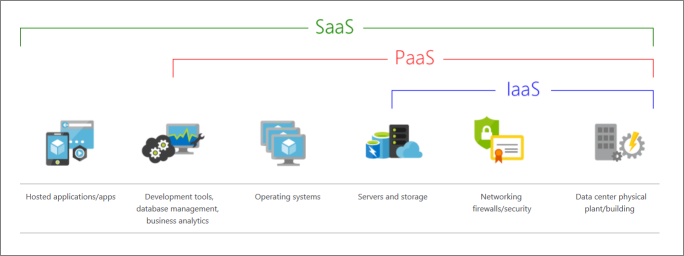
**AZURE**

**SERVICE MODELS AND REGIONS**

Service models:

* IaaS: Infrastructure-as-a-Service
* PaaS: Platform-as-a-Service
* SaaS: Software-as-a-Service



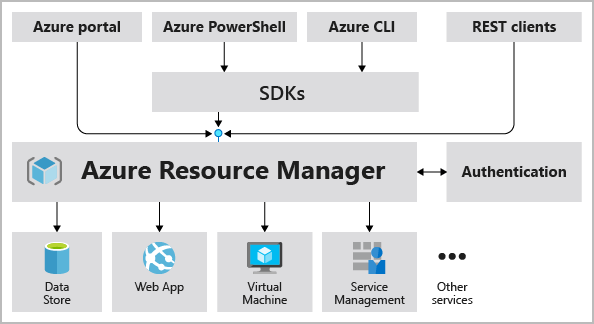
Regions:

* A region is a geographical area on the planet that contains one or multiple datacenters that are nearby and networked together with a low-latency network.
* Availability zones are physically separate datacenters within an Azure region. Each availability zone is made up of one or more datacenters equipped to be independent. If one availability zone goes down, the other continues working.

**SUBSCRIPTIONS, GROUPS AND RESOURCES**

Resources:

* Resources are instances of services that you create, like virtual machines, storage, or SQL databases.
* A resource group is a logical container for resources deployed on Azure.
* All resources must be in a resource group, and a resource can only be a member of a single resource group.
* Many resources can be moved between resource groups with some services having specific limitations or requirements to move.
* Resources are combined into resource groups, which act as a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed. If you delete a resource group, all resources contained within it are also deleted.
* Resource groups can't be nested.
* Azure Resource Manager is the deployment and management service for Azure. It provides a management layer that enables you to create, update, and delete resources in your Azure account. You use management features like access control, locks, and tags to secure and organize your resources after deployment.

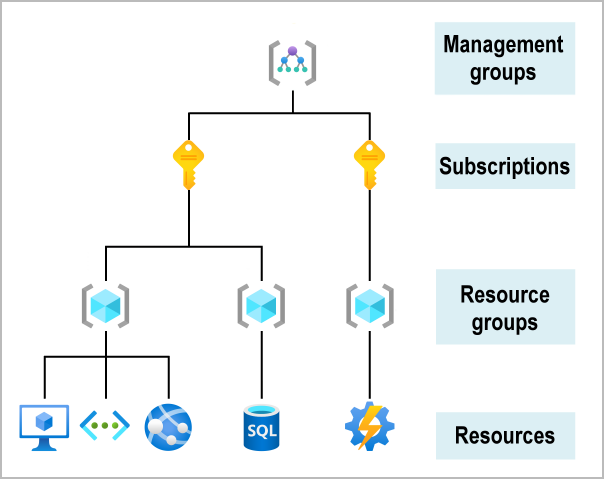


Subscriptions:

* Azure subscription is a logical unit of Azure services that links to an Azure account.
* A subscription provides you with authenticated and authorized access to Azure products and services and also allows you to provision resources.
* A subscription groups together user accounts and the resources that have been created by those user accounts.
* For each subscription, there are limits or quotas on the number of resources that you can create and use.
* Organizations can use subscriptions to manage costs and the resources that are created by users, teams, or projects.
* An account can have one subscription or multiple subscriptions that have different billing models.

Management groups:

* These groups help you manage access, policy, and compliance for multiple subscriptions.
* All subscriptions in a management group automatically inherit the conditions applied to the management group.



**VIRTUAL MACHINES**

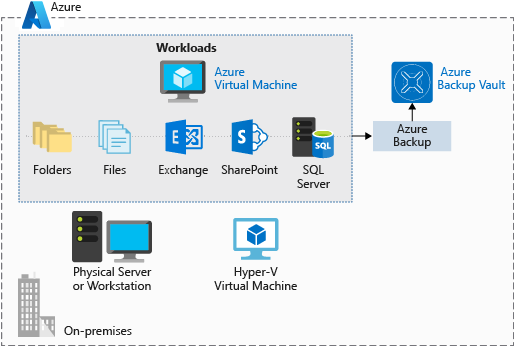
A virtual machine is a computer that you can run on-demand without needing any hardware.

Required resources for IaaS Virtual Machines:

* Network:
  + VMs and services that are part of the same virtual network can access one another.
  + By default, services outside the virtual network cannot connect to services within the virtual network.
  + After deciding the virtual network address space(s), you can create one or more subnets for your virtual network.
  + By default, there is no security boundary between subnets, so services in each of these subnets can talk to one another.
* Name: used as the computer name and also defines a manageable Azure resource, and it's not trivial to change later.
* Location: you must select a region where you want the resources (CPU, storage, etc.) to be allocated.
* Size: the best way to determine the appropriate VM size is to consider the type of workload your VM needs to run.
  + General purpose: designed to have a balanced CPU-to-memory ratio. Ideal for testing and development, small to medium databases, and low to medium traffic web servers.
  + Compute optimized: designed to have a high CPU-to-memory ratio. Suitable for medium traffic web servers, network appliances, batch processes, and application servers.
  + Memory optimized: designed to have a high memory-to-CPU ratio. Great for relational database servers, medium to large caches, and in-memory analytics.
  + Storage optimized: Storage optimized VMs are designed to have high disk throughput and IO. Ideal for VMs running databases.
  + GPU: specialized for heavy graphics rendering and video editing. These VMs are ideal options for model training and inferencing with deep learning.
  + High performance computes: the fastest and most powerful CPU virtual machines with optional high-throughput network interfaces.
* Pricing model: there are two separate costs the subscription will be charged for every VM, compute and storage.
  + Compute costs: compute expenses are priced on a per-hour basis but billed on a per-minute basis.
  + Storage costs: you are charged separately for the storage the VM uses.
* Storage:
  + All Azure virtual machines will have at least two virtual hard disks (VHDs). The first disk stores the operating system, and the second is used as temporary storage.
  + You can add additional disks to store application data; the maximum number is determined by the VM size selection (typically two per CPU).
* Operating system: Azure provides a variety of OS images that you can install into the VM, including several versions of Windows and flavors of Linux.

Back up your virtual machines:

* Azure Backup is a backup as a service offering that protects physical or virtual machines no matter where they reside: on-premises or in the cloud.
* Azure Backup can be used for a wide range of data backup scenarios



**TOOLS TO CREATE AND ADMINISTER RESOURCES IN AZURE**

1. Azure Resource Manager.
2. Azure PowerShell.
   1. PowerShell is a cross-platform shell that provides services like the shell window and command parsing.
   2. Azure PowerShell is an optional add-on package that adds the Azure-specific commands (referred to as cmdlets).
3. Azure CLI.
   1. Azure CLI is Microsoft's cross-platform command-line tool for managing Azure resources such as virtual machines and disks from the command line.
4. Azure REST API
   1. The Azure REST API provides developers with operations categorized by resource as well as the ability to create and manage VMs.
   2. Operations are exposed as URIs with corresponding HTTP methods (GET, PUT, POST, DELETE, and PATCH) and a corresponding response.
5. Azure Client SDK
   1. The Azure Client SDK encapsulates the Azure REST API, making it much easier for developers to interact with Azure.
6. Azure VM Extensions
   1. Azure VM extensions are small applications that enable you to configure and automate tasks on Azure VMs after initial deployment.
7. Azure Automation Services
   1. Azure Automation enables you to integrate services that allow you to automate frequent, time-consuming, and error-prone management tasks with ease.
   2. These services include process automation, configuration management, and update management.

**STORE DATA IN AZURE**

Approaches to storing data in the cloud:

* Structured data:
  + Data adheres to a strict schema, so all of the data has the same fields or properties.
  + The shared schema allows this type of data to be easily searched with query languages such as SQL (Structured Query Language).
* Semi-structured data:
* Less organized than structured data, and is not stored in a relational format, as the fields do not neatly fit into tables, rows, and columns.
* Data contains tags that make the organization and hierarchy of the data apparent - for example, key/value pairs and is also referred to as non-relational or NoSQL data.
* The expression and structure of the data in this style is defined by a serialization language.
* Serialization languages:
  + XML, or extensible markup language
  + JSON – or JavaScript Object Notation
  + YAML – or YAML Ain’t Markup Language
* Unstructured data:
  + Unstructured data is often delivered in files, such as photos or videos.
  + Examples of unstructured data include: media files, Office files, text files or log files.

Transactions:

* A transaction is a logical group of database operations that execute together.
* Transactions are often defined by a set of four requirements, referred to as ACID guarantees (Atomicity, Consistency, Isolation, and Durability)
  + Atomicity means a transaction must execute exactly once and must be atomic; either all of the work is done, or none of it is.
  + Consistency ensures that the data is consistent both before and after the transaction.
  + Isolation ensures that one transaction is not impacted by another transaction.
  + Durability means that the changes made due to the transaction are permanently saved in the system.
* OLTP vs OLAP.
  + Transactional databases are often called OLTP (Online Transaction Processing) systems. OLTP systems commonly support lots of users, have quick response times, and handle large volumes of data.
  + On the contrary, OLAP (Online Analytical Processing) systems commonly support fewer users, have longer response times, can be less available, and typically handle large and complex transactions.
* Product catalog data should be stored in a transactional database.
* Photos and videos in a product catalog don't require transactional support.
* For the business data, because all of the data is historical and unchanging, transactional support is not required.

**AZURE STORAGE SERVICES**

Microsoft Azure Storage is a managed service that provides durable, secure, and scalable storage in the cloud.

* Managed: Microsoft Azure handles maintenance and any critical problems.
* Durable: redundancy ensures that your data is safe in the event of transient hardware failures.
* Secure: all data written to Azure Storage is encrypted by the service.
* Scalable: Azure Storage is designed to be massively scalable to meet the data storage and performance needs of today's applications.

Azure storage includes four types of data:

* Blobs:
  + A blob is a massively scalable object store for text and binary data.
  + It is ideal for serving images or documents directly to a browser, storing files for distributed access, streaming video or audio and storing other types of data like backups, recoveries, etc.
  + Azure Storage supports three kinds of blobs:
    - Block blobs - used to hold text or binary files up to ~5 TB (50,000 blocks of 100 MB) in size.
    - Page blobs - used to hold random-access files up to 8 TB in size, primarily for the backing storage in Azure Virtual Machines.
    - Append blobs - made up of blocks like block blobs, but they are optimized for append operations. These blobs are frequently used for logging information from one or more sources into the same blob.
* Files:
  + Azure File storage enables you to set up highly available network file shares that can be accessed using the standard Server Message Block (SMB) protocol.
  + Multiple VMs can share the same files with both read and write access.
  + You can also read the files using the REST interface or the storage client libraries.
  + File shares can be used for:
    - Storing shared configuration files for VMs, tools, or utilities.
    - Log files such as diagnostics, metrics, and crash dumps.
    - Shared data between on-premises applications and Azure VMs.
* Queues:
  + Azure Queue Storage is used to store and retrieve messages.
  + Queue messages can be up to 64 KB in size, and a queue can contain millions of messages.
  + Queues are used to store lists of messages to be processed asynchronously.
* Table Storage: a NoSQL store for schema-less storage of structured data.

Code example. For additional information, see:

https://docs.microsoft.com/en-us/learn/modules/connect-an-app-to-azure-storage/

#!/usr/bin/env node

require('dotenv').config();

const { BlobServiceClient } = require("@azure/storage-blob");

const storageAccountConnectionString = process.env.AZURE\_STORAGE\_CONNECTION\_STRING;

const blobServiceClient = BlobServiceClient.fromConnectionString(storageAccountConnectionString);

async function main() {

// Create a container (folder) if it does not exist

const containerName = 'photos';

const containerClient = blobServiceClient.getContainerClient(containerName);

if ( !containerClient.exists()) {

const createContainerResponse = await containerClient.createIfNotExists();

console.log(`Create container ${containerName} successfully`, createContainerResponse.succeeded);

}

else {

console.log(`Container ${containerName} already exists`);

}

// Upload the file

const filename = 'docs-and-friends-selfie-stick.png';

const blockBlobClient = containerClient.getBlockBlobClient(filename);

blockBlobClient.uploadFile(filename);

// Get a list of all the blobs in the container

let blobs = containerClient.listBlobsFlat();

let blob = await blobs.next();

while (!blob.done) {

console.log(`${blob.value.name} --> Created: ${blob.value.properties.createdOn} Size: ${blob.value.properties.contentLength}`);

blob = await blobs.next();

}

}

main();

**AZURE STORAGE SECURITY**

All data written to Azure Storage is automatically encrypted by Storage Service Encryption (SSE) with a 256-bit Advanced Encryption Standard (AES) cipher.

For virtual machines (VMs), Azure lets you encrypt virtual hard disks (VHDs) by using Azure Disk Encryption. This encryption uses BitLocker for Windows images, and it uses dm-crypt for Linux.

Keep your data secure by enabling transport-level security between Azure and the client. Always use HTTPS to secure communication over the public internet.

Every request to a secure resource must be authorized. The service ensures that the client has the permissions required to access the data

Storage account keys:

* In Azure Storage accounts, shared keys are called storage account keys.
* Azure creates two of these keys (primary and secondary) for each storage account.
* The keys give access to everything in the account.

For untrusted clients, use a shared access signature (SAS). A SAS is a string that contains a security token that can be attached to a URI. Use a SAS to delegate access to storage objects and specify constraints, such as the permissions and the time range of access.

Types of shared access signatures:

* You can use a *service-level SAS* to allow access to specific resources in a storage account.
* Use an *account-level SAS* to allow access to anything that a service-level SAS can allow, plus additional resources and abilities.
* You'd typically use a SAS for a service where users read and write their data to your storage account. Accounts that store user data have two typical designs:
  + Clients upload and download data through a front-end proxy service, which performs authentication.
  + A lightweight service authenticates the client, as needed. Next, it generates a SAS. After receiving the SAS, the client can access storage account resources directly.

Follow these steps to change default network access in the Azure portal.

1. Go to the storage account you want to secure.
2. Select Networking.
3. To restrict traffic from selected networks, select Selected networks. To allow traffic from all networks, select All networks.
4. To apply your changes, select Save.

**BLOB STORAGE**

Blobs give you object storage in the cloud and an API that lets you build apps to access the data.

Blobs are files for the cloud. Apps work with blobs in much the same way as they work with files on a disk. However, unlike a local file, you can reach blobs from anywhere with an internet connection.

Azure Blob Storage is unstructured, meaning that there are no restrictions on the kinds of data it can hold.

Blobs aren't efficient for structured data that needs to be queried frequently. They have higher latency than memory and local disks, and don't have the indexing features. However, blobs are frequently used in combination with databases to store non-queryable data.

In Blob Storage, every blob lives inside a blob container. You can store an unlimited number of blobs in a container and an unlimited number of containers in a storage account.

Design a storage organization:

* A single storage account is flexible enough to organize your blobs. However, you should use additional storage accounts as necessary to logically separate costs and control access to data.
* A storage account connection string contains all the information needed to connect to Blob storage, most importantly the account name and the account key.
* The nature of your app and the data it stores should drive your strategy for naming and organizing containers and blobs.
  + Apps using blobs as part of a storage scheme that includes a database often don't need to rely heavily on organization. Such apps commonly use identifiers like GUIDs as blob names and reference these identifiers in database records. The app uses the database to determine where blobs are stored and the kind of data they contain.
  + Other apps may use Azure Blob Storage more like a personal file system, where container and blob names are used to indicate meaning and structure. Blob names in these kinds of apps will often look like traditional file names and include file name extensions like .jpg to indicate what type of data they contain. They'll use virtual directories (see below) to organize blobs, and will frequently use metadata tags to store information about blobs and containers.
* By default, all blobs require authentication to access. However, you can configure individual containers to allow public downloading of their blobs without authentication.
* Technically, containers are "flat" and don't support any kind of nesting or hierarchy. But if you give your blobs hierarchical names that look like file paths (such as finance/budgets/2017/q1.xls), the API's listing operation can filter results to specific prefixes.
* Getting a blob reference does not make any calls to Azure Storage, it simply creates an object locally that can work with a stored blob.
* There are three different kinds of blobs in which you can store data:
  + Block blobs are composed of blocks of different sizes that can be uploaded independently and in parallel.
  + Append blobs are specialized block blobs that support only appending new data (not updating or deleting existing data), but they're very efficient at it.
  + Page blobs are designed for scenarios that involve random-access reads and writes.

**APP SERVICE**

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends.

Depending on the usage of the web app, you can scale your app up/down the resources of the underlying machine that is hosting your web app. Scaling out/in is the ability to increase, or decrease, the number of machine instances that are running your web app.

The Azure portal provides out-of-the-box continuous integration and deployment with Azure DevOps, GitHub, Bitbucket, FTP, or a local Git repository on your development machine.

App Service can also host web apps natively on Linux for supported application stacks. It can also run custom Linux containers (also known as Web App for Containers).

The pricing tier of an App Service plan determines what App Service features you get and how much you pay for the plan. There are a few categories of pricing tiers:

* Shared compute: Both Free and Shared share the resource pools of your apps with the apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources can't scale out.
* Dedicated compute: The Basic, Standard, Premium, PremiumV2, and PremiumV3 tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances are available to you for scale-out.
* Isolated: This tier runs dedicated Azure VMs on dedicated Azure Virtual Networks. It provides network isolation and the maximum scale-out capabilities.
* Consumption: This tier is only available to function apps. It scales the functions dynamically depending on workload.

Deploy to App Service:

* Azure supports automated deployment directly from:
  + Azure DevOps: You can push your code to Azure DevOps, build your code in the cloud, run the tests, generate a release from the code, and finally, push your code to an Azure Web App.
  + GitHub or Bitbucket: Azure supports automated deployment directly from both.
* Azure supports manual deployment from:
  + Git: App Service web apps feature a Git URL that you can add as a remote repository.
  + CLI: *webapp up* is a feature of the az command-line interface that packages your app and deploys it. Unlike other deployment methods, az webapp up can create a new App Service web app for you if you haven't already created one.
  + Zip deploys: Use curl or a similar HTTP utility to send a ZIP of your application files to App Service.
  + FTP/S: FTP or FTPS is a traditional way of pushing your code to many hosting environments, including App Service.

Azure App Service provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal or no code.

App Service supports Identity providers such as Microsoft, Google, Facebook, etc.

The authentication flow is the same for all providers, but differs depending on whether you want to sign in with the provider's SDK.

* Without provider SDK: The application delegates federated sign-in to App Service. This is typically the case with browser apps.
* With provider SDK: The application signs users in to the provider manually and then submits the authentication token to App Service for validation. This is typically the case with browser-less apps, which can't present the provider's sign-in page to the user.

By default, apps hosted in App Service are accessible directly through the internet and can reach only internet-hosted endpoints. But for many applications, you need to control the inbound and outbound network traffic.

Configure application settings:

* App settings are variables passed as environment variables to the application code.
* In the Configuration > General settings section you can configure some common settings for your app.

Scale apps:

* Autoscaling is a cloud system or process that adjusts available resources based on the current demand by performing scaling in and out, as opposed to scaling up and down. Autoscaling monitors the resource metrics of a web app as it runs and detects situations where additional resources are required.
  + Autoscaling responds to changes in the environment by adding or removing web servers and balancing the load between them.
  + Autoscaling is a suitable solution when you can't easily predict the workload in advance, or when the workload is likely to vary by date or time.
  + Autoscaling isn't the best approach to handling long-term growth.
* Metrics for autoscale rules:
  + CPU Percentage.
  + Memory Percentage.
  + Disk Queue Length. This metric is a measure of the number of outstanding I/O requests across all instances.
  + Http Queue Length. This metric shows how many client requests are waiting for processing by the web app.
  + Data In/Out.
* Autoscale best practices:
  + Ensure the maximum and minimum values are different and have an adequate margin between them.
  + Choose the appropriate statistic for your diagnostics metric.
  + Choose the thresholds carefully for all metric types.
  + Always select a safe default instance count.
  + Configure autoscale notifications.

**INTRODUCTION TO DOCKER**

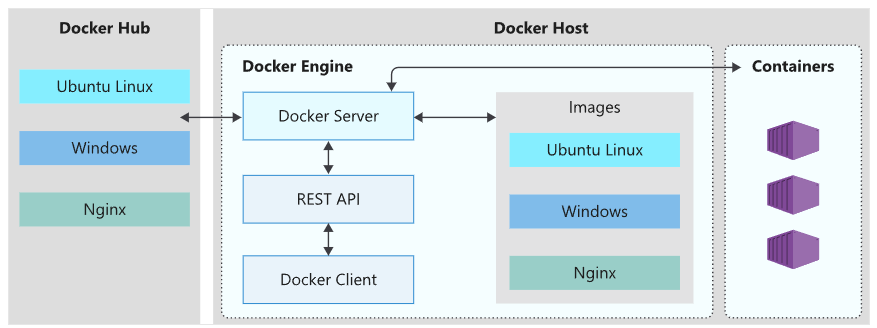
Docker is a containerization platform used to develop, ship, and run containers.

There are several basic concepts before we look at the Docker features:

* Container: a container is a loosely isolated environment that allows us to build and run software packages. These software packages are called *container images* and include the code and all dependencies to run applications any computing environment.
* Software containerization: it is an OS virtualization method that is used to deploy and run containers without using a virtual machine (VM). Containers can run on physical hardware, in the cloud, VMs, and across multiple OSs.

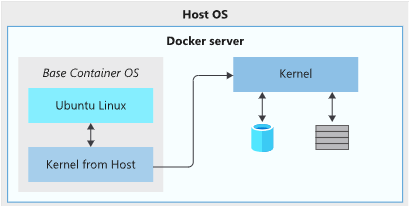
Docker architecture:

* Docker Engine: the Docker Engine consists of several components configured as a client-server implementation where the client and server run simultaneously on the same host.
  + The Docker client is a command-line application named *docker* to interact with a Docker server. The docker command uses the Docker REST API to send instructions to either a local or remote server and functions as the primary interface we use to manage our containers.
* The Docker server is a daemon named *dockerd* that responds to requests from the client via the Docker REST API and can interact with other daemons. The Docker server is also responsible for tracking the lifecycle of our containers.
* There are several objects that you'll create and configure to support your container deployments. These include networks, storage volumes, plugins, and other service objects.
* Docker Hub: the Docker Hub is a Software as a Service (SaaS) Docker container registry. Docker registries are repositories that we use to store and distribute the container images we create. Docker Hub is the default public registry Docker uses for image management.



How Docker images work:

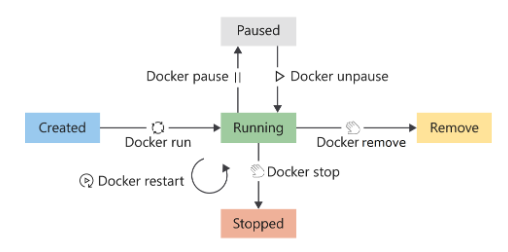
* A container image is a portable package that contains software applications. This includes application code, system packages, binaries, libraries, configuration files, and may include the operating system running in the container.
* The Docker engine runs in an OS. Docker containers running on Linux share the host OS kernel and don't require a container OS as long as the binary can access the OS kernel directly. However, Windows containers need a container OS.
* The container OS is the OS that is part of the packaged image. We have the flexibility to include different versions of Linux or Windows OSs in a container.



* Unionfs (Stackable Unification File System) is a filesystem used to create Docker images that allows you to stack several directories, called branches, in such a way that it appears as if the content is merged. However, the content is physically kept separate.
* Base/parent images:
  + A base image is an image that uses the Docker scratch image, which is an empty container image that doesn't create a filesystem layer. This image assumes that the application you're going to run can directly use the host OS kernel.
  + A parent image is a container image from which you create your images.
  + Both image types allow us to create a reusable image. However, base images allow us more control over the contents of the final image.
* A Dockerfile is a text file that contains the instructions we use to build and run a Docker image. The following aspects of the image are defined:
  + The base or parent image we use to create the new image.
  + Commands to update the base OS and install additional software.
  + Build artifacts to include, such as a developed application.
  + Services to expose, such as storage and network configuration.
  + Command to run when the container is launched.
* The Docker CLI allows us to manage images by building, listing, removing, and running them. We manage Docker images by using the docker client. The client doesn't execute the commands directly and sends all queries to the dockerd daemon.

How Docker containers work:

* A Docker container has a lifecycle that you can manage and track the state of the container.



* We can manage docker containers using different commands.
* Docker container storage is temporary, coupled to the underlying host machine and has poor performance.
* Containers can make use of two options to persist data. The first option is to make use of volumes, and the second is bind mounts.
  + A volume is stored on the host filesystem at a specific folder location. Docker will mount and manage the volumes in the container. After mounting, these volumes are isolated from the host machine.
  + Multiple containers can simultaneously use the same volumes. Volumes also don't get removed automatically when a container stops using the volume.
  + A bind mount is conceptually the same as a volume, however, instead of using a specific folder, you can mount any file or folder on the host.
  + Bind mounts have limited functionality compared to volumes, and even though they're more performant, they depend on the host having a specific folder structure in place.
* Docker also has a network configuration that allows for the isolation of containers on the Docker host. This feature enables you to build and configure apps that can communicate securely with each other.
  + Docker provides three pre-configured network configurations: Bridge, Host and None.
  + The bridge network is the default configuration applied to containers when launched without specifying any additional network configuration. This network is an internal, private network used by the container, and isolates the container network from the Docker host network.
  + The host network enables you to run the container on the host network directly. This configuration effectively removes the isolation between the host and the container at a network level.
  + To disable networking for containers, use the none network option.

Docker benefits:

* Efficient use of hardware, since containers run without using a virtual machine (VM).
* Container isolation, providing security features to run multiple containers simultaneously on the same host without affecting each other.
* Application portability, with Containers being run almost everywhere, including desktops, physical servers, VMs, and in the cloud.
* The container becomes the unit we use to distribute applications.
* Management of hosting environments.
* Cloud deployments.

Docker disadvantages:

* Security and virtualization: Containers provide a level of isolation. However, containers share a single host OS kernel, which can be a single point of attack.
* Service monitoring: managing the applications and containers are more complicated than traditional VM deployments. Logging features exist that tell us about the state of the running containers. However, more detailed information about services inside the container is harder to monitor.

**INTRODUCTION TO KUBERNATES**

Kubernetes is a portable, extensible open-source platform for managing and orchestrating containerized workloads. Kubernetes uses a container orchestrator, which is a system that automatically deploys and manages containerized apps.

Kubernetes benefits:

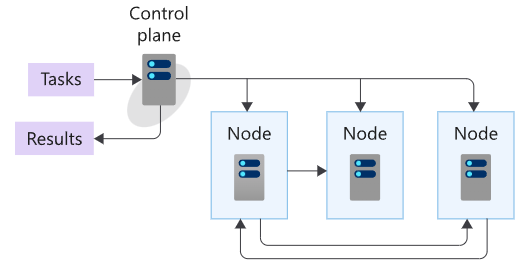
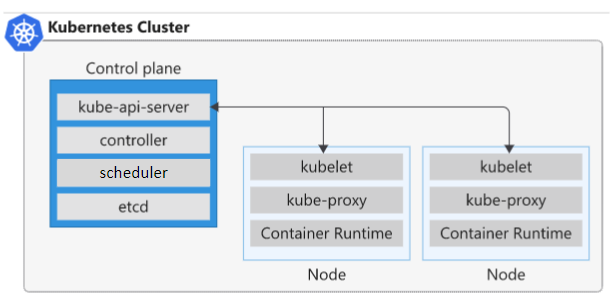
* Self-healing of containers, such as restarting containers that fail or replacing containers.
* Scaling deployed container count up or down dynamically, based on demand.
* Automating rolling updates and rollbacks of containers.
* Managing storage and network traffic.
* Storing and managing sensitive information, such as usernames and passwords.

Kubernetes considerations:

* Aspects such as deployment, scaling, load balancing, logging, and monitoring are all optional.
* If your app can run in a container, it can run on Kubernetes.
* Kubernetes doesn't provide middleware, data-processing frameworks, databases, caches, or cluster storage systems.
* For Kubernetes to run containers, it needs a container runtime, like Docker.
* You're responsible for maintaining your Kubernetes environment.

Kubernetes architecture:

* Kubernetes is based on clusters. Instead of having a single virtual machine (VM), it uses several machines working as one.
* A cluster is a set of computers that you configure to work together and view as a single system and they will typically do the same kinds of tasks.
  + A cluster is node-based. There are two types of nodes in a Kubernetes cluster that provide specific functionality.
    - Control plane: it runs a collection of services that manage the orchestration functionality in Kubernetes.
    - Node: it is where your compute workloads run. Each node communicates with the control plane via the API server to inform it about state changes on the node.
  + The computers in a cluster that run the tasks are called *nodes*, and the computers that run the scheduling software are called *control planes*.
  + A cluster uses centralized software that's responsible for scheduling and controlling these tasks.
  + A cluster contains at least one main plane and one or more nodes. Both the control planes and node instances can be physical devices, virtual machines, or instances in the cloud. The default host OS in Kubernetes is Linux, with default support for Linux-based workloads.
* A container registry is a remote storage where we can store our images safely so Kubernetes can download and run them.



Services that run on a control plane:

* API server.
  + Front end to the control plane in your Kubernetes cluster.
  + All the communication between the components is done through this API.
  + You can use a command-line app called *kubectl* to run commands against your Kubernetes cluster's API server
  + The component that provides this API is called *kube-apiserver*.
* Backing store.
  + A persistence store that your Kubernetes cluster uses to save the complete configuration of a Kubernetes cluster.
  + Kubernetes uses key-value store called *etcd*.
* Scheduler.
  + Component that's responsible for the assignment of workloads across all nodes
* Controller manager.
  + Control manager launches and monitors the controllers configured for a cluster through the API server.
  + Kubernetes uses controllers to track the state of objects in the cluster.
* Cloud controller manager.
  + Component that integrates with the underlying cloud technologies in your cluster when the cluster is running in a cloud environment.

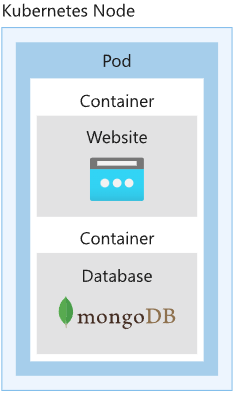
Services that run on a node:

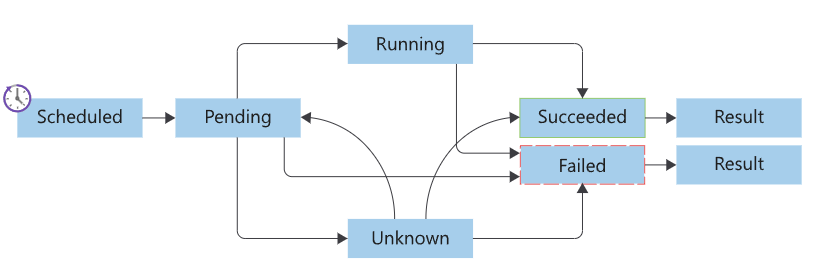
* Kubelet.
  + Agent that runs on each node in the cluster, and monitors work requests from the API server
* Kube-proxy.
  + Component responsible for local cluster networking.
  + It ensures that each node has a unique IP address.
* Container runtime.
  + Underlying software that runs containers on a Kubernetes cluster.
  + The runtime is responsible for fetching, starting, and stopping container images.

Interact with a Kubernetes cluster:

* Kubernetes provides a command-line tool called *kubectl* to manage your cluster.
* *kubectl* uses a configuration file that includes:
  + Cluster configuration specifies a cluster name, certificate information, and the service API endpoint associated with the cluster.
  + User configuration specifies the users and their permission levels.
  + Context configuration groups clusters and users by using a friendly name.

Kubernetes pods:

* A pod represents a single instance of an app running in Kubernetes and groups containers and applications into logical structure.
* The workloads that you run on Kubernetes are containerized apps. Unlike in a Docker environment, you can't run containers directly on Kubernetes. You package the container into a Kubernetes object called a pod.
* A single pod can hold a group of one or more containers. However, a pod typically doesn't contain multiples of the same app.
* A pod includes information about the shared storage and network configuration, and a specification about how to run its packaged containers.
* Lifecycle of a Kubernetes pod.



* Containers inside pods has three different states:
  + Waiting: default state of a container and the state that the container is in when it's not running or terminated.
  + Running: the container is running as expected without any problems.
  + Terminated: the container is no longer running. The reason is that either all tasks finished, or the container failed for some reason.

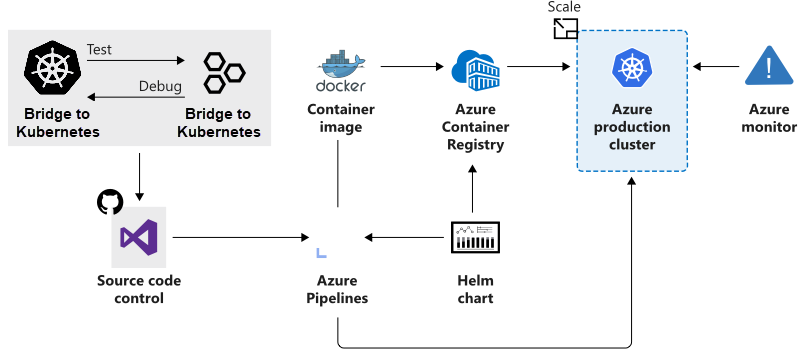
**KUBERNATES DEPLOYMENT**

To deploy Kubernetes components, we deploy pods.

Pod deployment has several options:

* Pod templates.
  + A pod template enables you to define the configuration of the pod you want to deploy.
  + You can use templates to deploy pods manually. However, a manually-deployed pod isn't relaunched after it fails, is deleted, or is terminated.
* Replication controllers.
  + A replication controller uses pod templates and defines a specified number of pods that must run.
  + The controller helps you run multiple instances of the same pod, and ensures pods are always running on one or more nodes in the cluster.
  + The controller replaces running pods in this way with new pods if they fail, are deleted, or are terminated.
* Replica sets.
  + A replica set replaces the replication controller as the preferred way to deploy replicas.
  + A replica set includes the same functionality as a replication controller. However, it has an extra configuration option to include a selector value.
  + A selector enables the replica set to identify all the pods running underneath it. Using this feature, you can manage pods labeled with the same value as the selector value, but not created with the replicated set.
* Deployments.
  + A deployment creates a management object one level higher than a replica set, and enables you to deploy and manage updates for pods in a cluster.
  + Deployments, by default, provide a rolling update strategy for updating pods.
  + Deployments also provide you with a rollback strategy, which you can execute by using *kubectl*.
* Deployment considerations:
  + Networking: Kubernetes expects you to configure networking in such a way that:
    - Pods can communicate with one another across nodes without Network Address Translation (NAT).
    - Nodes can communicate with all pods, and vice versa, without NAT.
    - Agents on a node can communicate with all nodes and pods.
  + Services: provide stable networking for pods.
    - It abstracts the IP address for networked workloads and acts as a load balancer and redirects traffic to the specific ports of specified ports
    - It enables communication between nodes, pods, and users of your app, both internal and external, to the cluster.
    - It also enables you to target and manage specific pods in your cluster by using selector labels instead of IP addresses.
  + Storage: Kubernetes uses the same storage volume concept as Docker.
  + Cloud integration considerations: Kubernetes doesn't provide any of the following services: middleware, data-processing frameworks, databases, caches, cluster storage systems.

**KUBERNATES DEPLOYMENT IN AZURE**



Configure an AKS (Azure Kubernetes) cluster:

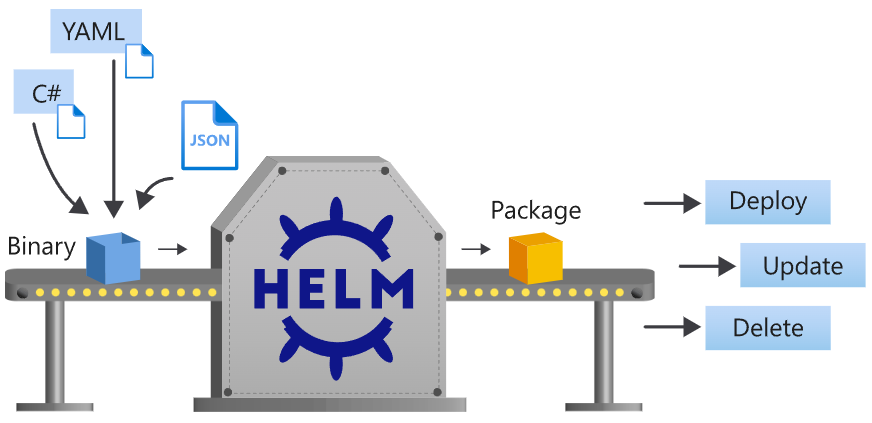
When you create a new AKS cluster, you have several different items of information that you need to configure. Each item affects the final configuration of your cluster. These items include:

* Node pools: you create node pools to group nodes in your AKS cluster.
* Node count: the node count is the number of nodes your cluster will have in a node pool.
* Automatic routing: a Kubernetes cluster blocks all external communications by default, so you need to manually create an ingress with an exception that allows incoming client connections to that particular service. AKS allows you to overcome the complexity of this process by enabling what's called HTTP application routing.
* Ingress controllers: they provide the capability to deploy and expose your applications to the world without the need to configure network-related services.

Using Helm as package manager:

Helm is a package manager for Kubernetes that combines all your application's resources and deployment information into a single deployment package. You specify the name of the application you want to install, update, or remove, and Helm takes care of the process.

Helm allows you to create templated, human-readable YAML script files to manage your application's deployment.



Helm uses four components to manage application deployments on a Kubernetes cluster.

* Helm client.
  + The Helm client is a client installed binary responsible for creating and submitting the manifest files required to deploy a Kubernetes application. The client is responsible for the interaction between the user and the Kubernetes cluster.
* Helm charts.
  + A Helm chart consists of several files and folders to describe the chart.
  + It contains all the information required to build and deploy the manifest files for an application to run on a Kubernetes cluster.
* Helm releases.
  + A Helm release is the application or group of applications deployed using a chart.
  + Each time you install a chart, a new instance of an application is created on the cluster.
* Helm repositories.
  + A Helm repository is a dedicated HTTP server that stores information on Helm charts.

