Sebastian Lopez Garcia A00377582

Introduction to Terraform Lab with Kubernetes

In this lab, we'll work with **Terraform** for infrastructure-as-code management, using **kubectl** to interface with a Kubernetes cluster. The main objective will be to deploy a **pod with Nginx** and visualize it through **Lens Desktop**, a graphical tool that facilitates the administration of Kubernetes clusters.

az-cli:

curl -sL https://aka.ms/InstallAzureCLIDeb installation | sudo bash

sudo apt-get update

sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release

sudo mkdir -p /etc/apt/keyrings

curl -sLS https://packages.microsoft.com/keys/microsoft.asc |

gpg --dearmor | sudo tee /etc/apt/keyrings/microsoft.gpg > /dev/null

sudo chmod go+r /etc/apt/keyrings/microsoft.gpg

AZ_DIST=$(lsb_release -cs)

echo "Types: deb

URIs: https://packages.microsoft.com/repos/azure-cli/

Suites: ${AZ_DIST}

Components: main

Architectures: $(dpkg --print-architecture)

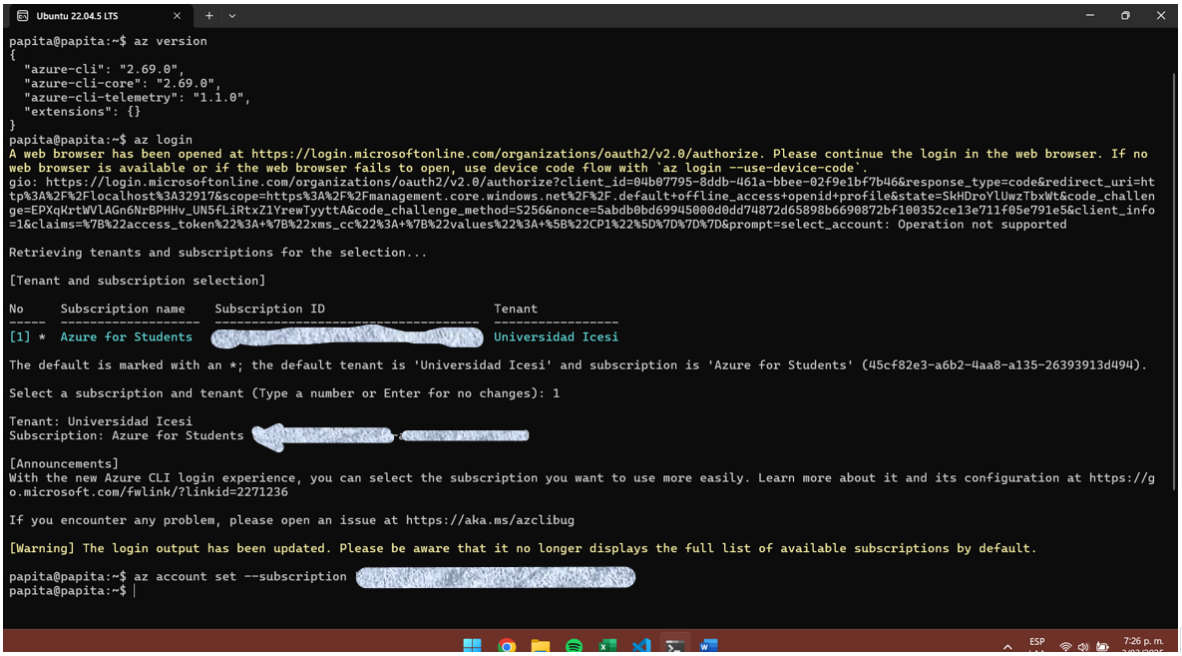
Signed-by: /etc/apt/keyrings/microsoft.gpg" | sudo tee /etc/apt/sources.list.d/azure-cli.sources

sudo apt-get update

sudo apt-get install azure-cli

It's critical to authenticate to **Microsoft Azure** to manage resources from Terraform and Kubernetes. We'll use the az login command from the **Azure CLI** to authenticate our session and gain access to the subscription where we'll deploy our environment.

After these steps, we would already have Az-cli:



Azure Provider

```
provider "azurerm" {
  features {}
}
```

This block defines the Azure Resource Manager provider ('azurerm'). The 'features {}' block is required but may be empty.

Resource Group

```
resource "azurerm_resource_group" "labTerraform" {
  name     = "labs_plataformas_rg"
  location = "East US"
}
```

This block creates a resource group in Azure called 'labs_plataformas_rg' in the 'East US' location.

Cluster de Kubernetes

```
resource "azurerm_kubernetes_cluster" "sl-aks1" {
  name                = "sl-aks1"
  location            = azurerm_resource_group.labTerraform.location
  resource_group_name = azurerm_resource_group.labTerraform.name
  dns_prefix          = "sl1"

  default_node_pool {
    name       = "default"
    node_count = 1
    vm_size    = "Standard_D2_v2"
  }

  identity {
    type = "SystemAssigned"
  }

  tags = {
    Environment = "testings"
  }
}
```

This block creates a Kubernetes cluster on Azure (AKS) named 'sl-aks1'. Here are the details:

- 'location' and 'resource_group_name' refer to the previously created resource group.

- 'dns_prefix' is the DNS prefix for the cluster.

- 'default_node_pool' defines the default node pool with a single node ('node_count = 1') and a VM size 'Standard_D2_v2'.

- 'identity' states that the cluster will use a system-managed identity ('SystemAssigned').

- 'tags' adds a tag to identify the environment as "testings".

Outputs

```
output "client_certificate" {
  value     = azurerm_kubernetes_cluster.sl-aks1.kube_config[0].client_certificate
  sensitive = true
}

output "kube_config" {
  value = azurerm_kubernetes_cluster.sl-aks1.kube_config_raw
  sensitive = true
}
```

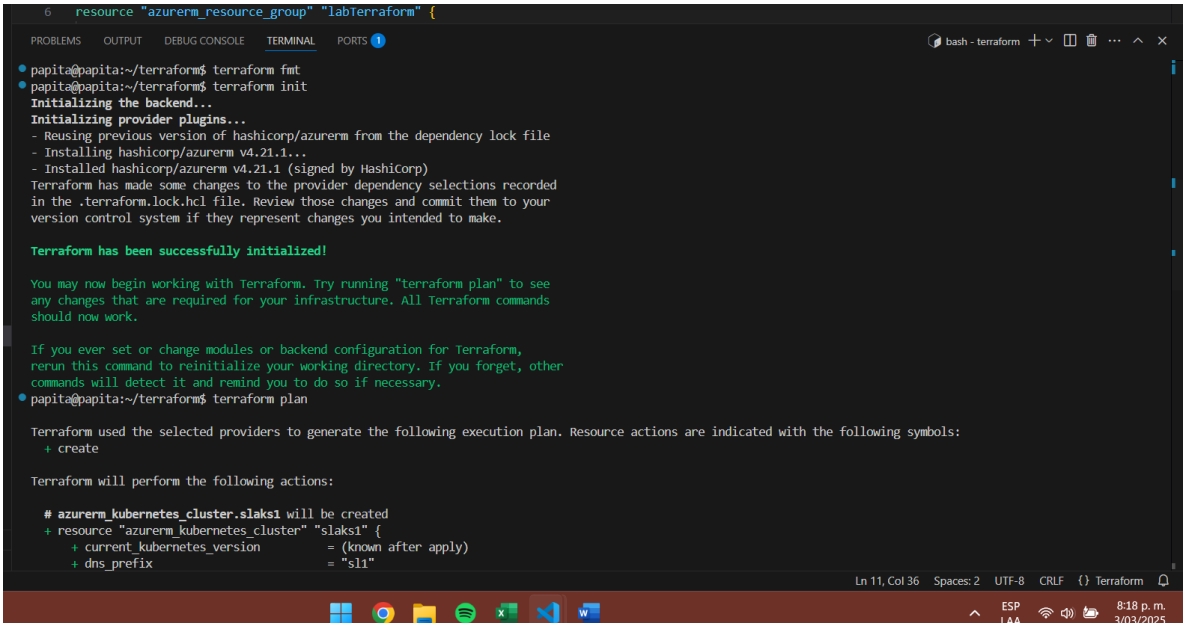
These output blocks expose sensitive information from the Kubernetes cluster:

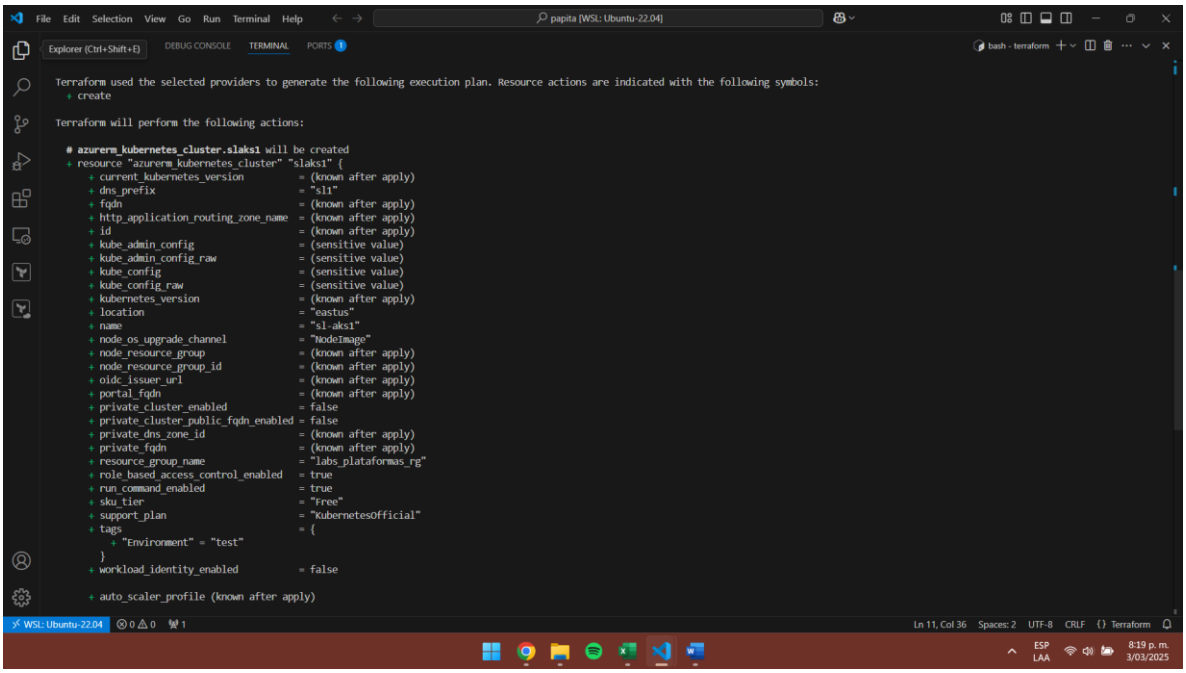
- 'client_certificate' exposes the cluster client certificate.

- 'kube_config' exposes the complete configuration of the cluster in RAW format.

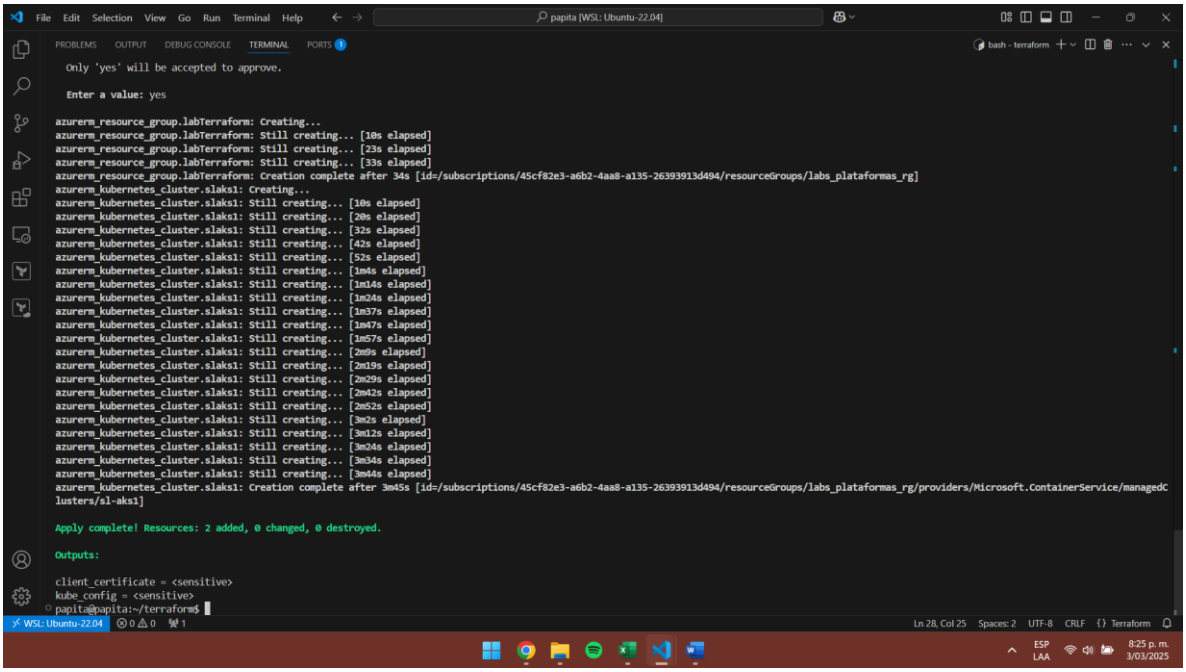
Both values are marked as 'sensitive = true' to prevent them from being displayed in Terraform's output logs.

The next step is to prepare Terraform to deploy our infrastructure. First, we use the **terraform init command**, which is responsible for initializing the work environment. This includes offloading the necessary providers and configuring the backend where the state of the infrastructure will be stored. Without this step, we wouldn't be able to execute any actions with Terraform.

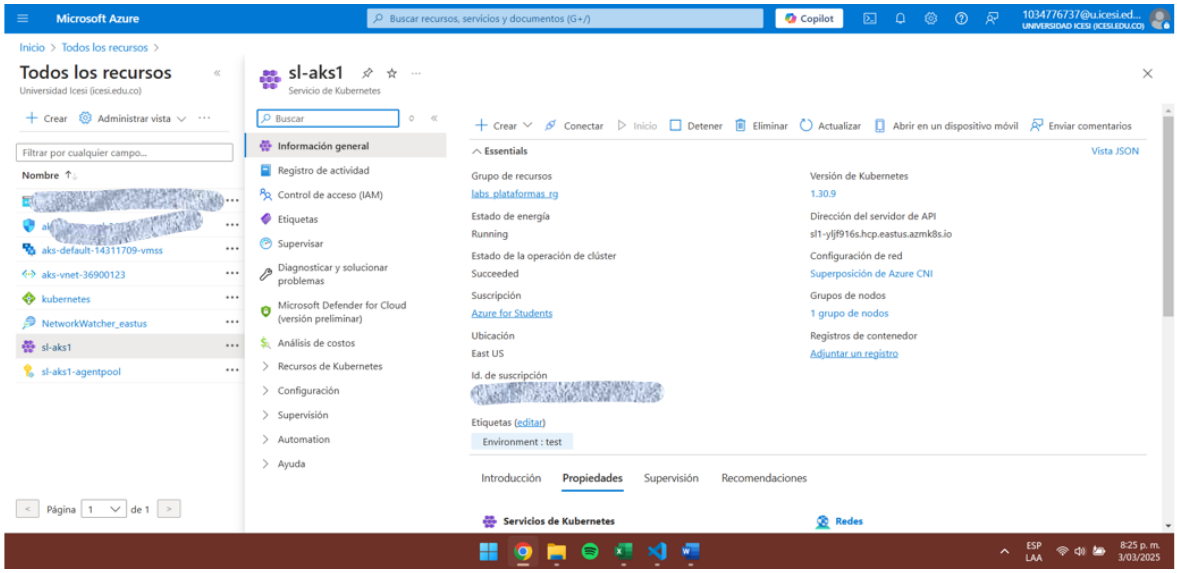


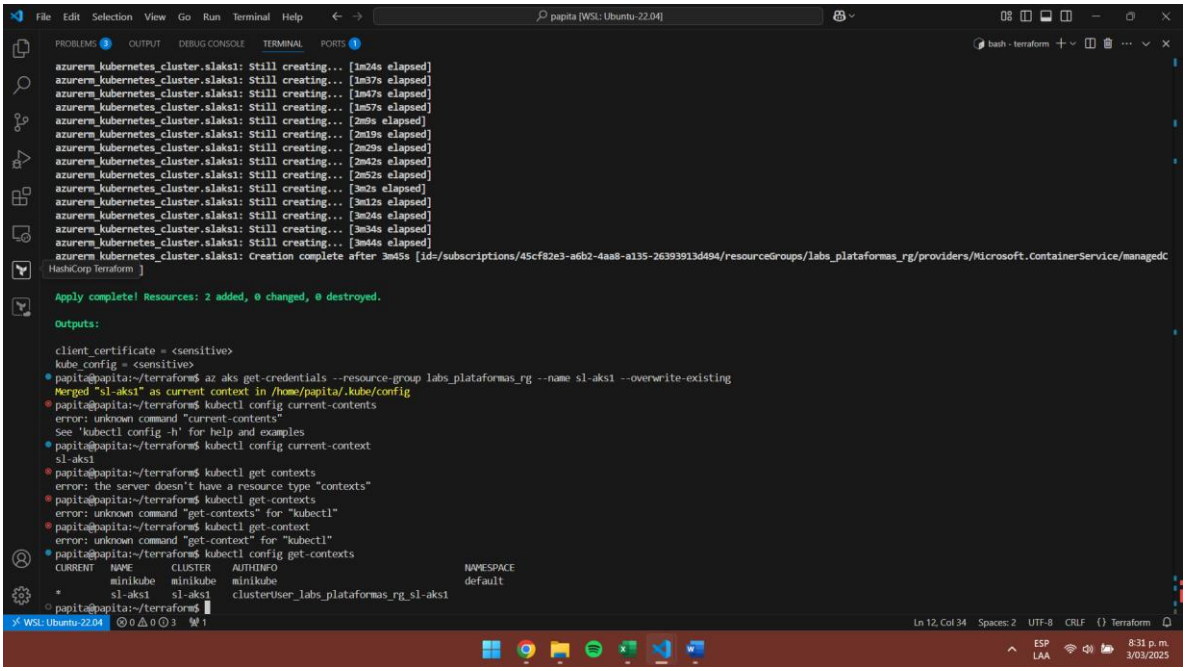


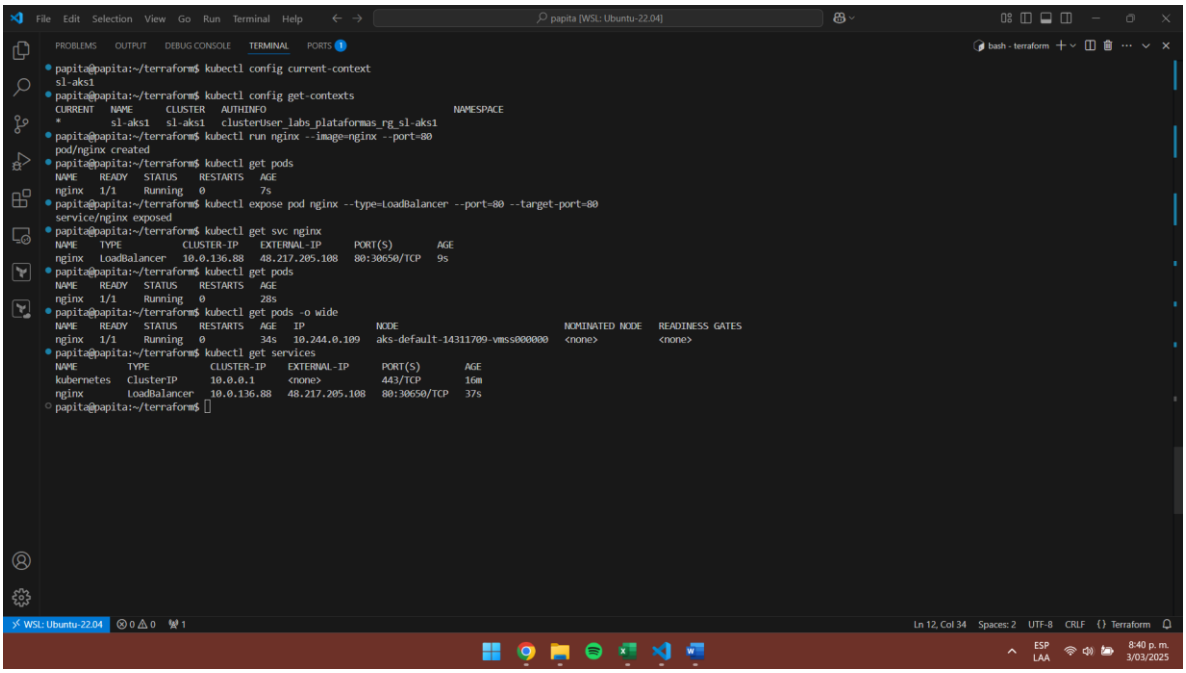
Finally, we apply the changes with **terraform apply**, which is responsible for creating or modifying the infrastructure as defined in the Terraform files. This command generates an execution plan, and upon confirmation, proceeds to deploy the resources to Azure. With this, our infrastructure is ready for use.



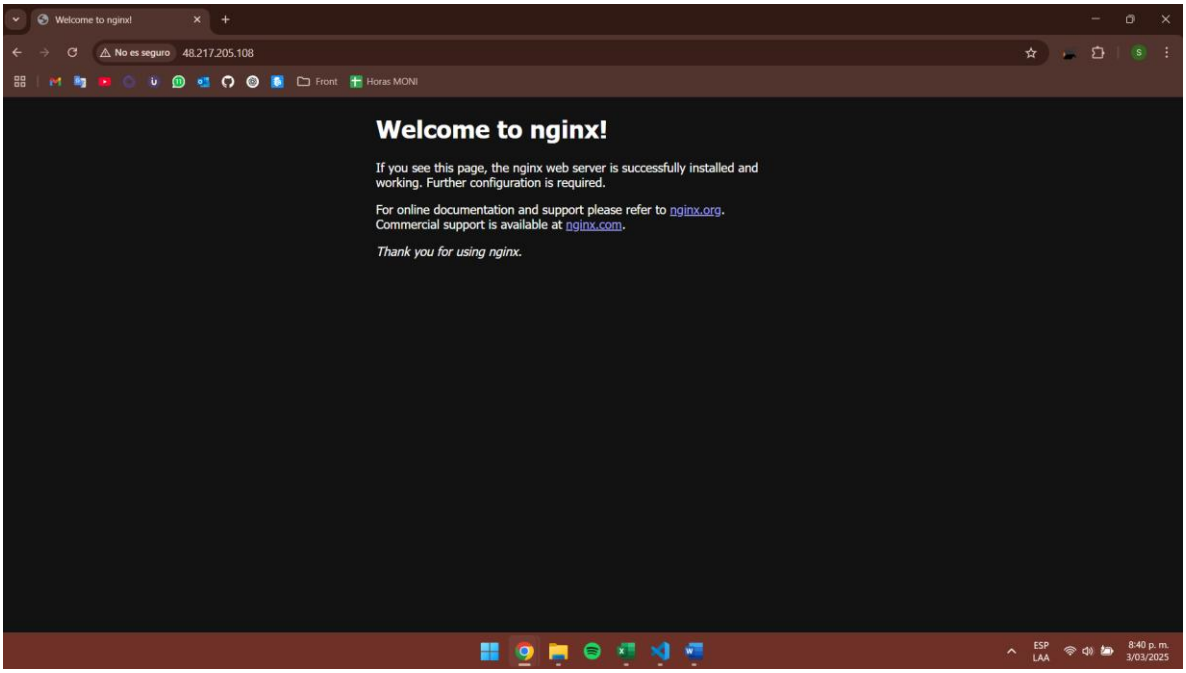
Once we've run **terraform apply**, we can verify that our pod with Nginx is running inside Azure. To do this, we use **kubectl**, which allows us to interact with the Kubernetes cluster. With the **kubectl get pods** command, we can see the status of the pods and make sure they are in **the Running** state. We can also inspect the services with **kubectl get services**, which will show us if the external IP has been assigned correctly.



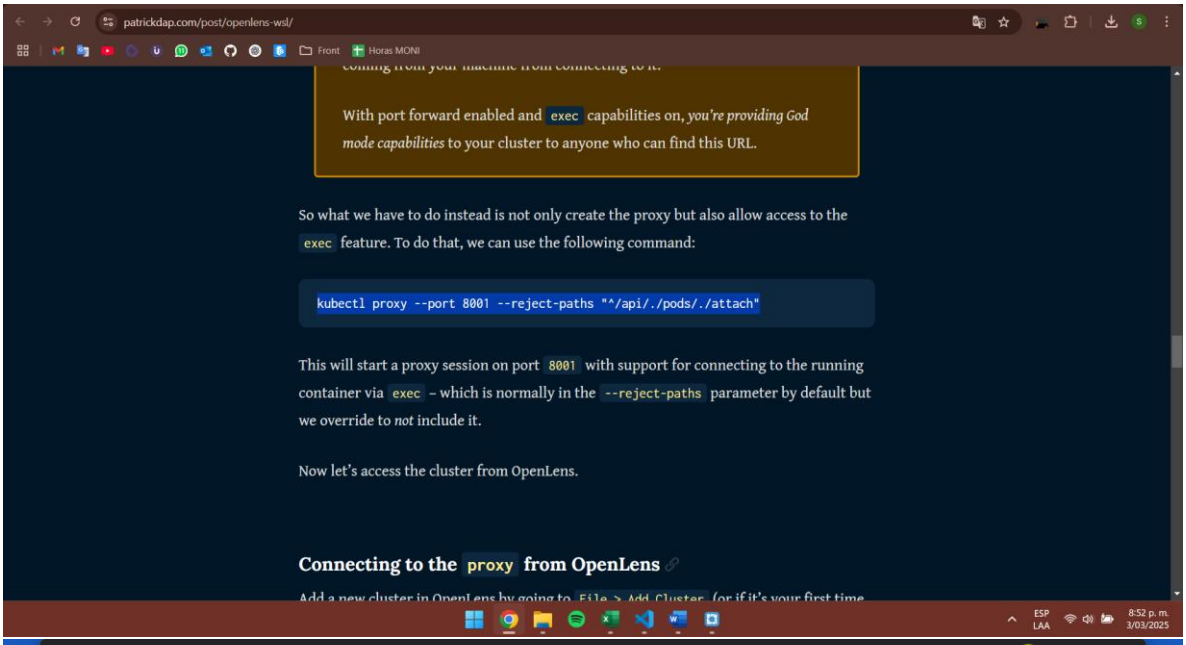


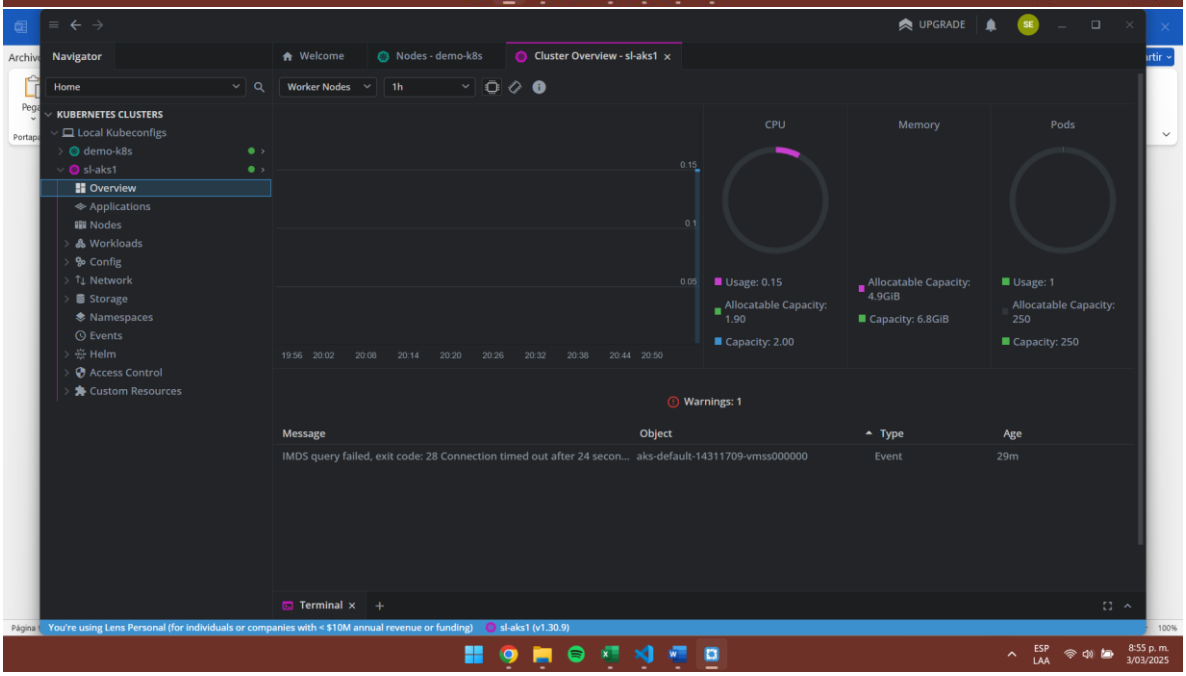


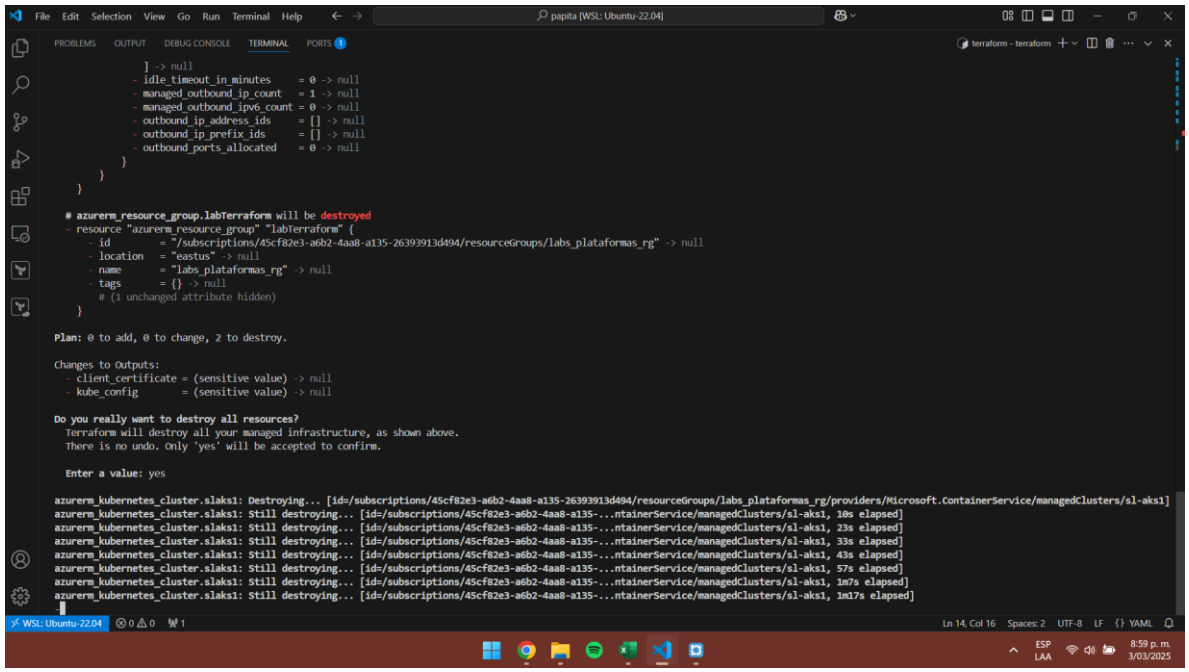
To access the website served by Nginx, we use the public IP address assigned to the service. This will allow us to view the Nginx welcome page and confirm that the deployment was successful.



Finally, we can use **Lens Desktop** to visually manage our cluster. Lens provides us with a graphical interface where we can see the pods, services, and other resources deployed in Kubernetes. To connect, we add the cluster to Lens using the **kubeconfig** file, which will allow us to monitor the state of the system, review logs, and manage resources in a more intuitive way.






File Edit Selection View Go Run Terminal Help
papita [WSL: Ubuntu-22.04]
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1
terraform - terraform
] -> null
- idle_timeout_in_minutes = 0 -> null
- managed_outbound_ip_count = 1 -> null
- managed_outbound_ipv6_count = 0 -> null
- outbound_ip_address_ids = [] -> null
- outbound_ip_prefix_ids = [] -> null
- outbound_ports_allocated = 0 -> null
}
}
}
# azurerm_resource_group.labTerraform will be destroyed
- resource "azurerm_resource_group" "labTerraform" {
- id = "/subscriptions/45cf82e3-a6b2-4aa8-a135-26393913d494/resourceGroups/labs_plataformas_rg" -> null
- location = "eastus" -> null
- name = "labs_plataformas_rg" -> null
- tags = {} -> null
# (1 unchanged attribute hidden)
}
Plan: 0 to add, 0 to change, 2 to destroy.
Changes to Outputs:
- client_certificate = (sensitive value) -> null
- kube_config = (sensitive value) -> null
Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.
Enter a value: yes
azurerm_kubernetes_cluster.slaks1: Destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-26393913d494/resourceGroups/labs_plataformas_rg/providers/Microsoft.ContainerService/managedClusters/sl-aks1]
azurerm_kubernetes_cluster.slaks1: Still destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-...ntainerService/managedClusters/sl-aks1, 10s elapsed]
azurerm_kubernetes_cluster.slaks1: Still destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-...ntainerService/managedClusters/sl-aks1, 23s elapsed]
azurerm_kubernetes_cluster.slaks1: Still destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-...ntainerService/managedClusters/sl-aks1, 33s elapsed]
azurerm_kubernetes_cluster.slaks1: Still destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-...ntainerService/managedClusters/sl-aks1, 43s elapsed]
azurerm_kubernetes_cluster.slaks1: Still destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-...ntainerService/managedClusters/sl-aks1, 57s elapsed]
azurerm_kubernetes_cluster.slaks1: Still destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-...ntainerService/managedClusters/sl-aks1, 1m7s elapsed]
azurerm_kubernetes_cluster.slaks1: Still destroying... [id=/subscriptions/45cf82e3-a6b2-4aa8-a135-...ntainerService/managedClusters/sl-aks1, 1m17s elapsed]
WSL: Ubuntu-22.04
Ln 14, Col 16 Spaces: 2 UTF-8 LF {} YAML
ESP
LAA
8:59 p. m.
3/03/2025