Developing Containerized Apps with Azure

Duration: 5 Days (~8 hours per day)

Audience: .NET developers and microservices practitioners with GitLab, PostgreSQL experience
Focus: Containers, Microservices, AKS, GitOps, App Insights
Tools: .NET, Docker, Azure CLI, Azure Portal, GitLab CI/CD

Day 1 – Containers, Azure Basics, Azure App Service

Azure Fundamentals:

- Azure subscriptions, Resource Groups
- Overview of Compute, Storage, IAM, Networking
- Role-based access control (RBAC), Managed Identity overview

Microservices in .NET Core:

- Clean architecture and separation of concerns
- Dependency injection, API gateway considerations

Docker Fundamentals:

- Dockerfile syntax, multi-stage builds, Docker Compose
- Container networking & environment variables

Azure Container Registry (ACR):

- Creating ACR instances
- Tagging, pushing/pulling images securely
- Using ACR with Managed Identity from Azure services

Azure App Service (for Containers):

- App Service Plans
- Deploying single-container images from ACR
- Configuring environment variables, custom domains, SSL

Labs:

- Build and containerize a .NET Core microservice
- Push container to ACR
- Deploy container to **Azure App Service using ACR**
- Enable logging and diagnostic settings on App Service

Day 2 – Azure Storage & Messaging Patterns

Azure Storage Services Overview:

- Blob (binary data), Table (NoSQL), Queue (simple messaging), File (SMB shares)
- Securing access using SAS tokens and RBAC

Queues & Messaging Patterns:

- Storage Queue: FIFO model
- Service Bus Queue: enterprise messaging features (dead-lettering, sessions)
- Service Bus Topics & Subscriptions: pub-sub patterns

Message Delivery Mechanisms:

- At least once vs exactly once delivery
- Dead-letter queues
- Message de-duplication and ordering guarantees

.NET Integration & Retry Logic:

- Using Azure SDKs in .NET Core for Storage and Service Bus
- Implementing retry policies and exponential backoff

Labs:

- Use Blob Storage to store image/doc content
- Use **Storage Queue with a producer/consumer model**
- Replace Storage Queue with **Service Bus Queue**, then **Topic & Subscriptions**
- Add retry/failure handling logic

Day 3 – Azure Functions & Logic Apps (with Containers)

Azure Functions in Containers:

- Custom handlers and isolated worker model
- Benefits of using containers over in-process models
- Function hosting plans: Consumption vs Premium (for containers)

Triggers and Bindings:

- HTTP, Queue, Blob, Timer
- Input and output bindings with declarative syntax

Durable Functions:

- Chaining, Fan-out/Fan-in, Human interaction (approval steps)

Azure Logic Apps (Standard):

- Trigger types and actions
- Calling Azure Functions or containerized APIs
- Error handling and retries

Event-driven Microservice Communication:

- Command vs Event messaging
- Orchestration vs Choreography

Labs:

- Create **Azure Function in a Docker container**, triggered by a queue
- Deploy to Azure Functions (Premium Plan via ACR)
- Build a Logic App to:
    - Monitor a Blob Storage event
    - Trigger a containerized API
- Combine Logic App + Function for chained orchestration

Day 4 – Azure Kubernetes Service (AKS), ACR, and Key Vault

Kubernetes Concepts:

- Pods, Deployments, ReplicaSets, Services (ClusterIP, LoadBalancer)
- ConfigMaps, Secrets, Autoscaling (HPA)

Azure Kubernetes Service (AKS):

- Creating and managing AKS clusters
- Authentication with Azure AD and RBAC
- Enabling Monitoring and Diagnostics

Container CI/CD with GitLab:

- GitLab runners, .gitlab-ci.yml setup
- CI to build/push to ACR, CD to deploy on AKS

Helm Charts:

- Writing Helm templates for microservices
- Using Helm values to override configs
- Deploying Helm charts to AKS

Azure Key Vault:

- Storing and accessing secrets

- Integration with AKS using CSI driver and pod identity

Labs:

- Deploy a multi-container .NET solution to AKS using **Helm**
- Use **ACR as image source** for AKS
- Integrate **Azure Key Vault** for secrets (connection strings, keys)
- Configure ingress routing with TLS (e.g., NGINX + cert-manager)

Day 5 – Azure App Insights, GitOps, and Best Practices

Azure Monitor & Application Insights:

- Metrics, Logs, Requests, Dependencies
- Distributed tracing and transaction correlation
- Dashboards, Alerts, Availability testing

Kusto Query Language (KQL):

- Basic queries: requests, dependencies, exceptions
- Joining across telemetry tables for end-to-end views

GitOps with GitLab:

- Git as source of truth
- Automating infra and app delivery with GitLab → AKS

Microservices Best Practices:

- Observability (tracing, logging, metrics)
- Fault tolerance (circuit breakers, retries)
- Authentication (JWT, Azure AD, Managed Identity)
- Inter-service communication (REST, gRPC, message queues)
- API Gateway pattern (overview)

Labs:

1. Enable App Insights for App Service, Azure Functions, AKS
2. Trace an HTTP request across microservices in App Insights
3. Use KQL to identify slow dependencies and failures
4. Set up GitLab GitOps workflow:
    - Auto-deploy Helm chart when code is pushed
    - Rollback deployment on failure