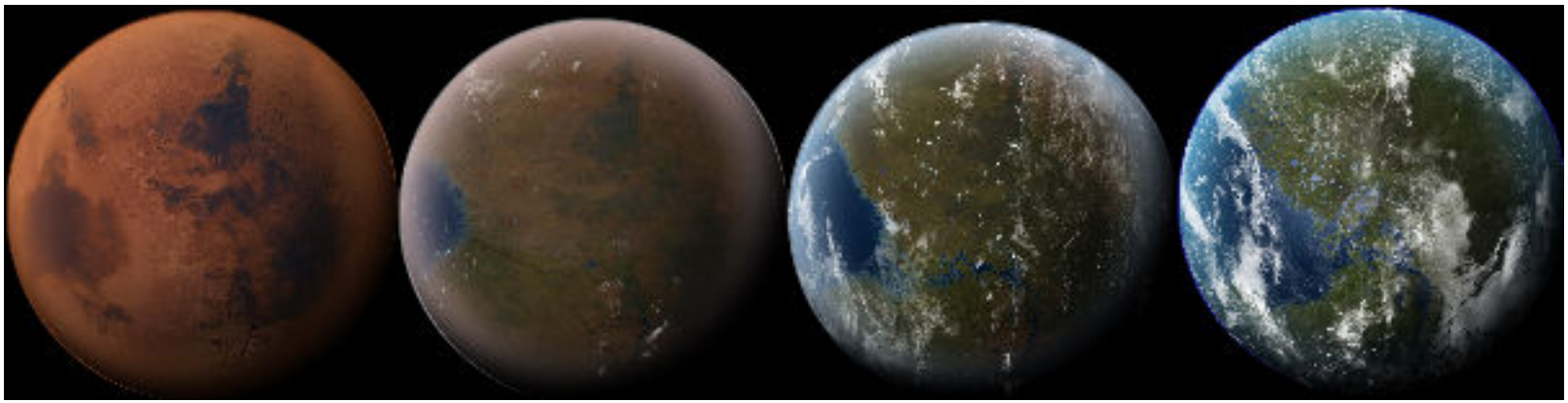
Terraform

Ein Einblick der Möglichkeiten von Infrastructure-as-Code

Dr. Sebastian Oehlke

27.04.2019

Terraforming - Begriffserklärung



https://en.wikipedia.org/wiki/Image:MarsTransitionV.jpg

Motivation

Wir brauchen jetzt n-Ressourcen, die folgende Anforderungen erfüllen …

Terraform - Kurzbeschreibung

- Infrastructure-as-code
- Hashicorp (Vagrant, Vault, Packer)

Terraform - Kurzbeschreibung

- Kodifiziert API-Befehle in deklaritive Konfigurationsdateien
- Open-Source

Terraform - Kurzbeschreibung

- Sicher und vorhersagbar
- Erstellen, Ändern, Verbessern der Infrastruktur
- Kombination von verschiedenen Providern
- Code teilen, Versionskontrolle, Review-Möglichkeit
- Modular
- Gut zu automatisieren

Terraform - Kurzbeschreibung

- Sicher und vorhersagbar
- Erstellen, Ändern, Verbessern der Infrastruktur
- Kombination von verschiedenen Providern
- Code teilen, Versionskontrolle, Review-Möglichkeit
- Modular
- Gut zu automatisieren

Auszug an Möglichkeiten mit Terraform

Provider-Gruppen:

- Verwaltung von Cloud-Diensten
- Verwaltung von Container-Plattformen
- Network Providern
- Version Control
- Monitoring & System Management
- Databases

⇒ Wenn es eine API hat, geht vermutlich das Management via Terraform

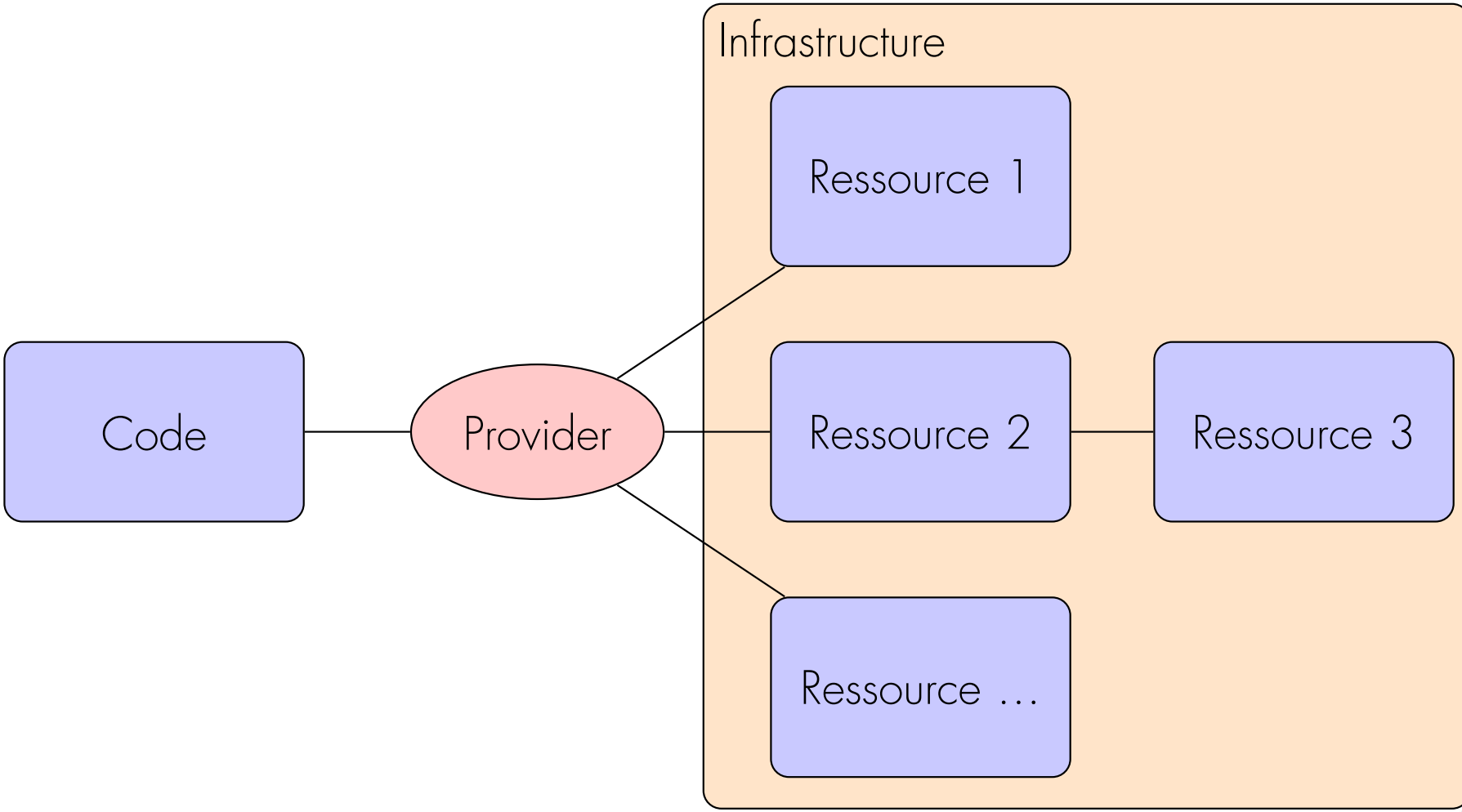
Auszug an Möglichkeiten mit Terraform

Provider-Gruppen:

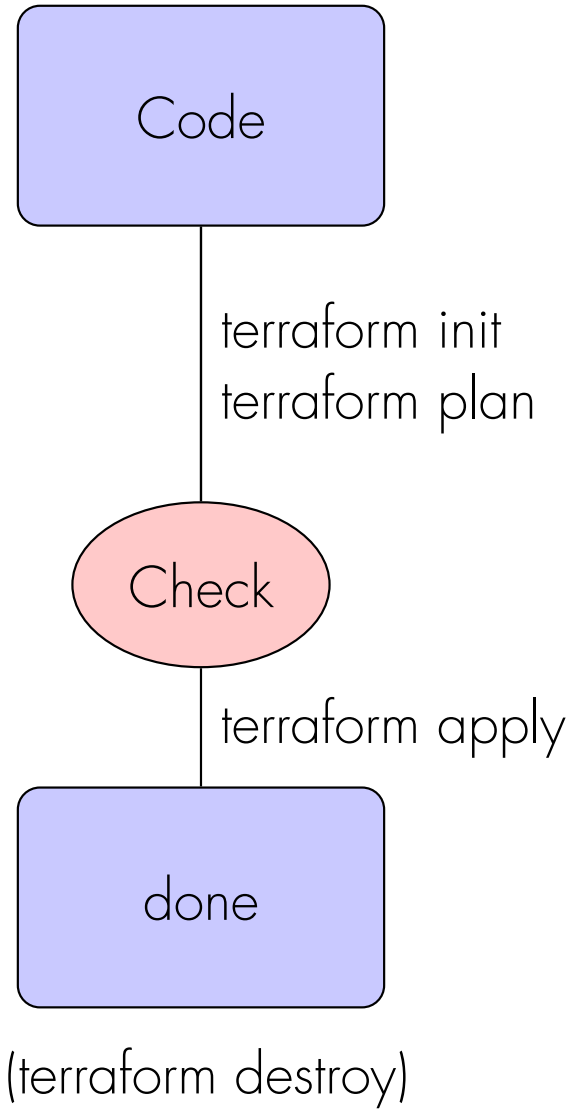
- Verwaltung von Cloud-Diensten
- Verwaltung von Container-Plattformen
- Network Providern
- Version Control
- Monitoring & System Management
- Databases

⇒ Wenn es eine API hat, geht vermutlich das Management via Terraform

Schema

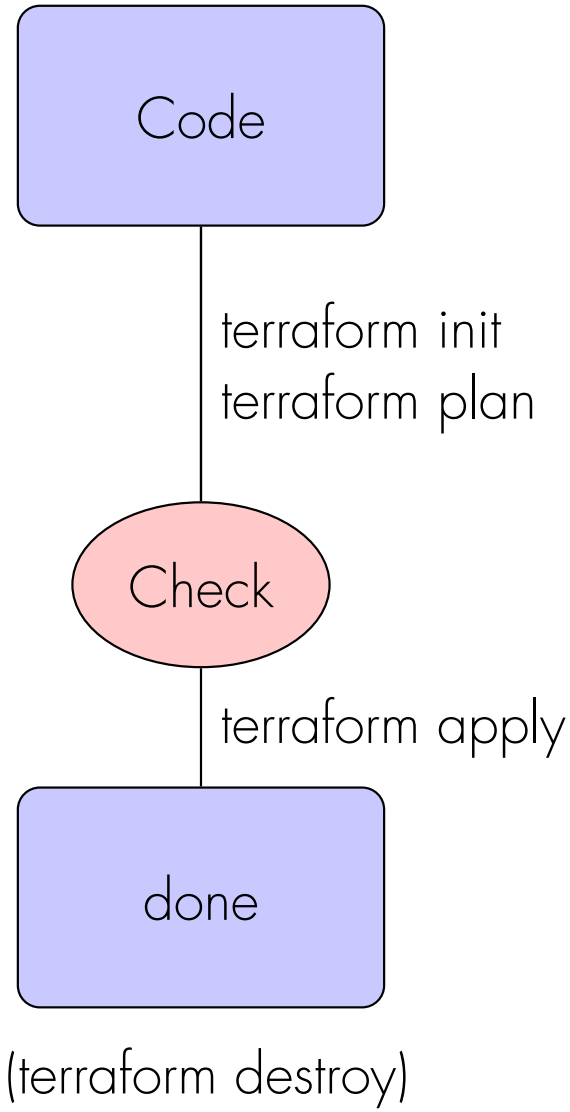

Infrastructure
Ressource 1
Code
Provider
Ressource 2
Ressource 3
Ressource …

Ablaufplan



- Check der Syntax und laden der benötigten Plugins
- Erstellen von Übersicht
  - Was bereits vorhanden
  - Was wird geändert
  - Was für Operationen werden durchgeführt
- Provider Kommunikation (API)
- (Entfernen der Ressource(n))

Ablaufplan



- Check der Syntax und laden der benötigten Plugins
- Erstellen von Übersicht
  - Was bereits vorhanden
  - Was wird geändert
  - Was für Operationen werden durchgeführt
- Provider Kommunikation (API)
- (Entfernen der Ressource(n))

Variables

In Datei: variables.tf

```
variable "region" {
  description = "Region to be used"
  }
```

In Datei: terraform.tfvars

```
    region = "eu-central-1"
```

Im Code:

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

⇒ Kombinierbar mit Modulen

Variables

In Datei: variables.tf

```
variable "region" {
  description = "Region to be used"
  }
```

In Datei: terraform.tfvars

```
    region = "eu-central-1"
```

Im Code ab Version 0.12:

```
    region = var.region
```

⇒ Kombinierbar mit Modulen

Modules

```
module "name" {
  source = "/path/to/folder"
  variable1 = "foo"
  variable2 = "bar"
}
```

```
module "frontend" {
  source = "git::git@github.com:user/i-modules.git//m-name?ref=v0.1"

  variable1 = 8
  variable2 = 20
}
```

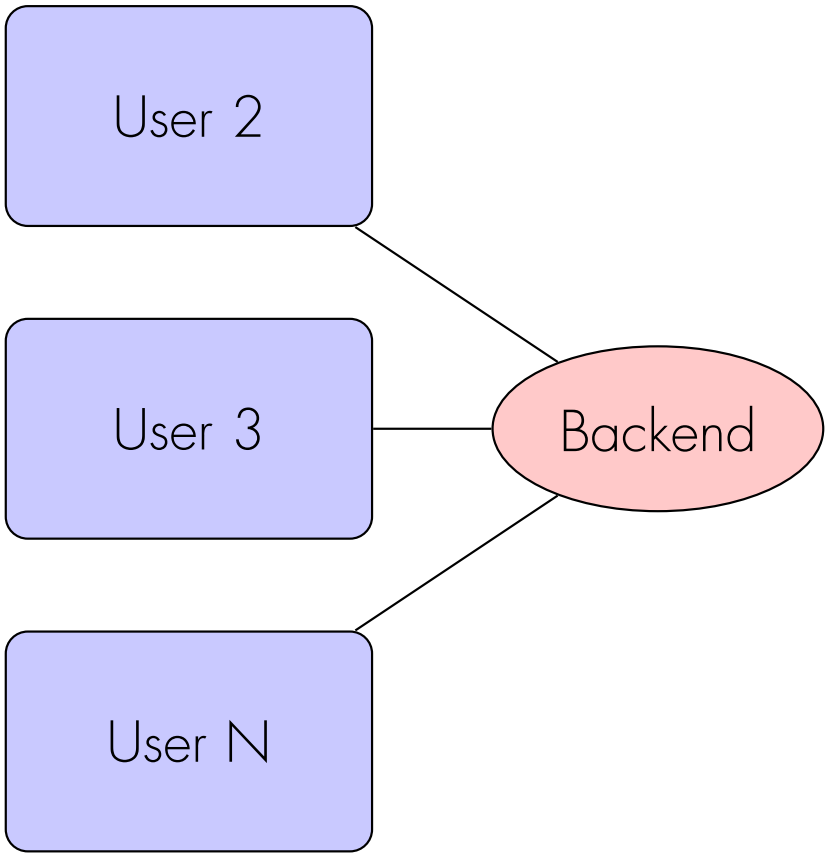
Modules

```
module "name" {
  source = "/path/to/folder"
  variable1 = "foo"
  variable2 = "bar"
}
```

```
module "frontend" {
  source = "git::git@github.com:user/i-modules.git//m-name?ref=v0.1"

  variable1 = 8
  variable2 = 20
}
```

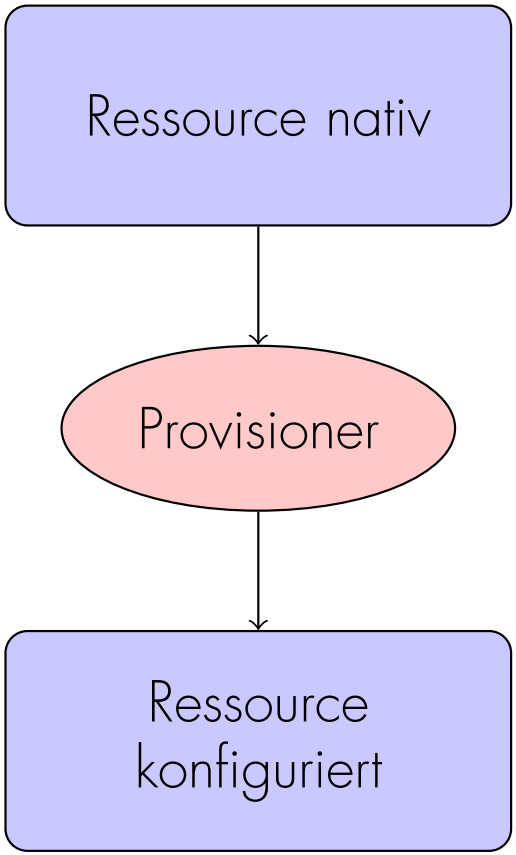
Zusammenarbeit mit Backends



Rolle von Provisioning

- Notwendigkeit abhängig von Provider/Ressourcentyp (VM vs. Container)
- Einrichten von 1-N Ressourcen
- Nutzen von Facts und Idempotenz

Rolle von Provisioning



- Notwendigkeit abhängig von Provider/Ressourcentyp (VM vs. Container)
- Einrichten von 1-N Ressourcen
- Nutzen von Facts und Idempotenz

Ansible vs. Terraform

Ansible

- Configuration management
- Verwenden der Facts für weitere Operationen
- Geeignet um Ressource einzurichten
- Dry-run nicht so informativ was geändert werden soll

Terraform

- Ressource orchestration
- Übersichtliche Auflistung, was geändert wird
- Sehr rudimentäres provisioning

⇒ Kombination um Vorteile beider zu nutzen

Ansible vs. Terraform

Ansible

- Configuration management
- Verwenden der Facts für weitere Operationen
- Geeignet um Ressource einzurichten
- Dry-run nicht so informativ was geändert werden soll

Terraform

- Ressource orchestration
- Übersichtliche Auflistung, was geändert wird
- Sehr rudimentäres provisioning

⇒ Kombination um Vorteile beider zu nutzen

Ansible vs. Terraform

Ansible

- Configuration management
- Verwenden der Facts für weitere Operationen
- Geeignet um Ressource einzurichten
- Dry-run nicht so informativ was geändert werden soll

Terraform

- Ressource orchestration
- Übersichtliche Auflistung, was geändert wird
- Sehr rudimentäres provisioning

⇒ Kombination um Vorteile beider zu nutzen

Kombination von Ansible und Terraform

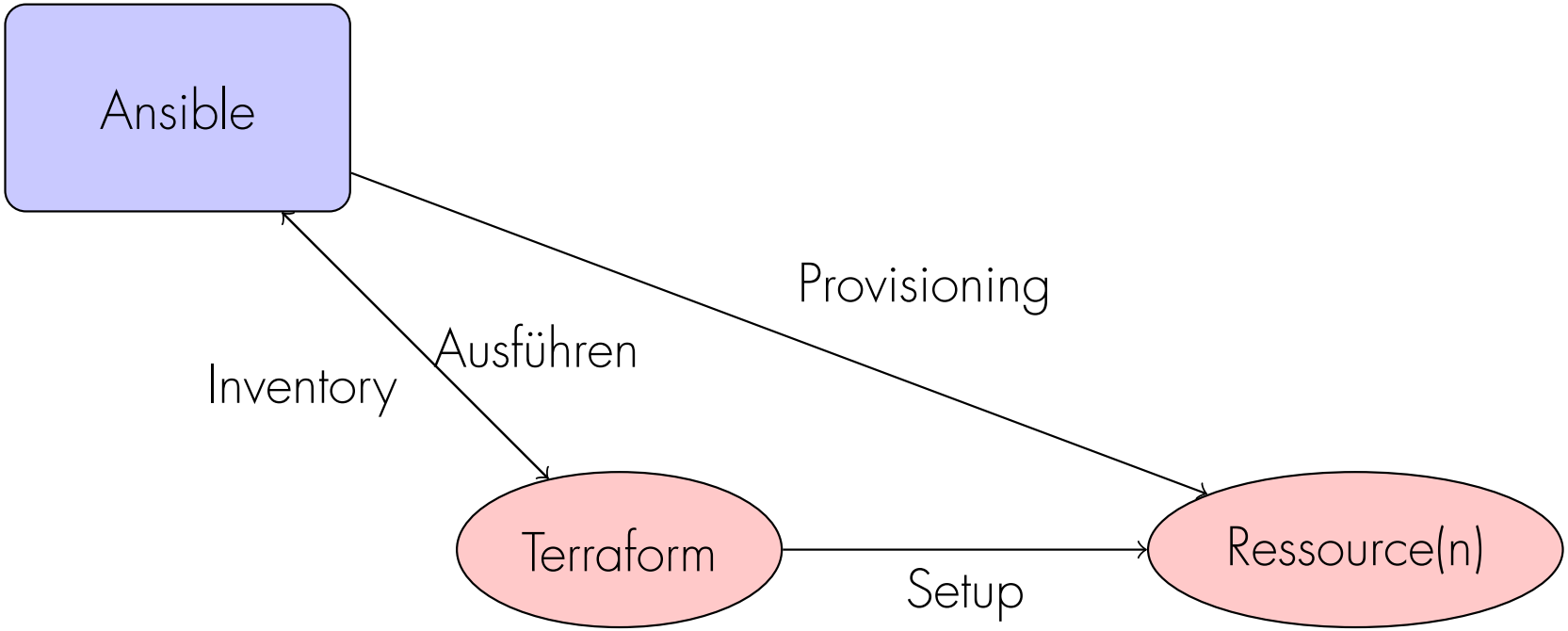
Vorteil Ansible

- Wiederverwendung existierender Rollen
- Nutzung von Distro-unabhängigen Modulen

Vorteil Terraform

- Ressourcenmanagement
- Detailliertere Übersicht der Änderungen
- Drift detection
- Komplette Lifecycle Kontrolle (Erstellen, Ändern, Entfernen)
- Gleiche Sprache um Infrastruktur und Ressourcen einzurichten

Kombination von Ansible und Terraform



Demo - Ansible - Terraform - AWS

```
ansible-playbook --inventory=terraform-inventory ../main.yaml
```

Demo - Ansible - terraform - AWS

```
---
- hosts: localhost
  connection: local
  gather_facts: False
  tasks:
  - name: Run terraform
    terraform:
      project_path: '{{ project_dir }}'
      state: present
      variables:
        counts : "{{ counts | default(1)}}"
  - name: Reload inventory
    meta: refresh_inventory

- hosts: "aws"
  remote_user: user
  become: true
  roles:
  -  role: nginx
...
```

Terraform Code

```
provider "aws" {
  region                  = "${var.region}"
  shared_credentials_file = "${var.loc}"
  profile                 = "${var.profile}"
}
```

Terraform Code

```
resource "aws_instance" "web" {
  count = "${var.counts}"
  ami                    = "${var.ami}"
  instance_type          = "${var.ami_type}"
  key_name               = "${var.aws_keyname}"
  vpc_security_group_ids = ["${var.sg_id}"]
  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update && sudo apt -y install python",
    ]
    connection {
      type        = "ssh"
      user        = "ubuntu"
      private_key = "${file(var.pkey)}"
      agent       = false
    }
  }
  tags {
    Name = "${var.aws_tag}-${count.index}"
  }
}
```

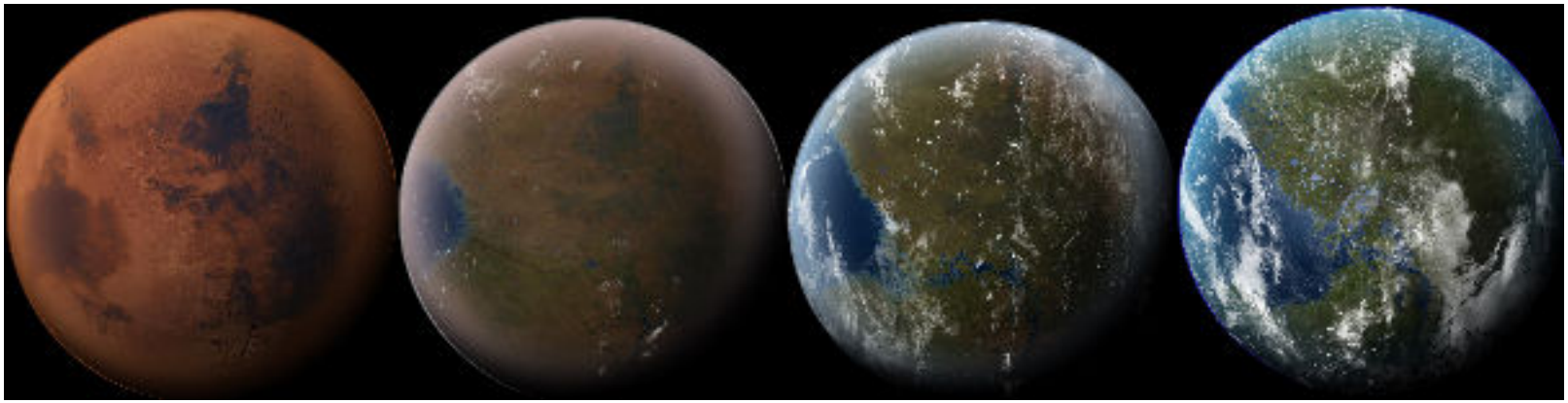
Demo - Ansible - Terraform - AWS

Aktuelle Kritikpunkte

- Sehr gute Übersicht der potentiellen Änderungen bevor diese durchgeführt werden
- Code ist recht Provider-Spezifisch
  - Durch unterschiedlicher Technologie nicht alle Settings lassen sich 1:1 übertragen
- Provisinors sind recht limitiert
  - aktuell nur Chef, Salt und rudimentäre Provisinors
  - aktuell kein Support für Ansible oder Puppet

Terraform ist immer noch in 0.x Status dewegen werden viele Sachen noch addressiert und können innerhalb nächster Zeit schon umgesetzt werden

Terraform - Ein Versuch ist es Wert



https://en.wikipedia.org/wiki/Image:MarsTransitionV.jpg