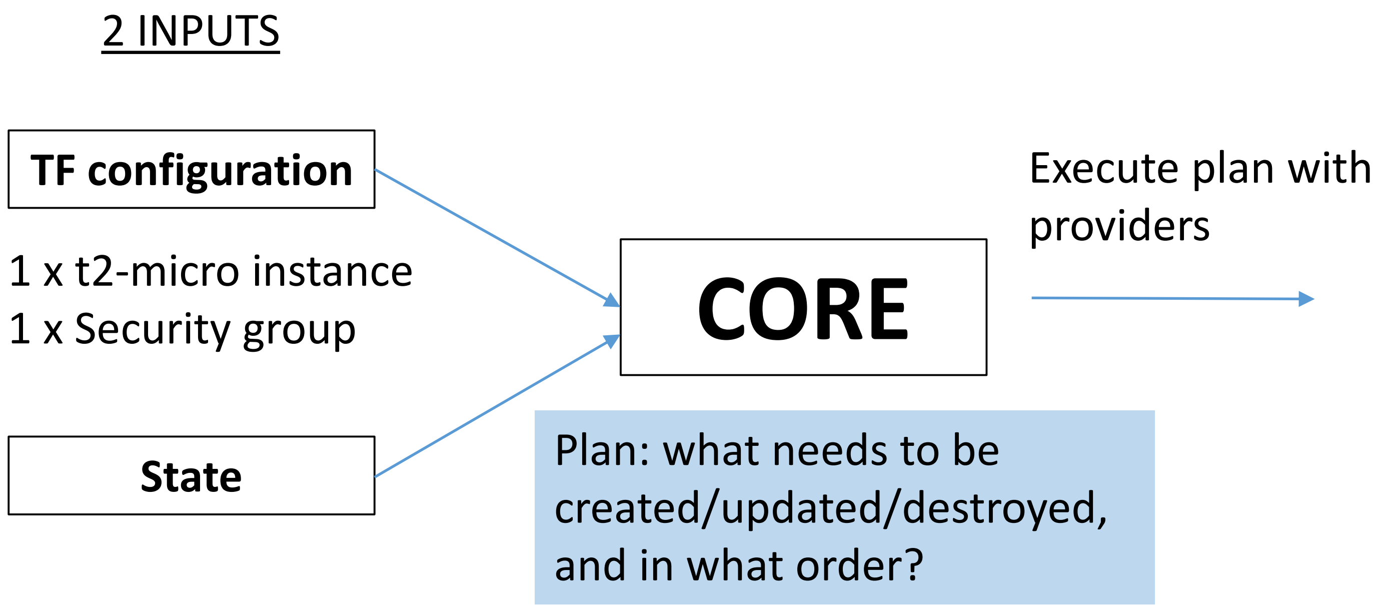
Resource graph

- Terraform builds a resource graph and creates or modifies non-dependent resources in parallel.



Terraform

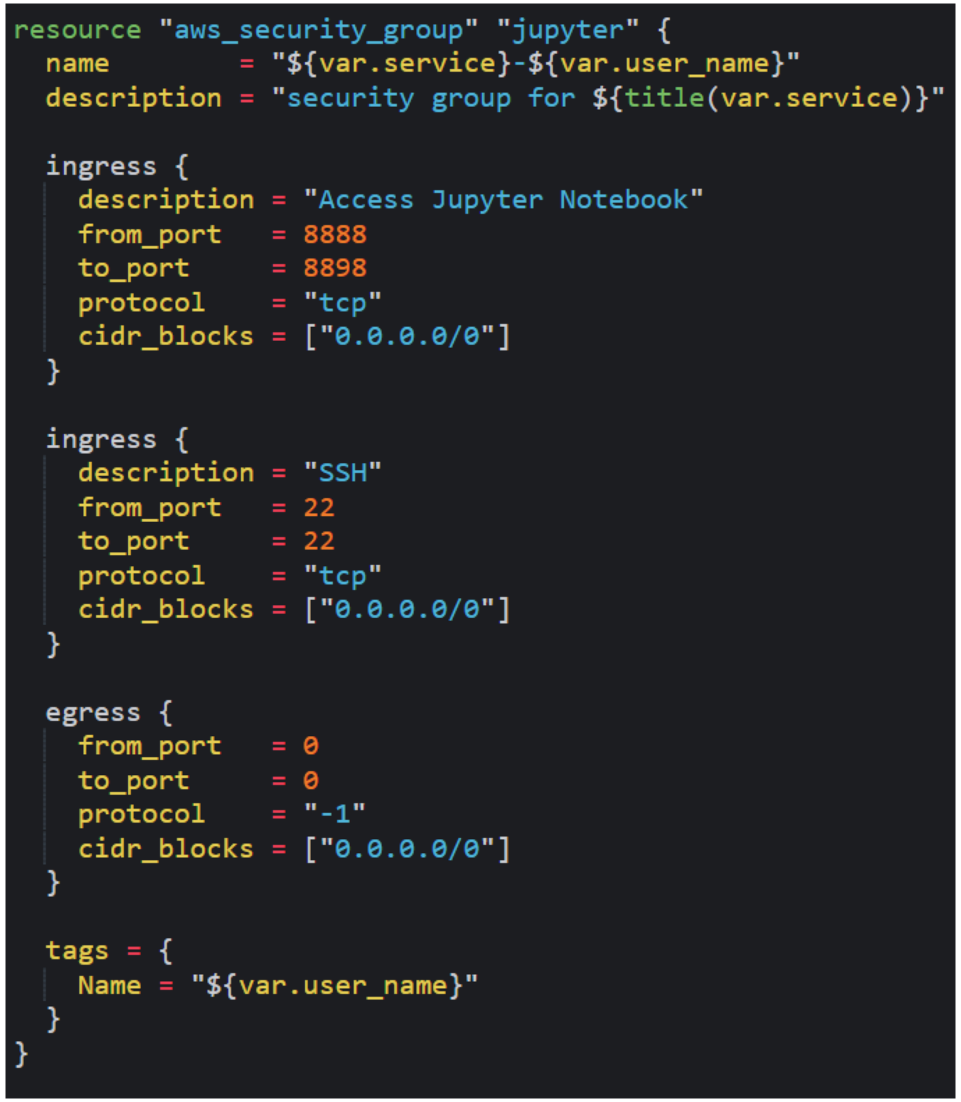
- Mainly used for <u>infrastructure provisioning</u>.
- Declarative



Plan: what needs to be created/updated/destroyed, and in what order?

Each provider adds a set of resource types and/or data sources that Terraform can manage. In the set up, we can have multiple providers.

Terraform commands



```
resource "aws_security_group" "jupyter" {
  name        = "${var.service}-${var.user_name}"
  description = "security group for ${title(var.service)}"

  ingress {
    description = "Access Jupyter Notebook"
    from_port   = 8888
    to_port     = 8898
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "${var.user_name}"
  }
}
```

INIT	Initialize a working directory that contains a Terraform configuration.
PLAN	Creates execution plan
APPLY	Executes plan.
DESTROY	Destroy resources or infrastructure

Links

https://github.com/loojovi/terraform-qs