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:

Step 1: Create an Azure DevOps Project and Repository

Before creating the pipeline, ensure that you have:

· A project created in Azure DevOps.

· The code for your application stored in the repository within Azure DevOps (either Git or TFVC).

Step 2: Create a New Pipeline

1. Log in to Azure DevOps

2. select Pipelines.

3. Create a New Pipeline:

a. Click New Pipeline.

b. Choose your repository where your application code is stored (e.g., Azure Repos Git or GitHub).

c. Select the pipeline type: You can either use YAML for a more customizable pipeline or the Classic Editor for a GUI-based approach. Here we'll use YAML for more flexibility, but the process can be similar for Classic Editor.

4. Choose a Template or Start from Scratch:

a. If you are starting from scratch, select Starter pipeline or create your own YAML file.

b. If your app has specific requirements (e.g., .NET, Node.js), you can choose the appropriate template.

Step 3: Define the YAML Pipeline

Step 4: Set Up Deployment Failure Notifications

To notify your team when a deployment fails, you need to set up a failure notification.

Option 1: Using Azure DevOps Notifications

1. Navigate to Project Settings:

a. Go to your Azure DevOps project.

b. Click on Project Settings (at the bottom left).

2. Set up Notifications:

a. In the Notifications section, select Service Hooks.

b. Click + New Service Hook to add a new notification.

c. Choose Email (or any other preferred method, like Slack, Teams, etc.) as the service to notify on failure.

d. In the Event drop-down, select Build failed (or Pipeline failed).

e. Configure who should receive the notification (e.g., specific team members, group, etc.).

f. You can configure the conditions, like only sending the email if the build status is failed.

Option 2: Configure Failure Notification in YAML Pipeline

You can use the "Email" notification via a custom task in the pipeline or use Azure DevOps service hooks, but another approach could be using a custom script (for example, with PowerShell or Bash) that triggers an email notification on failure.

Step 5: Test and Monitor the Pipeline

1. Run the Pipeline:

a. Commit your changes to your repository.

b. The pipeline should trigger on the specified branch (e.g., main).

c. Monitor the build and deployment process in Azure DevOps under Pipelines > Runs.

2. Check Notifications:

a. If the deployment fails, the team should receive the notifications you've set up (email, Slack, Teams, etc.).

3. Review Logs:

a. If the deployment fails, go to the Logs section of the pipeline to view detailed information on why the deployment failed.

b. This can help you troubleshoot issues such as configuration errors, connection issues, or missing dependencies.

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.