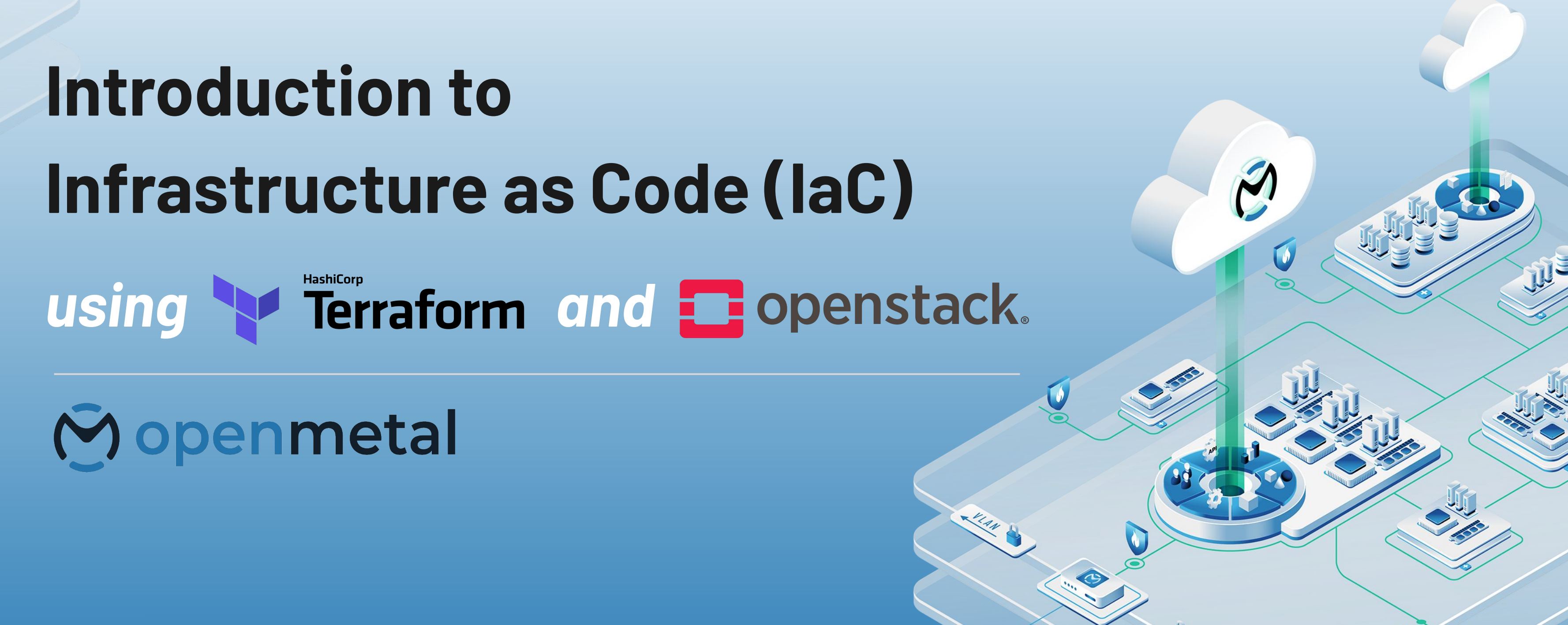
Introduction to Infrastructure as Code (IaC)

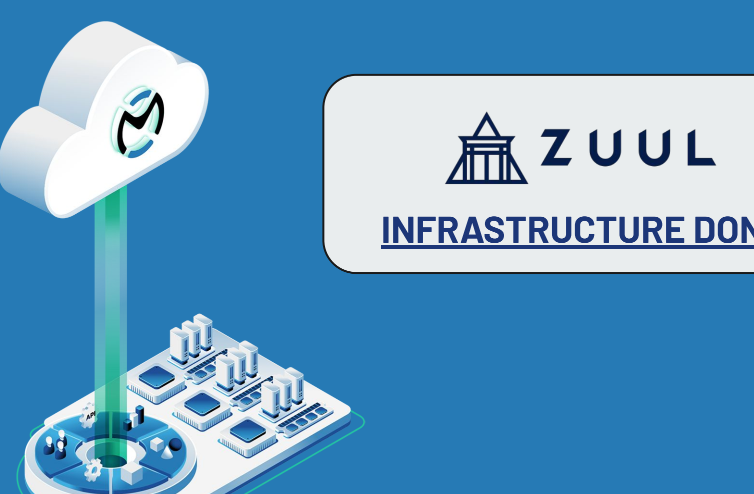


Chris Bermudez

Director, DevOps Engineering
OpenMetal

chrisb@openmetal.io

- ➔ OpenMetal Baremetal and Backend Automation and Deployment system
- ➔ Monitoring and instrumentation for end user clouds.
- ➔ Research & development of new technologies and platform improvements



ZUUL
INFRASTRUCTURE DONOR

OpenMetal is an Infrastructure as a Service (IaaS) provider that believes in the collective and fundamental good of open source in the information technology world.

Our Mission: Make highly complex open source systems available on-demand to increase accessibility for smaller teams.

Agenda

Brief Introduction to IaC, Terraform, and OpenStack

- Quick overview of the importance of IaC
- Quick overview of Terraform and its advantages
- Quick overview of OpenStack and its role in cloud computing

Basic Terraform Usage

- Introduction to Terraform Configuration Files
- Applying Terraform Configuration
- Exercise 1: Students write and apply a basic Terraform configuration to create an OpenStack resource

Agenda (cont)

Terraform with OpenStack

- Brief explanation and demonstration of configuring the Terraform OpenStack provider
- Managing OpenStack Resources with Terraform
- Exercise 2: Students write and apply a Terraform configuration to set up a basic OpenStack architecture

Wrap Up and Q&A

- Key Takeaways and Best Practices
- Q&A Session

The Importance of Infrastructure as Code (IaC)

- ➔ Overall Automation and Efficiency
- ➔ Consistency and Standardization
- ➔ Version Control and Collaboration
- ➔ Rapid Scalability
- ➔ Cost Savings
- ➔ Disaster Recovery

The Importance of Infrastructure as Code (IaC)

IaC is a fundamental practice for teams that aim to **improve efficiency, consistency, and scalability** in their infrastructure management processes.

- ➔ Helps reduce the potential for human error
- ➔ Enhances collaboration
- ➔ Can ultimately contribute to the delivery of higher quality software, faster

Terraform and its advantages

- Platform Agnostic
- Declarative Language
- Modular and Reusable Code
- Resource Relationships
- Change Automation and Management
- State Management
- Immutable Infrastructure

Terraform and its advantages

Terraform's **multi-platform support, declarative nature, and robust change management capabilities** make it a powerful tool for managing complex infrastructures.

OpenStack and its role in cloud computing

openstack®

- Offers complete control and customization
- Provides a wide range of services
- Infrastructure Management
- Scalability and Efficiency
- Multi-tenancy
- Large Community and Ecosystem
- Can make Private and Hybrid Clouds

Basic Terraform Usage

Providers, Resources, and Data Sources

Providers are responsible for managing resources of a specific cloud or service.

provider.tf

```
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  password    = "password"
  auth_url    = "http://openstack.example.com:5000/v3"
  region      = "RegionOne"
}
```

Resources represent the infrastructure objects you want to manage, such as virtual machines, networks, or storage volumes. Each resource block defines a specific resource type and its properties.

resources.tf

```
resource "openstack_compute_instance_v2"
"my_instance" {
  name = "terraform-instance"
  image_name = "image_id"
  flavor_name = "flavor_id"
  key_pair = "keypair_name"

  network {
    name = "network_name"
  }
}
```

Data sources provide information from external sources, such as querying existing resources in your cloud environment. You can use data sources to retrieve details about existing networks, images, or flavors.

data-sources.tf

```
data "openstack_networking_network_v2"
"example_network" {
  name = "my-network"
}
```

Variables and Outputs

Variables are a convenient way to customize Terraform configurations. For example, you could define the image name as a variable, then reuse it across resources and modules

Outputs are a way to tell Terraform what data to return at the end of apply. You could use it to print an IP address, a URL, or any other information about the resources

variables.tf

```
variable "image_name" {
  description = "The name of the image to use for the instance"
  type        = string
  default     = "image_id"
}

resource "openstack_compute_instance_v2" "my_instance" {
  // ...
  image_name = var.image_name
  // ...
}
```

outputs.tf

```
output "ip_address" {
  value =
openstack_compute_instance_v2.my_instance.access_ip_v4
}
```

Applying Terraform Configuration

- `terraform init` - This command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration. It downloads the necessary provider plugins
- `terraform plan` - This command creates an execution plan. It is used to see what changes Terraform will make to your infrastructure before actually making those changes.
- `terraform apply` This command applies the desired changes to reach the desired state of the configuration, or the predetermined set of actions generated by a `terraform plan` execution plan.
- `terraform destroy` - This command is used to destroy the Terraform-managed infrastructure. It's the opposite of `terraform apply`, it terminates all the resources specified in the configuration.

Exercise 1

Lets write a basic Terraform configuration to create an OpenStack network resource

example-1.tf

```
# Dont worry about this part for now!
provider "openstack" {
}

resource "openstack_networking_network_v2" "oif_workshop_net" {
  name           = "oif_workshop_net"
  admin_state_up = true
}
```

Using Terraform with OpenStack

Terraform OpenStack Provider

To interact with OpenStack, Terraform uses a configured provider to make API requests on your behalf. Before you can use the provider, you must configure it with the proper credentials.

provider.tf

```
provider "openstack" {
  auth_url          = "https://<auth_url>"
  username          = "<username>"
  password          = "<password>"
  tenant_name       = "<tenant_name>"
  project_name      = "<project_name>"
  user_domain_id    = "<user_domain_id>"
  project_domain_id = "<project_domain_id>"
}
```

Terraform OpenStack Provider (cont.)

For security reasons, it's recommended not to hardcode your credentials into your Terraform files. A best practice is to load them from environment variables like the ones provided by the OpenStack RC

os_env.sh

```
export OS_USERNAME="my-username"
export OS_TENANT_NAME="my-tenant"
export OS_PASSWORD="my-password"
export OS_AUTH_URL="http://openstack.example.com:5000"
export OS_REGION_NAME="my-region"
```

Then, your provider configuration would simply look like this:

provider.tf

```
provider "openstack" {}
```

Managing OpenStack Resources - Instances

Let's start by defining a provider and creating an instance.

`openstack_compute_instance_v2` resource type is used to create an instance.

instance.tf

```
provider "openstack" {}

resource "openstack_compute_instance_v2" "oif_ubuntu" {
  name = "oif_ubuntu"
  image_name = "Ubuntu 22.04 (Jammy)"
  flavor_name = "gen.small"
  key_pair = "my_keypair"

  network {
    name = "oif_workshop_net"
  }
}
```

Managing OpenStack Resources - Network

openstack_networking_network_v2 resource type is used to create a network and openstack_networking_subnet_v2 can be used to create a subnet on it.

instance.tf

```
resource "openstack_networking_network_v2" "oif_workshop_net" {
  name = "oif_workshop_net"
  admin_state_up = "true"
}

resource "openstack_networking_subnet_v2" "oif_workshop_subnet" {
  name = "oif_workshop_subnet"
  network_id = openstack_networking_network_v2.oif_workshop_net.id
  cidr = "192.168.1.0/24"
  ip_version = 4
}
```

Managing OpenStack Resources - Floating IPs

You can use **openstack_networking_floatingip_v2** to create a floating IP.
To assign it, use **openstack_compute_floatingip_associate_v2**

floating-ip.tf

```
resource "openstack_networking_floatingip_v2" "oif_fip" {
  pool = "External"
}

resource "openstack_compute_floatingip_associate_v2" "oif_fip_1" {
  floating_ip = openstack_networking_floatingip_v2.oif_fip.address
  instance_id = openstack_compute_instance_v2.oif_ubuntu.id
}
```

Exercise 2

Let's put everything together and write a Terraform configuration to set up a basic OpenStack architecture.

This will comprise of a Network with Subnet, Instance, and Floating IP.

Further Education/Best Practices

- Terraform Recommended Practices
- Terraform Registry - OpenStack Provider
- Terraform Standard Module Structure
- Running Terraform in Automation
- Terraform Backends
- Workshop GitHub Repository

openmetal

Thank you!

Q/A