OPEN TELEKOM CLOUD

Hands-on workshop: **Infrastructure Provision with Terraform or OpenTOFU**

Open Telekom Cloud, Ecosystem Squad

- **Anton Sidelnikov (Dev Lead)**
- **Nils Magnus (Principal Cloud Architect)**

OpenInfra Summit Europe in Paris, France

October 15 – 17, 2025

Who we are

Ecosystem Squad of the Open Telekom Cloud

Nils Magnus

DevRel and Community Outreach Manager

Berlin, Germany
GitHub: @Nils-Magnus

Anton Sidelnikov

Lead Developer Golang and Terraform

Granada, Spain
GitHub: @anton-sidelnikov

Open Telekom Cloud

Creating a Secure Connected World

Today‘s Workshop: System Provisioning

100% automation for all devops tasks

- Why Terraform?
  - have something better than **Heat**,
  - platform **independent**,
  - could be used for **multi-platforms** (Multicloud).
- Skill **prerequisites**? Basic **Linux** administration, familiar with **OpenStack** resources, an idea of **Infrastructure**, look up **documentation**, basic **Git.**
- What will you **learn** today? How to use the **Open Telekom Cloud provider** to provision arbitrary resources
- Is this session **specific** for the Open Telekom Cloud? For OpenStack?
  It can be applied to any Terraform environment.
- Ease of use of **a language (DSL): HCL** is a de facto equivalent with Json or YAML, but more readable.

Agenda for today‘s workshop

Three interactive examples plus some background on slides

- **Infrastructure as Code (IaC)** – Introduction and concepts

- **Background and Terraform Basics**
- **Practical Scenario #1** – Provision compute instances

#1: compute instance

- **Terraform Advanced Concepts:** Complex structures and patterns
- **Practical Scenario #2** – Deploy a full serverless application

#2: complex real world setup

- Integrate **Terraform** into **GitOps** with **Atlantis**

- **Pulumi** – IaC with general-purpose programming languages (Python, C#, Go, etc.)
- **Practical Scenario #3** – Provisioning compute instances

#3: alternatives

Prerequisites and Installation

Workshop accounts for the Open Telekom Cloud

- We prepared OTC accounts with IAM user credentials for SSH-login (sign password from list
  - Prepared Accounts oise-01 .. oise-20
  - Domain OTC00000010000000000447
- Use domain responsibly; all data is going to be deleted by October 31, 2025 without further notice.
- Preferred code editor or use Visual Studio Code with devcontainer inside /**.devcontainer**/ folder
- Terraform and Ansible are installed:
  - For Terraform just copy the latest binary from the website into executable path: https://developer.hashicorp.com/terraform/install
  - Terraform provider: https://registry.terraform.io/providers/opentelekomcloud/**opentelekomcloud**
- **We prepared a GitHub repo for all the examples of this workshop: https://github.com/opentelekomcloud-community/terraform-workshop**

Infrastructure As Code

Everything is a text file

- Overarching goal: Managing infrastructure with tools that use configuration files (typically declarative) to consistently provision hardware into a defined state with each execution.
- Idempotence is a fundamental principle of Infrastructure as Code (IaC).
  - Each deployment execution consistently enforces the same desired state.
  - For example, if a server is already configured with the correct user accounts and packages, reapplying the configuration does not duplicate users or reinstall packages unnecessarily.
- Integration in a GitOps infrastructure (powered by a CI-Server like Zuul) is easy.

Benefits of IaC

Why to spend time on the learning curve?

- Reproducible infrastructure
  - → Infrastructure can be reliably recreated in the same way every time.
- Easily redeployable
  - → Environments can be quickly redeployed with minimal effort.
- Solution to infrastructure drift
  - → Ensures consistency by preventing or correcting infrastructure drift.
- Can be stored in VCS
  - → Configurations can be version-controlled in source repositories.
- Easily automated
  - → Infrastructure management can be fully automated.

Ansible, Chef, Puppet, ...

Too much choice or: the right tool for the right task?

- Configuration Management Tools: manage resources after provisioning
  - Provisioning: hardware / OS (e.g., Terraform, CloudFormation)
  - Configuration: software setup (e.g., Ansible, Puppet, Chef)
- IaC + Configuration Management
  - work best when combined
- *Example*: Use **Terraform** to create servers and infratsructure, then **Ansible** to install and configure applications
- Theoretically, everything is possible with any tool (Ansible can also provision servers and Terraform can also run scripts to confiure your software), but there are best practices ...

Terraform

The defacto standard in vendor-agnostic provisioning

- Terraform is an IaC-tool by Hashicorp (bought by IBM) two operation modes:
  - Terraform CLI – to run locally in your own CI/CD
  - Terraform Cloud for Enterprise
    - Cloud – Remote State Management with CI/CD integrations.
    - Enterprise – Self-hosted distribution of Terraform Cloud

- Used to be free software, but since 08/2023 they switched from MPL 2.0 to BSL 1.1: now usable without costs, but not free
- Terraform has many providers for provisioning many types of resources
- Registry of community build providers and modules: https://registry.terraform.io/providers/opentelekomcloud
- The provider works also for OpenTofu (complete community version)

Basics

Providers

- HashiCorp Configuration Language (HCL): declarative syntax
- Define desired state, Terraform computes required changes
- Providers connect Terraform to target platforms (e.g., OpenTelekomCloud)

Example provider configuration:

```
provider "opentelekomcloud" {
  region       = "eu-de"
  auth_url     = "https://iam.eu-de.otc.t-systems.com/v3"
  domain_name  = "OTC-EU-DE"
  username     = var.username
  password     = var.password
  project_name = "eu-de"
}
```

Basics

Variables and Outputs

- Variables allow reusability and parameterization
- Outputs provide key information after deployment for subsequent pipeline steps of the GitOps process, for instance
- Example:

```
variable "username" { type = string }
output "vm_ip" { value = opentelekomcloud_compute_instance_v2.vm.access_ip_v4 }
```


Basics

Minimal OpenStack resource configuration

```
resource "opentelekomcloud_compute_instance_v2" "vm" {
  name        = var.name
  image_name  = var.image_name
  flavor_name = "s2.small.1"
  key_pair    = var.kp.name
  network {
    uuid = opentelekomcloud_vpc_subnet_v1.subnet.network_id
  }
}
```

Helpcenter

https://docs.otc.t-systems.com/ terraform-provider-opentelekomcloud/

List of all resources:

https://docs.otc.t-systems.com/terraform-provider-opentelekomcloud/data-sources/i

How does a service work?

https://docs.otc.t-systems.com/
➔ go to page with service description and API-reference

- **Which resource types are available? Where can I find a list?**
- **Which resource attributes are available? Where can I find them? Which are mandatory, which optional?**
- **How do I know I can use an attribute as the value for another resource?**

Interactive Scenario #1

Provision compute instance with NGINX

Task: Create a VM in a custom VPC/subnet, attach a Floating IP, install NGINX via Ansible, and expose it over SSH/HTTP

Key Elements in the Code

- Key pair: Generated in Terraform and saved locally as pk.pem (0600).
- Network: VPC + subnet (custom CIDRs, DNS, AZ).
- Security group: Allow SSH (22) and HTTP (80) from anywhere.
- Port: Created in the subnet and attached to the SG.
- Boot volume: 50 GB SSD from the selected image; flavor chosen by name.
- Compute instance: Attached to the port and boot volume.
- Public access: Floating IP allocated and associated to the port.
- Readiness/provisioning: Wait for SSH/cloud-init, then run Ansible to install NGINX.
- Outputs: Ready-made SSH command and HTTP URL.

Benefits

- Declarative IaC with Terraform (+ Ansible for config).
- Reproducible VM + networking with public reachability.
- Safer defaults: key stays local; SG limited to SSH/HTTP only.
- Delivers a working web server immediately after apply.
- Easy to extend (e.g., add count later for multiple VMs).

Advanced Terraform features

Loops: `count` and `for_each`

- **`count` – deploy multiple identical resources**
- **`for_each` – deploy one resource per item in a list/map**
  - **Use for_each with maps to name resources based on keys**

Example: create multiple VMs or subnets dynamically

```
variable "dns_zones" {
  type    = list(string)
  default = ["eu-de", "eu-nl"]
}

resource "opentelekomcloud_dns_zone_v2" "pub" {
  count = length(var.dns_zones)

  name        = "${var.dns_zones[count.index]}.example.com."
  email       = "${var.dns_zones[count.index]}@example.com"
  description = "Public DNS zone for ${var.dns_zones[count.index]}"
  ttl         = 3000
  type        = "public"
}
```

```
resource "opentelekomcloud_dns_zone_v2" "pub" {
  for_each = {
    a_group     = "first domain"
    b_group = "second domain"
  }

  name        = "${each.key}.example.com."
  email       = "${each.key}@example.com"
  description = "Public DNS zone for ${each.value}"
  ttl         = 3000
  type        = "public"
}
```

Advanced Terraform features

Conditionals

- Conditional expressions: *condition* ? *true_value* : *false_value*
- Control number of resources or parameter values dynamically

Example: Deploy HA VMs only if high_availability = true

```
variable "high_availability" {
  type    = bool
  default = false
}

resource "opentelekomcloud_compute_instance_v2" "example" {
  # If HA is true → create 2 instances, else only 1
  count = var.high_availability ? 2 : 1

  name            = "example-${count.index}"
  flavor_id       = "s2.medium.1"
  image_id        = "your-image-id"
  key_pair        = "your-keypair"
  security_groups   = ["default"]
  availability_zone = "eu-de-01"

  network {
    uuid = "your-network-id"
  }
}
```

Advanced Terraform features

Local Values

- Locals store computed values to avoid repetition
- Example: common_tags = { Environment = var.env, Project = var.project }
- Merge locals with specific tags for resources

```
variable "secgroup_rules" {
  description = "Map of security group rules"
  type = list(object({
    direction       = optional(string, "ingress")
    ethertype       = optional(string, "IPv4")
    protocol        = optional(string, "tcp")
    port_range_min   = number
    port_range_max   = number
    remote_ip_prefix = optional(string, "0.0.0.0/0")
  }))
  default = []
}

locals {
  sg_rules = [for r in var.secgroup_rules : merge({}, r)]
}

resource "opentelekomcloud_networking_secgroup_rule_v2" "rules" {
  for_each        = { for idx, r in local.sg_rules : idx => r }

  direction       = each.value.direction
  ethertype       = each.value.ethertype
  protocol        = each.value.protocol
  port_range_min   = each.value.port_range_min
  port_range_max   = each.value.port_range_max
  remote_ip_prefix  = each.value.remote_ip_prefix
  security_group_id = opentelekomcloud_networking_secgroup_v2.apigw.id
}
```

Advanced Terraform features

State Management and Remote Backends

- State stores the current infrastructure configuration
- Remote backends allow team collaboration
- OTC OBS (Object Storage) can act as S3-compatible backend
- Enables state locking and versioning
- Be careful with configuration drift!

https://registry.terraform.io/providers/opentelekomcloud/opentelekomcloud/latest/docs/guides/backends

```
terraform {
 required_version = ">= 1.6.3"

 required_providers {
  opentelekomcloud = {
   source  = "opentelekomcloud/opentelekomcloud"
   version = ">= 1.36.0"
  }
 }
 backend "s3" {
  endpoints = {
   s3 = "https://obs.eu-de.otc.t-systems.com/"
  }
  key                         = "terraform_state/test"
  bucket                      = "tf-test-bucket"
  region                      = "eu-de"
  skip_credentials_validation = true
  skip_region_validation      = true
  skip_requesting_account_id  = true
  skip_metadata_api_check     = true
  skip_s3_checksum            = true
  secret_key                  = "secret"
  access_key                  = "access"
 }
}
```

Advanced Terraform features

Importing Existing Resources

- Import allows bringing manually created resources under Terraform control
- Syntax:

```
terraform import resource_type.name id
```

- Example:

```
terraform import opentelekomcloud_compute_instance_v2.vm uuid
```

Multitenancy

Folder-based Separation

- Each tenant/environment has its own folder
- Separate `backend.tf` with unique OBS bucket/key per tenant
- Can use different credentials or backend types per tenant
- Shared modules for consistent infrastructure code

Pros & Cons:

✅ Full isolation: separate OBS buckets or keys

✅ Different credentials per tenant possible

✅ Flexible backend configuration

❌ More backend.tf files to maintain

❌ Need to init each folder separately

Multitenancy

Workspace-based Separation

- Single codebase for all tenants
- One backend configuration shared by all workspaces
- Backend key includes ${terraform.workspace} to separate state files
- Same OBS bucket and credentials for all tenants

Pros & Cons:

✅ Minimal duplication of backend config

✅ Easy to switch tenants with workspace commands

✅ State isolation via workspace naming in key

❌ Same OBS bucket and credentials for all tenants

❌ No per-tenant backend customization

Multitenancy

Summary

Folders: choose for strong isolation or varying backend settings

Workspaces: choose for simplicity and similar tenant requirements

Hybrid: shared modules + separation method that fits security/state needs

Interactive Scenario #2

Serverless API with FunctionGraph, API Gateway & Redis

Task: Package a Python function, deploy it to FunctionGraph, expose it via API Gateway over HTTPS at /hello/{username}, and use a managed Redis cache - all inside a custom VPC/subnet with controlled security.

Key Elements in the Code

- Networking (module): VPC + subnet and a security group allowing HTTP (80) and HTTPS (443).
- Packager (module): Builds and uploads the function artifact from **function** to OBS (workshop-bucket), producing an object URL used by the function.
- Redis (module): Managed Redis (workshop-redis) v6.0, tiny HA flavor, in the same VPC/subnet; password passed via variable; maintenance window set.
- Function (module):
  - Name birthday-api, runtime Python 3.9, handler handler.func_handler, 256 MB, 10 s timeout.
  - Code from OBS via code_url; agency (fg_agency) for access.
  - Attached to the VPC/subnet; gets Redis host/port/password; logging group/stream configured.
- API Gateway (module):
  - Dedicated gateway workshop-gw (BASIC spec) in the same VPC/subnet/SG.
  - API birthday-api served over HTTPS, method ANY, path /hello/{username} with a mapped path parameter.
  - Integrated with the FunctionGraph URN, sync invocation, 5 s backend timeout.

Benefits

- End-to-end automation: From packaging to a live HTTPS endpoint in one terraform apply.
- Modular & reusable: Networking, packaging, Redis, function, and gateway are cleanly separated modules.
- Secure by design: Private function + Redis in VPC; only the gateway is exposed; SG limited to 80/443.
- Configurable & portable: Environment and secrets via variables;
- Serverless scale: API scales with demand; no servers to manage.

GitOps - Atlantis?

Architecture

- Atlantis integrates Terraform with Git-based workflows
- Listens for pull requests and comments with plan output
- Authorized reviewers can trigger apply directly from PR comments

- Open Source under the Apache License 2.0

GitOps - Atlantis

Configuration

```
version: 3
automerge: true

projects:
  - name: minimal
    dir: terraform-minimal
    workspace: default
    autoplan:
      enabled: true
      when_modified:
        - "**/*.tf"
        - ".terraform.lock.hcl"
    terraform_version: v1.7.5
    apply_requirements: ["approved", "mergeable"]
```

```
workflows:
  default:
    plan:
      steps:
        - init
        - plan
    apply:
      steps:
        - apply
```

GitOps - Atlantis

Flow

- GitOps – manage infrastructure via Git pull requests
- Atlantis – automation tool for Terraform/Terragrunt in GitOps
- Enables policy control, team collaboration, audit trail

Pulumi

Language-based IaC

- Pulumi implements IaC using general-purpose languages
- Supports Python, TypeScript/JavaScript, Go, C#, ...
- Enables leveraging programming language features (loops, tests, packages)

Pulumi

Mapping to opentelekomcloud/OpenStack Provider

- Pulumi uses cloud provider SDKs and APIs
- For the Open Telekom Cloud, we use the OpenStack provider with matching resource definitions
- Concepts similar to Terraform's provider, but with code syntax

```
import pulumi
from pulumi_openstack import images, networking, compute

cfg = pulumi.Config()
instance_count = cfg.get_int("instance_count") or 2

# External network lookup (router:external = true)
ext_net = networking.get_network(external=True)

# Router with external gateway + interface to our subnet
router = networking.Router(
    "workshop-router",
    admin_state_up=True,
    external_gateway=ext_net.id,
)
```

Pulumi

Quick Start

- Create a new project
  - $ pulumi new – bootstrap your project
- Manage and view state
  - $ pulumi stack

```
Current stack is dev:
  Owner: anton-sidelnikov-org
  Last updated: 3 weeks ago (2025-09-09 19:58:01.275854 +0200 CEST)
  Pulumi version used: v3.193.0
Current stack resources (0):
  No resources currently in this stack
```

- Set the default destination org for all stack operations
  - $ pulumi org set-default NAME
- View backend, current stack, pending operations, and versions
  - $ pulumi about

```
CLI
Version      3.193.0
Go Version   go1.25.1
Go Compiler  gc

Plugins
KIND      NAME       VERSION
resource  openstack  3.15.2
language  python     3.193.0
```

Open Telekom Cloud

Creating a Secure Connected World

Interactive Scenario #3

Provision compute instances

Task: Provision three identical VMs in a custom network with router and floating IPs

Key Elements in the Code

- Configurable instance count (pulumi config set example:instance_count 3)
- Network setup: Private subnet, router with external gateway
- Security group: ICMP (ping) enabled
- Loop over count
- Create port in subnet
- Create VM attached to port
- Allocate Floating IP and associate to port
- Exports: Internal IPs, floating IPs, network, subnet, router IDs

Benefits

- Declarative infrastructure as code in Python
- Scales easily with parameterized parameters like instance_count
- Reusable pattern for multi-VM labs, demos, or clusters
- Finally, we export the VM's public IP as an output, so it's displayed after the deployment.

Links & Resources

Terraform documentation:
https://developer.hashicorp.com/terraform/docs

Infrastructure as Code in the Open Telekom Cloud Helpcenter:
https://docs.otc.t-systems.com/developer/iac.html

Open Telekom Cloud Terraform provider:
https://registry.terraform.io/providers/opentelekomcloud/opentelekomcloud/latest

Source Code and Issues:
https://github.com/opentelekomcloud/terraform-provider-opentelekomcloud

Pulumi: https://www.pulumi.com/docs

Atlantis: https://www.runatlantis.io

Contact, we accept PRs and Issues in GitHub

Ecosystem Squad of the Open Telekom Cloud
Powered by T-Systems International GmbH

{nils.magnus,a.sidelnikow,artem.lifshits}@t-systems.com