**Scenario 1:**

Your team needs to deploy a virtual machine in Azure Portal to test a new software application. Here the team has requested both Linux and Windows virtual machines.

**Question:**

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

**Steps to Set Up Virtual Machines in Azure Portal:**

1. Log in to Azure Portal:

Open the Azure Portal and ensure you have the necessary permissions to create resources.

2. Create a Virtual Machine for Linux:

Navigate to Create a resource > Compute > Virtual Machine.

Choose Region: Select the region that is geographically close to your team.

Under Image, select a Linux distribution (e.g., Ubuntu, CentOS, Red Hat, etc.).

Set up the VM Size: Based on the required performance (CPU, RAM, etc.), choose the appropriate size.

Authentication Type: For Linux, you will typically use SSH public key for authentication (or a password, depending on your preferences).

Networking: Choose a Virtual Network (VNet) or create a new one. You will also need to select a subnet and set up network security groups (NSGs) for access control.

After configuring the VM, review the details and click Create.

3. Create a Virtual Machine for Windows:

Similarly, navigate to Create a resource > Compute > Virtual Machine.

Choose the Region: Preferably the same region as the Linux VM for consistency.

Under Image, select a Windows Server version VM Size: Select the appropriate size based on your requirements.

Authentication Type: For Windows VMs, use Password authentication (you will need to create an admin username and password).

Networking: Choose the same VNet used for the Linux VM if you need them to communicate with each other or create a new VNet if isolation is required.

Review the configuration and click Create.

**Considerations for Pricing and OS Licensing:**

1. Pricing for Linux Virtual Machines:

Linux VMs in Azure generally do not incur any additional OS licensing charges.

Pricing for Linux VMs is based on the VM size (CPU, RAM, etc.), the storage used (Standard SSD, Premium SSD, etc.), and data transfer.

Example: For a VM running Ubuntu, the only charges will be for the compute (VM) and storage resources used.

Pay-as-you-go pricing: With Linux VMs, you pay only for the resources you consume (compute, storage, and networking).

Discount Options:

You can use Azure Reserved Instances (RI) for up to 3 years to save costs if you expect to use these VMs long-term.

Azure Spot VMs: These are lower-cost options if your application can tolerate interruptions.

2. Pricing for Windows Virtual Machines:

Windows VMs include OS licensing costs as part of the pricing, which is factored into the VM price.

The OS license for Windows Server is embedded into the cost of the VM.

The exact cost depends on the VM size (CPU, RAM), region, and Windows version.

Azure Hybrid Benefit: If your organization has existing Windows Server licenses with Software Assurance, you can apply the Azure Hybrid Benefit to reduce the OS licensing cost when using Windows VMs in Azure.

This is a major cost-saving benefit, and if you are eligible for it, you should always leverage it.

Pay-as-you-go pricing: Similar to Linux, you pay for the compute resources and storage, but the Windows license cost is added to the overall price.

3. Storage Considerations:

Disk Storage: You will need to choose between different types of disks based on performance needs (e.g., Standard HDD, Standard SSD, Premium SSD).

Linux VM: Choose the appropriate storage type (Standard SSD or Premium SSD based on performance).

Windows VM: The pricing for storage is the same as for Linux VMs, but you should ensure that you choose the right performance tier depending on the application’s needs.

Managed Disks: Both Windows and Linux VMs in Azure use Azure Managed Disks, which offer better performance, scalability, and availability than unmanaged disks.

Backup and Recovery: Azure offers Azure Backup for taking regular backups of both Linux and Windows VMs. This should be factored in if you want to ensure data recovery.

4. Networking and Data Transfer:

If both VMs need to communicate with each other, they should be on the same Virtual Network (VNet) or have VNet peering configured.

Public IP Addresses: If your VMs need to be publicly accessible (e.g., to access the application externally), you will need to assign Public IPs to each of them, which incurs additional costs.

Data Transfer Costs: Azure typically charges for outbound data transfer (i.e., data leaving the Azure region). Keep this in mind if you expect heavy data transfer between regions or to the internet.

5. Other Cost Considerations:

Azure Reserved Instances (RI): For long-term use, you can commit to 1-3 years to reduce the overall cost of both Linux and Windows VMs.

Virtual Machine Scale Sets (VMSS): If you plan to scale up the number of VMs based on load, VMSS can be used to auto-scale resources, but this should also be factored into pricing.

**Scenario 2:**

The IT Security team has requested that sensitive data stored in an Azure storage account be encrypted to meet compliance requirements.

**Question:**

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

**Step 1: Understand Azure Storage Encryption Options**

Azure provides multiple encryption options to secure sensitive data stored in Azure Storage Accounts. Encryption is automatically enabled for data at rest and data in transit, but you have several encryption options that provide different levels of security.

The main types of encryption available in Azure Storage are:

Encryption at Rest: Data is encrypted when stored in Azure Storage and remains encrypted while it is stored.

Azure Storage Service Encryption (SSE): Automatically encrypts data at rest using Microsoft-managed keys or your own keys (via Customer-Managed Keys (CMK)). This is the default method for encryption in Azure Storage.

Azure Key Vault Integration: If you need to have full control over encryption keys, you can store your encryption keys in Azure Key Vault and manage them yourself.

Encryption in Transit: Ensures that data is encrypted during transmission over the network.

HTTPS (TLS/SSL): Data transferred between your application and Azure Storage is encrypted using Transport Layer Security (TLS), ensuring secure transmission.

Azure Disk Encryption: Applies specifically to Azure Virtual Machines that use managed disks (this applies if you're working with VM-based storage).

Encryption at the VM disk level using BitLocker (Windows) and DM-Crypt (Linux).

**Step 2: Enable Encryption for Azure Storage Account**

Azure Storage automatically encrypts data at rest using Storage Service Encryption (SSE). However, you may want to use customer-managed keys (CMK) to have more control over the encryption process.

Encryption at Rest:

Storage Service Encryption (SSE): This is the default encryption mechanism that ensures that your data at rest is automatically encrypted.

How to Enable:

SSE is enabled by default in Azure Storage accounts. When you create a new Storage Account or have an existing one, encryption at rest is enabled automatically.

You don’t need to take any action for basic encryption. It’s managed by Microsoft using Microsoft-managed keys.

Customer-Managed Keys (CMK) for SSE:

When to Use: If you need control over the encryption keys or have compliance requirements that require it, you can use your own encryption keys for SSE.

How to Enable:

Create an Azure Key Vault and store your Key Encryption Key (KEK).

When creating or modifying your storage account, you can choose Customer-Managed Keys (CMK) from Azure Key Vault for SSE.

Once the storage account is set up with CMK, all data written to the storage account will be encrypted using the customer-managed key.

Azure Key Vault Integration:

To ensure compliance and enhance security, you can store and manage your encryption keys in Azure Key Vault. This way, you have full control over access and lifecycle management of the keys.

How to Configure:

Navigate to the Storage Account in the Azure Portal.

Under the Encryption section, select Use your own keys and provide the Key Vault and key that will be used for encryption.

Note: Azure Key Vault also provides features like Key Vault policies, Access Control (IAM), and Audit Logging to ensure you have full control over who can access and use the keys.

**Step 3: Ensure Encryption for Data in Transit**

While data at rest is encrypted using SSE, data in transit (data being transferred to/from Azure Storage) is encrypted by default.

HTTPS (TLS/SSL): Ensure that your applications and services communicate with Azure Storage over HTTPS. This is important for ensuring that data is encrypted as it moves across the network.

Always access Azure Storage resources using the HTTPS endpoint.

Note: Azure enforces the use of HTTPS for all data transfers by default, and HTTP access is blocked by default for all storage accounts.

**Step 4: Verify and Audit Encryption Settings**

Check Encryption Status:

You can verify whether your storage account is using Microsoft-managed keys or Customer-managed keys (CMK) in the Azure Portal.

Navigate to your Storage Account > Encryption.

You will see the status of the encryption and whether Azure Key Vault is being used for customer-managed keys.

Enable Soft Delete:

Soft Delete for Blobs and File Shares should be enabled to avoid accidental data loss, which ensures that deleted data is recoverable for a specified retention period.

Use Azure Monitor & Logs:

Use Azure Monitor and Azure Security Center to monitor and track any security-related activities and key usage in your storage account.

Azure Activity Log and Key Vault logs can help you keep track of when encryption keys were accessed or changed.

Encryption Types Available in Azure Storage

Storage Service Encryption (SSE) with Microsoft-managed Keys:

Automatically encrypts all data stored in the Azure Storage account.

Managed by Microsoft.

Storage Service Encryption (SSE) with Customer-Managed Keys (CMK):

Gives you control over the encryption keys.

You can store and manage keys in Azure Key Vault.

Encryption in Transit (Data Transfer via HTTPS):

All data transferred between your application and Azure Storage is encrypted using Transport Layer Security (TLS).

**Step 5: Compliance and Key Management**

Compliance Certifications:

Azure Storage encryption options meet numerous industry compliance standards, including GDPR, ISO 27001, HIPAA, PCI-DSS, etc. Encryption with Customer-managed keys offers additional control for compliance needs.

Access Control and Auditing:

To comply with security and compliance policies, manage access to encryption keys with Azure Key Vault using Access Control (IAM) and Audit Logging.

You can set access policies to ensure that only authorized users and applications can access and manage the keys.

**Scenario 3:**

You are responsible for setting up 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.

**Question:**

How could you configure this pipeline to meet the requirements?

**Step 1: Set Up the Azure DevOps Project**

Create a Project in Azure DevOps:

If you haven’t already, create a new project in Azure DevOps to store your repository, pipelines, and other resources.

Go to the Azure DevOps portal and click New Project.

Give the project a name, select visibility (Private or Public), and click Create.

**Step 2: Set Up the Repository**

Choose Your Repository:

If your code is hosted in Azure DevOps Git repositories, create a new Git repository within the project or connect an existing repository.

Alternatively, you can use other repository services such as GitHub, Bitbucket, etc., by connecting them to your Azure DevOps project.

Push Your Application Code:

Push your application code (e.g., a web application) into the chosen repository in Azure DevOps.

**Step 3: Create a New Pipeline in Azure DevOps**

Navigate to Pipelines:

In your Azure DevOps project, go to the Pipelines section.

Click New Pipeline to create a new pipeline for your application.

Select the Repository:

Select the repository where your application code is stored (Azure Repos Git, GitHub, etc.).

Choose the repository from the list and click Continue.

Select a Pipeline Template:

Azure DevOps provides various templates for common scenarios. For deploying to Azure App Service, choose the Azure Web App template from the list.

If no template fits, you can start from a YAML file or an empty pipeline.

**Step 4: Define the Pipeline for Code Deployment**

Define Build and Deploy Stages:

Build Stage:

The Build stage is responsible for compiling the code, running tests, and packaging it into a deployable artifact (e.g., a ZIP file or Docker container).

You can use tasks like dotnet build, npm install, npm run build, or any relevant build commands depending on your application.

Deploy to Azure App Service:

The Deploy stage will deploy the packaged code to your Azure App Service.

To add this stage, you will need to use the Azure Web App deployment task.

Publish Artifacts:

To make sure the built code is available for deployment, use the Publish Build Artifacts task:

**Step 5: Set Up Notifications for Deployment Failures**

Enable Notifications in Azure DevOps:

To notify the team if the deployment fails, configure email notifications or use Azure DevOps Service Hooks.

Email Notification:

Go to Project Settings in Azure DevOps.

Under General, select Notifications.

Create a new subscription with conditions such as:

Event type: Pipeline run failed.

Subscribers: Select team members or distribution lists to receive the notification.

Azure DevOps Service Hook (Optional):

You can integrate Azure DevOps with Slack, Teams, or other communication platforms to receive failure alerts.

To set this up:

Go to Project Settings > Service Hooks.

Choose the desired service (e.g., Slack).

Set the condition to trigger when a Pipeline fails.

Set up the Slack channel where the failure message should be posted.

Set Failure Conditions in the Pipeline:

In your pipeline YAML, you can set conditions to ensure failure notifications are triggered when deployment fails.

**Step 6: Review and Run the Pipeline**

Review the Pipeline Configuration:

Ensure the pipeline YAML file is correct and that all necessary configurations (e.g., Azure subscription, app name, etc.) are set.

Run the Pipeline:

Once everything is set up, trigger the pipeline by making a code change (e.g., push a commit to the main branch).

Azure DevOps will build and deploy the application to Azure App Service and notify the team if the deployment fails.

Monitor Pipeline and Fix Issues:

If the pipeline fails, check the logs in Azure DevOps to identify the problem.

Resolve the issues and rerun the pipeline.

**Scenario 4:**

Your organization is moving its on premises SQL database to Azure. The database must remain accessible during migration with minimal down time.

**Question:**

Which Azure service would you use, and how could you perform the migration? configuration, deployment issues, etc.).

**Step 1: Choose the Right Azure Service**

For migrating an on-premises SQL database to Azure with minimal downtime, the best Azure service to use is Azure SQL Database Migration using the Azure Database Migration Service (DMS). Specifically, DMS supports near-zero downtime migrations by using the online migration method.

Additionally, if you're moving to Azure SQL Managed Instance or Azure SQL Database, the following services are relevant:

Azure Database Migration Service (DMS): Supports both online and offline migrations for SQL Server databases. This service allows for minimal downtime and real-time data replication.

SQL Server transactional replication: This method allows ongoing data changes to be replicated during migration.

**Step 2: Prerequisites for Migration**

Prepare Azure Environment:

Set up the Azure SQL Database or Azure SQL Managed Instance where the database will be migrated.

Ensure that the Azure environment meets the requirements for the SQL server instance, including network, storage, and performance configurations.

Prepare On-Premises Environment:

Install SQL Server Management Studio (SSMS) on the source server.

Ensure that the SQL Server instance on-premises is running a compatible version for migration.

Ensure Network Connectivity:

The on-premises SQL Server and Azure resources must be able to communicate over the network.

Consider setting up VPN or ExpressRoute if you're migrating a large database for better network security and speed.

**Step 3: Perform the Migration Using Azure Database Migration Service (DMS)**

Set Up Azure Database Migration Service:

Go to the Azure portal and navigate to Azure Database Migration Service.

Click on Create a resource, select Database Migration Service, and configure it:

Choose the subscription and resource group.

Define the migration project and specify the source (on-premises SQL Server) and destination (Azure SQL Database or Azure SQL Managed Instance).

Configure the Migration:

Select Migration Type: Choose the migration type as Online Migration for minimal downtime (this option replicates data continuously during migration, keeping both the source and destination in sync).

Configure Source and Target:

Choose your source SQL Server instance (on-premises).

Select your target Azure SQL Database or Azure SQL Managed Instance.

Run the Assessment:

Before proceeding, you can run an assessment to identify any compatibility issues with the on-premises database and the target Azure environment.

Azure Database Migration Service can check for things like version compatibility, unsupported features, and configuration recommendations.

Perform the Data Migration:

Start the Migration: Once the assessment is successful, you can start the migration process.

Initial Sync: The DMS tool will first perform a full copy of the database (initial sync).

Ongoing Sync: After the initial sync, the service will continuously sync data changes from the on-premises database to the Azure database to ensure both databases stay in sync during the migration.

Cut Over:

When you are ready for the final switchover to the Azure database, perform a cutover. This is where the final delta changes are replicated to the Azure SQL Database.

Once the cutover is complete, you can stop replication, and the database will now be fully operational in Azure.

Monitor the Migration:

During the migration process, monitor the status of the migration in the Azure portal.

Use Azure Monitor to check for performance metrics and any issues that may arise during the migration.

**Step 4: Validation and Post-Migration Tasks**

Validate the Migration:

After the migration is complete, validate that the database has been successfully migrated:

Check data integrity.

Validate that the application and queries are working as expected.

Switch the Application to the New Azure Database:

Update your application’s connection strings to point to the new Azure SQL Database or Managed Instance.

Perform Post-Migration Optimization:

Check performance metrics (e.g., query performance, indexing, auto-scaling) in the new Azure environment.

Consider using Azure SQL Database features like automatic tuning, auto-scaling, and serverless computing for optimization.

Decommission the On-Premises Database:

Once the migration is successful and the new environment is fully operational, decommission the on-premises database to avoid unnecessary costs.