አማራ ባንክ

AMHARA BANK

ከባንክ ባሻገር!

Beyond Financing!

+

CPU AUTOMIZATION RUN BOOK

Contents

Introduction

CPU automatic extension using Terraform and Python. This powerful combination allows for automated scaling of CPU resources in order to optimize performance and efficiency.

By leveraging Terraform's infrastructure-as-code capabilities and Python's scripting capabilities, we can develop a solution that dynamically adjusts CPU capacity based on workload demands. This approach ensures cost-effectiveness and enables your infrastructure to scale seamlessly as your application's computing needs fluctuate.

Purpose

The purpose of integrating CPU automatic extension using Terraform and Python is to create a scalable and efficient infrastructure that dynamically adjusts CPU resources based on real-time workload demands. This automation ensures optimal performance, reduces costs by avoiding overprovisioning, and enables seamless scalability for applications and services. By combining Terraform's infrastructure-as-code capabilities with Python's scripting capabilities, we can achieve a streamlined and automated approach to managing CPU resources, enhancing overall productivity and resource utilization.

Scope

When implementing CPU automatic extension using Terraform and Python in a VMware vSphere environment, the scope includes the following considerations:

vSphere Resource Management: The scope involves leveraging Terraform to provision and manage virtual machines (VMs) and their CPU resources within the vSphere environment.

Workload Monitoring: The scope includes developing BASH scripts to monitor the workload demands on VMs, including CPU utilization and performance metrics, in real-time.

Threshold Analysis: The scope encompasses setting up predefined thresholds or rules to determine when CPU resources should be automatically scaled up or down based on the observed workload demands.

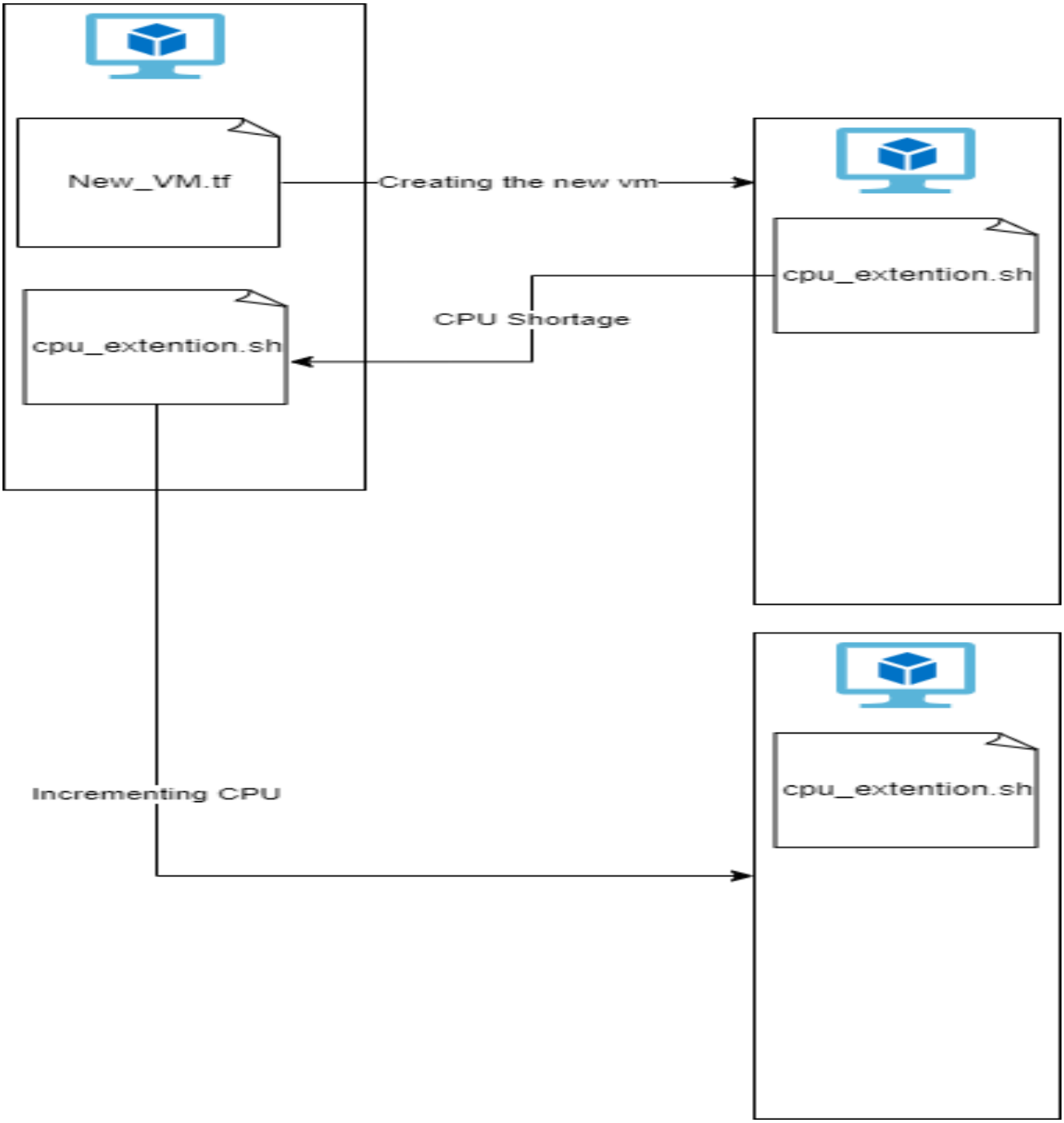
Dynamic CPU Scaling: The scope involves utilizing Terraform and Python to automate the adjustment of CPU resources for VMs in the vSphere environment.

CPU Automation using Terraform and Bash

In this method we use Terraform to define the VM and bash for editing the terraform definition and apply it again.

Limitation: we can enable hot swapping of CPU but we cannot use terraform apply to hot swap since terraform is immutable.

Overall view



1st Creating a VM with the necessary setup that enable it to make ssh request back to the master VM requesting additional CPU core.

```
data "vsphere_virtual_machine" "Redhat9Template"{
   name="Redhat9Template"
    datacenter_id = data.vsphere_datacenter.datacenter.id
}
resource "vsphere_virtual_machine" "CPU-EX-VM" {
 name             = "CPU-EX-VM"
 resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
 datastore_id     = data.vsphere_datastore.datastore.id
 cpu_hot_add_enabled=true
 cpu_hot_remove_enabled=true
 num_cpus=3
 memory           = 8024
 guest_id         = data.vsphere_virtual_machine.Redhat9Template.guest_id
 scsi_type        = data.vsphere_virtual_machine.Redhat9Template.scsi_type
 firmware = data.vsphere_virtual_machine.Redhat9Template.firmware
 network_interface {
  network_id   = data.vsphere_network.network.id
  adapter_type = data.vsphere_virtual_machine.Redhat9Template.network_interface_types[0]
 }
  clone {
  template_uuid = data.vsphere_virtual_machine.Redhat9Template.id
 }
 disk {
  label            = "disk0"
  size=54
  thin_provisioned = data.vsphere_virtual_machine.Redhat9Template.disks.0.thin_provisioned
 }
```

Here's a breakdown of what each line does in the provided Terraform code:

- data "vsphere_virtual_machine" "Redhat9Template": This line defines a data block to retrieve information about the existing Redhat9Template virtual machine in the vSphere environment.
- name="Redhat9Template": This line specifies the name of the data block, which is used for reference and retrieval in other parts of the code.
- resource "vsphere_virtual_machine" "CPU-EX-VM": This line begins the definition of a new virtual machine resource named "CPU-EX-VM" to be created in the vSphere environment.
- name = "CPU-EX-VM": This line specifies the name of the new virtual machine resource.
- resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id: This line sets the resource pool ID for the new virtual machine, which determines the allocation of CPU and memory resources.
- datastore_id = data.vsphere_datastore.datastore.id: This line specifies the datastore ID where the virtual machine's files will be stored.
- cpu_hot_add_enabled = true: This line enables hot adding of CPU resources to the virtual machine, allowing for dynamic scaling of CPU capacity.
- cpu_hot_remove_enabled = true: This line enables hot removal of CPU resources from the virtual machine, allowing for dynamic scaling of CPU capacity.
- num_cpus = 3: This line sets the number of virtual CPUs allocated to the virtual machine.
- memory = 8024: This line specifies the amount of memory (in MB) allocated to the virtual machine.
- guest_id = data.vsphere_virtual_machine.Redhat9Template.guest_id: This line sets the guest operating system type for the virtual machine based on the retrieved data.
- scsi_type = data.vsphere_virtual_machine.Redhat9Template.scsi_type: This line specifies the SCSI controller type for the virtual machine based on the retrieved data.
- firmware = data.vsphere_virtual_machine.Redhat9Template.firmware: This line sets the firmware type for the virtual machine based on the retrieved data.
- network_interface { ... }: This block defines the network interface configuration for the virtual machine, including the network ID and adapter type.
- clone { ... }: This block specifies the cloning configuration for the virtual machine, including the template UUID to be used for cloning.
- disk { ... }: This block defines a disk configuration for the virtual machine, including the label, size (in GB), and whether it is thin provisioned based on the retrieved data.

Each line contributes to the overall definition and configuration of the "CPU-EX-VM" virtual machine, including its resource allocation, network interface, cloning settings, and disk

```
 connection{
  type="ssh"
  user="root"
  password="Test@123!"
  host=vsphere_virtual_machine.CPU-EX-VM.guest_ip_addresses[0]
 }
 provisioner "remote-exec"{
  inline=[
```

```
    "echo '#!/bin/bash' >> /cpu_extension.sh",
    "echo '       threshold=100' >> /cpu_extension.sh",
    "echo 'cpu_usage=$(top -bn1 | awk '\"'\"'/^%Cpu/ {print $2}'\"'\"' | cut -d. -f1)' >> /cpu_extension.sh",
    "echo 'if [[ $cpu_usage -gt $threshold ]]; then' >> /cpu_extension.sh",
    "echo \"               sshpass -p 'Test@123!@' ssh -o StrictHostKeyChecking=no 'root@192.168.2.195' 'cd
/home/aba/Downloads/Terraform/Virtual-Machines/CPU-EX-VM/   && bash cpu_extension.sh'\" >>
/cpu_extension.sh",
    "echo '       fi' >> /cpu_extension.sh",
    "echo '*/30 * * * * /bin/bash /cpu_extension.sh'  | crontab -",
    "subscription-manager register --username --password auto-attach",
    "yum install -y sshpass"
    ]
  }
}
```

Creating SSH connection and creating bash file for checking CPU status and make ssh connection back to the Master VM this file will be scheduled using crontab.

- connection { ... }: This block defines the SSH connection settings for executing remote commands on the virtual machine.
- type = "ssh": This line specifies the connection type as SSH.
- user = "root": This line sets the username for the SSH connection as "root".
- password = "Test@123!": This line specifies the password for the SSH connection.
- host = vsphere_virtual_machine.CPU-EX-VM.guest_ip_addresses[0]: This line sets the IP address of the virtual machine as the host for the SSH connection.
- provisioner "remote-exec" { ... }: This block defines the provisioner settings for remote command execution on the virtual machine.
- inline = [...]: This line begins a list of inline commands to be executed on the remote virtual machine.
- "echo '#!/bin/bash' >> /cpu_extension.sh": This line appends a shebang and creates a shell script file named "cpu_extension.sh" in the root directory of the virtual machine.
- "echo ' threshold=100' >> /cpu_extension.sh": This line adds a variable called "threshold" with a value of 100 to the "cpu_extension.sh" script.
- "echo 'cpu_usage=$(top -bn1 | awk \"/^%Cpu/ {print $2}\" | cut -d. -f1)' >> /cpu_extension.sh": This line calculates the CPU usage percentage and assigns it to the "cpu_usage" variable in the "cpu_extension.sh" script.
- "echo 'if [[$cpu_usage -gt $threshold]]; then' >> /cpu_extension.sh": This line adds a conditional statement in the "cpu_extension.sh" script to check if the CPU usage is above the threshold.
- "echo \"sshpass -p 'Test@123!@' ssh -o StrictHostKeyChecking=no 'root@192.168.2.195' 'cd /home/aba/Downloads/Terraform/Virtual-Machines/CPU-EX-VM/   && bash cpu_extension.sh'\" >> /cpu_extension.sh": This line appends a command to the "cpu_extension.sh" script that SSHes into another machine (192.168.2.195) and executes the "cpu_extension.sh" script remotely.

- "echo 'fi' >> /cpu_extension.sh": This line adds the closing keyword "fi" to the conditional statement in the "cpu_extension.sh" script.
- "echo '*/30 * * * * /bin/bash /cpu_extension.sh' | crontab -": This line sets up a cron job to execute the "cpu_extension.sh" script every 30 minutes.
- "subscription-manager register --username --password --auto-attach": This line registers the virtual machine with Red Hat Subscription Manager using the provided username and password, and enables automatic attachment.
- "yum install -y sshpass": This line installs the "sshpass" package using the package manager "yum".

Clear Bash Script

```
#!/bin/bash


# Define the threshold percentage for CPU shortage
threshold=80


# Get the current CPU usage percentage
cpu_usage=$(top -bn1 | awk '/^%Cpu/ {print $2}' | cut -d. -f1)


# Check if CPU usage is above the threshold
if [[ $cpu_usage -gt $threshold ]]; then
    sshpass -p 'Test@123!@' ssh -o StrictHostKeyChecking=no 'root@192.168.2.195' 'cd /home/aba/Downloads/Terraform/Virtual-Machines/CPU-EX-VM/   && bash cpu_extension.sh'
fi
```

The BASH script written on the newly created vm.

- #!/bin/bash: This line is called a shebang and specifies the interpreter to use, in this case, the Bash shell.
- threshold=80: This line sets the threshold percentage for CPU shortage to 80%. You can adjust this value as needed.
- cpu_usage=$(top -bn1 | awk '/^%Cpu/ {print $2}' | cut -d. -f1): This line retrieves the current CPU usage percentage by executing the top command, filtering the output to find the line starting with "%Cpu", extracting the second field (the CPU usage percentage), and removing any decimal places.
- if [[$cpu_usage -gt $threshold]]; then: This line starts an if statement to check if the CPU usage is greater than the defined threshold.
- sshpass -p 'Test@123!@' ssh -o StrictHostKeyChecking=no 'root@192.168.2.195' 'cd /home/aba/Downloads/Terraform/Virtual-Machines/CPU-EX-VM/   && bash cpu_extension.sh': This line contains the command to be executed if the CPU usage exceeds the threshold. It connects to the remote machine (192.168.2.195) as the "root" user using SSH and runs the "cpu_extension.sh" script located in the specified directory.
- fi: This line marks the end of the if statement.

2nd Creating cpu_extenstion.sh file inside the folder the VM is created in. this file will open the terraform file used to define the VM and change(Increment) the CPU amount and apply the updated definition.

```
#!/bin/bash

# Set the file path and the amount to increment the disk value

FILE_PATH="$(pwd)/$(basename "$PWD").tf"

INCREMENT_AMOUNT=1


# Check if the file exists

if [ ! -f "$FILE_PATH" ]; then

    echo "File $FILE_PATH does not exist."

    exit 1

fi

# Extract the current disk value

CURRENT_VALUE=$(grep -oP '(?<=num_cpus=)\d+' "$FILE_PATH")


# Calculate the updated value

UPDATED_VALUE=$((CURRENT_VALUE + INCREMENT_AMOUNT))

# Search for the pattern "size=" in the file and update the value

sed -r "s/(num_cpus=)([0-9]+)/\1$UPDATED_VALUE/" "$FILE_PATH" > "$FILE_PATH.tmp"

# Rename the temporary file to the original file

mv "$FILE_PATH.tmp" "$FILE_PATH"

"$(terraform apply -auto-approve)"
```

The provided shell script performs the following actions:

- FILE_PATH="$(pwd)/$(basename "$PWD").tf": This line sets the FILE_PATH variable to the current working directory concatenated with the base name of the directory and the file extension .tf. This assumes that the script is executed in the directory containing the Terraform file.
- INCREMENT_AMOUNT=1: This line sets the INCREMENT_AMOUNT variable to the desired value by which the num_cpus parameter in the Terraform file will be incremented.
- if [! -f "$FILE_PATH"]; then ... fi: This block checks if the Terraform file exists. If the file does not exist, it prints an error message and exits with a status code of 1.
- CURRENT_VALUE=$(grep -oP '(?<=num_cpus=)\d+' "$FILE_PATH"): This line extracts the current value of the num_cpus parameter from the Terraform file using grep with a regular expression pattern.
- UPDATED_VALUE=$((CURRENT_VALUE + INCREMENT_AMOUNT)): This line calculates the updated value of num_cpus by adding the INCREMENT_AMOUNT to the CURRENT_VALUE.

- sed -r "s/(num_cpus=)([0-9]+)/\1$UPDATED_VALUE/" "$FILE_PATH" > "$FILE_PATH.tmp": This line uses sed to search for the pattern num_cpus= followed by a number in the Terraform file and replaces it with the updated value. The modified content is redirected to a temporary file.
- mv "$FILE_PATH.tmp" "$FILE_PATH": This line renames the temporary file to replace the original Terraform file with the updated value.
- "$(terraform apply -auto-approve)": This line executes the terraform apply command with the -auto-approve flag to automatically apply the changes in the Terraform file.