Terraforming the Cloud to Teach HPC
Félix-Antoine Fortin

Principal developer of Magic Castle
Université Laval
Calcul Québec
Digital Research Alliance of Canada

he/him
@cmd-ntrf
Terraforming the Cloud to Teach HPC

- Why Cloud to teach HPC?
- Overview of existing HPC in the cloud tools
- Introduction to Magic Castle
- Magic Castle in the Wild
Why Cloud to teach HPC?
macro
Advanced Research Computing (ARC)
Research infrastructure landscape in Canada
Advanced Research Computing (ARC)
Research infrastructure landscape in Canada

150 workshops / year

95+ % usage

How to train users at scale without impacting research?
micro
Research support staff development

Over 200 research support staff from various scientific and engineering backgrounds, but less and less system administrators by trade.

How to provide staff inexpensive parallel computing cluster to experiment and learn?
Why are there more wizards in Harry Potter than in Lord of the Rings?
Why are there more wizards in Harry Potter than in Lord of the Rings?
Traditional Data Centre

On-Demand Infrastructure
We want accessible, inexpensive sandbox environments, designed to facilitate teaching and experimentation.
macro controlled replicable teaching environments to audiences of variable sizes
micro
inexpensive sandboxes for individuals to experiment and learn at their own pace.
Cloud can provide the building blocks for both scale: macro and micro.
What are the existing tools?
<table>
<thead>
<tr>
<th>Name</th>
<th>Creator</th>
<th>First public release date</th>
<th>Software license</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS ParallelCluster</td>
<td>AWS</td>
<td>November 12, 2018</td>
<td>Apache v2</td>
</tr>
<tr>
<td>Azure CycleCloud</td>
<td>Microsoft</td>
<td>October 17, 2018</td>
<td>MIT</td>
</tr>
<tr>
<td>Azure HPC On-Demand</td>
<td>Microsoft</td>
<td>April 23, 2021</td>
<td>MIT</td>
</tr>
<tr>
<td>Google HPC-Toolkit</td>
<td>Google</td>
<td>May 26, 2022</td>
<td>Apache v2</td>
</tr>
<tr>
<td>Slurm on GCP</td>
<td>SchedMD</td>
<td>March 14, 2018</td>
<td>Apache v2</td>
</tr>
</tbody>
</table>
## Multi-cloud

<table>
<thead>
<tr>
<th>Name</th>
<th>Creator</th>
<th>First public release date</th>
<th>Software license</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster in the Cloud</td>
<td>Matt Williams - University of Bristol</td>
<td>March 27, 2019</td>
<td>MIT</td>
</tr>
<tr>
<td>ElastiCluster</td>
<td>Riccardo Murri - University of Zurich</td>
<td>July 17, 2013</td>
<td>GPLv3</td>
</tr>
</tbody>
</table>
## Multi-cloud: supported providers

<table>
<thead>
<tr>
<th>Name</th>
<th>Alibaba Cloud</th>
<th>AWS</th>
<th>Azure</th>
<th>Google Cloud</th>
<th>Open Stack</th>
<th>Oracle Cloud</th>
<th>OVH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster-in-the-Cloud</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>ElastiCluster</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>-</td>
</tr>
</tbody>
</table>
## Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Infrastructure definition</th>
<th>Configuration management</th>
<th>Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS ParallelCluster</td>
<td>CLI generating YAML</td>
<td>Chef</td>
<td>Slurm</td>
</tr>
<tr>
<td>Azure CycleCloud</td>
<td>WebUI or CLI + templates</td>
<td>Chef</td>
<td>many</td>
</tr>
<tr>
<td>Azure HPC On-Demand</td>
<td>YAML files + shell scripts</td>
<td>Ansible, Packer</td>
<td>Open PBS, Slurm</td>
</tr>
<tr>
<td>Cluster in the Cloud</td>
<td>CLI generating TF code</td>
<td>Ansible, Packer</td>
<td>Slurm</td>
</tr>
<tr>
<td>ElastiCluster</td>
<td>CLI interpreting an INI file</td>
<td>Ansible</td>
<td>Grid Engine, Slurm</td>
</tr>
<tr>
<td>Google HPC-Toolkit</td>
<td>CLI generating TF code</td>
<td>Ansible, Packer</td>
<td>Slurm</td>
</tr>
<tr>
<td>Slurm GCP</td>
<td>Terraform modules</td>
<td>Ansible, Packer</td>
<td>Slurm</td>
</tr>
</tbody>
</table>
Why proposing another tool?

1. We wanted an open source multi-cloud solution that included OpenStack as a first class citizen.
2. We wanted Puppet to be the configuration management tool. Regional partners are Puppet-shops or at least familiar with it.
3. All cloud API interactions would have to be done by a third-party tool. No homemade CLI or wrapper.
Designing an accessible tool for learning HPC

- Focus on re-creating the HPC environment
- Provide an accessible experience for beginners, with minimal prior HPC knowledge required
- Include key HPC features: job scheduling, data transfer, parallel and distributed computation, GPU, etc.
- Require minimal knowledge of cloud and minimal cost
- It should take a few minutes to setup a sandbox.
Introduction to Magic Castle
Open source infrastructure-as-code aiming to reproduce the HPC user experience in the cloud
Design choices

- **Infrastructure**: 100% Terraform - no CLI or wrapper
  - A single interface to interact with all major cloud providers
- **Configuration**: cloud-init and Puppet
  - No knowledge of Puppet is required. The agent is autonomous.
- **Scheduler**: Slurm
  - Main scheduler used by the Alliance in Canada.
- **Cloud providers**: AWS, Azure, Google, OpenStack, OVH
  - Other providers can be added by following the documentation
Design choices

- Spawn instances: management, login, compute, dtn, proxy, etc.
- Create volumes, network, network acls
- Create dns records
- Bootstrap passwords, certificates, secrets, keys, etc.
- Scale compute resources automatically based on job queue
- Customization via input parameters and YAML file

[github.com/computecanada/magic_castle]
configure
Over 3000 scientific software are one “module load” away thanks to

Users can also install software using

Spack
1k+ workshops and university courses have used Magic Castle to teach advanced research computing since its initial release in 2018.
How does it work?
What is Terraform?

Terraform is an infrastructure-as-code software tool created by HashiCorp. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL).
How does it work?

How does it work?

resource "openstack_compute_instance_v2" "mgmt01" {
  name            = "mgmt01"
  flavor_id       = "p4-6gb"
  key_pair        = "ssh-ed25519 ..."
  security_groups = ["default"]

  block_device {
    image_name            = "Rocky-8"
    source_type           = "image"
    volume_size           = "50"
    boot_index            = 0
    destination_type      = "volume"
    delete_on_termination = true
  }
}
# IaC to create a Kubernetes cluster in GCP

```hcl
module "gke" {
  source      = "...
  project_id  = "<PROJECT ID>"
  name        = "gke-test-1"
  region      = "us-central1"
  zones       = ["us-central1-a"]
  network     = "vpc-01"
  http_load_balancing = false
  ...
}
```
How does it work?

```
$ terraform plan
$ terraform apply
Terraform will perform the following actions:
...
Do you want to perform these actions?
Enter a value: yes
```
How useful is Terraform?

Terraform supports a number of cloud infrastructure providers such as Amazon Web Services, Cloudflare, Microsoft Azure, IBM Cloud, Serverspace, Google Cloud Platform, DigitalOcean, and OpenStack.

Combined with its ability to build infrastructure using high level building blocks, Terraform is an excellent choice for building complex environment like HPC clusters in the cloud.
Installing Terraform

Terraform can be installed easily on all platforms as it is a single standalone Go binary.

You can download it from here:

https://developer.hashicorp.com/terraform/downloads
infrastructure-as-code ⇒
The infrastructure is defined in a main Terraform module. Each cloud provider has its dedicated main module:
The main modules share common inputs:

```
common/variables.tf
```

- AWS
- Azure
- Google Cloud
- OpenStack
- OVHcloud
And common outputs:

- `common/variables.tf`
- `common/outputs.tf`

Diagram:

- `aws`
- `Azure`
- `Google Cloud`
- `OpenStack`
- `OVHcloud`

These components are connected to:

- `common/variables.tf`
- `common/outputs.tf`
Each main module uses 3 common sub-modules:

- `common/variables.tf`
- `common/design`
- `common/configuration`
- `common/provision`
**design** sub-module transforms the inputs into **maps** used to generate the resources specific to each provider:

- `var.cluster_name`
- `var.domain`
- `var.instances`
- `var.volumes`
- `out.instances`
- `out.volumes`
configuration sub-module creates the cloud-config file (user_data). This file configures SSH access and bootstraps Puppet on first boot.
#cloud-config

run cmd:

- yum -y upgrade -x puppet*

{% if contains(tags, "puppet") %}
- yum -y install puppetserver
- systemctl enable puppetserver
- git clone ${puppetenv_git} /etc/puppetlabs/code/environments/production
{% endif %}

- yum -y install puppet-agent
- /opt/puppetlabs/bin/puppet config set certname ${node_name}
- /opt/puppetlabs/bin/puppet config set waitforcert 15s

users:

- name: ${sudoer_username}
  ssh authorized keys:

{% for key in ssh authorized keys %}
  - ${key}
{% endfor %}

provision copies the state (instances, #cpus, #gpus, volumes, etc.) via SSH to the Puppet server as a YAML file (terraform_data.yaml).

```
var.instances
...
var.hieradata
```

```
common/provision
provisioner file
remote-exec
```
"node4":
  "hostkeys":
    "ed25519": ssh-ed25519 ...
    "rsa": ssh-rsa ...
  "id": "droid-node4"
  "local_ip": "10.0.0.11"
  "public_ip": ""
  "specs": { "cpus": "2", "gpus": 0, "ram": "8000" }
  "tags": ["node", "pool"]
This set of common submodules creates an easy to use interface without vendor lock-in.
source         = "./aws"
config_git_url = "https://github.com/ComputeCanada/puppet-magic_castle.git"
config_version = "13.0.0"

cluster_name   = "phoenix"
domain         = "your-domain-name.cloud"
image          = "ami-09ada793ee1559e6"

instances = {
    mgmt  = {
        type = "t3.medium",
        count = 1,
        tags = ["mgmt", "puppet", "nfs"]
    },
    login = {
        type = "t3.medium",
        count = 1,
        tags = ["login", "public", "proxy"]
    },
    node  = {
        type = "t3.medium",
        count = 50,
        tags = ["node", "pool"]
    }
}

volumes = {
    nfs = {
        home   = { size = 100 }
        project = { size = 500 }
        scratch = { size = 500 }
    }
}

source = "./gcp"
config_git_url = "https://github.com/ComputeCanada/puppet-magic_castle.git"
config_version = "13.0.0"

cluster_name = "phoenix"
domain = "your-domain-name.cloud"
image = "rocky-8-gcp-optimized"

instances = {
    mgmt = { type = "n2-standard-2", count = 1, tags = ["mgmt", "puppet", "nfs"] },
    login = { type = "n2-standard-2", count = 1, tags = ["login", "public", "proxy"] },
    node = { type = "n2-standard-2", count = 50, tags = ["node", "pool"] }
}

volumes = {
    nfs = {
        home = { size = 100 }
        project = { size = 500 }
        scratch = { size = 500 }
    }
}

source = "../gcp"
config_git_url = "https://github.com/ComputeCanada/puppet-magic_castle.git"
config_version = "13.0.0"

cluster_name = "phoenix"
domain = "your-domain-name.cloud"
image = "rocky-8-gcp-optimized"

instances = {
    mgmt = { type = "n2-standard-2", count = 1, tags = ["mgmt", "puppet", "nfs"] },
    login = { type = "n2-standard-2", count = 1, tags = ["login", "public", "proxy"] },
    node = { type = "n2-standard-2", count = 50, tags = ["node", "pool"] }
}

volumes = {
    nfs = {
        home = { size = 100 },
        project = { size = 500 },
        scratch = { size = 500 }
    }
}

The roles of each instance are defined by tags
Terraform

terraform_data.yaml

puppet

terraform_data.yaml
Puppet manages the configuration
Puppet configuration customization: YAML

- Magic Castle configuration is done entirely through Puppet classes.
- There are over 40 classes that can be customized.
- Customization can happen before a cluster is launched or after.

```yaml
---
profile::users::ldap::users
  alice
    groups: ['engineering']
    public_keys: ['ssh-rsa ... user@local' 'ssh-ed25519 ...']
profile::fail2ban::ignoreip
  132.203.0.0/16
```
Scaling & Autoscaling
Scaling

instances = {
    mgmt = {
        type = "n2-standard-2",
        count = 1,
        tags = ["mgmt", "puppet", "nfs"]
    },
    login = {
        type = "n2-standard-2",
        count = 1,
        tags = ["login", "public", "proxy"]
    },
    node = {
        type = "n2-standard-2",
        count = 3,
        tags = ["node"]
    }
}
Autoscaling

```
instances = {
    mgmt = {
        type = "n2-standard-2"
        count = 1
        tags = ["mgmt", "puppet", "nfs"]
    },
    login = {
        type = "n2-standard-2"
        count = 1
        tags = ["login", "public", "proxy"]
    },
    node = {
        type = "n2-standard-2",
        count = 3,
        tags = ["node", "pool"]
    }
}
```
Autoscaling: resume
Autoscaling: suspend
What is Terraform Cloud?

Terraform Cloud manages Terraform runs in a consistent and reliable environment, and includes easy access to shared state and secret data, access controls for approving changes to infrastructure.

Teams can connect Terraform to version control, share variables, run Terraform in a remote environment, and securely store remote state.

Terraform Cloud is available as a hosted service at https://app.terraform.io.
1. Initialize a new git repository on GitLab or GitHub with the Magic Castle release for your cloud provider
2. Add `data.yaml` to the repo, you will use this with `main.tf` to define your cluster
3. Link the repo with a Terraform cloud workspace
4. Configure credentials for your providers, and a workspace variable: `pool = []`
5. Define the workspace ID and an API token in `data.yaml`
6. Launch the run execution in Terraform Cloud

See autoscaling documentation
The autoscaling logic is *cloud-agnostic* and is expressed in 100 lines of Python.

The API token requires only 2 permissions: modify a variable and create a plan.

The compute nodes can be heterogeneous (GPU, x86, ARM64). Slurm determines which nodes to power-up based on the job queue.
In the wild
Calcul Québec

- Uses Magic Castle as the hands-on exercise platform for their entire 2023-2024 training program

- Provides and administers Magic Castle clusters to graduate courses from various disciplines: AI, bioinformatics, neuroscience
uses Magic Castle as its platform to compile and test software built with EasyBuild before deploying them on CVMFS

https://www.eessi.io/
Magic Castle is integrated in CACAO and can be launched easily in Jetstream cloud.

https://docs.jetstream-cloud.org/ui/cacao/deployment_magic_castle/
Simple to use
Autoscaling infrastructure
Ideal software environment
to teach and learn HPC
cloud-agnostic and open source

https://www.github.com/computecanada/magic_castle