**Scenario - 1**

### **1. Log in to Azure CLI:**

az login

### **2. Create a Resource Group:**

az group create --name MyResourceGroup --location eastus

### **3. Create a Virtual Network (VNet):**

az network vnet create --resource-group MyResourceGroup --name MyVNet --subnet-name MySubnet

### **4. Create a Public IP Address:**

az network public-ip create --resource-group MyResourceGroup --name MyPublicIP

### **5. Create Network Interface (NIC):**

az network nic create --resource-group MyResourceGroup --name MyNIC --vnet-name MyVNet --subnet MySubnet --public-ip-address MyPublicIP

### **6. Deploy a Windows Virtual Machine:**

az vm create \  
 --resource-group MyResourceGroup \  
 --name MyWindowsVM \  
 --nics MyNIC \  
 --image Win2022Datacenter \  
 --admin-username azureuser \  
 --admin-password myPassword123! \  
 --size Standard\_B1s

### **7. Deploy a Linux Virtual Machine:**

az vm create \  
 --resource-group MyResourceGroup \  
 --name MyLinuxVM \  
 --nics MyNIC \  
 --image UbuntuLTS \  
 --admin-username azureuser \  
 --generate-ssh-keys \  
 --size Standard\_B1s

### **8. Verify the Deployment:**

az vm show --resource-group MyResourceGroup --name MyWindowsVM  
az vm show --resource-group MyResourceGroup --name MyLinuxVM

**Scenario - 2**

### **1. Enable Default Storage Encryption (at Rest)**

Azure Storage automatically encrypts all data at rest by default using **Storage Service Encryption (SSE)**. You do not need to enable anything if you are okay with Microsoft-managed keys.

* If you want to use **Customer-managed keys (CMK)** stored in **Azure Key Vault**, follow these steps:

**Create Key Vault:**

az keyvault create --name MyKeyVault --resource-group MyResourceGroup --location eastus

**Create Encryption Key:**

az keyvault key create --vault-name MyKeyVault --name MyEncryptionKey --kty RSA --size 2048

**Enable CMK for Azure Storage:**

az storage account encryption-scope create \  
 --resource-group MyResourceGroup \  
 --account-name mystorageaccount \  
 --name MyEncryptionScope \  
 --key-id "<https://MyKeyVault.vault.azure.net/keys/MyEncryptionKey>"

### **2. Ensure Data in Transit is Encrypted (HTTPS)**

* **Force HTTPS:** Ensure that all data transferred to/from Azure Storage uses HTTPS by setting the "Secure transfer required" option.

**Enable Secure Transfer:**

az storage account update \  
 --resource-group MyResourceGroup \  
 --name mystorageaccount \  
 --https-only true

### **3. Enable Azure Disk Encryption - For Virtual Machines**

If you're using Azure VMs, enable **Azure Disk Encryption** for OS and Data disks to encrypt them.

**Enable Disk Encryption:**

az vm encryption enable \  
 --resource-group MyResourceGroup \  
 --name MyVM \  
 --disk-encryption-keyvault MyKeyVault \  
 --volume-type ALL

### **4. Cost Optimization:**

* **Use Microsoft-managed keys** for cost-effective encryption if customer-managed keys (CMK) are not a compliance requirement.
* **Use Standard Storage Tiers** (e.g., Standard Blob Storage or Standard File Storage) instead of Premium, unless performance is a requirement.
* **Consider Azure Blob Lifecycle Management** to automatically move infrequently accessed data to **Cool** or **Archive** tiers for lower costs.

**Set Lifecycle Policy to Archive:**

az storage blob service-properties update \  
 --account-name mystorageaccount \  
 --delete-soft-delete-retention-days 30 \  
 --container-deletion-available true

### **Encryption Types Available in Azure:**

1. **Storage Service Encryption (SSE)**:
   1. **Azure-managed keys**: Default encryption using Microsoft-managed keys.
   2. **Customer-managed keys (CMK)**: Allows you to use your own encryption keys from Azure Key Vault for greater control over encryption.
2. **Azure Disk Encryption (ADE)**:
   1. For Virtual Machine disks, using **BitLocker** (Windows) or **DM-Crypt** (Linux).
3. **Client-Side Encryption**:
   1. You can also perform encryption on the client side before uploading data to Azure. This gives you full control over the encryption process but requires managing your own encryption keys.
4. **Encryption in Transit**:
   1. Data is encrypted in transit using **HTTPS**, ensuring secure communication with Azure Storage.

### **Steps to Optimize Costs While Ensuring Encryption:**

1. **Use Microsoft-Managed Keys for Simplicity and Cost Savings**:
   1. By default, Azure Storage uses Microsoft-managed keys for encryption. This option incurs no additional cost as Azure handles the key management for you. If your compliance requirements allow it, this is the most cost-effective and simplest solution.
2. **Leverage Azure Key Vault**:
   1. If you require customer-managed keys (CMK) for compliance, use **Azure Key Vault** to store your keys. While Key Vault incurs a small cost (based on the number of keys and operations), it is cost-effective when using Azure’s built-in key management services.
   2. **Cost optimization tip**: Consider using **Azure Key Vault Standard** tier rather than the Premium tier, unless you specifically need features like HSM-backed keys or higher throughput.
3. **Choose the Right Storage Performance Tier**:
   1. Azure Storage offers different performance tiers (Standard and Premium). Standard is much more cost-effective for most use cases, while Premium is optimized for high-performance workloads.
   2. **Cost optimization tip**: If encryption is not required for performance, choose the **Standard** tier for Blob Storage and File Storage.
4. \*\*Utilize **Azure Blob Lifecycle Management**:
   1. If you are storing large amounts of data that needs to be encrypted and retained for compliance purposes, consider

**Scenario – 3**

### **1. Create a New Pipeline in Azure DevOps:**

* **Go to Azure DevOps**: Navigate to your Azure DevOps project and go to **Pipelines**.
* **Create a New Pipeline**:
  + Click on **New Pipeline** and select the source (e.g., GitHub, Azure Repos).
  + Choose the repository that contains your application code.

### **2. Configure the Pipeline YAML File:**

You'll need to define the pipeline configuration in a YAML file or use the classic editor. Below is an example of how to create a YAML pipeline that deploys to Azure App Service and includes a failure notification.

#### **Example Pipeline YAML Configuration:**

yaml

Copy

trigger:  
 branches:  
 include:  
 - main  
  
stages:  
 - stage: Build  
 jobs:  
 - job: BuildJob  
 pool:  
 vmImage: 'ubuntu-latest'  
 steps:  
 - task: UseNode@2  
 inputs:  
 versionSpec: '14.x'  
 addToPath: true  
 - task: Npm@1  
 inputs:  
 command: 'install'  
 - task: Npm@1  
 inputs:  
 command: 'run build'  
  
 - stage: Deploy  
 dependsOn: Build  
 jobs:  
 - job: DeployJob  
 pool:  
 vmImage: 'ubuntu-latest'  
 steps:  
 - task: AzureWebApp@1  
 inputs:  
 azureSubscription: 'YOUR-AZURE-SUBSCRIPTION'  
 appName: 'YOUR-APP-SERVICE-NAME'  
 package: '$(Build.ArtifactStagingDirectory)/your-package.zip'  
 deployToSlotOrASE: true  
 resourceGroupName: 'YOUR-RESOURCE-GROUP'  
  
 - stage: NotifyFailure  
 condition: failed()  
 jobs:  
 - job: NotifyJob  
 pool:  
 vmImage: 'ubuntu-latest'  
 steps:  
 - script: |  
 echo "Deployment Failed!"  
 # Send a notification email or message via Teams/Slack  
 # Example using Azure DevOps API or third-party tools like Slack/Teams webhook  
 curl -X POST -H 'Content-type: application/json' --data '{"text":"Deployment failed for App Service!"}' YOUR-WEBHOOK-URL  
 displayName: 'Notify Team via Slack/Teams'

### **Breakdown of the Pipeline:**

1. **Trigger**:
   1. The pipeline triggers on changes to the main branch.
2. **Build Stage**:
   1. This stage installs dependencies (Node.js in this case) and builds the application (customize this step based on your app’s technology).
3. **Deploy Stage**:
   1. **AzureWebApp** task: This deploys the application to your **Azure App Service**.
      1. azureSubscription: This is your Azure service connection configured in Azure DevOps.
      2. appName: Name of the Azure App Service.
      3. package: Path to the build artifact (e.g., .zip file of your app).
      4. deployToSlotOrASE: Set to true if deploying to a slot.
4. **NotifyFailure Stage**:
   1. This stage is triggered if the deployment fails, using the condition: failed() clause.
   2. In this stage, a notification is sent out (you can customize it to notify via Slack, Teams, or email).

### **3. Set Up Azure Subscription and Service Connection:**

* In Azure DevOps, create a service connection to your **Azure subscription**.
  + Go to **Project Settings** > **Service Connections** > **New Service Connection**.
  + Choose **Azure Resource Manager** and authenticate to link your Azure subscription.

### **4. Set Up Notification for Deployment Failure:**

To notify the team if the deployment fails, there are a few ways to handle this:

* **Email Notifications**: You can use the built-in Azure DevOps notification system for failures.
  + Go to **Project Settings** > **Notifications** > **+ New subscription**.
  + Choose **Build** or **Release pipeline** and set the subscription to notify on **failure**.
* **Using a Webhook (e.g., Slack/Teams)**:
  + Set up a webhook in your messaging platform (Slack, Microsoft Teams).
  + In the NotifyFailure stage above, use a script to send a message to Slack/Teams via the configured webhook URL.

Example Slack webhook integration:

bash

Copy

curl -X POST -H 'Content-type: application/json' --data '{"text":"Deployment failed for App Service!"}' YOUR-WEBHOOK-URL

### **5. Run and Test the Pipeline:**

* Commit the YAML file and push it to your repository.
* The pipeline should run automatically based on your trigger (e.g., when code is pushed to the main branch).
* Verify the deployment to the **Azure App Service** and confirm that notifications are sent when a failure occurs.

**Scenario - 4**

#### **1. Prepare Your Environment**

1. **Set Up Azure Environment:**
   1. **Azure SQL Managed Instance** or **Azure SQL Database**: Depending on your migration requirements (e.g., compatibility), create either an **Azure SQL Managed Instance** or **Azure SQL Database**.
      1. Navigate to **Azure Portal** > **Create a resource** > **Databases** > **SQL Managed Instance** or **SQL Database**.
      2. Configure the **server**, **storage**, **networking**, and **security** settings for your instance.
2. **Ensure Connectivity**:
   1. Set up a **VPN** or **ExpressRoute** between your on-premises environment and Azure to ensure secure communication.
   2. Ensure that **firewall rules** on the Azure side allow connections from your on-prem network.
3. **Install/Update SQL Server**:
   1. Ensure your on-premises **SQL Server** version is supported by DMS (typically SQL Server 2008 and above).

#### **2. Set Up Azure Database Migration Service (DMS)**

1. **Provision Azure DMS**:
   1. Go to the **Azure Portal** > **Create a resource** > **Databases** > **Azure Database Migration Service**.
   2. Provide the necessary details like name, subscription, resource group, and region, and then click **Create**.
2. **Create a Migration Project in DMS**:
   1. Navigate to **Azure Database Migration Service** > **Migration Projects** > **+ New Migration Project**.
   2. Select the project type: **SQL Server to Azure SQL Database** or **SQL Server to SQL Managed Instance**.
   3. Provide a name for the project and select **Online Migration** to minimize downtime.

#### **3. Set Up Source and Target Connections**

1. **Source (On-Prem SQL Server) Setup**:
   1. In **DMS**, configure the source by providing the details of your **on-premises SQL Server**:
      1. Server name, authentication details (SQL Server authentication or Windows authentication), and database name.
      2. Ensure that the **SQL Server instance** is accessible from Azure (via VPN or ExpressRoute).
2. **Target (Azure SQL Database/Managed Instance) Setup**:
   1. Provide the details of the target Azure SQL Database or SQL Managed Instance:
      1. Server name, database name, and authentication credentials.
      2. Ensure that your **Azure SQL** instance is reachable and properly configured for incoming data.

#### **4. Perform Initial Data Migration (Offline)**

1. **Migrate Schema and Data**:
   1. Start the **initial migration** process in DMS. This step will transfer your **schema**, **tables**, and **data** to Azure.
   2. During this process, your on-prem database will be **offline** for a short period while the schema and data are copied over.
   3. The **DMS** will copy the existing data from your on-prem database to the Azure SQL target.
2. **Monitor the Progress**:
   1. DMS will show you the progress of the migration, including any warnings or issues encountered.

#### **5. Enable Continuous Data Synchronization (Online Migration Mode)**

1. **Set Up Continuous Data Sync**:
   1. After the initial migration is complete, DMS will start replicating changes from your on-prem database to Azure continuously.
   2. **Online migration** will keep both databases in sync, ensuring that any new changes (inserts, updates, deletes) made to the on-prem SQL database are reflected in the Azure target database.
   3. This step allows your on-prem database to remain **online** and operational while the data sync happens in real time.
2. **Monitor the Sync**:
   1. Regularly monitor the sync process to ensure data changes are being applied correctly. The sync should occur in near real-time, with minimal lag.

#### **6. Finalize the Migration (Cutover)**

1. **Prepare for Cutover**:
   1. When you are ready to complete the migration, schedule a **cutover** window where the final sync will occur. This is typically done during a low-traffic period.
2. **Perform Final Sync**:
   1. DMS will perform one last **data sync** to ensure that any changes that happened after the initial migration are applied to the Azure target database.
   2. The **on-prem SQL Server** will be locked for changes for a very brief period during this final sync.
3. **Switch Applications to Azure**:
   1. Once the final sync is complete, **switch your applications** to point to the Azure SQL Database or SQL Managed Instance.
   2. During this cutover, there may be a brief **downtime** (usually just a few minutes) to ensure that all data is fully synchronized and the application starts using the new database.

#### **7. Post-Migration Tasks**

1. **Validate Data**:
   1. Run validation tests on the Azure SQL database to ensure all data has migrated successfully and that your application works as expected with the new database.
2. **Optimize Azure SQL Performance**:
   1. Once migration is complete, consider optimizing performance on Azure SQL by reviewing indexes, queries, and configurations.
   2. Ensure **automatic backups** and **monitoring** are enabled on the Azure SQL database.
3. **Decommission On-Prem Database**:
   1. After confirming that everything is functioning properly in Azure, you can choose to decommission your on-premises SQL Server, or you may choose to keep it as a backup for a period of time.