



**Independent
Skill Development
Mission**



ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION)

INTRODUCTION TO AZURE VIRTUAL MACHINES (VMs) – TYPES & PRICING

CHAPTER 1: INTRODUCTION TO AZURE VIRTUAL MACHINES

What is an Azure Virtual Machine (VM)?

Azure Virtual Machines (VMs) are **scalable, on-demand computing resources** that provide businesses with **flexibility to run applications without investing in physical infrastructure**. VMs in Azure operate like traditional computers, with an OS, storage, and network capabilities.

Why Use Azure Virtual Machines?

- **Flexibility** – Run different OS types (Windows, Linux, etc.).
- **Scalability** – Easily scale up or down based on workload.
- **Cost Efficiency** – Pay only for the computing power you use.
- **Customization** – Choose VM size, CPU, memory, and storage according to business needs.

📌 **Example:** A software company hosts its web application on **Azure Virtual Machines** to handle traffic spikes during peak hours.

CHAPTER 2: TYPES OF AZURE VIRTUAL MACHINES

Azure VMs are categorized based on **workload requirements**, ensuring that users can select the right VM for their use case.

General-Purpose VMs

- **Best For:** Balanced CPU & memory workloads.
- **Use Cases:** Web servers, small databases, testing environments.

Series	Description
B-Series (Burstable)	Cost-effective for low CPU usage workloads.
D-Series	Optimized for applications requiring a balance of compute, memory, and storage.
Av2-Series	Entry-level VMs for development and testing.

✦ **Example:** A startup deploys a **D-Series VM** to run its small e-commerce platform.

Compute-Optimized VMs

- **Best For:** High CPU-to-memory ratio applications.
- **Use Cases:** Gaming servers, batch processing, high-performance applications.

Series	Description
F-Series	High-performance VMs with powerful CPUs.
HBv3-Series	Optimized for high-performance computing (HPC).

✦ **Example:** A financial analytics firm uses **F-Series VMs** for real-time stock market simulations.

Memory-Optimized VMs

- **Best For:** High memory applications.
- **Use Cases:** Large databases, real-time analytics, SAP workloads.

Series	Description
E-Series	Designed for memory-intensive applications.
M-Series	High memory VMs for in-memory workloads like SAP HANA.

✦ **Example:** A healthcare company runs a **large electronic medical record (EMR) database** using an **M-Series VM**.

Storage-Optimized VMs

- **Best For:** Applications requiring high disk throughput and IOPS.
- **Use Cases:** Big data, analytics, high-performance databases.

Series	Description
Lsv3-Series	High disk throughput, ideal for databases and analytics.
Ls-Series	Designed for applications requiring low-latency storage.

✦ **Example:** A telecom provider stores call records on **Ls-Series VMs** to process customer data efficiently.

GPU-Optimized VMs

- **Best For:** Workloads requiring graphical processing capabilities.
- **Use Cases:** AI/ML workloads, 3D rendering, video editing.

Series	Description
NC-Series	GPU-enabled VMs for deep learning and AI workloads.
NV-Series	Designed for visualization and virtual desktops.

📌 **Example:** A movie production studio uses **NV-Series VMs** for real-time 3D rendering.

High-Performance Computing (HPC) VMs

- **Best For:** High-performance parallel computing workloads.
- **Use Cases:** Scientific simulations, engineering computations, molecular modeling.

Series	Description
HB-Series	Optimized for computational fluid dynamics and other simulations.
HC-Series	Designed for high-performance computing applications.

📌 **Example:** A weather forecasting agency runs **HB-Series VMs** for climate simulations.

CHAPTER 3: AZURE VM PRICING MODELS

Azure provides different **pricing models** to optimize cloud costs.

Pay-As-You-Go Pricing

- No upfront payment.

- Pay only for the hours the VM runs.
- Best for **short-term, unpredictable workloads**.
- **Expensive for long-term use**.

✚ **Example:** A business runs an **on-demand video streaming service** and pays only for **peak usage hours**.

Reserved Instances (RI)

- **Up to 72% discount** compared to Pay-As-You-Go.
- Requires **1 or 3-year commitment**.
- Best for **long-term workloads**.
- **Not flexible; must commit to a specific VM type**.

✚ **Example:** A company hosting an **ERP system** for 3 years purchases a **Reserved Instance**, saving costs.

Spot VMs

- **Up to 90% cheaper** than Pay-As-You-Go.
- Best for **batch processing, testing, and fault-tolerant workloads**.
- **VMs can be interrupted anytime**.

✚ **Example:** A machine learning startup uses **Spot VMs** to train AI models at a low cost.

Azure Hybrid Benefit

- Allows customers with **existing Windows Server or SQL Server licenses** to save up to **85% on VM costs**.
- Best for businesses migrating from **on-premises to Azure**.

📌 **Example:** A company running **Windows Server workloads** moves to Azure and **reuses existing licenses** to reduce costs.

CHAPTER 4: COST ESTIMATION USING AZURE PRICING CALCULATOR

To estimate the cost of an Azure Virtual Machine:

- Go to [Azure Pricing Calculator](#).
- Select **"Virtual Machines"** from the list of Azure services.
- **Configure the VM:**
 - Choose **Region** (e.g., East US).
 - Select **Operating System** (Windows/Linux).
 - Pick **VM Size** (e.g., D2s v3).
 - Set **Hours per month** (e.g., 730 for full-time usage).
- **Review the Estimated Monthly Cost.**
- **Export the Estimate for Budget Planning.**

📌 **Example:** A development team **estimates the monthly cost of running a D2s v3 VM in the East US region before deployment.**

CHAPTER 5: EXERCISE & REVIEW QUESTIONS

Exercise:

- **Estimate the cost** of a Standard B2s Virtual Machine in **East US** using the **Azure Pricing Calculator**.
- **Create a Virtual Machine** in Azure Portal with:
 - **Ubuntu OS**

- **2 vCPUs, 4GB RAM**
- **Standard SSD Storage**
- **Compare the pricing** of a Pay-As-You-Go vs. Reserved Instance for a **D2s v3 VM** in the Azure Calculator.

Review Questions:

- What are the **different types of Azure Virtual Machines** and their use cases?
- How do **Spot VMs** help reduce cloud costs?
- What is the difference between **Pay-As-You-Go, Reserved Instances, and Hybrid Benefit pricing**?
- Which VM series is best for **AI/ML workloads**?
- How can businesses **estimate their VM costs using Azure Pricing Calculator**?

CONCLUSION: CHOOSING THE RIGHT AZURE VM

Understanding **Azure Virtual Machines (VMs)** and pricing models helps businesses:

- Select the **right VM type** for their needs.
- Optimize costs using **pricing models like Reserved Instances & Spot VMs**.
- Improve **performance and scalability** for cloud workloads.

CONFIGURING AND DEPLOYING WINDOWS/LINUX VIRTUAL MACHINES (VMs) ON AZURE

CHAPTER 1: INTRODUCTION TO AZURE VIRTUAL MACHINE DEPLOYMENT

What is an Azure Virtual Machine (VM)?

An **Azure Virtual Machine (VM)** is an **on-demand, scalable computing resource** that allows users to run Windows or Linux operating systems in the cloud. VMs in Azure provide **full control over the computing environment**, making them ideal for hosting applications, databases, and development environments.

Why Deploy a Virtual Machine on Azure?

- **Scalability** – Easily adjust resources based on demand.
- **Cost Efficiency** – Pay only for what you use.
- **Customization** – Choose from a variety of operating systems and configurations.
- **Global Reach** – Deploy applications in **multiple regions worldwide**.

✦ **Example:** A software company deploys **Windows and Linux VMs** on Azure for application development and testing.

CHAPTER 2: PREPARING FOR VM DEPLOYMENT

Prerequisites

Before deploying a VM, ensure you have:

- ✓ **An active Azure account** (Sign up at [Azure Free Tier](#)).
- ✓ **Permissions to create Azure resources** (e.g., Owner or Contributor role).
- ✓ **A Virtual Network (VNet) and Subnet** (to allow network communication).

Choosing Between Windows and Linux VMs

Factor	Windows VM	Linux VM
Best For	Microsoft applications, .NET, SQL Server	Web servers, DevOps, AI/ML workloads
Cost	Higher licensing fees	Lower cost
Performance	Requires more memory	Lightweight and fast
Security	Requires frequent patching	Open-source security updates

✦ **Example:** A financial firm deploys **Windows VMs** for running **SQL Server databases**, while a tech startup uses **Linux VMs** for hosting Python-based applications.

CHAPTER 3: CONFIGURING AND DEPLOYING A WINDOWS VM

Step 1: Log in to Azure Portal

- Open [Azure Portal](#).
- Click "Create a resource" → Select "Virtual Machine".

Step 2: Configure the Windows VM

- **Subscription:** Select **Free Trial** or **Pay-As-You-Go**.

- **Resource Group:** Click **"Create New"** and name it **MyWindowsRG**.
- **Virtual Machine Name:** Enter **WinServerVM**.
- **Region:** Choose **East US** or your nearest location.
- **Availability Options:** Select **No infrastructure redundancy required** (or choose Availability Zones for high availability).
- **Image:** Select **Windows Server 2019 Datacenter**.
- **Size:** Choose **Standard B2s (Free Tier eligible)**.
- **Administrator Account:**
 - Username: adminuser
 - Password: YourSecurePassword123!

Step 3: Configure Networking

- **Public IP:** Enable (to allow remote access).
- **Inbound Ports:** Select **RDP (3389)** to allow remote access.

Step 4: Review and Deploy

- Click **"Review + Create"** → **"Create"**.
- Azure will take a few minutes to deploy the Windows VM.

Step 5: Connect to the Windows VM via RDP

- In **Azure Portal**, go to **Virtual Machines** → Select **WinServerVM**.
- Click **"Connect"** → Choose **RDP** → Download the **RDP file**.
- Open the **RDP file**, enter your **username and password**, and log in.

📌 **Example:** A developer logs into a **Windows Server VM** via **RDP** to install a web application.

CHAPTER 4: CONFIGURING AND DEPLOYING A LINUX VM

Step 1: Log in to Azure Portal

- Open [Azure Portal](#).
- Click "Create a resource" → Select "Virtual Machine".

Step 2: Configure the Linux VM

- **Subscription:** Select **Free Trial** or **Pay-As-You-Go**.
- **Resource Group:** Click "Create New" and name it **MyLinuxRG**.
- **Virtual Machine Name:** Enter **UbuntuVM**.
- **Region:** Choose **East US** or your preferred location.
- **Availability Options:** Choose **Availability Zones** if required.
- **Image:** Select **Ubuntu Server 20.04 LTS**.
- **Size:** Choose **Standard B2s** (Free Tier eligible).
- **Authentication Type:** Choose **SSH Public Key** (recommended) or **Password**.
- **Username:** azureuser.
- If using SSH, generate a key using:
 - `ssh-keygen -t rsa -b 2048`
 - Copy the **public key** into the Azure Portal.

Step 3: Configure Networking

- **Public IP:** Enable (to allow SSH access).
- **Inbound Ports:** Select **SSH (22)** to allow remote access.

Step 4: Review and Deploy

- Click "**Review + Create**" → "**Create**".
- Azure will take a few minutes to deploy the Linux VM.

Step 5: Connect to the Linux VM via SSH

- In **Azure Portal**, go to **Virtual Machines** → Select **UbuntuVM**.
- Copy the **Public IP Address**.
- Open **Terminal (Mac/Linux)** or **PowerShell (Windows)** and run:
- `ssh azureuser@<public-ip-address>`
- If using a private key, run:
- `ssh -i ~/.ssh/id_rsa azureuser@<public-ip-address>`

🔴 **Example:** A DevOps engineer logs into an **Ubuntu VM via SSH** to deploy a Docker container.

CHAPTER 5: POST-DEPLOYMENT CONFIGURATION & BEST PRACTICES

Install Software on the VM

- **Windows:** Install applications via **RDP and PowerShell**.
- **Linux:** Install packages using `apt` (Ubuntu) or `yum` (CentOS):
- `sudo apt update && sudo apt install nginx -y`

Enable Automatic Updates

- **Windows:** Enable updates via **Windows Update settings**.
- **Linux:** Configure unattended-upgrades for automatic security updates.

Secure the VM

- Change default SSH port (Linux):
- `sudo nano /etc/ssh/sshd_config`
- Enable **firewall rules** to limit access.
- `sudo ufw enable && sudo ufw allow 22/tcp`

Monitor VM Performance

- Use **Azure Monitor** to track CPU, memory, and network usage.
- Configure **Alerts** for high resource consumption.

CHAPTER 6: EXERCISE & REVIEW QUESTIONS

Exercise:

- Deploy a **Windows Server 2019 VM** and **connect via RDP**.
- Deploy an **Ubuntu Server 20.04 VM** and **connect via SSH**.
- Install **Nginx (Linux) or IIS (Windows)** to host a basic website.

Review Questions:

- What are the **key differences** between Windows and Linux VMs?
- How do you **connect to a Windows VM via RDP**?
- How do you **SSH into a Linux VM**?

- What are the **best security practices** for managing an Azure VM?
 - How do you **monitor a Virtual Machine's performance in Azure?**
-

CONCLUSION: DEPLOYING AND MANAGING VMS IN AZURE

Deploying **Windows and Linux VMs in Azure** provides businesses with **flexibility, scalability, and security**. By following best practices, users can **optimize performance, reduce costs, and enhance security**.

AZURE APP SERVICE – WEB APPS, API APPS, MOBILE APPS


CHAPTER 1: INTRODUCTION TO AZURE APP SERVICE

What is Azure App Service?

Azure App Service is a **fully managed platform-as-a-service (PaaS)** offering that enables developers to build, deploy, and scale web applications, APIs, and mobile backends without managing infrastructure.

Why Use Azure App Service?

- **Simplified Deployment** – No need to manage servers or OS.
- **Scalability** – Automatically scales based on demand.
- **Security & Compliance** – Integrated **SSL, authentication, and compliance** features.
- **Multi-Language Support** – Works with **.NET, Java, Python, PHP, Node.js, Ruby, and more.**
- **Global Reach** – Deploy apps in **multiple Azure regions** worldwide.


 **Example:** A retail company hosts its **e-commerce website** on Azure App Service, benefiting from high availability and global reach.

CHAPTER 2: TYPES OF APPLICATIONS IN AZURE APP SERVICE

Web Apps


- Used for **hosting websites and web applications.**

- Supports **ASP.NET, Java, PHP, Python, Node.js, and more.**
- Can be **integrated with Azure DevOps and GitHub** for CI/CD deployment.

 **Example:** A blog site running on **WordPress** is hosted on Azure Web Apps.


API Apps

- Used to build and deploy **RESTful APIs.**
- Comes with **built-in security, authentication, and monitoring.**
- Supports **API connectors and integration with Azure Functions.**

 **Example:** A company provides an API that allows users to **retrieve stock market data** in real-time.

Mobile Apps

- Provides **backend support for mobile applications.**
- Offers **offline sync, push notifications, and authentication.**
- Supports **iOS, Android, and cross-platform development frameworks.**

 **Example:** A mobile app for **food delivery services** uses Azure Mobile Apps to handle user authentication and notifications.

CHAPTER 3: DEPLOYING AN AZURE WEB APP

Step 1: Log in to Azure Portal

- Go to [Azure Portal](#).

- Click **"Create a resource"** → Select **"App Service"**.

Step 2: Configure the Web App

- **Subscription:** Select **Free Trial** or **Pay-As-You-Go**.
- **Resource Group:** Click **"Create New"** and name it **MyWebAppRG**.
- **App Name:** Enter a unique name (e.g., mywebapp123).
- **Runtime Stack:** Choose the programming language (e.g., Node.js, Python, .NET).
- **Operating System:** Select **Windows** or **Linux**.
- **Region:** Choose a location close to your audience (e.g., **East US**).

Step 3: Configure Hosting Plan

- **Pricing Plan:** Select **Free (F1)** for testing or **Basic/Standard** for production.
- **Enable Auto-Scaling:** Choose **"Enable Auto Scaling"** for dynamic resource allocation.

Step 4: Deploy the Web App

- Click **"Review + Create"** → **"Create"**.
- Wait for the **Web App deployment to complete**.

Step 5: Access the Web App

- Navigate to **Azure Portal** → **App Services** → **MyWebApp**.
- Click **"Browse"** to view your deployed app in the browser.

 **Example:** A company deploys a **React-based web application** using Azure Web Apps.

CHAPTER 4: DEPLOYING AN API APP IN AZURE

Step 1: Log in to Azure Portal

- Navigate to **Azure Portal** → **App Services**.
- Click "**Create a resource**" → Select "**API App**".

Step 2: Configure the API App

- **Resource Group**: Use an existing one or create **MyAPIAppRG**.
- **App Name**: Enter a unique name (e.g., myapi123).
- **Runtime Stack**: Choose **.NET Core**, **Node.js**, or **Python**.
- **Region**: Select a location (e.g., **West Europe**).

Step 3: Deploy the API

- **Connect to GitHub or Azure DevOps** for CI/CD integration.
- Use **Swagger/OpenAPI** for API documentation.

Step 4: Test the API

- Get the API URL from the Azure Portal.
- Test API endpoints using **Postman** or **Curl**.

✦ **Example:** A logistics company hosts an **API for real-time tracking of shipments** on Azure API Apps.

CHAPTER 5: DEPLOYING AN AZURE MOBILE APP

Step 1: Set Up a Mobile App Backend

- Navigate to **Azure Portal** → Click "**Create a resource**" → Select "**Mobile Apps**".

- Create a **new App Service instance**.
- Choose a **backend stack** (e.g., Node.js, .NET, Python).

Step 2: Enable Offline Sync & Push Notifications

- Use **Azure Notification Hubs** for push notifications.
- Enable **offline sync** to allow mobile apps to store and retrieve data locally.

Step 3: Connect a Mobile App

- Use SDKs for **iOS, Android, or Xamarin** to connect with the backend.
- Secure authentication using **Azure Active Directory or OAuth**.

 **Example:** A healthcare app uses Azure Mobile Apps to **send real-time appointment reminders via push notifications**.

CHAPTER 6: MONITORING & SCALING AZURE APP SERVICE

Monitoring

- Use **Azure Monitor** to track performance metrics.
- Enable **Application Insights** to analyze logs and detect errors.
- Set up **alerts** for downtime notifications.

Scaling

- Configure **Auto-Scaling** based on CPU and memory usage.
- Use **Traffic Manager** for load balancing across multiple regions.

✦ **Example:** A social media app uses **Azure Auto-Scaling** to handle traffic surges during peak hours.

CHAPTER 7: EXERCISE & REVIEW QUESTIONS

Exercise:

- Deploy a **Web App** on Azure and configure it to run a **Node.js** application.
- Deploy an **API App** and test its endpoint using **Postman**.
- Set up **Push Notifications** using **Azure Notification Hubs** for a Mobile App.

Review Questions:

- What are the **key benefits of Azure App Service**?
 - What is the difference between **Web Apps, API Apps, and Mobile Apps**?
 - How do you deploy an **API using Azure App Service**?
 - What is the role of **Azure Notification Hubs** in Mobile Apps?
 - How does **Auto-Scaling** improve application performance?
-

CONCLUSION: POWERING APPLICATIONS WITH AZURE APP SERVICE

Azure App Service is a **versatile, scalable, and fully managed** platform for deploying **Web Apps, APIs, and Mobile Apps**. With built-in **security, monitoring, and global scalability**, it helps businesses rapidly build and deploy cloud applications.

AZURE KUBERNETES SERVICE (AKS) AND AZURE CONTAINER INSTANCES (ACI)

CHAPTER 1: INTRODUCTION TO AZURE CONTAINER SERVICES

What is Containerization?

Containerization is a method of **packaging applications** along with their dependencies into a lightweight, portable environment known as a **container**. This allows applications to run consistently across **different environments (development, testing, and production)**.

Why Use Containers in Azure?

- **Portability** – Run anywhere (on-premises, cloud, or hybrid environments).
- **Scalability** – Handle workload spikes efficiently.
- **Isolation** – Each container runs in its own environment.
- **Fast Deployment** – Quick startup times compared to Virtual Machines (VMs).

📌 **Example:** A fintech company deploys its microservices-based application using **containers**, allowing seamless updates without downtime.

Chapter 2: Overview of Azure Kubernetes Service (AKS)

What is Azure Kubernetes Service (AKS)?

Azure Kubernetes Service (AKS) is a **fully managed Kubernetes solution** that simplifies deploying, managing, and scaling containerized applications. It provides an environment for **running**

containers at scale without needing to manage underlying infrastructure.

Key Features of AKS

- ✓ **Automated Kubernetes Management** – Azure handles updates, scaling, and security patches.
- ✓ **Integrated Monitoring & Logging** – Works with **Azure Monitor, Log Analytics, and Prometheus**.
- ✓ **Built-in Security** – Supports **RBAC (Role-Based Access Control), Azure Active Directory, and Private Clusters**.
- ✓ **Auto-Scaling & Load Balancing** – Automatically adjusts resources based on demand.

✚ **Example:** An e-commerce company runs its **microservices-based checkout system on AKS**, ensuring high availability and auto-scaling during peak sales periods.

How AKS Works?

1. **Developers package applications into containers** using Docker.
2. **Containers are deployed into Kubernetes Pods** for execution.
3. **AKS manages networking, scaling, and security** for the cluster.
4. **Users access the application via public or private endpoints.**

CHAPTER 3: DEPLOYING AZURE KUBERNETES SERVICE (AKS)

Step 1: Log in to Azure Portal

- Open [Azure Portal](#).

- Click **"Create a resource"** → Search for **"Kubernetes Service"** → Click **Create**.

Step 2: Configure the AKS Cluster


- **Subscription:** Select **Azure Free Trial** or **Pay-As-You-Go**.
- **Resource Group:** Click **"Create New"** and name it **MyAKSClusterRG**.
- **Cluster Name:** Enter a unique name (e.g., **MyAKSCluster**).
- **Region:** Select a location (e.g., **East US**).
- **Node Size:** Choose **Standard_DS2_v2 (2 vCPUs, 7GB RAM)**.
- **Node Count:** Start with **3 nodes** (can be scaled later).
- **Enable Auto-Scaling:** Set **Min: 3, Max: 5 nodes**.

Step 3: Deploy the AKS Cluster

- Click **"Review + Create"** → **"Create"**.
- Wait for deployment to complete (5–10 minutes).

Step 4: Connect to AKS Cluster

- Open **Azure Cloud Shell** and run:
- `az aks get-credentials --resource-group MyAKSClusterRG --name MyAKSCluster`
- Verify cluster status:
- `kubectl get nodes`

 **Example:** A logistics company deploys **Kubernetes workloads on AKS**, using auto-scaling to manage increased traffic.

CHAPTER 4: OVERVIEW OF AZURE CONTAINER INSTANCES (ACI)

What is Azure Container Instances (ACI)?

Azure Container Instances (ACI) is a **serverless container runtime** that enables users to deploy and run **containers without managing virtual machines or Kubernetes clusters**.

Key Features of ACI

- ✓ **No Infrastructure Management** – Run containers without setting up VMs or clusters.
- ✓ **Fast Deployment** – Spin up containers **in seconds**.
- ✓ **Billing by the Second** – Pay only for the exact resources used.
- ✓ **Secure & Isolated** – Supports **VNET integration and private endpoints**.

📌 **Example:** A data analytics firm runs **one-time batch processing jobs** using ACI, reducing costs compared to running full-time VMs.

When to Use ACI Instead of AKS?

Feature	Azure Kubernetes Service (AKS)	Azure Container Instances (ACI)
Best for	Large-scale, long-running applications	Temporary, event-driven workloads
Infrastructure	Requires node pools, clusters	No infrastructure setup
Scaling	Auto-scaling based on traffic	Runs single containers
Cost	More expensive, but scalable	Pay-per-use, cost-efficient for short tasks

CHAPTER 5: DEPLOYING AN AZURE CONTAINER INSTANCE (ACI)

Step 1: Log in to Azure Portal

- Go to [Azure Portal](#).
- Click “Create a resource” → Select “Container Instances”.

Step 2: Configure the Container Instance

- **Resource Group:** Create **MyACIRG**.
- **Container Name:** Enter **MyContainerInstance**.
- **Region:** Select **East US**.
- **Image Source:** Choose **Quickstart Image** or enter your **Docker Hub image URL** (e.g., nginx).
- **CPU & Memory:** Set **1 vCPU, 1.5 GB RAM**.

Step 3: Deploy the Container

- Click “Review + Create” → “Create”.
- Azure will deploy the container in **seconds**.

Step 4: Access the Container

- Navigate to **Azure Portal** → **Container Instances** → **MyContainerInstance**.
- Copy the **public IP address** and access it via a browser.

📌 **Example:** A marketing company deploys **temporary data processing tasks in ACI**, reducing infrastructure costs.

CHAPTER 6: MONITORING AND SCALING CONTAINERS IN AZURE

Monitoring AKS and ACI

- **Azure Monitor** – Tracks CPU, memory, and container health.

- **Azure Log Analytics** – Analyzes logs for troubleshooting.
- **Application Insights** – Monitors app performance inside containers.

Scaling Strategies

- **AKS Auto-Scaling** – Automatically adjusts cluster size.
- **Horizontal Pod Autoscaler (HPA)** – Adds/removes Kubernetes pods dynamically.
- **ACI Scaling via Logic Apps** – Triggers more instances as demand increases.

✚ **Example:** A streaming platform uses **AKS auto-scaling** to handle thousands of concurrent users during live events.

CHAPTER 7: EXERCISE & REVIEW QUESTIONS

Exercise:

- Deploy an **AKS Cluster** with **three nodes** and connect to it using kubectl.
- Deploy an **Azure Container Instance (ACI)** running an **Nginx web server**.
- Scale an **AKS deployment** using the Horizontal Pod Autoscaler (HPA).

Review Questions:

- What are the **main differences between AKS and ACI**?
- How does **Azure Kubernetes Service (AKS)** handle **auto-scaling**?

- What are the **benefits of Azure Container Instances (ACI) for short-term workloads?**
 - How do you deploy a **Docker container in ACI?**
 - What monitoring tools can be used for **tracking Kubernetes and container performance?**
-

CONCLUSION: CHOOSING THE RIGHT CONTAINER SOLUTION IN AZURE

- **Use AKS for large-scale, long-running, production workloads.**
- **Use ACI for temporary, event-driven tasks** with minimal infrastructure.
- **Monitor and scale efficiently using Azure Monitor, Log Analytics, and Kubernetes auto-scaling tools.**

SERVERLESS COMPUTING – AZURE FUNCTIONS & LOGIC APPS

CHAPTER 1: INTRODUCTION TO SERVERLESS COMPUTING

What is Serverless Computing?

Serverless computing is a **cloud-native execution model** that allows developers to build and run applications without managing infrastructure. Cloud providers handle **server provisioning, scaling, and maintenance**, enabling developers to focus on writing code.

Why Use Serverless Computing?

- **No Infrastructure Management** – No need to maintain servers or VMs.
- **Scalability** – Automatically scales based on demand.
- **Cost-Efficiency** – Pay only for the execution time of functions (pay-as-you-go model).
- **Event-Driven Architecture** – Functions are triggered by events such as HTTP requests, database changes, or timers.

✚ **Example:** A travel booking website uses **serverless Azure Functions** to process **real-time flight booking confirmations** when users make payments.

CHAPTER 2: OVERVIEW OF AZURE FUNCTIONS

What are Azure Functions?

Azure Functions is a **serverless compute service** that allows developers to run small, **event-driven pieces of code** in the cloud without managing infrastructure. It supports multiple programming languages such as **C#, Python, Java, JavaScript, and PowerShell**.

Key Features of Azure Functions

- ✓ **Supports Multiple Triggers** – HTTP requests, Queue messages, Event Grid events, and more.
 - ✓ **Pay-Per-Use Pricing** – Charged only for execution time, making it cost-efficient.
 - ✓ **Automatic Scaling** – Functions scale dynamically based on demand.
 - ✓ **Integration with Other Azure Services** – Works with **Azure Storage, SQL, Cosmos DB, and Event Hub**.
- 📌 **Example:** A retail store uses **Azure Functions** to send **automated order confirmation emails** when customers place an order.
-

CHAPTER 3: DEPLOYING AN AZURE FUNCTION

Step 1: Log in to Azure Portal

- Go to [Azure Portal](#).
- Click "Create a resource" → Search for "Function App" → Click "Create".

Step 2: Configure the Function App

- **Subscription:** Select **Azure Free Trial** or **Pay-As-You-Go**.
- **Resource Group:** Click "Create New", name it **MyFunctionRG**.

- **Function App Name:** Enter a unique name (e.g., MyFunctionApp).
- **Region:** Choose the closest Azure region (e.g., **East US**).
- **Runtime Stack:** Choose the programming language (**Python, C#, JavaScript, Java**).
- **Plan Type:** Select **Consumption Plan** (pay only for execution time).

Step 3: Deploy a Sample Function

- Navigate to **Functions** → Click "**Create**".
- Choose "**HTTP Trigger**" → Name it **HttpFunctionTest**.
- Select **Anonymous Access** (for testing).

Step 4: Test the Azure Function

- Copy the **Function URL** and paste it into a browser:
- `https://myfunctionapp.azurewebsites.net/api/HttpFunctionTest?name=John`
- You should receive a response:
- `{"message": "Hello, John" }`

✦ **Example:** A chatbot service uses an Azure Function to process user requests and return personalized responses in real time.

CHAPTER 4: OVERVIEW OF AZURE LOGIC APPS

What are Azure Logic Apps?

Azure Logic Apps is a **low-code/no-code** service that allows businesses to automate workflows and integrate applications and data across different cloud services.

Key Features of Azure Logic Apps

- ✓ **Workflow Automation** – Automate processes like data syncing, email notifications, and approvals.
- ✓ **Built-in Connectors** – Integrates with **Office 365, Dynamics 365, Salesforce, and more.**
- ✓ **Event-Driven Execution** – Triggers workflows based on **HTTP requests, database updates, file uploads, etc.**
- ✓ **Drag-and-Drop Interface** – No coding required, making it easy for non-developers.

📌 **Example:** A finance company automates **invoice approvals** using **Logic Apps**, where a new invoice in SharePoint triggers an approval request in Teams.

CHAPTER 5: CREATING AN AZURE LOGIC APP

Step 1: Log in to Azure Portal

- Open [Azure Portal](#).
- Click **"Create a resource"** → Search for **"Logic App"** → Click **"Create"**.

Step 2: Configure the Logic App

- **Subscription:** Select **Azure Free Trial** or **Pay-As-You-Go**.
- **Resource Group:** Choose an existing one or create **MyLogicAppRG**.
- **Logic App Name:** Enter **MyLogicWorkflow**.

- **Region:** Choose the nearest Azure region.
- **Plan Type:** Select **Consumption Plan** (serverless).

Step 3: Design a Simple Workflow

- Open **Logic Apps Designer**.
- Select **“When an HTTP request is received”** as a trigger.
- Add an **Action:** Choose **“Send an email (Office 365)”**.
- Enter the **email details** (recipient, subject, body).

Step 4: Test the Logic App

- Save and run the Logic App.
- Send an HTTP request using **Postman** or a browser.
- Check the recipient’s email inbox.

📌 **Example:** A customer support team uses **Logic Apps** to **automatically send email responses** when a support ticket is created.

Chapter 6: Comparing Azure Functions and Logic Apps

Feature	Azure Functions	Azure Logic Apps
Best for	Running code in response to events	Automating workflows without coding
Execution Mode	Event-driven serverless function	Workflow automation

Pricing	Pay-per-use (execution time)	Pay-per-action workflow pricing
Common Triggers	HTTP, Event Grid, Queue storage	Email, SharePoint, SQL, API calls
Integration	Ideal for coding-based automation	Ideal for no-code automation

📌 **Example:** A business automates **invoice processing** by combining **Logic Apps (workflow automation)** with **Azure Functions (data processing)**.


CHAPTER 7: MONITORING & SCALING SERVERLESS APPLICATIONS

Monitoring Tools

- **Azure Monitor** – Tracks function execution and performance.
- **Application Insights** – Analyzes logs and detects failures.
- **Logic Apps Run History** – Tracks workflow execution and errors.

Scaling Serverless Applications

- **Azure Functions Auto-Scaling** – Automatically scales based on execution demand.
- **Logic Apps Parallel Execution** – Runs multiple workflows concurrently.
- **Event-Driven Scaling** – Use **Event Grid** to trigger additional functions.

 **Example: A banking app scales Azure Functions to process millions of transactions in real time.**

CHAPTER 8: EXERCISE & REVIEW QUESTIONS

Exercise:

- Deploy an **Azure Function** that returns a JSON response for HTTP requests.
- Create a **Logic App** that sends an email when a new **Azure Storage Blob** is uploaded.
- Set up **monitoring** for an **Azure Function** using **Application Insights**.

Review Questions:

- What are the **advantages of serverless computing** in Azure?
 - How do **Azure Functions** differ from **Azure Logic Apps**?
 - What types of applications benefit most from **Azure Functions**?
 - What are the **pricing differences** between Functions and Logic Apps?
 - How can you **monitor and debug serverless applications** in Azure?
-

CONCLUSION: EMBRACING SERVERLESS WITH AZURE FUNCTIONS & LOGIC APPS

Azure Functions and Logic Apps **simplify development, reduce costs, and improve scalability**. Whether you need **event-driven execution (Functions) or workflow automation (Logic Apps)**, Azure serverless solutions enable businesses to build powerful, efficient applications.

MONITORING AND SCALING AZURE COMPUTE SERVICES

CHAPTER 1: INTRODUCTION TO MONITORING AND SCALING AZURE COMPUTE SERVICES

Understanding the Importance of Monitoring and Scaling in Azure

Azure provides robust compute services that allow businesses to run workloads efficiently in the cloud. However, to maintain **high availability, performance, and cost-effectiveness**, organizations must implement **monitoring and scaling strategies**.

Monitoring ensures that administrators can track system performance, detect bottlenecks, and proactively address issues before they impact end users. Scaling, on the other hand, enables businesses to **dynamically adjust computing resources** based on demand, ensuring that applications remain responsive while optimizing costs.

Why Monitoring and Scaling Matter?

- **Ensure Application Availability:** Prevents downtime by proactively detecting and resolving system failures.
- **Optimize Performance:** Ensures that compute resources operate at peak efficiency.
- **Reduce Costs:** Adjusts resource usage based on demand, avoiding over-provisioning.
- **Improve Security:** Detects unauthorized access attempts and other anomalies.



Example:

A **healthcare platform** running on Azure Virtual Machines (VMs) experiences traffic spikes during telemedicine consultations. By

implementing **Azure Monitor** and **Auto-Scaling**, the system ensures smooth performance by automatically adding VMs when user load increases and scaling down during off-peak hours.

CHAPTER 2: MONITORING AZURE COMPUTE SERVICES

Overview of Azure Monitor

Azure Monitor is a **cloud-based telemetry and observability solution** that helps track performance metrics, collect logs, and trigger alerts for various Azure compute services. It integrates seamlessly with **Azure Virtual Machines (VMs)**, **Azure Kubernetes Service (AKS)**, **Azure App Services**, and other Azure resources.

Key Features of Azure Monitor

- ✓ **Metrics and Logs Collection** – Monitors CPU, memory, disk usage, network latency, and system logs.
- ✓ **Application Insights** – Tracks application health and user interactions.
- ✓ **Alerting System** – Notifies administrators about critical system issues.
- ✓ **Integration with Power BI & Logic Apps** – Enables custom dashboards and workflow automation.



Example:

A financial services company uses **Azure Monitor with Application Insights** to track transaction failures in its online banking system. If the transaction failure rate exceeds 5%, an alert is triggered, and a response workflow is executed using Azure Logic Apps.

CHAPTER 3: SCALING AZURE COMPUTE SERVICES

Types of Scaling in Azure

Azure supports two main scaling strategies:

1. Vertical Scaling (Scale Up/Down)

- Increases or decreases the compute resources (CPU, RAM, storage) of a single instance.
- Suitable for **database servers and memory-intensive applications**.



Example:

A **data analytics firm** running large SQL queries on **Azure SQL Database** scales up its **compute tier** from **Standard** to **Premium** when processing complex queries and scales down afterward.

2. Horizontal Scaling (Scale Out/In)

- Adds or removes multiple instances of a resource to balance the workload.
- Suitable for **web applications, APIs, and microservices architectures**.



Example:

An **e-commerce website** running on **Azure Virtual Machine Scale Sets (VMSS)** automatically **scales out** during Black Friday sales and **scales in** once traffic stabilizes.

Scaling Methods in Azure

- ✓ **Manual Scaling** – Manually increase or decrease the number of compute resources.
- ✓ **Scheduled Scaling** – Define rules to scale resources at specific times.
- ✓ **Auto-Scaling** – Automatically adjusts resources based on performance metrics.

CHAPTER 4: IMPLEMENTING AUTO-SCALING IN AZURE

Configuring Auto-Scaling for Azure Virtual Machines

Step 1: Log in to Azure Portal

- Open the **Azure Portal** and navigate to **Virtual Machine Scale Sets**.
- Click **Create a resource** → Select **Virtual Machine Scale Sets**.

Step 2: Configure Scaling Policies

- Select **Auto-scale based on CPU utilization**.
- Set thresholds: **Scale out at 70% CPU usage, Scale in at 30%**.
- Define **maximum and minimum instance limits**.

Step 3: Test Auto-Scaling

- Simulate **high traffic** using load testing tools.
- Observe how the **scale set dynamically adjusts instances**.



Example:

A **video streaming platform** scales Azure VMs when a new movie release generates heavy traffic, ensuring smooth video playback without manual intervention.

Chapter 5: Case Study – Scaling a Cloud-Based Learning Platform

Problem Statement:

An **online education platform** experiences **unpredictable traffic spikes** due to students accessing video lectures simultaneously.

Solution:

1. **Implemented Azure Kubernetes Service (AKS)** to manage containerized microservices.
2. **Configured Auto-Scaling Policies** to add more pods during peak hours.
3. **Integrated Azure Monitor** to track CPU and memory usage.
4. **Set up Alert Rules** to notify admins if **server response time** exceeds 2 seconds.

Results:

- ✓ **99.8% application uptime**, improving student engagement.
- ✓ **Reduced operational costs by scaling down resources** during non-peak hours.
- ✓ **Faster response times**, enhancing the learning experience.

CHAPTER 6: COMPARING AZURE COMPUTE SCALING SOLUTIONS

Feature	Virtual Machine Scale Sets (VMSS)	Azure Kubernetes Service (AKS)	Azure App Service Scaling
Best for	Traditional VMs workloads	Microservices and containers	Web applications
Scaling Type	Horizontal	Horizontal	Horizontal & Vertical
Auto-Scaling	Based on CPU, memory	Based on container usage	Based on HTTP traffic
Cost Efficiency	Moderate	High	High

Example Use Case	Hosting enterprise applications	Managing microservices	Running web applications
-------------------------	---------------------------------	------------------------	--------------------------

**Example:**

A logistics company uses **VM Scale Sets** for its internal tracking system, while its customer-facing mobile **API** is hosted on **Azure App Services** with **Auto-Scaling**.

CHAPTER 7: MONITORING & SCALING BEST PRACTICES

Monitoring Best Practices

- ✓ **Set Up Alerts** – Configure alerts for CPU, memory, and disk usage thresholds.
- ✓ **Use Log Analytics** – Aggregate logs from multiple services for deeper insights.
- ✓ **Enable Distributed Tracing** – Track requests across microservices.

Scaling Best Practices

- ✓ **Use Auto-Scaling** – Set dynamic rules to optimize resource usage.
- ✓ **Define Scaling Limits** – Prevent unnecessary over-provisioning.
- ✓ **Optimize Workloads** – Use **Azure Advisor** to recommend cost-efficient scaling strategies.

**Example:**

A retail chain uses **Azure Log Analytics** to detect slow database queries and triggers **Azure Function auto-scaling** to improve query performance during peak hours.

CHAPTER 8: EXERCISE & REVIEW QUESTIONS

Exercise:

1. Configure an **Auto-Scaling Policy** for an **Azure Virtual Machine Scale Set (VMSS)**.
2. Deploy an **Azure Kubernetes Cluster (AKS)** and enable **Horizontal Pod Auto-Scaling**.
3. Set up **Azure Monitor Alerts** for CPU usage exceeding 80%.

Review Questions:

1. What are the **differences between Vertical and Horizontal Scaling** in Azure?
2. How does **Azure Monitor** help optimize compute performance?
3. When should you use **Azure VMSS vs. Azure App Service Auto-Scaling**?
4. How can **Event-Driven Scaling** improve cost-efficiency in cloud applications?
5. What are the best practices for **monitoring Azure Compute Services**?

CONCLUSION: ENSURING EFFICIENT COMPUTE PERFORMANCE IN AZURE

By leveraging **Azure Monitor** and **Auto-Scaling** capabilities, businesses can ensure **high availability, optimal performance, and cost efficiency**. Whether using **VMSS, AKS, or App Services**, implementing the right monitoring and scaling strategies helps organizations **maintain seamless operations in the cloud**.

ASSIGNMENT

DEPLOY A SCALABLE WEB APPLICATION USING AZURE APP SERVICE

ISDM-NxT

SOLUTION: DEPLOY A SCALABLE WEB APPLICATION USING AZURE APP SERVICE

Step-by-Step Guide

Step 1: Access Azure Portal

To deploy a scalable web application using Azure App Service, you first need access to the Azure Portal.

1.1 Open Azure Portal

- Go to [Azure Portal](#).
- Sign in with your Microsoft Azure account credentials.
- In the Azure Portal dashboard, use the search bar and type **"App Services"** to access the App Service panel.

✚ **Example:** A startup company wants to host a scalable e-commerce web application and needs to set up an App Service for auto-scaling during peak shopping hours.

Step 2: Create an Azure App Service

Azure App Service is a fully managed platform for building, deploying, and scaling web applications.

2.1 Select 'Create a Resource'

- Click **"Create a resource"** from the Azure Portal.
- Search for **"App Service"** and select it from the list.
- Click **"Create"** to start configuring the web application.

Step 3: Configure Web Application Settings


To ensure the web application is scalable, configure its essential settings.

3.1 Choose Subscription & Resource Group

- Select the **Azure Subscription** (e.g., Pay-As-You-Go).
- Choose an **existing Resource Group** or create a new one (e.g., "MyAppResourceGroup").

3.2 Configure Web App Name and Hosting

- **Web App Name:** Choose a unique name (e.g., "MyScalableWebApp").
- **Publish Type:** Select **Code** (for Node.js, Python, etc.) or **Docker Container** for containerized applications.
- **Runtime Stack:** Choose the runtime (e.g., .NET, Node.js, Python, Java).
- **Operating System:** Select **Windows** or **Linux** based on your web app's requirements.
- **Region:** Choose the nearest Azure region to minimize latency (e.g., East US).

 **Example:** A SaaS company selects **Node.js** with **Linux hosting** to deploy its real-time chat application.

Step 4: Configure App Service Plan for Scaling

Azure App Service Plan determines the compute resources allocated to the web app.

4.1 Select Pricing Plan

- Choose the appropriate **App Service Plan** (Basic, Standard, or Premium).
- Select an **auto-scaling tier** to adjust resources dynamically.

4.2 Enable Auto-Scaling

- Navigate to **Scale Out** in the App Service settings.
- Set auto-scaling rules based on:
 - **CPU Usage (e.g., scale up when CPU > 70%)**
 - **Memory Consumption**
 - **HTTP Request Load**
- Define the **minimum and maximum number of instances**.

✚ **Example:** A travel booking platform configures auto-scaling to add more instances when CPU usage exceeds 75%, ensuring high availability during peak traffic.

Step 5: Deploy the Web Application

5.1 Deploy from GitHub or Azure DevOps

- In **Deployment Center**, select your deployment source (GitHub, Azure DevOps, or FTP).
- Connect your repository and set up CI/CD for automated deployments.

5.2 Configure Environment Variables & App Settings

- Define environment variables such as database connection strings.

- Use **Azure Key Vault** for securing credentials.

✚ **Example:** A financial analytics web app pulls sensitive database credentials securely using **Azure Key Vault**.

Step 6: Monitor and Optimize Performance

Monitoring is crucial for ensuring scalability and cost-effectiveness.

6.1 Enable Azure Application Insights

- Navigate to **Monitoring** → **Application Insights**.
- Enable logging for performance and error tracking.

6.2 Set Up Alerts for Auto-Scaling

- Configure **Azure Monitor** to send alerts when resource usage spikes.

✚ **Example:** A sports streaming service uses **Azure Monitor** to track viewer traffic and scale resources automatically during live games.

Step 7: Implement Security & Backup Strategies

7.1 Enable HTTPS & Authentication

- Enforce **HTTPS-only connections**.
- Configure **Azure Active Directory (Azure AD)** for authentication.

7.2 Configure Backups

- Enable **Azure Backup** to store snapshots of web app data.
- Set up **Geo-Replication** for disaster recovery.

✚ **Example:** A university's learning management system (LMS) enables **geo-redundant backup** to prevent data loss.

Exercise

1. Deploy a web app using Azure App Service with the following:
 - **Runtime:** Python
 - **Scaling:** Auto-scale with a minimum of 1 instance and a maximum of 5
 - **Monitoring:** Enable Application Insights
 2. Configure **environment variables** and deploy a sample **Django** application from GitHub.
 3. Enable **custom domain & SSL certificate** for secure access.
-

CASE STUDY: SCALING A FOOD DELIVERY APP WITH AZURE APP SERVICE

Background

A food delivery startup, "**QuickEats**", experiences fluctuating traffic, with peak orders during lunch and dinner hours. The company needs an auto-scaling solution to handle high traffic without over-provisioning resources.

Solution Using Azure App Service

1. **Deployed Web App:** Hosted on Azure App Service using **.NET Core** and **SQL Database**.
2. **Configured Auto-Scaling:**
 - Scales up to **8 instances** when CPU exceeds **80%**.

- Scales down to **1 instance** during off-hours to reduce costs.
- 3. **Integrated Azure CDN:** Ensured **faster load times** for menu images.
- 4. **Implemented Azure Application Insights:** Tracked response time and optimized SQL queries.

Outcome

- ✓ **Improved performance:** Reduced app response time by **30%**.
- ✓ **Cost efficiency:** Saved **40%** on infrastructure costs by scaling down during off-peak hours.
- ✓ **Higher reliability:** **99.99% uptime** during peak usage.
- 📌 **Key Takeaway:** Azure App Service's auto-scaling and monitoring capabilities enabled QuickEats to efficiently handle thousands of orders without service disruption.

✓ **Congratulations! You have successfully deployed and scaled a web application using Azure App Service.** 🎉

SET UP AN AZURE FUNCTION TO TRIGGER AUTOMATED TASKS

ISDM-NxT

SOLUTION: SET UP AN AZURE FUNCTION TO TRIGGER AUTOMATED TASKS

Step-by-Step Guide

Step 1: Access the Azure Portal

Azure Functions is a **serverless compute service** that enables users to run event-driven code without managing infrastructure. It is used to automate workflows, process data, and integrate with other services.

1.1 Open the Azure Portal

- Navigate to [Azure Portal](#).
- Sign in with your **Azure account credentials**.

1.2 Create a New Azure Function App

- In the **Azure Portal**, search for "**Function App**" in the search bar.
- Click **+ Create** to start setting up a new function.
- Choose the correct **Subscription** and **Resource Group** (or create a new one).

Step 2: Configure the Azure Function App

Azure Functions require configuration before deployment.

2.1 Basic Settings

- **Function App Name:** Enter a unique name (e.g., MyAutomationFunction).
- **Region:** Choose a location (e.g., **East US**).
- **Runtime Stack:** Select the programming language (e.g., Python, Node.js, or C#).
- **Operating System:** Choose **Windows** or **Linux** based on the environment.

2.2 Hosting Options

- **Plan Type:** Choose **Consumption Plan** (pay per execution) for cost savings.
- **Storage Account:** Use an existing **Azure Storage Account** or create a new one.

2.3 Monitoring

- Enable **Application Insights** for real-time monitoring.

✦ **Example:** A company sets up an Azure Function in **East US** with **Node.js runtime** to automate order processing from a database.

Step 3: Define the Trigger for the Azure Function

Azure Functions supports multiple triggers such as HTTP requests, timers, and event-based triggers.

3.1 Choose a Trigger Type


Click "Add Function" and select a trigger type:

- **HTTP Trigger:** Runs when an HTTP request is received.
- **Timer Trigger:** Executes on a schedule (e.g., every 5 minutes).

- **Blob Storage Trigger:** Runs when a new file is uploaded to Blob Storage.
- **Queue Trigger:** Processes messages from an Azure Queue Storage.

3.2 Configure the Trigger

- For a **Timer Trigger**, enter a cron expression (e.g., `0 */5 * * * *` runs every 5 minutes).
- For an **HTTP Trigger**, set the authorization level (Anonymous, Function, or Admin).

 **Example:** A timer-based function runs every night at **2:00 AM** (`0 0 2 * * *`) to generate automated reports.

Step 4: Write the Function Code

Azure Functions can be written in various languages like **C#, Python, Java, or JavaScript**.

4.1 Example Code for an HTTP Trigger (JavaScript)

```
module.exports = async function (context, req) {  
    context.res = {  
        status: 200,  
        body: "Azure Function executed successfully!"  
    };  
};
```

4.2 Example Code for a Timer Trigger (Python)

```
import datetime
```

```
import logging
```

```
def main(myTimer: func.TimerRequest) -> None:
```

```
    utc_timestamp = datetime.datetime.utcnow().isoformat()
```

```
    logging.info(f"Timer function executed at {utc_timestamp}")
```

✦ **Example:** A **Python Timer Function** runs every Monday at 9 AM to clean up temporary database records.

Step 5: Deploy the Function

Once the function is written, it needs to be deployed.

5.1 Deploy Using the Azure Portal

- Click **Save and Run** to test the function in the portal.
- Deploy automatically through **Azure DevOps or GitHub Actions**.

5.2 Deploy Using VS Code (Optional)

- Install the **Azure Functions Extension** in VS Code.
- Sign in to **Azure Account**.
- Deploy the function directly from VS Code.

✦ **Example:** A development team deploys an **Azure Function via GitHub Actions** to automate error logging in a production system.

Step 6: Test and Monitor the Azure Function

Once the function is deployed, testing ensures it runs correctly.

6.1 Test Execution

- **For HTTP Trigger:** Open the function URL in a browser or send a request using Postman.
- **For Timer Trigger:** Wait for the next scheduled execution or manually trigger it.

6.2 Monitor Execution Logs

- Navigate to the **Azure Function App > Monitoring > Logs**.
- Use **Application Insights** to track performance and failures.

📌 **Example:** An e-commerce site sets up a function to send **automated emails** when a new order is placed and monitors it via Application Insights.

Step 7: Optimize and Scale Azure Function

Azure Functions offer multiple cost and performance optimization features.

7.1 Cost Optimization

- Use **Consumption Plan** to pay only for executed functions.
- Optimize function execution time to reduce billing.
- Use **Azure Durable Functions** for long-running workflows.

7.2 Scaling the Function

- **Auto-scaling:** Azure Functions automatically scale based on demand.
- **Function Throttling:** Set execution limits to prevent overuse of resources.

✦ **Example:** A stock trading company optimizes costs by **throttling function executions** to prevent exceeding API request limits.

CASE STUDY: AUTOMATING DATA PROCESSING WITH AZURE FUNCTIONS

Problem Statement:

A financial company wanted to **automate daily report generation** without maintaining expensive servers.

Solution:

1. **Deployed a Timer Trigger Azure Function** to run at **midnight daily**.
2. The function **processed financial transactions** from an **Azure SQL Database**.
3. It **generated and stored reports** in **Azure Blob Storage**.
4. Sent automated email notifications with **download links**.

Results:

- **Reduced operational costs by 60%** compared to traditional servers.
 - **Fully automated reporting system**, eliminating manual work.
 - **Improved efficiency**, generating reports **30% faster** than before.
-

Exercise

Question 1:


What is the difference between **HTTP Trigger** and **Timer Trigger** in Azure Functions? Provide a use case for each.

Question 2:

Which **Azure pricing plan** is best suited for a function that runs **once per day**? Why?

Question 3:

A logistics company wants to **automatically process new order files** uploaded to **Azure Blob Storage**. Which Azure Function trigger should they use?

 **Congratulations!** You have successfully set up an Azure Function to trigger automated tasks. 