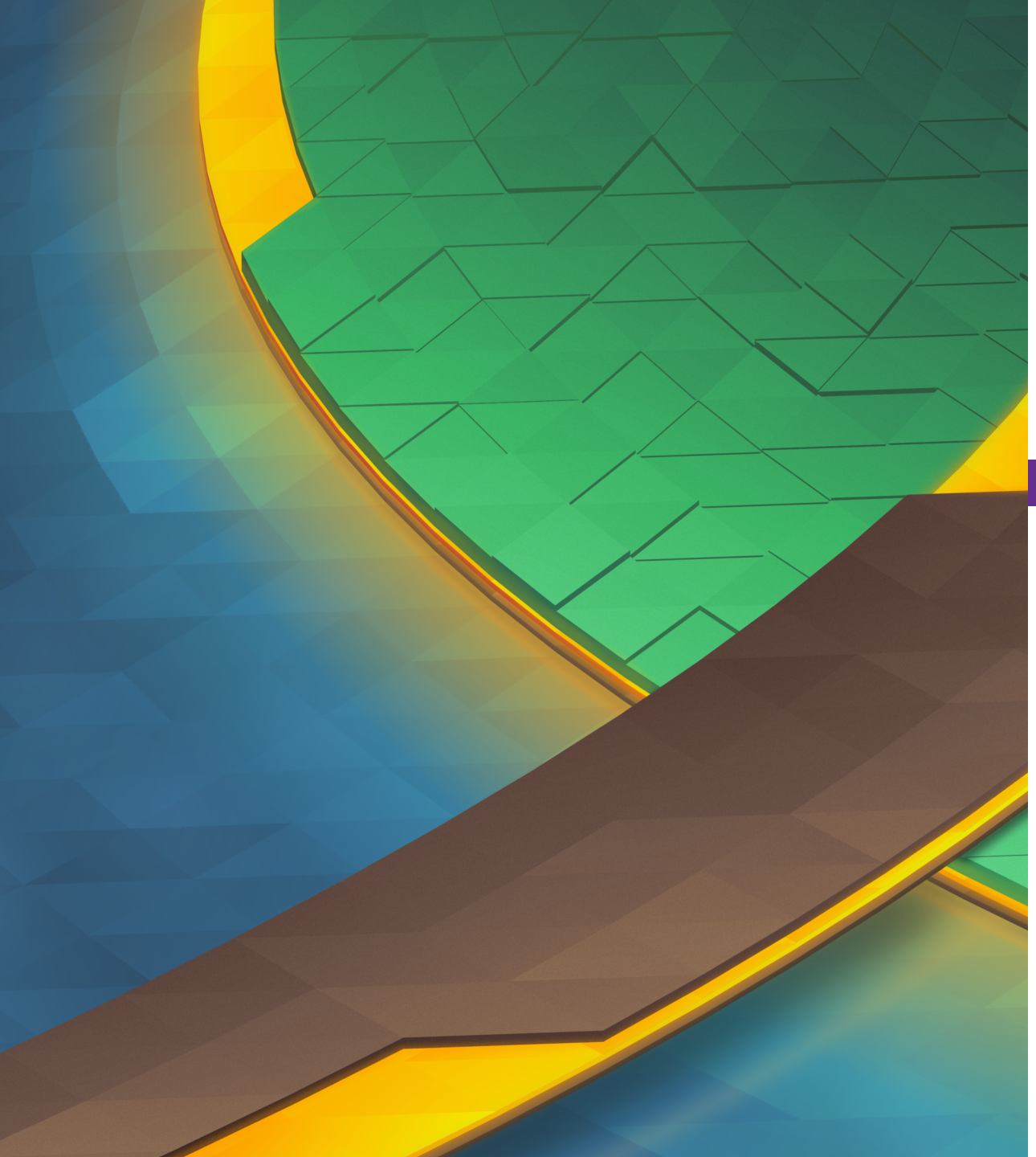
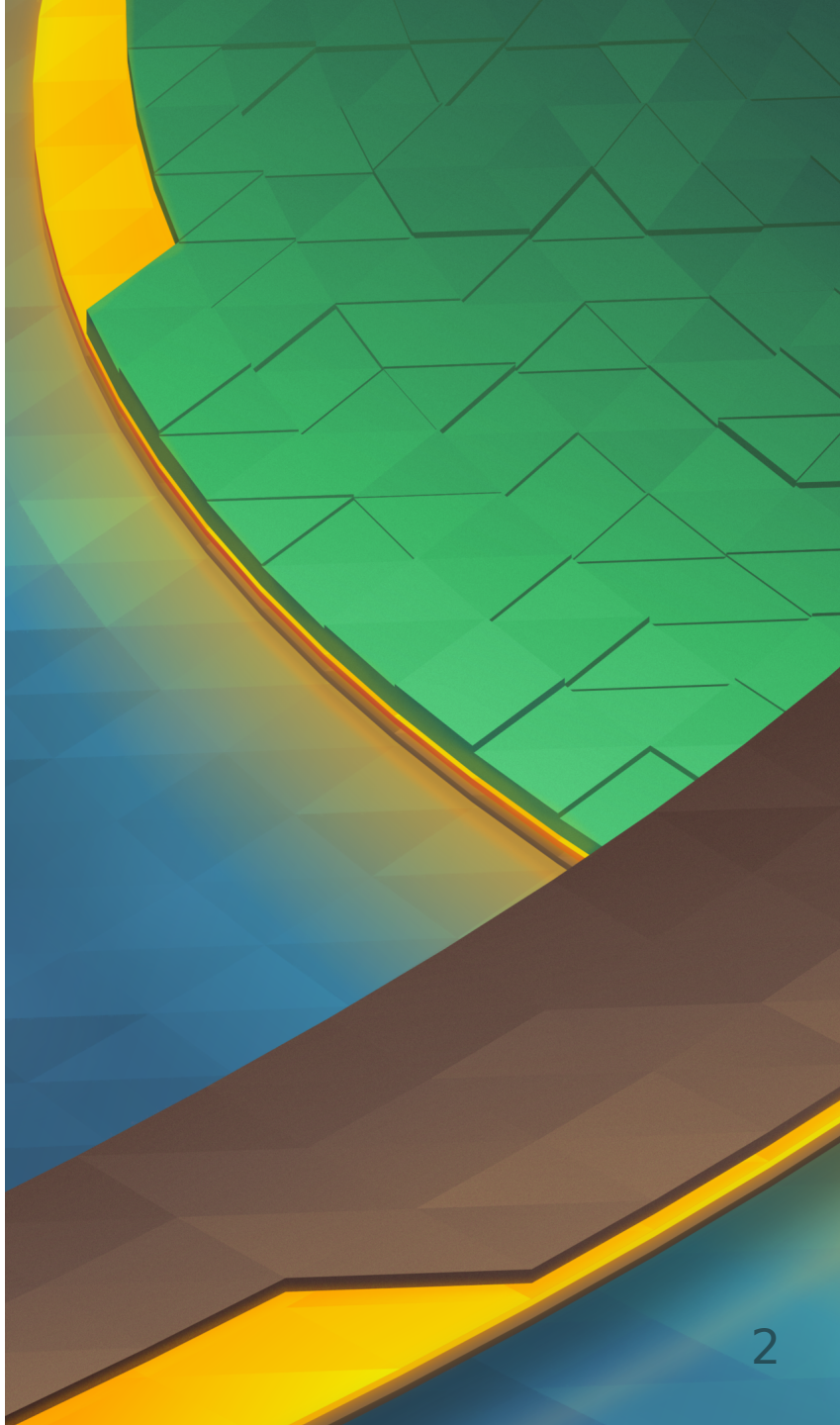


Terraform

Glenn Y. Rolland
<teaching@glenux.net>

Préambule



Objectifs de la formation

- Orchestrer des déploiements d’infrastructure avec Terraform
- Utiliser Terraform pour respecter les standards de l’Infrastructure as Code
- Structurer leurs projets pour les réutiliser efficacement
- Organiser leurs équipes pour travailler de concert autour de Terraform
- Reprendre en main une infrastructure existante pour la faire évoluer avec Terraform

Qui êtes vous ?

Petit tour de présentation... avec 3 questions

- Quel est votre "bagage" ? (expérience, compétences, etc.)
- Pourquoi participez vous à cette formation aujourd'hui ?
- Comment utiliserez-vous ces nouvelles compétences d'ici 2 ou 5 ans ?

Qui suis-je ?

2021 → aujourd'hui	**Auteur, conférencier et co-fondateur de CRYPTO-CHEMISTS** Formation et conseil sur l'impact des Blockchain & des technologies P2P.
2018 → aujourd'hui	**Directeur technique et co-fondateur de BOLDCODE** Développement et audit logiciel, web et mobile, offshoring éthique au Népal.
2017 → 2022	**Directeur technique et co-fondateur de DATA-TRANSITION** Gestion éthique des données, audit des SI, conformité au RGPD.
2010 → 2017	**Gérant et co-fondateur de NETCAT (GNUSIDE)** Infrastructures & systèmes en réseau, optimisation de la fiabilité, de la sécurité et de la performance.
2006 → 2010	**Ingénieur de recherche chez BEWAN (Pace Group)** Conception de systemes embarqués, et automatisation de la qualité logicielle.

La formation chez Orsys - en quelques mots

ORSYS formation

Learning for success

- 45 ans d'existance
- 7700 entreprises et administrations clientes
- 85000 personnes formées par an
- 1700 intervenants experts
- 97.5% de clients satisfaits

- 27 centres de formation en France
- 5000+ sessions intra-entreprise
- 7000+ sessions inter-entreprise
- 72,2M€ de chiffres d'affaires

Note: Chiffres de 2019

Déroulement de la formation & règles du jeu

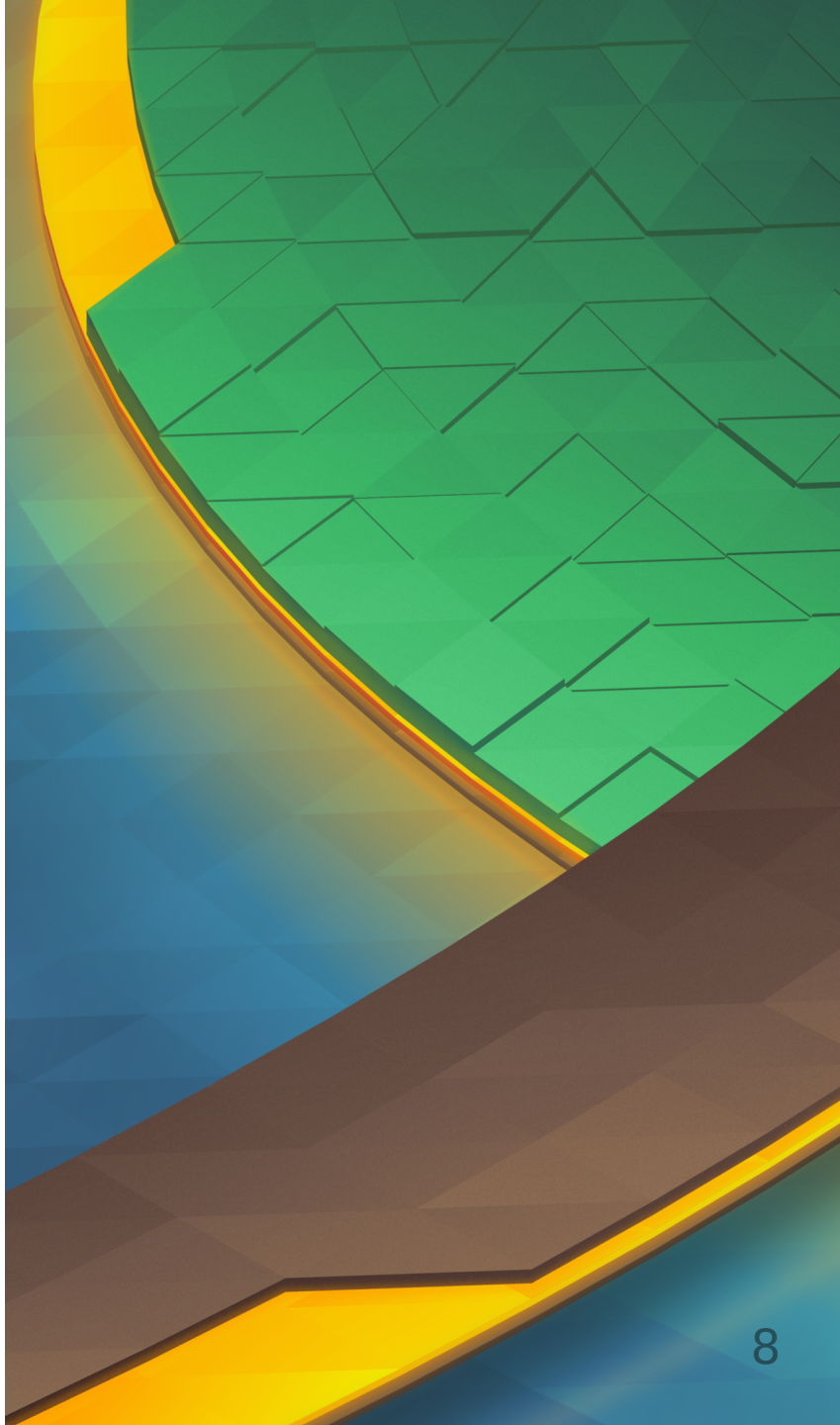
Horaires

- 9h00 - 12h30
- 13h30 - 17h00
- Des pauses le matin et l'après midi

Le cadre

- Liberté de parole dans le respect des autres et des objectifs de la formation
- Bienveillance, nous sommes dans un espace d’apprentissage
- Confidentialité de l’animateur et des participants sur les échanges

Introduction



À propos de Terraform

Hashicorp et écosystème de produits

Vagrant (2010): Construction et de gestion de infrastructure as code. Il permet de créer et de gérer des infrastructures de manière reproductible de façon virtualisée.

Packer (2013): Création d'images machine identiques (VM, etc.) pour plusieurs plateformes à partir d'une configuration source unique.

Consul (2014): Découverte, surveillance et configuration de services. Il est conçu pour être hautement disponible et peut être déployé dans un cluster.

Terraform (2014): Construction et de gestion de infrastructure as code. Il permet de créer et de gérer des infrastructures de manière reproductible et efficace directement chez les opérateurs cloud.

Vault (2015): Gestion des secrets, tels que les mots de passe, les clés API et les certificats. Il permet le stockage et l'accés aux secrets de façon sécurisée et offre des fonctionnalités telles que l'expiration, la révocation et l'audit.

Nomad (2016): Gestion des déploiements et de l'infrastructure. Il permet aux développeurs de décrire leurs applications de manière déclarative et de laisser Nomad gérer le déploiement, la mise à

Historique Terraform

- Créé le 22 mai 2014
- 89 versions
- URL: https://github.com/hashicorp/terraform

Projet Open Source

- Licence: Mozilla Public License 2.0
- 1129 contributeurs
- 9433 étoiles
- 3464 forks

Les providers

Les opérateurs les plus connus :

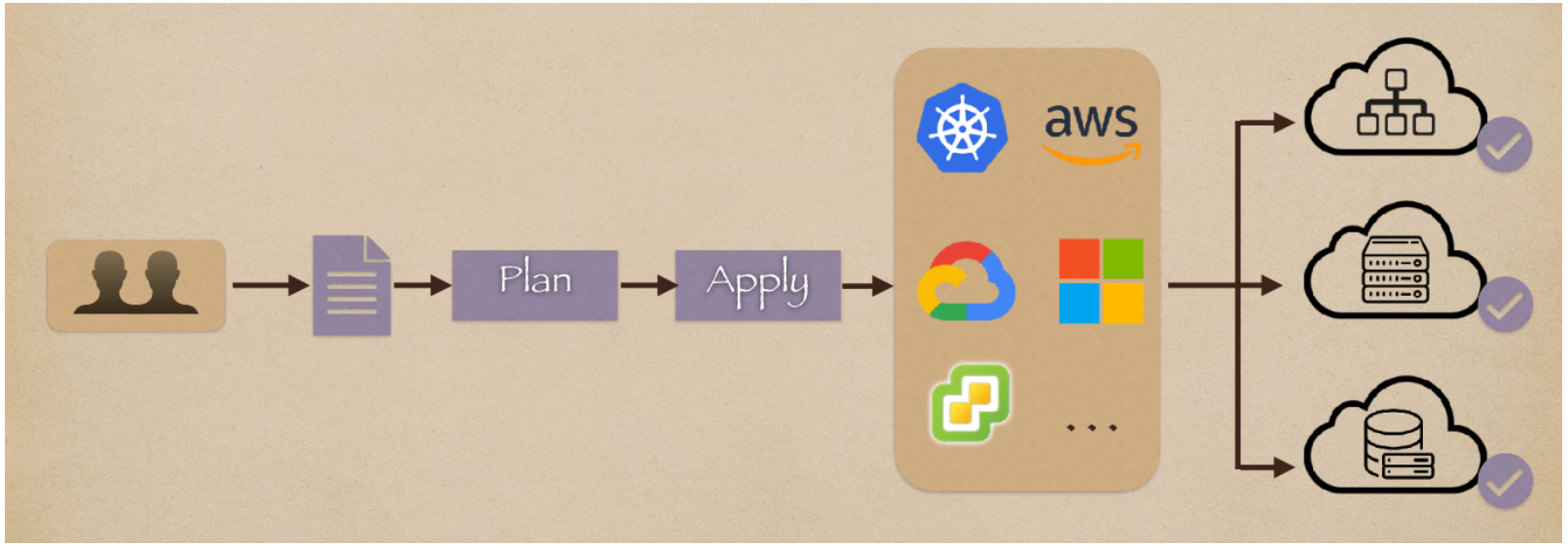
- Amazon (AWS)
- Google (GCP)
- Microsoft (Azure)
- Heroku
- Digital Ocean

Mais aussi :

> 1&1 Alicloud Archive Arukas AWS Bitbucket CenturyLinkCloud Chef Circonus Cloudflare CloudStack Cobbler Consul Datadog DNS DNSMadeEasy DNSimple Docker Dyn External Fastly GitHub Gitlab Grafana HTTP Icinga2 Ignition InfluxDB Kubernetes Librato Local Logentries Mailgun New Relic Nomad NS1 MySQL Oracle Public Cloud OpenStack OpsGenie OVH Packet PagerDuty PostgreSQL PowerDNS ProfitBricks RabbitMQ Rancher Random Rundeck Scaleway SoftLayer StatusCake Spotinst Template Terraform Terraform Enterprise TLS Triton UltraDNS Vault VMware vCloud Director VMware vSphere...

Certains de ces providers concernent des services qui n'ont rien à voir avec des hébergeurs cloud généralistes ! Ils sont en local ou à distance, et ils sont de nature très variée...

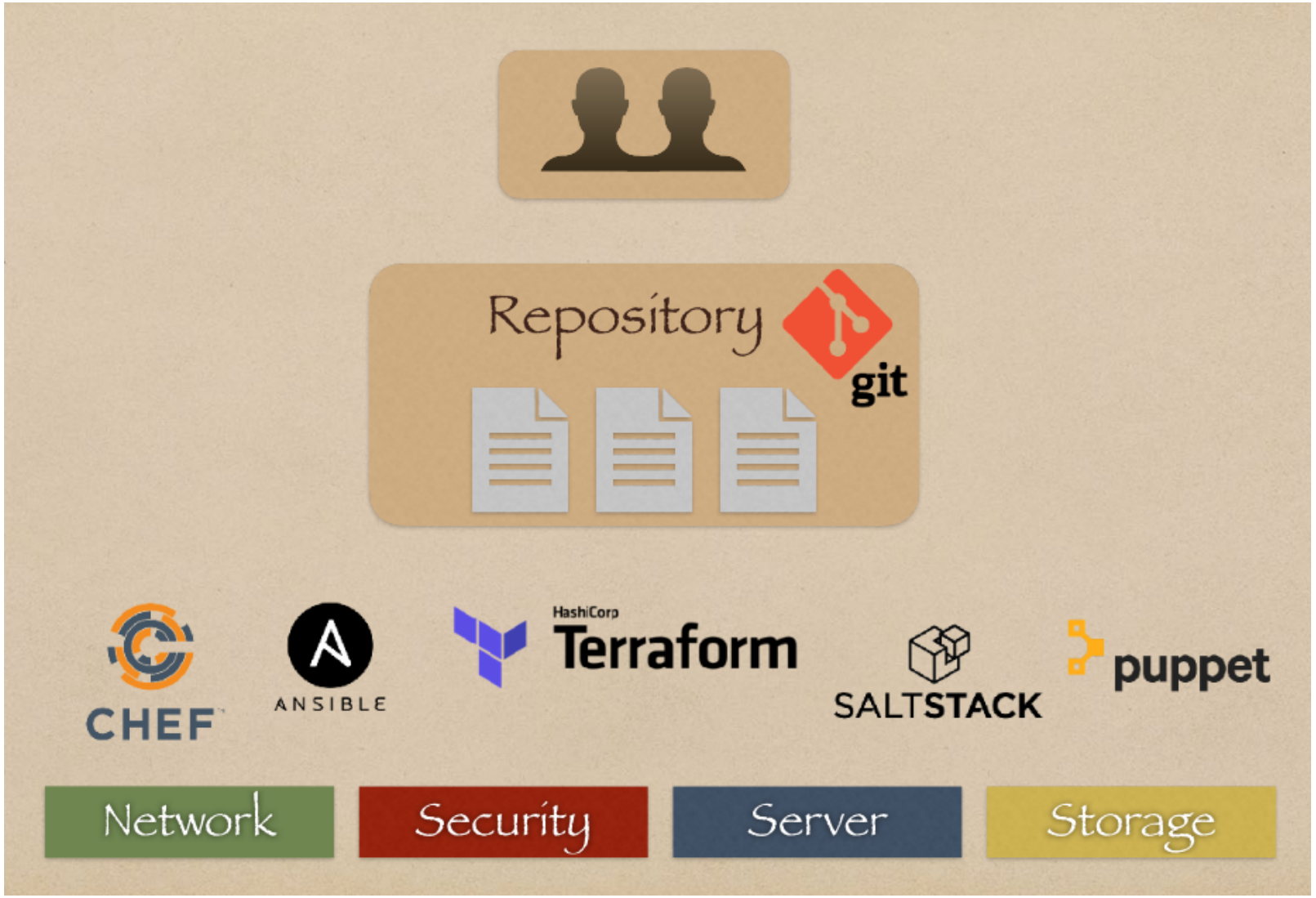
À quoi sert terraform ?



- **Définir une infrastructure** sous forme de code
  - De manière **déclarative et idempotente**
  - **Pour produire et garantir un état souhaité** automatiquement
- N'est pas une couche d'abstraction d'API
  - Chaque fournisseur a ses propres ressources !

Infrastructure As Code

Définition de IaC



IaC et typologies de provisioning

Software Provisioning

(aka "Configuration management")

- Chef
- Ansible
- Puppet
- Saltstack

Infrastructure Provisioning

- Vagrant
- Terraform
- Pulumi

IaC et mutabilité

Mutable

Infrastructure qui va être modifiée au fil du temps (ajouts, suppression, modifications).

✓ Permet de répondre rapidement aux changements de votre environnement en faisant des adaptations ad-hoc.

✗ Ne permet pas de garantir l'état de l'infrastructure à un instant T.

Ex: Chef, Ansible, Puppet, Saltstack

Immutable

Infrastructure qui ne va pas être modifiée une fois qu'elle est mise en place.

✗ Nécessite de repasser par du code, du Git, du CI/CD, etc. avant de redéployer.

✓ Permet de garantir que votre infrastructure est toujours fiable et évolue de manière prévisible d'après sa définition dans le code.

Ex: Terraform

IaC declarative VS procédurale

Procédurale

- **Séquence d'instructions ordonnées** qui décrivent les étapes de configuration et déploiement d'une infrastructure.
- On suppose une connaissance des état initiaux et des états finaux de chaque instruction, du comportement de l'infrastructure et de la technologie sous-jacente.
- Nécessite plus de maintenance pour vérifier les changements... donc risque plus élevé lors de l'exécution.

Ex: Shellscripts, Chef

Déclarative

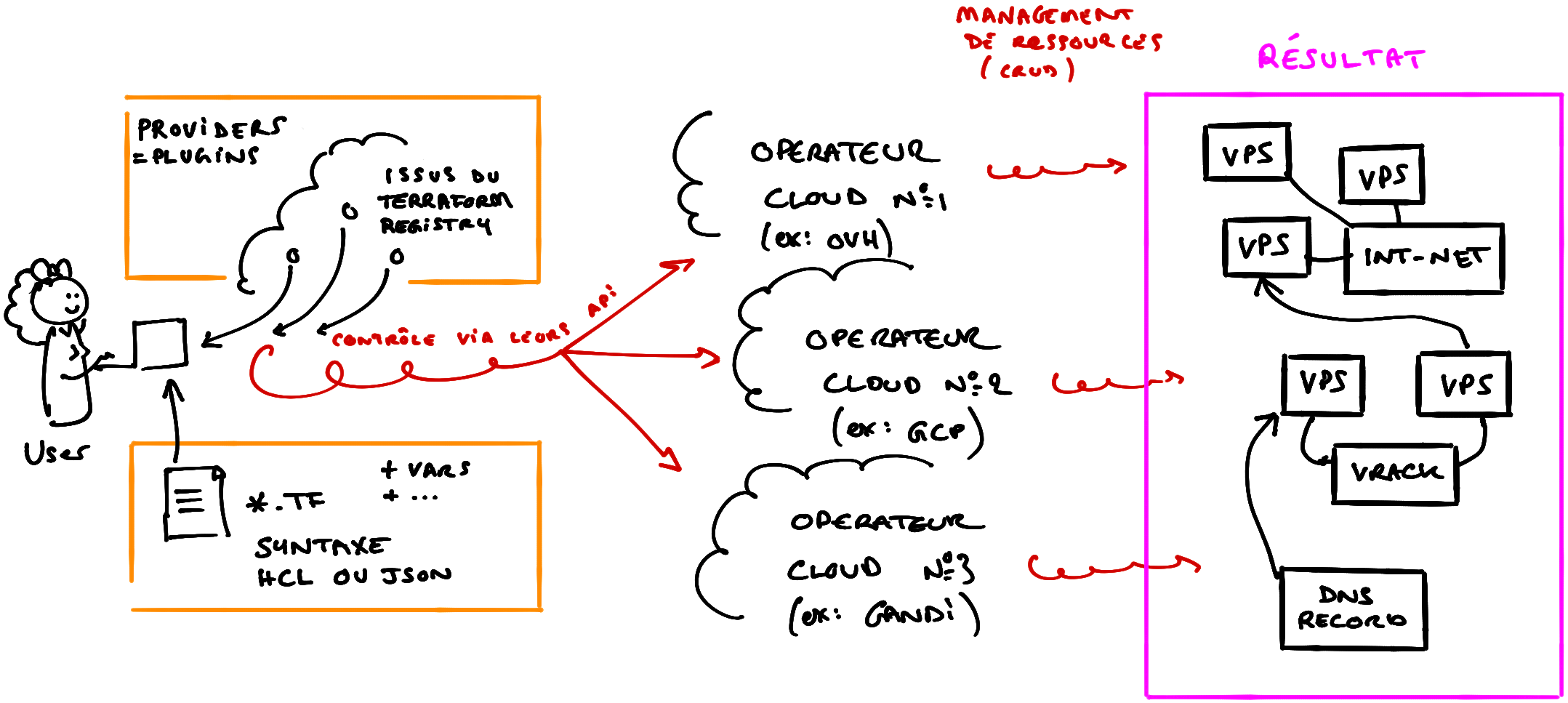
- **Spécifie un état souhaité** pour une infrastructure, sans préciser les étapes nécessaires pour y parvenir.
- Peut être exécutée sur une infrastructure existante, même si son état initial est inconnu.
  - L'outil commencera par faire un état des lieux (ex: `gathering facts...`)
- Peut être exécutée plusieurs fois sans créer d'effets secondaires indésirables (**idempotence**)

Ex: Puppet, Terraform, Saltstack

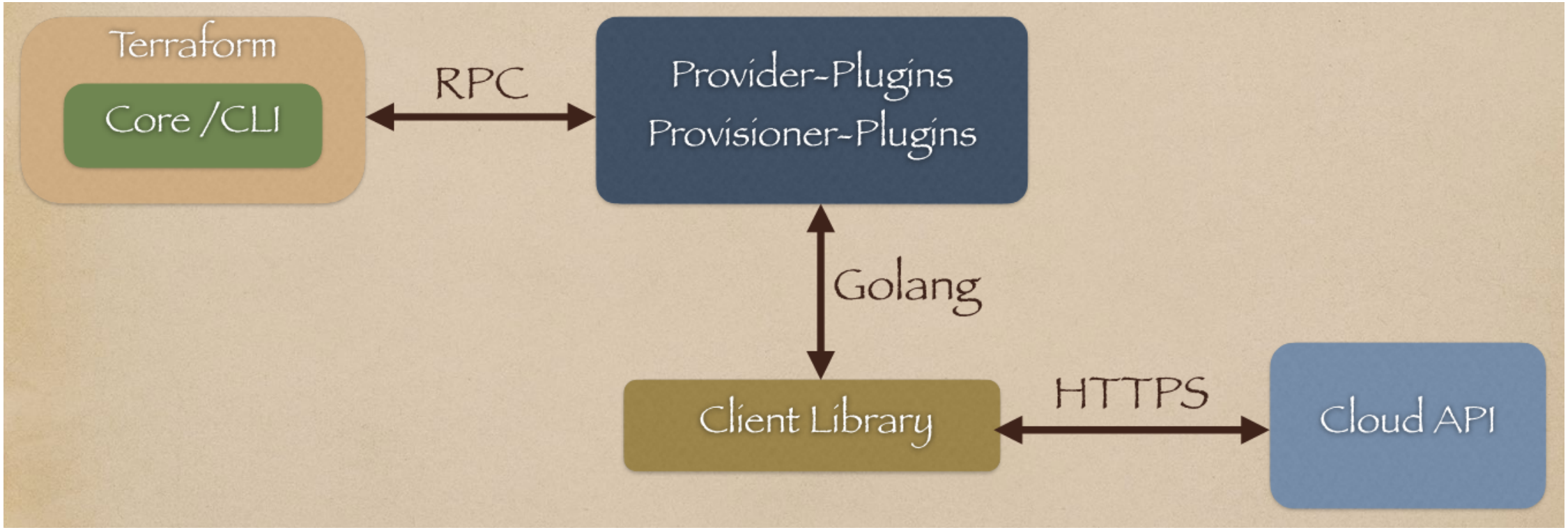
Note: Ansible est un peu entre les deux.

Architecture

Architecture de Terraform



Vue d'ensemble



Principes de fonctionnement

Démarche usuelle (simplifiée)

- Write
  - Décrire l'infrastructure comme du Code (IaC)
- Plan
  - Prévisualiser les changements avant de les appliquer
- Apply
  - Créer l'infrastructure d'après le code... de façon reproductible

Processus de travail (incluant Terraform)

1. Sur l'opérateur Cloud

- Créer des utilisateurs
- Gérer les groupes de sécurité

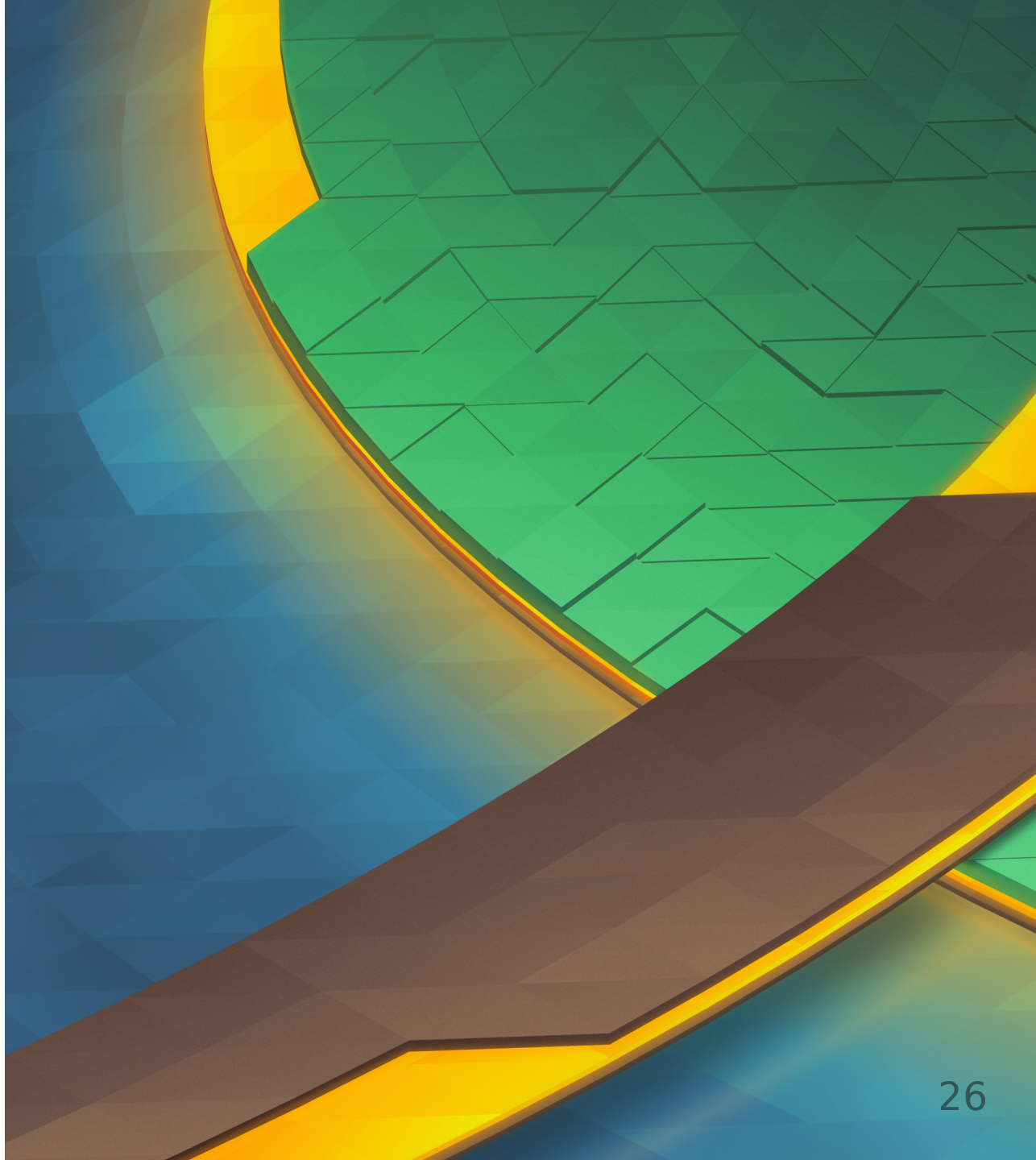
2. Sur le poste de travail

- Configurer les clés SSH
- Configuration de l'environnement et les variables de configuration
- Écrire le code Terraform
- Plannifier, comparer,
- Appliquer les changements

3. Dans Terraform

- Créer des ressources à distance (VPS, etc.).
- Exécuter les scripts de provisioning après la création des ressources.
- Obtenir les informations de sortie (adresses IP)

Installation



Installation sous Linux

Option 1 : Avec votre distribution préférée

- Installation du package avec `apt-get`, `yum`, `dnf`, etc.
- Vérification de l'installation

Option 2 : Avec ASDF

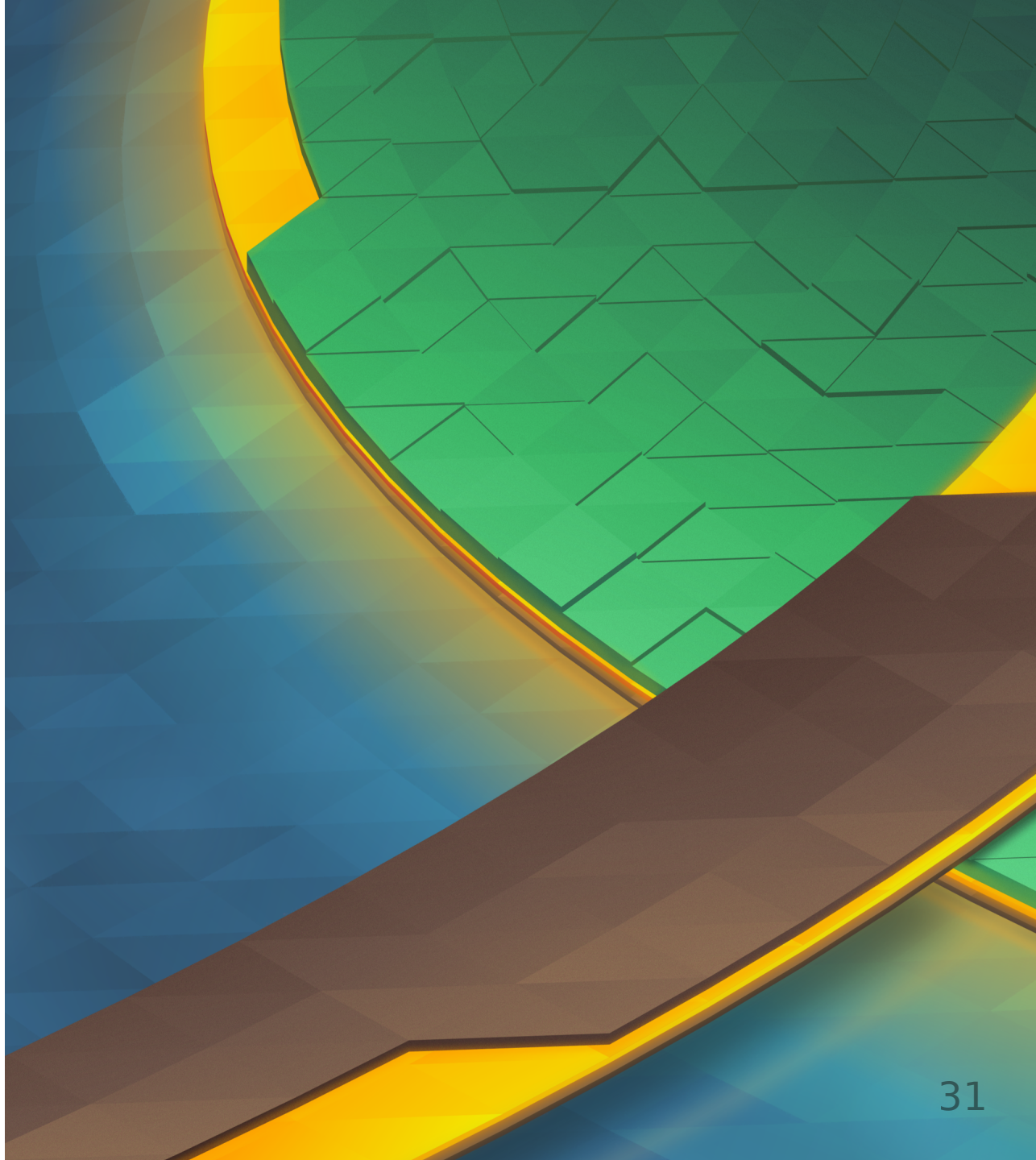
Voir: https://asdf-vm.com/

```
asdf plugin-add terraform
asdf install terraform latest
asdf global terraform latest
```

Option 3 : D'après le binaire

- Téléchargement du binaire
- Execution
- C'est tout

Utilisation de la ligne de commandes



Bases

Gestion du Plan

```
$ terraform
[...]
apply       Builds or changes infrastructure
[...]
destroy     Destroy Terraform-managed infrastructure
[...]
plan        Generate and show an execution plan
[...]
show        Show the current state or a saved plan
[...]
```

Plan

La commande *plan* sert à générer un plan avant de déployer notre infrastructure.

```
$ terraform plan
```

Show

La commande "show" sert à afficher des informations sur un état ou un plan Terraform sous forme lisible.

```
$ terraform show
```

Apply

- **Sert à appliquer les modifications à la cible**.
- Terraform utilise pour cela un plan (des modifications), qui est un fichier texte contenant les modifications à apporter à la cible.
  - Si on ne spécifie pas de plan, Terraform génère un plan en mémoire.
  - Si on le spécifie, Terraform récupère les modifications à apporter à la cible et les applique.

```
$ terraform apply
$ terraform apply -auto-approve
$ terraform apply -plan=plan.out
```

Destroy

La commande "destroy" sert à détruire l'infrastructure.

```
$ terraform destroy
```

Gestion de l'État

```
$ terraform
[...]
init        Initialize a Terraform working directory
output      Read an output from a state file
[...]
refresh     Update local state file against real resources
[...]
```

Init

- **Prépare l'environnement terraform** pour un module spécifique.
- **Initialise les différents composants** qui seront nécessaires au bon fonctionnement du module
  - (ex: providers utilisés, les variables, le backend, etc.)

Note: La commande `terraform init` ne peut être utilisé qu'une seule fois par module. Si vous avez besoin de le réinitialiser, vous devez d'abord le détruire avec la commande `terraform destroy`.

Output

- **Extrait les variables d'état Terraform et les affiche.**
  - Les variables d'état sont stockées dans le fichier `terraform.tfstate`
- Les utilisateurs peuvent spécifier le format de sortie souhaité (JSON, CSV, etc.).

```
$ terraform output
id = i-12345678
public_ip = 1.2.3.4
```

Refresh

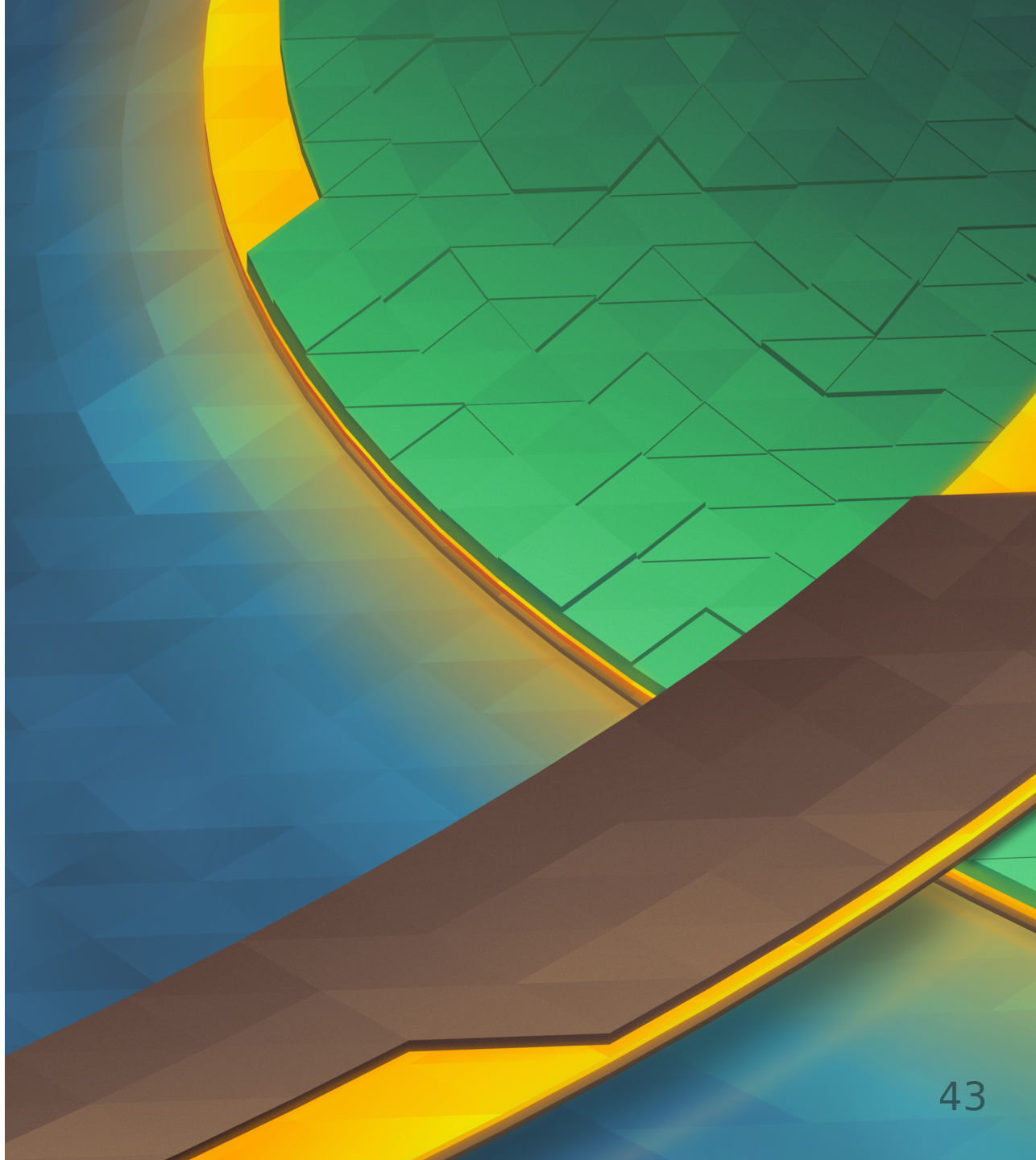
- Permet de rafraîchir l'état local de Terraform. Cet état est enregistré dans un fichier JSON (ex: `terraform.tfstate`) et correspond à la dernière exécution de la commande `terraform apply`.
- Donne une vision à jour de l'état du Cloud Provider, par exemple AWS.

Note: Cette commande génère PAS de différence entre l'état local et l'état distant (voir la commande `terraform plan`).

Et aussi...

```
$ terraform
[...]
fmt           Reformat your configuration in the standard style
[...]
graph         Generate a Graphviz graph of the steps in an operation
[...]
```

Première infrastructure



3 étapes

- Plan
- Apply
- Destroy

Un dossier par projet Cloud

```
$ mkdir terraform-example
$ cd terraform-example
$ touch main.tf
```

A propos des fichiers *.tf

- Le nom des fichiers `*.tf` n'a aucune importance.
  - Noms usuels : `main.tf`, `variables.tf`, `outputs.tf`
  - Ou sinon : `haproxy.tf`, `php.tf`, `banana.tf`,
- Terraform prend en compte tous les fichiers pour calculer l'état final à atteindre.

Cloud public Amazon Web Services

`main.tf` :

```
terraform {
  required_version = ">= 1.2.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
    # ...
  }
}
```

Cloud public Microsoft Azure

`main.tf` :

```
terraform {
  required_version = ">= 1.2.0"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.56.0"
    }

    # ...
  }
}
```

Cloud privé OpenStack

`main.tf` :

```
terraform {
  required_version = ">= 1.2.0"

  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.42.0"
    }

    # ...
  }
}
```

Cloud public Amazon Web Services

```
$ export AWS_ACCESS_KEY_ID="anaccesskey"
$ export AWS_SECRET_ACCESS_KEY="asecretkey"
$ export AWS_REGION="us-west-2"
```

Cloud public Microsoft Azure

Option 1 : Via la CLI

```
$ az login # ...
$ az account set \
      --subscription "xxx-subscription-id"
```

Option 2 : Service Principal

```
$ az ad sp create-for-rbac \
    --role="Contributor" \
    --scopes="/subscriptions/<SUBSCRIPTION_ID>"
```

```
$ export ARM_CLIENT_ID="<APPID_VALUE>"
$ export ARM_CLIENT_SECRET="<PASSWORD_VALUE>"
$ export ARM_SUBSCRIPTION_ID="<SUBSCRIPTION_ID>"
$ export ARM_TENANT_ID="<TENANT_VALUE>"
```

Cloud privé OpenStack

```
$ export OS_AUTH_URL="https://..."
$ export OS_IDENTITY_API_VERSION=3
$ export OS_USER_DOMAIN_NAME=${OS_USER_DOMAIN_NAME:-"Default"}
$ export OS_PROJECT_DOMAIN_NAME=${OS_PROJECT_DOMAIN_NAME:-"Default"}
$ export OS_TENANT_ID="..."
$ export OS_TENANT_NAME="..."
$ export OS_USERNAME="..."
$ export OS_PASSWORD=$OS_PASSWORD_INPUT
$ export OS_REGION_NAME="GRA"
```

```
$ . ./openrc.sh
```

Configurer un fournisseur (provider)

AWS

```
provider "aws" {
  # region = "us-west-2"
  # access_key = "my-access-key"
  # secret_key = "my-secret-key"
  # ...
}
```

OpenStack (ex: chez OVH)

```
provider "openstack" {
  auth_url    = "https://auth.cloud.ovh.net/v3/"
  domain_name = "default"
  alias       = "ovh"
  region = "GRA5"
}
```

Configurer un fournisseur (provider)

Azure

```
provider "azurerm" {
  features {}
  # ...
}
```

Définir une instance

AWS

```
resource "aws_instance" "web" {
  ami = "ami-70728c08"
  instance_type = "t2.micro"

  tags {
    name = "HelloWorld"
  }
}
```

Définir une instance

OpenStack

```
resource "openstack_compute_instance_v2" "web" {
  image_name = "Debian 11"
  flavor_name = "s1-2"

  metadata = {
    name = "HelloWorld"
  }
}
```

Initialiser le projet

```
$ terraform init
Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "aws" (1.0.0)...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 1.0"

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.
```

Voir le plan d'execution

```
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_instance.web
id: <computed>
ami: "ami-70728c08"
associate_public_ip_address: <computed>
source_dest_check: "true"
subnet_id: <computed>
tags.%: "1"
tags.Name: "HelloWorld"
Plan: 1 to add, 0 to change, 0 to destroy.
```

Appliquer le plan

```
$ terraform apply
aws_instance.web: Creating...
ami: "" => "ami-70728c08"
instance_type: "" => "t2.micro"
source_dest_check: "" => "true"
subnet_id: "" => "<computed>"
tags.%: "" => "1"
tags.Name: "" => "HelloWorld"
tenancy: "" => "<computed>"
volume_tags.%: "" => "<computed>"
vpc_security_group_ids.#: "" => "<computed>"
aws_instance.web: Still creating... (10s elapsed)
aws_instance.web: Still creating... (20s elapsed)
aws_instance.web: Creation complete after 22s (ID: i-0c0b4a732311b8bce)
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Vérifier les instances

Voir le plan d'execution (encore)

Pas de changement(s) prévu(s)

Etat du fichier `terraform.tfstate`

```
$ cat terraform.tfstate
```

Redimensionner l'infrastructure

Modifier le fichier des instances

```
 resource "aws_instance" "web" {
   ami = "ami-70728c08"
   instance_type = "t2.micro"

+  count = 3
   tags {
-    name = "Helloworld"
+    name = "Helloworld ${count.index}"
   }
 }
```

et refaire `terraform plan`

Ajouter un load balancer

```
resource "aws_elb" "lb" {
  name = "example-lb"
  availability_zones = ["us-west-2a"]
  instances = ["${aws_instance.web.*.id}"]
  listener {
    instance_port = "8000"
    instance_protocol = "http"
    lb_port = "80"
    lb_protocol = "http"
  }
}
```

Données en sortie (outputs) (1)

```
output "lb_dns" {
  value = "${aws_elb.lb.dns_name}"
}
```

Terraform refresh

```
$ terraform refresh
aws_instance.web[0]: Refreshing state... (ID: i-0c0b4a732311b8bce)
aws_instance.web[1]: Refreshing state... (ID: i-0b8bde17992d4dc0c)
aws_instance.web[2]: Refreshing state... (ID: i-06e74e8dc04eb1843)
aws_elb.lb: Refreshing state... (ID: example-elb)
Outputs:
lb_dns = example-elb-1640688516.us-west-2.elb.amazonaws.com
```

Données en sortie (outputs) (2)

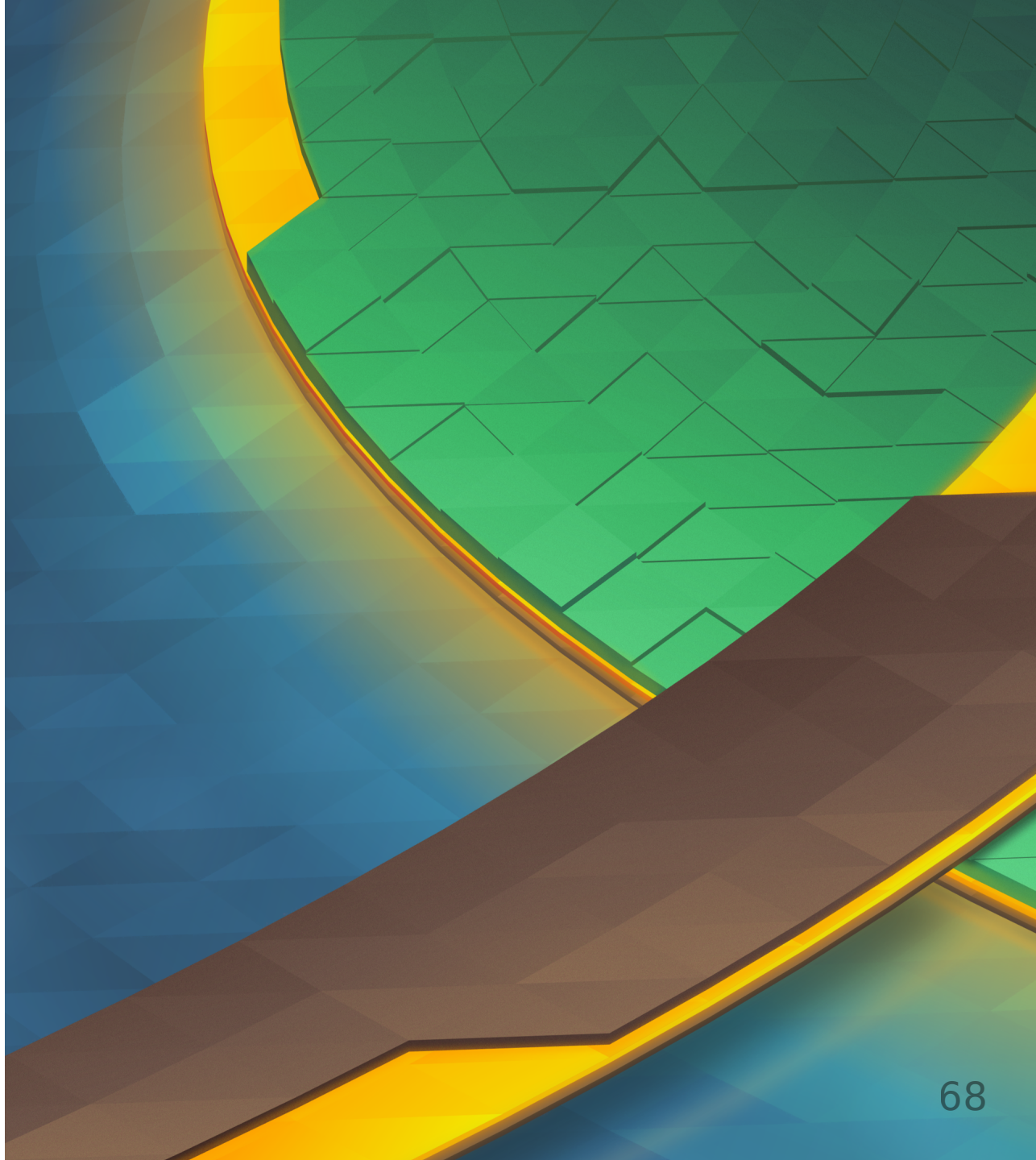
Terraform output -json

```
$ terraform output -json
{
  "lb_dns": {
  "sensitive": false,
  "type": "string",
  "value": "example-elb-1640688516.us-west-2.elb.amazonaws.com"
  }
}
```

Suppression de l'infrastructure

```
$ terraform destroy
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
- destroy
Terraform will perform the following actions:
- aws_elb.lb
- aws_instance.web[0]
- aws_instance.web[1]
- aws_instance.web[2]
Plan: 0 to add, 0 to change, 4 to destroy.
Do you really want to destroy?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.
Enter a value:
```

Le langage HCL



Présentation du HCL

Définition

"HCL": HashiCorp Configuration Language

Deux syntaxes

- Syntaxe HCL (ex: dans les fichiers `*.tf`)
- Syntaxe JSON (ex: dans les fichiers `*.tf.json`)

Syntaxe HCL

Argument

```
identifier = expression
```

Bloc

```
type [label] {
    block body [arguments / blocs]
}
```

Commentaire

```
# blabla         commentaire type shell
/* blabla */     commentaire type C */
// blabla        commentaire type C++
```

Syntaxe JSON (1)

Propriété JSON

```
"identifier": "expression"
```

Objet JSON

```
"identifier": {
    Object body
    [properties/objects/arrays]
}
```

Tableau JSON

```
"identifier": [
    Object body
    [properties/objects/arrays]
]
```

Commentaire

```
"//": "This ..."
```

Syntaxe HCL

```
terraform {
    required_version = ">= 1.0“
    required_providers {
        aws = {
            source = "hashicorp/aws"
            version = "~> 3.56.0"
        }
    }
}
```

```
provider "aws" {
    region = "eu-central-1"
        profile = "profile-name"
}
```

```
resource "aws_instance" "project-server" {
    ami = "ami-029c64b3c205e6cce"
    instance_type = "t4g.micro"
    tags = {
        Name = "Default Instance"
    }
}
```

Syntaxe JSON

```
{
    "terraform": {
        "required_version": ">= 1.0",
            "required_providers": {
                "aws": {
                    "source": "hashicorp/aws",
                    "version": "~> 3.56.0"
                }
            }
    },
    "provider": {
        "aws": {
            "region": "eu-central-1",
            "profile": "profile-name"
        }
    },
    "resource": {
        "aws_instance": {
            "project-server": {
                "ami": "ami-029c64b3c205e6cce",
                "instance_type": "t4g.micro",
                "tags": {
                    "Name": "Default Instance"
                }
            }
        }
    }
}
```

Providers

Role des providers

- Les providers sont des plugins qui permettent à Terraform de communiquer avec différents services Cloud.
- Ils sont responsables de la création, de la modification et de la suppression des ressources sur ces services.
- Terraform propose des providers pour de nombreux services, mais les utilisateurs peuvent également créer leurs propres providers.

Le *Terraform Registry*

- Le Terraform Registry est un service en ligne qui permet aux utilisateurs de partager et de découvrir des modules Terraform.
- Il permet aux utilisateurs de rechercher des modules selon différents critères, notamment le type de fournisseur, le type de resource et le type de module.
- Le Terraform Registry est hébergé par HashiCorp et est accessible à tous les utilisateurs de Terraform.

De nombreux providers : pour du cloud (1)

Amazon Web Services (AWS)

- Terraform Registry: AWS Provider

Google Cloud Platform (GCP)

- Terraform Registry: Google Cloud Platform Provider

OpenStack @ OVH

- Github: diodonfrost/terraform-openstack-examples sfdsf
- Comment utiliser Terraform sur le Public Cloud OVHcloud | Documentation OVH

Digital Ocean

- Terraform Registry: Digital Ocean Provider

De nombreux providers : pour des machines virtuelles (2)

Libvirt / KVM

- Terraform Registry: Libvirt Provider
- Github: dmacvicar/terraform-provider-libvirt
- Stephane Robert: Terraform Provider Libvirt
- Computing For Geeks: How to provision VMs on KVM with Terraform

De nombreux providers : pour des containers (3)

Linux Containers (LXC) + Proxmox / LXC

- Terraform Registry: Proxmod Provider
- Computing For Geeks: Create LXC containers using Terraform

Docker

- GNU/Linux Magazine FRance: Utilisez Terraform pour vos projets Docker

Ressources

Definition

```
resource "aws_instance" "project_server" {
    ami = "ami-029c64b3c205e6cce"
    instance_type = "t4g.micro"
}
```

Utilisation

Forme générale :

```
<Resource_type>.<Name>.<Attribute>
```

Par exemple :

```
foobar = aws_instance.project_server.somevalue * 3
```

Attention aux caractères utilisés dans le nom des variables : même s'ils sont acceptés par Terraform dans la définition d'une ressources certains caractères seront évalués bizarrement quand la variable sera utilisée. Par exemple le caractère `-` sera évalué comme l'opération algébrique « moins »...

Variables

Définition

```
variable "image_id" {
    type = string
    description = "The id of the machine image (AMI) to use for the server."
    default = "ami-029c64b3c205e6cce"
    validation {
        condition = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"
        error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."
    }
}
```

Utilisation

Forme générale

```
var.<name>
```

Exemple

```
foobar = var.image_id
```

Hierarchie d'évaluation

1. Variables d'environnement `TF_VAR_...`

```
$ export TF_VAR_image_id=ami-abc123
```

2. Fichiers `terraform.tfvars` et `terraform.tfvars.json`

```
region = "us-east-2"
project = "workshop"
stage = "testing"
image_id = "ami-029c64b3c205e6cce"
```

3. Fichiers `*.auto.tfvars` et `*.auto.tfvars.json`
4. Paramètres `-var` et `-var-file` de la ligne de commande Terraform:

```
$ terraform apply -var="image_id=ami-abc123"
$ terraform apply -var-file="testing.tfvars"
```

Interpolation

```
resource "aws_instance" "web" {
    count = "3"
    ami = "ami-70728c08"
    instance_type = "t2.micro"
    tags {
        Name = "web-${count.index}"  # <==
    }
}
```

External variables

Dans `variables.tf`

```
variable "environment" {
    type = string
}
```

Dans `main.tf`

```
resource "aws_instance" "web" {
    count = "3"
    ami = "ami-70728c08"
    instance_type = "t2.micro"
    tags {
        Name = "web-${count.index}"
        Environment = "${var.environment}"
    }
}
```

Déclaration des variables

Depuis l'environnement

```
$ export TF_VAR_ENVIRONMENT=production
```

Ou depuis la CLI

```
terraform -var environment=production
```

Ou depuis un fichier

```
terraform -var-file my.tfvars
```

Valeurs de sortie (output)

Fonctionnement (1)

Définition

```
output "ec2_instance_public_ip" {
  value = aws_instance.project_server.public_ip
}
```

Utilisation

Format général

```
module.<MODULE NAME>.<OUTPUT NAME>
```

Exemple

```
foobar = module.my_module.ec2_instance_public_ip
```

Resultat

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

hostname = terraform.example.com
private_ip = 10.5.4.82
public_ip = 94.237.45.221
```

Valeurs locales

Définitions

```
locals {
   /*
    * RDS (database)
    */
   rds_instance_allocated_storage = var.stage == "dev" ? 5 : 10
   rds_instance_class = var.stage == "dev" ? "db.t3.micro" : "db.t2.large"
   rds_database_name = var.stage == "dev" ? "projectdevdb" : "projectproddb"
   rds_database_user_name = "dbuser"
   rds_database_backup_retention_period = 14
   rds_database_deletion_protection = var.stage == "dev" ? false : true
}
```

Utilisation

Format général

```
local.<Name>
```

Exemple

```
name = local.rds_database_name
```

Expressions et fonctions

Expressions

Conversion de type

```
"true" converts to true / "5" converts to 5
```

Interpolation de chaîne

```
"Hello, ${var.name}!"
```

Map

```
[for o in var.list : o.id]
```

Splat

```
var.list[*].id
```

```
var.list.*.id
```

Expression conditionnelle

```
var.dbname != "" ? var.dbname : "defaultname"
```

Fonctions

Concaténation d'éléments de tableau

```
resource_prefix = join("-", [var.project, var.stage])
```

Sous-chaine

```
substr("hello world", 1, 4)
```

Concaténation de tableaux

```
concat(["a", ""], ["b", "c"])
```

Présence d'un élément

```
contains(["a", "b", "c"], "a")
```

Base64

```
base64decode("SGVsbG8gV29ybGQ=")
```

Existance d'un fichier

```
fileexists("${path.module}/hello.txt")
```

Boucles

Nécessitent une liste

```
variable "server_names" {
    description = "Create server with these names"
    type = list(string)
    default = ["neo", "trinity", "morpheus"]
}
```

Utilisation au sein d'une ressourcce (loop + count)

```
resource "aws_instance" "server" {
    ami = data.aws_ami.amazon_linux_2_arm64.image_id
    instance_type = "t2.micro"
    count = length (var.server_names)
    tags = {
        name = var.server_names[count.index]
    }
}
```

Utilisation avec `for_each`

```
resource "aws_instance" "server" {
    ami = data.aws_ami.amazon_linux_2_arm64.image_id
    instance_type = "t2.micro"
    for_each = toset(var.server_names)
    tags = {
        name = each.value
    }
}
```

Mapping de valeurs

```
output "upper_server_names" {
    value = [for name in tolist(var.server_names) : upper(name)]
}
```

Attention: `for_each` produit un *set*, pas une liste !

Provisioning software

En exécutant une commande en local

```
resource "aws_instance" "web" {
    count = "3"
    ami = "ami-70728c08"
    instance_type = "t2.micro"
    tags {
        ...snip...
    }
    provisioner "local-exec" {
        command = <<-MARK
            ansible-playbook -u ubuntu \
                --private-key ./aws-key.pem \
                -i '${self.public_ip},' \
                playbook.yml
        MARK
    }
}
```

En exécutant une commande à distance

```
resource "aws_instance" "web" {
    ...snip...

    provisioner "remote-exec" {
        connection {
            type = "ssh"
            user = "root"
            password = "${var.root_password}"
        }
        inline = [
            "sudo apt-get update -q",
            "sudo apt-get install -y -q nginx",
            "sudo systemctl restart nginx"
        ]
    }
}
```

Configurer la connexion

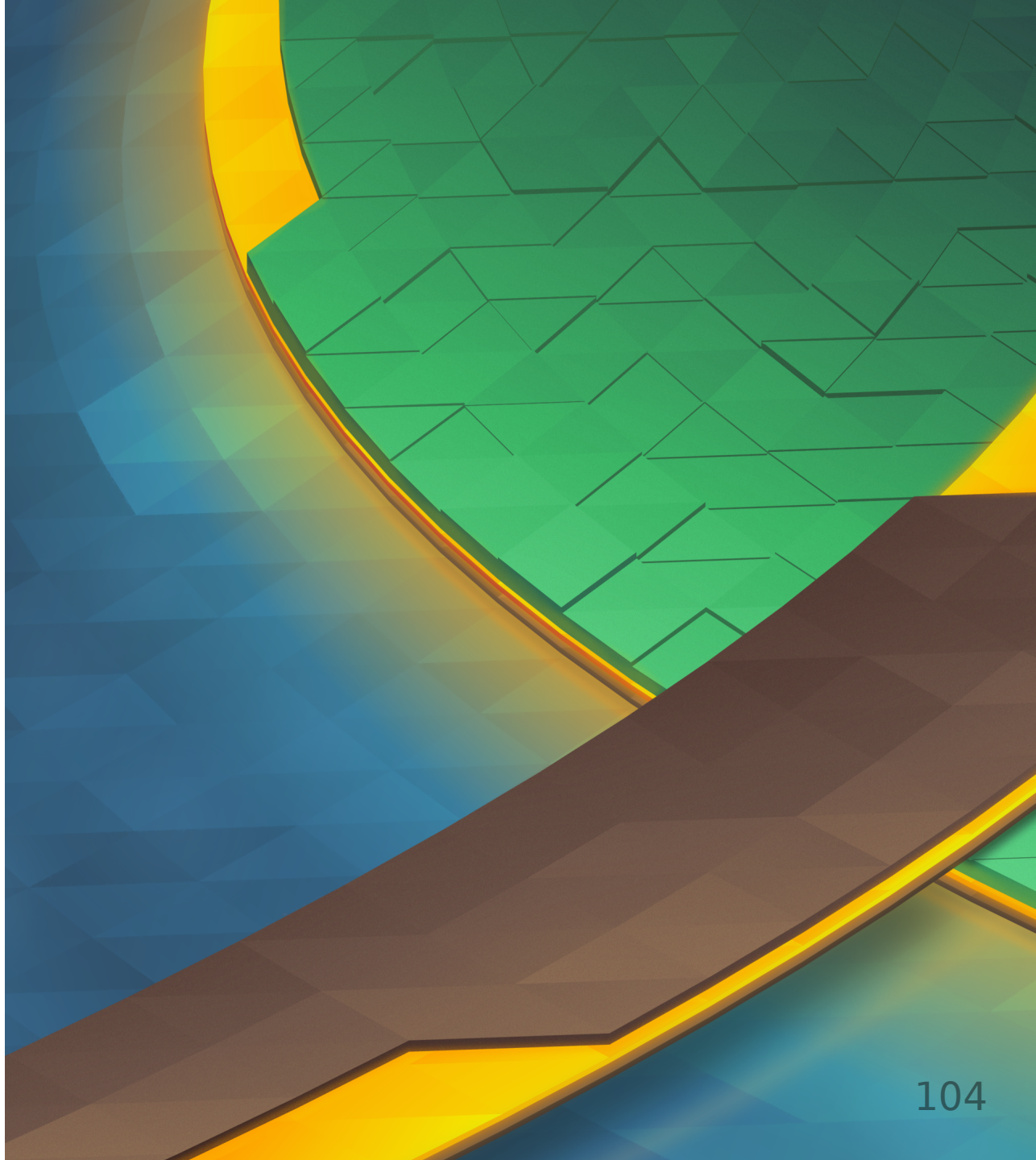
Avec agent SSH:

```
connection  {
    type = "ssh"
    agent = true
    host = "..."

    # private_key = ...
}
```

!!! note Si la clef protégée par un password, alors il vaut mieux utiliser l'agent SSH à la place

Modules



Utilisation des modules

Création d'un module

```
$ tree
.
|-- main.tf
|-- output.tf
`-- variables.tf
```

Utilisation d'un module (local)

```
module "pirate" {
  # le 'include'
  source = "./demo"

  # les variables d'entrée du module
  environment = "Pirate"
}

module "ninja" {
  # le 'include'
  source = "./demo"

  # les variables d'entrée du module
  environment = "Ninja"
}
```

Utilisation d'un module (registry)

cf: https://registry.terraform.io/

```
module "pirate" {
    source = "mheap/aws/full-env"
    environment = "Pirate"
}
module "ninja" {
    source = "mheap/aws/full-env"
    environment = "Ninja"
}
```

Utilisation d'un module (git)

```
module "pirate" {
    source = "git::https://hashicorp.com/consul.git?
    ref=1.0.3"
    environment = "Pirate"
}
module "ninja" {
    source = “git::https://hashicorp.com/consul.git?
    ref=1.8.14"
    environment = "Ninja"
}
```

Mais aussi...

- github
- bitbucket
- mercurial
- s3
- http
- etc.

Sorties des modules

```
output "pirate_lb" {
    value = module.pirate.lb_dns
}
output "ninja_lb" {
    value = module.ninja.lb_dns
}
```

Arborescence de travail

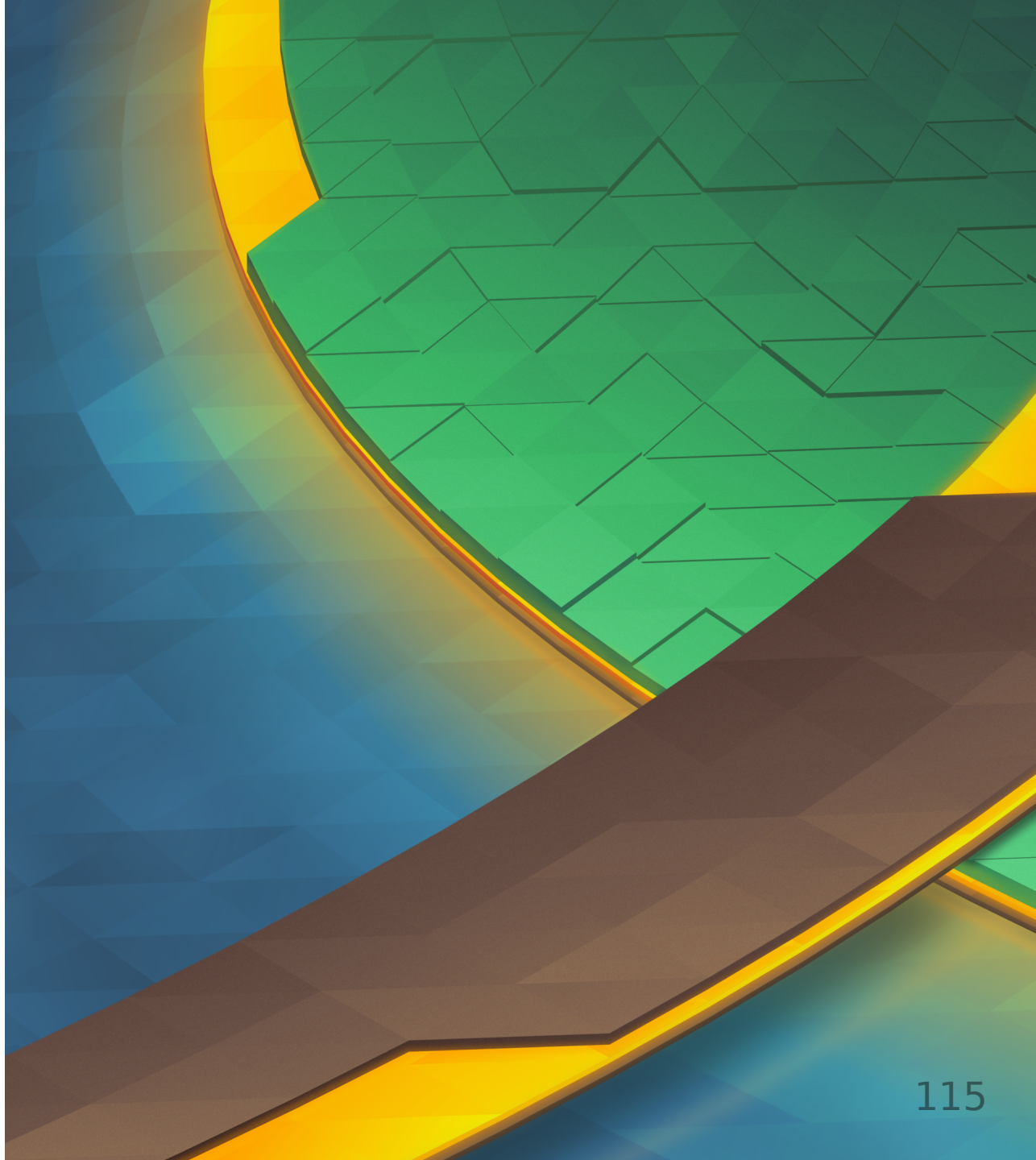
Pas bien ❌

```
.
|-- demo
| !-- main.tf
| !-- output.tf
| `-- variables.tf
|-- main.tf
|-- outputs.tf
`-- terraform.tfstate
```

Bien ✓

```
.
|-- environments
| |-- ninja
| | |-- main.tf
| | |-- output.tf
| | `-- terraform.tfstate
| `-- pirate
|   |-- main.tf
|   |-- output.tf
|   `-- terraform.tfstate
`-- modules
  `-- demo
    |-- main.tf
    |-- output.tf
    `-- variables.tf
```

Collaboration



Partage de l'état

3 choix

- Amazon S3
- Consul
- Terraform Entreprise

Amazon S3

```
terraform {
    backend "s3" {
        bucket = "mybucket"
            key = "path/to/my/key"
            region = “us-west-2"
    }
}
```

Consul

```
terraform {
    backend "consul" {
        address = "demo.consul.io"
            path = "example_app/
            terraform_state"
    }
}
```

Références

- Terraform Documentation: S3 Backend
- https://developer.hashicorp.com/terraform/language/settings/backends/s3

Workspaces

Usage des workspace

- Les Workspace Terraform servent à gérer les différents environnements.
- Chaque Workspace correspond à un environnement.

Création d'un nouveau Workspace

Pour créer un nouveau Workspace, il faut utiliser la commande `terraform workspace new [WORKSPACE]`.

Sélection d'un Workspace

Pour sélectionner un Workspace, il faut utiliser la commande `terraform workspace select [WORKSPACE]`.

Suppression d'un Workspace

Pour supprimer un Workspace, il faut utiliser la commande `terraform workspace delete [WORKSPACE]`.

Liste des Workspaces

Pour lister les Workspaces, il faut utiliser la commande `terraform workspace list`.

Workspace par défaut

Le Workspace par défaut est le Workspace `default`.

Références

- https://k21academy.com/terraform-iac/hashicorp-infrastructure-automation-certification-terraform-associate/
- https://learn.hashicorp.com/terraform?utm_source=terraform_io
- https://learn.hashicorp.com/collections/terraform/aws-get-started
- https://www.terraform.io/docs/language/state/workspaces.html

Bonnes pratiques pour la collaboration

Surcharger les ressources

- Utiliser le fichier `override.tf` (en developpement, ou dans des tests sur CI/CD)
- Permet d'écraser (fusionner) les valeurs définies dans les fichiers `*.tf`
- ... plutôt que plutôt que d'ajouter une nouvelle ressource

Fichier `instance.tf`

```
resource "aws_instance" "web" {
  ami = "ami-70728c08"
  instance_type = "t2.micro"

  tags {
    name = "Helloworld"
  }
}
```

Fichier `override.tf`

```
resource "aws_instance" "web" {
   count = 3
}
```

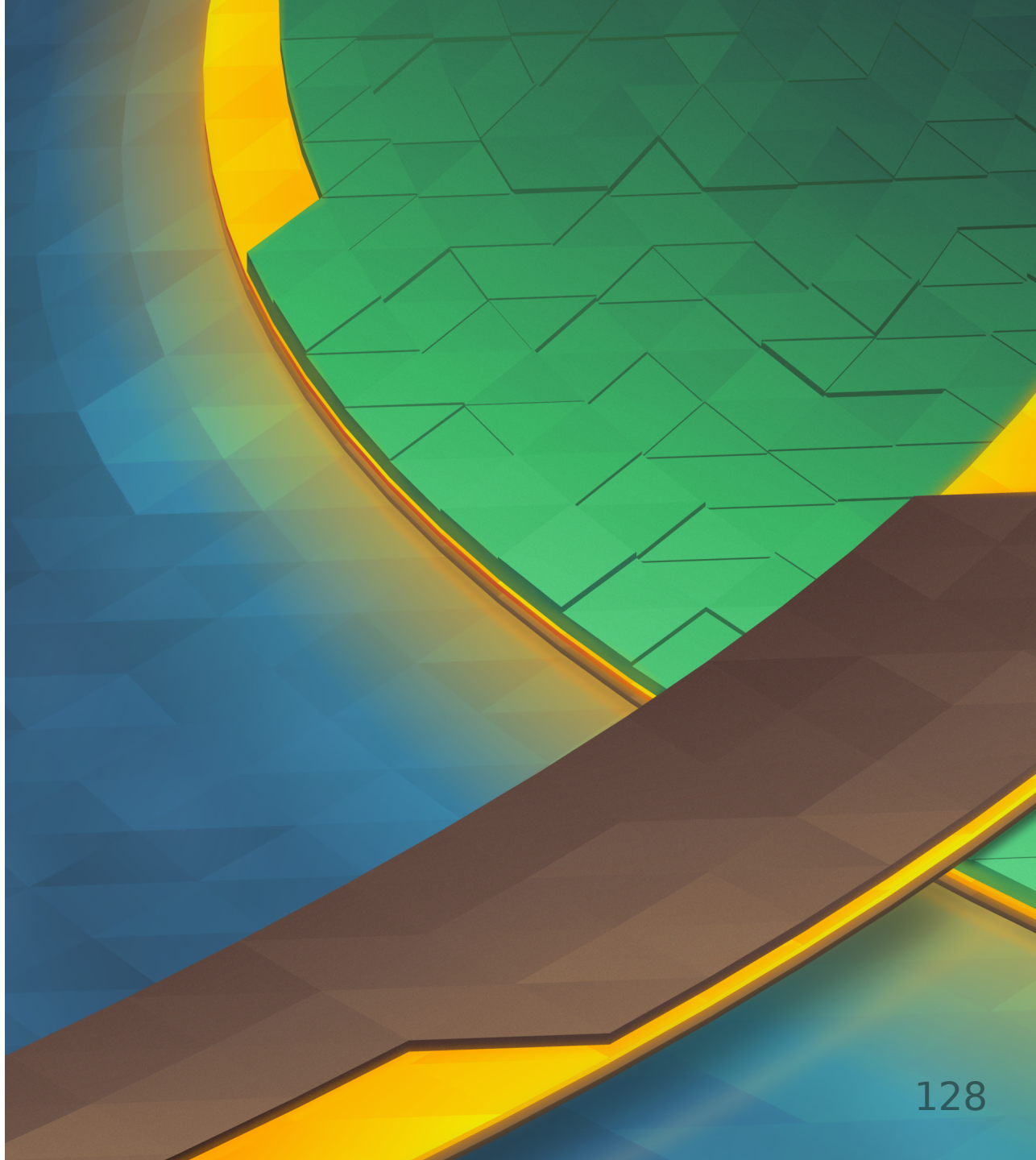
📘 Terraform Documentation: Override

Terraform configuration block

```
terraform {
    required_version = "> 0.8.0"
}
```

- Pour assurer qu'aucun collègue ne va "changer" le format du `terraform.tfstate` par mégarde.
- Important quand vous avez un état géré à distance et qu'il faut garantir la compatibilité.

Utilisation en production



Gestion du cycle de vie

Empêcher la destruction

```
lifecycle {
    prevent_destroy = true
}
```

Créer avant de détruire

```
lifecycle {
    create_before_destroy = true
}
```

Ignorer les changements

```
lifecycle {
    ignore_changes = true
}
```

Préconditions

```
lifecycle {
    # The AMI ID must refer to an AMI that contains an operating system
    # for the `x86_64` architecture.
    precondition {
        condition     = data.aws_ami.example.architecture == "x86_64"
        error_message = "The selected AMI must be for the x86_64 architecture."
    }
}
```

Références

- Terraform Documentation: Lifecycle

Sécurité

Quelques règles fondamentales

Dans votre projet Terraform

- Ne stockez pas vos secrets dans le code, ni dans le dépôt Git.
- Utilisez des modules Terraform pour organiser le code.
- Utilisez des variables Terraform pour contrôler et restreindre les configurations de vos ressources.
- Limitez l'accès aux fichiers Terraform aux utilisateurs autorisés.

Sur vos opérateurs Cloud

- Configurez les *contrôles d'accès* et les *autorisations* les plus restrictives possible sur vos ressources cloud.
- Utilisez des *régions* et des *zones* séparées pour vos ressources.
- Créez des plans de sauvegarde pour vos ressources cloud.
- Utilisez des *politiques de sécurité* et de *stratégies de confidentialité* pour controler vos réseaux et services.

Sécuriser avec des outils supplémentaires

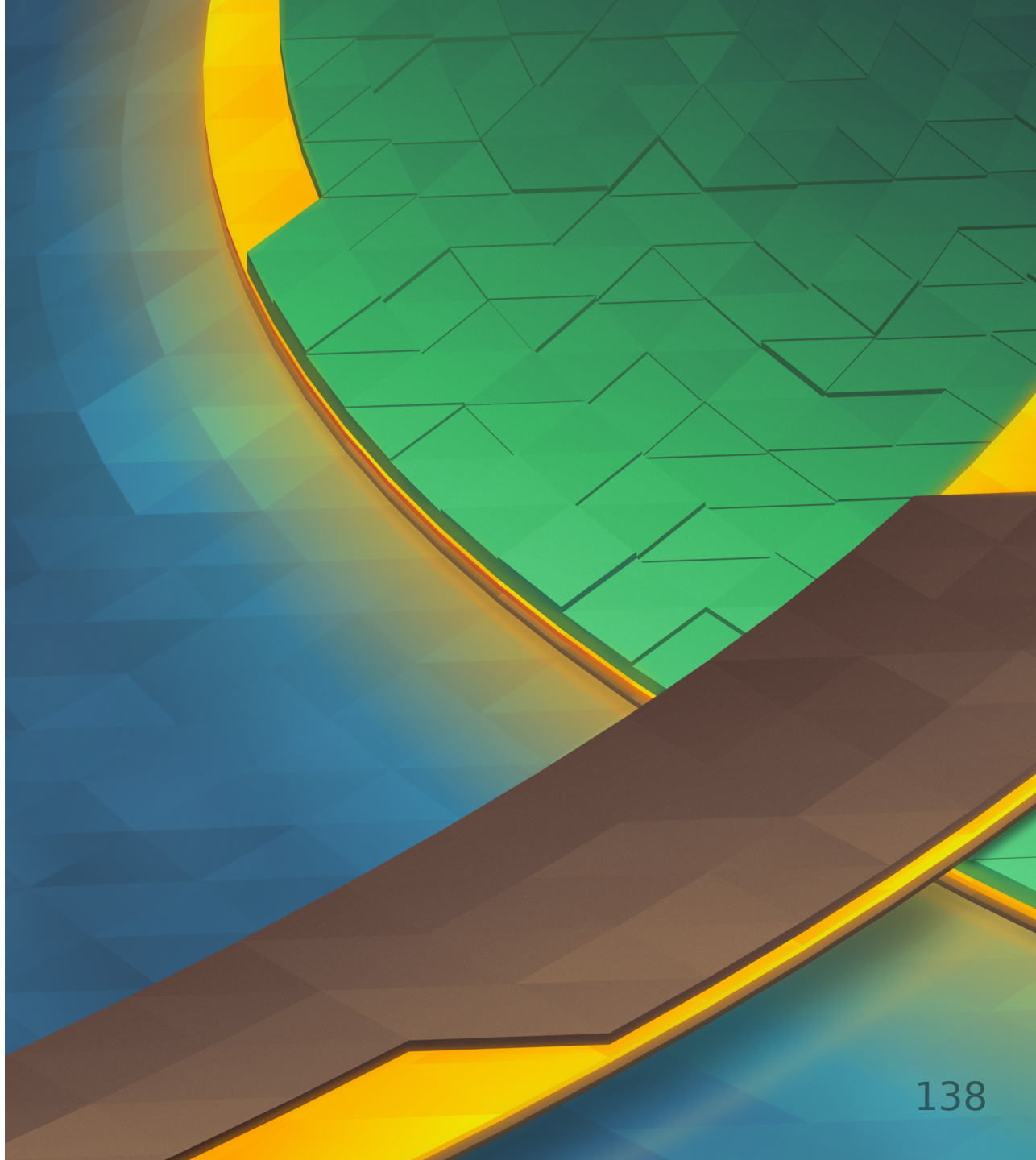
Objectifs

- Gérez vos clés d'accès et contrôles d'accès.
- Intégrez des tests de sécurité et de conformité continus dans votre processus de déploiement.
- Surveiller et auditer vos déploiements.

Outils

- https://github.com/aquasecurity/tfsec : Security scanner for your Terraform code
- https://github.com/tenable/terrascan : Terrascan is a static code analyzer for Infrastructure as Code

Bonus



Migration vers Terraform

Import d'une resource cloud existante

- Renseigne une ressource existante dans l'état Terraform.
  - ... lors de la conversion d'un projet existant vers Terraform,
  - ... quand une ressource est créé avec un outil non Terraform et doit etre ensuite gérée par Terraform
- Pour importer une ressource, il faut connaitre
  - son type
  - son nom d'identification dans le provider,
  - le chemin d'accès complet vers le fichier d'état

```
$ terraform import TYPE.RESOURCE ID
```

Question : Comment retrouver les ID de ressource chez les opérateurs?

Import d'une infrastructure complète

- Github: GoogleCloudPlatform/terraformer
- Github: cycloidio/terracognita

Utiliser plusieurs providers à la fois

Exemple: HEROKU + DNSIMPLE

```
provider "heroku" {
    email = "ops@company.com"
        api_key = "${var.heroku_api_key}"
}

provider "dnsimple" {
    token = "${var.dnsimple_token}"
        account = "${var.dnsimple_account}"
}

resource "heroku_app" "default" {
    ...
}
resource "dnsimple_record" "foobar" {
    domain = "${var.dnsimple_domain}"
        name = ""
        value = “${heroku_app.default.heroku_hostname}"
        type = "CNAME"
        ttl = 3600
}
```

Read-only data providers

ex: `aws_ami` pour trouver les AMI sur AWS

```
data "aws_ami" "ubuntu" {
    most_recent = true
        filter {
            name = "name"
                values = ["ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-*"]
        }
    filter {
        name = "virtualization-type"
            values = ["hvm"]
    }
    owners = ["099720109477"] # Canonical
}
```

Debugging

Contrôle de la verbosité

Activer les logs

```
$ export TF_LOG=DEBUG
```

- Niveaux possibles: `TRACE`, `DEBUG`, `INFO`, `WARN` ou `ERROR`

Stocker les logs dans un fichier

```
export TF_LOG_PATH="terraform.txt"
```

Activer le debug des providers

```
$ export TF_LOG_PROVIDER=DEBUG
```

- Attention c'est verbeux
- Cela ne montre essentiellement l'interaction entre Terraform et ses providers,
- Il peut manquer des détails sur le fonctionnement de ce que fait le provider

Activer le debug spécifique à certains providers

Pour openstack:

```
$ export OS_DEBUG=1
```

Références

- Terraform Documentation: Debugging
- SpaceLift: How to Debug & Troubleshoot Terraform Projects: Tutorial

Tester l'infrastructure produite

ServerSpec

```
# spec/ec2_spec.rb

require "spec_helper"

before do
    @client = Aws::EC2::Client.new
    @ec2 = Aws::EC2::Resource.new(client: @client)
end

describe "Instances" do
    subject { @ec2.instances.count }
    it { is_expected.to eq(3) }
end
```

```
$ rake spec
```

ServerSpec: Home

TestInfra

```
# test_nginx.py

def test_nginx_is_installed(host):
    nginx = host.package("nginx")
    assert nginx.is_installed

def test_nginx_running_and_enabled(host):
    nginx = host.service("nginx")
    assert nginx.is_running
    assert nginx.is_enabled
```

```
$ pytest test_nginx.py
```

📘 Testinfra: Test your infrastructure

Ansible + Assert module

```
---
- name: Test infrastructure with Ansible assert module.
  hosts: all
  tasks:
    - name: Check if port 80 is open
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 80
        timeout: 5
        state: started
      register: result

    - name: Assert port 80 is open
      assert:
        that:
          - result.state == "started"
        success_msg: "Port 80 is open."
        fail_msg: "Port 80 is not open on {{ inventory_hostname }}."
```

```
ansible-playbook --check -i inventory infra_testing.yaml
```

📘 Ansible Documentation: Assert Module

Terragrunt

Présentation de Terragrunt

- **Terragrunt** est un outil complémentaire à Terraform qui vise à réduire la répétition de code et à faciliter la gestion des configurations Terraform à grande échelle.
- Il s'appuie sur les fichiers de configuration Terraform existants pour ajouter des fonctionnalités supplémentaires et améliorer la gestion de l'infrastructure as code.
- **Gestion de la configuration à grande échelle :** Simplifier la gestion des configurations Terraform pour des projets complexes ou à grande échelle.
- **DRY (Don't Repeat Yourself) :** Minimiser la répétition du code Terraform à travers des environnements multiples.
- **Automatisation :** Automatiser les tâches répétitives liées à la gestion de l'infrastructure.

Avantages de Terragrunt

- **Gestion centralisée :** Permet la gestion centralisée des états Terraform et des configurations de backend.
- **Dépendances entre modules :** Gère facilement les dépendances entre les modules Terraform.
- **Réutilisabilité :** Facilite la réutilisabilité du code Terraform à travers différents environnements et configurations.
- **Configuration simplifiée :** Simplifie la configuration de l'infrastructure en réduisant le besoin de scripts externes ou de configurations répétitives.

Inconvénients de Terragrunt

- **Complexité supplémentaire :** Peut introduire une couche supplémentaire de complexité pour les projets simples.
- **Maintenance :** Nécessite de rester à jour avec les versions et les pratiques de Terraform et Terragrunt.

Interfaces graphiques

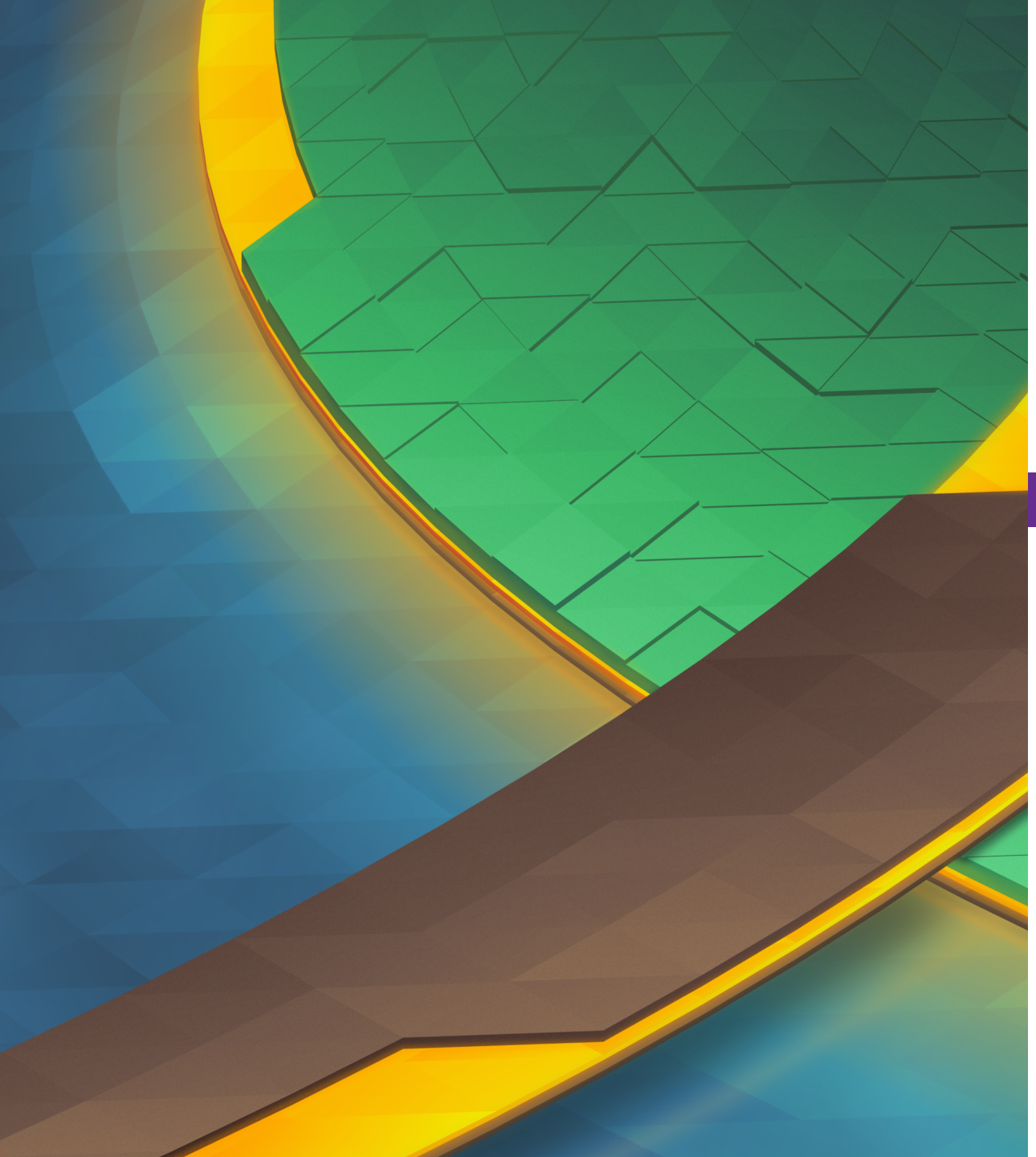
Terraboard

- https://terraboard.io/

InfraMap

- GitHub - cycloidio/inframap Read your tfstate or HCL to generate a graph specific for each provider, showing only the resources that are most important/relevant.

Terragraph



Merci pour votre attention !