Towards “Plug and Play” Research Cloud Node Deployments

Timothy Carr
University of Cape Town
Technical Specialist

cat /proc/timinfo

- Technical Specialist @ University of Cape Town for 17 years.
- Focused on the building and delivery of enterprise and research ICT applications to the community.
- Two demanding boys.
- Amazing supportive wife.

Agenda

- The current problem
- Possible solution
- What is IAC ?
- Provide me with the tools.
- Building consistency into my environment.
- Questions

The current problem

- Shared research user environments pose a few challenges
  - Unresponsive compute nodes
  - Non-existant or limited privilege escalation - (Docker, Reboot Node)
- Manually maintaining and scaling compute stacks.
- What about configuration drift.
- Manual operating system / application patch cycles.

Possible Solution

- Research Purpose Built Environments
  - Deployed and defined with IAC.
  - Self-maintained with dynamic updates, scale the infrastructure on demand.
  - Block storage on-demand (BeeGFS) or (CephFS)
  - Pre-defined or custom virtual machine images (private glance image repo)
  - Software no longer resides in the virtual machine images (light-weight deploy) but in containers ie: Docker or Singularity.
  - Allows for privilege escalation, researcher is able to reboot unresponsive nodes.

What is IAC ?

- IAC (infrastructure as code) is defined as just that, “building/deploying/scaling infrastructure with code” against a set of cloud API endpoints.
- These endpoints are public cloud endpoints from cloud providers such as GCP, AWS, Rackspace, Digital Ocean, iLifu.
- IAC can maintain small compute stacks initiated for a specific purpose to managing the deployment of entire data centers.

Provide me with the tools

Cloud Native Landscape

v0.9.9

App Definition & Development

Database & Data Analytics

Streaming

SCM

Application Definition

CI/CD

Orchestration & Management

Scheduling & Orchestration

kubernetes CNCF Project

Coordination & Service Discovery

CoreDNS CNCF Project

Service Management

linkerd CNCF Project

gRPC CNCF Project

envoy CNCF Project

Runtime

Cloud-Native Storage

Container Runtime

containerd CNCF Project

rkt CNCF Project

Cloud-Native Network

CNI CNCF Project

Provisioning

Host Management / Tooling

Infrastructure Automation

Container Registries

Secure Images

Notary CNCF Project

TUF CNCF Project

Key Management

Cloud

Public

Private

Platforms

PaaS / Container Service

Serverless/Event-based

Observability & Analysis

Monitoring

Prometheus CNCF Project

Logging

fluentd CNCF Project

Tracing

OpenTracing CNCF Project

Jaeger CNCF Project

github.com/cncf/landscape

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

CLOUD NATIVE COMPUTING FOUNDATION

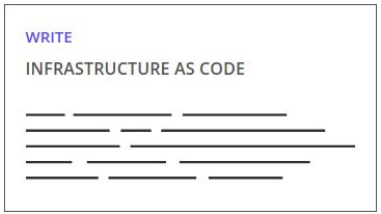
Redpoint

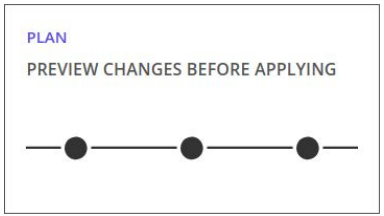
Amplify PARTNERS

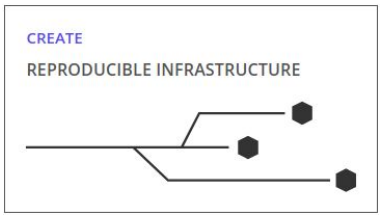
seriously…... provide me with some realistic tooling

- **Hashicorp Terraform.io**: IAC tool to maintain infrastructure.
- **Hashicorp Packer.io**: Build / maintain virtual machine images to be consistent / immutable.
- **Configuration Mgmt Tools:** Ansible, Puppet, Chef, Saltstack …....
- **Openstack Heat**: Similar to Terraform but with integration points to other Openstack services. Has the autoscaling templates built-in for reference to “Openstack Telemetry”.

Tool: Terraform.io







- Declarative configuration: User *declares* exactly which portion of the infrastructure needs to change.
- Execution Plan: A “plan” will display the output of what will be committed to the environment. Consider it something similar to a “dry-run”.
  - **“$terraform plan -out plan.file”**
- Change Automation: Terraform drafts a resource graph which will display which components will be changed in which order.
- Create: Apply the infrastructure changes to the environment.
  - **“$terraform apply plan.file”**
- Destroy: **“$terraform destroy plan.file”**

Code Snips

provider.tf

```
# Configure the OpenStack Provider
provider "openstack" {
  user_name   = "username@uct.ac.za"
  tenant_name = "Tim Project"
  password    = "#######"
  auth_url    = "https://api.ilifu.ac.za:5000/"
  region      = "UCT-ICTS-DC"
}
```

security.tf

```
resource "openstack_compute_secgroup_v2" "worker-icmp" {
  name        = "worker-icmp"
  description = "Worker ICMP"
  rule {
    from_port = -1
    to_port = -1
    ip_protocol = "icmp"
    cidr = "10.0.0.0/16"
  }
}
```

floating_ip.tf

```
# Request floating ip
resource "openstack_networking_floatingip_v2" "headnode" {
  pool        = "${var.external_network}"
}

# Attach floating ip to instance
resource "openstack_compute_floatingip_associate_v2" "headnode" {
  floating_ip = "${openstack_networking_floatingip_v2.headnode.address}"
  instance_id = "${openstack_compute_instance_v2.headnode.id}"
}
```

instances.tf

```
# Create Meerkat SLURM-headnode instance
resource "openstack_compute_instance_v2" "headnode" {
  name            = "${var.head_name}"
  image_name      = "${var.image}"
  flavor_name     = "${var.flavor_headnode}"
  key_pair        = "${openstack_compute_keypair_v2.user_key.name}"
  security_groups = ["${openstack_compute_secgroup_v2.ssh.id}"]
  user_data       = "${file("scripts/deploy.sh")}"
  network {
    uuid          = "${openstack_networking_network_v2.private_network.id}"
  }
  # Install system in volume
  block_device {
    volume_size           = "${var.volume_headnode}"
    destination_type      = "volume"
    delete_on_termination = true
    source_type           = "image"
    uuid                  = "${data.openstack_images_image_v2.centos_current.id}"
  }
}
```

```
# Create Meerkat worker instances
resource "openstack_compute_instance_v2" "worker" {
  count = "${var.worker_count}"
  name = "${format("${var.worker_name}-%02d", count.index+1)}"
  image_name      = "${var.image}"
  flavor_name     = "${var.flavor_workernode}"
  key_pair        = "${openstack_compute_keypair_v2.user_key.name}"
  security_groups = ["${openstack_compute_secgroup_v2.ssh.id}",
                     "${openstack_compute_secgroup_v2.worker-icmp.id}"]
#  user_data       = "${file("scripts/first-boot.sh")}"
  network {
    uuid          = "${openstack_networking_network_v2.private_network.id}"
  }
```

variables.tf

```
variable "flavor_workernode" {
  default = "ilifu-A"
}

variable "private_network" {
  default = {
    network = "private_network"
    subnet_name = "private_subnet"
    cidr        = "10.0.0.0/16"
  }
}

variable "worker_count" {
  default = "5"
}
```

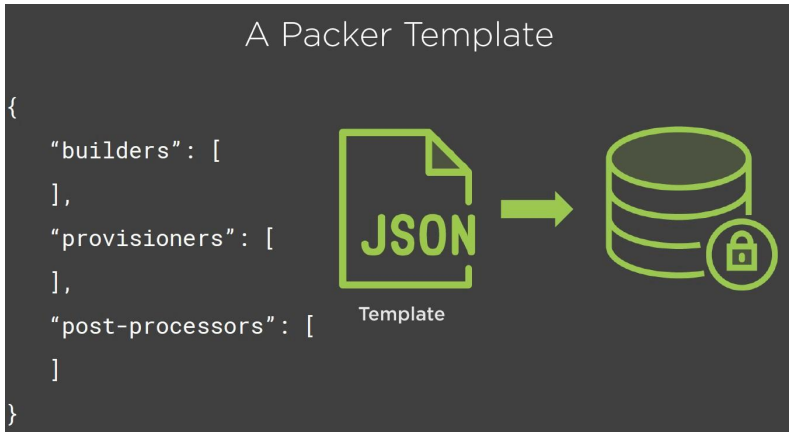
ilifu

Installing Terraform [##########100%]

- Terraform is written in GoLang.
- Single binary file and no dependency installs required
- Multiple Provider Support - AWS, GCP, Openstack, Vmware, many many others
- Information provided in your **provider.tf** downloads the relevant provider modules.
    - **“$ terraform init “**
- Installation on your desktop machine or a shared environment.

Tool: Packer.io

- Written in GO / X-platform
- Build / Automate/ Maintain virtual machine images.
- Packer could be used as an alternative to configuration management.
- Packer includes support for provisioners.
- The packer JSON defines everything which is in the image.
- Packer automatically builds and ships to Openstack Glance.
- Additional metadata tags can be included about the image import.



Building consistency into my environment

- It is important to maintain a robust and consistent environment.
  - If it is broken, destroy and redeploy.
  - Your reference data should be elsewhere so your compute environment should be flexible to destroy.
  - The environment should be immutable.
  - Software should be built into containers.

Questions ?