**2.Create Different Node Pools for Products:**

**1. Implement Terraform code to create multiple node pools within the AKS cluster.**

📘 What are node pools?

📘 What are System and User node pools?

🛠️ How to schedule application pods on a specific node pool using Labels and node Selector?

🛠️ How to allow only specific application pods to be scheduled on a node pool using Taints and Tolerations?

🛠️ How node pools could be used to reduce the risk behind upgrading a cluster?

🛠️ How to set auto scalability for each node pool?

🖥️ In a Kubernetes cluster, the containers are deployed as pods into VMs called worker or agent nodes.

🌐 These nodes are identical as they use the same VM size or SKU.

🧠 This was just fine until we realized we might need nodes with different SKU for the following reasons:

👉 Prefer to deploy Kubernetes system pods (like CoreDNS, metrics-server, Gatekeeper addon) and application pods on different dedicated nodes. This is in order to prevent misconfigured or rogue application pods to accidentally killing system pods.

👉 Some pods requires either CPU or Memory intensive and optimized VMs.

👉 Some pods are processing ML/AI algorithms and needs GPU enabled VMs. These GPU enabled VMs should be used only by certain pods as they are expensive.

👉 Some pods/jobs want to leverage spot/preemptible VMs to reduce the cost.

👉 Some pods running legacy Windows applications requires Windows Containers available with Windows VMs.

👉 Some teams want to physically isolate their non-production environments (dev, test, QA, staging...) within the same cluster. This is because it is easier to manage less clusters.

❓🧐 **These teams realized that logical isolation with namespaces is not enough.**

These reasons led to the creation of heterogeneous nodes within the cluster. To make it easier to manage these nodes, Kubernetes introduced the NodePool.

The nodepool is a group of nodes that share the same configuration (CPU, Memory, Networking, OS, maximum number of pods...).

By default, one single (system) nodepool is created within the cluster.

However, we can add nodepools during or after cluster creation.

We can also remove these nodepools at any time.

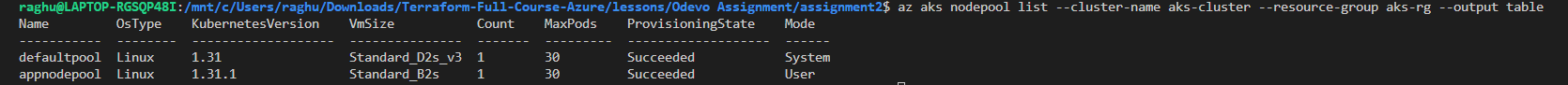
**There are 2 types of nodepools:**

🖥️ System nodepool: used to preferably deploy system pods. Kubernetes could have multiple system nodepools. At least one nodepool is required with at least one single node. System nodepools must run only on Linux (no support for Windows).

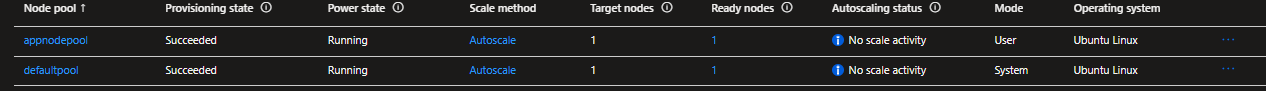
🖥️User nodepool: used to preferably deploy application pods. Kubernetes could have multiple user nodepools or none. All user nodepools could scale down to zero nodes. A user nodepool could run on Linux or Windows nodes.

We'll start by creating a new AKS cluster using the terraform.

This will create a new cluster(aks-cluster) with one single nodepool called defaultnodepool. This node pool is of type System. It doesn't have any taints. We'll add a new user nodepool called appnodepool. This node pool is of type User.

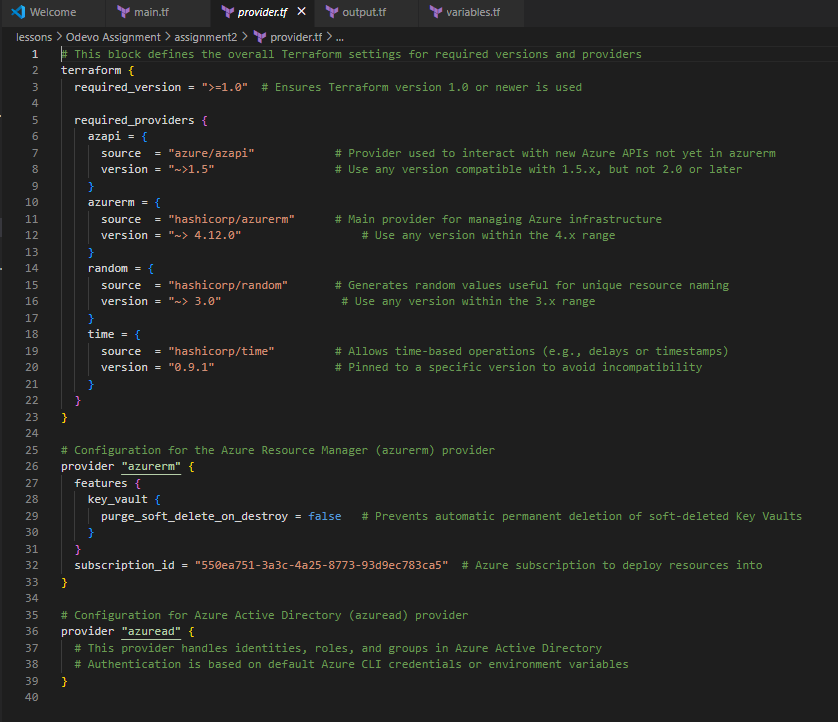
We can then view the 2 nodepools from the portal or command line.

This could be also done with the Azure portal. Go to the cluster, Node pools section.



**Now I am explaining the Terraform files and the configurations used to achieve this assignment.**

Starting with the**🔌** **provider.tf file** — I used the standard configuration available in the official Azure Terraform documentation. This file sets up the required provider block to authenticate and interact with Azure resources. It includes all the necessary settings to allow Terraform to provision resources in the Azure environment.



**🏗️main.tf Explanation**

The main.tf file contains the core Terraform configuration to provision an Azure Kubernetes Service (AKS) cluster along with a resource group and multiple node pools. Here's a breakdown of its components:

**1.Resource Group:**

* Creates an Azure Resource Group named aks-rg in the swedencentral region.
* This group acts as a container for all the resources in this deployment.

**2.Azure Kubernetes Service (AKS) Cluster**

* Provisions an AKS cluster named aks-cluster within the previously created resource group.
* The cluster uses a default node pool, which is a required component for AKS.

Default Node Pool Configuration:

* Defines the default node pool with autoscaling enabled, a fixed node count of 1, and high availability via multiple zones.
* The VM type is Standard\_D2s\_v3, and it uses Virtual Machine Scale Sets (VMSS) for scalability.

Identity & Access:

* Assigns a system-managed identity to the AKS cluster, which is used for resource access permissions.

Linux Profile:

* Specifies admin access credentials using SSH keys for secure login to nodes.

Networking:

* Configures the load balancer to use the standard SKU for improved performance and reliability.

**3. Additional Node Pools**

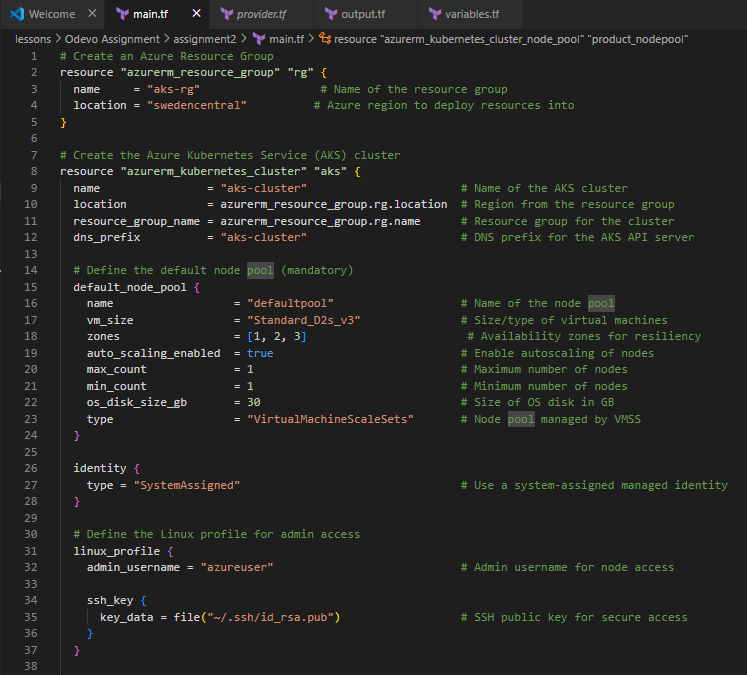
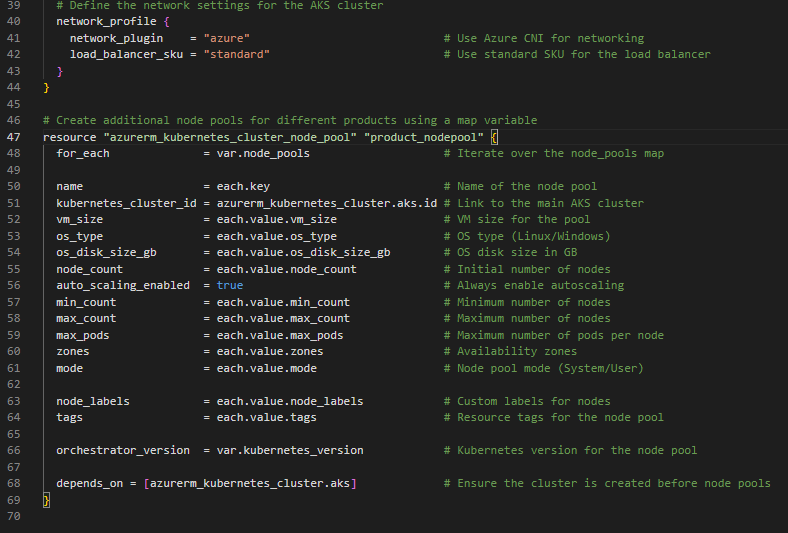
* Dynamically creates multiple user-defined node pools based on the node pools map variable defined in variables.tf.
* each pool supports autoscaling, availability zones, and tagging for identification.

**Important fields:**

* vm\_size, os\_type, os\_disk\_size\_gb, and scaling parameters (min\_count, max\_count, node\_count).
* vnet\_subnet\_id, zones, mode for network and scheduling preferences.
* node\_labels and tags for grouping and management.

Dependency:

* Ensures the node pools are created only after the main AKS cluster is successfully provisioned.

**** ****

**🧱variables.tf Explanation**

The variables.tf file defines configurable input variables used across the Terraform project. These variables allow flexibility and reusability in the infrastructure code. It contains two main variables:

**1.node\_pools Variable**

* This variable is a map of objects, where each object defines the configuration for a separate AKS node pool.
* It's used in the for\_each loop in main.tf to create multiple node pools dynamically.
* Each node pool configuration includes the following attributes:

Attributes:

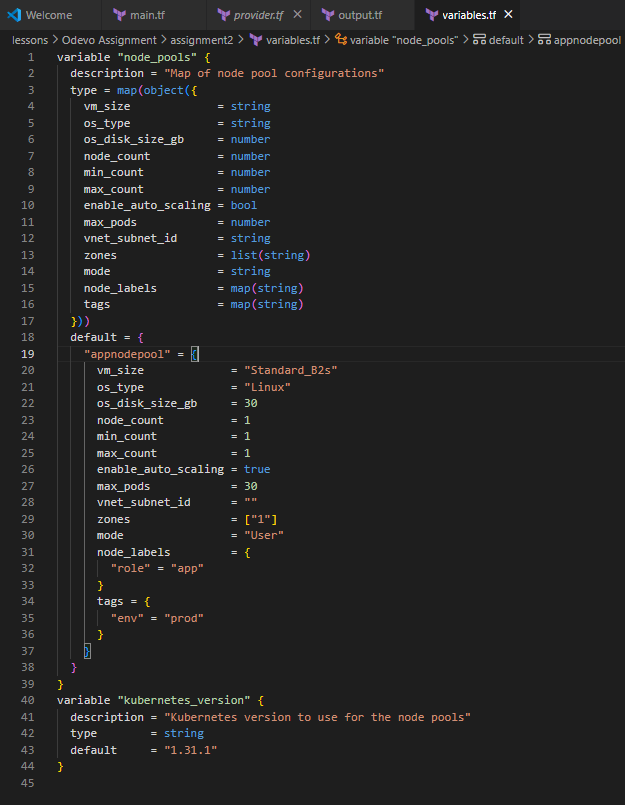
| **Attribute** | **Description** |
| --- | --- |
| vm\_size | The size of the virtual machine to use for the node pool. |
| os\_type | Specifies the operating system type, e.g., Linux or Windows. |
| os\_disk\_size\_gb | Size (in GB) of the OS disk attached to each VM in the pool. |
| node\_count | The number of nodes to initially deploy in the pool. |
| min\_count / max\_count | Minimum and maximum limits for autoscaling. |
| enable\_auto\_scaling | Enables or disables autoscaling. |
| max\_pods | The maximum number of pods that can run on each node. |
| vnet\_subnet\_id | The subnet ID to place the node pool in (optional/empty by default). |
| zones | Availability zones for distributing nodes. |
| mode | Pool mode (User or System). |
| node\_labels | Key-value labels assigned to the nodes for scheduling or identification. |
| tags | Resource tags for organizing and managing resources. |

**Default Example:**

* This sets up a minimal Linux-based node pool named appnodepool with basic resources, labels, and tags for a production environment.

**2. Kubernetes version Variable**

* Defines the version of Kubernetes to be used across the cluster and all node pools.
* Helps ensure compatibility and consistency in your AKS deployment.



**🎯output.tf Explanation**

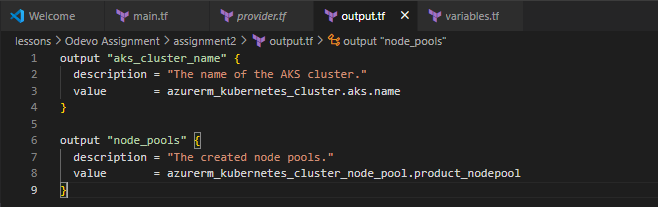
The output.tf file defines the outputs of your Terraform configuration. Outputs are useful for displaying relevant information after deployment and for referencing values in other Terraform configurations (via remote state or modules).

**1. aks\_cluster\_name Output**

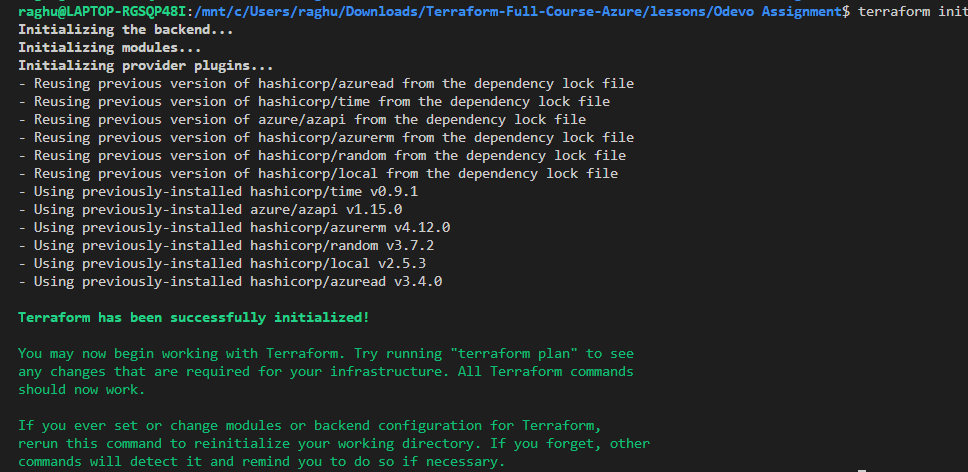
* **Purpose:** Outputs the name of the Azure Kubernetes Service (AKS) cluster created.
* **Usage:** Helpful for verification, automation, or when referencing the cluster name in subsequent operations or scripts.

**2. node\_pools Output**

* **Purpose:** Displays information about the additional node pools created via the product\_nodepool resource.
* **Usage:** Useful for debugging, validation, or exporting data to other modules or tools that need to interact with specific node pools

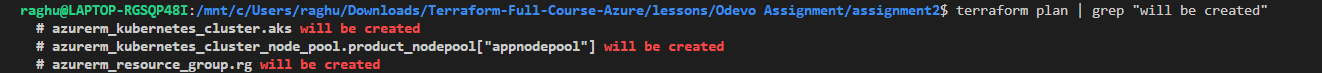
****

**✅terraform init**



* Purpose: Initializes the Terraform working directory.
* What it does: Downloads the necessary provider plugins (e.g., azurerm for Azure).
* When to use: Run this first, only once per configuration (or again if you add providers or modules).

**✅terraform plan**



* Purpose: Shows the execution plan.
* What it does: Displays what Terraform intends to do based on the current state and configuration (add, change, destroy).
* Benefit: Helps you review changes before applying them.

**✅terraform apply**

* Purpose: Applies the configuration to provision resources.
* What it does: Builds or updates your infrastructure.
* Bonus: After applying, you will see the outputs defined in output.tf

**✅terraform destroy**

* Purpose: Deletes all resources defined in your configuration.
* Use case: When you want to tear down the environment completely.

NodePools could be used to upgrade the cluster with less risk

Upgrading the entire cluster (control plane and all nodepools) might be a risky operation.

NodePools could be leveraged to reduce the risk.

Instead of upgrading the nodepool, we proceed with blue/green upgrade:

1) Upgrade only the control plane to the newer version

2) Create a new nodepool with the newer version

3) Deploy the application pods in the newer nodepool

4) Verify the application works fine

5) Delete the old nodepool

**Resources:**

https://docs.microsoft.com/en-us/azure/aks/use-system-pools

<https://docs.microsoft.com/en-us/azure/aks/use-multiple-node-pools>

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/kubernetes\_cluster\_node\_pool