Three-Tier Architecture Deployment on Microsoft Azure

Table of content

Project Overview

This documentation details the deployment of a **three-tier cloud architecture** on **Microsoft Azure**, designed to deliver a **scalable, highly available, and secure** foundation for modern web applications. The architecture logically segregates application components into distinct layers:

- **Presentation Tier (Frontend):** Serves as the user interface, handling client interactions.
- **Application Tier (Backend):** Processes business logic and communicates between the frontend and data layer.
- **Data Tier (Storage):** Manages static assets and product information without requiring transactional database capabilities.

Business Scenario: E-Commerce Product Showcase Web Application

Organization: *Contoso Boutique*

Use Case:

Contoso Boutique, a small online retail business, requires a **lightweight, cost-effective web application** to display its product catalog to customers. The solution must:

- **Showcase product images and descriptions** without the need for user authentication or payment processing.
- **Leverage Azure's cloud infrastructure** for reliability and scalability.
- **Minimize complexity** by avoiding unnecessary database overhead, as the application does not handle transactions or customer accounts.

This deployment focuses on **efficient resource utilization** while ensuring a seamless customer experience.

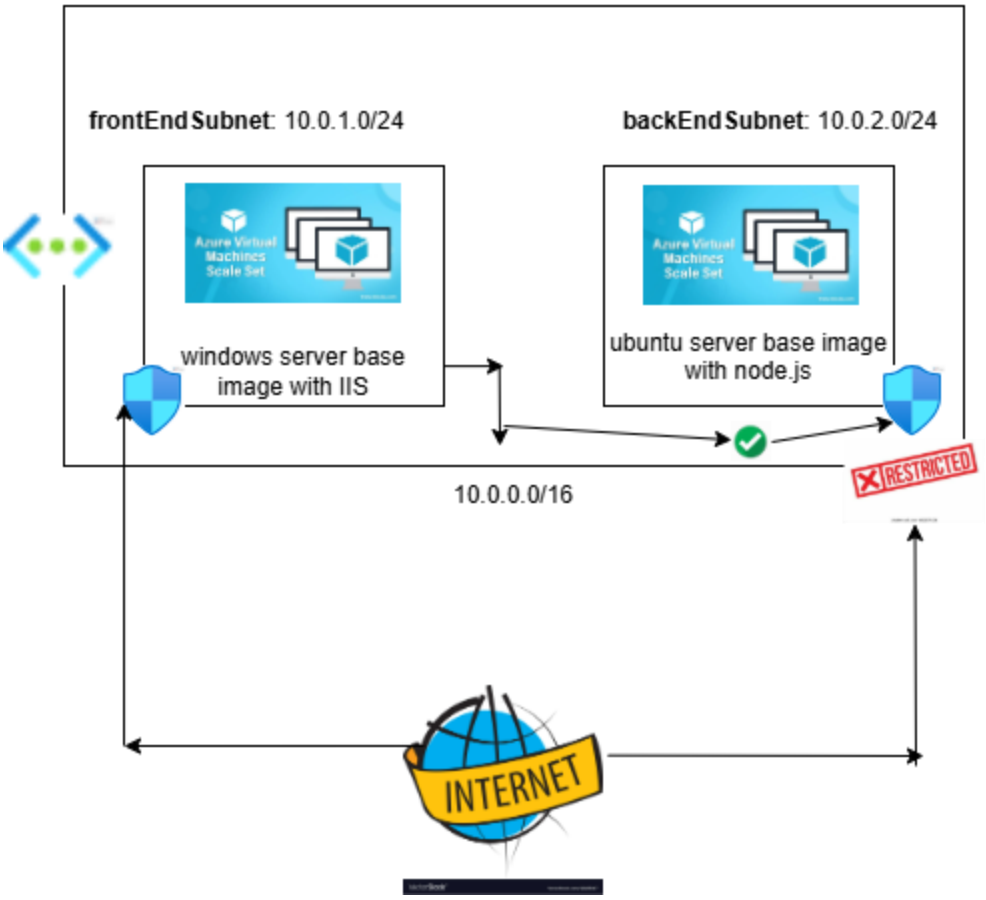
Virtual Network (VNet) Architecture

To enforce **secure and controlled communication** between application tiers, a single **Azure Virtual Network (VNet)** is provisioned, implementing logical isolation and granular traffic management.

Subnet Design & Security Policies

1. `frontEndSubnet`
    a. **Purpose:** Hosts the **frontend Virtual Machine Scale Set (VMSS)**, responsible for serving the web interface.
    b. **Network Security Group (NSG) Rules:**
        i. **Allow inbound HTTP (Port 80)** and **HTTPS (Port 443)** traffic from the public internet.
        ii. **Deny all other unnecessary traffic** to reduce exposure.
2. `backEndSubnet`
    a. **Purpose:** Hosts the **backend Virtual Machine Scale Set (VMSS)**, handling application logic and API interactions.
    b. **Network Security Group (NSG) Rules:**
        i. **Restrict inbound traffic** to **only Port 3000**, permitting communication **exclusively from `frontEndSubnet`.**
        ii. **Block direct public access** to enhance security posture.

This segmentation ensures **defense-in-depth**, minimizing attack surfaces while maintaining efficient inter-tier communication.



Tier 1: Presentation Layer (Frontend)

Purpose

The **Presentation Layer** serves as the **user-facing entry point** of the application, delivering a responsive and secure web interface. This layer is hosted on a **Windows Server-based Virtual Machine Scale Set (VMSS)** running **Internet Information Services (IIS)**, ensuring high availability and dynamic scalability to accommodate fluctuating user traffic.

⚙ Core Components

📄 *Virtual Machine Scale Set (VMSS) – Windows Server with IIS*

- **High Availability & Fault Tolerance**
  - Deployed across **three Availability Zones** for **zone-level redundancy**.
  - Configured with an **Availability Set** for additional resilience:
    - **One Fault Domains (FD):** `resiliency is already ensured using availability zones; hence I used one fault domain.`
    - **Update Domains (UD):** 5 (ensures seamless updates without downtime).
- **Elastic Scalability**
  - **Auto-scaling Policy:** Dynamically adjusts VM instances based on **CPU utilization thresholds**.
  - **Manual Scaling Option:** Allows administrators to manually adjust capacity during anticipated traffic spikes.

📄 *Frontend Application (Sample `index.html`)*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Product Gallery</title>
  <style>
    body { font-family: Arial; padding: 20px; }
    .gallery { display: flex; flex-wrap: wrap; gap: 20px; }
    .item { text-align: center; border: 1px solid #ddd; padding: 10px; border-radius: 8px; }
    .item img { max-width: 200px; height: auto; border-radius: 4px; }
  </style>
</head>
```

```
<body>
  <h1>Product Gallery</h1>
  <div class="gallery" id="gallery">Loading...</div>

  <script>
    // internal load balancer IP address
    fetch('http://10.0.2.4:3000/api/images')
      .then(response => response.json())
      .then(images => {
        const gallery = document.getElementById('gallery');
        gallery.innerHTML = '';
        images.forEach(img => {
          const item = document.createElement('div');
          item.className = 'item';
          item.innerHTML = `<h3>${img.name}</h3><img src="${img.url}"
alt="${img.name}">`;
          gallery.appendChild(item);
        });
      })
      .catch(err => {
        document.getElementById('gallery').innerHTML = 'Failed to load
images.';
        console.error(err);
      });
  </script>
</body>
</html>
```

- **Backend Integration:** The frontend retrieves product images from the **Application Tier (Backend)** via an **internal load balancer** on **Port 3000**.

🌐 Public Load Balancer (Internet-Facing Traffic Management)

- **Frontend Configuration:**
  - **Public IP Address:** Assigned for global accessibility.
  - **Listener Rules:** Accepts **HTTP (Port 80)** traffic from the internet.
- **Backend Pool:**
  - Linked to the **frontend VMSS instances** for traffic distribution.
- **Health Monitoring:**
  - **Health Probe:** Regularly checks VMSS instances to ensure only healthy nodes receive traffic.
- **Administrative Access (NAT Rules):**
  - Enables **secure, port-specific remote management** (e.g., RDP) for administrative tasks.

🔒 Security Measures

- **Network Security Group (NSG) Policies:**
  - **Subnet-Level Restrictions:** Only **HTTP (80)** and **HTTPS (443)** inbound traffic permitted.
  - **Deny All Other Traffic:** Reduces attack surface by blocking unnecessary ports.
- **Load Balancer Security:**
  - **Strict Traffic Routing:** Ensures only web traffic reaches the frontend tier.
  - **No Direct Backend Exposure:** The backend tier remains inaccessible from the public internet.

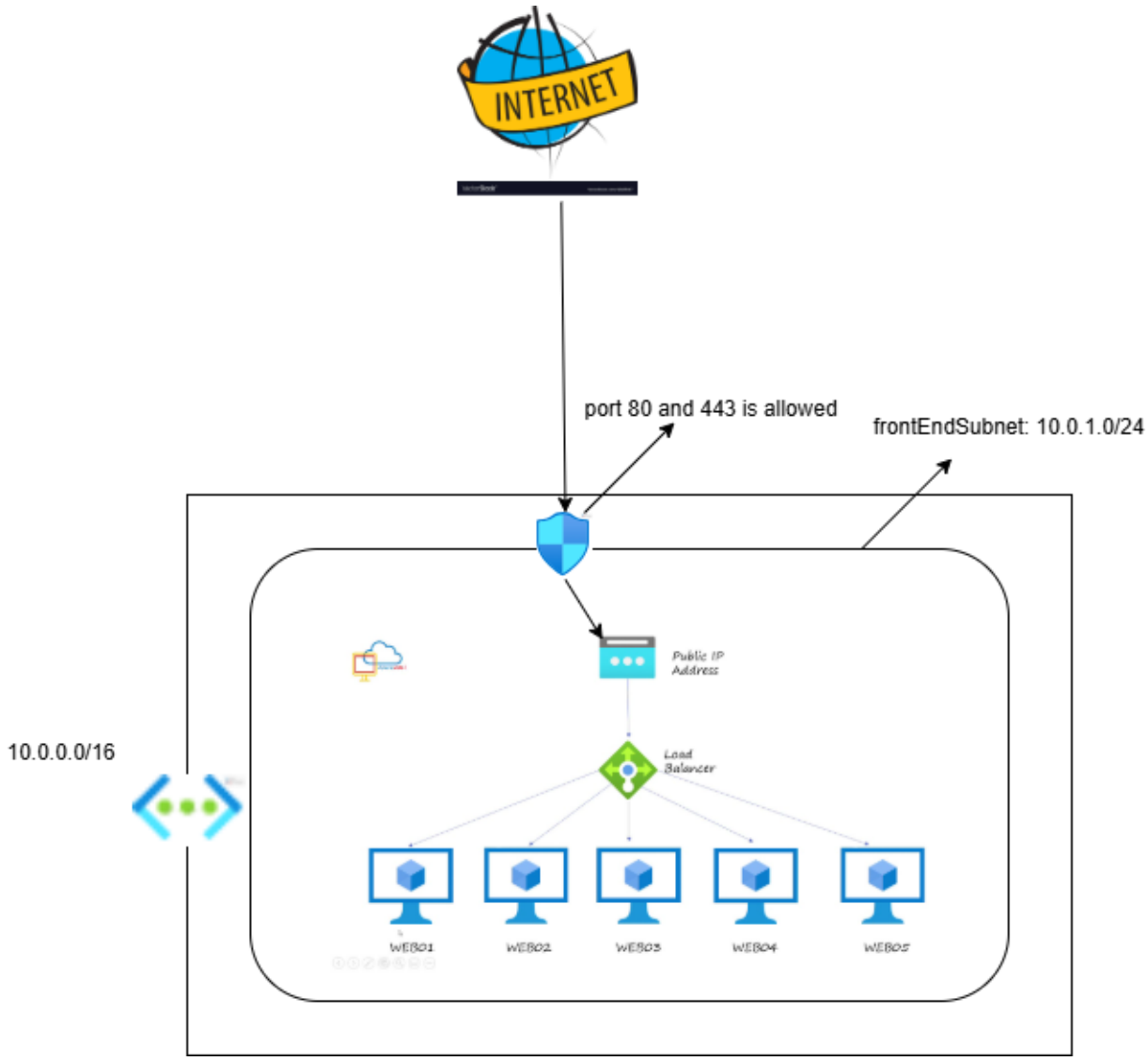
📌 Key Takeaways

✓ **Highly Available:** Multi-zone deployment minimizes downtime risks.

✓ **Elastically Scalable:** Automatically adapts to user demand.

✓ **Secure by Design:** Strict traffic filtering and no public backend exposure.

This architecture ensures **optimal performance, reliability, and security** for Contoso Boutique's e-commerce showcase.



⚙️ Tier 2: Application Layer (Backend)

🎯 Purpose

The **Application Layer** serves as the **business logic hub**, processing requests from the frontend and managing interactions with the data storage tier. Designed with **security-first principles**, this layer is **only accessible internally**—restricting exposure to the frontend tier to minimize attack surfaces while ensuring seamless functionality.

🧩 Core Components

📘 *Virtual Machine Scale Set (VMSS) – Ubuntu Server with Node.js*

- **Role:**
  - Processes **HTTP requests** from the **frontend tier** (e.g., image retrieval).
  - Hosts a **Node.js** application to handle API interactions.
- **High Availability & Fault Tolerance:**
  - Deployed across **three Availability Zones** for **zone-level redundancy**.
  - Configured with an **Availability Set** for update management:
    - **One Fault Domains (FD)**
    - **Five Update Domains (UD):** prevents simultaneous updates for zero downtime.
- **Elastic Scalability:**
  - **Auto-scaling Policy:** Adjusts VM instances dynamically based on **CPU utilization**.
  - **Manual Scaling Option:** Allows on-demand capacity adjustments.

🥏 *Backend Application (Sample `app.js`)*

```
const express = require('express');
const cors = require('cors');
const app = express();
const port = 3000;
```

```
// Allow requests from any origin
app.use(cors());


app.get('/api/images', (req, res) => {
  const images = [
    {
      name: "Price: $3000",
      url:
"https://storage665506.blob.core.windows.net/image/trouser.png"
    },
    {
      name: "Price: $40",
      url:
"https://storage665506.blob.core.windows.net/image/jacket.png"
    }
  ];
  res.json(images);
});


app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

- **Key Functionality:**
  - Exposes an `/image` endpoint to **fetch product images** stored in **Azure Blob Storage (storage665506)-- in this case**.
  - Communicates exclusively with the **frontend tier via Port 3000**.

🔐 Internal Load Balancer (Secure Tier-to-Tier Traffic)

- **Frontend IP Configuration:**
  - **Private IP Address:** Only accessible within the **Virtual Network (VNet)**.
- **Backend Pool:**
  - Linked to the **backend VMSS instances** for load distribution.
- **Health Monitoring:**
  - **Health Probe:** Validates backend instance responsiveness.
- **Traffic Rules:**
  - **Load Balancer Rule:** Permits **Port 3000** traffic **exclusively from the `frontEndSubnet`**.
  - **NAT Rule:** Enables internal request forwarding.

🔗 Storage Integration (Azure Blob Storage)

- **Storage Type: Azure Blob Storage** (for static product images).
- **Access Method:**
  - **Private Endpoint:** Ensures **no public internet exposure**—data remains within Azure's private network.

🔒 Security, Availability & Scalability

Security Measures

- **Network Security Group (NSG):**
  - **Subnet-Level Restrictions:** Only allows **Port 3000 inbound** from the `frontEndSubnet`.
  - **Denies All Other Traffic:** Eliminates unnecessary exposure.
- **No Public IPs:** The backend tier is **completely isolated** from the internet.

📌 Key Takeaways

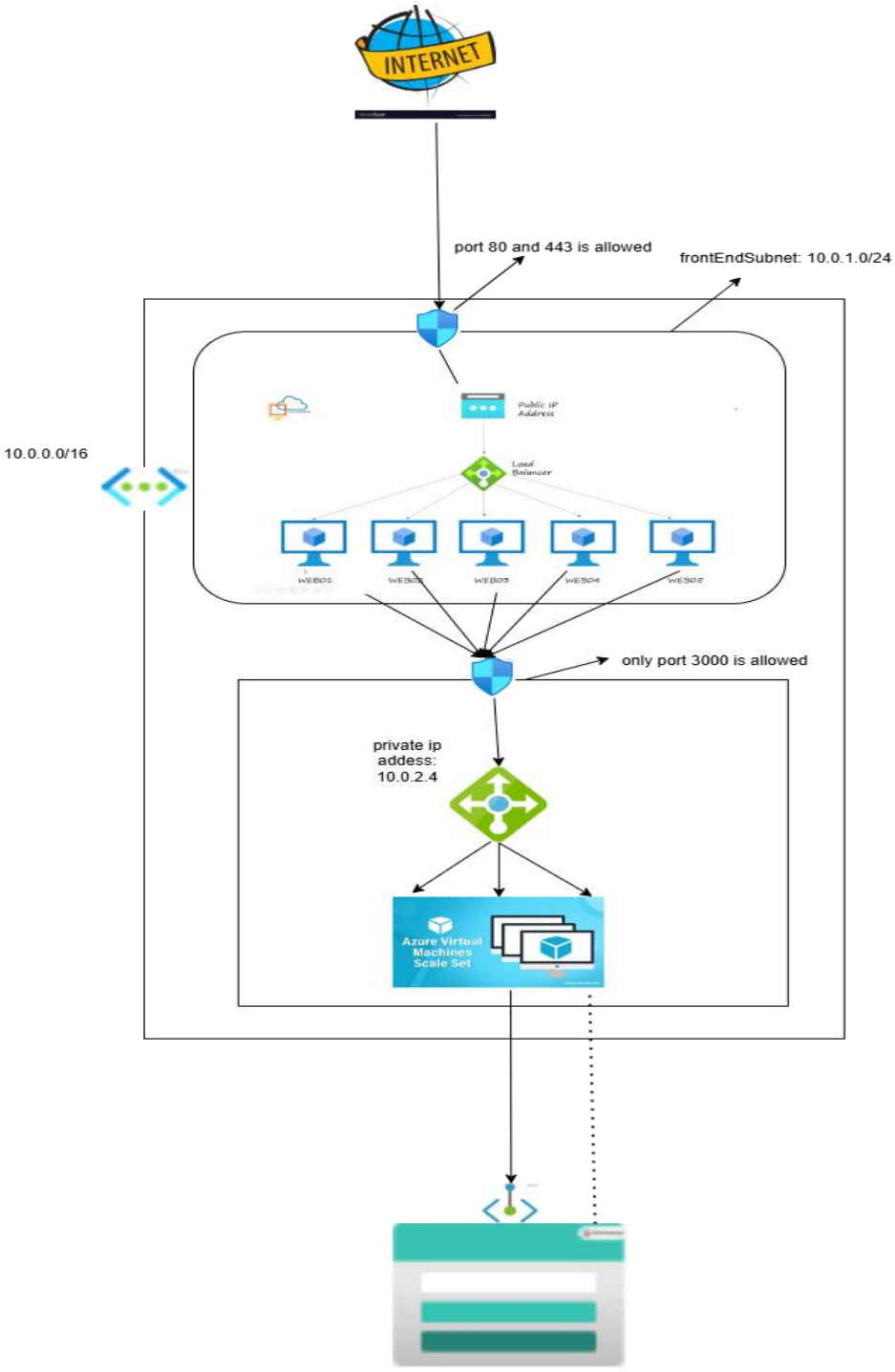
✓ **Secure by Default:** No internet exposure; strict internal traffic controls.

✓ **Highly Available:** Redundant across zones and update domains.

✓ **Cost-Efficient:** Auto-scaling optimizes resource usage.

✓ **Seamless Storage Integration:** Blob Storage + Private Endpoint ensures data privacy.

This architecture guarantees **performance, security, and reliability** for Contoso Boutique's backend operations.


INTERNET
port 80 and 443 is allowed
frontEndSubnet: 10.0.1.0/24
10.0.0.0/16
Public IP Address
Load Balancer
WEB01
WEB02
WEB03
WEB04
WEB05
only port 3000 is allowed
private ip addess: 10.0.2.4
Azure Virtual Machines Scale Set

🗄️ Tier 3: Data Storage Layer

🎯 Purpose

The **Data Storage Layer** serves as the **persistence tier** for static assets (e.g., product images) required by the backend application. Designed with **security and cost-efficiency** in mind, this layer leverages **Azure Blob Storage** instead of a traditional database due to current subscription constraints.

🔐 Core Component: Azure Blob Storage

📂 *Storage Configuration*

Setting	Value	Purpose
Account Type	Standard Blob Storage	Cost-effective for unstructured data (images)
Access Tier	Hot (frequent access)	Optimized for read-heavy workloads
Replication	GRS (Locally Redundant Storage)	Replicate data in to secondary region to ensure data resilience
Public Access	Disabled	Ensures no accidental public exposure

🛡️ *Security & Network Isolation*

- **Private Endpoint**
  - Restricts storage access **exclusively to the `backEndSubnet`**.
  - Eliminates exposure to the public internet.
- **Encryption**
  - **At-rest:** Azure Storage Service Encryption (SSE).
  - **In-transit:** TLS 1.2+ enforced.

🗁 *Blob Container Structure*

```
📁 image
│
├── 📄 product-1.jpg
├── 📄 product-2.jpg
```

⚡ Integration with Backend Tier

This storage account is securely integrated with the network hosting the backend tier via a private endpoint. It serves static images, which can be efficiently retrieved by the backend server.

🚫 Current Limitations & Justifications

Limitation	Workaround	Future Improvement
No managed database (e.g., Azure SQL, Cosmos DB)	Uses Blob Storage for simplicity	Migrate to PaaS database when budget permits
Geo redundant storage	Replicate data into other regions	To allow read access from both region, it is possible to use RA-GRS
Manual metadata management (e.g., `metadata.json`)	JSON files stored alongside images	Transition to a database for dynamic queries

Rationale for Blob Storage Over Database:

- **Cost:** Azure Blob Storage is significantly cheaper than managed databases on a student plan.
- **Simplicity:** No need for complex queries—only static file serving.
- **Scalability:** Blob Storage handles high read throughput efficiently.

Security Summary

✓ **No public access** (all traffic via Private Endpoint).

✓ **Encrypted data** (at-rest and in-transit).

✓ **Least-privilege access** (only backend subnet can connect).

Key Takeaways

- **Cost-effective solution** for static data storage under subscription limits.
- **Secure by design** with private networking and encryption.
- **Easily upgradable** to a database service when requirements evolve.

Project Conclusion

This project successfully implemented a **scalable, highly available, and secure three-tier architecture** on Microsoft Azure for Contoso Boutique's e-commerce product showcase. By logically separating the **presentation, application, and data storage layers**, the solution ensures **optimal performance, security, and cost-efficiency** while adhering to cloud best practices.

Key Achievements

1. Presentation Tier (Frontend)

- **Highly Available Web Interface**:
  - Deployed on a **Windows Server VM Scale Set (VMSS)** with **IIS**, spanning **three Availability Zones** for fault tolerance.
  - Auto-scaling ensures seamless handling of traffic spikes.
- **Secure Public Access**:
  - Restricted via **NSG rules (HTTP/HTTPS only)** and a **public load balancer** with health probes.

2. Application Tier (Backend)

- **Isolated Business Logic**:
  - Hosted on an **Ubuntu VMSS with Node.js**, accessible **only from the frontend subnet** (Port 3000).
  - **Internal Load Balancer** ensures secure, low-latency communication.
- **Defense-in-Depth Security**:
  - **NSG rules enforce strict subnet-level traffic control**, preventing unauthorized access.

3. Data Storage Tier (Azure Blob Storage)

- **Cost-Efficient & Secure Storage**:
  - Static product images stored in **Azure Blob Storage** with **Private Endpoints**, eliminating public exposure.
  - **Encrypted at rest and in transit**, with access restricted to the backend subnet.
- **Future-Proof Design**:
  - While currently using Blob Storage due to subscription limits, the architecture can **seamlessly integrate a managed database** (e.g., Azure SQL) when needed.

◇ Lessons Learned & Improvements

✅ **Security First**:

- Enforcing **least-privilege access** (NSGs, Private Endpoints) significantly reduces attack surfaces.
- **No public exposure** for backend/storage tiers prevents common cloud threats.

✅ **Scalability & Cost Optimization**:

- **Auto-scaling VMSS** ensures efficient resource usage.
- **Blob Storage** provides a budget-friendly alternative to databases for static content.

🚀 **Future Enhancements**:

- **Migrate to Azure SQL Database/Cosmos DB** when subscription limits allow for dynamic product metadata.
- **Implement Azure CDN** for faster global image delivery.
- **Add Azure Monitor & Log Analytics** for proactive performance tracking.

🎯 Final Thoughts

This project demonstrates how **a well-architected three-tier system** on Azure can deliver **enterprise-grade security, scalability, and reliability**—even under budget constraints. By leveraging **Azure's native services (VMSS, Private Endpoints, Blob Storage)**, Contoso Boutique now has a **modern, maintainable foundation** that can evolve with its business needs.

Contributor: Abinet Degefa