Question 1 : How could u setup these virtual machine? and what considerations are needed for pricing an OS license?

Answer :

Step 1: **Log in to the Azure Portal**

Step 2: **Create a Resource Group**

Step 3: **Create a Windows VM**

- Go to the "Virtual Machines" service and click on "Create" > "Azure Virtual Machine."
- Configure the VM details:
- **Subscription:** Choose the subscription.
- **Resource Group:** Select the resource group created in Step 2.
- **Instance Details:**
    - **Name:** Provide a name for the VM (e.g., `Win-VM-Test`).
    - **Region:** Choose the region closest to your users.
    - **Image:** Select a Windows OS .
    - **Size:** Select the appropriate VM size based on the testing requirements.
- **Administrator Account:** Set a username and password.
- Configure other settings like networking, disks, and monitoring (enable or disable as needed).
- Review and create the VM.

Step 4: **Create a Linux VM**

- Follow the same steps as above but select a Linux-based OS in the "Image" section (e.g., Ubuntu).
- Provide SSH public key or username/password credentials for admin access.
- Configure other settings similarly and create the VM.

Considerations for Pricing an OS License:
Azure offers two pricing options for operating systems:
1. **Bring Your Own License (BYOL):** If your organization already owns OS licenses
2. **Azure Pay-As-You-Go Licensing:** Azure automatically includes the license cost for the OS in the VM's pricing when using default OS images.

Question 2 : How could you ensure the data stored in azure storage is encrypted , and what encryption types are available.

Data in Azure Storage is Encrypted:

1. **Azure Storage Service Encryption (SSE)**

2. Azure automatically encrypts data at rest in all storage accounts using Storage Service Encryption (SSE). No additional configuration is required as this is enabled by default.

3. **Management Options**
    a. **Microsoft-Managed Keys**: Encryption keys are managed by Azure automatically. This is the default option.
    b. **Customer-Managed Keys (CMK)**: You can use your own keys stored in **Azure Key Vault** or a managed hardware security module (HSM). This provides more control over the encryption process and key rotation.
    c. **Customer-Provided Keys (CPK):** Allows you to provide your own encryption key when performing read/write operations to Azure Blob storage.
4. **Azure Disk Encryption for Virtual Machines**
5. **Double Encryption**

Types of Encryption Available in Azure Storage

1. Storage Service Encryption (SSE)
2. Azure Storage Transport Layer Security (TLS)
3. Azure SQL Transparent Data Encryption (TDE)
4. Azure Disk Encryption (ADE)

Steps to Verify and Configure Encryption

- **Verify Default Encryption**:
  Go to the **Azure Portal** → Navigate to your **Storage Account** → Under **Settings**, check the **Encryption** section to confirm encryption at rest is enabled.

- **Enable Customer-Managed Keys (CMK)**:
  Navigate to the storage account → **Encryption** → Select **Customer-Managed Keys** → Configure an Azure Key Vault.
- **Enforce Secure Transfer**:

Go to the storage account → **Configuration** → Enable **Secure Transfer Required** to ensure all traffic is encrypted.

- **Use Encryption Scope for Blob Containers**:
  Define an **encryption scope** at the container or blob level for more granular control of encryption settings.

Question 3 :
How could you configure this pipeline to meet this requirements?

If the deployment fails, notifying the team and providing actionable insights is critical. Here's how you can configure the pipeline to notify the team effectively while providing relevant failure details:

1. Enable Built-In Azure DevOps Notifications

1. Go to your **Azure DevOps Project** → **Project Settings** → **Notifications**.
2. Create a **New Notification Subscription**:
   a. **Trigger:** Pipeline Fails
   b. **Scope:** Select the specific pipeline(s) where this applies.
   c. **Recipients:** Add team members, group emails, or service accounts for integrations with Slack, Microsoft Teams, etc.
   d. **Details:** Include failure information in the notification.

2.Integrate Application Insights Alerts:

1. Configure **Azure Monitor** with **Application Insights** to detect runtime failures or performance issues after deployment.
2. Set up alert rules for error codes, latency, or availability issues.
3. Send notifications to the team via email, SMS, or integration with tools like PagerDuty, Slack, or Teams.

End-to-End Failure Notification Workflow

1. **Pipeline Fails:**

   Azure DevOps detects the failure and:

    a. Sends built-in email notifications to the team.
    b. (Optional) Posts a detailed message to Slack/Teams.
    c. (Optional) Creates a work item in Jira/Azure Boards.
2. **Error Details Provided:**

Notifications include:

    a. Error message (`$(System.TaskErrorMessage)`).
    b. Links to pipeline logs (`$(System.TaskLogUrl)`).
    c. Build and pipeline details (`$(Build.BuildNumber)` and `$(Release.ReleaseName)`).
3. **Team Takes Action:**

Team members use the provided information to troubleshoot, fix the issue, and trigger a redeployment.

Question 4: Which azure service would you use , and how could you perform the migration?

To migrate an on-premises SQL database to Azure while ensuring minimal downtime and maintaining database accessibility during the migration, you would use **Azure Database Migration Service (DMS)**. Azure DMS is designed specifically for seamless migrations with near-zero downtime.

Steps to Perform the Migration

1. Prepare the Source SQL Database

1. **Check Compatibility**:
    a. Use the **Data Migration Assistant (DMA)** to assess the database schema, compatibility issues, and unsupported features.
    b. Fix any compatibility issues identified by DMA.
2. **Enable Transactional Replication (for Online Migration)**:
    a. On the on-premises SQL Server, configure **transactional replication** to keep data synchronized during the migration.

2. Provision Azure Resources

1. **Create a Target Database**:
    a. In Azure, create a target database in **Azure SQL Database**, **Azure SQL Managed Instance**, or **Azure Virtual Machine (SQL Server)**, depending on your requirements.
2. **Set Up Azure Database Migration Service**:
    a. In the Azure Portal, create an instance of **Azure Database Migration Service**.
    b. Choose the appropriate pricing tier based on your performance and throughput requirements.

3. Perform the Migration

1. **Offline Migration (If Downtime Is Acceptable)**:
    a. Export the on-premises database using tools like **BACPAC**.
    b. Import the BACPAC file into the Azure SQL Database or Managed Instance.
    c. Steps:
        i. Export the database to a `.bacpac` file using SQL Server Management Studio (SSMS).
        ii. Upload the `.bacpac` file to Azure Storage.
        iii. Import the `.bacpac` file into the Azure SQL target database.
2. **Online Migration (Minimal Downtime)**:
    a. Use **Azure Database Migration Service** in **Online Mode**:
        i. Connect to the source (on-premises SQL Server) and the target (Azure SQL Database/Managed Instance).
        ii. Select the **online migration** option in the Azure DMS portal.
        iii. The service performs an initial data migration and keeps the target database synchronized with the source using transactional replication.
        iv. Once synchronization is complete, cut over to the Azure database during a scheduled maintenance window.
    b. This ensures minimal downtime, as the final switch happens almost instantaneously.

4. Validate the Migration

1. Verify that all data has been migrated successfully by comparing the source and target databases.
2. Test the functionality of applications connected to the Azure database.