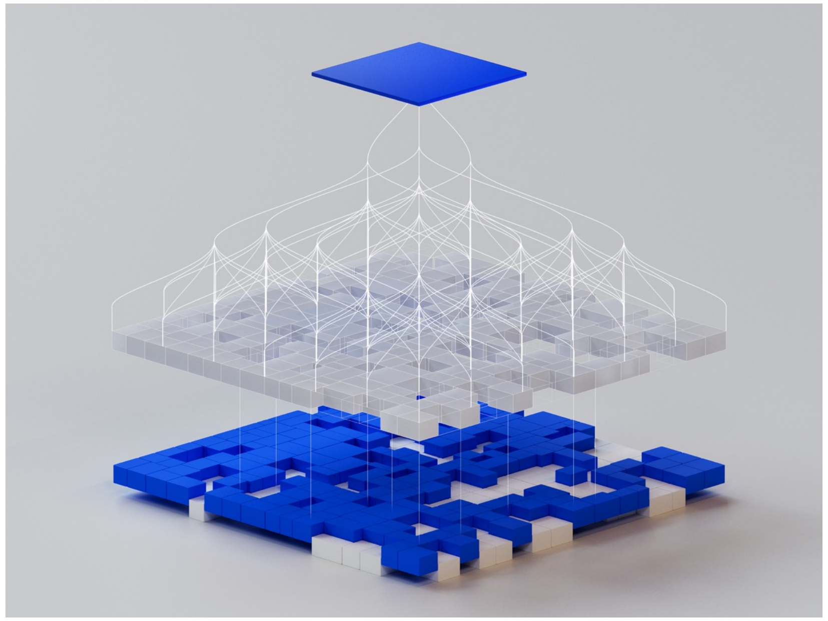
Simplification and Modernization with IBM Cloud Infrastructure Center

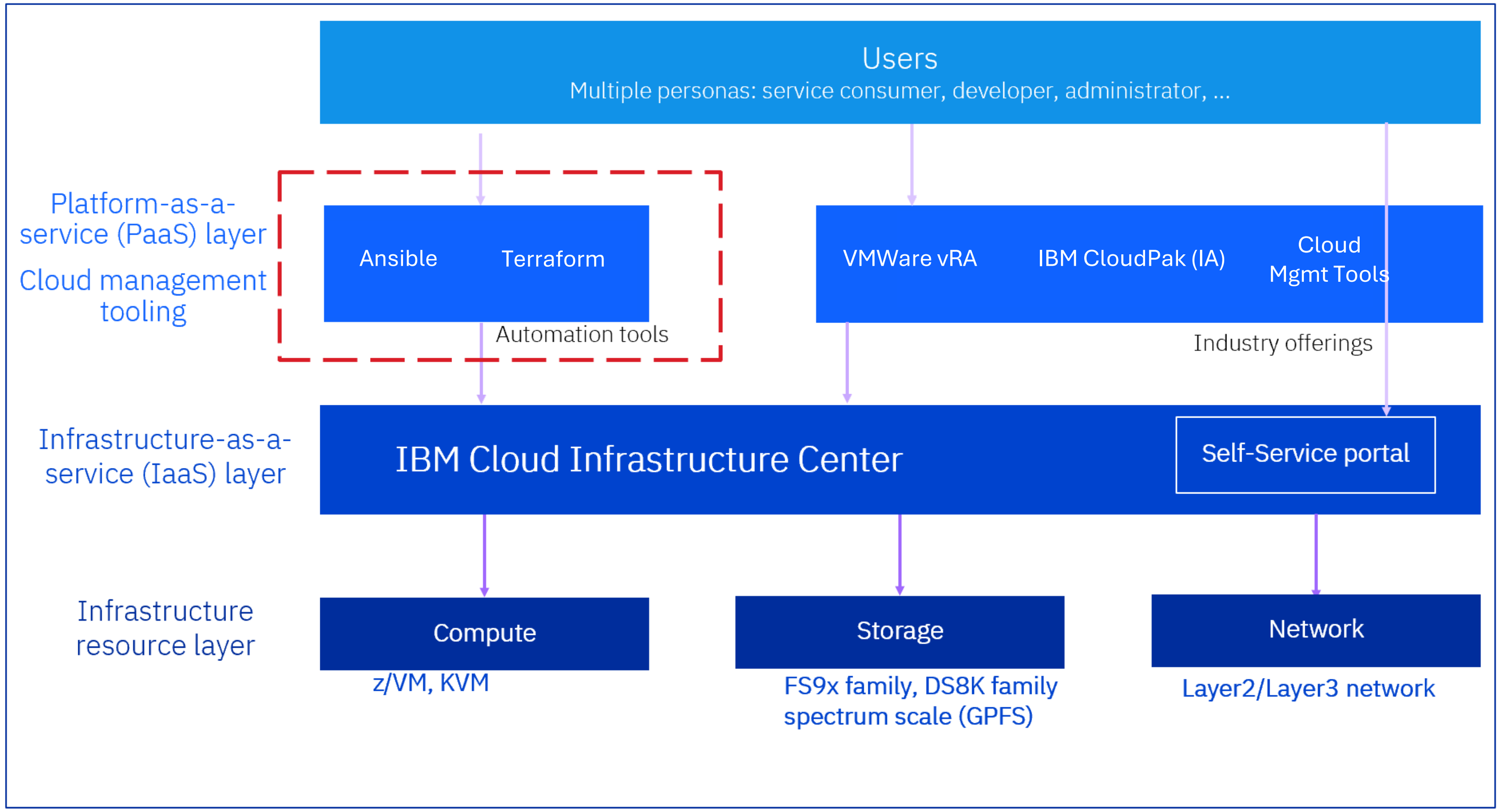


Michael Snihur
Solution Architect
snihurm@us.ibm.com

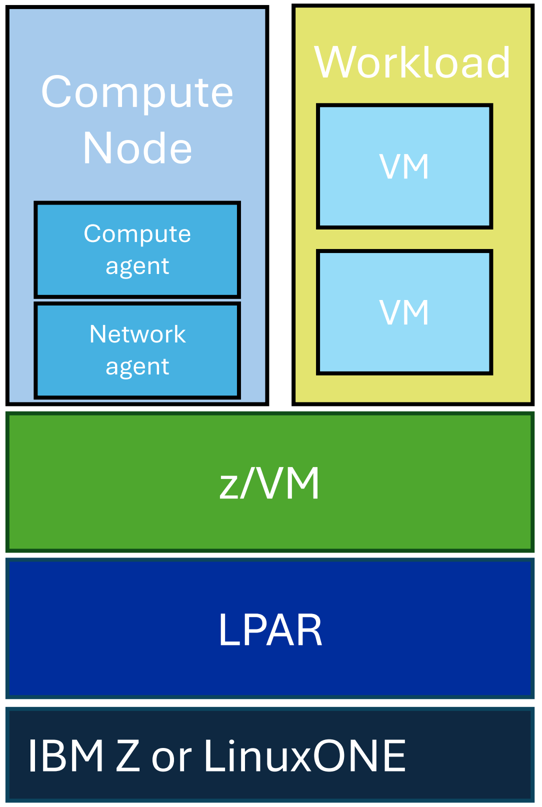
Agenda

- Simplification
  - automation, infrastructure as code
- Modernization
  - tooling, integration, devops
- Usage Scenarios
  - service provider, vm management, devops+
- Usage Examples

IBM Cloud Infrastructure Center provides the Infrastructure-as-a-Service layer



Users
Multiple personas: service consumer, developer, administrator, ...
Platform-as-a-service (PaaS) layer
Cloud management tooling
Ansible
Terraform
Automation tools
VMWare vRA
IBM CloudPak (IA)
Cloud Mgmt Tools
Industry offerings
Infrastructure-as-a-service (IaaS) layer
IBM Cloud Infrastructure Center
Self-Service portal
Infrastructure resource layer
Compute
z/VM, KVM
Storage
FS9x family, DS8K family spectrum scale (GPFS)
Network
Layer2/Layer3 network



z/VM Compute Node example

Simplification

Easier systems management

Flexible configurations

User friendly

Scalable and Robust

Simplification

Using scripts for simplified systems management is not a recent idea...

1995 VM Workshop at Wichita State University
U32 - How the CUNY Shepherds Tend their UNIX Flock
ABSTRACT:
The VM Group at CUNY recently inherited a growing flock of Unix machines. This presentation describes a methodology being used to keep watch over this flock using a simple REXX EXEC and REXX/Sockets.

Modernization

Tooling

Use industry standard tools and software on the mainframe for consistent processes

Integration

Service catalog offerings across multiple platforms

DevOps

Continuous integration, testing, delivery

Usage Scenarios

Service Provider

Virtual Machine Management

DevOps+

Usage Examples

Terraform to describe Infrastructure as Code

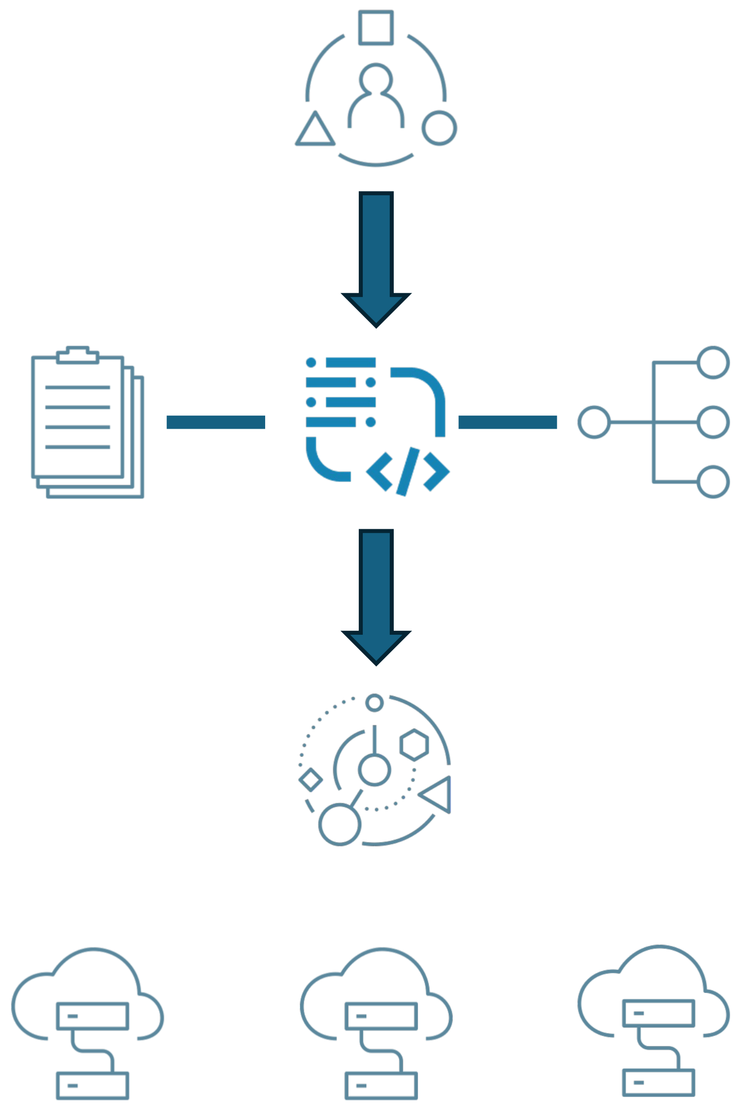
Ansible for deployment, configuration

Automate platform deployments like OCP, database

Usage Example 1

Terraform template

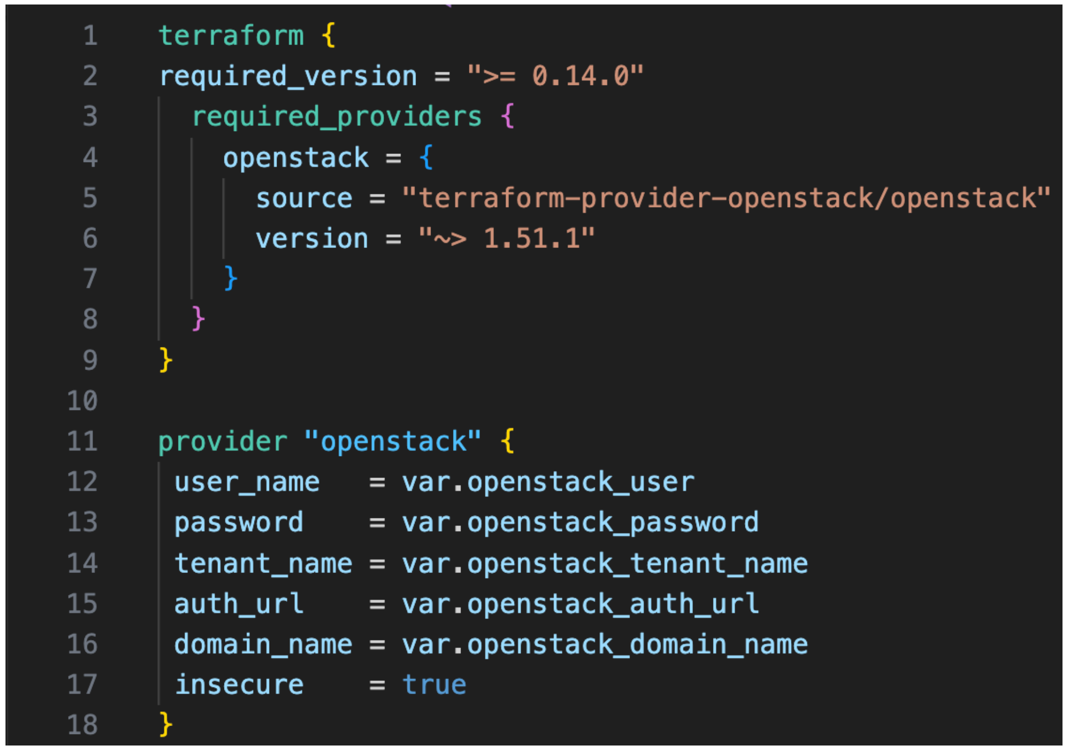
- Stateful resource definitions
- Declarative infrastructure
- Source Code Management
- Version control



Usage Example 1

main.tf

- Openstack Provider details
- Retrieve details from environment
- Provision instance from existing details
- Utilize variables for flexibility



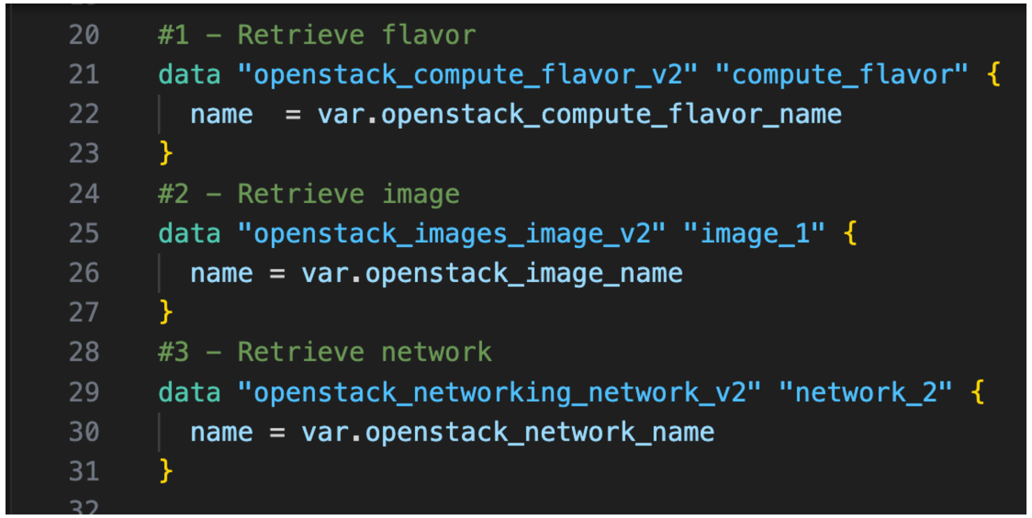
```
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source = "terraform-provider-openstack/openstack"
      version = "~> 1.51.1"
    }
  }
}

provider "openstack" {
  user_name   = var.openstack_user
  password    = var.openstack_password
  tenant_name = var.openstack_tenant_name
  auth_url    = var.openstack_auth_url
  domain_name = var.openstack_domain_name
  insecure    = true
}
```

Usage Example 1

main.tf

- Openstack Provider details
- Retrieve details from environment
- Provision instance from existing details
- Utilize variables for flexibility

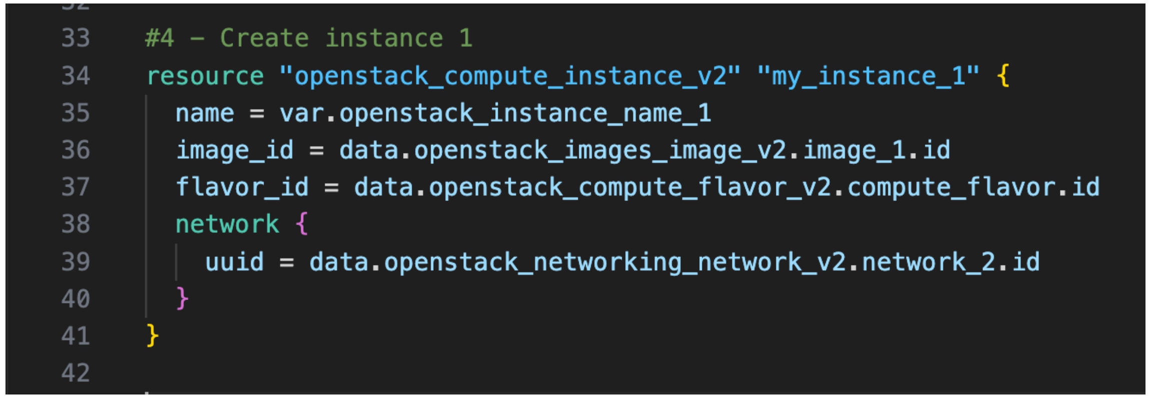


```
#1 - Retrieve flavor
data "openstack_compute_flavor_v2" "compute_flavor" {
  name   = var.openstack_compute_flavor_name
}
#2 - Retrieve image
data "openstack_images_image_v2" "image_1" {
  name = var.openstack_image_name
}
#3 - Retrieve network
data "openstack_networking_network_v2" "network_2" {
  name = var.openstack_network_name
}
```

Usage Example 1

main.tf

- Openstack Provider details
- Retrieve details from environment
- **Provision instance from existing details**
- Utilize variables for flexibility

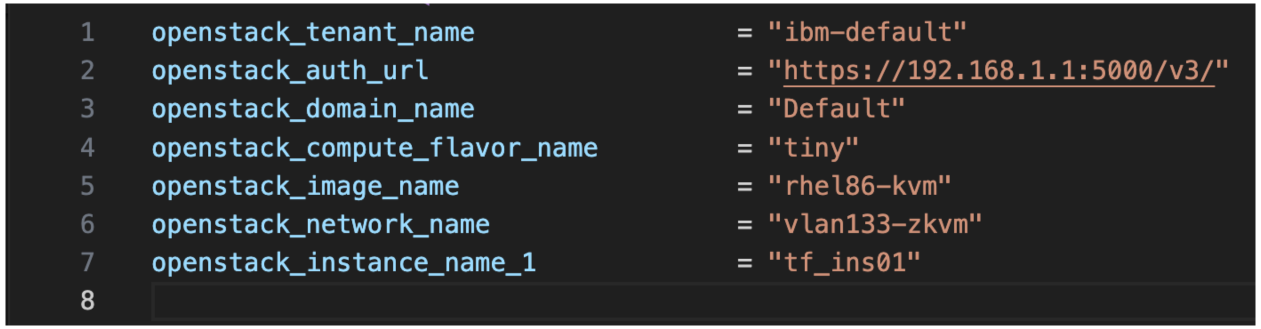


```
#4 - Create instance 1
resource "openstack_compute_instance_v2" "my_instance_1" {
  name = var.openstack_instance_name_1
  image_id = data.openstack_images_image_v2.image_1.id
  flavor_id = data.openstack_compute_flavor_v2.compute_flavor.id
  network {
    uuid = data.openstack_networking_network_v2.network_2.id
  }
}
```

Usage Example 1

main.tf

- Openstack Provider details
- Retrieve details from environment
- Provision instance from existing details
- Utilize variables for flexibility



```
openstack_tenant_name                 = "ibm-default"
openstack_auth_url                    = "https://192.168.1.1:5000/v3/"
openstack_domain_name                 = "Default"
openstack_compute_flavor_name         = "tiny"
openstack_image_name                  = "rhel86-kvm"
openstack_network_name                = "vlan133-zkvm"
openstack_instance_name_1             = "tf_ins01"
```

Usage Example 2

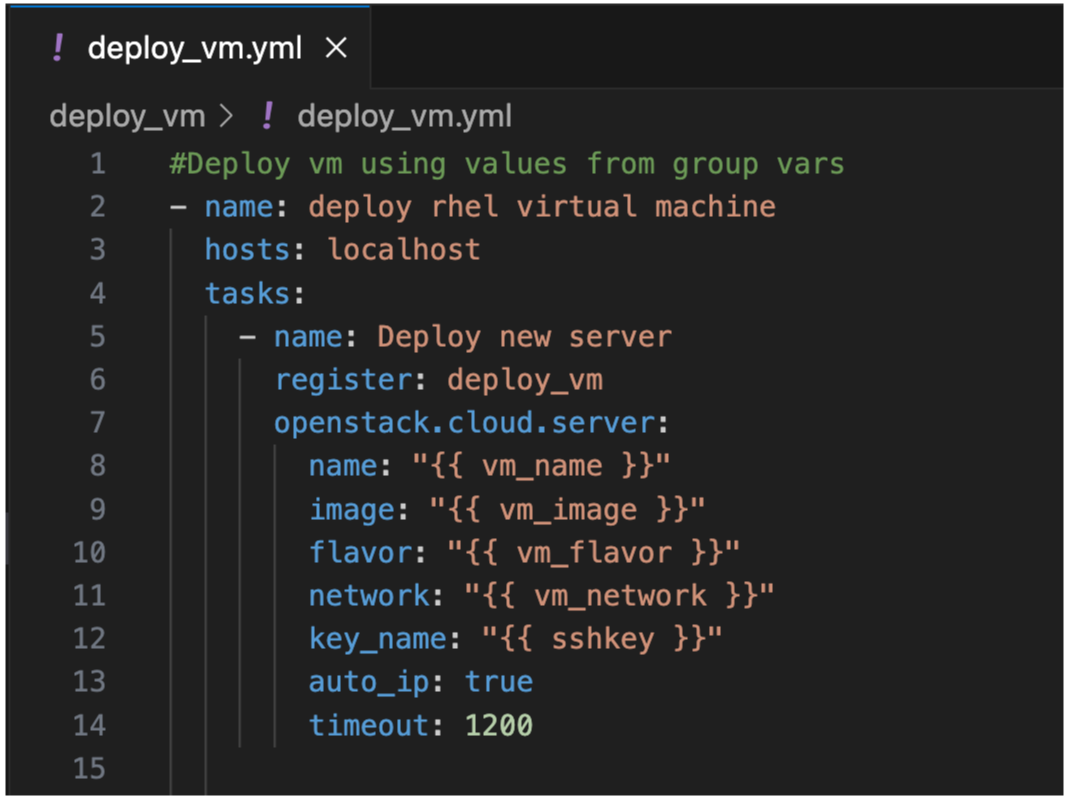
Ansible playbook

- Provision infrastructure
- Post-deployment configuration
- Collection support

Usage Example 2

Ansible playbook

- Provision infrastructure
- Post-deployment configuration
- Collection support



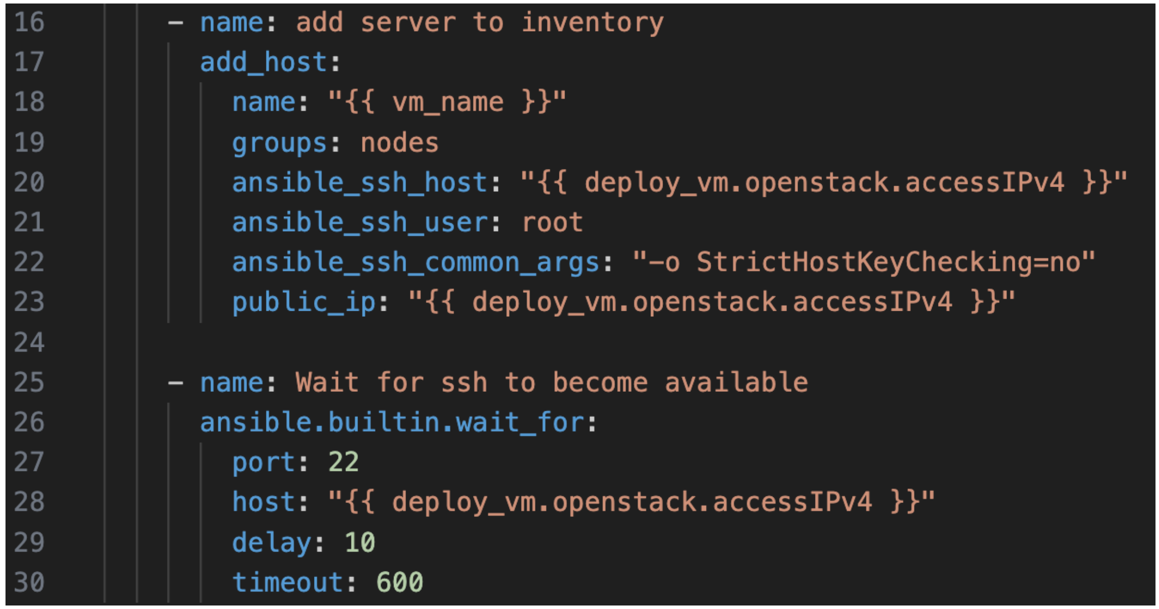
```yaml
#Deploy vm using values from group vars
- name: deploy rhel virtual machine
  hosts: localhost
  tasks:
    - name: Deploy new server
      register: deploy_vm
      openstack.cloud.server:
        name: "{{ vm_name }}"
        image: "{{ vm_image }}"
        flavor: "{{ vm_flavor }}"
        network: "{{ vm_network }}"
        key_name: "{{ sshkey }}"
        auto_ip: true
        timeout: 1200

```

Usage Example 2

Ansible playbook

- Provision infrastructure
- **Post-deployment configuration**
- Collection support



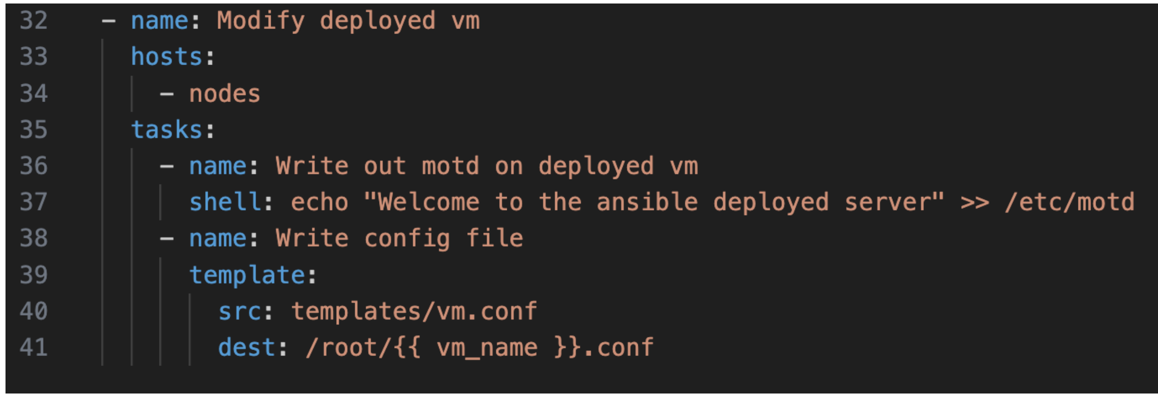
```yaml
    - name: add server to inventory
      add_host:
        name: "{{ vm_name }}"
        groups: nodes
        ansible_ssh_host: "{{ deploy_vm.openstack.accessIPv4 }}"
        ansible_ssh_user: root
        ansible_ssh_common_args: "-o StrictHostKeyChecking=no"
        public_ip: "{{ deploy_vm.openstack.accessIPv4 }}"

    - name: Wait for ssh to become available
      ansible.builtin.wait_for:
        port: 22
        host: "{{ deploy_vm.openstack.accessIPv4 }}"
        delay: 10
        timeout: 600
```

Usage Example 2

Ansible playbook

- Provision infrastructure
- Post-deployment configuration
- Collection support



```
  - name: Modify deployed vm
    hosts:
      - nodes
    tasks:
      - name: Write out motd on deployed vm
        shell: echo "Welcome to the ansible deployed server" >> /etc/motd
      - name: Write config file
        template:
          src: templates/vm.conf
          dest: /root/{{ vm_name }}.conf
```

Usage Example 2

Ansible playbook

- Provision infrastructure
- Post-deployment configuration
- Collection support

https://docs.ansible.com/ansible/latest/collections/openstack/cloud/index.html

Usage Example 3

OpenShift Container Platform UPI

- Customizable inventory.yaml

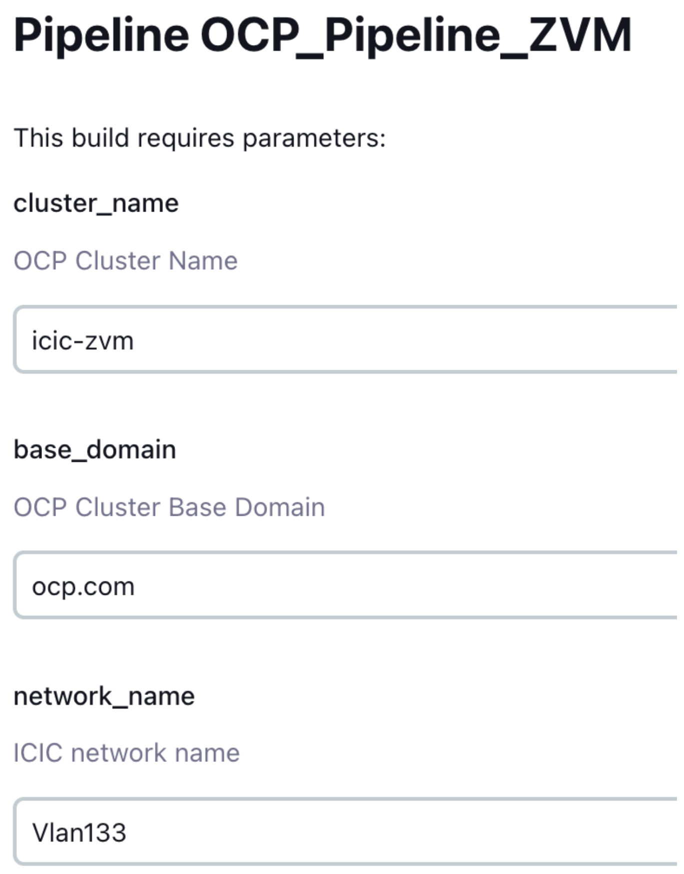
- Staged workflow to deploy OCP on s390x

https://github.com/IBM/z_ansible_collections_samples/tree/main/z_infra_provisioning/cloud_infra_center/ocp_upi

Usage Example 3

OpenShift Container Platform UPI

- Customizable inventory.yaml
- Staged workflow to deploy OCP on s390x



Pipeline OCP_Pipeline_ZVM

This build requires parameters:

cluster_name

OCP Cluster Name

icic-zvm

base_domain

OCP Cluster Base Domain

ocp.com

network_name

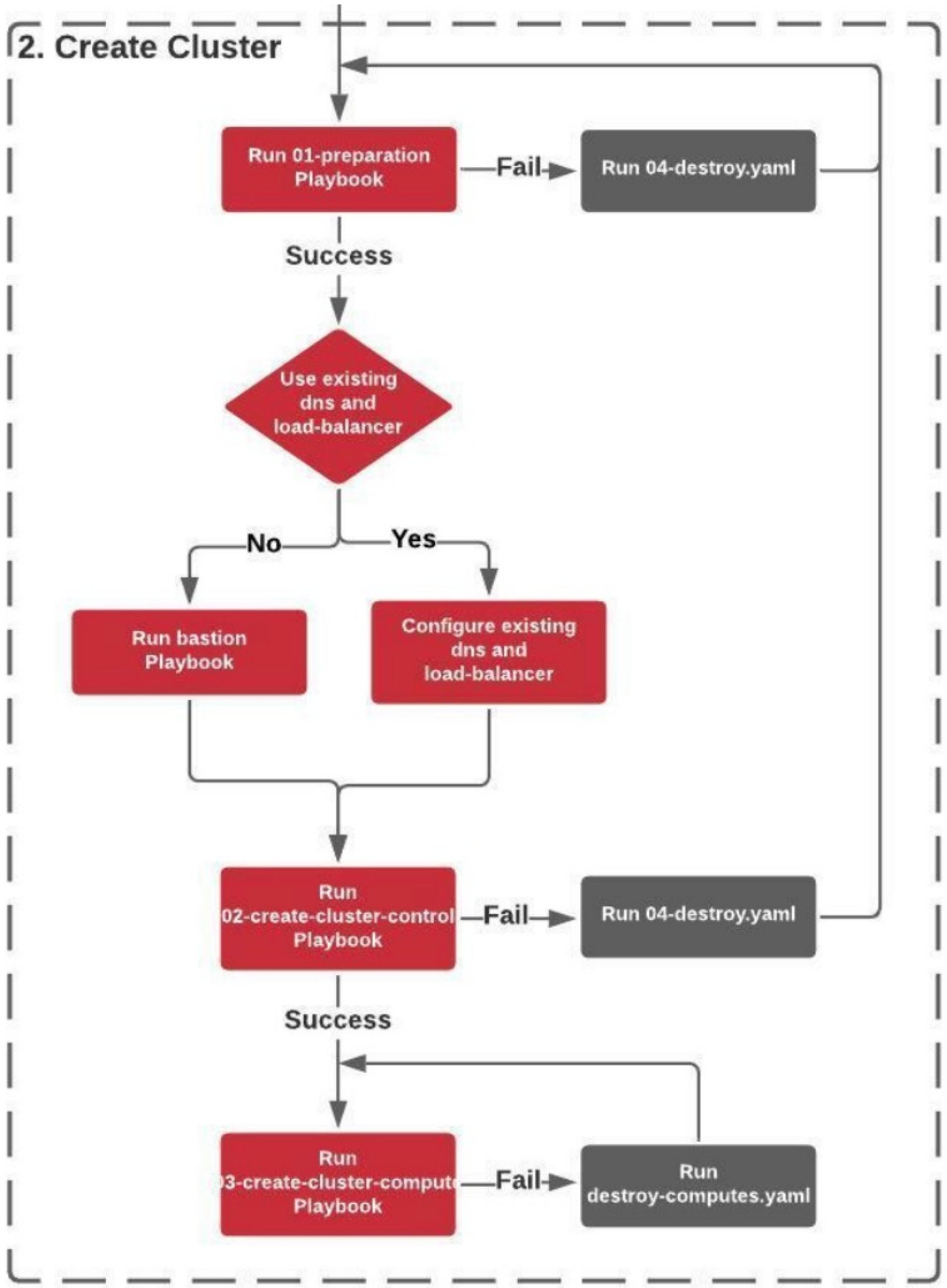
ICIC network name

Vlan133

Usage Example 3

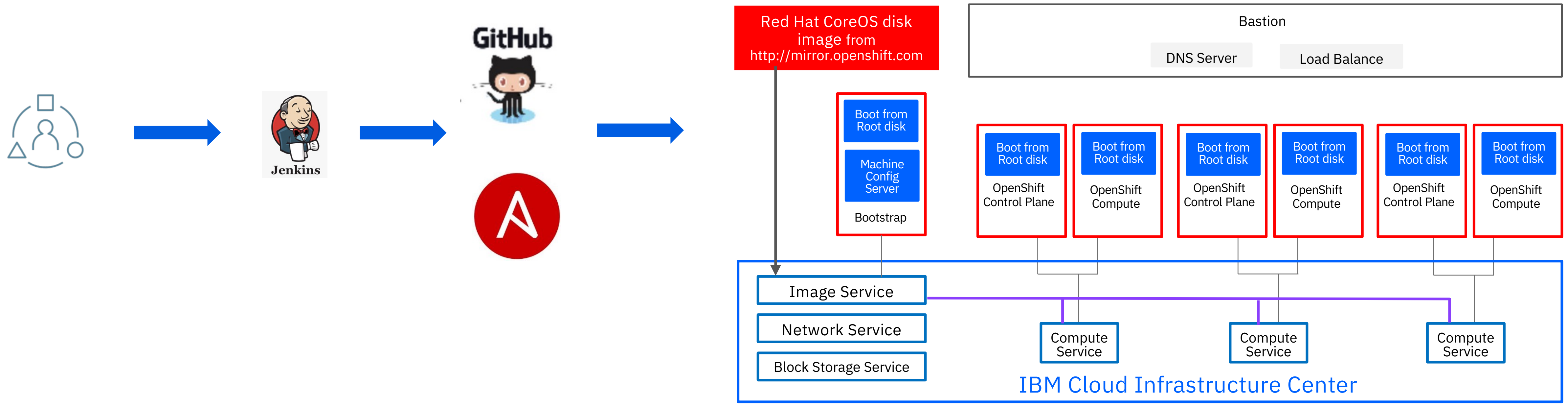
OpenShift Container Platform UPI

- Customizable inventory.yaml
- Staged workflow to deploy OCP on s390x


2. Create Cluster
Run 01-preparation Playbook
Fail
Run 04-destroy.yaml
Success
Use existing dns and load-balancer
No
Yes
Run bastion Playbook
Configure existing dns and load-balancer
Run 02-create-cluster-control Playbook
Fail
Run 04-destroy.yaml
Success
Run 03-create-cluster-compute Playbook
Fail
Run destroy-computes.yaml

Usage Example 3

OpenShift Container Platform UPI



https://github.com/IBM/z_ansible_collections_samples/tree/main/z_infra_provisioning/cloud_infra_center/ocp_upi

Thank You!