Q1:

Steps to Set Up Virtual Machines on Azure:

1. Log in to Azure Portal
2. Create a Windows VM:
    a. Select "Create a resource".
    b. Select "Virtual Machine" under the Compute category.
    c. In the "Create a virtual machine" wizard, choose Windows as the OS.
        i. Select a Region
        ii. Choose the VM Image (OS versions)
        iii. VM Size(memory size, storage)
        iv. Set up account
        v. Networking(public IP,vpn)
        vi. click Create.
3. Create a Linux VM:
    a. Repeat the above steps, but this time choose Linux as the OS in the Create a virtual machine wizard.
        i. Choose a Linux Distribution:
        ii. VM Size: Choose an appropriate VM size.
        iii. set up account.
        iv. Networking
        v. click Create.
4. Access the Virtual Machines:
    a. Windows VM: You can connect using Remote Desktop Protocol (RDP).
    b. Linux VM: You will connect using SSH.

Considerations for Pricing (OS-specific):

1. Windows VM Pricing:

- Licensing Costs: Windows VMs in Azure are typically more expensive because of the licensing fees. Azure provides both pay-as-you-go and Azure Hybrid Benefit options. Azure Hybrid Benefit allows you to use your existing Windows Server licenses (with Software Assurance) to reduce costs.
- VM Size & Type: Pricing varies based on the selected VM size and type (e.g., B-series for lower-cost workloads or D-series for high-performance workloads).

- Storage: The choice of storage type (Standard SSD, Premium SSD, etc.) also impacts the cost.
- Networking: Costs may arise from data transfer, public IP usage, and load balancing, if applicable.

2. Linux VM Pricing:

- No OS Licensing Fee: Linux VMs don't have the same licensing fees as Windows, so they are typically cheaper to run.
- Free Linux Distros: Some Linux distributions, like Ubuntu, CentOS, and Debian, are free to use. Red Hat and SUSE might have additional costs for support.
- VM Size & Type: Like Windows, the pricing for the VM size and storage will vary based on your choice of CPU, memory, and storage type.
- Storage: The cost of the disk (Standard or Premium) and data disks will apply, similar to Windows VMs.

Answer for 2 scenarios

Steps to Enable Encryption in Azure Storage:

1. Ensure Storage Service Encryption (SSE) is Enabled:
    a. By default, all Azure Storage accounts are encrypted using SSE. To confirm, follow these steps:
        i. Go to the Azure Portal.
        ii. Navigate to your Storage Account.
        iii. Under the Settings section, go to Encryption.
        iv. Check if Microsoft-managed keys or Customer-managed keys are being used.
2. Set Up Customer-Managed Keys (CMK):
    a. If you prefer to use customer-managed keys:
        i. Create or select an Azure Key Vault.
        ii. Create a Key in the Key Vault (or use an existing one).
        iii. Configure your Azure Storage Account to use this Key Vault key for encryption.
        iv. Enable SSE with CMK through the Azure Portal or Azure CLI.
3. Enable Azure Disk Encryption (ADE) for VMs:

    a. If using VMs and you want to encrypt the disks:
        i. Go to your Virtual Machine in the Azure Portal.
        ii. Under Settings, select Disks.
        iii. Enable Azure Disk Encryption and link it to your Key Vault.
4. Client-Side Encryption (CSE):
    a. If you choose to manage encryption on the client side:
        i. Use encryption libraries like AES or RSA in your application to encrypt data before uploading it to Azure Blob Storage.
        ii. Ensure that only the client has access to the decryption keys.

Answer for 3 scenarios

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.

Answer for 4 scenario

Step 1: Choose the Right Azure SQL Service

1. Azure SQL Database:
2. Azure SQL Managed Instance:

Step 2: Plan Your Migration Strategy

1. Assess Compatibility:

- Use SQL Server Migration Assistant (SSMA) to assess the compatibility of your on-premises SQL Server with Azure SQL Database or Azure SQL Managed Instance. This tool will help identify any potential issues during the migration.

2. Online Migration:

- Online Migration (Minimal Downtime): This approach allows you to keep your database accessible with minimal downtime during the migration. Azure Database Migration Service (DMS) supports online migration which minimizes downtime by continuously synchronizing changes to the database during the migration.

Step 3: Use Azure Database Migration Service (DMS)

Azure Database Migration Service (DMS) is the best tool to perform this migration with minimal downtime.

Steps for Online Migration with Azure DMS:

1. Create the Azure Database Migration Service:

2. Prepare the Source Database:
    a. On your on-premises SQL Server, ensure that SQL Server is version 2012 or higher and has TCP/IP enabled for remote connections.
    b. Open ports 1433 and any other required ports (SQL Server's default port).
    c. Backup your database to ensure you have a recovery point if anything goes wrong during migration.
3. Prepare the Target Azure SQL Database or Managed Instance:
    a. Set up your target Azure SQL Database or Managed Instance if not already done.
    b. Ensure the network connectivity between your on-premises SQL Server and the Azure target database. You may want to use a VPN or ExpressRoute for secure and fast communication between your on-premises environment and Azure.
4. Set Up Azure Database Migration Service:
    a. Go to the Azure Database Migration Service and select + New Migration.
    b. Choose Migration Type: Select Online Migration for minimal downtime.
    c. Source Database: Choose SQL Server and provide the necessary connection details for your on-premises SQL Server.
    d. Target Database: Select the target Azure SQL Database or Managed Instance and provide the connection details.
5. Start the Migration:
    a. Select the databases you want to migrate.
    b. Run the migration: DMS will start by copying the data from the source database to the target.
    c. Continuous Synchronization: DMS will continue to replicate changes from the source database to the target database, ensuring minimal downtime.
6. Cutover to Azure:
    a. Once the initial data is copied and changes are synced, perform a cutover to make the target Azure database the active database. During this cutover, there will be a brief period of downtime (typically minutes) where the final synchronization occurs, and the database becomes fully available in Azure.
    b. After cutover, the application will connect to the Azure SQL Database or Managed Instance instead of the on-premises SQL database.
7. Post-Migration Tasks:
    a. Test the application with the migrated database to ensure everything is working as expected.
    b. Monitor performance and check for any issues.
    c. Decommission the on-premises SQL Server if no longer needed.

Step 4: Minimize Downtime During the Migration

- Initial Data Load: The first phase involves a full backup and data load to the Azure database, which may take some time.
- Change Data Capture: DMS continuously captures changes in the on-premises database (inserts, updates, deletes) during the migration and applies those changes to the Azure target.
- Final Cutover: The final cutover happens when all changes are applied to the target database, and the application switches over to the Azure database. This cutover phase is usually brief, lasting only a few minutes, depending on the volume of changes.

Step 5: Monitor and Verify the Migration

- During migration, monitor the Azure Database Migration Service dashboard to ensure there are no issues with replication.
- Verify Application Connectivity: Ensure your applications are now pointing to the Azure SQL database or managed instance post-cutover.
- Test Performance and Functionality: Run tests to ensure the performance and functionality are as expected after the migration.