Day 1 : Azure

**🔹 1. Deploy application (new)**

**Use this if:**  
You already have a Kubernetes cluster (AKS or Arc-connected), and just want to **deploy an application quickly** without worrying about infrastructure.

**Real-world scenario:**  
You're a developer who wants to deploy a containerized app from a GitHub repo to a pre-existing AKS cluster. This is similar to GKE’s “deploy from marketplace” or “autopilot with app config.”

**Key Points:**

* No cluster creation
* Application-focused
* Uses GitOps or Bicep templates under the hood
* Great for DevOps teams

**🔹 2. Kubernetes cluster**

**Use this if:**  
You want **full control** over how the Kubernetes cluster is set up — VM size, node count, network settings, scaling, etc.

**Real-world scenario:**  
You're an SRE or cloud engineer setting up a **production-grade AKS cluster** for a team, with custom VNet, node pools, autoscaling, logging, etc.

**Key Points:**

* Full flexibility and control
* Ideal for production or complex environments
* Requires more decisions (but that’s a good thing for custom setups)
* Most common choice for serious workloads

**🔹 3. Automatic Kubernetes cluster (preview)**

**Use this if:**  
You want a **simplified experience** like GKE Autopilot — Azure manages infrastructure for you.

**Real-world scenario:**  
You’re prototyping or running **small teams or dev/test environments** and want a hands-off, fast setup with minimal effort.

**Key Points:**

* Similar to GKE Autopilot
* Less control but easier to use
* Still in preview (not recommended for critical production workloads yet)
* Azure handles upgrades, infra, autoscaling automatically

**🔹 4. Add a Kubernetes cluster with Azure Arc**

**Use this if:**  
You have an existing Kubernetes cluster **outside of Azure** (e.g., on-premises or AWS/GCP) and want to manage it from **Azure portal**.

**Real-world scenario:**  
Your company has clusters on-premises but wants **central governance, security, and monitoring via Azure**.

**Key Points:**

* Does not create cluster
* Registers external clusters in Azure
* Useful for hybrid cloud strategies

**🔹 5. Create a Kubernetes cluster with Azure Arc**

**Use this if:**  
You want to **provision a cluster on non-Azure infrastructure** (on-prem or other clouds) but with Azure-native tooling.

**Real-world scenario:**  
You're building a hybrid solution and want to spin up a Kubernetes cluster **in your own data center** but managed via Azure Arc.

**Key Points:**

* Provisions and connects cluster to Azure
* Azure Arc-enabled Kubernetes
* Extends Azure’s capabilities to non-Azure environments

A screenshot of a computer

AI-generated content may be incorrect.

**✅ 1. Subscription**

**What it is:**  
Your billing account in Azure. Every resource (like AKS) needs to be linked to a subscription to track cost and usage.

**✅ 2. Resource Group**

**What it is:**  
A **logical folder** in Azure to group related resources (VMs, disks, clusters, etc.). All resources in a resource group can be managed together.

**Real-world example:**  
You’re building a shopping app — so you might create a resource group shop-app-prod and include AKS, Load Balancer, and Storage inside it.

**✅ 3. Cluster Preset Configuration**

**What it is:**  
This lets you choose how much control/flexibility you want in setting up the AKS cluster. Azure provides **presets** to help you decide:

**Preset types usually include:**

* **Dev/Test**: For small workloads, fewer nodes, lower cost
* **Production**: Autoscaling, multiple node pools, system/user separation
* **Custom (manual)**: Fully customizable

**Real-world scenario:**

* For a **temporary testing environment**, use **Dev/Test**
* For a **critical production service**, select **Production** or manually configure everything

**What to do:**  
Pick based on your use case. If you’re learning — go with **Dev/Test** first.

**🧱 What is a Node Pool in AKS?**

* A **Node Pool** is a group of nodes (VMs) with the **same configuration** (size, OS, etc.)
* AKS supports **multiple node pools**, allowing different workloads to run on different node types (e.g., GPU vs. CPU, Linux vs. Windows)

**🔹 System Node Pool – Why It’s Needed?**

The **System Node Pool** is **critical for cluster operation**. It runs:

* kube-proxy (networking rules)
* CoreDNS (internal DNS)
* metrics-server
* AKS add-ons like:
  + azure-ip-masq-agent
  + kube-addon-manager
  + coredns-autoscaler

💡 **Best Practice:**  
System pool should be:

* Small (e.g., 1–2 nodes)
* VM size just enough for system workload (e.g., Standard\_DS2\_v2)
* Marked as **"system"** type explicitly

**🔸 User Node Pool – Why It's Important?**

The **User Node Pool** is where **your workloads (Pods)** run:

* You can create multiple pools for different types of workloads:
  + Web apps
  + Batch jobs
  + GPU ML workloads
  + Memory-heavy services

🎯 This enables:

* **Resource isolation** (apps don't compete with system components)
* **Scaling** user workloads independently
* **Cost optimization** by using different VM sizes

**Best Practice:**

* Taint the system node pool to **prevent accidental workload scheduling**:

kubectl taint nodes <node-name> CriticalAddonsOnly=true:NoSchedule

**🧠 Why Separate Node Pools Matter**

**Benefits of separating:**

1. **Reliability** – Prevents app workload issues from impacting system pods
2. **Security** – You can apply RBAC, NSGs, or taints on specific pools
3. **Flexibility** – Use spot VMs for non-critical user node pools
4. **Scaling** – Independently autoscale based on workload

🔧 How to Specify a Node Pool Type in Azure CLI

**Create a system node pool:**

az aks nodepool add \

--cluster-name myAKSCluster \

--resource-group myRG \

--name systempool \

--mode System \

--node-count 1 \

--node-vm-size Standard\_DS2\_v2

**Create a user node pool:**

az aks nodepool add \

--cluster-name myAKSCluster \

--resource-group myRG \

--name userpool \

--mode User \

--node-count 2 \

--node-vm-size Standard\_D4s\_v3

Enabling **spot instances** for an AKS **user node pool** is a great way to reduce costs — especially for **non-critical** or **batch workloads** that can tolerate interruptions.

Spot instances use **Azure's unused capacity** at **much lower prices** — up to **90% cheaper.**

You can add a **user node pool with Spot VMs** using the Azure CLI by setting --priority Spot.

az aks nodepool add \

--resource-group myRG \

--cluster-name myAKSCluster \

--name spotpool \

--mode User \

--priority Spot \

--spot-max-price -1 \

--eviction-policy Delete \

--node-count 1 \

--node-vm-size Standard\_D4s\_v3

**🔍 Parameter Breakdown**

| **Parameter** | **Description** |
| --- | --- |
| --priority Spot | Marks the node pool as Spot |
| --spot-max-price -1 | Pay up to the **current market price** (use a number to set a custom cap) |
| --eviction-policy Delete | When evicted, node is deleted automatically |
| --mode User | Ensures this is a user node pool |
| --node-vm-size | Choose based on your workload needs |

**--eviction-policy can only be Delete (for AKS). It doesn’t support Deallocate.**

**Azure AKS Virtual Nodes**

**What Are AKS Virtual Nodes?**

**Virtual Nodes** in AKS let your Kubernetes cluster **scale rapidly and elastically** — by using **Azure Container Instances (ACI)** as temporary "worker nodes" without actually managing VMs.

Think of them as:

⚡ "Just-in-time, serverless VMs" that run pods instantly when your cluster is full.

**Real-World Analogy**

**Scenario:**

You run a shopping website hosted on AKS.

* On regular days, you run **3 nodes** in your node pool.
* On a big sale day, traffic **spikes massively**, and your 3 nodes can't handle it.
* Normally, you’d autoscale by adding VMs — but that takes 5–10 minutes per VM.

👉 Instead of waiting, **AKS Virtual Nodes** use **ACI** to instantly launch pods **outside your node pool**.

These are your "**on-demand burst pods**".

**🔧 How It Works (Technically)**

1. **You enable Virtual Nodes** in your AKS cluster (requires Azure CNI and a subnet)
2. AKS creates a special **“virtual-kubelet” node**
3. When your cluster is full (node autoscaler kicks in), extra pods are **scheduled onto the virtual node**
4. These pods are actually **running in Azure Container Instances (ACI)**

**✅ Benefits of Virtual Nodes**

| **Feature** | **Benefit** |
| --- | --- |
| ⚡ Fast | Pods start in **seconds**, not minutes |
| 💵 Cost-efficient | Pay only for the **seconds used** in ACI |
| 🧠 No VM management | No need to manage or patch extra nodes |
| 🔄 Auto-scaling | Integrates with HPA/KEDA easily |
| 💼 Great for burst workloads | Ideal for unpredictable traffic/jobs |

**Example Real Case in Fintech**

A payments company has batch jobs that run at 6 PM daily (100+ pods).  
Instead of scaling up 20 VMs (and paying for idle time), they:

* Keep just 5 base VMs
* Use Virtual Nodes to run the 100 pods on-demand
* Finish jobs in minutes and shut down automatically

**Virtual Kubelet + Azure Container Instances (ACI) = Azure Virtual Nodes in AKS**

**🔍 What Each Component Is**

**🔹 Virtual Kubelet**

* A **Kubernetes component** that **pretends to be a node**
* It **forwards pod scheduling requests** to an external container runtime (like ACI)
* Think of it as a **bridge between Kubernetes and ACI**

It's not a real VM — just an interface that makes ACI look like a node to Kubernetes

**Azure Container Instances (ACI)**

* A **serverless container runtime** in Azure
* Lets you run containers without managing VMs
* Charges **per second** for container execution time

Think of it like a "one-shot container runner" in the cloud — no VM required

**➕ Combine the Two:**

When you **enable Virtual Nodes** in AKS:

1. Azure deploys a **Virtual Kubelet** into your AKS cluster
2. It registers as a special node in your cluster like:

**virtual-node-aci-linux**

1. When your cluster runs out of VM capacity, Kubernetes can schedule **extra pods** to this virtual node
2. These pods are actually **executed in ACI**, not on a VM

