AZURE SCENARIOS BASED QUESTIONS

Name: Aparnaa M

ACE NO: ACE12502

Scenario 1:

Your team needs to deploy a virtual machine in azure portal. to test a new software application. The team has requested both windows and Linux virtual machine

How could you set up these virtual machines and what considerations are needed for pricing and OS license?

To set up both a Windows and a Linux virtual machine (VM) in the Azure portal for testing a new software application, follow these steps and take note of the important considerations for pricing and OS licensing?

ANS:

Steps to Set Up VMs in Azure Portal:

1. **Log in to Azure Portal**:
2. **Create the Windows VM**:
    a. In the Azure portal, go to **"Create a resource"**.
    b. Select **"Virtual Machine"** under the "Compute" category.
    c. Choose the **Windows** operating system image. Azure provides several Windows Server versions such as Windows Server 2019, 2022, etc., and other Windows-based options.
    d. Fill out the required configuration settings such as:
        i. **Subscription** and **Resource Group**
        ii. **VM Name**
        iii. **Region** where you want the VM to be deployed (make sure it's close to your test environment or users for optimal performance).
        iv. **Size** (choose an appropriate VM size based on your resource requirements for testing the software).
        v. **Authentication Type** (password or SSH key for logging in).
        vi. **Public IP address** (if you want remote access).
        vii. **Networking**: Configure network settings and ensure it's placed in the correct Virtual Network (VNet).

   e. Under the **OS disk** settings, ensure you choose the default **Windows OS disk** type (usually Standard SSD or Premium SSD, depending on performance needs).
   f. **Review + create** the VM and proceed with deployment.
3. **Create the Linux VM**:
   a. Follow similar steps as above for the Linux VM:
      i. Under **"Create a resource"**, select **"Virtual Machine"** and choose a **Linux** distribution such as Ubuntu, CentOS, or Red Hat.
      ii. Fill out the same required configuration settings (name, region, size, authentication, networking).
   b. Select the appropriate **Linux OS disk** type for performance (Standard SSD or Premium SSD).
   c. After reviewing and validating the configurations, **create** the VM.

Considerations for Pricing and OS Licensing:

1. **Windows VM Pricing**:
   a. **Windows Licensing** is included in the cost of the VM when using Azure's pay-as-you-go pricing model.
   b. The pricing will include the cost of both the virtual machine (based on size and region) and the Windows license.
   c. You can choose from various Windows editions, including **Windows Server** or **Windows 10/11** (for specialized testing purposes), which may affect pricing.
   d. For significant cost savings, you can use **Azure Hybrid Benefit** if you have existing on-premises Windows Server licenses with Software Assurance (this can lower VM costs significantly).
2. **Linux VM Pricing**:
   a. **Linux VMs** are typically cheaper than Windows VMs because there is no licensing cost for the OS.
   b. However, pricing will depend on the VM size, region, and disk types (Standard vs. Premium).
   c. Some specific Linux distributions, like Red Hat or SUSE, may involve additional costs due to the subscription requirements for support.
3. **VM Size Considerations**:
   a. The **size of the VM** will greatly impact pricing. You’ll need to choose an appropriate size (based on CPU, memory, and storage requirements) for both VMs.
   b. Test the resource consumption of your application to determine the optimal size for each machine.

    c. Azure provides different **tiers** (General-purpose, Compute-optimized, Memory-optimized, etc.), so select accordingly.

4. **Storage and Backup Costs**:
    a. In addition to the VM itself, you will need to consider the cost of the attached **disks** (OS disk and data disks).
    b. Azure provides different types of storage, and the pricing varies depending on whether you're using Standard HDD, Standard SSD, or Premium SSD.
    c. If backup is required, use Azure Backup or other storage solutions, which will also incur additional costs.
5. **Network Considerations**:
    a. If your VMs need to communicate with other resources in Azure, be aware of **bandwidth and network traffic** costs. Data transfer between different regions or out of Azure can incur charges.

Cost Estimation:

To estimate the costs accurately for both VMs, you can use the **Azure Pricing Calculator** to input your configuration details (VM sizes, regions, OS, and storage types) to get an estimate of the monthly cost.

Scenario 2

The it security team has requested the sensitive data stored in an azure storage account be encrypted to meet complaints requirements?

How could you ensure the data stored in azure storage is encrypted,and what encryption types are available?

Ans:

To ensure that sensitive data stored in an Azure Storage account is encrypted and meets compliance requirements, Azure provides several encryption options. You can configure encryption in various ways based on your security and compliance needs.

Azure Storage Encryption Options:

1. **Encryption at Rest (Default Encryption)**:
    a. By default, all data stored in Azure Storage accounts (Blob Storage, File Storage, Disk Storage, etc.) is encrypted at rest using **Server-Side Encryption (SSE)**. This encryption is **automatic** and requires no action from you when you create a storage account.
    b. **Types of Server-Side Encryption**:
        i. **Microsoft-managed keys (SSE with Microsoft-managed keys)**: This is the default encryption method where Microsoft handles the management of the encryption keys. It provides automatic encryption and decryption of your data stored in Azure.
        ii. **Customer-managed keys (SSE with customer-managed keys)**: This allows you to control the encryption keys used to encrypt your data. You can store your keys in **Azure Key Vault** and provide them to Azure Storage when creating or managing the storage account. This option provides more control over key rotation, access management, and auditing.
        iii. **Azure Key Vault Managed Keys (with Bring Your Own Key - BYOK)**: For enhanced security, you can bring your own encryption keys stored in Azure Key Vault to manage your storage account encryption. This allows you to define key policies, access permissions, and track usage with auditing capabilities.
2. **Encryption in Transit (Secure Data Transfer)**:
    a. Data in Azure Storage is **always encrypted in transit** using **Transport Layer Security (TLS)**. When data is transmitted between clients and the Azure Storage service, it's automatically encrypted, ensuring secure data transfer over the network.
    b. You do not need to manually configure encryption for data in transit, as it's automatically handled by the Azure platform.
3. **Azure Storage Service Encryption (SSE) for Data at Rest**:
    a. **Blob Storage**: All blobs in Azure Storage are encrypted automatically using server-side encryption. This includes both **Standard Blob Storage** and **Blob Storage with premium performance tiers**.
    b. **File Storage**: Azure File Shares are also encrypted by default using server-side encryption.

    c. **Azure Disks**: Managed disks used for virtual machines (VMs) are also encrypted using the default SSE with Microsoft-managed keys.

4. **Azure Disk Encryption (ADE)**:
    a. For **Virtual Machine Disks**, you can apply **Azure Disk Encryption (ADE)** to ensure the data on your VM disks is encrypted using either **BitLocker** (for Windows OS) or **dm-crypt** (for Linux OS). ADE is an added layer of encryption that operates on the OS disk and data disks.
    b. This encryption is separate from the standard Azure storage encryption and is particularly useful when you're running virtual machines with sensitive data.
5. **Encryption Options for Blob Storage**:
    a. For **Blob Storage**, Azure supports **Storage Service Encryption (SSE)** and **Client-Side Encryption (CSE)**:
        i. **Client-Side Encryption (CSE)**: You can encrypt the data before uploading it to Azure Storage using your own encryption mechanisms (e.g., using a client application that encrypts data before sending it). This provides you full control over the encryption keys, but requires extra management of the encryption process.

How to Ensure Encryption of Data in Azure Storage:

1. **Use Microsoft-managed keys (SSE)** (Default):
    a. When you create a new **Storage Account**, encryption at rest is enabled by default with Microsoft-managed keys. There is no need to manually configure anything for this option, and it ensures that your data is encrypted.
2. **Use Customer-managed keys (SSE with customer-managed keys)**:
    a. If you require control over your encryption keys, you can enable customer-managed keys. Here's how you can configure it:
        i. Create an **Azure Key Vault** and store your keys in it.
        ii. During the creation of your **Storage Account**, under the **Encryption** section, select **Customer-managed keys**.
        iii. Choose your **Key Vault** and the encryption key you want to use.
        iv. Ensure proper **Access Control** is set for your Key Vault to restrict access to the encryption keys.
3. **Azure Key Vault Managed Keys**:
    a. When using customer-managed keys, you can also leverage Key Vault features for key rotation, auditing, and logging, ensuring compliance with security and regulatory standards.
4. **Compliance Standards**:

    a. Azure Storage encryption complies with numerous industry regulations and standards, such as **GDPR**, **HIPAA**, **ISO 27001**, and **FIPS 140-2**. If your organization is subject to specific compliance requirements, you should verify that your encryption settings align with these standards.
    b. **Azure Security Center** can help you continuously monitor and assess compliance against these standards and recommend security configurations.

Steps for Enabling Encryption with Customer-Managed Keys:

1. Set up **Azure Key Vault** and create a key.
2. When creating your **Azure Storage Account**, choose **Customer-managed keys** in the encryption settings.
3. Link the **Key Vault** and select the key you want to use for encryption.
4. Confirm the settings, and deploy the storage account with encryption using your keys.

Scenario 3

You are responsible for setting up the devops pipeline in azure devops for your application.the pipeline must deploy code to an azure app service and notify the team if the deployment fails.

How could you configure this pipeline to meet the requirements?

Ans:

Steps to Configure the Pipeline with an Alternate Approach:

1. **Create the Pipeline in Azure DevOps**:
    a. In the **Azure DevOps portal**, navigate to **Pipelines** and click **New Pipeline**.
    b. Select the **repository** that contains your application code (Azure Repos, GitHub, etc.).

    c. For this approach, we'll use the **Classic Editor**, which is a UI-driven way to create the pipeline, making it easier for those who prefer visual configuration.

2. **Configure Build and Deployment Tasks**:
    a. In the Classic Editor, you can choose from several pre-defined tasks in the UI.
    b. Select the **Build** and **Release** pipeline to set up separate steps for building and deploying the application.

a. Build Pipeline (Continuous Integration):

- **Task 1: Restore Dependencies** (for example, `.NET`, Node.js, or other technologies):
    - Add the `Restore` task to restore project dependencies (e.g., `npm install` or `dotnet restore`).
- **Task 2: Build Project**:
    - Add a build task (e.g., `MSBuild`, `DotNet Build`, `npm build`) to compile the application code.
- **Task 3: Publish Artifacts**:
    - Use the `Publish Build Artifacts` task to save your build output (e.g., `.zip` file or deployment package) so it can be used later in the release pipeline.

b. Release Pipeline (Continuous Deployment):

- Create a **Release Pipeline** to deploy the application to Azure App Service.
- Add a new **stage** for your deployment (e.g., "Production" or "Staging").

3. **Add the Azure App Service Deployment Task**:
    a. In the Release pipeline, add the **Azure App Service Deploy** task.
    b. Configure the task:
        i. **Azure Subscription**: Select the service connection to Azure.
        ii. **App Service Name**: Provide the name of the Azure App Service where the app will be deployed.
        iii. **Package or Folder**: Point to the published artifacts (e.g., `.zip` file) from the build pipeline.
        iv. Choose the **deployment method** (e.g., **ZipDeploy**, **AzureRM WebApp Deployment**, etc.).
4. **Set Up Notifications for Deployment Failures**:
    a. **Option 1: Azure DevOps Built-in Notifications**:

        i. Go to **Project Settings > Notifications** in the Azure DevOps portal.
        ii. Create a new **subscription** to notify the team when a pipeline fails. You can configure the notification to send an email or trigger other actions based on specific conditions like pipeline failure.
        iii. Set the notification scope to **Pipeline** and select the **Build Failed** or **Release Failed** condition.

    b. **Option 2: Custom Notifications via Webhooks or Third-Party Tools**:
        i. Add a **Webhook** task in the release pipeline to send a custom failure notification (e.g., to a Slack channel, Microsoft Teams, or other messaging platforms).
        ii. To send a **Slack** notification:
            1. Set up an incoming **Slack Webhook** URL in your Slack workspace.
            2. Add a **Webhook** task in the release pipeline that triggers only on failure.
            3. Configure the webhook to send a message to your Slack channel when the deployment fails.
        iii. For **Teams**, you can similarly add a task for sending notifications to a **Teams channel** using a **Webhook** URL.
    c. **Option 3: Email Notification in Release Pipeline**:
        i. If you'd prefer to send a **custom email** when the deployment fails, you can add a **Send Email** task after the deployment task.
        ii. Use the condition `failed()` to ensure the email is only sent when the deployment task fails.

Example Pipeline Overview (UI-based with Classic Editor):

1. **Trigger**:
    a. Trigger on code changes to a specific branch (e.g., `main`).
2. **Build Steps** (in Build Pipeline):
    a. **Restore Dependencies** (e.g., `npm install`, `dotnet restore`).
    b. **Build** (e.g., `MSBuild`, `npm run build`).
    c. **Publish Artifacts** (store the `.zip` or packaged application).
3. **Release Steps** (in Release Pipeline):
    a. **Deploy to Azure App Service**:
        i. Deploy the artifact to an Azure App Service.
        ii. Configure the deployment method (e.g., **ZipDeploy**).
4. **Failure Notification**:

a. **Azure DevOps Notifications** (email alert on failure).
b. Alternatively, configure a **Webhook** to notify Slack/Teams.
c. Or use the **Send Email** task after deployment with a failure condition (`failed()`).

Configuring the Email Notification (Classic UI Example):

1. After configuring your release tasks, go to the **Release Pipeline** settings.
2. Click on **Add Task**, and choose the **Send Email** task.
3. Under the **Condition** field, select `failed()` to ensure the email is sent only if the deployment fails.
4. Configure the email recipient list, subject, and body (including dynamic variables like error details if available).

Scenario 4

Your organization is moving its on premises sql database to azure.The database must remain accessible during migration with minimal downtime.

Which azure service could you use,How could you perform the migration?

Ans:

To migrate your on-premises SQL database to **Azure** with **minimal downtime**, the ideal service to use would be **Azure Database Migration Service (DMS)**. This service is designed to migrate your database to Azure with minimal disruption and allows you to keep the database accessible during the migration process.

Azure Database Migration Service (DMS):

Azure Database Migration Service provides a seamless way to perform online migrations with minimal downtime, and it supports migration of SQL Server databases (on-premises or other SQL Server instances) to **Azure SQL Database** or **Azure SQL Managed Instance**.

Steps to Perform the Migration Using Azure DMS:

1. **Set Up Azure Database Migration Service**:
    a. In the **Azure portal**, navigate to **"Create a resource"** and search for **"Azure Database Migration Service"**.
    b. Create a new instance of DMS by providing basic information (subscription, resource group, region, and name).
    c. Once the service is created, you'll have the option to create a **migration project**.
2. **Prepare the Source Database**:
    a. **Backup your source database**: Before starting any migration, it's a best practice to take a full backup of your on-premises SQL database.
    b. Ensure that the source database is compatible with the destination Azure database. Azure DMS supports both **Azure SQL Database** and **Azure SQL Managed Instance**.
    c. If using **SQL Server**, ensure that the database is in **Full Recovery Model** to allow for transactional log backups, which is necessary for the migration process.
3. **Create a Migration Project in DMS**:
    a. In **Azure Database Migration Service**, create a new **project** for your migration.
    b. Select **SQL Server** as the source and **Azure SQL Database** or **Azure SQL Managed Instance** as the target.
    c. Provide the necessary connection details for the **source database** (on-premises SQL Server) and **target database** (Azure SQL Database or Managed Instance).
4. **Select the Migration Method**:
    a. **Online Migration (Recommended for Minimal Downtime)**:
        i. Azure DMS allows you to perform an **online migration**, meaning the database remains online during the migration process.
        ii. **Online migration** is performed in two phases:
            1. **Initial Data Migration**: DMS begins by copying the schema and data from your on-premises database to the Azure target database.
            2. **Continuous Data Sync**: After the initial migration, Azure DMS keeps the source and target databases synchronized using **log-based replication**. This minimizes downtime

because changes made to the on-premises database during the migration (e.g., inserts, updates, deletes) are continuously applied to the target database.

    iii. You can set a **cutover time** when the final synchronization happens and the migration is complete. The downtime during this final cutover phase is typically very short.

  b. **Offline Migration** (if minimal downtime is not a strict requirement):
    i. With an offline migration, the source database is not accessible while the migration occurs. However, this approach is typically used when minimal downtime is not a priority.

5. **Run the Migration**:
   a. Once the migration project is set up, start the **migration process**.
   b. During the migration, DMS will handle:
      i. Schema and data migration.
      ii. Keeping the source and target databases in sync for **online migration**.
   c. You can monitor the progress of the migration directly in the Azure portal and review any issues or warnings.
6. **Perform the Cutover**:
   a. After the migration completes and synchronization is finished, you will schedule a **cutover** to finalize the migration.
   b. During the cutover, any remaining changes to the source database are applied to the target database, and the application will begin using the Azure database.
   c. This cutover phase is typically very short (a few minutes), resulting in minimal downtime.
7. **Post-Migration Tasks**:
   a. **Verify the target database**: After the cutover, ensure the Azure SQL Database or Managed Instance is fully operational and performs well.
   b. **Test application connectivity**: Make sure that all applications and services that rely on the database are correctly connected to the new Azure database.
   c. **Monitor the performance**: Azure provides various monitoring tools like **Azure Monitor** and **SQL Insights** to monitor the health and performance of the target database.
   d. **Backup and security**: Set up appropriate backup and security configurations for your Azure SQL Database, ensuring you follow best practices for **automated backups** and **threat detection**.

- **Compatibility**: Azure DMS supports a wide range of SQL Server versions and can migrate to both **Azure SQL Database** and **Azure SQL Managed Instance**.