Building scalable private cloud on a budget

How to use open source and automation to build scalable private cloud on a budget

Who am I?

I am a DevOps Engineer with 7+ years of experience in Networks and Systems Engineering. I am passionate about computer networks, infrastructure and open source.

I work for Transnomis Solution, which helps governments with road information management and communication software. This talk presents private cloud running at Transnomis.

Why run your own infrastructure?

- **Cost -** Cloud services are expensive unless you are very small(with no staff to run infrastructure) or very large, with high scalability needs. Most companies have very predictable infrastructure loads.
  Lot of cloud hosting companies have 30-40% FCF margin, even with high marketing and employee expenses.
- **Flexibility -** You can run exactly what you want and get granular level of control of your infrastructure. You may not need to self host everything. Some IaaS providers support good level of automation and you can just lease the hardware. Eg. Equinix Metal, OVH.
- **Open source -** Open source software means you don't need to build your own software.

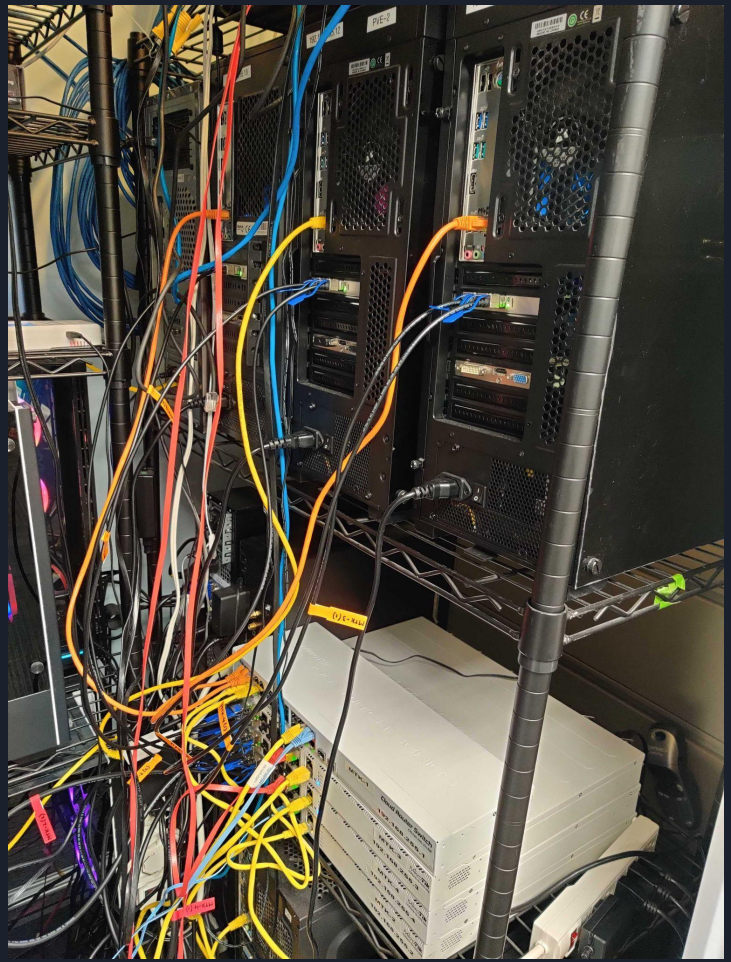
Topics we will be discussing - Building blocks for private cloud

- **VXLAN and OSPF -** For building scalable, isolated, highly redundant network.
- **Proxmox/Linux KVM -** for hypervisor.
- **Ceph -** For distributed storage.
- **Cloud-init/Cloudbase-init -** For automating Virtual Machine provisioning.
- **Ansible and Terraform -** For automation of VMs, network devices, hypervisors and VPNs.
- **LibreNMS -** For monitoring.
- **Hardware -** What to buy, sourcing etc.
- **Misc -** Identity management, VPNs.

Building the network

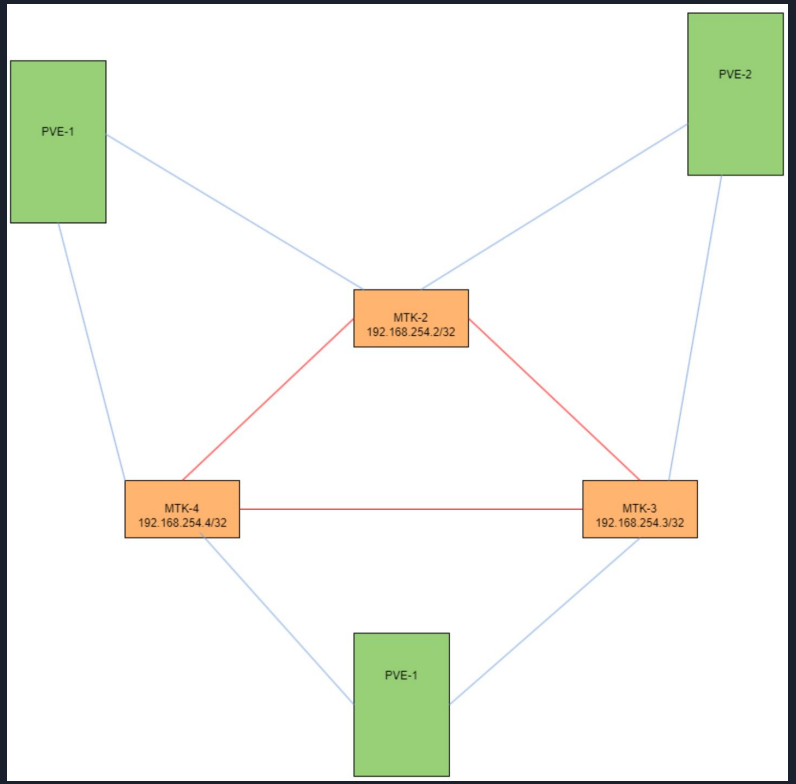
We want the network to be:

- **Scalable -** Should be able to scale to large number of physical machines.
- **Fast -** The network will run both the storage layer and other networks. We went with 25Gx2 per server as it was cheap enough to do.
- **Redundant -** Failure of multiple routers or network cables should not bring down the network.
- **Secure -** Networks should be isolated. VXLANs can provide VPC like functionality.



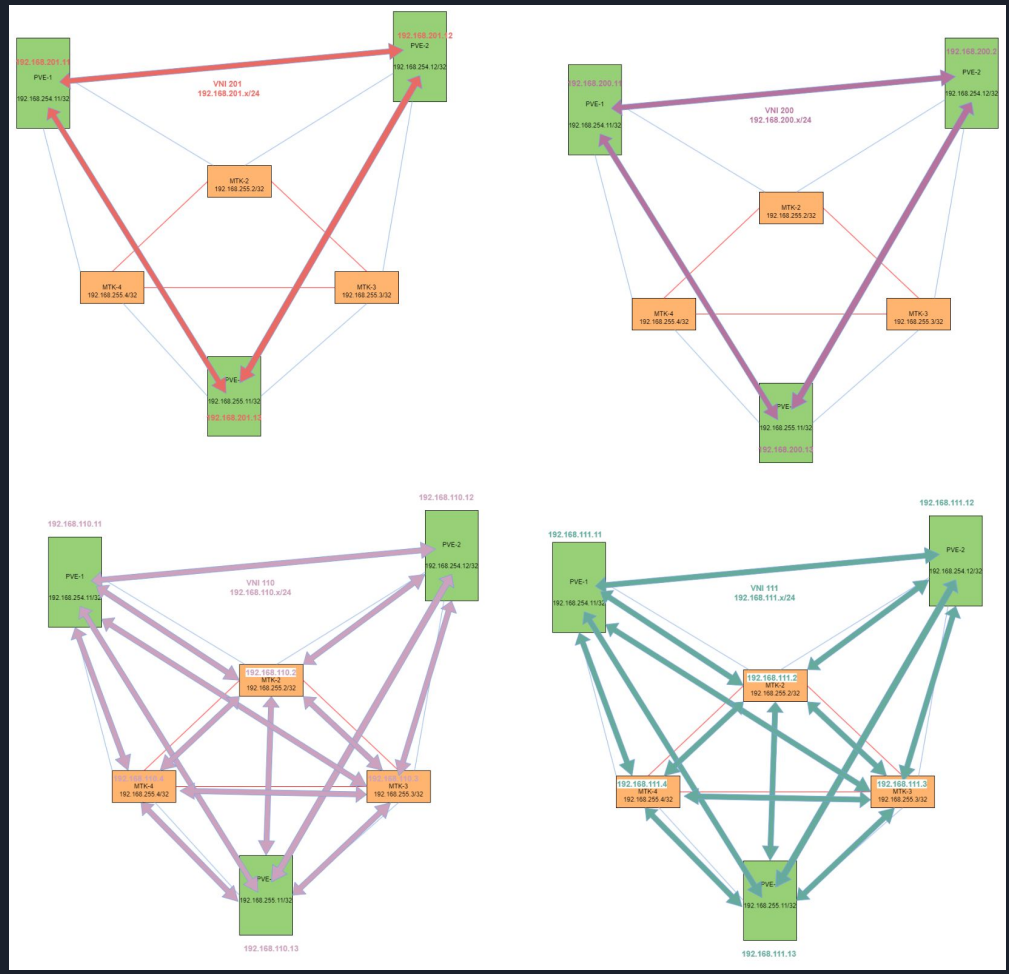
OSPF to maintain link state

- We useOSPF to maintain link state between all devices.
- Routers were connected over 100 Gbps backbone. For hypervisor hosts, they were connected to routers over 2x25 Gbps.
- OSPF makes all of devices connected work in a mesh and reroutes connections if we lose any devices or links.
- We use loopback IP on every device for them to communicate with each other.



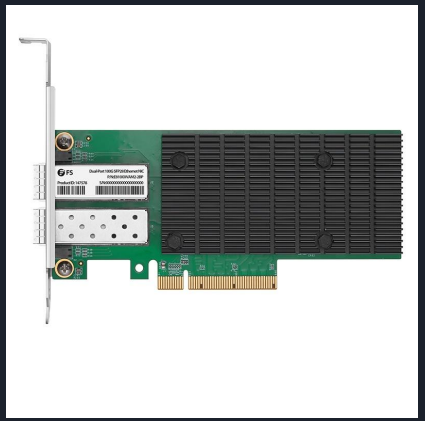
VXLAN over OSPF

- VXLANs can be thought of as layer 2 VPN tunnels over OSPF loopback IP, kind of like VPNs. They are similar to VPCs on other cloud providers.
- We run VXLANs between devices that need it.
- The top VXLAN tunnels is for Ceph and Proxmox Corosync clustering.
- The bottom two VXLAN tunnels is for VMs.
- VXLANs are scalable to 1000s of physical machines.
- For gateway we used VRRP.



Network budget

- We are using 3x **Mikrotik CCR2216-1G-12XS-2XQ** for core router. These have 12x 25 Gbps and 2x100 Gbps ports and can route packets at line rate. It's not perfect, but steal for the price. **Price: C$3380.**
- We are using Intel E810 NIC cards, but the cheaper Mellanox ConnectX-4 are cheaper if you can find them. **Price: C$330-580**.
- For cables DAC cables worked well for us. Fiber isn't necessary. **Price: C$40-50.**



Automating network

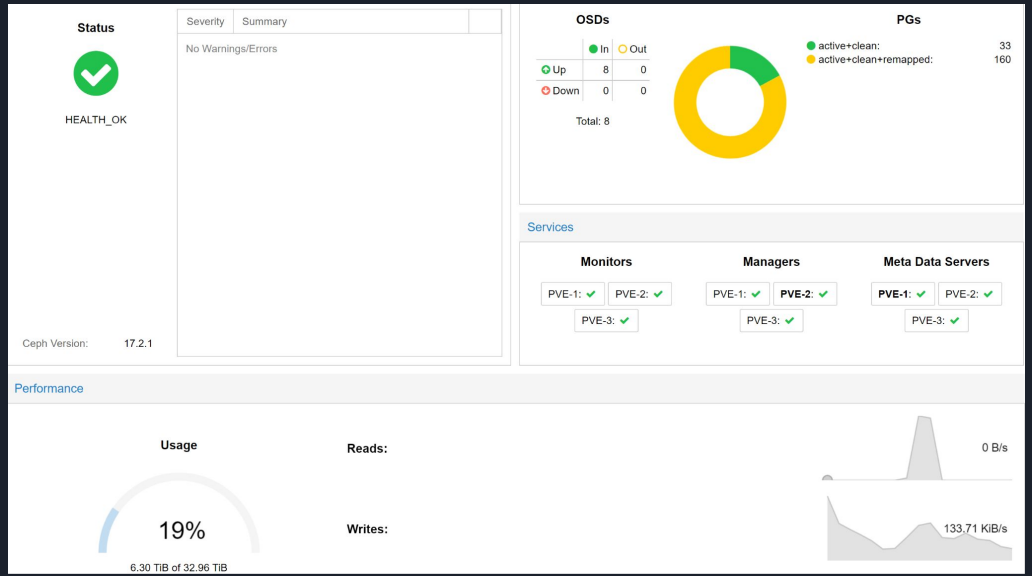
- We used Ansible to automate the network.
- We created reusable roles for every network function.
- Terraform is better, but the module for Mikrotik is lacking lot of functionality.
- We then plugged it into Gitlab CI/CD pipeline for infrastructure.

```
TASK [firewall : Add firewall rules - custom (src address/dst address defined)] ***
changed: [MTK-3] => (item={'action': 'accept', 'chain': 'forward', 'src_address': '192.168.1.0/24', 'dst_address': '192.
changed: [MTK-4] => (item={'action': 'accept', 'chain': 'forward', 'src_address': '192.168.110.0/24', 'dst_address': '19
changed: [MTK-3] => (item={'action': 'accept', 'chain': 'forward', 'src_address': '192.168.110.0/24', 'dst_address': '19
changed: [MTK-4] => (item={'action': 'accept', 'chain': 'forward', 'src_address': '192.168.1.0/24', 'dst_address': '192.
changed: [MTK-3] => (item={'action': 'accept', 'chain': 'forward', 'src_address': '192.168.1.0/24', 'dst_address': '192.
changed: [MTK-4] => (item={'action': 'accept', 'chain': 'forward', 'src_address': '192.168.192.0/24', 'dst_address': '19
changed: [MTK-3] => (item={'action': 'accept', 'chain': 'forward', 'src_address': '192.168.192.0/24', 'dst_address': '19
TASK [firewall : Add firewall rules - DENY all] ********************************
changed: [MTK-4]
TASK [firewall : Include NAT] **************************************************
included: /builds/infra/ansible-terraform/ansible/mikrotik/roles/firewall/tasks/nat.yml for MTK-4
TASK [firewall : Add firewall rules - DENY all] ********************************
changed: [MTK-3]
TASK [firewall : Include NAT] **************************************************
included: /builds/infra/ansible-terraform/ansible/mikrotik/roles/firewall/tasks/nat.yml for MTK-3
TASK [firewall : Add src NAT rule] *********************************************
changed: [MTK-4] => (item={'out_interface': 'bell', 'src_address': '192.168.110.0/23'})
TASK [firewall : Add src NAT rule] *********************************************
changed: [MTK-3] => (item={'out_interface': 'bell', 'src_address': '192.168.110.0/23'})
TASK [firewall : Add src NAT rule] *********************************************
changed: [MTK-4] => (item={'out_interface': 'bell', 'src_address': '192.168.143.1/32'})
TASK [firewall : Add src NAT rule] *********************************************
changed: [MTK-3] => (item={'out_interface': 'bell', 'src_address': '192.168.143.1/32'})
PLAY RECAP *********************************************************************
MTK-1                      : ok=27   changed=23   unreachable=0    failed=0    skipped=25   rescued=0    ignored=0
MTK-2                      : ok=34   changed=31   unreachable=0    failed=0    skipped=15   rescued=0    ignored=0
MTK-3                      : ok=45   changed=42   unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
MTK-4                      : ok=45   changed=42   unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
real   3m 58.00s
user   10m 3.50s
sys    0m 28.27s
Cleaning up project directory and file based variables
Job succeeded
```

```
      dst_address: "192.168.192.0/24"
      comment: "CUSTOM:8"
    - action: "accept"
      chain: "forward"
      src_address: "192.168.192.0/24"
      dst_address: "192.168.1.0/24"
      comment: "CUSTOM:9"
interfaces:
  - routeros_if: "ether1"
    desc: "Management"
    ipv4: "192.168.255.3/24"
  - routeros_if: "qsfp28-1-1"
    desc: "OSPF ring interface"
    ipv4: "192.168.141.2/30"
    mtu: 9050
    l2mtu: 9134
  - routeros_if: "qsfp28-2-1"
    desc: "OSPF ring interface"
    ipv4: "192.168.142.1/30"
    mtu: 9050
    l2mtu: 9134
  - routeros_if: "lo"
    desc: "loopback for OSPF"
    ipv4: "192.168.254.3/32"
  - routeros_if: "sfp28-1"
    desc: "PVE-1 (Port 2)"
    ipv4: "192.168.131.1/30"
    mtu: 9050
    l2mtu: 9134
  - routeros_if: "sfp28-2"
    desc: "PVE-2 (Port 2)"
    ipv4: "192.168.132.1/30"
    mtu: 9050
    l2mtu: 9134
ospf:
  router_id: "192.168.254.3"
  interfaces:
    - ospf_if: "qsfp28-1-1"
      type: "ptp"
      auth: "md5"
      auth_key: "transnomis"
    - ospf_if: "qsfp28-2-1"
      type: "ptp"
      auth: "md5"
      auth_key: "transnomis"
    - ospf_if: "lo"
      type: "passive"
    - ospf_if: "zerotier1"
      type: "passive"
    - ospf_if: "sfp28-1"
      type: "broadcast"
    - ospf_if: "sfp28-2"
      type: "broadcast"
```

Proxmox+Ceph=❤️

- We use Proxmox on hypervisor hosts, which has excellent integration with Ceph distributed and redundant storage.
- Ceph can store Petabytes of data easily.
- Ceph and Proxmox Corosync clustering allows us to make VMs highly available and have live migration.
- Proxmox uses Linux KVM as hypervisor.
- Proxmox host post-provisioning is done with Ansible.



Terraform+Cloud-init+Ansible for Linux VMs

- We are using Terraform and Cloud-init for automating Linux VM creation. telmate/proxmox Terraform provider interacts with Proxmox VMs for VM creation and works very well.
- Cloud-init sets up the VM with correct hostname, DNS, timezone, adds users, sets up SSH keys.
- Ansible post provisions the VMs, adds it to monitoring, sets up correct applications.

Cloud-init config

- We create cloud-init templates that are pushed to every Proxmox host using Ansible. This tells Proxmox how to configure the VM, including host name, time zone.
- We use Jumpcloud for identity management, pushing users and SSH keys. We setup the agent with Cloud init.

```
#cloud-config
package_update: true
package_upgrade: true
timezone: America/Toronto
packages:
  - sudo
  - python3
  - curl
  - apt-rdepends
  - apt-show-versions
  - coreutils
  - apt-curl
  - dpkg
  - grep
  - hostname
  - libc-bin
  - libnss3
  - libnss3-tools
  - lsb-release
  - lsof
  - mawk
  - openssl
  - passwd
  - procps
  - sysvinit-utils
  - tar

preserve_hostname: false
manage_etc_hosts: false
fqdn: {{ item.host }}.transnomis.com

write_files:
- path: /root/jc_register.sh
  owner: root:root
  permissions: '0755'
  content: |
    #!/bin/bash
    sleep 120
    conf="`cat /opt/jc/jcagent.conf`"
    regex="systemKey\":\"(\w+)\""

    if [[ $conf =~ $regex ]] ; then
      systemKey="${BASH_REMATCH[1]}"
    fi

    echo $systemKem
    API_KEY="{{ jumpcloud_api_key }}"
    GROUP_ID="{{ jumpcloud_vm_groupid }}"
    curl -X POST https://console.jumpcloud.com/api/v2/systemgroups/${GROUP_ID}/members \
      -H "Accept: application/json" \
      -H "Content-Type: application/json" \
      -H "x-api-key: ${API_KEY}" \
    -d@- <<EOF
        {
            "op": "add",
            "type": "system",
            "id": "$systemKey"
        }
    EOF

configjc:
  - &config_jc |
    curl --tlsv1.2 --silent --show-error --header 'x-connect-key: {{ jumpcloud_vm_connectkey }}' https://kickstart.jumpcloud.com/Kickstart | sudo bash

runcmd:
 - [ sh, -c, *config_jc ]
 - bash /root/jc_register.sh
 - rm -rf /root/jc_register.sh
```

Terraform+Ansible for provisioning Linux VMs

- Terraform then uses Cloud-init config shown earlier to provision the VM.
- Ansible scripts then post provision the VM with specific software.
- You can run any application you like on VMs - Including Kubernetes, load balancer, S3, Docker containers on top of VMs.

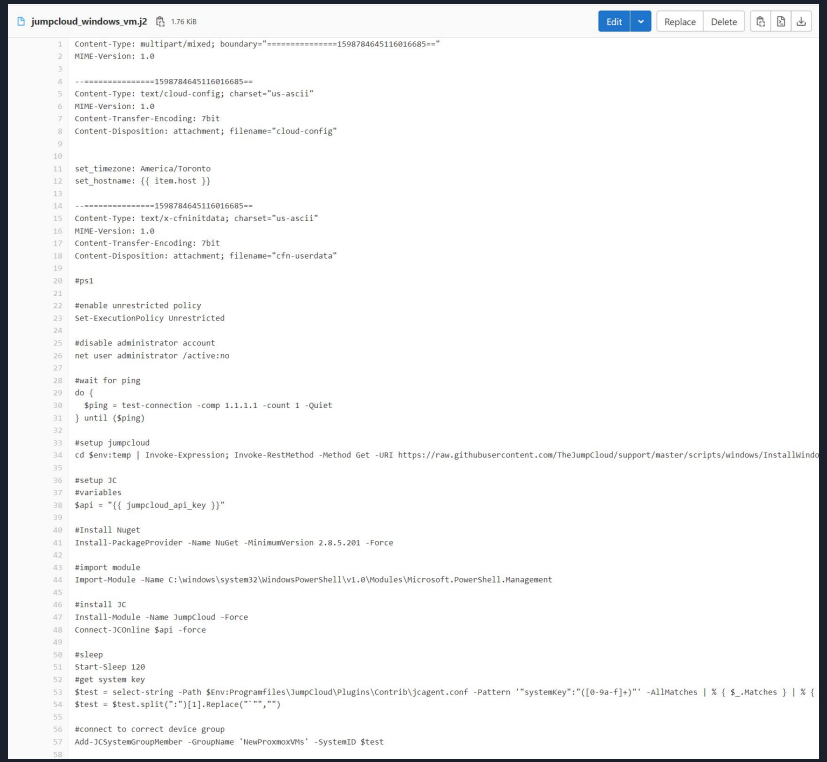
rod.tf 1.54 KiB

```
resource "proxmox_vm_qemu" "prod-sql14-1" {
  name         = "prod-sql14-1"
  target_node  = "PVE-2"
  clone        = "ubuntu2204-cloudinit"
  os_type      = "cloud-init"
  nameserver   = "192.168.111.21 192.168.111.22"
  searchdomain = "transnomis.com"
  cicustom     = "user=local:snippets/
  ipconfig0    = "ip=192.168.111.52/24
  hastate      = "started"
  hagroup      = "ha_default"
  onboot       = true
  memory       = 8192
  cores        = 4
  agent        = 1
  scsihw       = "virtio-scsi-pci"
  disk {
    size    = "30G"
    type    = "scsi"
    storage = "vm_data"
    backup  = 1
  }
  network {
    model    = "virtio"
    bridge   = "vmbr111"
    firewall = "false"
  }
  lifecycle {
    ignore_changes = [
```

```
$ time ansible-playbook qa-sql14.yaml
[WARNING]: Invalid characters were found in group names but not replaced, use
-vvvv to see details
PLAY [qa-sql14] ***************************************************************
TASK [Gathering Facts] ********************************************************
ok: [qa-sql14-1]
TASK [ansible.builtin.apt] ****************************************************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : include_tasks] ********************************
included: /builds/infra/ansible-terraform/ansible/vm/roles/ansible-role-postgresql/tasks/variables.yml for qa-sql14-1
TASK [ansible-role-postgresql : Include OS-specific variables (Debian).] *******
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Include OS-specific variables (RedHat).] *******
skipping: [qa-sql14-1]
TASK [ansible-role-postgresql : Include OS-specific variables (Amazon).] *******
skipping: [qa-sql14-1]
TASK [ansible-role-postgresql : Include OS-specific variables (Fedora).] *******
skipping: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_packages.] ******************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_version.] *******************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_daemon.] ********************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_data_dir.] ******************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_bin_path.] ******************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_config_path.] ***************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_unix_socket_directories_mode.] ***
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_log_dir.] *******************
ok: [qa-sql14-1]
TASK [ansible-role-postgresql : Define postgresql_effective_log_dir, if postgresql_log_dir is absolute] ***
skipping: [qa-sql14-1]
```

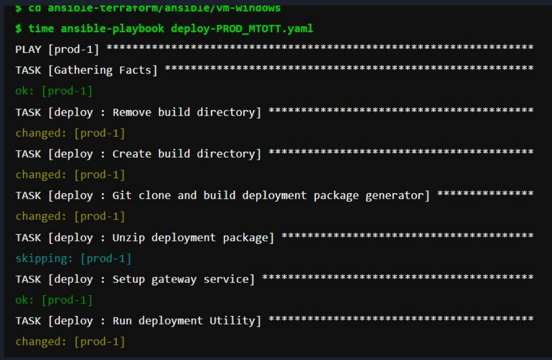
Cloudbase-init for provisioning Windows VMs

- Cloudbase-init does the same work as Cloud-init does on Linux.
- We can use Powershell to run scripts for the first time to ready the VM. We setup Jumpcloud this way.
- Terraform will use Cloudbase-init to provision the VM and Ansible(over WinRM) will post-provision it.



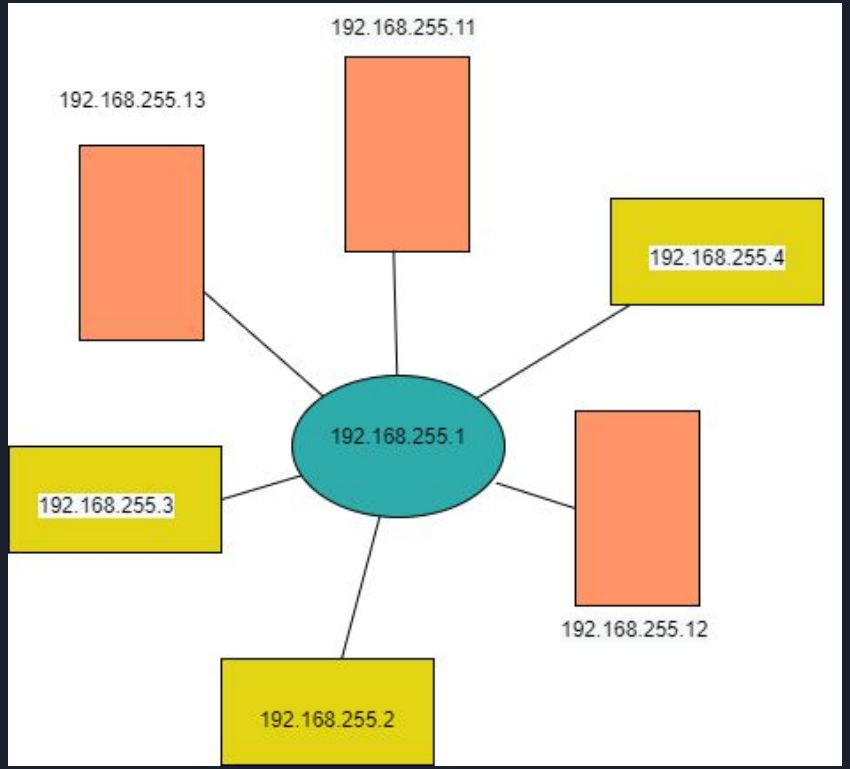
Provision ITS software with Ansible role

- We use Ansible role “ITS” to provision ITS software.
- This role will use variables defined per host to provision hosts appropriately.
- This then works with deploy script to get the site running.



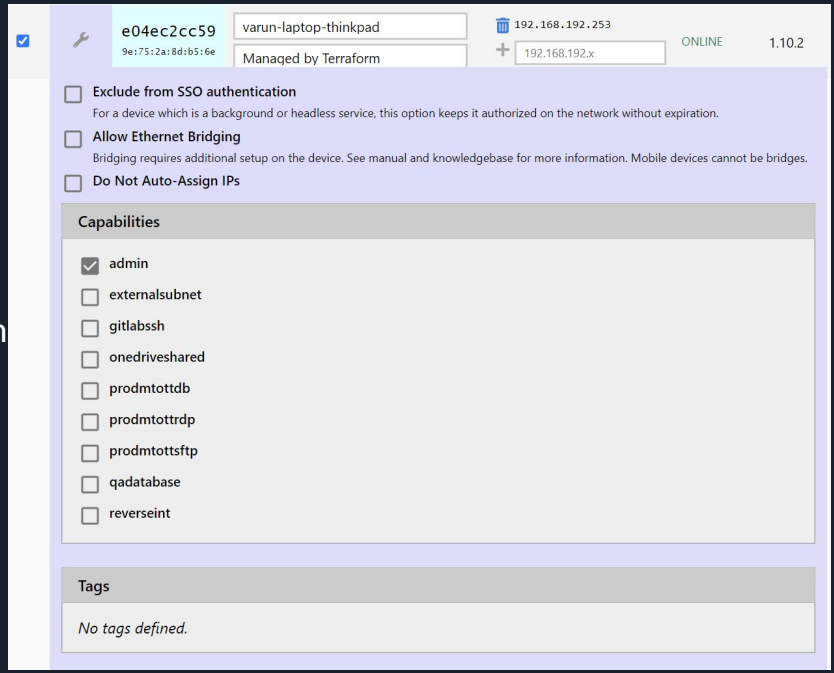
Monitoring and LibreNMS

- We use LibreNMS for monitoring hosts, services, network devices.
- We use separate network subnet to connect all devices together in separate network for out of band management.



Zerotier VPN

- We use Zerotier for VPN, which is a SDN VPN.
- Instead of traditional server-client model, every host is a node in the VPN cluster.
- We then can define, which network is behind which host and define network rules for every host.
- We have multiple sites and this is much easier than doing traditional VPNs.



Thank you!