Azure Cloud Computing: A Practical Guide

Table of Contents

Introduction to Azure Services

What is Microsoft Azure?

Microsoft Azure is a comprehensive cloud computing platform that offers over 200+ services designed to help organizations meet their business challenges. Azure provides the freedom to build, manage, and deploy applications on a massive, global network using your favorite tools and frameworks.

Key Benefits:

- **Flexibility**: Pay-as-you-go pricing & scalable resources
- **Integration**: Seamless integration with Microsoft tools
- **Global Reach**: Data centers worldwide
- **Security**: Enterprise-grade security & compliance

Core Azure Services

Azure's services can be categorized into several main pillars:

1. **Compute Services**
    - Virtual Machines: Highly configurable, on-demand scalable computing resources for running applications and workloads
    - App Services: Fully managed platform for building, deploying, and scaling web apps and APIs
    - Azure Functions: Serverless compute service that runs code in response to events without managing infrastructure
    - Container Instances: Fastest and simplest way to run containers in Azure without managing servers
2. **Storage Services**
    - Blob Storage: Massively scalable object storage for unstructured data like images, videos, and documents
    - File Storage: Fully managed file shares in the cloud accessible via industry-standard SMB (Server Message Block) protocol

- Queue Storage: Service for storing large numbers of messages for asynchronous processing between application components
- Table Storage: NoSQL key-value store for schema-less storage of structured data

3. **Database Services**

- Azure SQL Database: Fully managed relational database service with built-in intelligence and security
- Cosmos DB: Globally distributed, multi-model database service for any scale
- MySQL: Fully managed, scalable MySQL database service with high availability
- PostgreSQL: Enterprise-ready fully managed PostgreSQL database service with built-in security

4. **Networking**

- Virtual Networks: Isolated and highly secure network environment for your Azure resources
- Load Balancers: Distribute network traffic across multiple servers to ensure high availability
- Application Gateway: Web traffic load balancer with SSL termination and WAF capabilities
- CDN: Global content delivery network that delivers high-bandwidth content to users worldwide

Understanding Azure Resource Manager (ARM)

Azure Resource Manager (ARM) is the deployment and management service for Azure. It provides a management layer that enables you to create, update, and delete resources in your Azure account. When you send a request through any Azure tools, APIs, or SDKs, ARM receives the request, authenticates and authorizes it, and then makes the necessary changes to your Azure resources.

Key Concepts:

1. **Resources**

- Individual services you create (e.g., VMs, databases, web apps)
- Basic building blocks of your Azure solution
- Example: An Azure App Service instance

2. **Resource Groups**

- Logical containers for resources
- Resources can only exist in one resource group
- Resources in a group should share the same lifecycle
- Example structure:

```
MyProject-Production-RG/
├── product-service-app (App Service)
├── product-database (Azure SQL)
└── monitoring (Application Insights)
```

3. **Resource Providers**

- Services that supply Azure resources
- Common providers:

- Microsoft.Web (for App Services)
- Microsoft.SQL (for SQL databases)
- Microsoft.Storage (for storage accounts)

Benefits of ARM:

1. **Consistent Management**
   - Whether using PowerShell, Azure CLI, Portal, or REST APIs
   - Same results regardless of tool choice
2. **Security**
   - Built-in role-based access control (RBAC)
   - Audit trail for all changes
   - Resource locks to prevent accidental deletion
3. **Dependency Management**
   - Automatically handles resource dependencies
   - Parallel deployment of independent resources
   - Clear visualization of your architecture

🔨 Mini Activity: Resource Organization

Objective: Practice organizing Azure resources effectively

Steps:

1. Plan your resource hierarchy:

```
Subscription
└── Resource Group (e.g., "product-service-rg")
    ├── App Service Plan
    ├── App Service
    ├── Application Insights
    └── SQL Database
```

2. Apply naming conventions:

```
company-project-environment-resource-instance
Example: jecrc-batch1-products-dev-webapp-001
```

3. Apply tags for better organization:

```
az tag create --name Environment --value Production
az tag create --name CostCenter --value IT
az tag create --name Owner --value "Batch 1"
```

Azure Portal Navigation

🔨 Activity 1: Exploring the Azure Portal

Objective: Familiarize yourself with the Azure Portal interface

Steps:

1. Navigate to Azure Portal
2. Log in with your Azure account
3. Explore the following elements:
    - Home dashboard
    - Resource groups
    - Navigation menu
    - Search bar
    - Cloud shell

Tasks:

- Create a new dashboard
- Pin frequently used services
- Customize the portal theme
- Practice using the search functionality

Deploying Applications with Azure App Services

Understanding App Services

Azure App Service is a fully managed platform for building, deploying, and scaling web apps. It supports multiple programming languages and frameworks including:

- .NET
- Node.js
- Python
- Java
- PHP

Key Features:

- Built-in auto-scaling
- High availability
- Security and compliance
- DevOps integration
- Custom domains & SSL

Deployment Methods

There are several ways to deploy applications to Azure App Services:

1. **Visual Studio Deployment**

- Direct integration with Visual Studio
- One-click publish

2. **Git-based Deployment**
   - Azure DevOps
   - GitHub Actions
   - Local Git
3. **Container Deployment**
   - Docker containers
   - Container registries
4. **FTP/S**
   - Traditional file transfer
   - Legacy support

Practical Implementation

🔨 Activity 2: Deploy an ASP.NET Core Web API to Azure

Objective: Deploy an existing ASP.NET Core Web API (ProductService) to Azure App Service

Prerequisites:

- Azure account with active subscription
- Visual Studio 2022 installed
- Existing ProductService ASP.NET Core Web API project
- .NET 7.0 SDK or later installed

Steps:

1. **Prepare Your Application**
   - Open your ProductService solution in Visual Studio 2022
   - Ensure all NuGet packages are restored
   - Run the application locally to verify it works
   - Make sure your `appsettings.json` and `Program.cs` are properly configured
2. **Create Azure Resources**

```
# Login to Azure (if not already logged in)
az login

# Create a resource group
az group create --name product-service-rg --location eastus

# Create an App Service plan
az appservice plan create --name product-service-plan `
                          --resource-group product-service-rg `
```

```
                        --sku B1 `
                        --is-linux false

# Create a web app
az webapp create --name product-service-app `
                 --resource-group product-service-rg `
                 --plan product-service-plan `
                 --runtime "dotnet:7"
```

3. **Deploy Using Visual Studio**
    - Right-click on your project in Solution Explorer
    - Select "Publish"
    - Choose "Azure" as your publish target
    - Select "Azure App Service (Windows)"
    - Choose your subscription
    - Select the `product-service-app` you created
    - Click "Finish"
    - Click "Publish" to deploy your application
4. **Configure Application Settings**
    - In Azure Portal, navigate to your App Service
    - Go to "Configuration" under "Settings"
    - Add any necessary application settings:

      ```
      ASPNETCORE_ENVIRONMENT=Production
      WEBSITE_TIME_ZONE=UTC
      ```

    - Save the changes
5. **Verify Deployment**
    - Visit your API at: `https://product-service-app.azurewebsites.net/swagger`
    - Test a few endpoints using Swagger UI
    - Check Application Insights (if configured) for telemetry
    - Review deployment logs in Azure Portal

Challenge Tasks:

1. **Add Application Insights**

```
# Create Application Insights resource
az monitor app-insights component create `
    --app product-service-insights `
    --location eastus `
    --resource-group product-service-rg `
    --application-type web
```

- Add Application Insights connection string to your app configuration
- Monitor API performance in real-time

2. **Configure Custom Domain**

   - Add a custom domain to your App Service
   - Configure SSL certificate
   - Test the API using the custom domain

3. **Set Up Staging Slots**

   - Create a staging deployment slot
   - Deploy updates to staging first
   - Test swap functionality

Troubleshooting Tips:

- If deployment fails, check:
  - Build configuration (Debug vs Release)
  - Framework version compatibility
  - Connection strings and app settings
  - Deployment logs in Azure Portal
- For runtime issues:
  - Enable diagnostic logging
  - Stream logs from Azure Portal
  - Check Application Insights

🔨 Activity 3: Scale Your Application

Objective: Learn how to scale an Azure App Service

Steps:

1. **Manual Scaling**

   - Navigate to your App Service in Azure Portal
   - Go to "Scale up (App Service plan)"
   - Explore different pricing tiers
   - Understand the capabilities of each tier

2. **Auto Scaling**

   - Go to "Scale out (App Service plan)"
   - Enable autoscaling
   - Configure rules based on:
     - CPU percentage
     - Memory usage
     - HTTP queue length

3. **Monitor Scaling**

    - Use Azure Monitor
    - Set up alerts
    - Review scaling history

Best Practices and Tips

1. **Security**
    - Always use HTTPS
    - Implement authentication
    - Keep secrets in Key Vault
2. **Performance**
    - Enable caching
    - Use CDN for static content
    - Monitor performance metrics
3. **Cost Management**
    - Use dev/test pricing tiers for non-production
    - Set up budget alerts
    - Clean up unused resources

Additional Resources

- Azure Documentation
- Azure Learning Paths
- Azure Code Samples
- Azure Pricing Calculator

Next Steps

After completing these activities, you should:

1. Explore more complex deployment scenarios
2. Learn about Azure DevOps integration
3. Practice with different programming languages
4. Experiment with other Azure services

Remember to always clean up resources after completing exercises to avoid unnecessary charges on your Azure subscription.

This guide is designed for learning purposes. For production deployments, always refer to the official Azure documentation and follow your organization's best practices.