**Cloud-Native Cost-Optimized Architecture for .NET Core MVC & Web API on Azure**

**Objective:**

Designing a **cost-effective**, **scalable**, and **secure cloud-native** architecture for a **.NET Core MVC & Web API** application using **Azure**, ensuring multi-region **Active-Passive** failover, **no unnecessary dependencies**, and **optimized cost structure**.

**1. High-Level Architecture Overview**

🔹 **Compute:** **Azure App Services (MVC & Web API)** deployed in a VNET with **Active-Passive** strategy.  
🔹 **Database:** **Azure SQL Database** in a **private subnet**, no public access, secured with **Private Endpoint**.  
🔹 **Identity & Security:** **Duende Identity Server** for **custom authentication**, avoiding Azure AD B2C dependency.  
🔹 **Networking:** **VNET with subnets for each layer (MVC, Web API, DB)**, **Private Link for secure DB access**.  
🔹 **Deployment Strategy:** **Azure DevOps with CI/CD**, **Infrastructure as Code (Bicep/Terraform)**.  
🔹 **Cost Optimization Strategy:**

* **Active-Passive** multi-region to reduce compute cost.
* **Ocelot API Gateway** instead of Azure APIM (since no third-party API exposure).
* **Azure Hybrid Benefit & Reserved Instances** for SQL cost savings.
* **Serverless components (Azure Functions) for background processing** instead of VMs.

**2. Step-by-Step Implementation**

**Step 1: Azure Networking & Security Setup**

✅ **Create Virtual Network (VNET) & Subnets**

* **VNET-01** (E-commerce Network)
  + **Subnet-01:** For **Azure App Service (MVC - UI Layer)**
  + **Subnet-02:** For **Azure App Service (Web API - Backend Layer)**
  + **Private Subnet-03:** For **Azure SQL Database** (Networking → **Public Access Disabled**)

✅ **Private Connectivity Setup:**

* **Azure Private Endpoint for SQL** → Prevents exposure to the internet.
* **Azure Key Vault with Managed Identity** → Secure storage of credentials & connection strings.
* **Application Gateway (WAF Enabled)** → Protects **MVC & API layer** against attacks.

✅ **Security Hardening:**

* **Duende Identity Server (Custom Authentication, OAuth2.0, JWT, RBAC)** → No dependency on Azure AD B2C.
* **Managed Identity for API-to-Database Communication** → No credentials stored in code.
* **Azure Policy & Defender for Cloud** → Compliance & security monitoring.

**Step 2: Application Architecture & Communication Strategy**

✅ **.NET Core MVC (Front-End, Deployed on Azure App Services)**

* Communicates **securely with Web API** using **OAuth2 & JWT Token**.
* Caching applied using **Azure Cache for Redis** to optimize performance.

✅ **.NET Core Web API (Backend, Deployed on Azure App Services)**

* Exposes RESTful APIs to the **MVC UI**.
* Uses **Ocelot API Gateway** (instead of Azure APIM, since no third-party APIs).
* Asynchronous messaging using **Azure Service Bus** for event-driven communication.

✅ **Database Layer: Azure SQL Database (Private Subnet, No Public Access)**

* **Geo-Replication Enabled (Active-Passive)** to ensure disaster recovery.
* **Elastic Pool** for optimized cost usage across databases.

✅ **Communication Flow:**  
1️⃣ MVC App requests authentication via **Duende Identity Server**.  
2️⃣ Token is sent to **Web API via Ocelot Gateway**.  
3️⃣ Web API communicates with **Azure SQL (via Managed Identity, Private Endpoint)**.  
4️⃣ Event-driven processes use **Azure Service Bus for background jobs**.

**Step 3: Development Strategy (Best Practices for .NET Core Cloud-Native)**

✅ **MVC & API Development Best Practices**

* **ASP.NET Core MVC with Razor Views, Bootstrap for UI optimization**.
* **ASP.NET Core Web API with Repository Pattern & Dependency Injection**.
* **JWT Authentication & Authorization (OAuth2, Role-Based Access Control)**.

✅ **Code Architecture (Project Structure)**

pgsql

Copy code

/EcommerceApp

├── Ecommerce.MVC (Frontend)

├── Ecommerce.API (Backend)

├── Ecommerce.Identity (Duende Identity Server)

├── Ecommerce.Common (Shared Libraries)

├── Ecommerce.Infrastructure (Database & External Integrations)

├── Ecommerce.Tests (Unit & Integration Tests)

* **Modular Monolith Strategy Initially** → Transition to **Microservices in Future**.
* **Event-Driven Processing** for transactions instead of direct API calls.

✅ **Data Strategy (Azure SQL Optimization)**

* **Sharding for High-Volume Transactions**.
* **Replication for Read Scaling (Geo-Replication, Read-Replicas in Active-Passive Setup)**.

**Step 4: CI/CD & Deployment Strategy**

✅ **Infrastructure as Code (IaC) Setup**

* **Terraform/Bicep** to provision **VNET, Subnets, Private Endpoints, App Services, SQL DB**.

✅ **CI/CD Pipeline (Azure DevOps)**  
1️⃣ **CI Pipeline:**

* Code pushed to GitHub → Triggers Azure DevOps CI Pipeline.
* Runs **unit tests, SonarQube code analysis, OWASP security scans**.  
  2️⃣ **CD Pipeline:**
* **Deploys to Dev, QA, UAT, Prod (Blue-Green Deployment)**.
* Requires approval from **Dev Lead → Stage Lead → Prod Lead**.
* Infrastructure is provisioned **dynamically via Terraform**.

**Step 5: Monitoring & Observability Strategy**

✅ **Logging & Monitoring Stack:**

* **Azure Log Analytics & Kusto Queries (KQL) for API Logs**.
* **Azure Application Insights for Web/API Performance Tracking**.
* **Azure Monitor for Infrastructure Health**.

✅ **Health Check & Self-Healing:**

* **API Health Check Middleware** for automatic instance recovery.
* **Auto-Scaling Policies for App Services based on CPU & Memory Usage**.

**Step 6: Cost Optimization Strategy**

✅ **Key Cost-Saving Measures**

| **Component** | **Cost Optimization Strategy** |
| --- | --- |
| **Compute (App Services)** | **Use App Service Plan with Auto-Scaling** (PremiumV3 for Prod, Basic for Passive Region). |
| **Database (Azure SQL)** | **Elastic Pool for cost-efficient SQL scaling**. |
| **API Management** | **Use Ocelot API Gateway** instead of Azure APIM. |
| **Authentication** | **Use Duende Identity Server** instead of Azure AD B2C to avoid licensing costs. |
| **Storage & Backup** | **Use Azure Blob Storage with Lifecycle Management** to automatically archive logs. |

**Final Architecture Summary**

✅ **Multi-Region Active-Passive** → Cost savings while ensuring disaster recovery.  
✅ **Ocelot API Gateway instead of APIM** → No extra cost for managing APIs.  
✅ **Duende Identity Server for authentication** → Avoids Azure AD B2C costs.  
✅ **Private Endpoints & VNET Integration** → Ensures security without public exposure.  
✅ **Azure DevOps CI/CD with Blue-Green Deployment** → Ensures smooth deployments.  
✅ **Monitoring with Azure App Insights & Log Analytics** → Ensures proactive monitoring.

**Conclusion & Business Justification**

This cloud-native architecture **ensures high performance, scalability, and cost-efficiency** while **eliminating unnecessary dependencies**.

* 🚀 **Cost Savings:** Up to **30% lower cost** than a standard Active-Active setup.
* 🛡 **Security-First:** **No public database access, Private Endpoints, Managed Identity**.
* ⚡ **Future-Proof:** Can be **extended into microservices without major refactoring**.

**Next Steps**

* **PoC Deployment in Azure Sandbox Environment**.
* **Client Validation & Performance Testing**.
* **Full-Scale Rollout & Production Deployment**.

This **low-level design ensures an optimized, cost-effective, and scalable Azure architecture** for your **.NET Core MVC & Web API e-commerce application**. 🚀 Let me know if you need refinements!

