Trabalho prático

- **Aluno**: Guilherme Rodriguez Vicentin
- **E-mail**: vicentingr@gmail.com
- **Link do projeto**: https://github.com/gvicentin/inf0500/inf0550

1. Introdução

Para a realização da atividade prática da disciplina de **Computação em Nuvem I**, o provedor de Cloud escolhido foi a Microsoft Azure.

Utilizando o e-mail institucional da Unicamp é possível adquirir *U$ 100* de crédito com expiração após um ano.

Todo o código e os arquivos necessários para reproduzir o laboratório aqui apresentado, pode ser encontrado no GitHub da disciplina.

2. Configuração do sistema

Para criar e configurar as VMs de forma automatizada, iremos utilizar o HashiCorp Terraform.

Usando Infrastructure as Code (IaC), é póssivel experimentar mais e reduzir os custos, uma vez que podemos criar e remover os recursos da cloud sem intervenção manual, a qualquer momento. Além disso, não precisaremos manualmente configurar e rodar os testes de benchmark.

2.1. Configurando a Autenticação

Primeiramente iremos realizar o login na Azure utilizando a CLI, assim poderemos autenticar o provider do Terraform posteriormente. Abaixo segue os passos tomados:

1. Instalar a Azure CLI.
2. Rodar o comando `az login`. Uma nova janela será aberta no navegardor para ser realizado o login com um conta Microsoft.

Agora, criaremos um par de chaves SSH diretamente na Azure e salvaremos localmente a chave privada.

1. `ssh_key_name`: Escolhemos aleatóriamente um nome para o recurso.
2. `ssh_public_key_gen`: Inicializamos a ação que criará a um par de chaves.
3. `ssh_public_key`: Fazemos o **bind** entre a ação que cria uma chave e o recurso em si.
4. `private_key_file`: Salvamos o resultado da chave privada em um arquivo local.

Arquivo `infra/ssh.tf`:

```
resource "random_pet" "ssh_key_name" {
  prefix    = "ssh"
  separator = ""
}

resource "azapi_resource_action" "ssh_public_key_gen" {
  type        = "Microsoft.Compute/sshPublicKeys@2022-11-01"
  resource_id = azapi_resource.ssh_public_key.id
  action      = "generateKeyPair"
  method      = "POST"

  response_export_values = ["publicKey", "privateKey"]
}

resource "azapi_resource" "ssh_public_key" {
  type      = "Microsoft.Compute/sshPublicKeys@2022-11-01"
  name      = random_pet.ssh_key_name.id
  location  = azurerm_resource_group.rg.location
  parent_id = azurerm_resource_group.rg.id
}

resource "local_sensitive_file" "private_key_file" {
  content         = azapi_resource_action.ssh_public_key_gen.output.privateKey
  filename        = "${path.module}/id_azure"
  file_permission = "0600"
}
```

2.2. Criando recursos de Rede

Antes de criarmos nossas VMs, precisamos criar alguns recursos de rede como Virtual Network e Subnet.

1. Virtual Network com um CIDR block primário de classe A (10.0.0.0/16).
2. Subnet utilizando o CIDR 10.0.1/24 (256 hosts máximos).

Arquivo `infra/main.tf`:

```
# Create virtual network
resource "azurerm_virtual_network" "my_terraform_network" {
  name                = "myVnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

# Create subnet
resource "azurerm_subnet" "my_terraform_subnet" {
```

```
  name                = "mySubnet"
  resource_group_name = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.my_terraform_network.name
  address_prefixes    = ["10.0.1.0/24"]
}
```

2.3. Criando as VMs

Para cada VM, criaremos os seguintes recursos:

1. IP público que será utilizado para acessar a máquina utilizando SSH.
2. Security Group, onde serão especificadas as regras de Firewall a nível da instância (liberar porta 22 de qualquer IP).
3. NIC (Interface de rede) que será atribuida a instância.
4. Bind entre o Security Group (criado no item 2) e a interface de rede (criada no item 3).
5. Virtual Machine propriamente dita. Os recursos criados nos itens anteriores serão utilizados aqui.

Arquivo `infra/main.tf`:

```
# Create public IPs
resource "azurerm_public_ip" "my_terraform_public_ip" {
  name                = "myPublicIP"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Dynamic"
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "my_terraform_nsg" {
  name                = "myNetworkSecurityGroup"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    name                       = "SSH"
    priority                   = 1001
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}
```

```
# Create network interface
resource "azurerm_network_interface" "my_terraform_nic" {
  name                = "myNIC"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "my_nic_configuration"
    subnet_id                     = azurerm_subnet.my_terraform_subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.my_terraform_public_ip.id
  }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "example" {
  network_interface_id      = azurerm_network_interface.my_terraform_nic.id
  network_security_group_id = azurerm_network_security_group.my_terraform_nsg.id
}

# Create virtual machine
resource "azurerm_linux_virtual_machine" "my_terraform_vm" {
  name                  = "myVM"
  location              = azurerm_resource_group.rg.location
  resource_group_name   = azurerm_resource_group.rg.name
  network_interface_ids = [azurerm_network_interface.my_terraform_nic.id]
  size                  = "Standard_DS1_v2"

  os_disk {
    name                 = "myOsDisk"
    caching              = "ReadWrite"
    storage_account_type = "Premium_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts-gen2"
    version   = "latest"
  }

  computer_name  = "hostname"
  admin_username = var.username

  admin_ssh_key {
```

```
    username   = var.username
    public_key = azapi_resource_action.ssh_public_key_gen.output.publicKey
  }

  boot_diagnostics {
    storage_account_uri = azurerm_storage_account.my_storage_account.primary_blob_endpoint
  }
}
```

2.4 Criando a infraestrutura

Basta rodarmos os seguintes comandos do Terraform para criar todos os recursos:

```
terraform init -upgrade
terraform plan -out main.tfplan
terraform apply main.tfplan
```

Depois de completar a aplicação do Terraform, podemos consultar os IP públicos criados utilizando o seguinte comando:

```
$ terraform output

hostnames = [
  "vm-general-c1m3",
  "vm-general-c2m7",
  "vm-compute-c2m4",
]
public_ip_addresses = [
  "40.71.41.40",
  "52.170.99.228",
  "52.224.124.159",
]
key_data = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDYFQ+lwmmurDZwRgNP1hnpUI+mlnnXuq+28lk04lie
resource_group_name = "rg-arriving-pipefish"
```

Para remover todos os recursos de uma única vez:

```
terraform destroy main.tfplan
```

3. Benchmarks escolhidos

Neste trabalho prático optaremos por realizar benchmarks relacionados a CPU e processamento.

Todos os provedores de Cloud disponibilizam diversas famílias de instâncias para diferentes propósitos. Por exemplo, na Azure temos as seguintes famílias de VMs:

1. *General purpose*

2. *Compute optimized*
3. *Memory optimized*
4. *Storage optimized*
5. *GPU accelerated compute*
6. Entre outras

Os tamanhos de VM de propósito geral fornecem uma proporção equilibrada entre CPU e memória. Ideais para testes e desenvolvimento, bancos de dados pequenos a médios e servidores web de tráfego baixo a médio.

Os tamanhos de VM otimizados para computação têm uma alta proporção de CPU para memória. Esses tamanhos são bons para servidores web de tráfego médio, dispositivos de rede, processos em lote e servidores de aplicativos.

Dentro de cada uma das famílias de VMs, temos ainda diferentes tamanhos. Por exemplo, na Azure as instâncias padrões tem a seguinte nomeclatura, onde `D<NUM_CPUS>` representa a quantidade de vCPUs:

1. *Standand_D2ds_v4*
2. *Standand_D4ds_v4*
3. *Standand_D8ds_v4*
4. *Standand_D16ds_v4*
5. *Standand_D32ds_v4*
6. Entre outras

A ídeia central deste projeto é realizar o benchmark entre instâncias de *General Purpose* e *Compute optimized*. Nossa hipótese é que uma instância *Compute optimized* com a mesma quantidade de núcleos desempenhe melhor que uma *General purpose*, pelo fato do processador ser mais avançado. Normalmente o custo das instâncias *Compute optimized* são maiores se comparados a *General purpose* para o mesmo número de núcleos.

Para realização dos testes de benchmark, a ferramenta Geekbench 6 foi escolhida. O Geekbench 6 mede o poder do seu processador em núcleo único e em múltiplos núcleos, utilizando cenários e conjuntos de dados práticos do dia a dia para medir o desempenho. Cada teste é baseado em tarefas encontradas em aplicativos populares do mundo real e utiliza conjuntos de dados realistas, garantindo que seus resultados sejam relevantes e aplicáveis.

4. Execução dos benchmarks

Para a realização dos testes de benchmark, a ferramenta ansible será utilizada. Desta forma poderemos rodar automaticamente e simultaneamente os testes em todas as máquinas ao mesmo tempo.

O primeiro passo, é definir nosso invetário no arquivo `benchmark/inventory.ini`. Os IPs públicos foram tirados do comando `terraform output`.

```
[benchmark]
vm-general-c1m3 ansible_ssh_host=40.71.41.40 ansible_ssh_user=azureadmin ansible_ssh_private
```

```
vm-general-c2m7 ansible_ssh_host=52.170.99.228 ansible_ssh_user=azureadmin ansible_ssh_priva
vm-compute-c2m4 ansible_ssh_host=52.224.124.159 ansible_ssh_user=azureadmin ansible_ssh_priv
```

A execução do benchmark consiste em fazer o *Download* das dependências e executar o software Geekbench. A seguir, no arquivo `benchmark/main.yml` descrevemos os comandos necessários:

```
- hosts: benchmark
  become: yes

  pre_tasks:
    - name: Update repositories on Ubuntu
      when: ansible_distribution == 'Ubuntu'
      apt:
        update_cache: yes
        upgrade: yes
      changed_when: False

  tasks:
    - name: Install packages on Ubuntu
      apt: name={{item}}
      with_items:
        - curl
        - openjdk-8-jdk

    - name: Create benchmark directory if it doesn't exist
      ansible.builtin.file:
        path: /benchmark
        state: directory

    - name: Download Geekbench
      command: "wget -O /benchmark/Geekbench-6.2.1-Linux.tar.gz https://cdn.geekbench.com/Ge

    - name: Extract Geekbench tarball
      ansible.builtin.unarchive:
        src: /benchmark/Geekbench-6.2.1-Linux.tar.gz
        dest: /benchmark
        remote_src: yes

    - name: Run benchmark
      shell: /benchmark/Geekbench-6.2.1-Linux/geekbench6 > /benchmark/results.txt

    - name: Fetch results
      fetch:
        src: /benchmark/results.txt
        dest: results/
```

Podemos rodar o Ansible Playbook com o comando a seguir. Após finalizado, teremos todos os resultados dos testes na nossa máquina.

```
$ ansible-playbook -i inventory.ini benchmark.yml

PLAY [benchmark] ***************************************************************

TASK [Gathering Facts] *********************************************************
ok: [vm-compute-c2m4]
ok: [vm-general-c2m7]
ok: [vm-general-c1m3]

TASK [Update repositories on Ubuntu] *******************************************
ok: [vm-compute-c2m4]
ok: [vm-general-c2m7]
ok: [vm-general-c1m3]

TASK [Install packages on Ubuntu] **********************************************
ok: [vm-general-c2m7] => (item=curl)
ok: [vm-compute-c2m4] => (item=curl)
ok: [vm-general-c1m3] => (item=curl)
changed: [vm-general-c1m3] => (item=openjdk-8-jdk)
changed: [vm-general-c2m7] => (item=openjdk-8-jdk)
changed: [vm-compute-c2m4] => (item=openjdk-8-jdk)

TASK [Create benchmark directory if it doesn't exist] **************************
changed: [vm-compute-c2m4]
changed: [vm-general-c2m7]
changed: [vm-general-c1m3]

TASK [Download Geekbench] ******************************************************
changed: [vm-general-c1m3]
changed: [vm-general-c2m7]
changed: [vm-compute-c2m4]

TASK [Extract Geekbench tarball] ***********************************************
changed: [vm-compute-c2m4]
changed: [vm-general-c2m7]
changed: [vm-general-c1m3]

TASK [Run benchmark] ***********************************************************
changed: [vm-general-c2m7]
changed: [vm-compute-c2m4]
changed: [vm-general-c1m3]

TASK [Fetch results] ***********************************************************
```

```
changed: [vm-general-c1m3]
changed: [vm-compute-c2m4]
changed: [vm-general-c2m7]

PLAY RECAP *********************************************************************
vm-compute-c2m4            : ok=8    changed=6    unreachable=0    failed=0
vm-general-c1m3            : ok=8    changed=6    unreachable=0    failed=0
vm-general-c2m7            : ok=8    changed=6    unreachable=0    failed=0
```

5. Resultados e discussão

5.1. General purpose 1 núcleos

Especificação de CPU e memória encontrados pelo Geekbench.

System Information	
Operating System	Ubuntu 22.04.4 LTS
Model	Microsoft Corporation Virtual Machine
Motherboard	Microsoft Corporation Virtual Machine

CPU Information	
Name	Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz
Topology	1 Processor, 1 Core
Identifier	GenuineIntel Family 6 Model 85 Stepping 7
Base Frequency	2.59 GHz
Cluster 1	0 Cores
L1 Instruction Cache	32.0 KB x 1
L1 Data Cache	32.0 KB x 1
L2 Cache	1.00 MB x 1
L3 Cache	35.8 MB x 1

Memory Information	
Size	3.32 GB

5.2. General purpose 2 núcleos

Especificação de CPU e memória encontrados pelo Geekbench.

System Information	
Operating System	Ubuntu 22.04.4 LTS
Model	Microsoft Corporation Virtual Machine
Motherboard	Microsoft Corporation Virtual Machine

CPU Information	
Name	Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz
Topology	1 Processor, 2 Cores
Identifier	GenuineIntel Family 6 Model 85 Stepping 7
Base Frequency	2.59 GHz
Cluster 1	0 Cores
L1 Instruction Cache	32.0 KB x 2
L1 Data Cache	32.0 KB x 2
L2 Cache	1.00 MB x 2
L3 Cache	35.8 MB x 1

Memory Information	
Size	6.76 GB

5.3. Compute optimized 2 núcleos

Especificação de CPU e memória encontrados pelo Geekbench.

System Information	
Operating System	Ubuntu 22.04.4 LTS
Model	Microsoft Corporation Virtual Machine
Motherboard	Microsoft Corporation Virtual Machine

CPU Information	
Name	Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz
Topology	1 Processor, 1 Core, 2 Threads
Identifier	GenuineIntel Family 6 Model 85 Stepping 7
Base Frequency	2.59 GHz
Cluster 1	0 Cores

CPU Information	
L1 Instruction Cache	32.0 KB x 1
L1 Data Cache	32.0 KB x 1
L2 Cache	1.00 MB x 1
L3 Cache	35.8 MB x 1

Memory Information	
Size	3.81 GB

5.4. Score finais

Comparação dos resultados obtidos no benchmark:

Resultado Final	
[General Purpose 1 core] Single-Core Score	989
[General Purpose 1 core] Multi-Core Score	985
[General Purpose 2 core] Single-Core Score	1088
[General Purpose 2 core] Multi-Core Score	2009
[Compute optimized 2 core] Single-Core Score	1142
[Compute optimized 2 core] Multi-Core Score	1374

Para consultar todos os resultados online, use os links a seguir:

- General Purpose 1 core: https://browser.geekbench.com/v6/cpu/6046933
- General Purpose 2 core: https://browser.geekbench.com/v6/cpu/6046914
- Compute Optimized 2 core: https://browser.geekbench.com/v6/cpu/6046925

6. Conclusão

Após realizar benchmarks em três instâncias de máquinas virtuais (VMs) da Azure, comparando as famílias General Purpose e Compute Optimized, podemos concluir que embora a performance da família Compute Optimized seja superior quando comparada apenas em termos de um único núcleo com a família General Purpose, outros fatores devem ser considerados ao analisar o desempenho em cenários de múltiplos núcleos.

Observamos que, apesar da vantagem em termos de poder de processamento individual, a Compute Optimized mostrou-se menos eficiente em ambientes que exigem um equilíbrio entre CPU e memória. Isso se traduziu em resultados inferiores nos testes que envolviam o uso de múltiplos núcleos, onde a capacidade de memória disponível desempenha um papel crucial.

Essa descoberta ressalta a importância de uma análise holística ao selecionar a configuração de instância de VM mais adequada para uma carga de trabalho específica. Embora o desempenho em um único núcleo possa ser um fator decisivo em certos casos, a otimização do equilíbrio entre CPU e memória pode ser crucial em cenários que exigem processamento paralelo ou multitarefa intensiva.

Portanto, ao considerar a escolha entre as famílias General Purpose e Compute Optimized, é fundamental avaliar não apenas a performance em cenários isolados, mas também as necessidades específicas da carga de trabalho e como cada configuração atende a esses requisitos de maneira abrangente. Esta abordagem permitirá uma seleção mais precisa e eficiente da instância de VM, resultando em melhor desempenho e utilização dos recursos disponíveis.