Automation of VM Creation

# About - Brief workflow

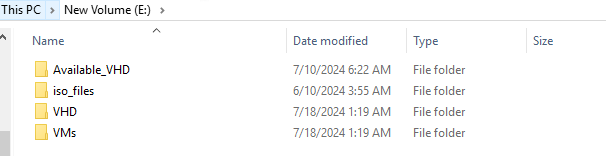
This project is intended to automate the monotonous process of VM creation on Hyper-V. It's intent is to make both developer and DevOps Engineer's life easy. Developer Initiates Request by raising a ticket for a new VM on Azure DevOps (ADO). This triggers underlying Jenkins Pipeline which is heart of this project, which ultimately execute some ansible playbooks. The VM Creation process is entirely automated and streamlined using jenkins pipeline , except an approval step. If request is Approved the VM will be created on base machine. The developer will receive a Build Status email conveying whether the VM was created, and the details of created VM.

# Getting Started

This project has some assumptions which are required to harness it’s use. Currently this project supports VM creation with following Operating Systems:

* Windows
* Ubuntu

Now let’s unwrap the assumptions:

* Firstly you need a base machine, with Hyper-V configured. Also you need to create a directory with a folder structure to place your VM’s and VHD’s. For reference you could use this:  
  
* Now what more you need is a local machine where all the code is actually going to run. This is required so that locally the git fetch and checkout can be taken and also to run Jenkins. On this machine you need to install following dependencies:
  + Ansible
  + Python3
  + Jenkins
* Thirdly, you need ADO and OIM to take input from user and sync the input fields with Jenkins parameters, end goal is to trigger jenkins pipeline with the necessary input fields. This step can be customized based on your need.

# Understanding Code and Implementation in deep

Let’s breakdown the our project code in the order of execution, better sit tight since we got a lot of ground to cover. I’ll try covering as many improvements and errors I faced, as possible:

* Jenkins: This is required to basically automate and streamline the multiple steps of this project. From taking input parameters, to halting at approval, to creating VMs using ansible, to notifying user with an Email notification.
  + Plugin Requirements:
    - Default Plugins
    - Blueocean​
    - Role-based Authorization strategy​
    - Build User Vars plugin​
    - Email Extension Plugin​
    - Gitea Plugin​
    - Extended Choice Parameter Plugin​
    - Hashicorp vault​
    - Hashicorp vault pipeline
    - Input Step Plugin

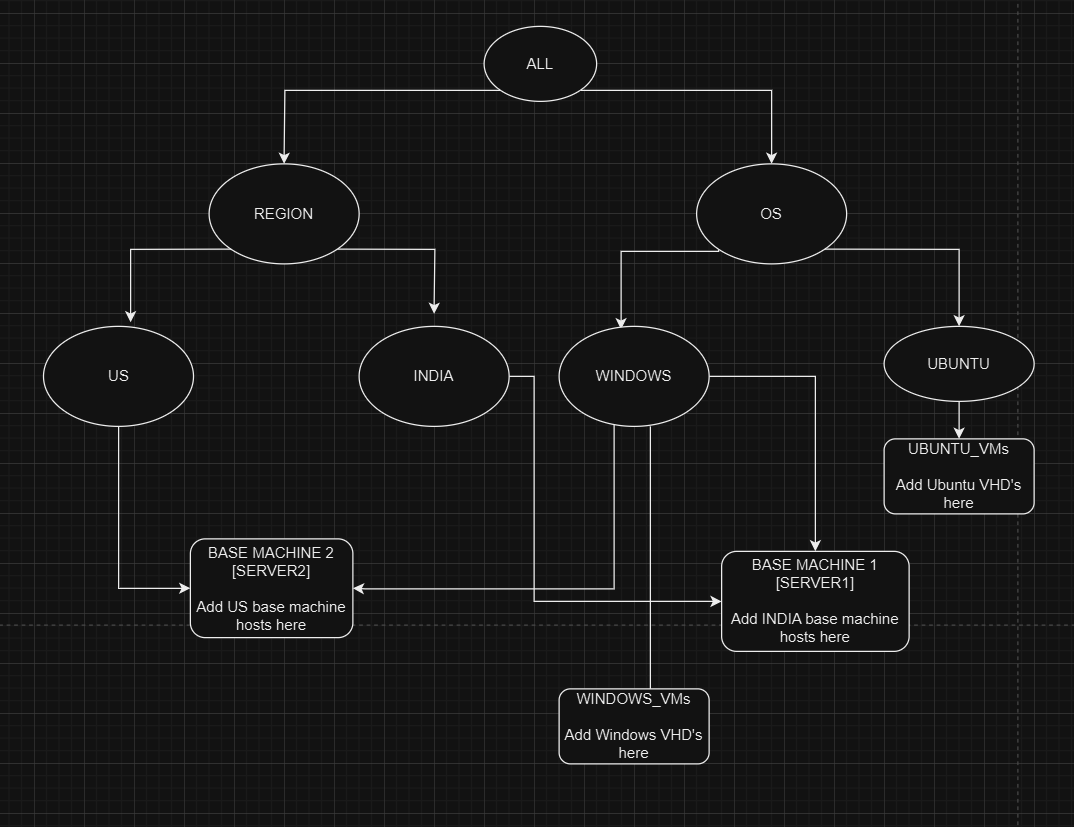
Below starts the stream which I have basically implemented in Jenkins script, consider each as a stage in our flow:

* **Stage1** - **Input Parameters**: We need following parameters from user
  + VM Name [string, required]
  + Region [lookup]
  + Purpose [lookup]
  + Cores [lookup]
  + RAM [lookup]
  + Email [string, required]

[NOTE] “VM Name” is mandatory field, and it must be unique i.e. there must not be any other VM with same name on the base machine. “Email” field is required to notify user. Also I am removing old “error.log” file , if any, so that user doesn’t receive any false errors in email notification. RAM and CPU usage errors will be sent through “error.log”.

* Stage2 - **Scan for IP:** This step scans in a IP range, navigate inside “ScanIP” directory, you’ll see three files for now, our focus is scan\_ip\_range\_\* files. If you carefully read the code it basically pings each machine in the private network of our local machine, so ensure the base machine is pingable. Make sure to double-check the range provided for scanning IP. This code picks the first IP available in the range and returns to jenkins stage. Currently we have two regions INDIA and US, so two subnets, so two files.
  + env.IP variable is set, which is basically an environment variable. Here I want to clarify variable naming convention in jenkins.
    - “def” => local variable, only scoped till the stage.
    - Without “def” => global variable, accessible anywhere in Jenkinsfile.
    - Environment variables => many ways to set, most helpful one when it comes to set variables that can be used cross files in whole Jenkins workspace. One way is “env.VARIABLE\_NAME”.
  + If no IP is found in the provided range, Jenkins execution will halt here for an IP, which only only be approved by “USERS\_CAN\_APPROV”. No email notification is yet implemented for this case.
  + IMPROVEMENTS:
    - Use arp-scan rather than ping, as it works on one layer below ping, and it even bypass firewall and sleep state of machine.
    - The above two files can be merged into one if “subnet” is passed as argument to bash script.
  + ERRORS:
    - If two pipeline jobs are running simultaneously, or even if one of the VM is still under the process of completion, same IP can be scanned by this script and provided for the other VM.
* Stage3 - **Request Approval:** This step is basically a halt, and a kind of manual intervention which is quite important when it comes to automated workflows. The Jenkins Pipeline basically waits for DevOps Engineer to provide an input either to i). Approve ii). Abort and iii). Manual IP. If approved build continues, or it aborts.
  + Now I have put a timeout which ensures that code doesn’t get halted forever, by default it will get aborted after “t” hours. So this stage is timed. Update the timeout using global variable “INP\_TIMEOUT”.
  + An email gets sent to DevOps Engineer [ a list of emails, so that multiple people can approve ]. You can add more email addresses in global variable “APPROV\_EMAIL\_LIST”. Also you can add jenkins user using global variable “USERS\_CAN\_APPROVE”, for now only administer users that can approve are “admin” and “kinjal”.
  + Now the input windows consists of three parameters, which will get submitted in Jenkins pipeline as a dictionary [array of key value pairs]. “Id” field uniquely identifies the input window, can be treated as an API endpoint. The Manual IP field gets through careful IP check, if provided.
* Stage4 - Local API Hosting
  + We have to host a server that actively listens for API requests, so on the local machine , it will get forcibly [in the sense, it will kill any current running process at the port] hosted at port 5000.
  + A python script calls two methods which enables sending POST request to the specific URL we want to hit to, this bypasses the dependency of local machine for Approval.
* Stage5 - Create VM
  + This stage uses Jenkins credentials to bring out the ansible-vault password which I have stored in HashiCorp-Vault. This is a best practice to store the anisble-vault pass in some third party Vault and dynamically fetch the password during pipeline execution.
  + The password decrypts “Hosts\_sensitive\_config.ini” file, and appends it to the “hosts.ini”. This step is crucial , since through my findings inventory in ansible follows a certain group dependency, which begins with “ALL” by default, and that does not permit defining hosts for a group in a separate inventory file. In simple words the dependency tree needs to be complete in order to reach from ALL [ consider root] to particular hosts [ consider branches].
  + Now for the entire runtime, we have ‘hosts.ini’ as the entire inventory file, which is basically a sample, since the IP’s of the created VM’s are dynamically fetched through their base machines. So you only need to store IP’s of base machines in inventory.
* Post Build Stage
  + This stage has one job , to notify user about the build status, or any errors [ if failed due to high RAM or CPU usage, this is the only job of ‘error.log’ file ], or a message [if in case VM creation request was rejected].
  + IMPROVEMENT : Currently the same template is used for both approval email and post build email, we need to customize them differently so different templates.

# SOME KEY CONCEPTS

* **Ansible-Vault :** It basically encrypts either a text or complete file or even multiple files with a vault key, or say a password. This way if any sensitive information needs to be stored in your git workspace, that can be presented in encrypted form. I simply have created one vault which encrypts one file “Hosts\_sensitive\_config.ini” , which consists VM/Base machine credentials.
* **Ansible-Roles :** These are not complete playbooks, but tasks which can be placed separately to enhance reusability and maintainability. They can be imported statically or dynamically during runtime in other ansible playbooks.
* **check\_ip.sh :** This file inside ScanIP directory takes an IP as input, and returns either “1” [ if ip is available] or “0” [ ip unavailable].
* **Inventory:** I can show the dependency tree of my Inventory file, remember this dependency tree will get created during runtime, after appends operation gets performed.  
  
* **Ansible-playbooks :** 
  + Order of Execution of yaml files =>
    - vm\_create\_config.yaml
    - CreateVM.yaml
    - FetchIP.yaml
    - AssignIP\_OS.yaml