Trabalho Final da Disciplina

**Administração de Containers Docker e Kubernetes**

Prof.: Paulo César Vigne Júnior

Aluno: Alexandre Nunes Martins

**1.Significado de Contêiner, Runtime de Contêiner e Solução Docker.**

**Contêiner** é uma unidade de software que contém todos os elementos necessários em um pacote para executar um aplicativo, incluindo código, bibliotecas, dependências e configurações necessárias para executar seus serviços de software. Tem um ambiente isolado dentro de um servidor, destinado à execução de aplicativos. Os contêineres permitem que um aplicativo seja executado de forma consistente e confiável em diferentes ambientes, independentemente das diferenças de hardware e software. Eles também permitem que múltiplas instâncias do mesmo aplicativo sejam executadas em um único servidor, aumentando a eficiência e a escalabilidade. Os contêineres são frequentemente usados em conjunto com a tecnologia de virtualização e orquestração de contêineres, como o Docker e o Kubernetes.

Um **Runtime de Contêiner** é um software responsável por executar e gerenciar contêineres em um sistema operacional. Ele é responsável por criar e iniciar o ambiente isolado em que o contêiner é executado, gerenciar os recursos do sistema, como CPU, memória e rede, e garantir a comunicação adequada entre os contêineres. O Runtime de Contêiner é a parte do sistema que realmente implementa a virtualização de nível de sistema operacional necessária para criar e executar contêineres. Ele é responsável por fornecer a interface entre o sistema operacional e os contêineres, garantindo que eles possam compartilhar recursos do sistema de forma segura e eficiente. Alguns exemplos de runtimes de contêiner disponíveis são: o Docker Engine, o containerd, o CRI-O e o runc. Cada um deles com suas próprias características e funcionalidades, todos projetados para fornecer um ambiente seguro e isolado para a execução de aplicativos em contêineres.

A **Solução Docker** é uma plataforma de software que inclui várias ferramentas e tecnologias para criar, distribuir e executar aplicativos em contêineres. A Solução Docker simplifica o processo de desenvolvimento, implantação e gerenciamento de aplicativos em contêineres. Com o Docker, é possível criar ambientes de desenvolvimento, testes e produção consistentes e portáteis, reduzindo a complexidade e aumentando a eficiência do processo de desenvolvimento e implantação de aplicativos. Os três principais componentes do Docker são:

- O Docker Engine que é o Runtime de Contêiner da plataforma Docker. Ele permite criar e executar contêineres em um sistema operacional.

- O Docker Hub que é um serviço baseado em nuvem que permite compartilhar, armazenar e distribuir imagens de contêineres onde é possível usar imagens pré-fabricadas disponíveis ou criar suas próprias imagens personalizadas e disponibilizá-las no Docker Hub.

- O Docker Compose, é uma ferramenta que permite definir e executar aplicativos de vários contêineres. É usado para definir a configuração do aplicativo e os requisitos de recursos em um arquivo de manifesto YAML e, logo em seguida, pode usar esse arquivo para iniciar todos os contêineres necessários.

**2.Diferença de Virtual Machines e Contêineres.**

Virtual Machines e Contêineres são tecnologias de virtualização que fornecem ambientes isolados para a execução de aplicativos.

**Virtual Machines** (VMs) são máquinas virtuais que emulam um sistema operacional completo, incluindo o kernel, drivers e aplicativos. Cada VM é executada em um “hypervisor”, que é um software que simula a arquitetura de hardware necessária para executar uma máquina virtual. Com isso uma VM é essencialmente uma máquina dentro de outra máquina.

**Contêineres** são ambientes isolados para a execução de aplicativos que compartilham o kernel do sistema operacional hospedeiro. Cada contêiner contém apenas os componentes necessários para executar um aplicativo específico, incluindo código, bibliotecas e dependências, porém compartilha o kernel e outros recursos do sistema operacional com outros contêineres e o hospedeiro.

As principais diferenças entre Virtual Machines e Contêineres estão nos seguintes pontos de atenção:

- Isolamento: as VMs oferecem um nível mais alto de isolamento, uma vez que cada VM é uma máquina virtual completa e independente, enquanto os contêineres compartilham recursos do sistema operacional.

- Tamanho: as VMs tendem a ser maiores e mais pesadas do que os contêineres, visto que precisam incluir um sistema operacional completo, enquanto os contêineres contêm apenas os componentes necessários para executar um aplicativo.

- Desempenho: os contêineres são geralmente mais rápidos e mais eficientes do que as VMs, pois não precisam emular um sistema operacional completo.

- Portabilidade: os contêineres são mais portáteis do que as VMs, uma vez que podem ser facilmente movidos entre diferentes ambientes de execução, enquanto as VMs tendem a ser mais difíceis de mover entre plataformas e ambientes.

**3.Explicação conceitual sobre o Docker Registry e Copy-on-Write.**

**Docker Registry** é um serviço de armazenamento de imagens de contêineres que permite aos usuários compartilhar e distribuir imagens de contêineres criadas com o Docker. É possível usar o Docker Registry público como repositório, que é mantido pela Docker, Inc., ou configurar um registro privado em um servidor local ou na nuvem. O Docker Registry armazena as imagens de contêineres em um formato chamado Docker Image Manifest, que contém metadados sobre a imagem, como nome, versão, dependências e configurações de ambiente. As imagens são identificadas por um identificador de hash criptográfico exclusivo, que garante a integridade dos dados.

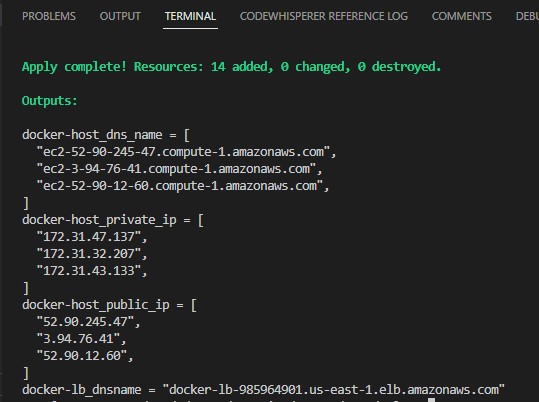
**Copy-on-Write** (CoW) é uma técnica de gerenciamento de memória usada pelo Docker para criar e gerenciar contêineres. Utilizando a técnica CoW, o Docker cria uma camada de arquivo somente de leitura que contém o sistema de arquivos do contêiner base, e adiciona camadas adicionais somente de leitura para as alterações feitas pelos contêineres que derivam a partir do contêiner base. Quando um contêiner é iniciado, o Docker cria uma camada somente gravação que pode ser modificada pelo contêiner durante a execução. Isso permite que vários contêineres compartilhem o mesmo sistema de arquivos base, reduzindo o espaço de armazenamento necessário e acelerando o processo de criação e inicialização do contêiner.

**Atividade 1:** A primeira arquitetura será de um cluster docker swarm com 3 nós sendo um master e dois workers, com o service do nginx possuindo 3 réplicas, deverá possuir health check e um mapeamento de config para o index.hml, com o código html de preferência do aluno. O Balanceamento externo ao cluster deverá ser desprezado.

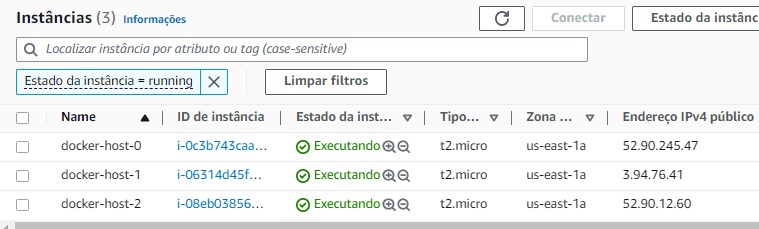
**1.1 Implementação da infraestrutura.**

Trabalho elaborado em ambiente WSL do Windows 11 e estrutura implementada através do Terraform em nuvem AWS utilizando o arquivo docker-main.tf da [*Tabela* *1*](#Tabela1).

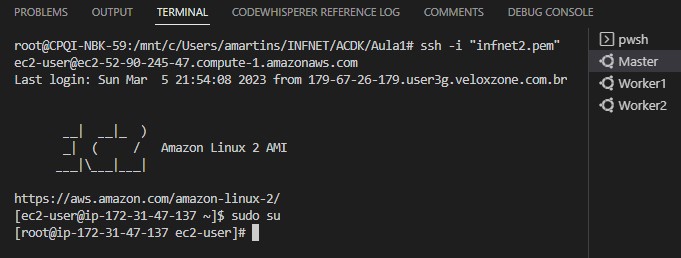
Output do Terraform:



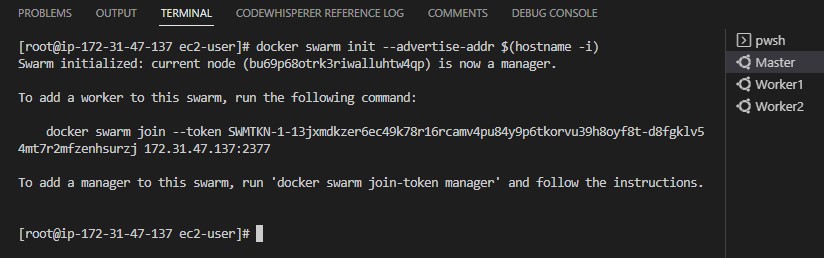
Painel EC2 AWS:



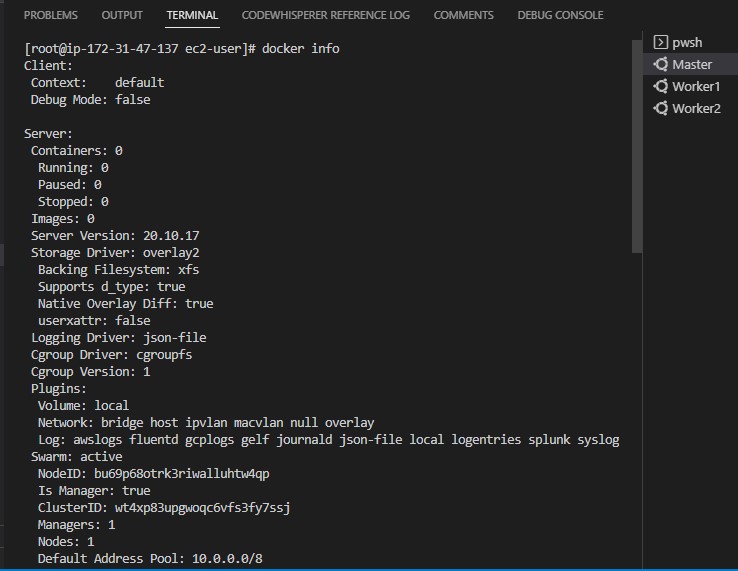
**1.2 Conectando-se via SSH ao docker-host-0(Master).**



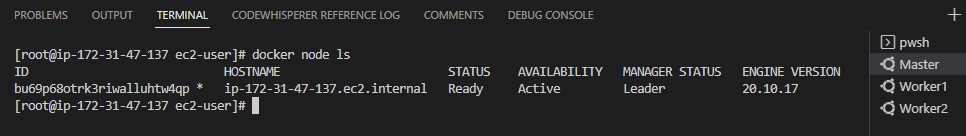
**1.3 Instalando o Manager.**



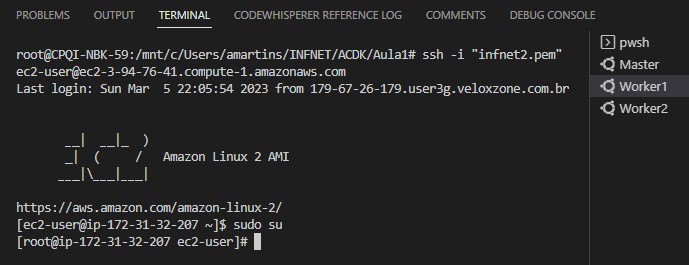
**1.4 Comando docker info.**



**1.5 Comando docker node ls.**



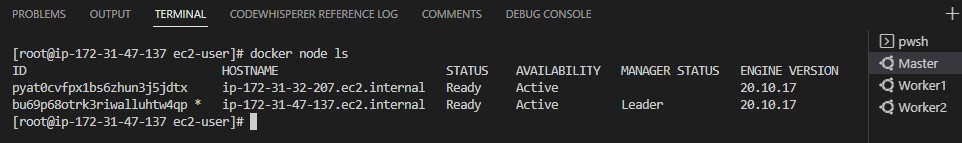
**1.6 Conectando-se via SSH ao docker-host-1(Worker1).**



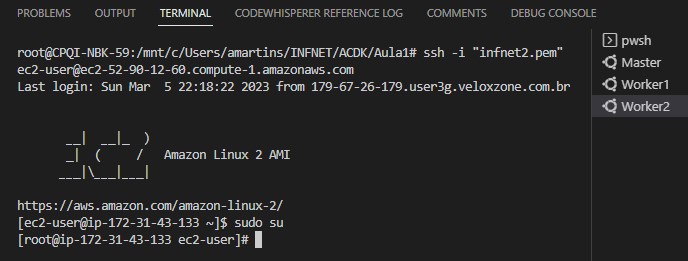
**1.7 Instalando o Worker1.**



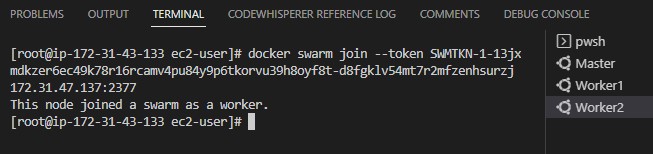
**1.8 Comando docker node ls.**



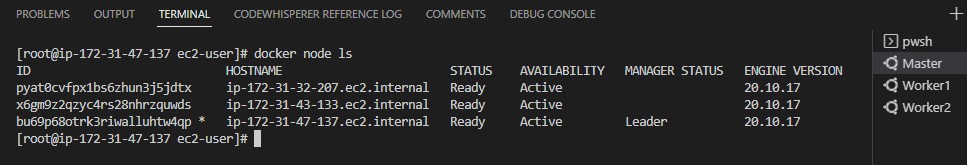
**1.9 Conectando-se via SSH ao docker-host-2(Worker2).**



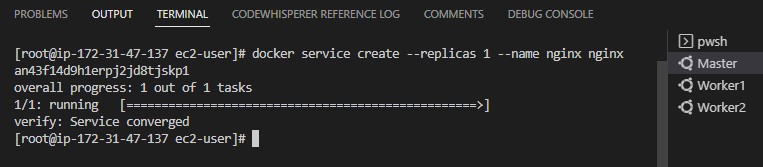
**1.10 Instalando o Worker2.**



**1.11 Comando docker node ls.**



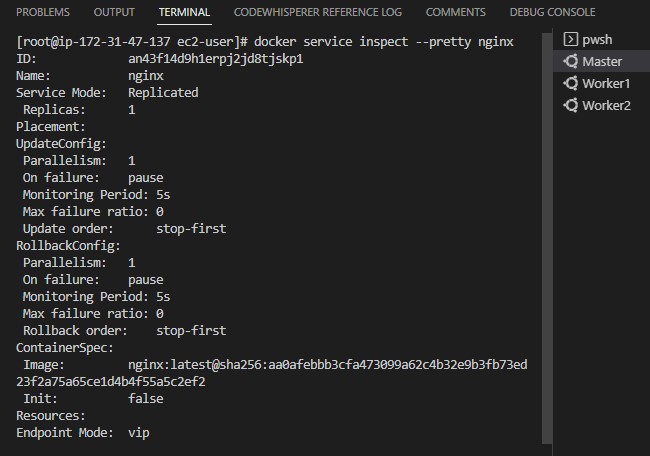
**1.12 Deploy de um Service.**



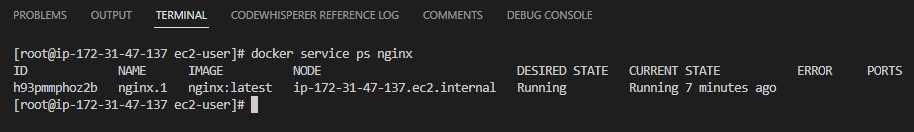
**1.13 Comando docker service ls.**



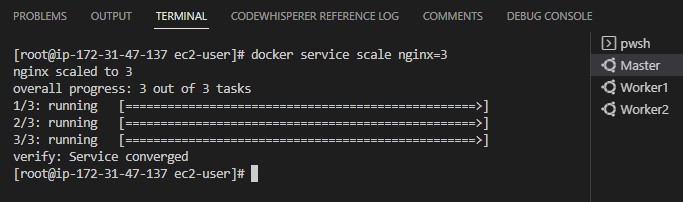
**1.14 Comando docker service inspect.**



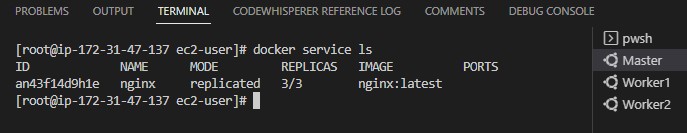
**1.15 Comando docker service ps.**



**1.16 Scale a service.**



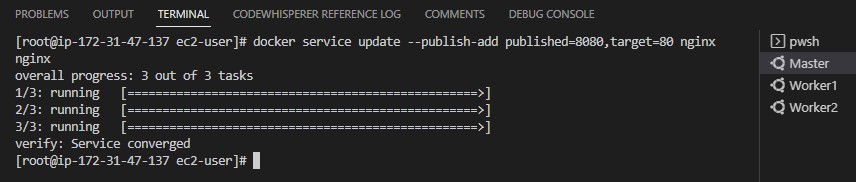
**1.17 Comando docker service ls.**



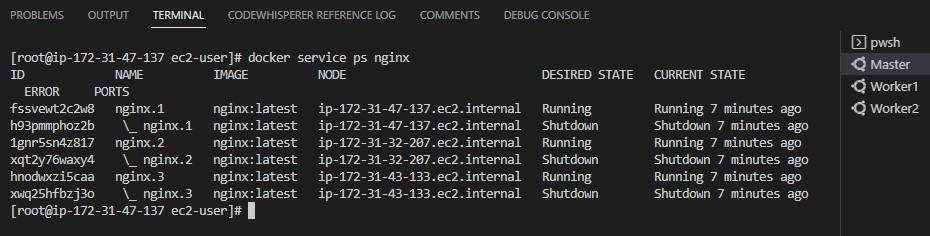
**1.18 Comando docker service ps.**



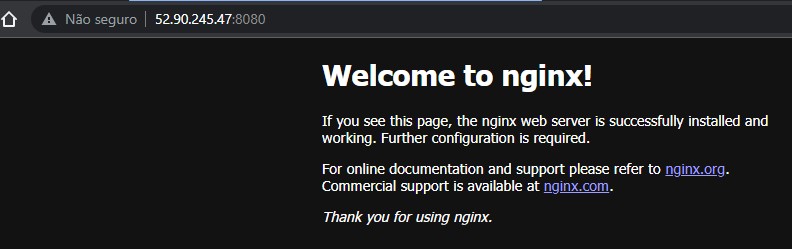
**1.19 Expor o Serviço externamente.**

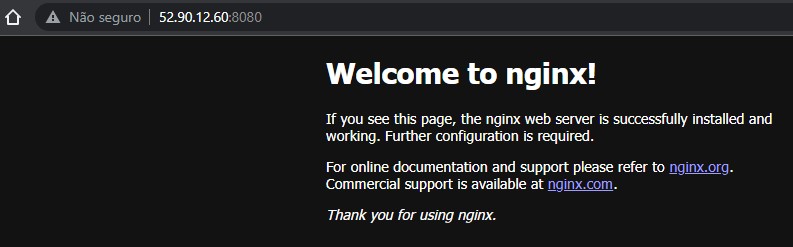
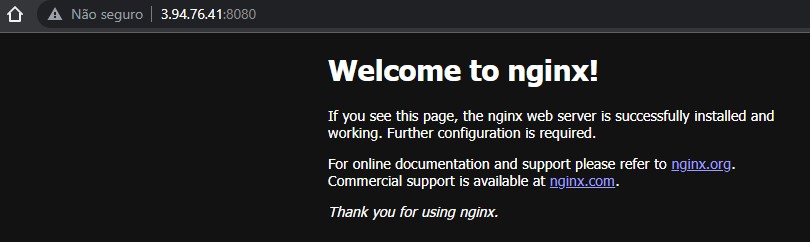


**1.20 Comando docker service ps.**

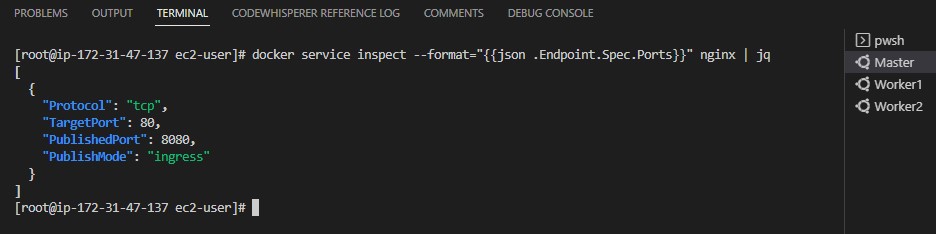


**1.21 Acessando o Serviço externamente.**

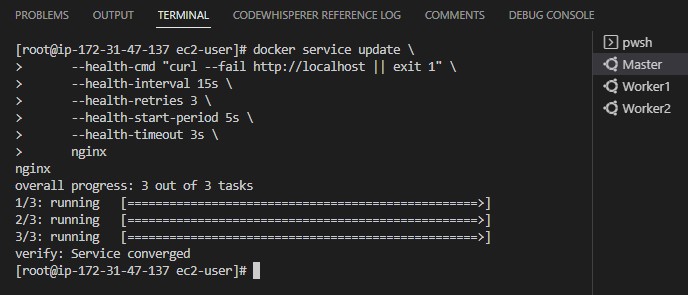




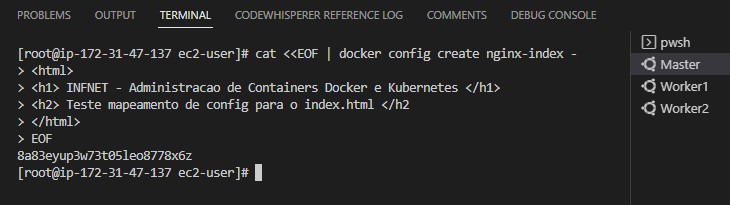
**1.22 Comando docker service inspect.**



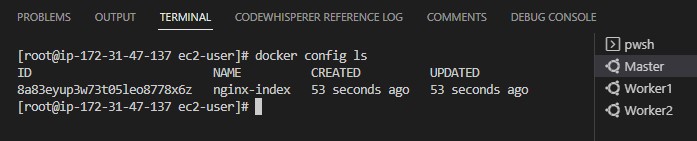
**1.23 Add Health Check.**



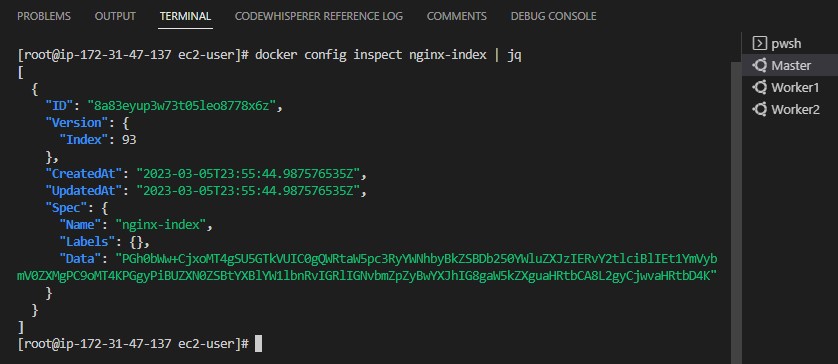
**1.24 Mapeamento de config para o index.html.**



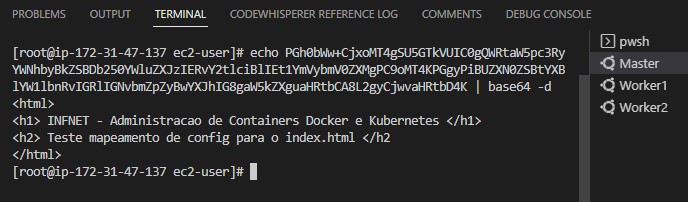
**1.25 Comando docker config ls.**



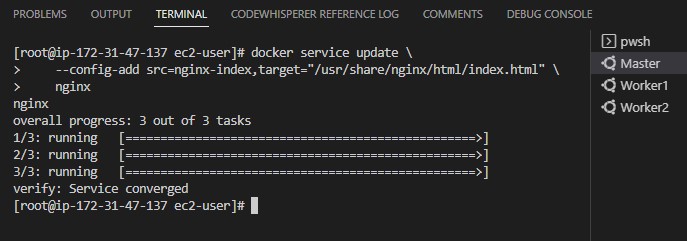
**1.26 Comando docker config inspect.**



**1.27 Conteúdo decodificado do atributo “Data”.**



**1.28 Service update.**



**1.29 Acessando o Serviço externamente.**



**1.30 Acessando o Serviço via ALB.**



**Atividade 2:** A segunda arquitetura contemplará um cluster kubernetes com 3 nós sendo um master e dois workers, com um deployment do wordpress, um statefulset do mysql, sendo o deployment escalável com 3 réplicas e externalizado por ingress. O Wordpress deverá possuir volume persistente com o Longhorn para abrigar o conteúdo do site assim com o banco mysql, os dados sensíveis deverão ser consumidos por secret e as demais configurações por configMap. O Balanceamento externo ao cluster deverá ser desprezado.

**2.1 Implementação da infraestrutura.**

Trabalho elaborado em ambiente WSL do Windows 11 e estrutura implementada através do Terraform em nuvem AWS utilizando o arquivo kubernetes\_k3s\_main.tf da [*Tabela* *2*](#Tabela2).

**Output do Terraform**:

Texto

Descrição gerada automaticamente

**Painel EC2 AWS**:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**2.2 Conectando-se via SSH ao k3s-server(Master):**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**2.3 Inspeção do Cluster:**

**Texto

Descrição gerada automaticamente**

**Texto

Descrição gerada automaticamente**

**2.4 Checando Longhorn:**

**Texto

Descrição gerada automaticamente**

**2.5 wordpress-pvc-mysql.yaml:**

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: mysql-pvc

spec:

  accessModes:

    - ReadWriteOnce

  storageClassName: longhorn

  resources:

    requests:

      storage: 2Gi

**2.5 Aplicando e checando mysql-pvc:**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**2.5 wordpress-configmap.yaml:**

apiVersion: v1

kind: ConfigMap

metadata:

  name: mysql-param

  labels:

    app: wordpress

data:

  hostname: wordpress-mysql

  username: wp\_user

  database: wordpress

**2.6 Aplicando e checando configmap:**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**2.7 wordpress-secret.yaml:**

apiVersion: v1

kind: Secret

metadata:

  name: mysql-pass

  labels:

    app: wordpress

type: Opaque

data:

  password: UGFzc1dvcmQwMA==

**2.6 Aplicando e checando secret:**

Texto

Descrição gerada automaticamente

**2.7 wordpress-statefulset.yaml:**

|  |
| --- |
| apiVersion: apps/v1  kind: StatefulSet  metadata:    name: wordpress-mysql    labels:      app: wordpress  spec:    selector:      matchLabels:        app: wordpress        tier: db    serviceName: "wordpress-mysql"    replicas: 1    template:      metadata:        labels:          app: wordpress          tier: db      spec:        containers:        - image: mysql:5.6          name: mysql          env:          - name: MYSQL\_ROOT\_PASSWORD            valueFrom:              secretKeyRef:                name: mysql-pass                key: password          - name: MYSQL\_PASSWORD            valueFrom:              secretKeyRef:                name: mysql-pass                key: password          - name: MYSQL\_USER            valueFrom:              configMapKeyRef:                name: mysql-param                key: username          - name: MYSQL\_DATABASE            valueFrom:              configMapKeyRef:                name: mysql-param                key: database          ports:          - containerPort: 3306            name: mysql          volumeMounts:          - mountPath: /var/lib/mysql            name: data        volumes:        - name: data          persistentVolumeClaim:            claimName: mysql-pvc |

**2.8 Aplicando e checando statefulset:**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

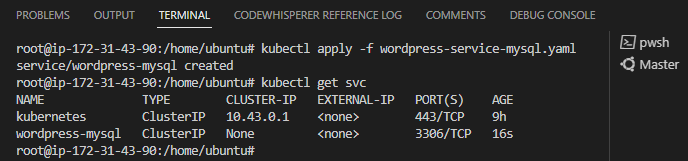
Texto

Descrição gerada automaticamente

**2.9 wordpress-service-mysql.yaml:**

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:    name: wordpress-mysql    labels:      app: wordpress  spec:    ports:      - port: 3306    selector:      app: wordpress      tier: db    clusterIP: None |

**2.10 Aplicando e checando mysql-service:**



**2.11 wordpress-pvc.yaml:**

|  |
| --- |
| apiVersion: v1  kind: PersistentVolumeClaim  metadata:    name: wordpress-pvc  spec:    accessModes:      - ReadWriteMany    storageClassName: longhorn    resources:      requests:        storage: 2Gi |

**2.12 Aplicando e checando wordpress-pvc:**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**2.13 wordpress-deployment.yaml:**

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:    name: wordpress    labels:      app: wordpress  spec:    replicas: 3    selector:      matchLabels:        app: wordpress        tier: frontend    strategy:      type: Recreate    template:      metadata:        labels:          app: wordpress          tier: frontend      spec:        containers:        - image: wordpress:5.4-php7.2-apache          name: wordpress          env:          - name: WORDPRESS\_DB\_PASSWORD            valueFrom:              secretKeyRef:                name: mysql-pass                key: password          - name: WORDPRESS\_DB\_USER            valueFrom:              configMapKeyRef:                name: mysql-param                key: username          - name: WORDPRESS\_DB\_NAME            valueFrom:              configMapKeyRef:                name: mysql-param                key: database          - name: WORDPRESS\_DB\_HOST            valueFrom:              configMapKeyRef:                name: mysql-param                key: hostname          ports:          - containerPort: 80            name: wordpress          volumeMounts:          - mountPath: /var/www/html            name: data        volumes:        - name: data          persistentVolumeClaim:            claimName: wordpress-pvc |

**2.14 Aplicando e checando deployment:**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**2.15 wordpress-service.yaml:**

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:    name: wordpress    labels:      app: wordpress  spec:    ports:    - name: http      port: 80      protocol: TCP      targetPort: 80    selector:      app: wordpress      tier: frontend    type: ClusterIP |

**2.16 Aplicando e checando wordpress-service:**

Texto

Descrição gerada automaticamente

**2.17 wordpress-ingress.yaml:**

|  |
| --- |
| apiVersion: networking.k8s.io/v1  kind: Ingress  metadata:    labels:      app: wordpress    name: wordpress  spec:    rules:    - host: wordpress.18.212.119.97.sslip.io      http:        paths:        - backend:            service:              name: wordpress              port:                number: 80          path: /          pathType: Prefix |

**2.18 Aplicando e checando ingress:**

Texto

Descrição gerada automaticamente

**2.18 Acessando wordpress:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Tabela 1:** docker-main.tf

|  |
| --- |
| # export AWS\_ACCESS\_KEY\_ID=""  # export AWS\_SECRET\_ACCESS\_KEY=""  # ssh user is ec2-user  /\*====  Variables  ======\*/  variable "region" {    description = "Region that the instances will be created"    default     = "us-east-1"  }  variable "docker-host-quantity" {    description = "Quantity of vm nodes"    type        = number    default     = 3  }  /\*  locals {    my-ssh-pubkey = file("~/.ssh/id\_rsa.pub")  }  \*/  locals {    allow-ports = [{      description = "Default"      protocol    = "-1"      cidrblk     = []      self        = true      port        = "0"      }, {      description = "outside ssh access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "22"      }, {      description = "outside http access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "8080"      }, {      description = "outside http access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "8081"      }, {      description = "outside http access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "8082"      }, {      description = "outside http access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "8083"      }, {      description = "outside http access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "80"      }, {      description = "outside https access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "443"    }]  }  locals {    custom-data-docker = <<CUSTOM\_DATA  #!/bin/bash  yum -y install wget curl jq openssl11  yum -y install docker && systemctl start docker && systemctl enable docker  yum -y install socat conntrack ipset  sysctl -w net.ipv4.conf.all.forwarding=1  wget https://github.com/docker/compose/releases/download/1.28.5/docker-compose-Linux-x86\_64  sudo mv docker-compose-Linux-x86\_64 /usr/local/bin/docker-compose  sudo chmod 755 /usr/local/bin/docker-compose  CUSTOM\_DATA  }  /\*====  Resources  ======\*/  provider "aws" {    region = var.region  }  /\*  resource "aws\_key\_pair" "deployer" {    key\_name   = "deployer-key"    public\_key = local.my-ssh-pubkey  }  \*/  data "aws\_ami" "amazon-linux-2" {    most\_recent = true    owners      = ["amazon"]    filter {      name   = "owner-alias"      values = ["amazon"]    }    filter {      name   = "name"      values = ["amzn2-ami-hvm\*"]    }    #  filter {    #      name   = "name"    #      values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-\*"]    #  }    #  filter {    #      name = "virtualization-type"    #      values = ["hvm"]    #  }    #  owners = ["099720109477"]  }  resource "aws\_instance" "docker-host" {    count                       = var.docker-host-quantity    ami                         = data.aws\_ami.amazon-linux-2.id    subnet\_id                   = aws\_default\_subnet.region\_a.id    associate\_public\_ip\_address = true    instance\_type               = "t2.micro"    #instance\_type               = "t2.medium"    #key\_name                    = aws\_key\_pair.deployer.id    key\_name                    = "infnet2"    user\_data\_base64 = base64encode(local.custom-data-docker)    tags = {      Name = "docker-host-${count.index}"      Env  = "docker"    }  }  resource "aws\_default\_vpc" "default" {    tags = {      Name = "Default VPC"    }  }  resource "aws\_default\_subnet" "region\_a" {    availability\_zone = "${var.region}a"    tags = {      Name = "Default subnet for ${var.region}a"    }  }  resource "aws\_default\_subnet" "region\_b" {    availability\_zone = "${var.region}b"    tags = {      Name = "Default subnet for ${var.region}b"    }  }  resource "aws\_default\_security\_group" "default" {    vpc\_id = aws\_default\_vpc.default.id    dynamic "ingress" {      for\_each = local.allow-ports      iterator = each      content {        description      = each.value.description        protocol         = each.value.protocol        self             = each.value.self        from\_port        = each.value.port        to\_port          = each.value.port        cidr\_blocks      = each.value.cidrblk        ipv6\_cidr\_blocks = []        prefix\_list\_ids  = []        security\_groups  = []      }    }    egress = [      {        description      = "Default"        from\_port        = 0        to\_port          = 0        protocol         = "-1"        cidr\_blocks      = ["0.0.0.0/0"]        ipv6\_cidr\_blocks = []        prefix\_list\_ids  = []        security\_groups  = []        self             = false      }    ]  }  output "docker-host\_private\_ip" {    description = "Private IP address of the EC2 instance"    value       = aws\_instance.docker-host.\*.private\_ip  }  output "docker-host\_public\_ip" {    description = "Public IP address of the EC2 instance"    value       = aws\_instance.docker-host.\*.public\_ip  }  output "docker-host\_dns\_name" {    description = "DNS name of the EC2 instance"    value       = aws\_instance.docker-host.\*.public\_dns  }  #########  #  ALB  #  #########  resource "aws\_lb" "docker" {    name               = "docker-lb"    internal           = false    load\_balancer\_type = "application"    security\_groups = [aws\_default\_security\_group.default.id]    subnets = [aws\_default\_subnet.region\_a.id,aws\_default\_subnet.region\_b.id]    enable\_deletion\_protection = false    tags = {      Env  = "docker"    }  }  resource "aws\_lb\_listener" "docker" {    load\_balancer\_arn = aws\_lb.docker.arn    port              = "80"    protocol          = "HTTP"    default\_action {      type             = "forward"      target\_group\_arn = aws\_lb\_target\_group.docker.arn    }  }  resource "aws\_lb\_listener\_rule" "docker" {    listener\_arn = aws\_lb\_listener.docker.arn    priority     = 100    action {      type             = "forward"      target\_group\_arn = aws\_lb\_target\_group.docker.arn    }    condition {      path\_pattern {        values = ["/"]      }    }  }  resource "aws\_lb\_target\_group" "docker" {    name     = "docker-tg"    port     = 8080    protocol = "HTTP"    vpc\_id   = aws\_default\_vpc.default.id    tags = {      Env  = "docker"    }    health\_check {      healthy\_threshold   = 3      unhealthy\_threshold = 10      timeout             = 5      interval            = 10      path                = "/"      port                = 8080    }  }  resource "aws\_lb\_target\_group\_attachment" "docker" {    count = length(aws\_instance.docker-host)    target\_group\_arn = aws\_lb\_target\_group.docker.arn    target\_id        = aws\_instance.docker-host[count.index].id    port             = 8080  }  data "aws\_lb" "docker" {    arn  = aws\_lb.docker.arn    name = aws\_lb.docker.name  }  output "docker-lb\_dnsname" {    description = "Public Name of Docker LB"    value       = data.aws\_lb.docker.dns\_name  } |

**Tabela 2:** kubernetes\_k3s\_main.tf

|  |
| --- |
| # export AWS\_ACCESS\_KEY\_ID=""  # export AWS\_SECRET\_ACCESS\_KEY=""  # ssh user is ec2-user  /\*====  Variables  ======\*/  variable "region" {    description = "Region that the instances will be created"    default     = "us-east-1"  }  variable "k3s-client-quantity" {    description = "Quantity of k3s worker nodes"    type        = number    default     = 2  }  /\*  locals {    my-ssh-pubkey = file("~/.ssh/id\_rsa.pub")  }  \*/  locals {    allow-ports = [{      description = "Default"      protocol    = "-1"      cidrblk     = []      self        = true      port        = "0"      }, {      description = "outside ssh access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "22"      }, {      description = "outside traffik access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "80"      }, {      description = "outside traffik access"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "443"      }, {      description = "proxy port-forward"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "8000"      }, {      description = "outside nodeport"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "30080"      }, {      description = "outside nodeport"      protocol    = "tcp"      cidrblk     = ["0.0.0.0/0"]      self        = false      port        = "30081"    }]  }  locals {    k3s\_token = base64encode("Token super secreto lab aws")  }  locals {    custom-data-server = <<CUSTOM\_DATA  #!/bin/bash  curl -sfL https://get.k3s.io | \    K3S\_TOKEN=${local.k3s\_token} \    sh -s - server \    --node-taint CriticalAddonsOnly=true:NoExecute  sleep 5  kubectl completion bash | sudo tee /etc/bash\_completion.d/kubectl > /dev/null  CUSTOM\_DATA  }  locals {    custom-data-client = <<CUSTOM\_DATA  #!/bin/bash  curl -sfL https://get.k3s.io | K3S\_URL=https://${aws\_instance.k3s-server.private\_ip}:6443 K3S\_TOKEN=${local.k3s\_token} sh -  yum install -y iscsi-initiator-utils.x86\_64 libiscsi.x86\_64 libiscsi-utils.x86\_64 nfs-utils.x86\_64  CUSTOM\_DATA  }  /\*====  Resources  ======\*/  provider "aws" {    region = var.region  }  /\*  resource "aws\_key\_pair" "deployer" {    key\_name   = "deployer-key"    public\_key = local.my-ssh-pubkey  }  \*/  data "aws\_ami" "amazon-linux-2" {    most\_recent = true    owners      = ["amazon"]    filter {      name   = "owner-alias"      values = ["amazon"]    }    filter {      name   = "name"      values = ["amzn2-ami-hvm\*"]    }  }  data "aws\_ami" "ubuntu" {    owners      = ["099720109477"]    most\_recent = true    filter {      name   = "name"      values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-\*"]    }    filter {      name   = "virtualization-type"      values = ["hvm"]    }  }  resource "aws\_instance" "k3s-server" {    subnet\_id = aws\_default\_subnet.region\_a.id    #ami                         = data.aws\_ami.amazon-linux-2.id    ami                         = data.aws\_ami.ubuntu.id    associate\_public\_ip\_address = true    instance\_type               = "t3a.medium"    #instance\_type               = "t2.micro"    #key\_name                    = aws\_key\_pair.deployer.id    key\_name                    = "infnet2"    user\_data\_base64            = base64encode(local.custom-data-server)    tags = {      Name = "k3s-server"      Env  = "k3s"    }  }  resource "aws\_instance" "k3s-client" {    count                       = var.k3s-client-quantity    depends\_on                  = [aws\_instance.k3s-server]    subnet\_id = aws\_default\_subnet.region\_a.id    #ami                         = data.aws\_ami.amazon-linux-2.id    ami                         = data.aws\_ami.ubuntu.id    associate\_public\_ip\_address = true    instance\_type               = "t3a.medium"    #instance\_type               = "t2.micro"    #key\_name                    = aws\_key\_pair.deployer.id    key\_name                    = "infnet2"    user\_data\_base64            = base64encode(local.custom-data-client)    root\_block\_device {      volume\_size           = "30"      volume\_type           = "gp2"      delete\_on\_termination = true    }    tags = {      Name = "k3s-client-${count.index}"      Env  = "k3s"    }  }  resource "aws\_default\_vpc" "default" {    tags = {      Name = "Default VPC"    }  }  resource "aws\_default\_subnet" "region\_a" {    availability\_zone = "${var.region}a"    tags = {      Name = "Default subnet for ${var.region}a"    }  }  resource "aws\_default\_subnet" "region\_b" {    availability\_zone = "${var.region}b"    tags = {      Name = "Default subnet for ${var.region}b"    }  }  resource "aws\_default\_security\_group" "default" {    vpc\_id = aws\_default\_vpc.default.id    dynamic "ingress" {      for\_each = local.allow-ports      iterator = each      content {        description      = each.value.description        protocol         = each.value.protocol        self             = each.value.self        from\_port        = each.value.port        to\_port          = each.value.port        cidr\_blocks      = each.value.cidrblk        ipv6\_cidr\_blocks = []        prefix\_list\_ids  = []        security\_groups  = []      }    }    egress = [      {        description      = "Default"        from\_port        = 0        to\_port          = 0        protocol         = "-1"        cidr\_blocks      = ["0.0.0.0/0"]        ipv6\_cidr\_blocks = []        prefix\_list\_ids  = []        security\_groups  = []        self             = false      }    ]  }  # Server outputs  output "k3s-server\_dns\_name" {    description = "DNS name of the Server EC2 instance"    value       = aws\_instance.k3s-server.\*.public\_dns  }  output "k3s-server\_public\_ip" {    description = "Public IP address of the Server EC2 instance"    value       = aws\_instance.k3s-server.public\_ip  }  output "k3s-server\_private\_ip" {    description = "Private IP address of the Server EC2 instance"    value       = aws\_instance.k3s-server.\*.private\_ip  }  # Client outputs  output "k3s-client\_dns\_name" {    description = "DNS name of the Client EC2 instance"    value       = aws\_instance.k3s-client.\*.public\_dns  }  output "k3s-client\_public\_ip" {    description = "Public IP address of the Client EC2 instance"    value       = aws\_instance.k3s-client.\*.public\_ip  }  output "k3s-client\_private\_ip" {    description = "Private IP address of the Client EC2 instance"    value       = aws\_instance.k3s-client.\*.private\_ip  }  #########  #  NLB  #  #########  /\*  resource "aws\_eip" "lb-region\_a" {    vpc      = true    tags = {      Env  = "k3s"    }  }  #resource "aws\_eip" "lb-region\_b" {  #  vpc      = true  #  tags = {  #    Env  = "k3s"  #  }  #}  resource "aws\_lb" "k3s" {    name               = "k3s-lb"    internal           = false    load\_balancer\_type = "network"    subnet\_mapping {      subnet\_id     = aws\_default\_subnet.region\_a.id      allocation\_id = aws\_eip.lb-region\_a.id    }    subnet\_mapping {      subnet\_id     = aws\_default\_subnet.region\_b.id      #allocation\_id = aws\_eip.lb-region\_b.id    }    enable\_deletion\_protection = false    tags = {      Env  = "k3s"    }  }  resource "aws\_lb\_listener" "k3s" {    load\_balancer\_arn = aws\_lb.k3s.arn    port              = "80"    protocol          = "TCP"    default\_action {      type             = "forward"      target\_group\_arn = aws\_lb\_target\_group.k3s.arn    }  }  resource "aws\_lb\_target\_group" "k3s" {    name     = "k3s-tg"    port     = 80    protocol = "TCP"    vpc\_id   = aws\_default\_vpc.default.id    tags = {      Env  = "k3s"    }  }  resource "aws\_lb\_target\_group\_attachment" "k3s" {    count = length(aws\_instance.k3s-client)    target\_group\_arn = aws\_lb\_target\_group.k3s.arn    target\_id        = aws\_instance.k3s-client[count.index].id    port             = 80  }  data "aws\_eip" "lb-region\_a" {    id = aws\_eip.lb-region\_a.id  }  output "k3s-lb-region\_a" {    description = "Public IP of k3s LB region\_a"    value       = data.aws\_eip.lb-region\_a.public\_ip  }  \*/ |