**CLOUD COMPUTING**

* Delivery of computing services
* Common IT infrastructure E.g. virtual machines, storage, databases and networking.
* Also include ML, AI, and IoT.
* Uses internet to deliver services.
* To increase or outgrow IT infrastructure we can use the datacenter already available at cloud we don’t need to built a new one from scratch.
* Not available offline but is online.
* Let’s us pay for the services only that us use.
* Multiple users can use it at one time and someone gets to monitor all the work being done.
* Basic services the cloud providers provide are storage and compute power.

Cloud service providers: Azure, AWS, Google

* **Compute power:** How much processing the computer can do.

**Storage:** Is the volume of data one can use on Computer.

**DEPLOYMENT MODELS**

* **Public Cloud:**

Cloud infrastructure is available to the public on the internet.

These are the cloud service providers.

* **Private Cloud:**

Exclusively operated by a single organization.

Can be managed by the organization or 3rd party.

* **Hybrid Cloud:**

Provides the services of public and private cloud.

**SERVICE MODELS**

* Are Hassle free.
* **IaaS:**

Refers to infrastructure as a service.

Users get access to basic computing structure.

Commonly used by IT administrators.

If we require storage and virtualization IaaS is the best option.

Just have to manage Application, Data, Runtime, Middleware and O/S.

* **PaaS:**

Platform as a Service

Provides cloud platforms in runtime for running, testing and managing applications.

Without the need for users to acquire manage and maintain related architecture.

If we require to build application models PaaS is the best option.

Only requires handling Application and data.

Microsoft Azure and AWS are PaaS.

* **SaaS:**

Software as a Service.

Includes services for hosting and managing Application software.

Software and hardware requirements are satisfied by the vendors so we don’t have to manage those.

If we don’t want to have hassles for owning any IT equipment SaaS is the best option.

Manages all the components solution required by the organization.

Two considerations for cloud deployment: availability and ability to handle demand.

* **High availability:**
  + High availability means that it ensures maximum availability, regardless of any disruptions or events.
  + Azure provides uptime guarantees depending on the service, which are a part of SLAs.
  + **SLAs:**
    - Standard industry term
    - Formal agreement between a service provider and a customer that ensures the service being provided is agreed upon.
    - Azure SLAs are % related to service or application’s availability.
    - When a service has 100% availability it means that the service had 100% uptime.
    - SLAs also includes the down time of a service and that assess if the credits one is entitled to are met.
    - Expensive to achieve 100% uptime:
      * Can’t update services or maintenance.
      * Will have to create more backups.
* **Scalability:**
  + The ability to adjust resources to meet demand.
  + Cloud is a consumption-based model: we pay only for what we use.
  + Comes in two variants: Vertical and Horizontal.
  + Vertical:
    - Focused on increasing or decreasing of resources.
    - Can add more processing power by adding more CPUs and RAMs when needed and can easily scale down if the needs are met.
  + Horizontal:
    - When there is sudden rise in demand of deployed resources more virtual machines, or containers can be added (scale out) and can be scaled in when there a drop in demand.

Cloud benefits that help develop solutions: Reliability and Predictability.

* **Reliability**
  + Ability to recover from failures.
  + Cloud has a decentralized design, if one region goes down all the other regions will still be up.
* **Predictability:**
  + Can be focused on Performance or Cost.
  + Azure’ well architected framework provides both performance and cost predictability.
* **Performance:**
  + Focuses on predicting the resources needed to deliver a positive experience.
  + Autoscaling, load balancing and high availability support performance predictability.
* **Cost:**
  + Focused on predicting or forecasting the cost of the cloud spent.
  + Real time resource tracking, monitoring resources, finding patterns and trends through data analytics to plan resource deployments.
  + Can use tools like Total Cost Ownership (TCO) or Price Calculator to get estimates of potential cloud spent.
* **Benefits of security and governance:**
  + Cloud computing offers features that facilitate governance, compliance, and security.
  + Templates ensure compliance with corporate standards and regulations, while auditing detects non-compliant resources.
  + Additionally, cloud providers offer security options, such as automatic updates and protection against DDoS attacks.
  + By establishing good governance practices, the cloud can be a secure and well-managed solution.
* **Management of the cloud:**
  + Cloud management involves efficiently managing cloud resources by automating resource scaling, deploying preconfigured templates, monitoring resource health, and receiving real-time performance alerts.
* **Management in the Cloud:**
  + Cloud management can be done through various methods, such as a web portal, command line interface, APIs, and PowerShell.

There are several major cloud providers, each offering a variety of products and services to cater to different business needs.

* **Amazon Web Services (AWS):**

AWS is one of the most popular cloud providers, offering a wide range of products, such as compute, storage, database, networking, security, and analytics services.

* + Elastic Compute Cloud (EC2): Allows users to rent virtual servers and run their applications on them.
  + Simple Storage Service (S3): Provides object storage for data backup and archiving.
  + Relational Database Service (RDS): Offers managed database services for various database engines.
  + Lambda: Allows users to run code without provisioning or managing servers.
* **Microsoft Azure:**

Azure is a cloud computing service from Microsoft that offers a range of products for computing, storage, analytics, and more.

* + Virtual Machines: Enables users to deploy and run virtual machines in the cloud.
  + Blob Storage: Provides object storage for unstructured data such as images, videos, and documents.
  + Azure SQL Database: Offers managed database services for SQL Server.
  + Azure Functions: Enables users to run event-driven serverless functions.
* **Google Cloud Platform (GCP):**

GCP offers a wide range of cloud products and services for compauting, storage, machine learning, and more.

* + Compute Engine: Enables users to run virtual machines in the cloud.
  + Cloud Storage: Provides object storage for unstructured data.
  + Cloud SQL: Offers managed database services for MySQL, PostgreSQL, and SQL Server.
  + Cloud Functions: Allows users to run serverless functions in response to events.

Other cloud providers, such as **IBM Cloud** and **Oracle Cloud**, also offer a range of cloud products and services for different business needs.

A trigger is an object that defines how an Azure Function is invoked. There are several types of triggers, including timer, HTTP, blob, queue, Cosmos DB, and event hub triggers. Each function must have exactly one trigger associated with it. A binding is a connection to data within user function and can be input or output. Bindings are optional and allow functions to receive and send data. In the lesson, we focus on timer, HTTP, and blob triggers and explore how to create a function on a schedule using a timer trigger.

* Timer Trigger
  + To execute a piece of logic at a set interval, we can use a timer trigger in Azure Function app with a Cron expression that sets the interval. A Cron expression is a string consisting of six fields that represent a set of times, and special characters like asterisk, comma, hyphen, and slash are used to define the interval. For example, a Cron expression of the 1st 2nd of every 5th minute of every hour of every day of every month means the logic will be executed every 5 minutes. we can create a timer trigger in the Azure portal and define the schedule using a Cron expression.
* HTTP Triggers
  + Azure Functions allow us to create logic that executes when an HTTP request is received through an HTTP trigger. The HTTP trigger has customizable options, including authorized access, supported HTTP verbs, receiving and returning data, and URL route templates. There are three Authorization levels: Function, Anonymous, and Admin, with different key-based authentication requirements. To create an HTTP trigger, we select it from the list of predefined triggers, enter the logic to execute, and customize the settings. To invoke the trigger, we send an HTTP request to the function URL, which can be obtained from the code page.
* Blob Creation
  + Azure Blob storage is a cloud-based object storage solution that stores large amounts of unstructured data. A blob trigger is a feature of Azure Functions that executes a function whenever a file is uploaded or updated in Azure Blob storage. To create a blob trigger, we specify the location that the trigger should monitor, and we can use filters to trigger the function only for specific types of files. The blob trigger passes the name of the file to the function as a parameter, which enables us to perform custom logic on the file.

Durable Functions, which is an extension of Azure Functions that enables us to perform long-lasting stateful operations in Microsoft Azure. Durable Functions can be used to orchestrate a long-running workflow, allowing us to automate processes that involve humans. The solution we implement needs to be cost-effective, flexible enough to run proprietary code, and able to orchestrate activities of varying duration. By the end of the lesson, we will design and implement a long-running approval process workflow using Durable Functions.

Durable Functions are an extension of Azure Functions that allow for the implementation of complex, long-running workflows in a cost-effective and scalable manner. They can retain state between function calls and offer benefits such as event-driven coding, function chaining, fan out/fan in, and human interaction. Durable Functions consist of three types: client, orchestrator, and activity functions. They can be used to implement common workflow patterns such as function chaining, fan out/fan in, async HTTP APIs, monitoring, and human interaction. An approval process is an example of a workflow that involves human interaction.

the use of durable timers in orchestrator functions to control long-running tasks and implement delays or timeouts. It is important to consider scenarios where tasks may take longer than expected or require escalation paths. Durable timers can be created using the create timer method and can be used to implement reminders or set up timeouts. Examples are given for using durable timers to send reminders and set up a timeout path if a task isn't completed within a certain timeframe. The full source code for these examples is available for reference.

Webhooks are user-defined HTTP callbacks that are triggered by events such as updating a Wiki page or pushing code to a repository. Azure Functions can be used to define logic that is run when a webhook message is received. Azure Functions is a serverless compute service that enables running code without having to manage infrastructure. Triggers cause a function to run, and bindings are used to connect resources to a function. Bindings are optional and a function may have one or more input and/or output bindings.

Setting up a webhook for a GitHub repository involves two steps. First, specify how the webhook should behave and what events it should listen to in GitHub's settings page for the repository. Second, set up a function in Azure Functions to receive and manage the payload received from the webhook. Webhooks require configuration options such as the payload URL and content type. Events are the core of webhooks and occur whenever actions are taken in the repository. A full list of webhook events and when they can run can be found in the webhook events reference. To set up a webhook for a repository, go to the repository's settings page in the GitHub portal, select "Webhooks", and then select "Add webhook".

To secure the payload received from GitHub, we can set up a secret token and validate the request using this token. Setting a webhook secret allows us to ensure that post requests sent to the payload URL are from GitHub. When usr secret token is set, GitHub uses it to create a hash signature for each payload, which is passed along with each request in the headers as x-hub-signature. To ensure that the payload is from GitHub, us need to compute the hash using usr secret and ensure that it matches the hash in the request header. GitHub uses a HMAC SHA1 hex digest to compute the hash, so we must calculate usr hash in the same way, using the key of usr secret and usr payload body. The hash signature starts with the text, sha1=.

The current architecture of the stock information application uses a polling-based design where the client fetches changes from the server based on a timer. The server stores stock information in an Azure Cosmos DB database, and the client uses vue.js and Axios HTTP client to handle requests to the function. However, the polling mechanism is inefficient as the client contacts the server whether or not changes exist, resulting in wasted resources and server strain. Choosing the best polling interval for the scenario is also challenging, and there can be delays in detecting new data. As the application scales, the amount of data exchange between the client and server becomes a problem.

In this text, we learn about a new design for updating data in an app using Azure Cosmos DB, Azure Functions, and SignalR. This design reduces traffic and makes the UI more efficient by only updating as data changes. Azure Cosmos DB listens to a database container for changes and outputs the sorted list of change documents. Azure Functions runs code anytime data is updated in an Azure Cosmos DB change feed. SignalR allows two-way communication between the client and server with persistent connections, transport fallbacks, and the ability to broadcast messages to all connected clients or specific clients. The abstraction layer provided by SignalR offers future-proofing and graceful degradation depending on the client's supported technologies. The text also shows how to use SignalR to broadcast information from a function that reads the Azure Cosmos DB change feed.

To publish the application to the Cloud, we need to deploy the local functions into Azure and make the static HTML and JavaScript files available on the web. Azure Storage provides a feature called static website support that enables hosting publicly available web pages by placing files in a specific storage container named Dollar web. The files in this container become available to web browsers through a secure server using a specific URI scheme that includes the account name, zone name, and the file name.

Azure Functions is a serverless service in Azure that allows us to write code without worrying about supporting infrastructure, and we are charged only for the time usr code runs. Azure API Management is a fully managed Cloud service that helps organizations publish, secure, transform, maintain, and monitor APIs. By publishing Azure Functions through API Management, we can implement a microservices architecture and build those microservices into an integrated distributed application. The consumption tier in API Management is especially suited for microservices-based architectures and event-driven systems.

Azure API Management is a fully managed cloud service that enables us to publish, secure, transform, maintain, and monitor APIs. It is useful in a serverless architecture because it allows us to build single APIs from individual microservices. Azure Functions, a serverless compute service, allows us to write functions in multiple languages without worrying about the underlying infrastructure. We can publish Azure Functions through API Management to implement a microservices architecture, where each function implements a microservice. The API Management Consumption Tier is well-suited for microservice-based architectures and event-driven systems because it employs an entirely different architecture based on shared, dynamically allocated resources, which aligns perfectly with serverless computing models.  
  
  
Microservices architecture is a modular approach to software development that allows for independent deployment of services, making continuous delivery easier. However, it can present challenges such as inconsistent APIs and security implementation. API management can address these challenges by acting as an intermediary between client apps and microservices, enforcing consistent rules and security requirements, and providing helpful tools for testing and monitoring. Azure API Management supports importing Azure functions apps as new APIs or appending them to existing ones.

Public cloud is the most common deployment model, where cloud resources are owned and operated by a third-party service provider and accessed over the internet. It offers scalability, cost-efficiency, and global accessibility. Public cloud is suitable for solutions that require rapid scalability, global reach, and cost-effective resource consumption.

Private cloud consists of computing resources exclusively used by a single organization. It can be located on-premises or hosted by a third-party provider. Private cloud provides greater control, security, and customization options. It is suitable for scenarios involving existing environments that can't be replicated in the public cloud, legacy applications, data sovereignty and security requirements, and regulatory compliance.

Hybrid cloud combines public and private cloud environments, allowing data and applications to be shared between them. It provides flexibility, scalability, and cost-efficiency. Hybrid cloud is suitable when there's a need to keep business-critical applications and data on-premises for security or compliance reasons while leveraging the scalability and resources of the public cloud for non-sensitive tasks. It also facilitates gradual migration to the cloud.

Ultimately, the choice of deployment model depends on factors such as the specific requirements of the healthcare company's systems, data security and compliance needs, scalability requirements, existing infrastructure investments, and long-term goals. It's important to evaluate each deployment model's advantages and align them with the organization's needs to make the best decision for each solution.  
  
Infrastructure as a Service (IaaS) is a cloud computing model that provides instant and scalable computing infrastructure over the Internet. It eliminates the need to purchase and manage physical servers and data center infrastructure, allowing businesses to quickly meet demand and pay only for what they use. Users can provision virtual machines, storage, firewalls, and other components as separate services, while being responsible for their configuration and management. IaaS offers flexibility for various scenarios, such as running specialized software, creating development environments, hosting websites, managing storage, enabling high-performance computing, and analyzing big data. It eliminates capital expenses, improves business continuity, enables quick response to changing conditions, and enhances stability and reliability. With IaaS, businesses can rapidly deploy infrastructure, reduce costs, and focus on developing and delivering applications to users faster.

Platform as a Service (PaaS) is a cloud-based development and deployment environment that offers a range of benefits. With PaaS, developers can create and deploy applications without worrying about infrastructure details such as servers or operating systems. It provides middleware, development tools, and other services, making it easier to build and customize cloud-based applications. PaaS also offers advantages for data analysis and business intelligence, allowing organizations to uncover insights and make informed decisions. The development tools and pre-coded components provided by PaaS reduce development time and expand the capabilities of development teams. The pay-as-us-go model makes sophisticated software and tools accessible, while the cloud-based nature enables remote collaboration. Overall, PaaS supports the complete web application lifecycle, from building to updating, within an integrated environment.

Software as a Service (SaaS) is a cloud-based model that allows users to access and utilize applications over the internet. Examples include email, calendars, and productivity tools like Microsoft 365. With SaaS, users connect to the service provider's data center through a web browser, and all the underlying infrastructure, software, and data are managed by the provider. SaaS offers organizations a cost-effective solution as they can rent applications instead of purchasing and maintaining the necessary hardware and software. It provides flexibility, scalability, and accessibility as users can access their information from any internet-connected device. Additionally, data is securely stored in the cloud, eliminating the risk of data loss due to device failures.

The Azure CLI is a cross-platform command line program that enables administrators and developers to manage Azure resources using command line prompts instead of a web browser. It can be installed on Linux, macOS, and Windows machines and supports interactive usage as well as scripting for automation. The installation process differs based on the operating system, with Linux and macOS utilizing package managers and Windows requiring the download and execution of an MSI file. Users should be mindful of shell or environment differences when running scripts, such as variable declaration variances between bash and PowerShell. Once installed, the Azure CLI provides a consistent set of commands for managing Azure resources across platforms.

To install the Azure CLI on a Windows machine, we can follow these steps: First, go to the official Azure CLI installation page for Windows. Then, click on the provided link to download the MSI installer. Once the installer is downloaded, run the file and accept the license terms. In the User Account Control dialog, select "Yes" to proceed with the installation. After the installation is complete, we can run the Azure CLI by opening a command prompt or PowerShell and entering the command "az --version" to verify the installation and display the version of Azure CLI installed. By setting up the Azure CLI on usr local machine, we gain the ability to administer Azure resources using commands or scripts locally, with the CLI forwarding usr instructions to the Azure datacenters where they will be executed within usr Azure subscription.

The Azure CLI is a powerful tool that allows us to control various aspects of Azure resources. To get started, we need to authenticate using the Azure CLI login command, which opens the Azure sign-in page in usr default browser. Once connected to usr Azure subscription, we can create a new resource group using the Azure CLI group create command, providing a unique name and location for the group. After creating the resource group, we can use the Azure CLI group list command to verify its successful creation. The Azure CLI provides a wide range of commands for managing different Azure services, and we can use the AZ find command to search for specific commands or get more detailed information using the --help argument. Overall, these steps allow us to connect to usr Azure subscription, create resources, and ensure successful deployment using the Azure CLI.

Used the Azure CLI to create a resource group and deploy a web app. First, we create variables for the resource group name, Azure region, app plan name, and web app name. Then, we create an app service plan with the free pricing tier and verify its successful creation. Next, us create the web app within the service plan. Once the web app is created, we can access its default URL. Finally, we deploy code from a GitHub repository to the web app, and after the deployment is complete, we can see the deployed page displaying "Hello World!".

In order to administer the Azure resources for testing a customer relationship management system, it is necessary to choose the right tool. The three options provided by Azure are the Azure portal, Azure CLI, and Azure PowerShell. While all three offer similar control, they differ in syntax, setup requirements, and automation support. The Azure portal is a graphical user interface that is easy to use but lacks automation capabilities. The Azure CLI is a cross-platform command-line program that allows interactive or scripted use, requiring knowledge of command syntax. Azure PowerShell is a module added to PowerShell, suitable for those with existing PowerShell expertise, offering both interactive and scripting modes. When deciding between these tools, factors such as automation requirements, team skills, and learning curve need to be considered. In a specific scenario where resource groups and multiple virtual machines need to be created, the Azure portal is suitable for one-time resource group creation, while automation for VM creation can be achieved with either Azure PowerShell or the Azure CLI. Azure PowerShell is recommended if the team has prior PowerShell experience, as it supports automation and has a quick learning curve. Ultimately, a combination of the portal and scripting solutions is often used for a mix of one-off and repetitive tasks.

The provided steps guide users on checking the version of PowerShell installed on their local Windows machines and installing the latest version if necessary. Users are instructed to search for PowerShell in the System tray, select the appropriate shortcut based on their preference, and then use the command "$PSVersionTable.PSVersion" or "pwsh -ver" to determine the installed version. If the major version number is lower than 7, users are advised to upgrade their existing Windows PowerShell by installing the .NET SDK and running the command "dotnet tool install --global PowerShell". Finally, users are encouraged to verify their installation by running the PowerShell version command again. These steps ensure that users have an updated and compatible version of PowerShell, which is essential for utilizing Azure Az PowerShell module and .NET tools effectively.

PowerShell cmdlets are commands used in PowerShell to perform specific actions. They follow a verb-noun naming convention, such as Get-Process or Start-Service. Cmdlets are designed to be simple and single-purpose, manipulating a specific feature or functionality.

Cmdlets are organized into modules, which are DLL files that contain the code to process each cmdlet. We can load modules into PowerShell using the Import-Module cmdlet. Modules can be added to PowerShell to expand its functionality and work with different features. For example, the Az module is the Azure PowerShell module that contains cmdlets to work with Azure features.

To install the Az module, we can use the Install-Module command in an elevated PowerShell shell. The Az module is available from the PowerShell Gallery, a global repository for PowerShell modules.

These steps demonstrate how to create a Linux virtual machine (VM) in Azure using Azure PowerShell. The process involves using the New-AzVm cmdlet to create the VM, specifying the resource group, VM name, location, image, credentials, and open ports. After the VM is created, we can query its information using the Get-AzVM cmdlet and access specific properties of the VM object. To delete the VM, we can use the Stop-AzVM and Remove-AzVM cmdlets, followed by manually deleting associated resources like network interfaces, disks, virtual networks, network security groups, and public IP addresses. Writing a script to automate these tasks would be a more efficient approach.

The provided script is a PowerShell script that automates the creation of virtual machines (VMs) in Azure. It takes a resource group name as a parameter and prompts for a username and password for the VM's admin account. The script then creates three VMs using a loop, generating unique names for each VM and displaying the progress on the console. This automation saves time and effort compared to manually creating the VMs through the Azure portal, demonstrating the power of scripting in streamlining resource provisioning.

The Total Cost of Ownership (TCO) calculator offered by Azure helps Tailwind Traders assess the cost savings of migrating from their on-premises data center to the cloud. The three-step process begins with defining the workloads, including service, databases, storage, and networking specifications. In the second step, assumptions about operating costs are adjusted, considering factors like software assurance, storage replication, and various cost parameters provided by Nucleus Research. The final step involves generating a report based on the entered information, allowing a side-by-side comparison of costs for on-premises versus Azure operations. The report can be downloaded, shared, or saved for future reference, providing Tailwind Traders with a comprehensive understanding of their total costs.

In this exercise, we use the Total Cost of Ownership (TCO) Calculator to compare the cost of running a sample workload in Tailwind Traders' European datacenter versus on Azure. We define the workloads by specifying the specifications of the on-premises infrastructure, including 50 Windows Server VMs running under Hyper-V virtualization and 50 Linux VMs running under VMware virtualization. We also consider the storage area network (SAN) with 60 TB of disk storage and estimated outbound network bandwidth of 15 TB per month. After adjusting the assumptions and selecting the currency, we view the generated report, which shows a comparison of running the workloads in the datacenter versus on Azure. The report estimates significant cost savings of over $15 million over three years when running on Azure.

Azure offers various subscription models, including free trial, pay-as-us-go, and member offers, to cater to different needs. A free trial subscription provides 12 months of free services and a credit to explore Azure for 30 days, while a pay-as-us-go subscription allows us to pay for what we use. Enterprise customers can sign an agreement with Microsoft for customized pricing over a three-year period, while web direct allows purchasing services directly from the Azure Portal at standard prices. Cloud Solution Providers (CSPs) are Microsoft partners who help build solutions on Azure, billing for usage at their determined price. The Azure portal facilitates resource provisioning and provides cost management and billing information.

The cost of usr Azure implementation depends on various factors such as the type and customization of resources, usage meters that track resource usage, Azure subscription types with usage allowances, and the use of third-party vendors through Azure Marketplace. Different resource types have different costs based on factors like storage type, performance tier, and access tier. Meters track specific usage types and correlate to billable units that are charged based on resource type rates. Minimizing costs can be achieved through actions like de-allocating VMs when not in use. Choosing the right Azure region is crucial as prices vary, and network traffic between regions can incur additional costs. Billing zones and bandwidth usage also affect pricing. The Azure pricing calculator helps estimate costs by configuring resources and providing a detailed breakdown of associated costs.

we used the Azure Pricing calculator to estimate the cost of running a basic web application for Tailwind Traders on Azure. The application consists of two virtual machines connected through an Application Gateway for load balancing, and it connects to an Azure SQL Database for inventory and pricing information. We configured the virtual machines to be similar to the ones in the on-premises datacenter, set up the Azure SQL Database with the required specifications, and configured the Application Gateway with the appropriate size and data processing volume. After inputting these details, the Pricing calculator provided us with an estimated monthly cost of $1,902.64. This estimate can be saved, exported, or shared with the team for further analysis and decision-making.

The recommended practices for minimizing costs in Azure for Tailwind Traders, a home improvement retailer, include understanding estimated costs before deployment, carefully selecting necessary products and resources, utilizing Azure Advisor to identify and optimize resource usage, implementing spending limits and quotas to prevent overspending, taking advantage of Azure reservations for discounted prices, leveraging cost-effective locations and regions, using Azure Cost Management and Billing for monitoring and optimization, resizing or shutting down underutilized VMs, automating VM start and stop schedules, shutting down unused systems, gradually moving from IaaS to PaaS services, considering licensing options, and choosing cost-effective operating systems. These practices help Tailwind Traders optimize their Azure usage and reduce unnecessary expenses.

In this Azure tutorial, we learned about resource groups and their role in organizing and managing Azure resources. Resource groups are logical containers for resources deployed on Azure, providing a way to group related resources together. They offer benefits such as a consistent naming convention, the ability to manage the life cycle of resources, authorization through RBAC permissions, and organization for billing and reporting purposes. We also learned how to create a resource group using the Azure portal and explored adding resources to the newly created resource group. Overall, resource groups are a powerful tool for maintaining order and structure in usr Azure environment.

Learned about the importance of using tags to organize and manage Azure resources effectively. we discovered that tags are name-value pairs that can be applied to resources and resource groups, allowing us to associate custom details and properties with usr resources. By adding tags to usr resources, such as virtual networks, and using filters, we can easily search and categorize them based on departments, environments, or other criteria. Tags can be added through various methods, including the Azure portal, Azure CLI, PowerShell, and REST API. Additionally, we explored the benefits of using tags for billing, monitoring, and automation purposes, such as grouping usage by cost center, identifying impacted resources during alerts, and automating actions based on specific tags. By leveraging tags effectively, we can enhance resource organization and streamline management within usr Azure environment.

Learned how to use Azure policies to enforce standards and ensure compliance in our Azure environment. We start by creating a policy definition that enforces the existence of a specific tag on resources. We then assign this policy to our core infrastructure resource group, so it applies to all resources within that group. Once assigned, any attempts to create a resource without the required tag will fail. We test the policy by creating a storage account without the tag and see that it fails validation. We then add the required tag and successfully create the storage account. Azure policies can be used to enforce naming conventions, restrict resource types and regions, and maintain consistency across Azure resources.

To ensure the protection of deployed Azure resources, we can use role-based access control (RBAC) alongside Azure Policy. While Azure Policy enforces internal standards during resource creation, RBAC allows us to manage access permissions for different user roles. With RBAC, we can assign specific rights to individuals based on their job requirements, such as IT personnel, developers, and administrators. This fine-grained access management ensures that users have the necessary privileges without granting unrestricted permissions. By segregating duties and granting the lowest privilege level needed, we can enhance security. Additionally, resource locks can be employed to prevent critical resource modification or deletion. These steps collectively provide comprehensive protection for our Azure resources while adhering to best practices.

In order to prevent accidental deletions of critical Azure resources, I implemented resource locks as a safeguard. Resource locks are settings that can be applied to resources to block modification or deletion. I applied a resource lock called "BlockDeletion" to the msftlearn-core-infrastructure-rg resource group, which contains important virtual networks and a storage account. This lock prevents the resource group and its contained resources from being deleted. When I attempted to delete one of the virtual networks, I received an error message stating that a lock was preventing its deletion. By using resource locks, I ensured that key components of our Azure environment are protected from inadvertent actions and provided an additional layer of security.

To better organize our resources, we will be separating the development resources from the production resources in our resource group. This will involve creating a dedicated resource group for development. By doing this, we ensure that resources with different lifecycles are appropriately grouped and can be managed together. To identify the development resources that need to be moved, we will navigate to the production resource group in the Azure portal and review the resources it contains. We can search, filter, and view the list of resources in the resource group, making it easier to identify the ones that don't belong. Additionally, applying tags to resources can help with organization and searchability. Once we have identified the development resources, we can proceed to move them into their own resource group, ensuring a more streamlined and efficient resource management process.

To better organize our resources, we will be separating the development resources from the production resources in our resource group. This will involve creating a dedicated resource group for development. By doing this, we ensure that resources with different lifecycles are appropriately grouped and can be managed together. To identify the development resources that need to be moved, we will navigate to the production resource group in the Azure portal and review the resources it contains. We can search, filter, and view the list of resources in the resource group, making it easier to identify the ones that don't belong. Additionally, applying tags to resources can help with organization and searchability. Once we have identified the development resources, we can proceed to move them into their own resource group, ensuring a more streamlined and efficient resource management process.

In order to assess the feasibility of moving development resources to a dedicated resource group, we need to follow a series of steps. Firstly, identify the resource types of the resources us want to move in the Azure portal. This can be done by checking the "Type" column on the all resources page or in the list of resources in the resource group. Once we have identified the resource types, we should consult the move support for resources list to determine if those resources can be moved and if there are any restrictions in place. This list provides information on whether each resource type can be moved between resource groups or between subscriptions. It is important to note any limitations specific to certain resource types, such as the inability to move 3rd party SSL certificates when moving a web app or the dependencies that must accompany a virtual machine. By considering these limitations, we can decide which resources are suitable candidates for the move.

To validate a move of resources using the Azure REST API, we can follow a simple process. First, send a POST request with the necessary authorization and Content-Type details, including the resource and target resource group information. The API will return a status code of 202 if usr request is accepted. Next, wait for the specified amount of time indicated in the response and retry after value before proceeding. Then, send a GET request to the provided location URL. If usr move is validated as successful, we will receive a 204 status code. Otherwise, we will receive an error message indicating that the move won't be successful. This validation process ensures that we can test and verify usr move operations before executing them, without affecting usr resources.

To move resources between resource groups in Azure, us need to follow a few steps. First, identify the resources we want to move and ensure they can be moved. Then, create a new resource group to hold the resources. we can use the Azure portal, Azure CLI, PowerShell, or Azure REST API for this process. Once the new resource group is created, use the appropriate command (az resource move for Azure CLI or Move-AzResource for PowerShell) along with the resource ID to move the resources to the new group. After the move, verify the resources have been successfully moved by using the appropriate command (az resource list for Azure CLI or Get-AzResource for PowerShell). Remember to update the resource IDs and any references in usr tools and scripts to reflect the new resource group.

Created a dedicated resource group called "devgroup" for our development resources. We moved our development resources from the production resource group to the newly created devgroup. We started by creating the devgroup resource group, specifying the necessary details such as subscription and region. Then, we assigned the "environment:development" tag to the resource group. Next, we selected the storage account we wanted to move and used the "Move to another resource group" option. We chose devgroup as the target resource group and validated the resources for the move. After confirming the move, we waited for the notification indicating the successful move. Finally, we verified the move by checking the devgroup resource group, where we could see that the development resources had been successfully moved.

To connect usr Java applications to Azure, we can choose between Eclipse and IntelliJ IDEA. For Eclipse, the installation process is straightforward using the Eclipse installer, and we can install the Azure Toolkit for Eclipse from the Eclipse marketplace. Similarly, for IntelliJ IDEA, we can use the IntelliJ IDEA installer and install the Azure Toolkit for IntelliJ from the IntelliJ IDEA marketplace. Visual Studio Code, a lightweight and free source code editor, is another option that supports Azure Cloud Services. We can install Visual Studio Code from code.visualstudio.com and enhance its capabilities with extensions from the Visual Studio Code marketplace, many of which target Azure features. Finally, Visual Studio is a full-featured IDE with extensive Azure development tools. We can install Visual Studio by downloading the installer and selecting the Azure Development workload. Additional Azure functionality can be added through NuGet packages or Visual Studio extensions. Visual Studio for Mac is available for macOS and offers contextual integration of Azure tooling, with the option to add specific Azure features through NuGet packages or Visual Studio for Mac extensions.

To create an Azure App Service web app using the Azure portal, we have to follow these steps: First, access the Azure portal, which provides a user-friendly graphical interface. Next, click on the option to create a web app. This will allocate hosting resources in App Service. Enter the required information, such as a valid Azure subscription, a resource group, and a unique name for usr web app. Choose whether to deploy application as code or as a Docker image, and select the appropriate runtime stack. Specify the operating system (Windows or Linux) and the Azure region for usr app. Choose an App Service plan, which determines the virtual server resources and performance characteristics for running usr app. We can create a new plan or select an existing one. Finally, enable features like Application Insights for monitoring and configure any additional settings.

To create a web app on Azure, sign in to the Azure portal using usr account. From the portal menu, select "Create a resource" and then choose "Web App" from the options. In the "Basics" tab, provide the necessary details such as subscription, resource group, web app name, publishing method (code), runtime stack (Node 16 LTS), operating system (Linux), region (East US), and pricing plan (F1). Review the settings and select "Create" to initiate the deployment process. Once the deployment is complete, We can access the App Service Overview pane for usr web app. To preview the default content of usr web app, click the URL under "Default domain" and observe the placeholder page that confirms usr app is up and running.

In order to prepare the code for deploying a web application, there are several approaches depending on the programming language. For C#, the dotnet command line tool can be used to create a new ASP.NET Core MVC application. Java developers can utilize Maven's archetypes feature to quickly generate starter code for a web app. Node.js provides the npm command line tool, which enables the creation of a new application and the definition of how it should be run in a package.json file. Python developers can use the Flask framework to create a minimal web application. Once the code is ready, it's recommended to integrate it into a source control repository like Git. This allows for versioning and collaboration. With Git initialized, the code can be committed and later synchronized with a remote repository, such as GitHub, enabling continuous integration and deployment for more efficient and reliable updates to the application.

We created a basic Node.js web application using Azure Cloud Shell. We started by creating a package.json file that describes our application. Then, we created an index.js file that contains a simple Node.js program. This program sets up an HTTP server that responds with "Hello World!" for any GET request. We made some edits to the package.json file to specify the start script as "node index.js". Finally, we tested our web app locally by starting it with the "npm start" command and accessing it through a browser using the curl command. This project serves as a starting point for building more complex web applications with Node.js.

To deploy an application to App Service, we have the option of automated deployment or manual deployment. For automated deployment, Azure provides support for direct deployment from Azure DevOps, GitHub, Bitbucket, OneDrive, and Dropbox. This allows to push code to these sources and have it automatically built, tested, and deployed to an Azure Web App. On the other hand, for manual deployment, we can use Git by adding the App Service web app's Git URL as a remote repository and pushing code to it. Another option is to use the az webapp up command-line feature, which packages and deploys the app, creating a new App Service web app if needed. Additionally, we can deploy application by sending a ZIP of our application files using az webapp deployment source config-zip or deploy Java web applications using WAR packages through the Kudu HTTP API. Visual Studio also offers a deployment wizard for App Service, and traditional FTP/S can be used as well. These options provide flexibility in deploying our application to App Service based on our preferences and workflow.

To deploy a Node.js application using the Azure CLI, we first gather the necessary information about our web app resource using Azure commands. We set variables for the app name, resource group name, plan name, SKU, and location. Then, we run the "az webapp up" command with the provided values to package and deploy the application to our App Service instance. After a few minutes, the deployment is completed, and we can verify it by clicking the link provided in the output. The deployed application should load in a new browser tab, displaying the greeting message from our app. Overall, these steps allow for a streamlined and efficient deployment process.

To prepare usr development environment for creating and deploying ASP.NET Core web applications, We need to install the necessary tools on usr local machine. Visual Studio 2019 is the recommended IDE, and we can download it for free from the Microsoft website. Once installed, us should ensure that the ASP.NET and web development workload, as well as the Azure development workload, are selected during the installation process. These workloads provide the templates and tools needed for building and deploying usr website to Azure. The Visual Studio Installer allows us to modify and customize usr installation, including adding or removing workloads. Once the necessary components are installed, we'll have the capabilities to develop web applications using ASP.NET and standards-based technologies, as well as the ability to connect and deploy usr site to Azure.

To deploy usr newly created website to Azure using Visual Studio, us need to consider leveraging Azure App Service. Azure App Service provides a scalable web hosting service for various applications, supporting different programming languages. The App Service plan determines the compute resources usr app will consume, including the location, number, and size of virtual machine instances. It also defines the pricing tier, ranging from free and shared tiers for small-scale projects to dedicated and isolated tiers for production workloads with high traffic demands. When creating an App Service plan, we choose a region close to usr target audience. It's important to ensure that the plan has sufficient resources to avoid performance issues or downtime for both new and existing apps. We can scale the plan up or down as needed and manage resources within a resource group, which serves as a logical container for organizing and managing Azure resources.

we learned about the project structure of an ASP.NET Core web app created in Visual Studio. The tutorial covered various folders and files, including Dependencies, Properties, Publish Profiles, wwwroot, and Pages. We also explored the concept of Razor templates, which allow us to create dynamic web pages using C# code. Additionally, we saw how to make basic changes to the website by leveraging Razor templates and understanding Razor syntax. Overall, the tutorial provided a comprehensive overview of the project structure and customization options for the web app.

Deployment slots in Azure App Service offer a convenient solution for testing and deploying web apps with minimal downtime. By creating multiple slots within a single web app, each with its own hostname, developers can efficiently test new app versions while the production slot continues to host the stable version. These slots allow for comprehensive testing through integration, acceptance, and capacity tests. Once satisfied with the results, the new version can be seamlessly deployed by swapping it with the production slot, instantly directing production traffic. If any issues arise, the slots can be easily swapped back to the previous version for a quick rollback. This approach ensures thorough testing and a smooth deployment process, without exposing partially deployed versions to the public web. Deployment slots share the resources of the app service plan and are available in different tiers to suit varying requirements.

In this exercise, we set up a web app in Azure and configured deployment slots to streamline the deployment of new versions of usr social media web app. we started by creating a web app resource with the necessary settings such as name, runtime stack, operating system, and region. Then, us configured git deployment by setting up a local git repository and cloning a sample web app source code. Next, we configured a git remote to deploy the app to the production slot and deployed the web app using git. After that, we created a new staging slot and set up git deployment for it. We added a remote for the staging slot in the git configuration. Finally, we modified the app source code, committed the changes, and deployed the new version to the staging slot. By following these steps, we were able to create and configure deployment slots for usr web app, allowing us to test and deploy new versions without any downtime.

To smoothly deploy a new version of usr social media web app from the staging slot to the production slot, we can configure the app's settings and slot settings. By setting the appropriate slot settings, such as configuring the database connection string, we can ensure that each slot uses the correct configuration. To view and configure these settings, go to the web app resource, select the desired deployment slot, and modify the settings accordingly. Once the settings are in place, we can swap the slots by navigating to the Deployment slots page and selecting "Swap." During the swap, the settings from the target slot are applied to the source slot's app version. Azure App Service also offers a swap with preview feature, allowing us to test the app in the source slot with the target slot's configuration before completing the swap. If we prefer automated deployments, we can enable auto swap for a slot, ensuring zero downtime when pushing code or content changes. Remember to thoroughly test usr web app in the staging slot and resolve any issues before initiating the swap or enabling auto swap for a smoother deployment process.

In these steps, we learn how to configure slot settings and manually swap slots for deploying different versions of a web app. First, we configure slot settings for the production and staging slots, specifying environment names and app versions. Then, we perform a manual swap, exchanging the staging and production slots, which deploys version 2 of the web app to production. Next, we enable auto swap for the staging slot, allowing continuous deployment by automatically swapping the slots whenever new code is deployed to staging. Finally, we modify the code to create version 3 of the web app, deploy it to the staging slot, and observe the auto swap in action. If any issues arise, we can easily roll back to the previous version by swapping the slots again.

In the Azure environment, manual scaling of a web app can be achieved through Azure App Service. The App Service plan defines the resources and availability for the web app. The available tiers range from Free to Isolated, with increasing capacity and cost. To scale out, additional instances can be added to the App Service plan, improving availability and accommodating expected increases in traffic. Monitoring performance metrics such as CPU utilization and response times helps determine when scaling is necessary. When the system is lightly loaded, scaling back in can reduce costs. By following these steps, web apps can be efficiently managed and optimized for performance and cost-effectiveness in Azure.

In this exercise, we created an App Service plan and deployed a web app using that plan. we monitored the performance of the web app under load and observed slow response times and HTTP 408 errors. Then, we scaled out the app by increasing the instance count to 5. After the scale-out, we ran the load test again and observed improved performance with reduced response times and fewer errors. we also had the option to further increase the instance count to 10 for additional performance gains. Finally, we scaled back down to a single instance. The exercise demonstrated the process of manual scaling to handle increased traffic and optimize the performance of a web app.

In order to meet the increasing demands and resource requirements of our hotel reservation system, we need to scale up our web app. This involves changing the pricing tier and hardware level of our app service plan to provide more instances and more powerful hardware. Scaling up can be done manually and may cause a temporary interruption in service, requiring client apps to disconnect and reconnect. Additionally, the outgoing IP addresses for the web app may change, necessitating reconfiguration of other services with incoming traffic restrictions. It's important to monitor the system's performance to ensure the desired effect of scaling up or down, and scaling up may be necessary before further scaling out can be achieved.

Azure Container Registry is an Azure service that allows we to create private Docker registries in the cloud. It offers advantages over Docker Hub, such as increased security and control over image visibility and usage. Container Registry encrypts images at rest and provides the option to sign images for enhanced trust. It can be replicated to store images closer to deployment locations and offers high scalability for concurrent Docker pools. Creating a registry is straightforward using either the Azure portal or the Azure CLIAC command. In addition to storing and hosting images, Container Registry can also build images by uploading the Docker file and other necessary files, which it will then build for us. Overall, Azure Container Registry is a powerful tool for securely managing and deploying Docker images in Azure.

To create a registry in Azure Container Registry, sign in to the Azure portal with usr Azure subscription and navigate to the Create a resource section. Select Containers, then Container Registry. Provide the necessary details such as subscription, resource group, registry name, location, and SKU. Review the settings and create the registry. Next, use Azure Cloud Shell to download the source code for a sample web app and navigate to the source folder. Run the command to send the folder's contents to the Container Registry, which will build the Docker image based on the instructions in the Docker file. Once the build process is complete, examine the container registry in the Azure portal to verify the presence of the web app's Docker image. This image can now be deployed to App Service.

To enable Docker access to the Azure Container Registry, we need to configure the registry settings and create a web app. First, sign in to the Azure portal and navigate to the Container Registry we created. In the Access keys settings, enable the Admin user option. Next, create a web app by selecting "Create a resource" from the Azure services menu and choosing Web App. Fill in the necessary details such as subscription, resource group, name, and region. Select the Docker Container option for publishing, and choose Linux as the operating system. Specify the registry, image, and tag from the Azure Container Registry options. Review the settings, create the web app, and wait for the deployment to complete. Once deployed, we can test the web app by browsing to its URL and confirming that the app is running successfully.

In this set of steps, we configured continuous deployment for a web app in the Azure portal. By enabling continuous deployment, we created a webhook that Container Registry uses to alert the web app whenever the Docker image changes. Then, we updated the web app's code by modifying the index.js file, changing the title property to "Microsoft Learn". We rebuilt the image for the web app and pushed it to Container Registry. In the Container Registry's Webhooks section, we could see the webhook record reflecting the push event. Finally, we tested the web app again, and upon browsing it, we observed that the contents had changed to reflect the updates made to the container image. This demonstrated the automatic redeployment of the web app triggered by the webhook notification.

The subscription for virtual machines (VMs) in Azure has two separate costs: compute costs and storage costs. These costs can be scaled independently, allowing we to pay only for what we need. Compute costs are based on a per-hour basis but are calculated on a per-minute basis. we are charged for the exact usage time, and if we stop and deallocate the VM, we won't be charged for compute capacity. The hourly price varies depending on the VM size and operating system. Storage costs are separate and incurred regardless of the VM's status. we can choose between pay-as-us-go or Reserved Virtual Machine Instances (RI) options for compute costs. Pay-as-us-go offers flexibility with no long-term commitment, while RI provides cost savings for a specified period. It's recommended to have at least two virtual hard disks (VHDs) for VMs, one for the operating system and another for temporary storage. Additional disks can be added for application data. Storage is measured and billed based on the space used in Azure storage. we can use either standard or premium storage accounts, with premium storage offering higher performance for I/O intensive workloads. Managed disks are recommended for easier management and scalability, while unmanaged disks require managing storage accounts. Azure provides various OS images, and we can also create usr own custom image if needed.

In this set of steps, we explore how to create a virtual machine (VM) running a web server on Ubuntu using the Azure portal. The process begins by signing into the Azure portal and selecting "Create a resource." Then, we choose "Virtual machine" and configure the VM by providing basic parameters such as the VM name, region, image, size, and authentication type. We also specify inbound port rules, allowing SSH access. After reviewing and validating the settings, we create the VM, download the private key, and monitor the deployment process. Once the deployment is successful, we can access and manage the VM through the Overview page, which includes the public IP address for SSH connectivity. This straightforward process enables the quick deployment of a Linux-based VM in Azure.

The Azure portal is a user-friendly way to create resources in Microsoft Azure, but it may not be efficient for managing multiple resources. Azure Resource Manager organizes related resources into resource groups, making it easier to deploy, update, and delete them together. Resource Manager templates, which are JSON files, allow us to create and deploy specific configurations, enabling us to quickly create copies of resources. Azure PowerShell and Azure CLI provide command-line interfaces for scripting and automating resource management tasks, while Azure REST API allows developers to interact with Azure programmatically. Azure Client SDKs offer higher-level abstractions for easier interaction with Azure services. Azure VM extensions automate post-deployment tasks, and Azure Automation helps automate management tasks through process automation, configuration management, and update management. These tools and services provide a comprehensive suite of options to efficiently create and administer resources in Microsoft Azure.

In order to ensure the availability and reliability of services provided by a services company like Microsoft Azure, service level agreements (SLAs) are crucial. Azure provides various tools and services to manage availability, data security, and monitoring. One important aspect is the use of availability sets, which involve deploying at least two instances of each virtual machine (VM) to avoid a single point of failure. These VMs are spread across fault domains and update domains, ensuring that hardware failures or maintenance events have minimal impact. Azure Site Recovery is another valuable service that allows replication of workloads to a secondary location, enabling failover and ensuring business continuity in case of primary site outages. It also offers easy testing of failovers and facilitates migration to Azure for new opportunities. Overall, these steps and services contribute to maintaining high availability, automating VM management tasks, and safeguarding data.

Azure Backup is a comprehensive backup as a service offering by Microsoft Azure that ensures the protection of both physical and virtual machines, regardless of their location. It caters to various backup scenarios, including files and folders, application-aware snapshots, popular Microsoft server workloads, and native support for Azure Virtual Machines. Unlike traditional backup solutions, Azure Backup optimizes the underlying Azure platform, providing benefits such as automatic storage management, unlimited scaling, multiple storage options, unlimited data transfer, data encryption, and application-consistent backups with long-term retention. It utilizes components like the Azure Backup agent, System Center Data Protection Manager, Azure Backup Server, and Azure Backup VM extension, all integrated with a recovery services vault backed by efficient and cost-effective Azure Storage blobs. With Azure Backup, we can effortlessly define backup policies, schedule snapshots, and securely store usr data for as long as we need.

To migrate usr existing website to the Azure Cloud, we can create an Azure Virtual Machine (VM) using the latest Ubuntu image. Azure VMs are scalable computing resources that provide processors, memory, storage, and networking capabilities. we have multiple options to define and deploy VMs, such as using the Azure portal, Azure CLI, Azure PowerShell, or Azure Resource Manager template. During VM creation, we’ll need to provide essential information like selecting the Ubuntu image and configuring associated resources like a storage account, virtual network, network interface, and optionally a public IP address. These resources work together to virtualize a computer and run the Linux operating system. We can choose from prebuilt images or even create and upload custom disk images. By leveraging Azure VMs, we can easily manage usr website and access the VM remotely using SSH.

To connect to usr Linux VM in Azure, we have a few options. One way is to use a public IP address assigned to the VM, which allows us to interact with it over the internet. Another option is to set up a virtual private network (VPN) that securely connects usr on-premises network to Azure, providing access to the VM without a public IP. In this module, we will use the public IP address. To establish an SSH connection, we will need the VM's public IP address, the username of the local account on the VM, the public key configured in that account, the corresponding private key, and ensure that Port 22 is open on the VM. Once connected via SSH, we will have access to the Linux command line tools.

To secure our Azure VM running Apache, we need to adjust the network configuration. By default, new VMs have restricted inbound traffic, allowing only communication from the virtual network. To support different protocols, we need to create a network security group (NSG) and define inbound rules. NSGs act as a software firewall, filtering traffic at the networking level. Each rule specifies the source and destination addresses, protocol, port, direction, and whether to allow or deny the traffic. Azure processes the security group associated with the subnet first, followed by the security group applied to the network interface. It's important to note that security groups are optional, and if not applied, all traffic is allowed. The rules are evaluated in priority order, with deny rules halting the evaluation. To allow HTTP access and block everything else, we can define a new rule in the NSG.

To manage the video data from their traffic cameras on Microsoft Azure, my company has decided to create VMs. The process involves creating a VM using the Azure portal and configuring it for remote access. We need to select a suitable VM image from the Azure marketplace, which includes the desired operating system and pre-installed software tools. Additionally, we need to choose the appropriate storage option, such as a storage account for virtual hard disks. These resources, including the VM, storage, virtual network, network interface, and optional public IP address, work together to virtualize a computer and run the Windows operating system. By organizing these resources into a resource group, we can efficiently manage and administer the VM.

To create a new Windows virtual machine using Azure, we can follow these steps. First, sign in to the Azure portal and select "Create a resource." Search for and select "Windows Server" from the marketplace. Choose the Windows Server 2019 Datacenter option and click "Create." In the configuration wizard, provide the necessary details such as subscription, resource group, VM name, region, image, administrator account, password, and inbound port rules. Configure the disks and network settings accordingly. Finally, review the settings and create the VM. Once the deployment is complete, we can access and manage the virtual machine through the Azure portal.

To connect to an Azure VM using Remote Desktop Protocol (RDP), we need the VM's public or private IP address and port number. we can download a pre-configured RDP file from the Azure portal. After downloading the file, we can adjust settings such as display resolution, local resource sharing, and visual experience. Then, we can connect to the Windows VM by entering usr username and password. Once connected, we can install worker roles, install custom software by copying files from usr local machine, and initialize data disks using the Disk Management tool. RDP provides Desktop UI access for administering the Azure VM similar to a local computer.

To enable FTP support on our Azure VM, we need to make configuration changes to the network settings. By default, new VMs have restricted inbound traffic, allowing only communication from the virtual network and Azure load balancer or probe checks. To open the necessary ports for FTP, we have to follow a two-step process. First, we create a Network Security Group (NSG) that acts as a software firewall and controls network traffic rules. Then, within the NSG, we create an inbound rule to allow traffic on ports 20 and 21 for active FTP support. These steps ensure that our VM can receive FTP connections and securely manage network traffic.

The MEAN stack is a popular choice for building web applications due to its reliability, open-source nature, and compatibility with JavaScript. MongoDB, Express, AngularJS, and Node.js are the key components of MEAN, offering flexibility and ease of use. One advantage of MEAN is its ability to handle unstructured data, as MongoDB, a NoSQL database, doesn't require rigid data structures. Additionally, MEAN benefits from extensive documentation and a wide range of available resources. It is platform-independent, allowing developers to work with their preferred operating system. However, if usr data is highly structured or we prefer a different programming language, alternative stacks like LAMP (Linux, Apache, MySQL, PHP) might be more suitable. Understanding the features and considerations of different development stacks can help in choosing the most appropriate one for usr specific needs.

In this set of instructions, we are creating an Ubuntu Linux virtual machine (VM) on Azure. We begin by creating a resource group to contain our VM. Then, using the Azure CLI, we create the Ubuntu VM named "MeanStack" with the specified image and administrator username. The command also generates SSH keys for secure access to the VM. We open port 80 on the VM to allow incoming HTTP traffic. Next, we establish an SSH connection to the VM using the public IP address obtained from the VM creation output. Finally, with the Ubuntu VM set up, we are ready to proceed with installing the components of the MEAN stack, starting with MongoDB.

The steps we need to follow to create the application:

Create the necessary files and folders for the application:

server.js

package.json

app/model.js

app/routes.js

public/script.js

public/index.html

Define the data model for the application in app/model.js, which includes setting up the MongoDB connection and defining the schema for the books collection. Implement the Express.js routes in app/routes.js, which handle HTTP requests for listing, adding, and deleting books from the database. Create the client-side JavaScript application in public/script.js, which sends HTTP requests to the server to interact with the database. Design the user interface in public/index.html, including a form to add new books and a table to display existing books. Set up the Express.js server in server.js, configure it to serve static files from the public folder, and define routes for the application. Once we have created all the necessary files and implemented the code, we can run the application by starting the Node.js server. To do this, navigate to the Books directory and run the following command:

bash

Copy code

node server.js

After starting the server, we should be able to access the web application by visiting the server's IP address or domain name in usr web browser.