VRIJE UNIVERSITEIT BRUSSEL

3RD VLAIO TETRA OPENCLOUDEDGE USER MEETING

OPENSTACK, TERRAFORM AND SERVERLESS COMPUTING

An Braeken, Kris Steenhaut, Steffen Thielemans, Luca Gattobigio, Priscilla Benedetti, Ruben De Smet

4 February 2021

AGENDA

- **Project announcements**
- Virtualized and Containerized workloads
- Deploying OpenStack infrastructure with Kolla-Ansible
- GPGPU and initial findings on OpenStack GPU passthrough
- Automating cloud image creation with Packer
- Terraform as cloud-agnostic way of specifying Infrastructure as Code
- Serverless computing and Lightweight virtualization with OpenFAAS
- Conclusions, Future Work and Q&A

PROJECT ANNOUNCEMENTS

- TETRA OpenCloudEdge project has been extended by 3 months
  - New deadline: 31 January 2022

- New members: AXS GUARD INTERNET SECURITY SOLUTIONS

Lumency

OUR CURRENT INFRASTRUCTURE

WHAT'S NEW?

- 3x HPE DL325 gen10
  - AMD EPYC 7302p 16 core processor, 64 GB RAM
  - Ceph mixed HDD/SSD distributed storage
  - **AMD Radeon Pro W5500 GPU**

- **CentOS 8.3** as operating system

- Kubernetes (+- production ready) & **OpenStack** (still under evaluation)

kubernetes

openstack®

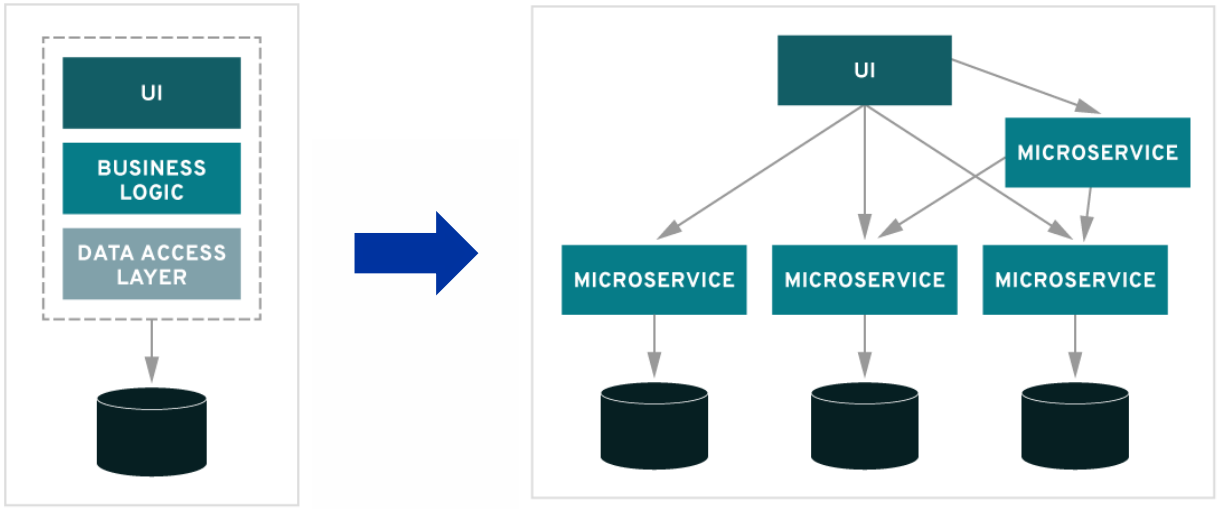
AGENDA

- Project announcements
- **Virtualized and Containerized workloads**
- Deploying OpenStack infrastructure with Kolla-Ansible
- GPGPU and initial findings on OpenStack GPU passthrough
- Automating cloud image creation with Packer
- Terraform as cloud-agnostic way of specifying Infrastructure as Code
- Serverless computing and Lightweight virtualization with OpenFAAS
- Conclusions, Future Work and Q&A

VIRTUALIZED AND CONTAINERIZED WORKLOADS

THE NEXT GENERATION OF CLOUD COMPUTING

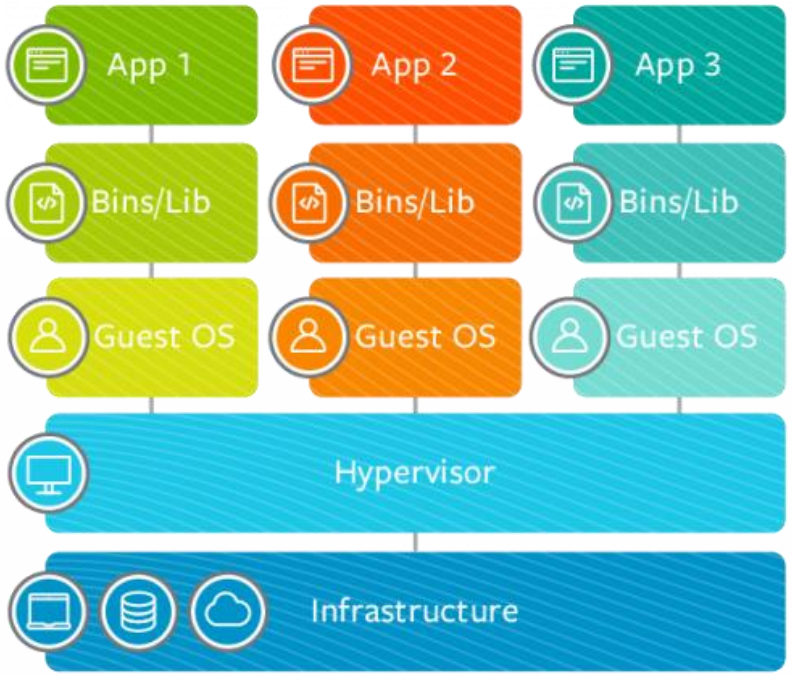
- Shift from **monolithic** applications towards **microservices**



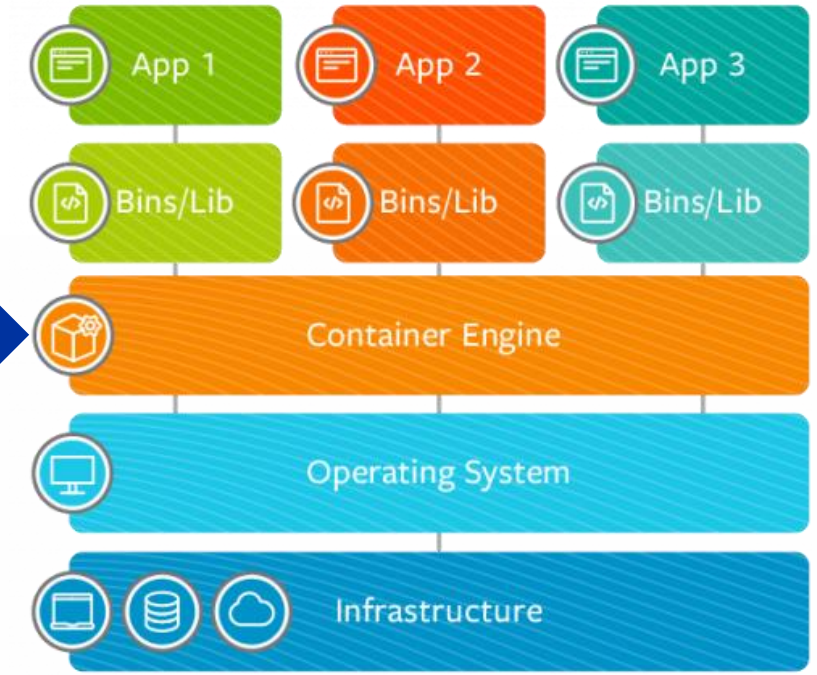
- Shift from **virtual machines** towards **containers**
  - Docker Containerization **+**
  - Kubernetes Orchestration
    - Details in our previous Opencloudedge user meeting

VIRTUALIZED AND CONTAINERIZED WORKLOADS

WHAT ARE THE MAIN DIFFERENCES



Virtualization



Containerization

- No more guest OS
  - Reduced overhead
  - Faster startup
  - → Improved elastic scaling

- Hardware level Virtualization
  - Guest OS's completely isolated

- Shared operating system kernel
  - Isolation via *namespaces & cgroups*

VIRTUALIZED AND CONTAINERIZED WORKLOADS

WHICH ONE SUITS MY TYPE OF APPLICATION?

There is no distinct answer… It depends (among others) on:

- **Design of the application**
  - Large monolithic → virtual machines
  - Microservices → containers
- **Application longevity**
  - Stateful & persistent apps → virtual machines (*)
  - Stateless & short-lived apps → containers
- **Scalability** → containers
- **Isolation requirements** → virtual machines
- **Application compatibility** → virtual machines (e.g. windows-only application)

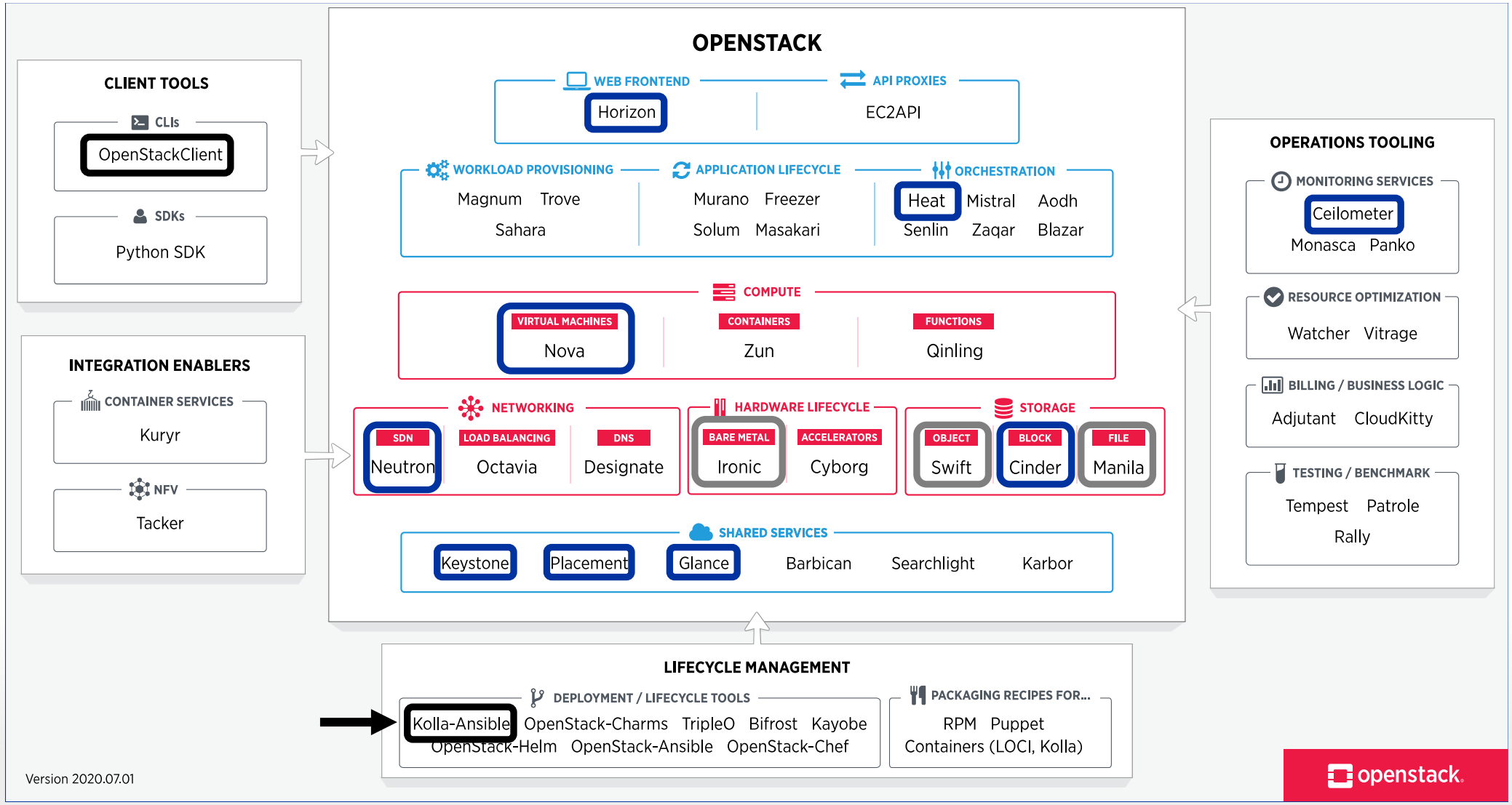
www.redhat.com/en/topics/containers/containers-vs-vms

AGENDA

- Project announcements
- Virtualized and Containerized workloads
- **Deploying OpenStack infrastructure with Kolla-Ansible**
- GPGPU and initial findings on OpenStack GPU passthrough
- Automating cloud image creation with Packer
- Terraform as cloud-agnostic way of specifying Infrastructure as Code
- Serverless computing and Lightweight virtualization with OpenFAAS
- Conclusions, Future Work and Q&A

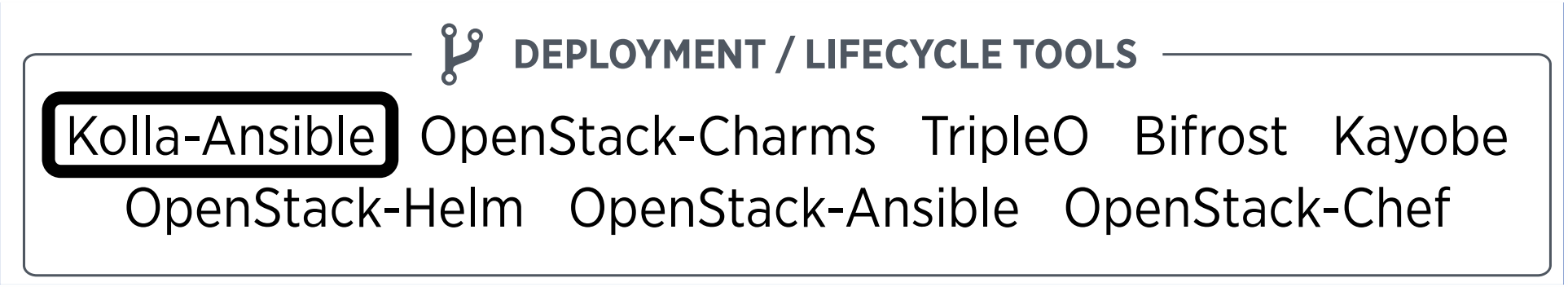
DEPLOYING OPENSTACK INFRASTRUCTURE

WHICH SERVICES ARE DEPLOYED AND/OR UTILIZED



DEPLOYING OPENSTACK INFRASTRUCTURE

KOLLA-ANSIBLE



- **Kolla**
  - Containerized versions of Openstack components
    - Easy to distribute, deploy and upgrade
    - Largely independent of host OS configuration

- **Ansible**
  - Automated software provisioning & deployments

- **Kolla-Ansible** deployment scripts

DEPLOYING OPENSTACK INFRASTRUCTURE

KOLLA-ANSIBLE RELEASES

- OpenStack *Ussuri* (05/2020) & *Victoria* (10/2020) releases
  - CentOS / RHEL 8
  - Debian 10 (Buster)
  - Ubuntu 20.04 (Focal Fossa)

- Kolla / Kolla-Ansible version releases trail by 1 – 2 months
  - … Which is not a bad thing for stability

- Out of the box configuration includes essential components & default configuration
  - Highly customizable component-specific configurations
  - Additional components can be enabled for deployment

DEPLOYING OPENSTACK INFRASTRUCTURE

KOLLA-ANSIBLE – DEPLOYMENT ISSUES

- OpenStack Kolla = Docker containers
- *kolla-ansible –i multinode bootstrap-servers*
  - Install and configure prerequisites (e.g. Docker)
- Docker already installed & configured for our Kubernetes cluster
  - Existing Kubernetes-based Docker config overwritten
  - *Incompatible cgroup drivers in use:* *cgroupfs* ↔ *systemd*
  - Worked, until out of memory issues occurred

Solution:

- Kolla-ansible with *docker_custom_config* environment variable in json file
  - *kolla-ansible –i multinode bootstrap-servers –e "cgroups.json"*

DEPLOYING OPENSTACK INFRASTRUCTURE

KOLLA-ANSIBLE – UPGRADE USSURI → VICTORIA

KOLLA
an OpenStack Community Project

Seamless migration from OpenStack *Ussuri* to *Victoria*

```
pip install --upgrade kolla-ansible
```

→ Migrate changes in inventory & *globals.yml* file

→ Generate passwords & migrate with existing passwords

- *kolla-genpwd* & *kolla-mergepwd*

`kolla-ansible pull`	fetch new Kolla docker images
`kolla-ansible upgrade`	upgrade OpenStack deployment

AGENDA

- Project announcements
- Virtualized and Containerized workloads
- Deploying OpenStack infrastructure with Kolla-Ansible
- **GPGPU and initial findings on OpenStack GPU passthrough**
- Automating cloud image creation with Packer
- Terraform as cloud-agnostic way of specifying Infrastructure as Code
- Serverless computing and Lightweight virtualization with OpenFAAS
- Conclusions, Future Work and Q&A

GENERAL PURPOSE COMPUTING ON GPU (GPGPU)

WHY?

For research:

- On **cryptography**: parallelized proof verification
- On **machine learning**: e.g. computer vision, neural networks

For OpenCloudEdge demos:

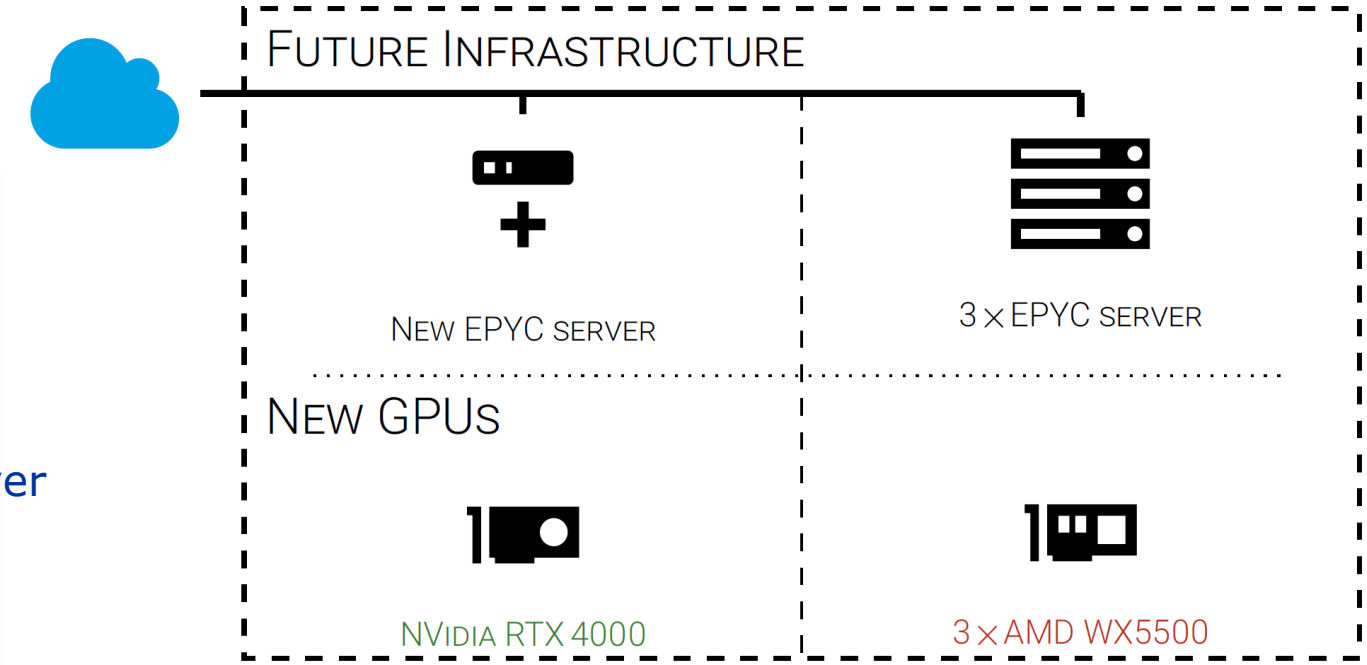
- Remote workstation for mechanics and robotics department
- Scalability over heterogeneous brands and types of GPUs

GENERAL PURPOSE COMPUTING ON GPU (GPGPU)

CURRENT AND FUTURE INFRASTRUCTURE

Our GPGPU cluster contains:

- Three EPYC servers with AMD WX5500
- One Old Desktop ® with Nvidia RTX 4000
- The RTX 4000 will move to a new EPYC server



OPENSTACK NOVA GPU PASSTHROUGH

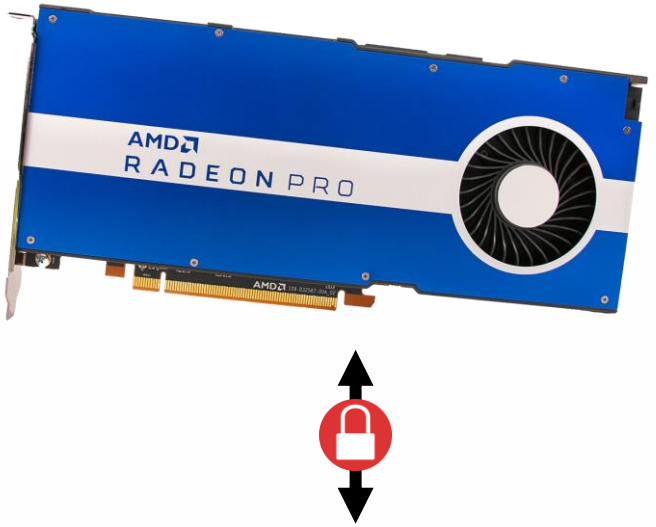
WHAT IS GPU PASSTHROUGH

What?

- Passthrough of physical GPU (or other PCI device) to virtual machine
  - Directly coupled and exclusive access between VM and device

Why?

- Allows a cloud tenant to leverage GPGPU acceleration



OPENSTACK NOVA GPU PASSTHROUGH

BRIEF PASSTHROUGH CONFIGURATION STEPS

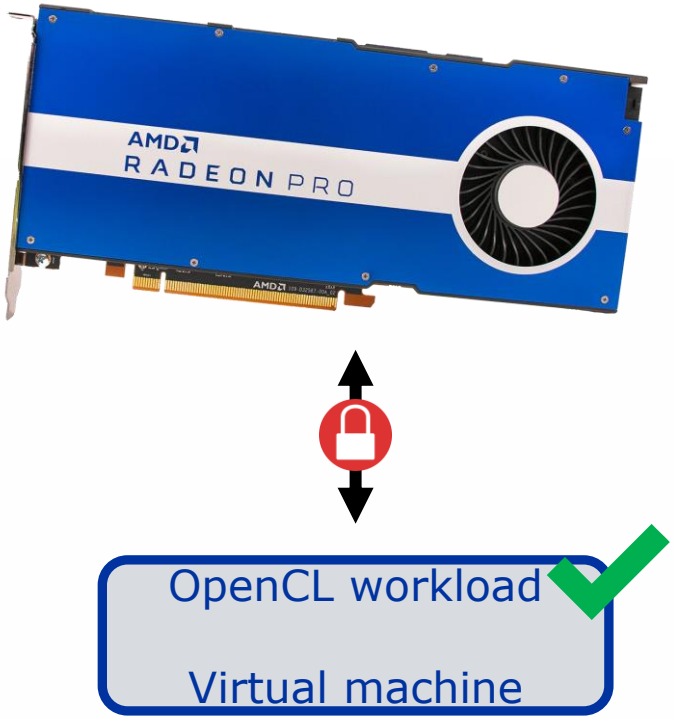
- Enable Input/Output Memory Management Unit (I/O MMU)
  - Essential for enabling PCI passthrough to VMs
- Set up kernel drivers
  - Disable (blacklist) *amdgpu* kernel drivers
  - Configure *vfio-pci* kernel driver for passing through GPU's vendor – product ID
  - (Permanently) load *vfio-pci* kernel driver
- Configure OpenStack Nova
  - Nova-compute (specific machine)
    - Whitelist vendor:product ID for passthrough
  - Nova-api (all controller nodes)
    - Provide an **alias** (e.g. *W5500*) to the whitelisted vendor – product ID
  - Nova-scheduler (all controller nodes)
    - Enable *PciPassthroughFilter* to allow filtering where to schedule a GPU workload

Virtual machine

OPENSTACK NOVA GPU PASSTHROUGH

LAUNCHING OPENSTACK INSTANCES WITH GPU / PCI PASSTHROUGH

- Create OpenStack Flavor(s)
  - Custom metadata: *"pci_passthrough:alias"="W5500:1"*
    - The :1 determines a single GPU should be passed through
- The corresponding drivers of the passed through GPU are required
  - No GPU drivers present in Linux cloud-based images
  - Corresponding CUDA and/or openCL libraries also essential
- Manual GPU driver install
  - *AMDGPU* Linux drivers for our Radeon Pro W5500 obsoleted …
  - *AMD's Radeon Open Compute* (ROCm) drivers do work (but bloated)
- **Success!** (*)
  - Validated with OpenCL workload



OPENSTACK NOVA GPU PASSTHROUGH

INITIAL FINDINGS ON OPENSTACK GPU PASSTHROUGH

OpenStack Nova (libvirt) GPU passthrough works on the first attempt

However, failure on all subsequent attempts

- Known bug in AMD firmware and/or drivers bricks the GPU at VM shutdown
  - The GPU is not correctly powered down / reset / re-initialized
- For now, only solution is to reboot the physical machine
  - Absolutely unacceptable for cloud infrastructure
- github.com/gnif/vendor-reset might provide a future solution
  - Tested but no success; AMD Radeon Pro W5500 is currently not supported

Note: This is not an OpenStack related issue!

OPENSTACK NOVA GPU PASSTHROUGH

CONCLUSION ON OUR GPU PASSTHROUGH FINDINGS

- Successful GPU passthrough with OpenStack Nova / libvirt

- Game breaking reset bug on the AMD Radeon Pro W5500
  - Unusable for cloud-based VM PCI passthrough in its current state
  - No issues for Kubernetes container-based “GPU passthrough”
    → Host always remains in control of GPU, no reinitializations/resets

- Future work: Nvidia Quadro RTX 4000
  - Will be validated using a different server or workstation

AGENDA

- Project announcements
- Virtualized and Containerized workloads
- Deploying OpenStack infrastructure with Kolla-Ansible
- GPGPU and initial findings on OpenStack GPU passthrough
- **Automating cloud image creation with Packer**
- Terraform as cloud-agnostic way of specifying Infrastructure as Code
- Serverless computing and Lightweight virtualization with OpenFAAS
- Conclusions, Future Work and Q&A

AUTOMATING VM IMAGE GENERATION

HASHICORP PACKER AS MULTI-PLATFORM IMAGE GENERATION TOOL

HashiCorp Packer

- Cloud deployments benefit from pre-baked images (e.g. with integrated GPU drivers)

- Creating & modifying VM/container images is a repetitive and time-consuming task
- There is a need for automated image creation
  - e.g. Triggered each time a new GIT commit is successfully built and ready for deployment (CI/CD)

- **Hashicorp packer** is a multi-platform tool which provides this automation
  - Packer binary
  - Image build template in *Hashicorp Configuration Language (HCL)*
    - Builders EC2, Azure, Google Cloud, DigitalOcean, Docker, **OpenStack**, QEMU, VMware, …
    - Provisioners File, Shell, Ansible, Chef, Puppet, …

AUTOMATING VM IMAGE GENERATION

PACKER IMAGE BUILD SCRIPT FOR CENTOS WITH AMD W5500 GPU DRIVERS

HashiCorp Packer

```
{
  "builders": [
    {
      "type": "openstack",
      "image_name": "CentOS-8.3-GPU",
      "source_image_name": "CentOS-8.3",
      "flavor": "m1.gpu",
      "ssh_username": "centos",
      "floating_ip_network": "d3871423-5949-4aea-8b20-9a7bc5f2dbbd"
    }
  ],
  "provisioners": [
    {
      "type": "file",
      "source": "amdgpu-pro-20.10-1101037-rhel-8.1.tar.xz",
      "destination": "~/amdgpu-pro-20.10-1101037-rhel-8.1.tar.xz"
    },
    {
      "type": "shell",
      "inline": [
                "tar -Jxvf amdgpu-pro-20.10-1101037-rhel-8.1.tar.xz",
                "sudo dnf install -y epel-release",
                "sudo dnf update -y",
                "sudo reboot"
                ],
      "expect_disconnect": true,
      "pause_after": "30s"
    },
    {
```

OpenStack environment variables already loaded

```
packer build centos-gpu.hcl
```

```
    openstack: Verified flavor. ID: 2
==> openstack: Creating temporary keypair: packer_601afe41-1c58-176c-0aab-ee8fa3eeaf9d ...
==> openstack: Created temporary keypair: packer_601afe41-1c58-176c-0aab-ee8fa3eeaf9d
    openstack: Found Image ID: 0cd0a0a5-efaa-4a6a-bf2c-cb049fce055c
==> openstack: Launching server...
==> openstack: Launching server...
    openstack: Server ID: 0912c28b-a2ab-4216-bae0-f02960897805
==> openstack: Waiting for server to become ready...
==> openstack: Creating floating IP using network d3871423-5949-4aea-8b20-9a7bc5f2dbbd ...
    openstack: Created floating IP: '729fbcca-d90e-458d-aefc-7bfe50348ec7' (10.20.28.206)
==> openstack: Associating floating IP '729fbcca-d90e-458d-aefc-7bfe50348ec7' (10.20.28.206) with inst
    openstack: Added floating IP '729fbcca-d90e-458d-aefc-7bfe50348ec7' (10.20.28.206) to instance!
==> openstack: Using ssh communicator to connect: 10.20.28.206
==> openstack: Waiting for SSH to become available...
==> openstack: Connected to SSH!
==> openstack: Uploading amdgpu-pro-20.10-1101037-rhel-8.1.tar.xz => ~/amdgpu-pro-20.10-1101037-rhel-
==> openstack: Provisioning with shell script: C:\Users\Steffen\AppData\Local\Temp\packer-shell703998
    openstack: amdgpu-pro-20.10-1101037-rhel-8.1/
    openstack: amdgpu-pro-20.10-1101037-rhel-8.1/repodata/
    openstack: amdgpu-pro-20.10-1101037-rhel-8.1/repodata/76fc93348700373562db7c110acff1fb0ac624d6bbc
```

AGENDA

- Project announcements
- Virtualized and Containerized workloads
- Deploying OpenStack infrastructure with Kolla-Ansible
- GPGPU and initial findings on OpenStack GPU passthrough
- Automating cloud image creation with Packer
- **Terraform as cloud-agnostic way of specifying Infrastructure as Code**
- Serverless computing and Lightweight virtualization with OpenFAAS
- Conclusions, Future Work and Q&A

CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

What is Terraform?

HashiCorp Terraform

Write, Plan, and Create Infrastructure as Code

- Opensource Infrastructure-as-Code software tool
- Efficient deployment, management and automation
- Compatible with 500+ providers (public/private clouds, network appliances, Platform as a Service, Software as a Service)
- Constantly updated and supported by Hashicorp and the community
- Part of Hashicorp's “Cloud-oriented” suite of tools

CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

Why Terraform?

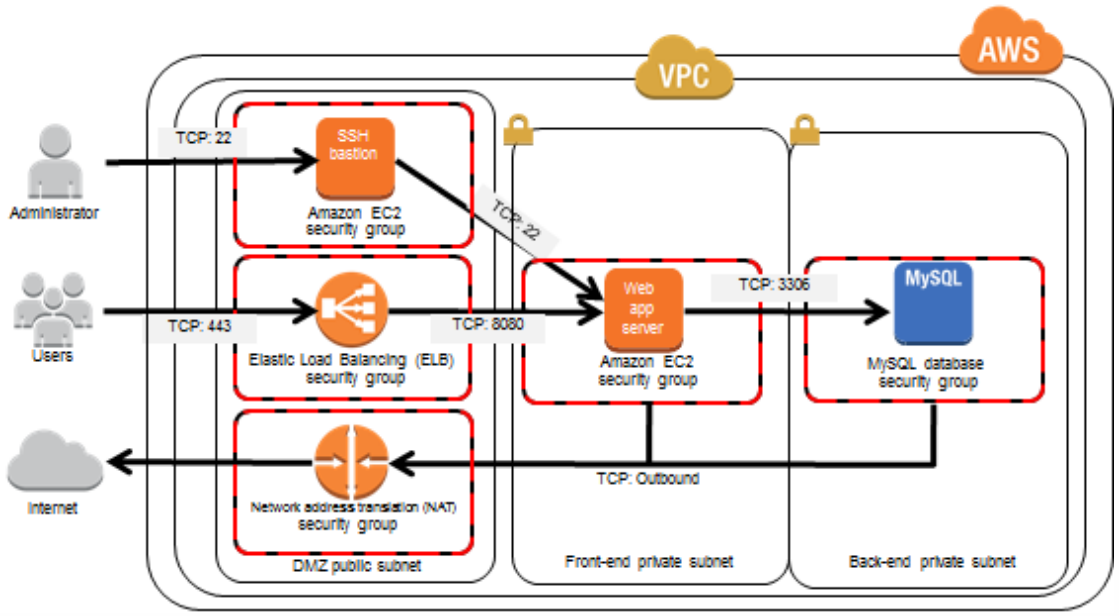
HashiCorp Terraform

Write, Plan, and Create Infrastructure as Code

- One tool to manage any resource, regardless of where
- Excellent to handle multi-cloud / hybrid cloud scenarios, but not only
- Deploy, manage and update with Infrastructure as Code

CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

INFRASTRUCTURE-AS-CODE (IaC)



Infrastructure example

```
provider "aws" {
  region = "us-west-2"
}

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.21.0"

  name = var.vpc_name
  cidr = var.vpc_cidr

  azs             = var.vpc_azs
  private_subnets = var.vpc_private_subnets
  public_subnets  = var.vpc_public_subnets

  enable_nat_gateway = var.vpc_enable_nat_gateway

  tags = var.vpc_tags
}

module "ec2_instances" {
  source  = "terraform-aws-modules/ec2-instance/aws"
  version = "2.12.0"

  name           = "my-ec2-cluster"
  instance_count = 2

  ami                    = "ami-0c5204531f799e0c6"
  instance_type          = "t2.micro"
  vpc_security_group_ids = [module.vpc.default_security_group_id]
  subnet_id              = module.vpc.public_subnets[0]

  tags = {
    Terraform   = "true"
    Environment = "dev"
  }
}

module "website_s3_bucket" {
  source = "./modules/aws-s3-static-website-bucket"

  bucket_name = "<UNIQUE BUCKET NAME>"

  tags = {
    Terraform   = "true"
    Environment = "dev"
  }
```

Configuration file example

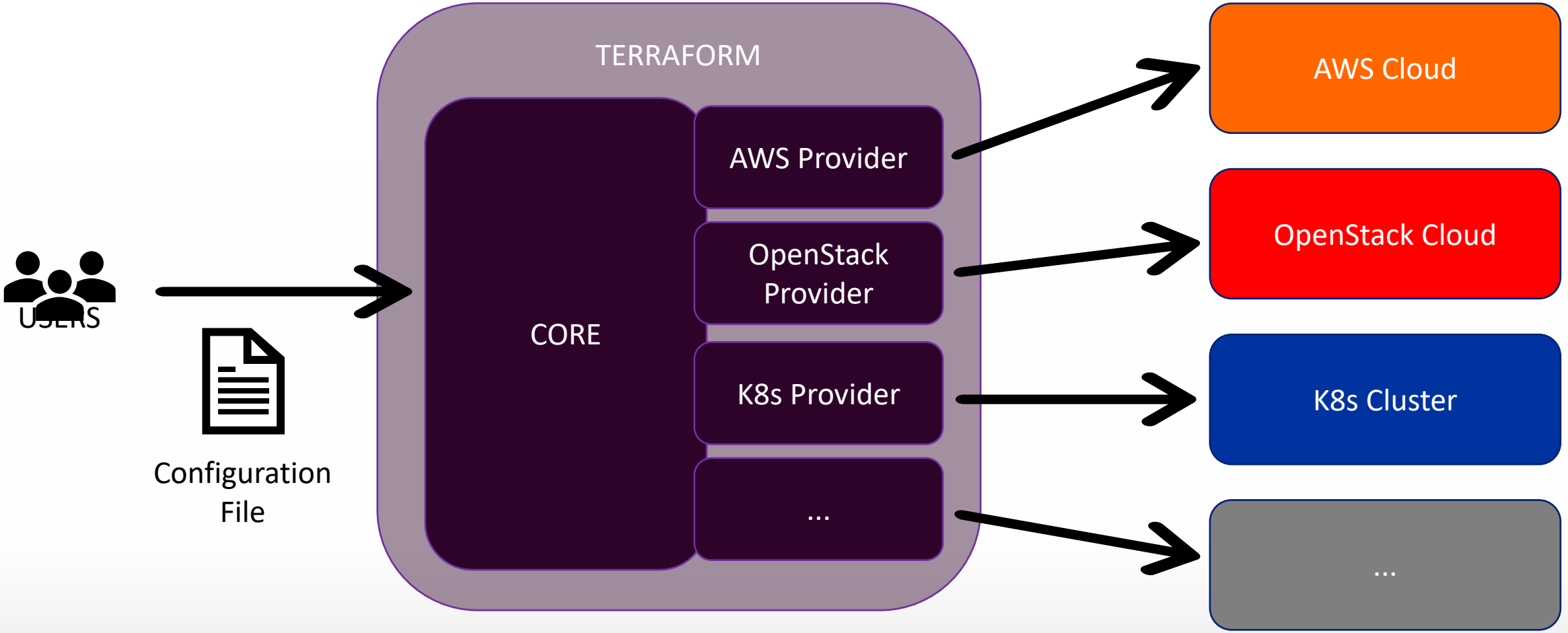
CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

Hashicorp Configuration Language (HCL)

- Makes Terraform "cloud-agnostic"
- Easy to learn
- Same language for most of the Hashicorp tools
- Tools to convert other languages into HCL (json, java, Typescript)

CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

Terraform Architecture: Core and providers



CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

Terraform core: Main commands

- **INIT**

Initialize Terraform and look for providers

- **PLAN**

Overview of what to execute to realize what is described in the configuration files

- **APPLY**

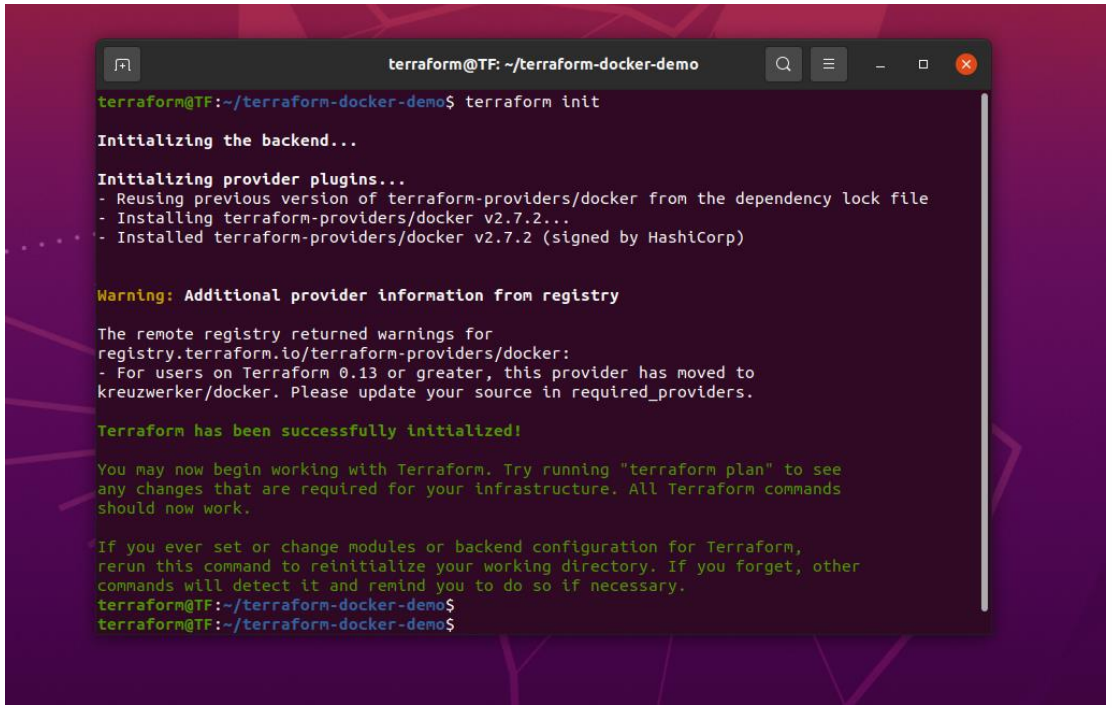
Perform the operations as planned

- **DESTROY**

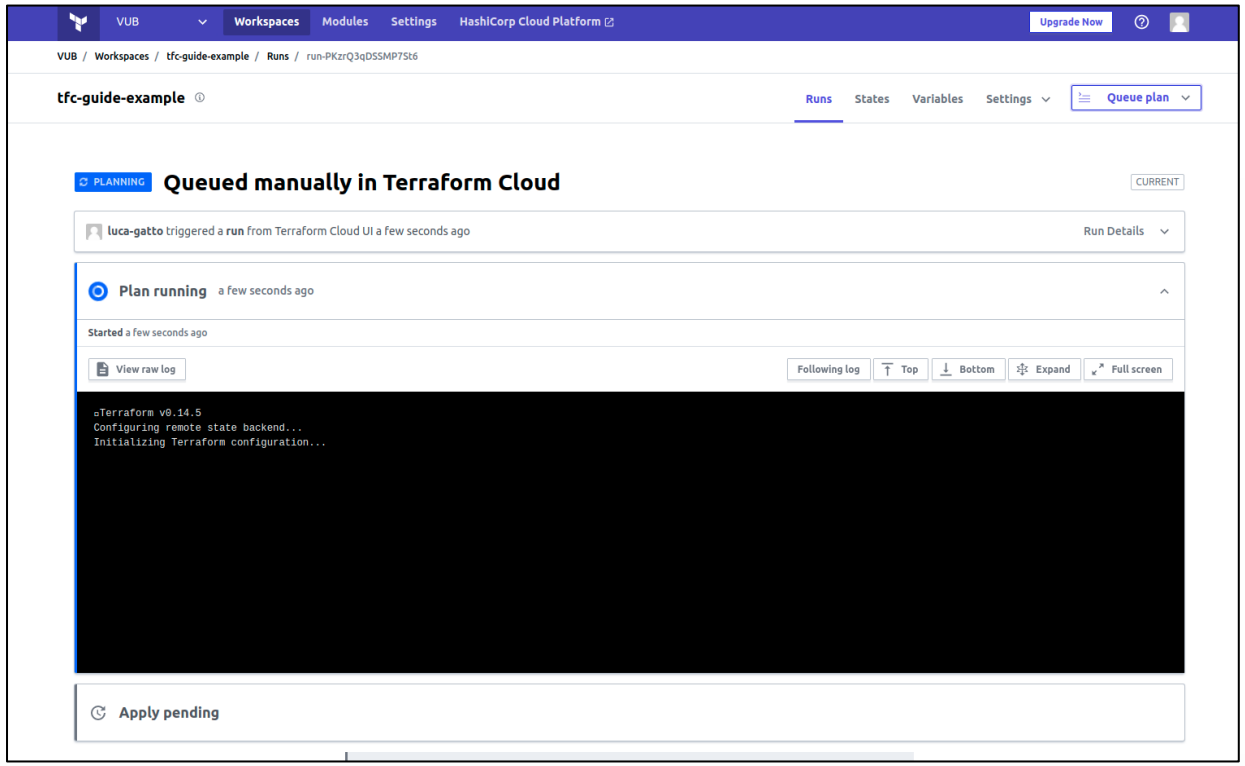
Deallocate and destroy all the resources

CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

Terraform CLA and Cloud Application: Two different approaches to launch Terraform commands



Terraform Command Line Interface (CLI)

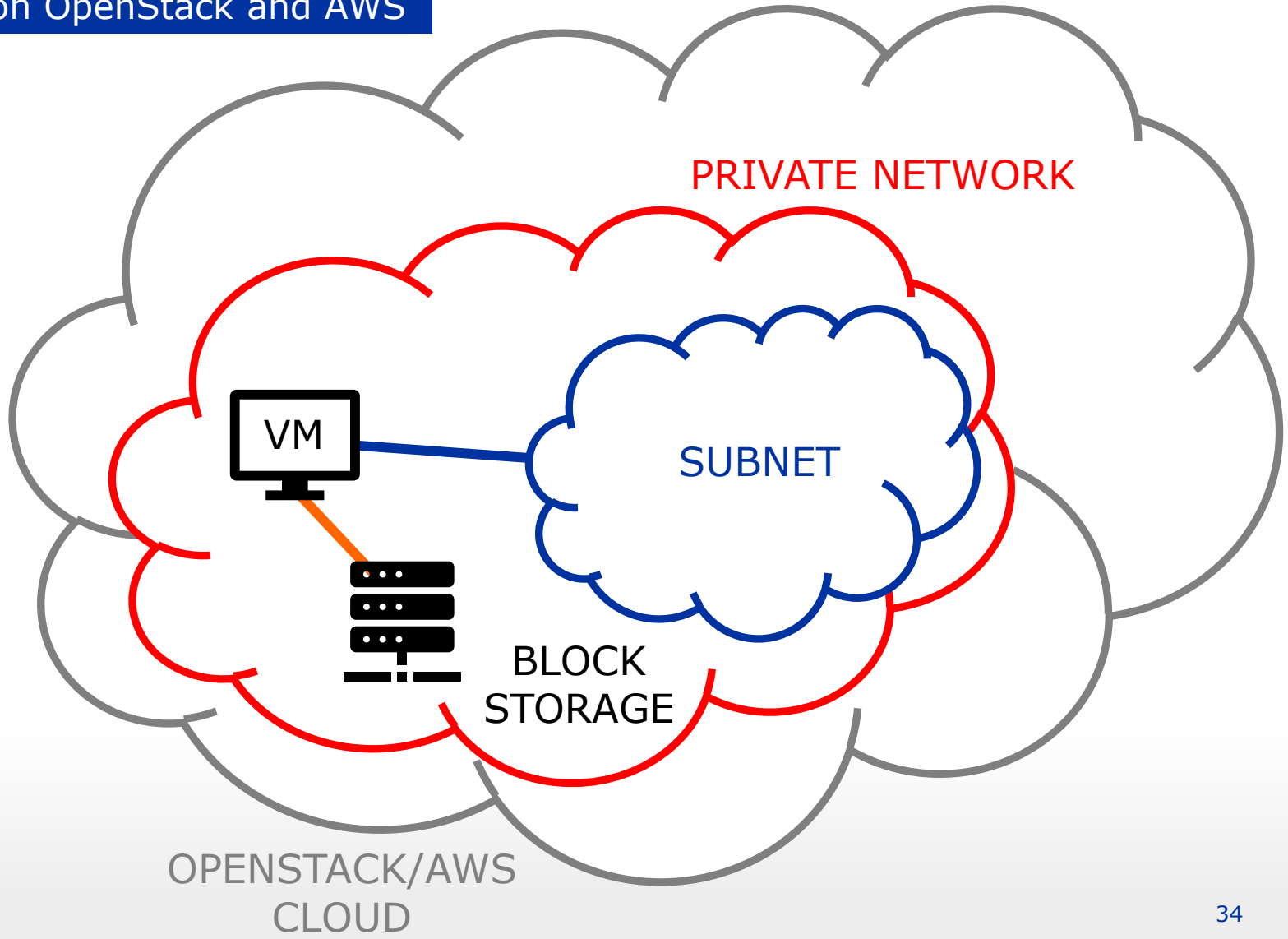


Terraform Cloud

CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

Deploy a simple infrastructure on OpenStack and AWS

1. Create a private network
2. Define a subnet
3. Run a VM
4. Connect it to the subnet
5. Create a block storage
6. Attach it to the VM



CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

OpenStack configuration file

```
terraform {
  required_providers {
    openstack = {
      source = "terraform-provider-openstack/openstack"
    }
  }
}

provider "openstack" {}

resource "openstack_networking_network_v2" "tf_network" {
  name = "tf_network"
}


resource "openstack_networking_subnet_v2" "tf_subnet" {
  name = "tf_subnet"
  network_id = openstack_networking_network_v2.tf_network.id
  cidr = "192.168.150.0/24"
}


resource "openstack_compute_instance_v2" "tf_instance" {
  name = "tf_instance"
  image_name = "cirros"
  flavor_name = "m1.tiny"
  network {name = openstack_networking_network_v2.tf_network.name}
}


resource "openstack_blockstorage_volume_v2" "tf_volume" {
  name = "tf_volume"
  size = 1
}


resource "openstack_compute_volume_attach_v2" "tf_attach" {
  instance_id = openstack_compute_instance_v2.tf_instance.id
  volume_id = openstack_blockstorage_volume_v2.tf_volume.id
}
```

AWS configuration file

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
    }
  }
}

provider "aws" {region = "eu-west-3"}

resource "aws_vpc" "tf_network" {
  tags = {Name = "tf_network"}
  cidr_block = "192.168.0.0/16"
}

resource "aws_subnet" "tf_subnet" {
  tags = {Name = "tf_subnet"}
  vpc_id = aws_vpc.tf_network.id
  cidr_block = "192.168.150.0/24"
  availability_zone = "eu-west-3a"
}

resource "aws_instance" "tf_instance" {
  tags = {Name = "tf_instance"}
  ami = "ami-0ea4a063871686f37"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.tf_subnet.id
  availability_zone = "eu-west-3a"
}

resource "aws_ebs_volume" "tf_volume" {
  tags = {Name = "tf_volume"}
  size = 1
  availability_zone = "eu-west-3a"
}

resource "aws_volume_attachment" "tf_attach" {
  volume_id = aws_ebs_volume.tf_volume.id
  instance_id = aws_instance.tf_instance.id
  device_name = "/dev/sdh"
}
```

CLOUD-AGNOSTIC INFRASTRUCTURA AS CODE WITH TERRAFORM

Next steps

- Terraform for our infrastructure and future growth (Hybrid cloud/Multi-cloud)
- Consul, Vault, Nomad (to be integrated with Terraform)
- Use cases

AGENDA

- Project announcements
- Virtualized and Containerized workloads
- Deploying OpenStack infrastructure with Kolla-Ansible
- GPGPU and initial findings on OpenStack GPU passthrough
- Automating cloud image creation with Packer
- Terraform as cloud-agnostic way of specifying Infrastructure as Code
- **Serverless computing and Lightweight virtualization with OpenFAAS**
- Conclusions, Future Work and Q&A

SERVERLESS COMPUTING WITH OPENFAAS

This topic is provided in a separate presentation

AGENDA

- Project announcements
- Virtualized and Containerized workloads
- GPGPU and initial findings on OpenStack GPU passthrough
- Initial findings on OpenStack GPU passthrough
- Automating cloud image creation with Packer
- Terraform as cloud-agnostic way of specifying Infrastructure as Code
- Serverless computing and Lightweight virtualization with OpenFAAS
- **Conclusions, Future Work and Q&A**

CONCLUSIONS, FUTURE WORK AND Q&A

- Our on-premise OpenStack and Kubernetes infrastructures are operational & maturing
  - Currently working on providing & consuming GPU accelerated features
- Hybrid/multi-cloud deployments using Terraform
  - This will be further extended into a connected hybrid cloud solution
- Still determining our roadmap regarding edge computing
  - We consider the edge as devices with restricted resources (e.g. Raspberry Pi as IoT gateway)
  - Likely will feature a ‘lightweight’ containerized (Kubernetes) or serverless (openFaaS) solution
- We are always open to suggestions regarding use-cases and desired future work