Integration d ’Ansible chez Bell Canada

Déploiement de serveurs virtuels

Dominic Charbonneau

2021-03-18

Portail de provisionnement de serveur

Défis et solutions

Défis

Le projet de remplacement de l'outil de déploiement chez Bell a commencé en Février 2020 afin de remplacer l'outils en place actuellement.

- Le défis de Bell était de mettre en place un outil qui est :
  - Rapide
  - Flexible
  - Facile a développer et a maintenir
  - Développer a l'interne afin de répondre au critères de l'entreprise.

Solutions

- Plusieurs compagnies ont été sollicité afin de trouvé la bonne solution. Après plusieurs tests et considérations, Ansible Tower a été choisi.
  - Le choix s'est arrêter sur Ansible puisqu'il est très facile d'interagir avec Ansible Tower en utilisant des appels d'API fait par le portail interne.
- C'est en Septembre 2020 que le travail a commencer sur l'automatisation et la phase 1 a été livrer le 8 Janvier 2021.
- Dans cette présentation, je vais vous montré comment nous avons mis en place un workflow pour déployé des serveurs virtuel automatiquement via un appel d'API fait par le portail.
  - L'exemple se base sur un serveur Windows, mais le workflow peux déployer des serveurs RedHat lorsque requis.

Bell

Implémentation

- Une équipe interne a bâti une application web sur OpenShift que les clients utilisent pour faire des demandes de serveurs.
- Le portail envoi un appel d'API a Ansible Tower afin de faire le déploiement.
- Tout le code d'Ansible est héberger sur un gitlab interne, pour l'instant un workflow manuel est utilisé afin de tester le code, un pipeline est en construction.

Bell

Workflow de déploiement

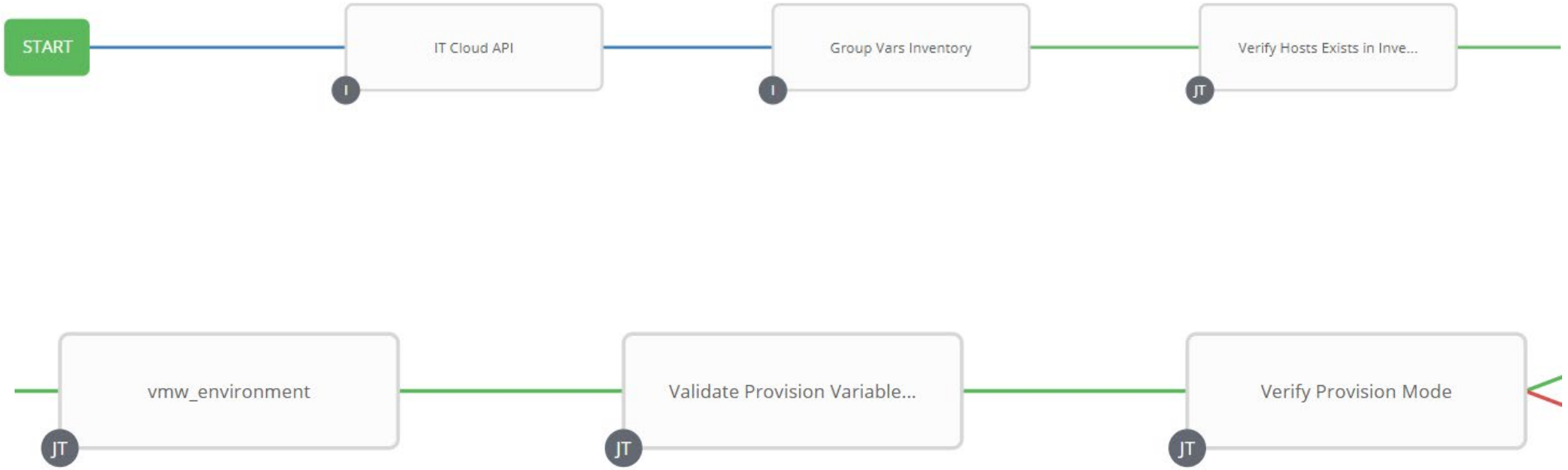
Avec Ansible Tower

Workflow Breakdown

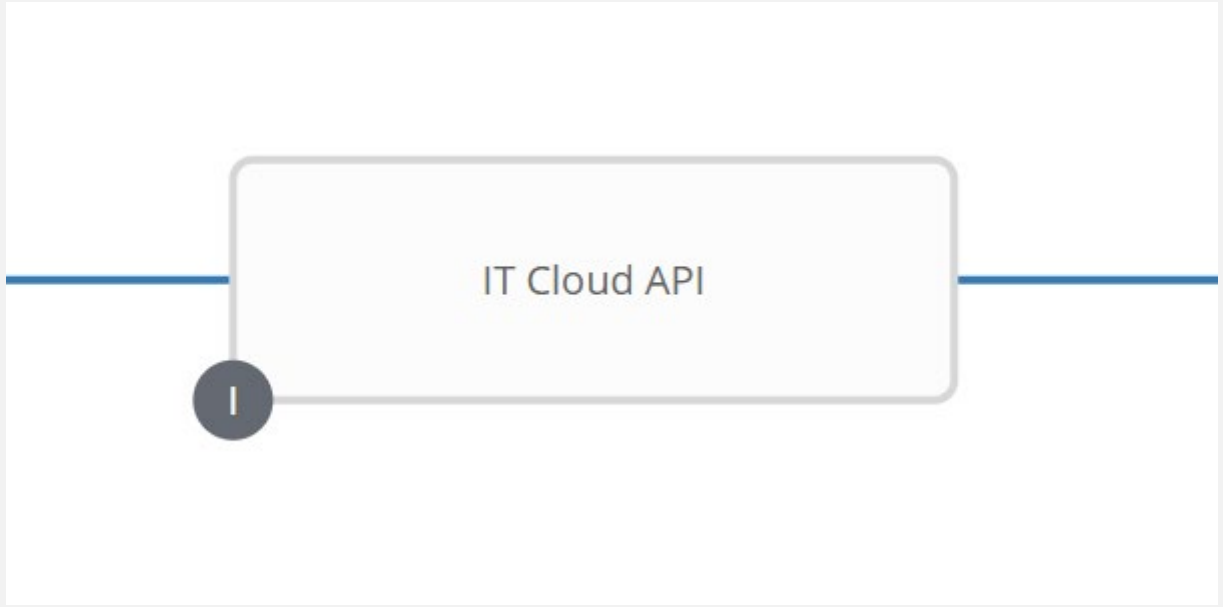
1. **Mise a jour de l’inventaire et définitions des variables**
2. **Réservation IPAM et DNS**
3. **Déploiement et configuration OS**
4. **Vérification finale et retour au portail**

Mise a jour de l’inventaire et définitions des variables

Workflow - Inventaire et groupes



Mise a jour de l’inventaire



- Ansible se connecte a la base de donnée du portail via un script Python et ramasse les serveurs.

Mise a jour de l’inventaire - Suite

- Ansible Tower crée ensuite des groupes selon les données reçu dans l’étape précédente et assigne des variables.

Mise a jour de l’inventaire - Suite

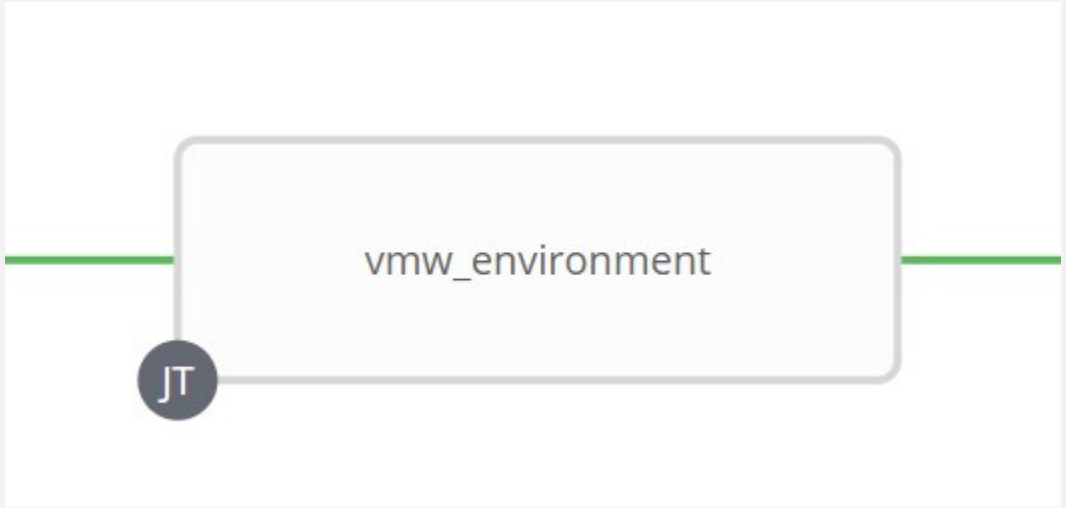
```yaml
---
all_servers:
  children:
    Linux:
      children:
        Linux Redhat-7:
          vars:
            vcenter_template: "RHEL7X_x64_C2_Base_"
        Linux Redhat-8:
          vars:
            vcenter_template: "RHEL8X_x64_C2_Base_"
      vars:
        timezone: "America/Toronto"
        target_os: "Linux"
        ansible_ssh_extra_args: "-o StrictHostKeyChecking=no -o Use
        templateDiskSize: 60
    Windows:
      children:
        Windows-2016:
          vars:
            vcenter_template: "2016STD_C2_Base_"
        Windows-2012:
          vars:
            vcenter_template: "2012R2STD_C2_Base_"
      vars:
        timezone: "035"
        target_os: "Windows"
        ansible_connection: winrm
        ansible_port: 5985
        ansible_winrm_transport: ntlm
        ansible_winrm_scheme: http
        ansible_winrm_server_cert_validation: ignore
        ansible_winrm_message_encryption: always
        ansible_winrm_operation_timeout_sec: 9001
        ansible_winrm_read_timeout_sec: 9002
```


Bell

Vérification de l’inventaire

- Confirme que le serveur est bien présent dans l’inventaire afin de le déployer.

Definitions des variables



- Assigne des variables selon les informations reçu par le portail afin de déployé le serveur au bon endroit.

Définitions des variables - Suite

```yaml
---
# tasks file for roles/vmw_deploy

- name: Include tasks for ICN
  include_tasks: deployICN.yaml
  when: "'ICN' in group_names"

- name: Include tasks for NML
  include_tasks: deployNML.yaml
  when: "'NML' in group_names"

- name: Include tasks for DMZ
  include_tasks: deployDMZ.yaml
  when: "'DMZ' in group_names"

- name: Include tasks for disk setup
  include_tasks: deployDisks.yaml

- name: Include tasks for RHEL subscription
  include_tasks: rhelSubscription.yaml
  when:
    - "'Linux' in group_names"
```

```yaml
- name: Set stats for workflow
  set_stats:
    data:
      vcenter_cluster: "{{ vcenter_cluster | default(omit) }}"
      vcenter_datastore: "{{ vcenter_datastore | default(omit) }}"
      deploy_network: "{{ deploy_network | default(omit) }}"
      ipam_subnet: "{{ ipam_subnet | default(omit) }}"
      deploy_gateway: "{{ deploy_gateway | default(omit) }}"
      deploy_netmask: "{{ deploy_netmask | default(omit) }}"
      deploy_network_fc: "{{ deploy_network_fc | default(omit) }}"
      ipam_subnet_fc: "{{ ipam_subnet_fc | default(omit) }}"
      deploy_gateway_fc: "{{ deploy_gateway_fc | default(omit) }}"
      deploy_netmask_fc: "{{ deploy_netmask_fc | default(omit) }}"
      deploy_network_bk: "{{ deploy_network_bk | default(omit) }}"
      ipam_subnet_bk: "{{ ipam_subnet_bk | default(omit) }}"
      deploy_gateway_bk: "{{ deploy_gateway_bk | default(omit) }}"
      deploy_netmask_bk: "{{ deploy_netmask_bk | default(omit) }}"
      dmz_route: "{{ dmz_route | default(omit) }}"
      dmz_dns_bc: "{{ dmz_dns_bc | default(omit) }}"
      dmz_user_password : "{{ dmz_user_password | default(omit) }}"
```

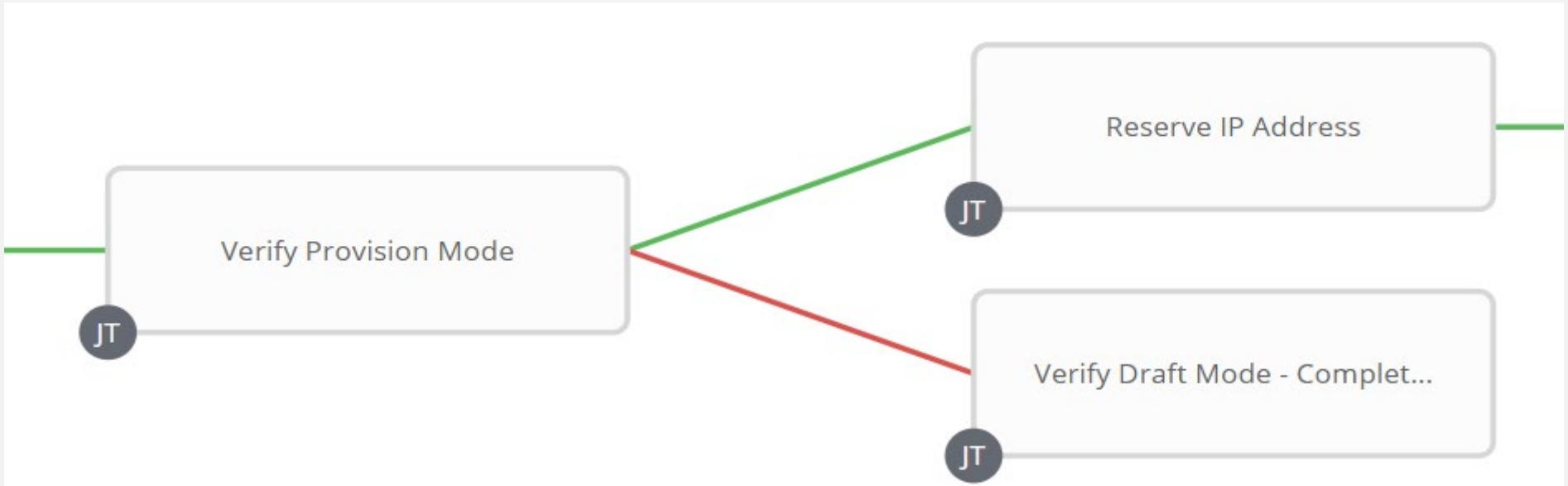

Bell

Definitions des variables - Suite

- Le rôle appel plusieurs taches tout dépendant des variables déjà présente dans l'inventaire et assigne les variables requise pour le rôle vmware_guest.
- Le rôle enregistre ensuite les variables avec un set_stats.
- Un Playbook roule ensuite afin de s'assurer que tout est bien assigné.

Verification du mode de provisionement

- Ce rôle est un “failsafe” afin de s’assurer que le serveur n’est pas un test. Il valide tout simplement que la variable provisionmode = Yes.
- Si provisionmode = no, le workflow se termine en mode Draft, sinon il se poursuit.



Réservation IPAM et DNS

Réservation IPAM et configuration DNS

- Ansible envoi ensuite un appel d'API vers le serveur IPAM selon les données définies dans la section précédente afin de réserver une adresse IP dans le système.

Création de l'entré DNS

- Le DNS est créé seulement lors d'un déploiement dans le DMZ ou dans le cas d'un serveur Linux.
- Le module "win_dns_record" est utilisé afin de crée l'entré, via un serveur Windows qui sert de Windows Bridge, utilisant credssp pour passer l'usager au serveur DNS.

```yaml
---
# tasks file for roles/ad_dns_entry
- name: Create DNS entry
  win_dns_record:
    name: "{{ inventory_hostname | lower }}"
    type: "A"
    value: "{{ ip }}"
    zone: "acme.corp"
    computer_name: <domain_controller>
  delegate_to: <Windows_bridge>
  when: "'DMZ' in group_names"
```

```yaml
- name: PTR - DMZ
  win_dns_record:
    name: "{{ (ip | ipaddr('revdns')).split('.')[0:2] | join('.') }}"
    type: "PTR"
    value: "{{ inventory_hostname | lower }}"
    zone: "{{ (ip | ipaddr('revdns')).split('.')[2:6] | join('.') }}"
    computer_name: <domain_controller>
  delegate_to: <Windows_bridge>
  when: "('Linux' in group_names and 'Production' in group_names) or 'DMZ' in group_names"
  ignore_errors: yes
```


Bell

Déploiement et configuration OS

Workflow du déploiement

Déploiement du serveur

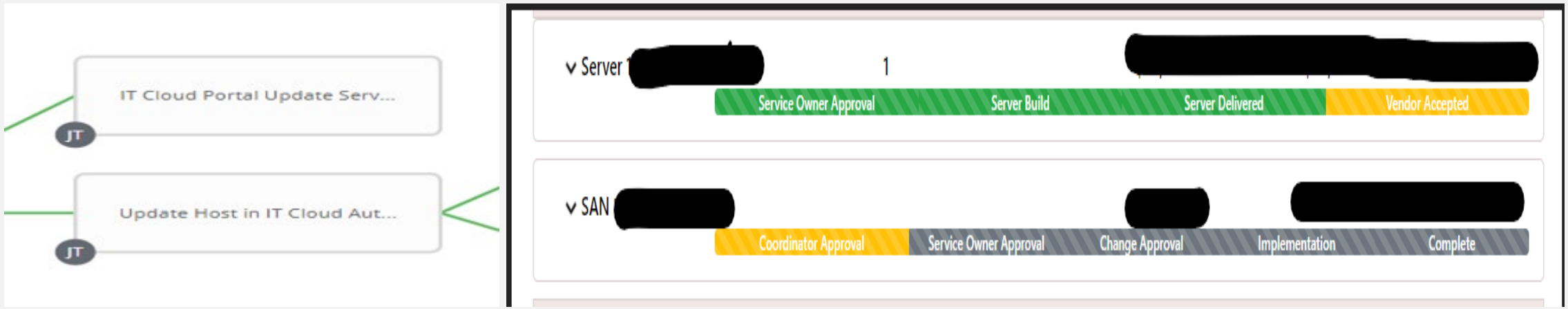
- En utilisant les variables défini dans le rôle vmw_environment, le serveur est déployé.

```yaml
- name: Clone the template
  vmware_guest:
    hostname: "{{ deploy_vcenter }}"
    username: bell\{{ vcenter_user }}
    password: "{{ vcenter_password }}"
    validate_certs: False
    name: "{{ inventory_hostname }}"
    template: "{{ deploy_datacentre }}/vm/Templates{{ (hostvars[inventory_hostname]['net
    datacenter: "{{ deploy_datacentre }}"
    folder: /
    state: poweredon
    wait_for_ip_address: yes
    datastore: "{{ vcenter_datastore }}"
    cluster: "{{ vcenter_cluster }}"
    hardware:
      memory_mb: "{{ ram }}"
      num_cpus: "{{ cpu }}"
      num_cpu_cores_per_socket: "{{ (cpu | int == 2) | ternary(2, 4) }}"
```

```yaml
    networks:
    - name: "{{ deploy_network }}"
      ip: "{{ ip }}"
      netmask: "{{ deploy_netmask }}"
      gateway: "{{ deploy_gateway }}"
      type: static
      start_connected: true
    customization:
      hostname: "{{ inventory_hostname | lower }}"
      password: "{{ guest_password }}"
      dns_servers: "{{ deploy_dns }}"
      dns_suffix: "{{ dns_suffix }}"
      domain: "{{ domain }}"
      joindomain: "{{ domain }}"
      domainadmin : "{{ (domain == 'acme.dev') | ternary(bell_fid_dev,bell_fid) }}"
      domainadminpassword : "{{ (domain == 'acme.dev') | ternary(bell_password_dev,bell_password) }}"
      orgname: "OU=<Retracted>"
      timezone: "{{ timezone }}"
    disk: "{{ vmware_extra_disks | default(omit) }}"
    convert: "{{ (vmw_disk_type | default('None') == 'thin') | ternary('thin', 'thick') }}"
    wait_for_customization: yes
  delegate_to: localhost
  register: vminfo
```


Bell

Mise a jour du portail / Inventaire



- Les informations sont ensuite envoyé au portail afin de mettre a jour le statu, ainsi que dans l'inventaire d'Ansible.

Deploiement de la deuxième carte réseau

- Ces playbooks qui roulent en parallèle ajoute une carte réseau, lorsque requis.


```yaml
- name: Deploy new network card
  vmware_guest_network:
    hostname: "{{ deploy_vcenter }}"
    username: bell\{{ vcenter_user }}
    password: "{{ vcenter_password }}"
    datacenter: "{{ deploy_datacentre }}"
    validate_certs: no
    name: "{{ inventory_hostname }}"
    folder: /Discovered virtual machine
    cluster: "{{ vcenter_cluster }}"
    networks:
    - name: "{{ deploy_network_extra }}"
      type: static
      start_connected: yes
      connected: yes
      state: new
  delegate_to: localhost
  register: vmnetinfo
  become: no
```

Deploiement de la deuxième carte réseau - Backup

- Dans le cas d'un serveur de base de donné, une carte réseau est ajouter pour le réseau de backup.

```yaml
- name: Add Backup network Windows
  include_role:
    name: win_network
  vars:
    ipam_host: "{{ inventory_hostname }}-bck"
    network_ipam_subnet: "{{ ipam_subnet_bk }}"
    network_name: "backup"
    vmw_network_name: "{{ deploy_network_bk }}"
    deploy_gateway_role: "{{ deploy_gateway_bk | default(omit) }}"
  when:
    - "'Windows' in group_names"
    - "target_os == 'Windows'"
    - "'Database' in group_names"
```

```yaml
- name: Reserve an IP Address
  include_role:
    name: itcloud.infoblox.ipam
  vars:
    hostname: "{{ ipam_host | default(inventory_hostname) }}"
    network: "{{ network_ipam_subnet }}"
    nv: "{{ ipam_view }}"
    dnsview: "{{ ipam_dns_view }}"
    state: present

- name: Configure New Network card
  include_role:
    name: vmw_network
  vars:
    deploy_network_extra: "{{ vmw_network_name }}"
```

```yaml
- name: Deploy new network card
  vmware_guest_network:
    hostname: "{{ deploy_vcenter }}"
    username: bell\{{ vcenter_user }}
    password: "{{ vcenter_password }}"
    datacenter: "{{ deploy_datacentre }}"
    validate_certs: no
    name: "{{ inventory_hostname }}"
    folder: /Discovered virtual machine
    cluster: "{{ vcenter_cluster }}"
    networks:
    - name: "{{ deploy_network_extra }}"
      type: static
      start_connected: yes
      connected: yes
      state: new
  delegate_to: localhost
  register: vmnetinfo
  become: no
```

Deploiement de la deuxième carte réseau – Front Channel

- Le deuxieme scenario pour une deuxième carte réseau est dans le cas d'un serveur DMZ. Le front channel est ajouter et des configuration OS sont faite pour mettre les nics privé et ajouter les routes requise.
- Les routes sont gardé dans des fichier de variable définie dans le rôle vmw_environment selon la localisation du serveur.

```
- name: Include vars for statics routes
  include_vars: "{{ hostvars[inventory_hostname]['dataCentre'] }}.yaml"
```

Deploiement de la deuxième carte réseau – Front Channel

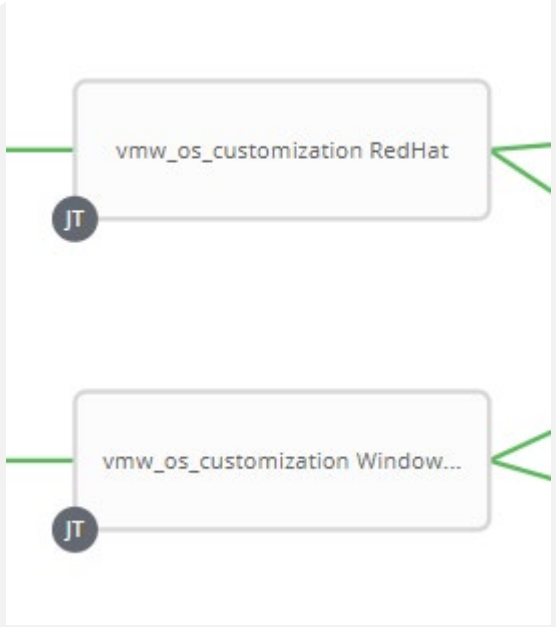
```yaml
- name: Add Frontchannel network Windows
  include_role:
    name: win_network
  vars:
    ipam_host: "{{ inventory_hostname }}-fc"
    network_ipam_subnet: "{{ ipam_subnet_fc }}"
    network_name: "frontchannel"
    vmw_network_name: "{{ deploy_network_fc }}"
    deploy_gateway_role: "{{ deploy_gateway_fc | default(omit) }}"
  when:
    - "'Windows' in group_names"
    - "target_os == 'Windows'"
    - "'DMZ' in group_names"
```

```yaml
- name: Configure the primary nic to be private
  vmware_vm_shell:
    hostname: "{{ deploy_vcenter }}"
    username: bell\{{ vcenter_user }}
    password: "{{ vcenter_password }}"
    datacenter: "{{ deploy_datacentre }}"
    cluster: "{{ vcenter_cluster }}"
    vm_id: "{{ inventory_hostname }}"
    vm_username: Administrator
    vm_password: "{{ guest_password }}"
    vm_shell: 'c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe'
    vm_shell_args: "-command (Set-NetConnectionProfile -InterfaceIndex 2 -NetworkCategory Private)"
    validate_certs: no
    wait_for_process: true
  delegate_to: localhost
```

```yaml
- name: Configure the route between Ansible and DMZ
  vmware_vm_shell:
    hostname: "{{ deploy_vcenter }}"
    username: bell\{{ vcenter_user }}
    password: "{{ vcenter_password }}"
    datacenter: "{{ deploy_datacentre }}"
    cluster: "{{ vcenter_cluster }}"
    vm_id: "{{ inventory_hostname }}"
    vm_username: Administrator
    vm_password: "{{ guest_password }}"
    vm_shell: 'c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe'
    vm_shell_args: '"-command (New-NetRoute -DestinationPrefix {{ item.destinationPr
    validate_certs: no
    wait_for_process: true
  delegate_to: localhost
  loop: "{{ dmz_route }}"
  when: '"DMZ" in group_names'
  ignore_errors: yes
```


Bell

Configuration OS




```yaml
# tasks file for roles/vmw_os_customization
- name: Include tasks to customize Red Hat Linux
  include_tasks: "RedHat.yaml"
  when:
    - "'Linux' in group_names"
    - "target_os == 'Linux'"

- name: Include tasks to customize Windows
  include_tasks: "windows.yaml"
  when:
    - "'Windows' in group_names"
    - "target_os == 'Windows'"
```

- A partir de ce point, le même playbook est rouler peux importe si le serveur est Windows ou Redhat, il charge les bonne taches selon le système d’exploitation.

Configuration OS - Suite

- Les fichier de taches windows.yaml et redhat.yaml appelle ensuite plusieurs rôles, fichiers de variables et taches selon
  - Le système d'exploitation
  - Les configurations requise
    - Par exemple si SQL doit être installé ou non et quelle version.

```yaml
- name: Install SQL when necessary
  include_role:
    name: win_sql
  when: "'Database' in group_names"
```

```yaml
# tasks file for roles/win_sql
- name: Open Firewall for SQL
  win_firewall_rule: <10 keys>

- name: Task for SQL 2014
  include_tasks: sql_2014.yaml
  when: 'dbs == "SQL-2014"'

- name: Task for SQL 2016
  include_tasks: sql_2016.yaml
  when: 'dbs == "SQL-2016"'

- name: Task for SQL 2017
  include_tasks: sql_2017.yaml
  when: 'dbs == "SQL-2017"'

- name: Task for SQL 2019
  include_tasks: sql_2019.yaml
  when: 'dbs == "SQL-2019"'
```

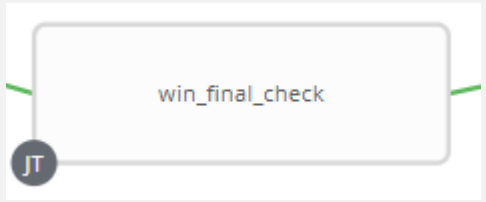
- Plusieurs agents sont installé, les groupes d'utilisateurs sont ajouté ainsi que toutes les configurations requises pour la sécurité.

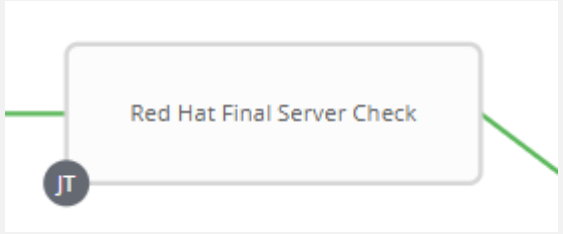
Bell

Vérification finale et retour au portail

Verification finale

- Dans ces playbooks, une vérification finale post-reboot est effectué afin de confirmer que tous les services requis roule.






```
# tasks file for roles/rhel_final_check
- name: Adding vars file for RedHat_{{ ansible_distribution_major_version }}
  include_vars: "RedHat_{{ ansible_distribution_major_version }}.yaml"

- name: Ensure required services are started and enabled
  service:
    name: "{{ item }}"
    state: started
    enabled: yes
  loop: "{{ check_required_services }}"
```

```
- name: Import vars for service check.
  include_vars: "Services_Windows_10.yaml"
  when: '"Windows-2016" in group_names'

- name: Make sure all the required services are running based on vars file - Not DB
  win_service:
    name: "{{ item.name }}"
    start_mode: auto
    state: started
  loop: "{{ check_required_services }}"
  when: "'Database' not in group_names"

- name: Make sure all the required services are running based on vars file - DB
  win_service:
    name: "{{ item.name }}"
    start_mode: auto
    state: started
  loop: "{{ check_required_services_db }}"
  when: "'Database' in group_names"
```


Bell

Verification finale

- A la fin des taches final_checks, le statu finale est envoyé au portail afin que celui-ci envoi le statu au client et au outils d'inventaire.

```yaml
- name: Final check
  block:
    - name: Run Windows final check
      include_role:
        name: win_final_check
  rescue:
    - name: Set status to FAILED and set error message
      set_fact:
        job_status: "FAILED"
        job_error_message: "Failed to execute final check"
  always:
    - name: Send status to Portal
      include_role:
        name: itportal_status
      vars:
        hostname: "{{ provision_host }}"
        status: "{{ job_status }}"
        type: "{{ itcloud_status_type }}"
        jobId: "{{ tower_workflow_job_id }}"
        message: "{{ job_error_message | default(omit) }}"
```


Bell

Questions ?