Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

INTRODUCTION TO DEVOPS & AGILE DEVELOPMENT IN AZURE

CHAPTER 1: UNDERSTANDING DEVOPS AND AGILE DEVELOPMENT

What is DevOps?

DevOps is a **software development approach** that combines **development (Dev) and IT operations (Ops)** to improve **software quality, accelerate delivery, and enhance collaboration** between teams. It focuses on **continuous integration (CI), continuous delivery (CD), and automation**.

What is Agile Development?

Agile development is a **software development methodology** that emphasizes **iterative progress, customer collaboration, and flexibility**. It follows frameworks such as **Scrum and Kanban** to deliver software in small, manageable increments.

How DevOps and Agile Work Together?

✓ **Agile** focuses on **incremental development**, while **DevOps** ensures **efficient deployment and operations**.

✓ **Agile teams** plan and prioritize work, while **DevOps automates builds, testing, and releases**.

✓ Together, they create a **seamless workflow** from **planning to deployment**.

📌 **Example:**
A **SaaS company** follows Agile **Scrum** for feature development and uses **DevOps CI/CD pipelines** to release updates frequently without manual intervention.

CHAPTER 2: BENEFITS OF DEVOPS AND AGILE IN AZURE

Key Advantages of DevOps

✓ **Faster Software Releases:** Automates **CI/CD pipelines** for rapid deployment.
✓ **Improved Collaboration:** Bridges the gap between **developers, testers, and IT operations**.
✓ **Higher Quality Software:** Continuous **testing and monitoring** prevent failures.
✓ **Scalability & Flexibility:** Supports **cloud-based, on-premises, and hybrid deployments**.

Key Advantages of Agile Development

✓ **Customer-Centric Development:** Regular feedback loops ensure **feature relevance**.
✓ **Incremental Updates:** Releases new features **every few weeks instead of months**.
✓ **Adaptability:** Adjusts project scope **based on changing requirements**.

📌 **Example:**
A **financial services company** follows Agile to **prioritize new regulatory features** and deploys them using **DevOps Pipelines** to ensure compliance.

Chapter 3: DevOps Principles and Agile Frameworks in Azure

Core DevOps Principles

✔ **Continuous Integration (CI):** Developers merge code frequently, triggering automated builds and tests.
✔ **Continuous Deployment (CD):** Automates software deployment to production environments.
✔ **Infrastructure as Code (IaC):** Manages infrastructure using scripts (e.g., Terraform, ARM templates).
✔ **Monitoring & Feedback:** Uses **Azure Monitor, Application Insights, and Log Analytics** for real-time tracking.

Agile Frameworks in Azure

✔ **Scrum:** Uses **Sprints, Product Backlogs, and Daily Standups** to manage development.
✔ **Kanban:** Visualizes workflow using **Boards and Work Items** to track progress.
✔ **Extreme Programming (XP):** Focuses on **code reviews, unit testing, and rapid releases**.

📌 **Example:**
A **healthcare startup** implements **Scrum** in **Azure DevOps Boards** and uses **CI/CD Pipelines** for frequent feature releases.

Chapter 4: Implementing DevOps in Azure

Step 1: Set Up Azure DevOps Organization

1. Go to **Azure DevOps Portal** (dev.azure.com).
2. Click **Create New Organization** → Enter project details.

3. Choose **public/private repository** and **work tracking method** (Scrum/Kanban).

Step 2: Use Azure Repos for Source Control

1. Navigate to **Repos** → Create a Git repository.
2. Clone repository and push code:
3. git clone https://dev.azure.com/your-org/your-project/_git/repository-name
4. git add .
5. git commit -m "Initial Commit"
6. git push origin main

Step 3: Create an Agile Board in Azure DevOps

1. Go to **Azure Boards** → Click **Backlogs**.
2. Add **User Stories, Features, and Tasks**.
3. Assign tasks to team members and track progress in **Kanban view**.

📌 **Example:**
A **game development team** tracks feature requests in **Azure Boards** and integrates them into the CI/CD pipeline for faster delivery.

CHAPTER 5: AUTOMATING CI/CD WITH AZURE PIPELINES

What is CI/CD?

✔ **Continuous Integration (CI):** Automates code **builds and tests**.
✔ **Continuous Deployment (CD):** Automates **releases to production**.

Setting Up Azure Pipelines

Step 1: Create a YAML Pipeline

1. Go to **Pipelines** → Click **New Pipeline**.
2. Select repository (Azure Repos, GitHub, or Bitbucket).
3. Define build steps in azure-pipelines.yml:

```
trigger:
 - main

pool:
 vmImage: 'ubuntu-latest'

steps:
 - task: UseNode@2
   inputs:
     version: '16.x'
 - script: npm install
   displayName: 'Install dependencies'
 - script: npm test
   displayName: 'Run tests'
 - script: npm run build
   displayName: 'Build Application'
```

4. Click **Run Pipeline** and monitor logs.

📌 **Example:**
An **AI-powered chatbot service** automates CI/CD using **Azure Pipelines**, ensuring continuous model updates.

Chapter 6: Agile Work Management with Azure Boards

Key Features of Azure Boards

✓ **Backlogs & Sprints:** Organizes work in **Scrum-based sprint planning**.
✓ **Kanban Boards:** Visualizes workflow and bottlenecks.
✓ **Work Item Tracking:** Tracks **features, bugs, and user stories**.

Creating a Sprint Plan

1. Navigate to **Azure Boards** → Click **Sprints**.
2. Add work items (User Stories, Tasks, Bugs).
3. Assign developers, set priorities, and track progress.

📌 **Example:**
A **travel booking company** manages its feature roadmap using **Azure Boards Sprints**, ensuring smooth software updates.

Chapter 7: Monitoring & Security in DevOps Pipelines

1. Monitoring DevOps Pipelines

✓ **Azure Monitor:** Tracks **pipeline performance and failures**.
✓ **Application Insights:** Monitors **web application logs and telemetry**.
✓ **Log Analytics:** Captures **CI/CD logs for debugging issues**.

2. Securing Azure DevOps

✓ **Enable Role-Based Access Control (RBAC):** Restricts permissions based on roles.
✓ **Use Azure Key Vault:** Manages **secrets, credentials, and API keys**.
✓ **Enable Multi-Factor Authentication (MFA):** Adds **extra security layers**.

📌 **Example:**
A **government IT agency** secures its **Azure DevOps workflows** by **encrypting sensitive credentials with Azure Key Vault**.

CHAPTER 8: CASE STUDY – IMPLEMENTING DEVOPS AND AGILE IN A CLOUD STARTUP

Problem Statement:

A **fintech startup** wants to **speed up software releases** while ensuring **security and stability**.

Solution Implementation:

1. **Adopted Agile Scrum** using **Azure Boards**.
2. **Configured CI/CD Pipelines** to automate testing and deployment.
3. **Used Azure Monitor** for real-time logging and issue tracking.
4. **Implemented RBAC and Key Vault** to secure access.

Results:

✓ **50% faster feature releases** using Agile & DevOps integration.
✓ **Improved system reliability** with automated testing.
✓ **Enhanced security** using **Azure Key Vault for credentials**.

CHAPTER 9: EXERCISE & REVIEW QUESTIONS

Exercise:

1. **Set up an Azure DevOps Project** and initialize a Git repository.
2. **Create an Azure Board** and add user stories and tasks.
3. **Define a YAML pipeline** for a simple Node.js application.
4. **Enable Azure Monitor and Key Vault** for security and logging.

Review Questions:

1. What are the **key benefits of DevOps** in software development?
2. How does **Agile complement DevOps workflows**?
3. What are the **best practices for CI/CD pipeline security**?
4. How do **Azure Boards improve sprint planning and backlog tracking**?
5. Why is **monitoring and logging important in DevOps pipelines**?

CONCLUSION: BUILDING FASTER & SMARTER SOFTWARE WITH DEVOPS & AGILE IN AZURE

By integrating **DevOps automation** with **Agile work tracking**, organizations can **accelerate software development, improve deployment reliability, and enhance team collaboration** in Azure. 🚀

Azure DevOps – Repositories, Pipelines, Boards, and Artifacts

Chapter 1: Introduction to Azure DevOps

Understanding DevOps and Its Importance

Azure DevOps is a **cloud-based DevOps platform** that enables teams to collaborate, build, test, and deploy applications efficiently. It offers a complete set of **development, automation, and collaboration tools**, improving **software delivery cycles** and enhancing **team productivity**.

Why Use Azure DevOps?

✓ **End-to-End CI/CD Pipeline:** Automates code builds, testing, and deployments.
✓ **Centralized Source Code Management:** Uses **Azure Repos** for Git version control.
✓ **Project Management & Tracking:** Provides **Agile boards and backlog management**.
✓ **Artifact Storage & Package Management:** Manages **dependencies and artifacts**.
✓ **Integration with Azure & Third-Party Tools:** Supports **GitHub, Jenkins, Terraform, Kubernetes, and Docker**.

📌 **Example:**
A **software development team** uses **Azure DevOps** to **manage code in Git repositories, automate deployments using Azure Pipelines, track work with Boards, and store build artifacts**.

Chapter 2: Azure Repositories – Managing Source Code

What is Azure Repos?

Azure Repos is a **source code management (SCM) system** that supports both **Git** (distributed version control) and **TFVC** (centralized version control). It allows developers to store, track, and collaborate on code efficiently.

Features of Azure Repos

✓ **Git & TFVC Support:** Choose between **Git (distributed) and TFVC (centralized)** version control.
✓ **Branching & Merging:** Supports **feature branching, pull requests, and code reviews**.
✓ **Code Policies & Security:** Enforce **branch policies, approvals, and commit validations**.
✓ **Integration with Azure Pipelines:** Automate **CI/CD workflows** for software deployment.

Setting Up a Git Repository in Azure Repos

1. Navigate to **Azure DevOps Portal** → Select a **Project**.
2. Click **Repos** → Click **Initialize Repository**.
3. Clone the repo using Git:
4. git clone https://dev.azure.com/your-org/your-project/_git/repository-name
5. Add and push code to the repository:
6. git add .
7. git commit -m "Initial commit"
8. git push origin main

📌 **Example:**
A **mobile app development team** uses **Azure Repos** to manage multiple branches for new features, hotfixes, and releases.

CHAPTER 3: AZURE PIPELINES – AUTOMATING CI/CD

What is Azure Pipelines?

Azure Pipelines is a **Continuous Integration (CI) and Continuous Deployment (CD) service** that automates application build, testing, and deployment processes. It supports **containers, Kubernetes, and multi-cloud environments**.

Key Features of Azure Pipelines

✓ **Multi-Platform Builds:** Supports **Windows, Linux, and MacOS**.
✓ **CI/CD Automation:** Automates **code builds, unit testing, and deployments**.
✓ **Parallel Jobs & Agent Pools:** Speeds up builds with **self-hosted and cloud-hosted agents**.
✓ **Integration with Containers:** Supports **Docker, Kubernetes, and Helm Charts**.

Setting Up a CI/CD Pipeline in Azure DevOps

Step 1: Create a Pipeline

1. Navigate to **Azure DevOps Portal** → Select **Pipelines**.
2. Click **New Pipeline** → Choose **"GitHub" or "Azure Repos"** as source.
3. Select a build template (e.g., .NET Core, Node.js, Python).

Step 2: Define a YAML Pipeline

Create a azure-pipelines.yml file to define the pipeline workflow:

```
trigger:
  - main
```

```
pool:
  vmImage: 'ubuntu-latest'

steps:
  - task: UseNode@2
    inputs:
      version: '16.x'
  - script: npm install
    displayName: 'Install dependencies'
  - script: npm test
    displayName: 'Run tests'
  - script: npm run build
    displayName: 'Build Application'
```

Step 3: Run and Monitor the Pipeline

1. Commit the azure-pipelines.yml file to the repository.
2. Go to **Pipelines** → Click **Run Pipeline**.
3. Monitor execution logs and troubleshoot failures.

📌 **Example:**
A **banking application** automates **CI/CD with Azure Pipelines**, ensuring **code changes are tested and deployed securely to Azure Kubernetes Service (AKS)**.

Chapter 4: Azure Boards – Agile Project Management

What is Azure Boards?

Azure Boards is a **work tracking system** that helps teams plan, track, and discuss **software development work** using Agile, Scrum, or Kanban methodologies.

Features of Azure Boards

✓ **Work Item Tracking:** Manage **user stories, tasks, and bugs**.
✓ **Agile Boards & Kanban:** Visualize **backlogs and sprint progress**.
✓ **Sprint Planning & Reporting:** Plan **iterations, velocity tracking, and burndown charts**.
✓ **Integration with GitHub & Azure Pipelines:** Link **work items to commits and CI/CD builds**.

Creating a New Work Item in Azure Boards

1. Navigate to **Azure DevOps Portal** → Click **Boards**.
2. Select **Work Items** → Click **New Work Item**.
3. Enter details like **Title, Description, Priority, and Assignee**.
4. Click **Save & Close**.

📌 **Example:**
A **software engineering team** manages their **product backlog** using **Azure Boards**, tracking **feature requests and bug fixes** for an upcoming release.

Chapter 5: Azure Artifacts – Managing Packages and Dependencies

What is Azure Artifacts?

Azure Artifacts is a **package management solution** that enables teams to **store, share, and manage software dependencies** securely within DevOps projects.

Features of Azure Artifacts

✓ **Supports Multiple Package Types:** Works with **NuGet, npm, Maven, Python, and Universal Packages**.
✓ **Secure & Private Feeds:** Stores **internal and external package dependencies**.
✓ **Version Control for Packages:** Ensures **stable builds with versioned packages**.

Publishing a Package to Azure Artifacts

Step 1: Create an Artifact Feed

1. Navigate to **Azure DevOps Portal** → Click **Artifacts**.
2. Click **+ New Feed** → Name the feed (e.g., MyAppPackages).
3. Set **visibility** (private/public).

Step 2: Publish a Package Using npm

1. Authenticate to Azure Artifacts:
2. npm login --registry=https://pkgs.dev.azure.com/your-org/_packaging/MyAppPackages/npm/registry/
3. Publish package:
4. npm publish --registry=https://pkgs.dev.azure.com/your-org/_packaging/MyAppPackages/npm/registry/

📌 **Example:**
A **mobile development team** stores **shared libraries and dependencies** in **Azure Artifacts**, ensuring version consistency across microservices.

Chapter 6: Case Study – Implementing Azure DevOps in a Software Project

Problem Statement:

A **SaaS startup** needs a **fully automated DevOps workflow** to accelerate **application development and deployment**.

Solution Implementation:

1. **Azure Repos:** Git repository for source code versioning.
2. **Azure Pipelines:** CI/CD automation for **build, test, and deployment**.
3. **Azure Boards:** Agile task tracking for feature development.
4. **Azure Artifacts:** Package management for shared libraries.

Results:

✓ **40% faster development cycles** with automated CI/CD.
✓ **Improved collaboration** with integrated Agile tracking.
✓ **Reduced deployment failures** with pipeline validation.

Chapter 7: Exercise & Review Questions

Exercise:

1. **Create an Azure DevOps Project** and initialize a Git repository.
2. **Set up an Azure Pipeline** with a YAML configuration.
3. **Create a work item in Azure Boards** and link it to a commit.

4. **Publish a package to Azure Artifacts** and retrieve it from a project.

Review Questions:

1. How does **Azure Pipelines automate CI/CD** workflows?
2. What are the differences between **Azure Repos and GitHub**?
3. How can **Azure Boards improve Agile project tracking**?
4. What are the advantages of using **Azure Artifacts**?
5. What security measures should be implemented in **Azure DevOps**?

Conclusion: Streamlining Development with Azure DevOps

Azure DevOps provides a **unified, scalable** solution for **code management, automation, collaboration, and package management**. By leveraging **Azure Repos, Pipelines, Boards, and Artifacts**, development teams can **accelerate software delivery and maintain high-quality applications**. 🚀

IMPLEMENTING CI/CD PIPELINES USING AZURE DEVOPS

CHAPTER 1: INTRODUCTION TO CI/CD AND AZURE DEVOPS

Understanding CI/CD Pipelines

Continuous Integration (CI) and Continuous Deployment (CD) are software development practices that enable teams to **automate the building, testing, and deployment** of applications. Azure DevOps provides **Azure Pipelines**, a cloud-based CI/CD service, to streamline software delivery.

Why CI/CD Pipelines Matter?

✓ **Automates Builds & Deployments** – Reduces manual effort and errors.

✓ **Speeds Up Release Cycles** – Enables faster, more reliable software updates.

✓ **Ensures Code Quality** – Runs automated tests before deployment.

✓ **Enhances Collaboration** – Integrates with Git for seamless teamwork.

📌 **Example:**
A **fintech company** uses Azure DevOps CI/CD pipelines to deploy new **mobile banking features weekly**, reducing deployment time from **days to minutes**.

CHAPTER 2: UNDERSTANDING AZURE DEVOPS & AZURE PIPELINES

What is Azure DevOps?

Azure DevOps is a suite of services for **version control, CI/CD, testing, and project management**.

Key Components of Azure DevOps

✓ **Azure Repos** – Source code management with Git.
✓ **Azure Pipelines** – CI/CD automation for builds and releases.
✓ **Azure Test Plans** – Automated and manual testing tools.
✓ **Azure Artifacts** – Package management for dependencies.
✓ **Azure Boards** – Agile project tracking and work management.

📌 **Example:**
A **retail e-commerce company** uses **Azure Boards** to track development tasks and **Azure Pipelines** for **automated deployments**.

CHAPTER 3: SETTING UP A CI/CD PIPELINE IN AZURE DEVOPS

Step 1: Create an Azure DevOps Project

1. **Go to Azure DevOps Portal** – dev.azure.com
2. Click **+ Create Project** → Enter project details.
3. Choose **Git for version control**.

📌 **Example:**
A **startup** creates a DevOps project named EcommerceApp for managing its **web store development**.

Step 2: Configure a Git Repository (Azure Repos)

1. In **Azure DevOps**, go to **Repos** → Click **New Repository**.
2. Clone the repo locally using:

3. git clone https://dev.azure.com/{organization}/{project}/_git/{repository}
4. Add source code files and commit changes:
5. git add .
6. git commit -m "Initial commit"
7. git push origin main

📌 **Example:**
A **software company** stores its **microservices codebase** in an Azure Repos **Git repository**.

Step 3: Configure a Build Pipeline (CI – Continuous Integration)

1. Navigate to **Pipelines** → **Create Pipeline**.
2. Choose **GitHub, Azure Repos, or Other Repositories** as the source.
3. Select **YAML** or **Classic Editor** for pipeline configuration.
4. Use a sample azure-pipelines.yml file:
5. trigger:
6.   branches:
7.   include:
8.   - main
9. pool:
10.   vmImage: 'ubuntu-latest'
11. steps:

12. - task: UseNode@1
13. inputs:
14. version: '14.x'
15. - script: npm install
16. displayName: 'Install Dependencies'
17. - script: npm run build
18. displayName: 'Build Application'
19. - script: npm test
20. displayName: 'Run Unit Tests'
21. Click **Run Pipeline** to execute the CI process.

📌 **Example:**
A **mobile app team** sets up a **CI pipeline** to **build and test React Native code** before deployment.

Step 4: Configure a Release Pipeline (CD – Continuous Deployment)

1. Navigate to **Pipelines** → **Releases** → **New Release Pipeline**.
2. Select **Azure App Service, Kubernetes, or VM** as the deployment target.
3. Configure **Artifacts**:
   - Select **Build Pipeline Output** as the source.
   - Define **staging and production environments**.
4. Add **Deployment Tasks** (e.g., Web App Deployment, Helm Chart Deployment).

5. Click **Create Release** → Deploy to the target environment.

📌 **Example:**
A **logistics company** automates the **deployment of a Node.js API** to **Azure Kubernetes Service (AKS)**.

CHAPTER 4: ENHANCING CI/CD WITH SECURITY & TESTING

Integrate Automated Testing

✓ **Unit Tests** – Runs before code is merged.
✓ **Integration Tests** – Ensures system functionality.
✓ **Security Scans** – Detects vulnerabilities using **SonarQube or WhiteSource**.

📌 **Example:**
A **financial app** runs **security scans and penetration tests** in its CI/CD pipeline before deploying new features.

Implement Blue-Green & Canary Deployments

✓ **Blue-Green Deployment** – Keeps two versions running for rollback.
✓ **Canary Deployment** – Gradually deploys updates to a small subset of users.

📌 **Example:**
A **social media platform** deploys new features to **5% of users first** before a full rollout.

CHAPTER 5: MONITORING & TROUBLESHOOTING CI/CD PIPELINES

Enable Monitoring & Logs

✔ Use **Azure Monitor & Application Insights** to track deployments.
✔ Set up **alerts** for failed pipelines.
✔ Integrate **Azure Log Analytics** for troubleshooting.

📌 **Example:**
An **e-learning platform** uses **Azure Monitor** to detect deployment failures and rollback **unstable updates**.

Chapter 6: Best Practices for CI/CD in Azure DevOps

✔ **Use Infrastructure as Code (IaC)** – Automate deployment of environments using **Terraform or Bicep**.
✔ **Enforce Pull Requests & Code Reviews** – Require **approvals before merging code**.
✔ **Optimize Build Pipelines** – Cache dependencies to **speed up builds**.
✔ **Implement Feature Flags** – Deploy features without releasing them immediately.

📌 **Example:**
A **healthtech company** uses **feature flags** to **gradually enable new features** in its mobile app.

Chapter 7: Case Study – Automating CI/CD for an E-Commerce Platform

Problem Statement:

An **e-commerce company** faced **slow software releases and frequent deployment failures**, leading to **downtime during high-traffic events**.

Solution:

1. **Implemented Azure DevOps Pipelines**:
   - **CI Pipeline** to build and test every commit.
   - **CD Pipeline** to deploy updates to staging and production.
2. **Configured Blue-Green Deployments** for rollback capability.
3. **Integrated Automated Security Scans** before deployment.

Results:

✓ **Reduced deployment time from 3 hours to 15 minutes**.
✓ **Improved stability, eliminating deployment failures**.
✓ **Increased developer efficiency with automated testing**.

CHAPTER 8: EXERCISE & REVIEW QUESTIONS

Exercise

1. **Create an Azure DevOps Project** and set up a **Git repository**.
2. **Build a CI pipeline** to test a simple **Node.js or .NET application**.
3. **Deploy the application using a CD pipeline** to **Azure Web App or Kubernetes**.

Review Questions

1. What are the benefits of using **Azure DevOps Pipelines** for CI/CD?
2. How do **feature flags** help in CI/CD deployment?
3. What is the difference between **Blue-Green and Canary Deployment**?

4. Why should security scans be integrated into CI/CD pipelines?
5. How does **automated testing** improve software reliability?

Conclusion: Building a Reliable CI/CD Pipeline with Azure DevOps

By implementing **Azure DevOps CI/CD Pipelines**, organizations can **automate software delivery, reduce deployment risks, and enhance collaboration**. Whether deploying **web apps, APIs, or microservices**, a well-structured CI/CD pipeline ensures **efficient, secure, and scalable deployments**. 🚀

MONITORING & LOGGING WITH AZURE MONITOR & APPLICATION INSIGHTS

CHAPTER 1: INTRODUCTION TO AZURE MONITORING AND LOGGING

Understanding Monitoring and Logging in Cloud Environments

Cloud-based applications and infrastructure require **continuous monitoring and logging** to ensure high availability, security, and performance. Azure provides **Azure Monitor** and **Application Insights** to collect, analyze, and respond to telemetry data from Azure resources and applications.

Why is Monitoring and Logging Important?

✓ **Detect and Resolve Issues Faster:** Identifies system failures and performance bottlenecks.
✓ **Improve Application Performance:** Optimizes application behavior using real-time data.
✓ **Enhance Security:** Detects and logs unauthorized access attempts.
✓ **Optimize Costs:** Identifies inefficient resource usage and prevents over-provisioning.

📌 **Example:**
A **global e-commerce platform** uses **Azure Monitor and Application Insights** to track website traffic, detect failed transactions, and trigger alerts when CPU usage exceeds 80%.

CHAPTER 2: OVERVIEW OF AZURE MONITOR

What is Azure Monitor?

Azure Monitor is a **cloud-based monitoring solution** that collects and analyzes telemetry data from **Azure resources, virtual machines, databases, applications, and networks**.

Key Features of Azure Monitor

✓ **Metrics Collection:** Tracks **CPU, memory, network, and disk usage**.
✓ **Log Analytics:** Gathers logs for **troubleshooting and auditing**.
✓ **Alerts & Notifications:** Sends alerts when a threshold is breached.
✓ **Application Performance Monitoring:** Monitors app behavior in real time.
✓ **Integration with Azure Services:** Works with **VMs, SQL Databases, Kubernetes, and more**.

Components of Azure Monitor

Component	Purpose
Metrics	Real-time numeric performance indicators (CPU, Memory, Requests/sec)
Logs	Stores structured/unstructured telemetry data for analysis
Alerts	Triggers notifications when predefined conditions occur
Dashboards	Provides visual insights into resource health and performance

📌 **Example:**
A **banking system** uses **Azure Monitor Alerts** to notify administrators when **network latency increases above 200ms**, ensuring faster issue resolution.

Chapter 3: Introduction to Application Insights

What is Application Insights?

Application Insights is a **monitoring and performance management tool** designed for **web applications**. It collects telemetry data like **request rates, response times, error logs, and user interactions** to improve application reliability.

Key Features of Application Insights

✓ **Performance Monitoring:** Tracks response times and dependency failures.

✓ **User Behavior Analytics:** Analyzes user journeys and session durations.

✓ **Exception Tracking:** Detects and logs application errors and crashes.

✓ **Live Metrics Stream:** Provides **real-time visibility** into application health.

✓ **Integration with DevOps:** Works with **Azure DevOps, GitHub, and CI/CD Pipelines**.

📌 **Example:**
A **news website** integrates **Application Insights** to monitor **page load times**, ensuring optimal performance for users in different regions.

Chapter 4: Setting Up Azure Monitor for Resource Monitoring

Step 1: Enable Azure Monitor

1. Go to **Azure Portal** → Navigate to **Azure Monitor**.
2. Click **Metrics** → Select the Azure resource (VM, Storage, SQL Database).

3. Add performance counters like **CPU usage, memory consumption, and disk I/O**.

Step 2: Configure Log Analytics Workspace

1. Navigate to **Azure Monitor** → Click **Logs**.
2. Click **+ Create Log Analytics Workspace**.
3. Choose a **Subscription, Resource Group, and Region**.
4. Enable **Diagnostic Settings** to store logs.

📌 **Example:**
A **cloud-based gaming company** enables **Azure Monitor** on **Azure Kubernetes Service (AKS)** to track **pod CPU utilization and prevent service crashes**.

Chapter 5: Setting Up Application Insights for Application Monitoring

Step 1: Enable Application Insights for a Web App

1. Go to **Azure Portal** → Navigate to **Application Insights**.
2. Click **+ Create** → Select **Resource Group and Region**.
3. Enable **Application Monitoring** for a Web App.

Step 2: Instrument an Application with Application Insights SDK

For a **.NET Core Web Application**, install the SDK:

dotnet add package Microsoft.ApplicationInsights.AspNetCore

Modify Startup.cs to enable monitoring:

```
public void ConfigureServices(IServiceCollection services)

{
```

services.AddApplicationInsightsTelemetry(Configuration["ApplicationInsights:InstrumentationKey"]);

}

📌 **Example:**
A **SaaS product** adds **Application Insights to its backend APIs**, tracking **HTTP request failures and identifying performance bottlenecks**.

Chapter 6: Configuring Alerts and Dashboards

Step 1: Create an Alert Rule in Azure Monitor

1. Navigate to **Azure Monitor** → Click **Alerts**.
2. Click **New Alert Rule** → Select a **Target Resource** (e.g., Virtual Machine).
3. Choose **Condition (e.g., CPU usage > 80%)**.
4. Define **Notification (email, SMS, webhook)**.
5. Click **Create**.

📌 **Example:**
A **logistics company** configures alerts to detect **database query slowdowns**, ensuring **fast shipment processing**.

Step 2: Build a Monitoring Dashboard

1. Navigate to **Azure Monitor** → Click **Dashboards**.
2. Add widgets like **CPU Usage, Memory Consumption, Request Failures**.
3. Customize dashboard layout and save.

📌 **Example:**
A **travel booking website** builds a **real-time dashboard** showing **active users, API response times, and failed requests**.

Chapter 7: Advanced Log Analytics with Kusto Query Language (KQL)

What is KQL?

Kusto Query Language (KQL) is used in **Log Analytics** to query and analyze **Azure logs**.

Example KQL Queries

1. Retrieve Failed Requests

```
requests
| where success == false
| order by timestamp desc
```

2. Track CPU Usage Over Time

```
Perf
| where CounterName == "Processor Time"
| summarize avg(CounterValue) by bin(TimeGenerated, 10m)
```

📌 **Example:**
A **cybersecurity company** uses **KQL queries in Azure Log Analytics** to track **failed login attempts and detect security threats**.

Chapter 8: Case Study – Monitoring a Retail E-commerce Platform

Problem Statement:

An **e-commerce company** experiences **high cart abandonment rates** and **slow page load times**.

Solution Implementation:

1. **Enabled Application Insights** to track **user sessions and failed transactions**.
2. **Configured Alerts** for **500-level errors** in the web application.
3. **Created a Monitoring Dashboard** to visualize **cart abandonment trends**.
4. **Optimized slow database queries** based on **Log Analytics reports**.

Results:

✓ **Reduced response times by 40%** using real-time telemetry.
✓ **Increased checkout conversions by 25%** after fixing high-latency pages.
✓ **Improved customer experience** by resolving **failed transactions faster**.

CHAPTER 9: EXERCISE & REVIEW QUESTIONS

Exercise:

1. **Enable Azure Monitor** and track **CPU usage for a Virtual Machine**.
2. **Instrument an ASP.NET application** with **Application Insights**.
3. **Create an alert rule** for **high memory usage**.
4. **Use KQL in Log Analytics** to query application logs.

Review Questions:

1. What are the **key differences** between **Azure Monitor and Application Insights**?
2. How does **Log Analytics help troubleshoot application issues**?
3. What are the **best practices for setting up alerts in Azure Monitor**?
4. How can **Application Insights improve user experience**?
5. Why is **Kusto Query Language (KQL) useful for log analysis**?

Conclusion: Ensuring Application Health with Azure Monitor & Application Insights

Azure Monitor and Application Insights provide **real-time visibility, automated alerts, and deep log analysis** for **proactive monitoring** of cloud applications. Implementing **performance monitoring, alerting, and log analytics** ensures **high availability, security, and efficiency** in **Azure environments**. 🚀

Azure DevOps – Repositories, Pipelines, Boards, and Artifacts

Chapter 1: Introduction to Azure DevOps

Understanding DevOps and Its Importance

Azure DevOps is a **cloud-based DevOps platform** that enables teams to collaborate, build, test, and deploy applications efficiently. It offers a complete set of **development, automation, and collaboration tools**, improving **software delivery cycles** and enhancing **team productivity**.

Why Use Azure DevOps?

✓ **End-to-End CI/CD Pipeline:** Automates code builds, testing, and deployments.

✓ **Centralized Source Code Management:** Uses **Azure Repos** for Git version control.

✓ **Project Management & Tracking:** Provides **Agile boards and backlog management**.

✓ **Artifact Storage & Package Management:** Manages **dependencies and artifacts**.

✓ **Integration with Azure & Third-Party Tools:** Supports **GitHub, Jenkins, Terraform, Kubernetes, and Docker**.

📌 **Example:**
A **software development team** uses **Azure DevOps** to **manage code in Git repositories, automate deployments using Azure Pipelines, track work with Boards, and store build artifacts**.

Chapter 2: Azure Repositories – Managing Source Code

What is Azure Repos?

Azure Repos is a **source code management (SCM) system** that supports both **Git** (distributed version control) and **TFVC** (centralized version control). It allows developers to store, track, and collaborate on code efficiently.

Features of Azure Repos

✓ **Git & TFVC Support:** Choose between **Git (distributed) and TFVC (centralized)** version control.
✓ **Branching & Merging:** Supports **feature branching, pull requests, and code reviews**.
✓ **Code Policies & Security:** Enforce **branch policies, approvals, and commit validations**.
✓ **Integration with Azure Pipelines:** Automate **CI/CD workflows** for software deployment.

Setting Up a Git Repository in Azure Repos

1. Navigate to **Azure DevOps Portal** → Select a **Project**.
2. Click **Repos** → Click **Initialize Repository**.
3. Clone the repo using Git:
4. git clone https://dev.azure.com/your-org/your-project/_git/repository-name
5. Add and push code to the repository:
6. git add .
7. git commit -m "Initial commit"
8. git push origin main

📌 **Example:**
A **mobile app development team** uses **Azure Repos** to manage multiple branches for new features, hotfixes, and releases.

CHAPTER 3: AZURE PIPELINES – AUTOMATING CI/CD

What is Azure Pipelines?

Azure Pipelines is a **Continuous Integration (CI) and Continuous Deployment (CD) service** that automates application build, testing, and deployment processes. It supports **containers, Kubernetes, and multi-cloud environments**.

Key Features of Azure Pipelines

✓ **Multi-Platform Builds:** Supports **Windows, Linux, and MacOS**.
✓ **CI/CD Automation:** Automates **code builds, unit testing, and deployments**.
✓ **Parallel Jobs & Agent Pools:** Speeds up builds with **self-hosted and cloud-hosted agents**.
✓ **Integration with Containers:** Supports **Docker, Kubernetes, and Helm Charts**.

Setting Up a CI/CD Pipeline in Azure DevOps

Step 1: Create a Pipeline

1. Navigate to **Azure DevOps Portal** → Select **Pipelines**.
2. Click **New Pipeline** → Choose **"GitHub" or "Azure Repos"** as source.
3. Select a build template (e.g., .NET Core, Node.js, Python).

Step 2: Define a YAML Pipeline

Create a azure-pipelines.yml file to define the pipeline workflow:

```
trigger:
  - main
```

```
pool:
  vmImage: 'ubuntu-latest'


steps:
  - task: UseNode@2
    inputs:
      version: '16.x'
  - script: npm install
    displayName: 'Install dependencies'
  - script: npm test
    displayName: 'Run tests'
  - script: npm run build
    displayName: 'Build Application'
```

Step 3: Run and Monitor the Pipeline

1. Commit the azure-pipelines.yml file to the repository.
2. Go to **Pipelines** → Click **Run Pipeline**.
3. Monitor execution logs and troubleshoot failures.

📌 **Example:**
A **banking application** automates **CI/CD with Azure Pipelines**, ensuring **code changes are tested and deployed securely to Azure Kubernetes Service (AKS)**.

CHAPTER 4: AZURE BOARDS – AGILE PROJECT MANAGEMENT

What is Azure Boards?

Azure Boards is a **work tracking system** that helps teams plan, track, and discuss **software development work** using Agile, Scrum, or Kanban methodologies.

Features of Azure Boards

✓ **Work Item Tracking:** Manage **user stories, tasks, and bugs**.
✓ **Agile Boards & Kanban:** Visualize **backlogs and sprint progress**.
✓ **Sprint Planning & Reporting:** Plan **iterations, velocity tracking, and burndown charts**.
✓ **Integration with GitHub & Azure Pipelines:** Link **work items to commits and CI/CD builds**.

Creating a New Work Item in Azure Boards

1. Navigate to **Azure DevOps Portal** → Click **Boards**.
2. Select **Work Items** → Click **New Work Item**.
3. Enter details like **Title, Description, Priority, and Assignee**.
4. Click **Save & Close**.

📌 **Example:**
A **software engineering team** manages their **product backlog** using **Azure Boards**, tracking **feature requests and bug fixes** for an upcoming release.

CHAPTER 5: AZURE ARTIFACTS – MANAGING PACKAGES AND DEPENDENCIES

What is Azure Artifacts?

Azure Artifacts is a **package management solution** that enables teams to **store, share, and manage software dependencies** securely within DevOps projects.

Features of Azure Artifacts

✓ **Supports Multiple Package Types:** Works with **NuGet, npm, Maven, Python, and Universal Packages**.
✓ **Secure & Private Feeds:** Stores **internal and external package dependencies**.
✓ **Version Control for Packages:** Ensures **stable builds with versioned packages**.

Publishing a Package to Azure Artifacts

Step 1: Create an Artifact Feed

1. Navigate to **Azure DevOps Portal** → Click **Artifacts**.
2. Click **+ New Feed** → Name the feed (e.g., MyAppPackages).
3. Set **visibility** (private/public).

Step 2: Publish a Package Using npm

1. Authenticate to Azure Artifacts:
2. npm login --registry=https://pkgs.dev.azure.com/your-org/_packaging/MyAppPackages/npm/registry/
3. Publish package:
4. npm publish --registry=https://pkgs.dev.azure.com/your-org/_packaging/MyAppPackages/npm/registry/

📌 **Example:**
A **mobile development team** stores **shared libraries and dependencies** in **Azure Artifacts**, ensuring version consistency across microservices.

Chapter 6: Case Study – Implementing Azure DevOps in a Software Project

Problem Statement:

A **SaaS startup** needs a **fully automated DevOps workflow** to accelerate **application development and deployment**.

Solution Implementation:

1. **Azure Repos:** Git repository for source code versioning.
2. **Azure Pipelines:** CI/CD automation for **build, test, and deployment**.
3. **Azure Boards:** Agile task tracking for feature development.
4. **Azure Artifacts:** Package management for shared libraries.

Results:

✓ **40% faster development cycles** with automated CI/CD.
✓ **Improved collaboration** with integrated Agile tracking.
✓ **Reduced deployment failures** with pipeline validation.

Chapter 7: Exercise & Review Questions

Exercise:

1. **Create an Azure DevOps Project** and initialize a Git repository.
2. **Set up an Azure Pipeline** with a YAML configuration.
3. **Create a work item in Azure Boards** and link it to a commit.

4. **Publish a package to Azure Artifacts** and retrieve it from a project.

Review Questions:

1. How does **Azure Pipelines automate CI/CD** workflows?
2. What are the differences between **Azure Repos and GitHub**?
3. How can **Azure Boards improve Agile project tracking**?
4. What are the advantages of using **Azure Artifacts**?
5. What security measures should be implemented in **Azure DevOps**?

CONCLUSION: STREAMLINING DEVELOPMENT WITH AZURE DEVOPS

Azure DevOps provides a **unified, scalable** solution for **code management, automation, collaboration, and package management**. By leveraging **Azure Repos, Pipelines, Boards, and Artifacts**, development teams can **accelerate software delivery and maintain high-quality applications**. 🚀

TESTING & DEBUGGING IN CI/CD WORKFLOWS

CHAPTER 1: INTRODUCTION TO TESTING & DEBUGGING IN CI/CD

Understanding Testing & Debugging in CI/CD

Testing and debugging are **critical components** of Continuous Integration (CI) and Continuous Deployment (CD) pipelines. These processes ensure that applications are **functional, secure, and reliable** before deployment. Azure DevOps provides **automated testing** and **debugging tools** to identify and fix issues early in the development cycle.

Why Testing & Debugging Matter?

✔ **Ensures Code Quality** – Detects **bugs, security vulnerabilities, and performance issues** before deployment.
✔ **Speeds Up Development** – Automated testing reduces manual effort and feedback time.
✔ **Prevents Production Failures** – Catches errors before they impact users.
✔ **Enhances Security** – Identifies vulnerabilities with security scanning.

📌 **Example:**
A **fintech company** runs **automated unit tests and security scans** in its CI/CD pipeline to prevent **financial transaction errors** and **data breaches**.

CHAPTER 2: TYPES OF TESTING IN CI/CD PIPELINES

1. Unit Testing

✓ Tests **individual components** of the application.

✓ Ensures that **functions, classes, and methods** work correctly.

📌 **Example:**

A **Python application** runs unit tests using pytest before merging code to main.

```
def add(a, b):
    return a + b


def test_add():
    assert add(2, 3) == 5
```

---

2. Integration Testing

✓ Ensures that **different components interact correctly**.

✓ Tests **database connections, APIs, and third-party services**.

📌 **Example:**

A **web application** runs integration tests to verify if the **frontend properly communicates with backend APIs**.

```
pytest --integration
```

---

3. Functional Testing

✓ Validates **end-to-end functionality** based on business requirements.

✓ Simulates **user interactions** using tools like **Selenium** and **Playwright**.

📌 **Example:**
A **healthcare portal** runs Selenium tests to verify if **patients can book an appointment online**.

```
from selenium import webdriver


driver = webdriver.Chrome()

driver.get("https://hospitalportal.com")

assert "Book Appointment" in driver.page_source
```

---

4. Performance Testing

✓ Evaluates **application response time, scalability, and load capacity**.
✓ Uses tools like **JMeter, k6, or Locust**.

📌 **Example:**
A **video streaming platform** runs **load tests** to ensure it can handle **10,000 concurrent users**.

```
k6 run load_test.js
```

---

5. Security Testing

✓ Identifies **vulnerabilities, misconfigurations, and security flaws**.
✓ Uses tools like **OWASP ZAP, SonarQube, or Snyk**.

📌 **Example:**
A **banking application** scans for **SQL injection vulnerabilities** before deployment.

```
zap-cli quick-scan https://bankingapp.com
```

---

6. Acceptance Testing

✓ Verifies that the **application meets user expectations**.
✓ Often performed manually or using automation frameworks.

📌 **Example:**
A **retail company** runs automated acceptance tests to verify **checkout and payment flows**.

Chapter 3: Implementing Testing in Azure DevOps Pipelines

Step 1: Configure a Testing Pipeline in Azure DevOps

1. **Go to Azure DevOps Portal** → Select **Pipelines**.
2. Click **New Pipeline** → Choose **YAML Pipeline**.
3. Add automated testing steps in azure-pipelines.yml:

```
trigger:
 branches:
  include:
   - main

pool:
 vmImage: 'ubuntu-latest'


steps:
 - script: npm install
```

```
  displayName: 'Install Dependencies'


- script: npm run test
  displayName: 'Run Unit Tests'


- script: k6 run performance_test.js
  displayName: 'Run Performance Tests'


- task: SonarCloudPrepare@1
  displayName: 'Run Security Scan'
```

4. Save and run the pipeline.

📌 **Example:**
A **logistics company** configures a CI/CD pipeline to **run unit tests, performance tests, and security scans** before deployment.

CHAPTER 4: DEBUGGING ISSUES IN CI/CD PIPELINES

1. Identify Failures in CI/CD Pipelines

✔ **Check Logs in Azure Pipelines** → Identify **error messages and failed tasks**.
✔ **Reproduce the Issue Locally** → Run failing tests on a local machine.
✔ **Enable Verbose Logging** → Add --debug flags to increase logging details.

📌 **Example:**
A **software company** identifies failing **database connection tests** in its CI pipeline and fixes the **wrong environment variable settings**.

---

2. Common CI/CD Errors & Solutions

Issue	Cause	Solution
Test Failures	Code bugs, missing dependencies	Fix code, verify libraries
Build Errors	Incorrect build configuration	Update build scripts
Timeout Issues	Slow API responses, long test execution	Optimize test execution
Security Scan Warnings	Detected vulnerabilities	Fix dependencies, update code
Deployment Failures	Incorrect environment variables	Verify configuration files

📌 **Example:**
A **travel booking platform** fixes a **deployment failure** by correcting **API key misconfigurations**.

CHAPTER 5: INTEGRATING DEBUGGING TOOLS IN CI/CD

1. Azure Monitor & Application Insights

✔ Provides **real-time application monitoring**.

✔ Captures **logs, exceptions, and telemetry data**.

📌 **Example:**
A **social media app** uses **Azure Monitor** to track **failed API requests** after deployment.

---

2. Azure Log Analytics

✓ Centralizes logs for **troubleshooting failed pipelines**.
✓ Queries logs using **Kusto Query Language (KQL)**.

📌 **Example:**
A **fintech company** uses Log Analytics to **debug a failing CI/CD pipeline step**.

```
AzureDiagnostics

| where TimeGenerated > ago(1h)

| where Category == "BuildLogs"

| project Message, Level, TimeGenerated
```

Chapter 6: Best Practices for Testing & Debugging in CI/CD

✓ **Automate Testing in CI/CD Pipelines** – Run **unit, integration, and performance tests** before deployment.
✓ **Use Staging Environments** – Deploy to **staging first** before production.
✓ **Enable Rollback Mechanisms** – Use **blue-green or canary deployments**.
✓ **Monitor & Set Alerts** – Track **pipeline failures and errors** with **Azure Monitor**.

📌 **Example:**
A **streaming service** uses **Azure Log Analytics alerts** to **detect and fix deployment errors in real-time**.

Chapter 7: Case Study – Debugging CI/CD Failures for an E-Commerce App

Problem Statement:

An **e-commerce company** experienced frequent **CI/CD pipeline failures**, delaying feature releases.

Solution:

1. **Added Automated Tests** – Integrated **unit and integration tests** into the CI pipeline.
2. **Implemented Debugging Tools** – Used **Azure Monitor & Log Analytics** to track errors.
3. **Optimized Build & Test Execution** – Reduced build time by **parallelizing test runs**.

Results:

✔ **Increased deployment success rate to 98%**.
✔ **Reduced bug detection time by 40%**.
✔ **Faster issue resolution using detailed logs**.

Chapter 8: Exercise & Review Questions

Exercise

1. **Configure a CI pipeline** with automated unit tests.
2. **Add a security scan step** to detect vulnerabilities.

3. **Enable Azure Monitor** to track failed deployments.

Review Questions

1. What are the key benefits of testing in CI/CD?
2. How can security scans be integrated into CI/CD pipelines?
3. What is the difference between **unit and integration tests**?
4. How does **Azure Log Analytics help in debugging**?
5. Why should you use **staging environments before production**?

Conclusion: Ensuring Reliable Software Delivery with Testing & Debugging

By implementing **testing and debugging best practices in CI/CD**, organizations can **improve software quality, reduce deployment risks, and enhance security**. Using **Azure DevOps tools**, teams can ensure **seamless and error-free releases**. 🚀

ASSIGNMENT

BUILD AN END-TO-END CI/CD PIPELINE USING AZURE DEVOPS

SOLUTION: BUILD AN END-TO-END CI/CD PIPELINE USING AZURE DEVOPS

Step-by-Step Guide

This guide walks through setting up a **Continuous Integration (CI) and Continuous Deployment (CD) pipeline** using **Azure DevOps**. The CI/CD pipeline will automate building, testing, and deploying an application.

Step 1: Set Up an Azure DevOps Project

1.1 Create an Azure DevOps Organization

1. Go to Azure DevOps Portal.
2. Click **Create Organization** (if not already created).
3. Enter an **organization name** and select a **region**.
4. Click **Continue**.

1.2 Create a New Azure DevOps Project

1. Click **+ New Project**.
2. Enter **Project Name** (e.g., MyAppProject).
3. Choose **Visibility** (Public/Private).
4. Click **Create**.

📌 **Example:**
A **SaaS startup** creates an **Azure DevOps project** to manage its **customer dashboard application**.

Step 2: Set Up Source Control with Azure Repos

2.1 Create a Git Repository in Azure Repos

1. Navigate to **Repos** → Click **Initialize Repository**.
2. Copy the repository **clone URL**.

2.2 Clone the Repository and Push Code

Run the following commands on your local machine:

git clone https://dev.azure.com/your-org/your-project/_git/MyAppRepo

cd MyAppRepo

git add .

git commit -m "Initial Commit"

git push origin main

📌 **Example:**
A **mobile development team** stores its **React Native application code** in **Azure Repos**.

Step 3: Create a CI/CD Pipeline in Azure Pipelines

3.1 Navigate to Azure Pipelines

1. Go to **Pipelines** → Click **New Pipeline**.
2. Choose **Azure Repos Git** as the source.
3. Select the repository (MyAppRepo).

3.2 Define a YAML Pipeline for CI/CD

Create a azure-pipelines.yml file in the root directory of the repository:

```
trigger:
  branches:
    include:
      - main  # Runs pipeline on every commit to main

pool:
  vmImage: 'ubuntu-latest'

steps:
  - task: UseNode@2
    inputs:
      version: '16.x'

  - script: npm install
    displayName: 'Install Dependencies'

  - script: npm run build
    displayName: 'Build Application'

  - script: npm test
    displayName: 'Run Tests'
```

```
- task: PublishBuildArtifacts@1
  inputs:
    artifactName: 'drop'
    targetPath: '$(Build.ArtifactStagingDirectory)'
```

3.3 Save and Run the Pipeline

1. Commit the azure-pipelines.yml file to the repository.
2. Click **Run Pipeline**.
3. Monitor logs for build status.

📌 **Example:**
A **React application** triggers a **CI build**, runs tests, and packages build artifacts.

Step 4: Set Up Continuous Deployment (CD) to Azure App Service

4.1 Create an Azure App Service

1. Go to **Azure Portal** → **App Services** → Click **+ Create**.
2. Select **Subscription, Resource Group, and Region**.
3. Choose **Runtime Stack (e.g., Node.js, .NET Core, Python)**.
4. Click **Review + Create**.

4.2 Configure Release Pipeline in Azure DevOps

1. Navigate to **Releases** → Click **+ New Release Pipeline**.
2. Choose **Azure App Service Deployment** template.
3. Click **Add an Artifact** → Select **Build Artifact** (drop).

4. Click **Stage 1 (Deploy to App Service)** → Select Azure Subscription.
5. Choose **App Service Name** from the dropdown.

4.3 Deploy Code to Azure App Service

1. Click **Create Release** → Select **Latest Build**.
2. Click **Deploy** and monitor logs.

📌 **Example:**
A **food delivery app** automates **CI/CD deployment** to **Azure App Service**, ensuring fast and reliable releases.

Step 5: Add Quality Gates with Azure Test Plans

5.1 Configure Unit & Integration Tests

Modify azure-pipelines.yml to include test execution:

```
- script: npm test
  displayName: 'Run Unit Tests'
```

5.2 Enable Automated Approval Gates

1. Navigate to **Releases** → Select **Stage 1**.
2. Click **Pre-deployment Conditions** → Enable **Approval Gates**.
3. Set up rules (e.g., Deployment Approval, Delay Policies).

📌 **Example:**
A **finance company** enables **pre-deployment approvals** to ensure compliance before releasing new features.

Step 6: Implement Security Best Practices

6.1 Secure Secrets with Azure Key Vault

1. Go to **Azure Portal** → Navigate to **Key Vault**.
2. Click **+ Create Secret** → Store Database Credentials.
3. Update azure-pipelines.yml to use Key Vault:

```
- task: AzureKeyVault@2
  inputs:
    azureSubscription: 'MyAzureSubscription'
    keyVaultName: 'MyKeyVault'
    secretsFilter: '*'
```

📌 **Example:**
A **healthcare provider** stores **API keys and database credentials** securely in **Azure Key Vault**.

6.2 Enable Role-Based Access Control (RBAC)

1. Go to **Azure DevOps** → **Project Settings** → **Permissions**.
2. Assign **Developers** CI/CD permissions.
3. Restrict **deployment access** to admins.

📌 **Example:**
A **government IT agency** restricts **production deployments** to **security-cleared personnel**.

Step 7: Monitor & Optimize CI/CD Performance

7.1 Enable Azure Monitor for CI/CD Pipelines

1. Go to **Azure Portal** → Navigate to **Monitor**.
2. Click **Azure DevOps** → Enable **Pipeline Logs Tracking**.

7.2 Set Up Alerts for Deployment Failures

1. Go to **Azure DevOps** → Click **Pipelines**.
2. Click **Edit Pipeline** → Navigate to **Triggers**.
3. Enable **Email Notifications for Failures**.

📌 **Example:**
A **video streaming company** monitors **deployment success rates** and **triggers rollback if failures exceed 5%**.

Step 8: Case Study – Implementing CI/CD for an AI-Powered Web App

Problem Statement:

A **startup** developing an **AI-powered web application** wants to automate **testing, building, and deploying new features**.

Solution Implementation:

1. **Configured Azure DevOps CI/CD Pipeline** to build & test AI models.
2. **Automated Deployments to Azure App Service** with gated approvals.
3. **Implemented Key Vault for API key security**.
4. **Set up monitoring with Azure Monitor and Log Analytics**.

Results:

✓ **80% faster release cycles** due to automation.
✓ **Improved deployment success rates by 95%**.
✓ **Enhanced security by integrating Key Vault & RBAC**.

Step 9: Exercise & Review Questions

Exercise:

1. **Create an Azure DevOps Repository** and commit sample application code.
2. **Set up an Azure Pipeline** for CI with automated testing.
3. **Deploy an application to Azure App Service using a Release Pipeline**.
4. **Configure deployment approvals & alert rules for monitoring**.

Review Questions:

1. What are the key benefits of **CI/CD automation in Azure DevOps**?
2. How does **YAML-based pipeline configuration** improve automation?
3. Why should applications store **secrets in Azure Key Vault**?
4. What role does **Azure Monitor** play in CI/CD pipeline tracking?
5. How can **release gates** improve production deployment security?

Conclusion: Optimizing Software Delivery with Azure DevOps CI/CD

By implementing **CI/CD pipelines in Azure DevOps**, businesses achieve **automated, secure, and efficient software deployments**. **Automated testing, gated approvals, and security integrations** ensure **fast, stable, and compliant releases**. 🚀

MONITOR AND LOG APPLICATION PERFORMANCE USING AZURE MONITOR

SOLUTION: MONITOR AND LOG APPLICATION PERFORMANCE USING AZURE MONITOR

Step-by-Step Guide

Step 1: Introduction to Azure Monitor

Azure Monitor is a **cloud-based monitoring solution** that collects, analyzes, and responds to telemetry data from **Azure resources, applications, and services**. It provides real-time insights into application performance, security, and infrastructure health.

Why Use Azure Monitor?

✓ **End-to-End Observability**: Monitors applications, VMs, containers, and networks.
✓ **Centralized Logging & Metrics**: Collects logs and metrics for performance analysis.
✓ **Alerts & Automation**: Sends notifications and triggers actions on predefined conditions.
✓ **Integration with Power BI & Logic Apps**: Visualizes data and automates workflows.

📌 **Example:**
A **global e-commerce platform** uses **Azure Monitor** to track user traffic, detect slow API responses, and automatically scale resources during peak sales periods.

Step 2: Enable Azure Monitor for Application Performance Monitoring (APM)

2.1 Configure Application Insights for Web Apps

Application Insights is a feature of Azure Monitor designed for **monitoring web applications** and providing deep insights into **performance and usage metrics**.

Step 1: Enable Application Insights for an App Service

1. Navigate to **Azure Portal** → Go to **App Services**.
2. Select your web application → Click **Application Insights**.
3. Click **Enable** → Choose **Log Analytics Workspace** for data storage.
4. Click **Apply** → Restart the application to activate monitoring.

Step 2: Install Application Insights SDK (For Manual Integration)

For .NET applications, install the SDK:

```
dotnet add package Microsoft.ApplicationInsights.AspNetCore
```

Modify Startup.cs to enable telemetry:

```
public void ConfigureServices(IServiceCollection services)
{

services.AddApplicationInsightsTelemetry(Configuration["ApplicationInsights:InstrumentationKey"]);
}
```

For Node.js applications, install the package:

```
npm install applicationinsights
```

Initialize the telemetry client:

```
const appInsights = require("applicationinsights");
```

appInsights.setup(process.env.APPINSIGHTS_INSTRUMENTATION KEY).start();

📌 **Example:**
A **SaaS-based CRM application** integrates **Application Insights** to track **user behavior, response times, and API failures**.

2.2 Collect Metrics from Virtual Machines (VMs)

Azure Monitor can track **CPU usage, memory consumption, and disk I/O** of Azure Virtual Machines.

Step 1: Enable VM Monitoring

1. Navigate to **Azure Portal** → Select **Virtual Machines**.
2. Click **Monitoring** → Select **Insights**.
3. Enable **Diagnostics Settings** to capture logs and metrics.
4. Choose **Log Analytics Workspace** → Click **Enable**.

Step 2: Install Azure Monitor Agent (AMA) for More Insights

For Windows VMs, install the agent using PowerShell:

Invoke-WebRequest -Uri https://aka.ms/InstallAMA -OutFile InstallAMA.ps1; powershell -ExecutionPolicy Unrestricted -File InstallAMA.ps1

For Linux VMs, install the agent using CLI:

wget https://aka.ms/InstallAMA && bash InstallAMA

📌 **Example:**
A **finance company** enables **VM monitoring** to detect **CPU spikes and memory leaks** in their trading application servers.

Step 3: Configure Log Analytics for Centralized Logging

Log Analytics is an Azure Monitor feature that **aggregates logs from multiple sources** for in-depth analysis.

3.1 Create a Log Analytics Workspace

1. Navigate to **Azure Portal** → Search for **Log Analytics Workspaces**.
2. Click **+ Create** → Select **Subscription & Resource Group**.
3. Choose **Region** → Set **Pricing Tier**.
4. Click **Review + Create** → Deploy the workspace.

3.2 Enable Logging for Azure Services

1. Navigate to **Azure Monitor** → Click **Diagnostic Settings**.
2. Select a resource (e.g., **VM, Storage, Function App**).
3. Click **+ Add Diagnostic Setting** → Choose **Log Categories** (e.g., Performance Metrics, Errors).
4. Select **Log Analytics Workspace** → Click **Save**.

3.3 Query Logs Using Kusto Query Language (KQL)

Azure Monitor logs can be queried using **KQL (Kusto Query Language)** in Log Analytics.

Example: Query to Find High CPU Usage VMs

```
Perf

| where CounterName == "% Processor Time"

| where CounterValue > 80

| project TimeGenerated, Computer, CounterValue

| order by TimeGenerated desc
```

📌 **Example:**
A **cybersecurity team** uses **KQL queries** to detect **failed login attempts and brute force attacks** on Azure VMs.

Step 4: Configure Alerts and Notifications

Azure Monitor allows **custom alerts** based on **metrics, logs, and activity events**.

4.1 Set Up Metric-Based Alerts

1. Navigate to **Azure Monitor** → Select **Alerts**.
2. Click **+ Create Alert Rule** → Choose a **Target Resource** (e.g., VM, App Service).
3. Select **Condition** (e.g., CPU usage > 80%).
4. Define an **Action Group** (e.g., Send an email, trigger an Azure Function).
5. Click **Create** to activate the alert.

4.2 Set Up Log-Based Alerts

1. Navigate to **Log Analytics** → Open **Logs**.
2. Enter a **KQL Query** (e.g., Find failed API requests).
3. Click **New Alert Rule** → Define conditions and actions.
4. Click **Save** to enable log-based alerts.

📌 **Example:**
A **gaming company** sets up **alerts for high response times** on game servers, triggering **auto-scaling of resources** when needed.

Step 5: Visualize Performance with Dashboards

Azure Monitor integrates with **Power BI and Azure Dashboards** for **custom visual reports**.

5.1 Create an Azure Dashboard

1. Navigate to **Azure Monitor** → Click **Dashboards**.
2. Click **+ New Dashboard** → Select **Resources to Monitor**.
3. Add **CPU, Memory, API Response Time, and Error Rate** widgets.
4. Save the dashboard for **real-time monitoring**.

5.2 Integrate Azure Monitor with Power BI

1. Open **Power BI Desktop** → Click **Get Data**.
2. Choose **Azure Monitor Logs** → Sign in with **Azure Credentials**.
3. Load log data → Create **custom reports and graphs**.

📌 **Example:**
A **marketing analytics firm** builds a **Power BI dashboard** to track **API failures and user engagement metrics** from an Azure-hosted website.

Step 6: Case Study – Monitoring a Cloud-Based E-Commerce Platform

Problem Statement:

A **large e-commerce company** faces **slow website response times** and **frequent server crashes** during high-traffic events.

Solution Implementation:

1. **Enabled Application Insights** to track **slow API calls and database queries**.
2. **Configured Azure Monitor Alerts** for **high CPU usage and memory leaks**.
3. **Used Log Analytics** to detect **failed payment transactions**.
4. **Created an Azure Dashboard** to **monitor real-time sales and performance metrics**.

Results:

✓ **Reduced downtime by 50%** with proactive monitoring.
✓ **Improved API response time from 3 seconds to 800ms**.
✓ **Identified and fixed database bottlenecks,** enhancing user experience.

Step 7: Exercise & Review Questions

Exercise:

1. **Enable Application Insights** for a sample web application.
2. **Create a Log Analytics query** to find all failed HTTP requests.
3. **Set up an alert** for high CPU usage in an Azure Virtual Machine.
4. **Build a custom dashboard** to monitor web app performance metrics.

Review Questions:

1. What are the **key benefits of using Azure Monitor**?
2. How does **Application Insights** improve web application monitoring?

3. What is the difference between **metric-based alerts and log-based alerts**?
4. How can **Power BI** enhance Azure Monitor reporting?
5. What are the best practices for **monitoring and optimizing Azure workloads**?

CONCLUSION: ENSURING APPLICATION PERFORMANCE WITH AZURE MONITOR

Azure Monitor provides a **comprehensive solution** for **tracking, diagnosing, and optimizing application performance**. By leveraging **Application Insights, Log Analytics, Alerts, and Dashboards**, businesses can ensure **high availability, security, and scalability** of their cloud applications. 🚀