[*Azure Basics* 5](#_Toc25531111)

[**1.** **What are Cloud models?** 5](#_Toc25531112)

[**2.** **What are the characteristics of Public cloud model?** 5](#_Toc25531113)

[**3.** **What are the characteristics of Private cloud model?** 5](#_Toc25531114)

[**4.** **What are the characteristics of Hybrid cloud model?** 6](#_Toc25531115)

[**5.** **What are various Azure services?** 6](#_Toc25531116)

[**6.** **What is Azure Security Center?** 7](#_Toc25531117)

[**7.** **What is Azure Key Vault?** 7](#_Toc25531118)

[**8.** **What are Azure Information Protection?** 8](#_Toc25531119)

[**9.** **What is Azure Firewall?** 8](#_Toc25531120)

[**10.** **What is Role Based Access Control (RBAC)?** 9](#_Toc25531121)

[**11.** **How RBAC works?** 9](#_Toc25531122)

[**12.** **What is Subscription?** 11](#_Toc25531123)

[**13.** **What is Azure AD?** 12](#_Toc25531124)

[**14.** **What is Tenant?** 12](#_Toc25531125)

[**15.** **What happens when a Tenant is deleted?** 13](#_Toc25531126)

[**16.** **What is Multi-tenancy?** 13](#_Toc25531127)

[**17.** **Terminology** 14](#_Toc25531128)

[**18.** **What is Domain and Domain model?** 15](#_Toc25531129)

[**19.** **What is Bounded-Context?** 15](#_Toc25531130)

[**20.** **What are cloud native applications?** 16](#_Toc25531131)

[**21.** **What are the factors of cloud native applications (Micro-services)?** 16](#_Toc25531132)

[**22.** **What are the key attributes of cloud native applications?** 17](#_Toc25531133)

[**23.** **What is cloud orchestration?** 17](#_Toc25531134)

[**24.** **What is the difference between Cloud orchestration and automation?** 18](#_Toc25531135)

[**25.** **What is Auto-scaling?** 19](#_Toc25531136)

[**26.** **What are the strategies for Auto-scaling?** 19](#_Toc25531137)

[**27.** **What are the points to consider when using Azure *Auto-scale*?** 20](#_Toc25531138)

[*IAAS* 22](#_Toc25531139)

[**1.** **What are Azure Compute Services?** 22](#_Toc25531140)

[**2.** **What is Availability set?** 22](#_Toc25531141)

[**3.** **What is Azure VM scale sets?** 22](#_Toc25531142)

[**4.** **What are Regions, Geography and Availability zones?** 23](#_Toc25531143)

[**5.** **Benefits of Resource Group.** 23](#_Toc25531144)

[**6.** **What are Azure Storage?** 23](#_Toc25531145)

[**7.** **What are the benefits of Message Queue?** 24](#_Toc25531146)

[**8.** **Queue storage Vs. Azure Service Bus?** 25](#_Toc25531147)

[**9.** **What is Blob storage?** 26](#_Toc25531148)

[**10.** **What is File storage?** 26](#_Toc25531149)

[**11.** **File Storage Vs. Blob storage?** 27](#_Toc25531150)

[**12.** **Table storage vs. SQL based storage?** 28](#_Toc25531151)

[**13.** **What is unmanaged disk?** 28](#_Toc25531152)

[**14.** **What is managed disk?** 29](#_Toc25531153)

[**15.** **What are the types of VHDs in Azure?** 29](#_Toc25531154)

[**16.** **What is managed disk snapshot and image?** 30](#_Toc25531155)

[**17.** **What is Storage redundancy?** 30](#_Toc25531156)

[**18.** **What is Subnet and Subnet masking?** 31](#_Toc25531157)

[**19.** **What are the benefits of Sub-netting?** 31](#_Toc25531158)

[**20.** **What is Azure Virtual network?** 32](#_Toc25531159)

[**21.** **How the communication with the internet happens in Azure virtual network?** 32](#_Toc25531160)

[**22.** **How the communication between the azure resources happens in Azure virtual network?** 33](#_Toc25531161)

[**23.** **What are the various connectivity available in Azure?** 33](#_Toc25531162)

[**24.** **How filtering Traffic network works in Azure?** 34](#_Toc25531163)

[**25.** **What is Traffic Manager?** 35](#_Toc25531164)

[**26.** **What are the traffic routing methods available in traffic manager?** 35](#_Toc25531165)

[**27.** **What is Traffic manager endpoint monitoring?** 36](#_Toc25531166)

[**28.** **What is Azure application gateway?** 37](#_Toc25531167)

[**29.** **Load-balancer Vs. Traffic manager Vs. Application gateway?** 37](#_Toc25531168)

[**30.** **What is Network Security Group (NSG)?** 38](#_Toc25531169)

[*PAAS* 40](#_Toc25531170)

[**1.** **What is an App Service?** 40](#_Toc25531171)

[**2.** **What is App Service Plan?** 40](#_Toc25531172)

[**3.** **What are the various pricing tiers in App Service Plan?** 40](#_Toc25531173)

[**4.** **What is Web-App?** 41](#_Toc25531174)

[**5.** **What is Mobile App?** 41](#_Toc25531175)

[**6.** **What is Logic-App?** 41](#_Toc25531176)

[**7.** **What are Azure Functions?** 42](#_Toc25531177)

[**8.** **What are triggers supported in Azure Functions?** 42](#_Toc25531178)

[**9.** **How much does Functions app cost?** 43](#_Toc25531179)

[**10.** **What are durable Functions?** 44](#_Toc25531180)

[**11.** **What are Web-Jobs?** 45](#_Toc25531181)

[**12.** **Web Jobs Vs. Azure functions?** 45](#_Toc25531182)

[**13.** **What are the Pros and Cons of Serverless?** 46](#_Toc25531183)

[**14.** **What is App Service Environment (ASE)?** 47](#_Toc25531184)

[**15.** **Normal App service(Multi-tenant) Vs. ASE?** 48](#_Toc25531185)

[**16.** **What is External ASE and ILB ASE?** 48](#_Toc25531186)

[**17.** **What is Enterprise Application Integration?** 49](#_Toc25531187)

[**18.** **What are the ways to implement EAI?** 50](#_Toc25531188)

[**19.** **What is Enterprise service bus?** 50](#_Toc25531189)

[**20.** **What are Azure Service Bus, Service Bus Queue and, Service Bus Topics?** 50](#_Toc25531190)

[**21.** **What is Azure service fabric?** 51](#_Toc25531191)

[**22.** **What is Event hub?** 51](#_Toc25531192)

[**23.** **What is Notification hubs?** 52](#_Toc25531193)

[**24.** **What is Azure Redis Cache?** 53](#_Toc25531194)

[**25.** **What is Azure CDN?** 53](#_Toc25531195)

[**26.** **How CDN works?** 53](#_Toc25531196)

[**27.** **What are the Pros and Cons of Container?** 54](#_Toc25531197)

[**28.** **Serverless vs. Container, what to choose when?** 54](#_Toc25531198)

[*OAuth* 56](#_Toc25531199)

[**1.** **What is OAuth?** 56](#_Toc25531200)

[**2.** **What is Access Token and Id Token?** 56](#_Toc25531201)

[**3.** **Define Scope** 57](#_Toc25531202)

[**4.** **What are the ways to use Scopes?** 57](#_Toc25531203)

[**5.** **What are the various Authorization grant types?** 57](#_Toc25531204)

[**6.** **What is Open Id Connect and how it is different than OAuth?** 58](#_Toc25531205)

[**7.** **What parameters are needed to get Auth-code?** 58](#_Toc25531206)

[**8.** **How an application is registered with STS?** 59](#_Toc25531207)

[*Azure DevOps* 60](#_Toc25531208)

[**1.** **What are the approaches for Infrastructure as code?** 60](#_Toc25531209)

[**2.** **What is Idempotence?** 60](#_Toc25531210)

[**3.** **What is a Technical debt?** 60](#_Toc25531211)

[**4.** **What is Configurational drift?** 61](#_Toc25531212)

[**5.** **What is Azure Resource Manager?** 61](#_Toc25531213)

[**6.** **What is ARM Template and its components?** 62](#_Toc25531214)

[**7.** **How deployment dependencies are handled?** 64](#_Toc25531215)

[**8.** **What are the deployment mode when templates are used?** 65](#_Toc25531216)

[**9.** **What is CSE?** 65](#_Toc25531217)

[**10.** **How to check Status of CSE?** 66](#_Toc25531218)

[**11.** **How a resource can be deployed using Resource Templates?** 66](#_Toc25531219)

[**12.** **How will you create VM using ARM template?** 66](#_Toc25531220)

[**13.** **on-premise AD account vs. Azure AD?** 66](#_Toc25531221)

[**14.** **What are Runbooks?** 66](#_Toc25531222)

[*.Net Core* 67](#_Toc25531223)

[**1.** **What are the advantages of ASP.Net Core over ASP.Net?** 67](#_Toc25531224)

[**2.** **Why Program class?** 67](#_Toc25531225)

[**3.** **Why Startup class?** 68](#_Toc25531226)

[*Database Architecture* 69](#_Toc25531227)

[**1.** **What are database cluster?** 69](#_Toc25531228)

[**2.** **In How many ways, replication can be done in database cluster?** 69](#_Toc25531229)

[**3.** **What is database partitioning?** 70](#_Toc25531230)

[**4.** **What is Materialized view?** 70](#_Toc25531231)

[*Hand-On* 71](#_Toc25531232)

[**1.** **How to create and deploy ARM template** 71](#_Toc25531233)

# *Azure Basics*

1. **What are Cloud models?**

There are 3 cloud models: Public cloud, Private cloud and, Hybrid cloud.

A **Public cloud** is owned by the cloud services provider (also known as a hosting provider). It provides resources and services to multiple organizations and users, who connect to the cloud service via a secure network connection, typically over the internet.

A **Private cloud** is owned and operated by the organization that uses the resources from that cloud. They create a cloud environment in their own datacenter, and provide self-service access to compute resources to users within their organization. The organization remains the owner, entirely responsible for the operation of the services they provide.

A **Hybrid cloud** combines both public and private clouds, allowing you to run your applications in the most appropriate location.

1. **What are the characteristics of Public cloud model?**

Public cloud models have the following characteristics:

* **Ownership**. This is the resources that an organization or end user uses. Examples include storage and processing power. Resources do not belong to the organization that is utilizing them, but rather they are owned and operated by a third party such as the cloud service provider.
* **Multiple End Users**. Public cloud modes may make their resources available to multiple organizations.
* **Public Access**. This provides access to the public.
* **Availability**. This is the most common cloud-type deployment model.
* **Connectivity**. Users and organizations are typically connected to the public cloud over the internet using a web browser.
* **Skills**. Public clouds do not require deep technical knowledge to set up and use its resources.

A common use case scenario is deploying a web application or a blog site on hardware and resources that are owned by a cloud provider. Using a public cloud in this scenario allows cloud users to get their website or blog up quickly, and then focus on maintaining the site without having to worry about purchasing, managing or maintaining the hardware on which it runs.

1. **What are the characteristics of Private cloud model?**

Private cloud models have the following characteristics:

* **Ownership**. The owner and user of the cloud services are the same.
* **Hardware**. The owner is entirely responsible for the purchase, maintenance, and management of the cloud hardware.
* **Users**. A private cloud operates only within one organization and cloud computing resources are used exclusively by a single business or organization.
* **Connectivity**. A connection to a private cloud is typically made over a private network that is highly secure.
* **Public access**. Does not provide access to the public.
* **Skills**. Requires deep technical knowledge to set up, manage, and maintain.

A use case scenario for a private cloud would be when an organization has data that cannot be put in the public cloud, perhaps for legal reasons. For example, they may have medical data that cannot be exposed publicly.

Another scenario may be where government policy requires specific data to be kept in-country or privately.

A private cloud can provide cloud functionality to external customers as well, or to specific internal departments such as Accounting or Human Resources.

1. **What are the characteristics of Hybrid cloud model?**

Hybrid cloud models have the following characteristics:

* **Resource location**. Specific resources run or are used in a public cloud, and others run or are used in a private cloud.
* **Cost and efficiency**. Hybrid cloud models allow an organization to leverage some of the benefits of cost, efficiency, and scale that are available with a public cloud model.
* **Control**. Organizations retain management control in private clouds.
* **Skills**. Technical skills are still required to maintain the private cloud and ensure both cloud models can operate together.

An example of a hybrid cloud usage scenario would be hosting a website in the public cloud and linking it to a highly secure database hosted in a private cloud.

Hybrid cloud scenarios can be useful when organizations have some things that cannot be put in a public cloud, possibly for legal reasons. For example, you may have medical data that cannot be exposed publicly.

Another example is one or more applications that run on old hardware that can’t be updated. In this case, you can keep the old system running locally in your private cloud, and connect it to the public cloud for authorization or storage.

1. **What are various Azure services?**

Azure services are: **Compute** Services, **Storage** Services, **Network** Services and **Database** Services

1. **What is Azure Security Center?**

Azure Security Center is a monitoring service that provides threat protection across all of your services both in Azure, and on-premises. Security Center can:

* Provide security recommendations based on your configurations, resources, and networks.
* Monitor security settings across on-premises and cloud workloads, and automatically apply required security to new services as they come online.
* Continuously monitor all your services, and perform automatic security assessments to identify potential vulnerabilities before they can be exploited.
* Use machine learning to detect and block malware from being installed on your virtual machines and services. You can also define a list of allowed applications to ensure that only the apps you validate are allowed to execute.
* Analyze and identify potential inbound attacks, and help to investigate threats and any post-breach activity that might have occurred.
* Provide just-in-time access control for ports, reducing your attack surface by ensuring the network only allows traffic that you require.

1. **What is Azure Key Vault?**

*Azure Key Vault* is a centralized cloud service for storing your applications' secrets. Key Vault helps you control your applications' secrets by keeping them in a single, central location and by providing secure access, permissions control, and access logging capabilities.

**Usage Scenarios**

* *Secrets management*. You can use Key Vault to securely store and tightly control access to tokens, passwords, certificates, Application Programming Interface (API) keys, and other secrets.
* *Key management*. You also can use Key Vault as a key management solution. Key Vault makes it easier to create and control the encryption keys used to encrypt your data.
* *Certificate management*. Key Vault lets you provision, manage, and deploy your public and private Secure Sockets Layer/ Transport Layer Security (SSL/ TLS) certificates for your Azure, and internally connected, resources more easily.
* *Store secrets backed by hardware security modules (HSMs)*. The secrets and keys can be protected either by software, or by FIPS 140-2 Level 2 validated HSMs.

**Key Vault benefits**

The benefits of using Key Vault include:

* *Centralized application secrets*. Centralizing storage for application secrets allows you to control their distribution, and reduces the chances that secrets may be accidentally leaked.
* *Securely stored secrets and keys*. Azure uses industry-standard algorithms, key lengths, and HSMs, and access requires proper authentication and authorization.
* *Monitor access and use*. Using Key Vault, you can monitor and control access to company secrets.
* *Simplified administration of application secrets*. Key Vault makes it easier to enroll and renew certificates from public Certificate Authorities (CAs). You can also scale up and replicate content within regions, and use standard certificate management tools.
* *Integrate with other Azure services*. You can integrate Key Vault with storage accounts, container registries, event hubs and many more Azure services.

1. **What are Azure Information Protection?**

Microsoft **Azure Information Protection** (**MSIP**) is a cloud-based solution that helps organizations classify and (optionally) protect its documents and emails by applying labels. Labels can be applied automatically (by administrators who define rules and conditions), manually (by users), or with a combination of both (where users are guided by recommendations).

An administrator can configure a label based on various factors, like Credit card information, PHIs etc. If a user saves a document containing sensitive information, the document is labeled.

Once the label has been applied (classified contents), the document can be tracked and control how the document can be used.

1. **What is Azure Firewall?**

A Firewall is a service that grants server access based on the originating IP address of each request. You create firewall rules that specify ranges of IP addresses. Only clients from these granted IP addresses will be allowed to access the server. Firewall rules, generally speaking, also include specific network protocol and port information. It is a fully stateful firewall as a service with built-in high availability and unrestricted cloud scalability.

Azure Firewall provides many features, including:

* Built-in high availability.
* Unrestricted cloud scalability.
* Inbound and outbound filtering rules.
* Azure Monitor logging.

*Azure Application Gateway* also provides a firewall, called the *Web Application Firewall* (WAF). However, WAF is different to Azure Firewall. WAF provides centralized, inbound protection for your web applications against common exploits and vulnerabilities. While in contrast, Azure Firewall provides outbound, network-level protection for all ports and protocols, and application-level protection for outbound HTTP/S. In addition, Azure Firewall provides inbound protection for non-HTTP/S protocols. Examples of non-HTTP/S protocols include: Remote Desktop Protocol (RDP), Secure Shell (SSH), and File Transfer Protocol (FTP). Azure Firewall's extended functionality make it suitable for different uses.

1. **What is Role Based Access Control (RBAC)?**

**Role-based access control (RBAC)** provides fine-grained access management for Azure resources, enabling you to grant users only the rights they need to perform their jobs. It helps you manage who has access to Azure resources, what they can do with those resources, and what areas they have access to. **RBAC** is provided at no additional cost to all Azure subscribers.

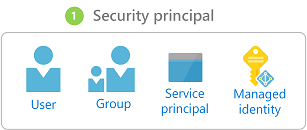
**Usage Scenarios**

* Allow one user to manage VMs in a subscription, and another user to manage virtual networks.
* Allow a database administrator (DBA) group to manage SQL databases in a subscription.
* Allow a user to manage all resources in a resource group, such as VMs, websites, and subnets.
* Allow an application to access all resources in a resource group.

1. **How RBAC works?**

The way you control access to resources using **RBAC** is to create role assignments. This is a key concept to understand – it’s how permissions are enforced. A role assignment consists of three elements: ***security principal****,* ***role definition***, and ***scope***.

**Security principal**



A **security principal** is an object that represents a user, group, service principal, or managed identity that is requesting access to Azure resources.

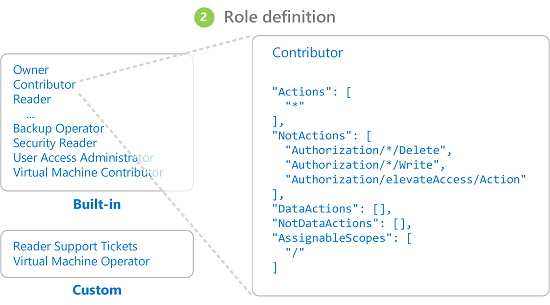
* **User** - An individual who has a profile in Azure Active Directory. You can also assign roles to users in other tenants.
* **Group** - A set of users created in Azure Active Directory. When you assign a role to a group, all users within that group have that role.
* **Service principal** - A security identity used by applications or services to access specific Azure resources. You can think of it as a *user identity* (username and password or certificate) for an application.
* **Managed identity** - An identity in Azure Active Directory that is automatically managed by Azure. You typically use [managed identities](https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview) when developing cloud applications to manage the credentials for authenticating to Azure services.

**Role definition**

A **role definition** is a collection of permissions. It's sometimes just called a **role**. A role definition lists the operations that can be performed, such as read, write, and delete. Roles can be high-level, like owner, or specific, like virtual machine reader.

Azure includes several [built-in roles](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles) that you can use. The following lists four fundamental built-in roles. The first three apply to all resource types.

* **Owner** - Has full access to all resources including the right to delegate access to others.



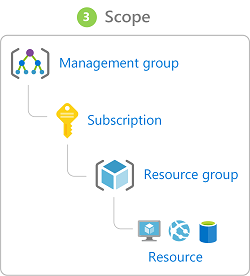
* **Contributor** - Can create and manage all types of Azure resources but can’t grant access to others.
* **Reader** - Can view existing Azure resources.
* **User Access Administrator** - Lets you manage user access to Azure resources.

The rest of the built-in roles allow management of specific Azure resources. For example, the [Virtual Machine Contributor](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#virtual-machine-contributor) role allows a user to create and manage virtual machines. If the built-in roles don't meet the specific needs of your organization, you can create your own [custom roles for Azure resources](https://docs.microsoft.com/en-us/azure/role-based-access-control/custom-roles).

Azure has data operations that enable you to grant access to data within an object. For example, if a user has read data access to a storage account, then they can read the blobs or messages within that storage account.

**Scope**

**Scope** is the set of resources that the access applies to. When you assign a role, you can further limit the actions allowed by defining a scope. This is helpful if you want to make someone a [Website Contributor](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#website-contributor), but only for one resource group.



In Azure, you can specify a scope at multiple levels: [*management group*](https://docs.microsoft.com/en-us/azure/governance/management-groups/overview), *subscription*, *resource group*, or *resource*. **Scopes** are structured in a parent-child relationship.

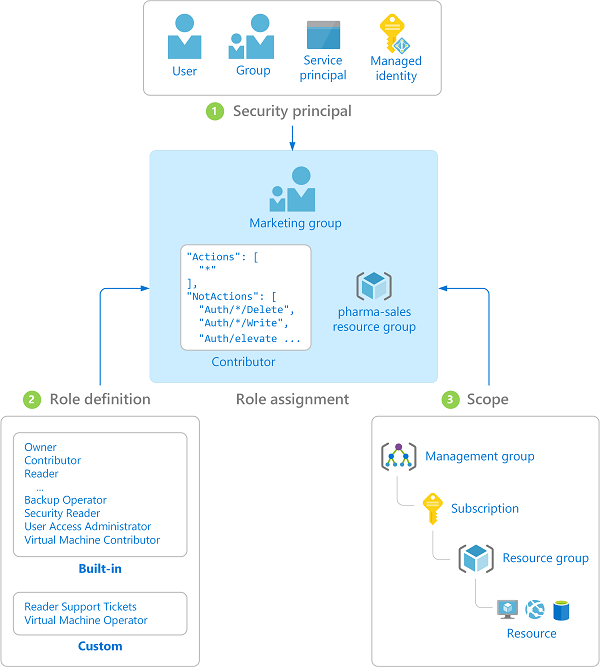
When you grant access at a parent scope, those permissions are inherited to the child scopes. For example:

* If you assign the [Owner](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#owner) role to a user at the management group scope, that user can manage everything in all subscriptions in the management group.
* If you assign the [Reader](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#reader) role to a group at the subscription scope, the members of that group can view every resource group and resource in the subscription.
* If you assign the [Contributor](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#contributor) role to an application at the resource group scope, it can manage resources of all types in that resource group, but not other resource groups in the subscription.

**Role assignments**

A **role assignment** is the process of attaching a role definition to a user, group, service principal, or managed identity at a particular scope for the purpose of granting access. Access is granted by creating a role assignment, and access is revoked by removing a role assignment.

The following diagram shows an example of a role assignment. In this example, the Marketing group has been assigned the [Contributor](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#contributor) role for the pharma-sales resource group. This means that users in the Marketing group can create or manage any Azure resource in the pharma-sales resource group. Marketing users do not have access to resources outside the pharma-sales resource group, unless they are part of another role assignment.



1. **What is Subscription?**

A **subscription** is an active agreement with Microsoft which is needed to provision resources in Microsoft Azure. Every subscription also has a trust relationship with an Azure Active Directory (**Azure AD**) instance. This means that it trusts that Azure AD to authenticate users, services and devices. A subscription will only trust one Azure AD, but we can have multiple subscriptions trust the same Azure AD.

Every resource provisioned in Azure is a child-resource to an Azure subscription. If the subscription is expired or stops, then those child-resources also stops.

1. **What is Azure AD?**

**Azure AD** is a robust, secure, multitenant directory service that provides identity and access management in the cloud. **Azure AD** is the directory store for many of Microsoft’s premium cloud services, such as Microsoft Office 365, Microsoft Dynamics CRM Online, Windows Intune, **and**, of course, **Azure**.

Much like **Windows Server Active Directory** provides identity and access management for on-premises solutions, **Azure AD** does so as a service available in Azure. However, instead of you assuming the responsibility of provisioning and configuring the multiple servers necessary for on-premises Active Directory, Microsoft is responsible for managing the entirety of the **Azure AD** infrastructure (high availability, scalability, disaster recovery, and so on). As a consumer of the **Azure AD** service (directory as a service), you decide what users and which of their related information should reside in the directory, who can use the information, and what applications have access to the information. **Azure AD** helps your employees sign in and access resources in:

* External resources, such as Microsoft Office 365, the Azure portal, and thousands of other SaaS applications.
* Internal resources, such as apps on your corporate network and intranet, along with any cloud apps developed by your own organization.

**Azure AD** should not be considered a full replacement for **Windows Server Active Directory**. Instead, it is a complementary service. If you already have Active Directory on-premises, the users and groups can be synchronized to your **Azure AD** directory by using **Azure AD Connect.**

An instance of **Azure AD** (*also called* ***Tenant***) can be created without having a **subscription**. In that case, the tenant users cannot use much of the resources of **Azure**, like VM creation, Web-App etc. An **Azure AD** instance hold many subscriptions. However, one subscription will belong to one Azure AD instance only.

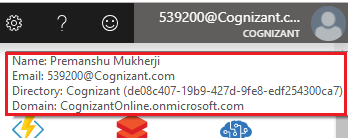
If the subscription expires, we lose access to all the other resources associated with the subscription. However, the **Azure AD** directory remains in Azure, letting us associate and manage the directory using a different **Azure subscription**.

1. **What is Tenant?**

A **Tenant** is a dedicated instance of Azure AD that you own when you sign up for a Microsoft cloud service (Azure, Office 365, and so on). Each tenant directory is isolated from the others in the service and designed to ensure user data is not accessible from other tenants, meaning others cannot access data in your directory unless an administrator grants explicit access.

Azure AD organizes objects like users and apps into groups called Tenants. Tenants allows an administrator of an organization to set policies on the users and the apps within the organization, to meet their security and operational properties.

To find the Tenant Id, hover over the account name on the top right corner.



<https://msandbu.org/overview-of-azure-active-directory-subscriptions-accounts-role-based-access-control/>

1. **What happens when a Tenant is deleted?**

When a **tenant** is deleted, all resources that are contained in the tenant are also deleted. We must prepare the tenant by minimizing its associated resources before we delete. Only an **Azure Active Directory** (Azure AD) **global administrator** can delete an **Azure AD** tenant from the portal.

We can't delete a tenant in **Azure AD** until it passes several checks. These checks reduce risk that deleting a tenant negatively impacts user access, such as the ability to sign in to Office 365 or access resources in Azure. For example, if the tenant associated with a subscription is unintentionally deleted, then users can't access the Azure resources for that subscription. The following explains the conditions that are checked:

* There can be no users in the tenant except one global administrator who is to delete the tenant. Any other users must be deleted before the tenant can be deleted. If users are synchronized from on-premises, then sync must be turned off, and the users must be deleted in the cloud tenant using the Azure portal or Azure PowerShell cmdlets.
* There can be no applications in the tenant. Any applications must be removed before the tenant can be deleted.
* There can be no multi-factor authentication providers linked to the tenant.
* There can be no subscriptions for any Microsoft Online Services such as Microsoft Azure, Office 365, or Azure AD Premium associated with the tenant. For example, if a default tenant was created for you in Azure, you cannot delete this tenant if your Azure subscription still relies on this tenant for authentication. Similarly, you can't delete a tenant if another user has associated a subscription with it.

1. **What is Multi-tenancy?**

The term "**Multi-tenancy**" refers to a software architecture in which a single instance of software runs on a server and serves multiple tenants. A tenant is a group of users who share a common access with specific privileges to the software instance. With a multitenant architecture, a software application is designed to provide every tenant a dedicated share of the instance - including its data, configuration, user management, tenant individual functionality and non-functional properties. Multi-tenancy contrasts with multi-instance architectures, where separate software instances operate on behalf of different tenants.

<https://cloudacademy.com/blog/creating-multi-tenant-applications-in-microsoft-azure/>

<http://www.andrewconnell.com/blog/azure-ad-what%E2%80%99s-the-difference-between-single-vs-multi-tenant>

<https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-convert-app-to-be-multi-tenant>

https://docs.microsoft.com/en-us/azure/active-directory/develop/single-and-multi-tenant-apps

1. **Terminology**

To better understand Azure AD and its documentation, you should review the following terms.

|  |  |
| --- | --- |
| **Term or concept** | **Description** |
| Azure subscription | Used to pay for Azure cloud services. You can have many subscriptions and they're linked to a credit card. |
| Azure tenant | A dedicated and trusted instance of Azure AD that's automatically created when your organization signs up for a Microsoft cloud service subscription, such as Microsoft Azure, Microsoft Intune, or Office 365. An Azure tenant represents a single organization. |
| Single tenant | Azure tenants that access other services in a dedicated environment are considered single tenant. |
| Multi-tenant | Azure tenants that access other services in a shared environment, across multiple organizations, are considered multi-tenant. |
| Azure AD directory | Each Azure tenant has a dedicated and trusted Azure AD directory. The Azure AD directory includes the tenant's users, groups, and apps and is used to perform identity and access management functions for tenant resources. |
| Azure AD account | An identity created through Azure AD or another Microsoft cloud service, such as Office 365. Identities are stored in Azure AD and accessible to your organization's cloud service subscriptions. This account is also sometimes called a Work or school account. |
| Custom domain | Every new Azure AD directory comes with an initial domain name, domainname.onmicrosoft.com. In addition to that initial name, you can also add your organization's domain names, which include the names you use to do business and your users use to access your organization's resources, to the list. Adding custom domain names helps you to create user names that are familiar to your users, such as alain@contoso.com. |
| Account Administrator | This classic subscription administrator role is conceptually the billing owner of a subscription. This role has access to the [Azure Account Center](https://account.azure.com/Subscriptions) and enables you to manage all subscriptions in an account. |
| Service Administrator | This classic subscription administrator role enables you to manage all Azure resources, including access. This role has the equivalent access of a user who is assigned the Owner role at the subscription scope.. |
| Owner | This role helps you manage all Azure resources, including access. This role is built on a newer authorization system called role-base access control (RBAC) that provides fine-grained access management to Azure resources. |
| Azure AD Global administrator | This administrator role is automatically assigned to whomever created the Azure AD tenant. Global administrators can perform all of the administrative functions for Azure AD and any services that federate to Azure AD, such as Exchange Online, SharePoint Online, and Skype for Business Online. You can have multiple Global administrators, but only Global administrators can assign administrator roles (including assigning other Global administrators) to users. Note This administrator role is called Global administrator in the Azure portal, but it's called Company administrator in Microsoft Graph API, Azure AD Graph API, and Azure AD PowerShell. |
| Microsoft account (also called, MSA) | Personal accounts that provide access to your consumer-oriented Microsoft products and cloud services, such as Outlook, OneDrive, Xbox LIVE, or Office 365. Your Microsoft account is created and stored in the Microsoft consumer identity account system that's run by Microsoft. |

1. **What is Domain and Domain model?**

The domain, is the problem to be addressed with a software effort. A domain can be decomposed into sub-domain which typically reflects some organizational structure. For example, e-commerce is a domain, and sub-domains are Product Catalog, Billing, Customer Information, Accounting etc.

A Domain model is an abstract of domain taking what’s necessary to satisfy requirements.

1. **What is Bounded-Context?**

While sub-domains delimit the applicability of domains, ***Bounded Context***  delimit the applicability of domain models. As such, the bounded context is within the solution space. Each sub domain should have explicit responsibilities so it has a boundary to limit their functionalities, the boundary will help sub domain focus to do only 1 thing and do well. This boundary is considered as ***Bounded Context*** of the sub domain. The bounded context will define:

* How many domain models needed for the sub domain?
* Which properties needed in the each model?
* Which functionalities needed in sub domain?

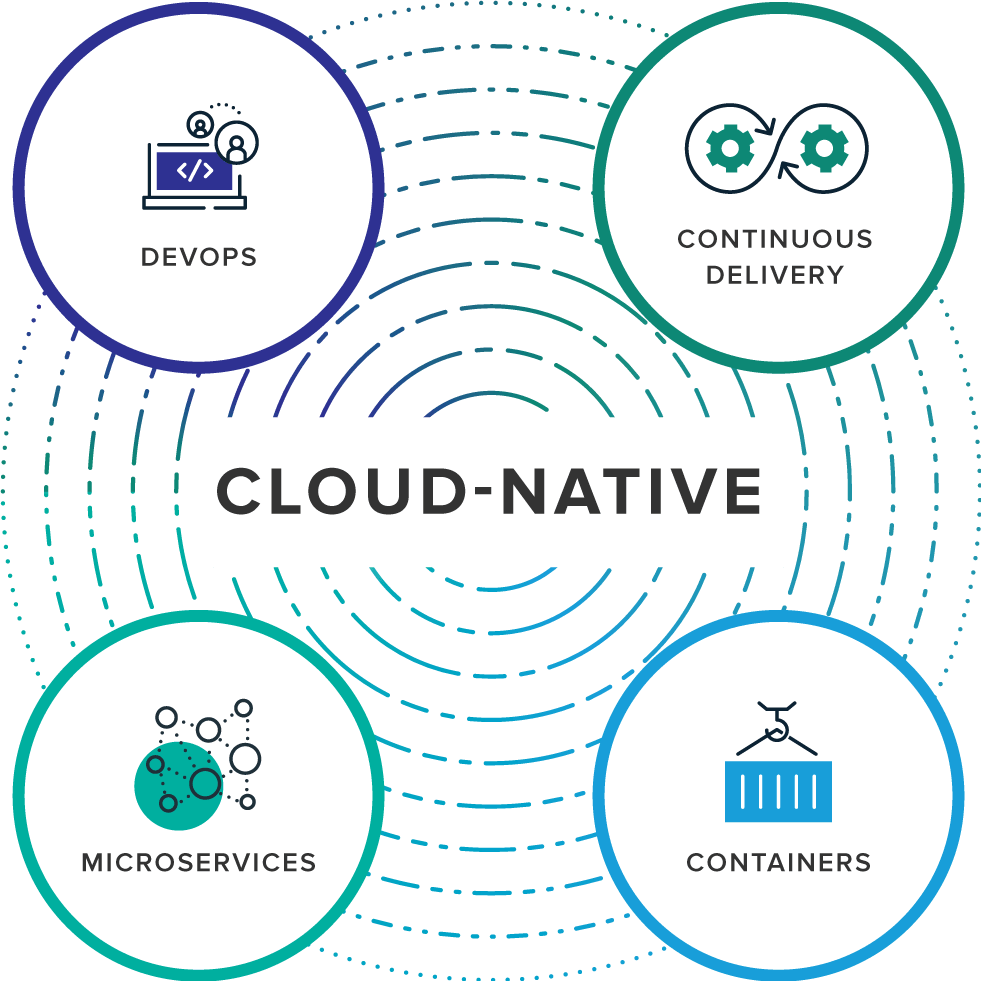
Ex: Shopping Cart sub domain needs models: Cart, Product, Customer Info... and contains functions to perform CRUD on the cart.

The Product and Customer model in the Shopping Cart sub domain maybe not the same with the models in Product Catalogs and Customer Profiles sub domain, they just contain necessary properties to display on Shopping.

1. **What are cloud native applications?**

Cloud-native is an approach to building and running applications that exploits the advantages of the cloud computing model.

Cloud Native promotes five architectural principles:



* Use infrastructure-as-a-service: run on servers that can be flexibly provisioned on demand.
* Design systems using, or evolve them towards, a Micro-services architecture: individual components are small and decoupled.
* Automate and encode: replace manual tasks with scripts or code.
* Containerize: package processes with their dependencies making them easy to test, move and deploy.
* Orchestrate: abstract away individual servers in production using off-the-shelf management and orchestration tools.

1. **What are the factors of cloud native applications (Micro-services)?**

Cloud native applications adheres to 12 factors. These factors are:

*Ref: https://12factor.net*

1. Codebase: One codebase tracked in revision control, many deploys
2. Dependencies: Explicitly declare and isolate dependencies
3. Configuration: Store configuration in the environment
4. Backing Services: Treat backing services as attached resources
5. Build, release, run: Strictly separate build and run stages
6. Processes: Execute the app as one or more stateless processes
7. Port binding: Export services via port binding
8. Concurrency: Scale out via the process model
9. Disposability: Maximize robustness with fast startup and graceful shutdown
10. Dev/prod parity: Keep development, staging, and production as similar as possible
11. Logs: Treat logs as event streams
12. Admin processes: Run admin/management tasks as one-off processes
13. **What are the key attributes of cloud native applications?**

The 10 key attributes of cloud native applications are:

1. Packaged as lightweight containers
2. Developed with best-of-breed languages
3. Designed as loosely coupled micro-services
4. Centered around APIs for interaction and collaboration
5. Architected with a clean separation of stateless and stateful services
6. Isolated from server and operating system dependencies
7. Deployed on self-service, elastic, cloud infrastructure
8. Managed through agile DevOps processes
9. Automated capabilities
10. Defined, policy driven resources allocation
11. **What is cloud orchestration?**

Cloud computing handles a prodigious amount of data and processes across multiple systems. This heterogeneity makes manageability and coherence a major challenge in cloud computing. The solution to this challenge lies in implementing a tool or a product that can manage these interconnections and interactions among cloud connected units. We call this solution, **cloud orchestration**. A cloud orchestration involves the end-to-end automation and coordination of multiple processes to deliver a desired service to its clients.

**Cloud orchestration** enforces a workflow order to automated tasks, and enhances security with identity and access management policies. It can also unite disparate cloud deployments to work together for a given workload.

**Cloud orchestration** is the use of programming technology to manage the interconnections and interactions among workloads on public and private cloud infrastructure. It connects automated tasks into a cohesive workflow to accomplish a goal, with permissions oversight and policy enforcement.

1. **What is the difference between Cloud orchestration and automation?**

**Automation** is a subset of **orchestration**, which means that orchestration provides coordination among and across many automated activities. Automation focuses on making one task repeatable rapidly with minimal operator intervention; orchestration on the whole.

Given the many moving parts in cloud, orchestration brings high availability, scaling, failure recovery, dependency management, and numerous other tasks and attributes into a single process that can tremendously reduce staff effort. Orchestration also provides visibility into resources and processes that simple cloud automation lacks; for example, a business can regulate capacity via preset resource templates for application deployment and track who requests what resources.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Cloud automation** | **Cloud orchestration** |
| No. of tasks involved | It involves a single task | It is concerned with combining multiple such tasks into workflows |
| Pre-requisites for performance | Automated tasks must be performed in a definite sequence, under strict security guidelines, be granted permissions etc. | Orchestration ensures the performance of each of the automated tasks in a definite order w.r.t one another, within a workflow |
| Error-proneness | Cloud automation involves huge amount of manual coding by an engineer, which implies fat fingering the commands. The problem arising in such situations is that the fear of error looms large | Cloud orchestration helps evade error-proneness by using optimizing the amount of coding required. This it does by using the automated tasks as building blocks, that can be reused, instead of redundant coding. |
| Resource utilization | Automated tasks exist as standalone entities, thus employing resources independently. This leads to resource wastefulness. | Cloud orchestration enumerates the resources, IAM roles, instance types, etc. to help save time and deliver more accurate results |
| Example | Launching a web server, configuring a web server | Combining such automated tasks into a single workflow to meet client requests |

Cloud orchestration focuses on process optimization. One of the simplest ways to do so is to remove the repetitive steps to eliminate redundancy. It identifies the automated tasks that make up a process, and ensures their reusability across operations. Orchestration, therefore, takes advantage of the building blocks (read as automated tasks) by reusing them.

1. **What is Auto-scaling?**

Autoscaling is the process of dynamically allocating resources to match performance requirements. As the volume of work grows, an application may need additional resources to maintain the desired performance levels and satisfy service-level agreements (SLAs). As demand slackens and the additional resources are no longer needed, they can be de-allocated to minimize costs.

Autoscaling takes advantage of the elasticity of cloud-hosted environments while easing management overhead. It reduces the need for an operator to continually monitor the performance of a system and make decisions about adding or removing resources.

There are two main ways that an application can scale:

* **Vertical scaling**, also called scaling up and down, means changing the capacity of a resource. For example, you could move an application to a larger VM size. Vertical scaling often requires making the system temporarily unavailable while it is being redeployed. Therefore, it's less common to automate vertical scaling.
* **Horizontal scaling**, also called scaling out and in, means adding or removing instances of a resource. The application continues running without interruption as new resources are provisioned. When the provisioning process is complete, the solution is deployed on these additional resources. If demand drops, the additional resources can be shut down cleanly and deallocated.

<https://docs.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling>

1. **What are the strategies for Auto-scaling?**

An *Auto-scaling* strategy typically involves the following pieces:

* Instrumentation and monitoring systems at the application, service, and infrastructure levels. These systems capture key metrics, such as response times, queue lengths, CPU utilization, and memory usage.
* Decision-making logic that evaluates these metrics against predefined thresholds or schedules, and decides whether to scale.
* Components that scale the system.
* Testing, monitoring, and tuning of the *Autoscaling* strategy to ensure that it functions as expected.

Azure provides built-in *Autoscaling* mechanisms that address common scenarios. If a particular service or technology does not have built-in *Autoscaling* functionality, or if you have specific *Autoscaling* requirements beyond its capabilities, you might consider a custom implementation. A custom implementation would collect operational and system metrics, analyze the metrics, and then scale resources accordingly.

1. **What are the points to consider when using Azure *Auto-scale*?**

Consider the following points when using Azure *Auto-scale*:

* Consider whether you can predict the load on the application well enough to use scheduled Auto-scaling, adding and removing instances to meet anticipated peaks in demand. If this isn't possible, use reactive *Auto-scaling* based on runtime metrics, in order to handle unpredictable changes in demand. Typically, you can combine these approaches. For example, create a strategy that adds resources based on a schedule of the times when you know the application is most busy. This helps to ensure that capacity is available when required, without any delay from starting new instances. For each scheduled rule, define metrics that allow reactive *Auto-scaling* during that period to ensure that the application can handle sustained but unpredictable peaks in demand.
* It's often difficult to understand the relationship between metrics and capacity requirements, especially when an application is initially deployed. Provision a little extra capacity at the beginning, and then monitor and tune the *Auto-scaling* rules to bring the capacity closer to the actual load.
* Configure the *Auto-scaling* rules, and then monitor the performance of your application over time. Use the results of this monitoring to adjust the way in which the system scales if necessary. However, keep in mind that *Auto-scaling* is not an instantaneous process. It takes time to react to a metric such as average CPU utilization exceeding (or falling below) a specified threshold.
* *Auto-scaling* rules that use a detection mechanism based on a measured trigger attribute (such as CPU usage or queue length) use an aggregated value over time, rather than instantaneous values, to trigger an *Auto-scaling* action. By default, the aggregate is an average of the values. This prevents the system from reacting too quickly, or causing rapid oscillation. It also allows time for new instances that are auto-started to settle into running mode, preventing additional *Autoscaling* actions from occurring while the new instances are starting up. For Azure Cloud Services and Azure Virtual Machines, the default period for the aggregation is 45 minutes, so it can take up to this period of time for the metric to trigger *Auto-scaling* in response to spikes in demand. You can change the aggregation period by using the SDK, but be aware that periods of fewer than 25 minutes may cause unpredictable results. For Web Apps, the averaging period is much shorter, allowing new instances to be available in about five minutes after a change to the average trigger measure.
* If you configure *Auto-scaling* using the SDK rather than the portal, you can specify a more detailed schedule during which the rules are active. You can also create your own metrics and use them with or without any of the existing ones in your *Auto-scaling* rules. For example, you may wish to use alternative counters, such as the number of requests per second or the average memory availability, or use custom counters that measure specific business processes.
* When *Auto-scaling* Service Fabric, the node types in your cluster are made of VM scale sets at the backend, so you need to set up auto-scale rules for each node type. Take into account the number of nodes that you must have before you set up auto-scaling. The minimum number of nodes that you must have for the primary node type is driven by the reliability level you have chosen. For more info, see scale a Service Fabric cluster in or out using auto-scale rules.

# *IAAS*

1. **What are Azure Compute Services?**

*Azure compute* is an on-demand computing service for running cloud-based applications. It provides computing resources such as disks, processors, memory, networking and operating systems. The resources are available on-demand and can typically be made available in minutes or even seconds. You pay only for the resources you use and only for as long as you're using them.

There are two common service types for performing compute in Azure: **Virtual** machines, and **Containers**.

Compute service includes **VM**s, **Scale sets**, **App** services, **Functions**, **Containers** instances, **Kubernetes**.

1. **What is Availability set?**

Availability sets are a way for you to ensure your application remains online if a high-impact maintenance event is required, or a hardware a failure occurs. Availability sets are made up of **update** domains and **fault** domains.

* **Update domains (UD)**. When a maintenance event occurs (such as a performance update or critical security patch applied to the host), the update is sequenced through update domains. Sequencing updates using update domains ensures that the entire datacenter isn't unavailable during platform updates and patching. Update domains are a logical section of the datacenter, and they are implemented with software and logic.
* **Fault domains (FD)**. Fault domains provide for the physical separation of your workload across different hardware in the datacenter. This includes power, cooling, and network hardware that supports the physical servers located in server racks. In the event the hardware that supports a server rack becomes unavailable, only that rack of servers would be affected by the outage.

1. **What is Azure VM scale sets?**

Virtual machine scale sets are an Azure compute resource that you can use to deploy and manage a set of identical VMs. With all VMs configured the same, VM scale sets are designed to support true auto-scale—no pre-provisioning of VMs is required—and as such makes it easier to build large-scale services targeting big compute, big data, and containerized workloads. So, as demand goes up more virtual machine instances can be added, and as demand goes down virtual machines instances can be removed. The process can be manual, automated, or a combination of both.

1. **What are Regions, Geography and Availability zones?**

A **region** is a set of datacenters deployed within a latency-defined perimeter and connected through a dedicated regional low-latency network. Microsoft Azure is made up of datacenters located around the globe. These datacenters are organized and made available to end users by region.

A **geography** is a discrete market, typically containing two or more regions, that preserves data residency and compliance boundaries.

Geographies allow customers with specific data-residency and compliance needs to keep their data and applications close. Geographies are fault-tolerant to withstand complete region failure through their connection to our dedicated high-capacity networking infrastructure.

**Availability Zones** are physically separate locations within an Azure region. Each Availability Zone is made up of one or more datacenters equipped with independent power, cooling, and networking.

Availability Zones allow customers to run mission-critical applications with high availability and low-latency replication.

1. **Benefits of Resource Group.**

A **resource group** is a unit of management for your resources in Azure. You can think of your resource group as a container that allows you to aggregate and manage all the resources required for your application in a single manageable unit. This allows you to manage the application collectively over its life cycle, rather than manage components individually.

You can manage and apply the following resources at resource group level:

* Metering and billing
* Policies
* Monitoring and alerts
* Quotas
* Access control

1. **What are Azure Storage?**

Microsoft **Azure Storage** is a Microsoft-managed service that provides durable, scalable, and redundant storage. Microsoft takes care of maintenance and handles critical problems for you. An Azure subscription can host up to 100 storage accounts, each of which can hold 500 TB.

There are two types of Azure storage account:

1. **General purpose storage** account consists of four data services: **Blob storage**, **File storage**, **Table storage**, and **Queue storage**.
2. **Blob storage account**: The Blob storage account is a specialized storage account used to store block blobs and append blobs. You can’t store page blobs in these account, therefore, you can’t store VHD files. These accounts allow you to set an access tier to Hot or Cool; the tier can be changed at any time. The **hot access tier** is used for files that are accessed frequently. For blobs stored in the hot access tier, you pay a higher cost for storing the blobs, but the cost for accessing the blobs is much lower. The **cool access tier** is used for files that are accessed infrequently. For blobs stored in the cool access tier, you pay a higher cost for accessing the blobs, but the cost of storage is much lower.
3. **What are the benefits of Message Queue?**

In modern cloud architecture, applications are decoupled into smaller, independent building blocks that are easier to develop, deploy and maintain. Message Queues. provide communication and coordination for these distributed applications.

Message queues can significantly simplify coding of decoupled applications, while improving performance, reliability and scalability.

The benefits are:

* **Better performance**- Message Queues enables asynchronous communication, which means that the endpoints that are producing and consuming messages interact with the queue, not each other. Producers can add requests to the queue without waiting for them to be processed. Consumers process messages only when they are available. No component in the system is ever stalled waiting for another, optimizing data flow.
* **Increased reliability**- Queues make your data persistent, and reduce the errors that happen when different parts of your system go offline. By separating different components with Message Queues , you create more fault tolerance. If one part of the system is ever unreachable, the other can still continue to interact with the queue. The queue itself can also be mirrored for even more availability.
* **Granular scalability**- Message queues make it possible to scale precisely where you need to. When workloads peak, multiple instances of your application can all add requests to the queue without risk of collision. As your queues get longer with these incoming requests, you can distribute the workload across a fleet of consumers. Producers, consumers and the queue itself can all grow and shrink on demand.
* **Simplified decoupling**- Message queues remove dependencies between components and significantly simplify the coding of decoupled applications. Software components aren’t weighed down with communications code and can instead be designed to perform a discrete business function.  
  Message queues are an elegantly simple way to decouple distributed systems, whether you're using monolithic applications, micro-services or serverless architectures.

1. **Queue storage Vs. Azure Service Bus?**

Azure storage Queues are a part of Azure storage infrastructure feature, a simple REST based Get/Put/Peek interface, offering persistent messaging within and between services.

Azure Service Bus queues are part of Azure messaging infrastructure that supports queuing as well as to publish subscribe web service remoting and integration patterns.

* Queue storage is used when we require logs of all of the transactions executed against the queues.
* Queue storage can store messages over 80 GB in a queue, whereas Service Bus is used where the messages are less than 80 GB.
* Azure service bus queue supports brokered messaging, where senders and receivers do not need to be online at the same time.
* Azure service bus guarantees FIFO pattern. Messages are delivered in the order they come. Storage Queues can be, sometimes, out of order.
* Cost wise Azure queue is cheaper than a service bus queue.
* Azure storage queues have 10 ms average latency whereas service bus queues have 20-25 ms latency.
* Service bus queues supports role based access model to the queues, and different rights/permissions for senders and receivers.
* Service bus queues supports Pub/Sub model.
* Storage queues offer a visibility timeout that you can set upon the enqueuing or dequeuing of a message. In addition, you can update a message with different lease values at run-time, and update different values across messages in the same queue. Service Bus lock timeouts are defined in the queue metadata; however, you can renew the lock by calling the RenewLock method.
* Storage queues does not support duplicate detection, message groups, automated dead lettering, state management.
* Service bus queue does not support purge queue function, server-side transaction log.

1. **What is Blob storage?**

The word blob is an acronym for binary large object. Blobs are basically files like those that you store on your computer (or tablet, mobile device, etc.). They can be pictures, Microsoft Excel files, HTML files, virtual hard disks (VHDs)—pretty much anything.

The **Azure Blob service** gives you the ability to store files and access them from anywhere in the world by using URLs, the REST interface, or one of the Azure SDK storage client libraries. Storage client libraries are available for multiple languages, including .NET, Node.js, Java, PHP, Ruby, and Python. To use the Blob service, you have to create a storage account. Once you have a storage account, you can create containers, which are similar to folders, and then put blobs in the containers. You can have an unlimited number of containers in a storage account and an unlimited number of blobs in each container, up to the maximum size of a storage account, which is 500 TB. The Blob service supports only a single-level hierarchy of containers; in other words, containers cannot contain other containers.

Azure Storage supports three kinds of blobs: **Block blobs**, **Page blobs**, and **Append blobs**.

* Block blobs are used to hold ordinary files up to 195 GB in size (4 MB × 50,000 blocks). The primary use case for block blobs is the storage of files that are read from beginning to end, such as media files or image files for websites. They are named block blobs because files larger than 64 MB must be uploaded as small blocks, which are then consolidated (or committed) into the final blob.
* Page blobs are used to hold random-access files up to 1 TB in size. Page blobs are used primarily as the backing storage for the VHDs used to provide durable disks for Azure Virtual Machines (Azure VMs), the IaaS feature in Azure Compute. They are named page blobs because they provide random read/write access to 512-byte pages.
* Append blobs are made up of blocks like block blobs, but they are optimized for append operations. These are frequently used for logging information from one or more sources into the same blob. For example, you might write all of your trace logging to the same append blob for an application running on multiple VMs. A single append blob can be up to 195 GB.

Blobs are addressable through a URL, which has the following format:

https://[storage account name]/blob.core.windows.net/[container]/[blob name]

1. **What is File storage?**

The Azure Files service enables you to set up highly available network file shares that can be accessed by using the standard **Server Message Block** (**SMB**) protocol.  This allows clients, such as Windows Explorer, to connect and browse File storage (such as a typical network file share). This means that multiple VMs can share the same files with both read and write access. The files can also be accessed using the REST interface or the storage client libraries. The Files service removes the need for you to host your own file shares in an Azure VM and go through the tricky configuration required to make it highly available.

One thing that’s really special about Azure file shares versus file shares on-premises is that you can access the file from anywhere by using a URL that points to the file (similar to the blob storage URL displayed above). To do this, you have to append a shared access signature (SAS). We’ll talk more about shared access signatures in the section on Security.

File shares can be used for many common scenarios:

* Many on-premises applications use file shares; this makes it easier to migrate those applications that share data to Azure. If you mount the file share to the same drive letter that the on-premises application uses, the part of your application that accesses the file share should work without any changes.
* Configuration files can be stored on a file share and accessed by multiple VMs.
* Diagnostic logs, metrics, crash dumps, etc. can be saved to a file share to be processed and analyzed later.
* Tools and utilities used by multiple developers in a group can be stored on a file share to ensure that everyone uses the same version and that they are available to everyone in the group.

To make the share visible to a VM, you just mount it as you would any other file share, and then you can access it through the network URL or the drive letter to which it was assigned. The network URL has the format \\[storage account name].file.core.windows.net\[share name]. After the share is mounted, you can access it using the standard file system APIs to add, change, delete, and read the directories and files.

To create or view a file share or upload or download files to it from outside Azure, you can use the Azure portal, PowerShell, the Azure Command-Line Interface (CLI), the REST APIs, one of the storage client libraries, or AzCopy, a command-line tool provided by Microsoft.

1. **File Storage Vs. Blob storage?**

**Azure file** provides an SMB interface, client libraries, and a REST interface that allows access from anywhere to stored files.

When to use:

* You want to "lift and shift" an application to the cloud which already uses the native file system APIs to share data between it and other applications running in Azure.
* You want to store development and debugging tools that need to be accessed from many virtual machines.

**Azure Blob** provides client libraries and a REST interface that allows unstructured data to be stored and accessed at a massive scale in block blobs.

Also supports Azure Data Lake Storage Gen2 for enterprise big data analytics solutions.

When to use:

* You want your application to support streaming and random access scenarios.
* You want to be able to access application data from anywhere.
* You want to build an enterprise data lake on Azure and perform big data analytics.

**Pricing**: Blob storage is much cheaper than file storage.

**Portability**: With blob storage if you decide to migrate to a diff platform in future you may have to change your app code but with File storage you can migrate your app to any other platform that supports SMB (assuming you are using native file system APIs in your app).

1. **Table storage vs. SQL based storage?**

**Azure SQL** database is a relational database service. Similar to traditional SQL, it has frameworks and tools to access the data.

**Table storage** is a No-SQL key-value store. It is a non-relation, un-schematic data at a low cost for applications with simplified data-access patterns.

Despite differences, both offering are highly available, managed services with 99.99% monthly SLA.

Notable differences are:

**Azure SQL** is a relational database, so joins, ACID transactions, stored procedures etc. are supported whereas **Table Storage** does not support joins, stored procedures, however transactions are supported. Moreover, the rows within the same table can have different structures, thus the retrieval of simple relational data is efficient.

**Table storage** offers 500 Terabytes of data, while **Azure SQL** provides 150GB max.

**Azure SQL** is used when the requirement is for data processing over schematic, highly structured data set with relationship.

**Table store** is preferred where volume of data is more and does not involve complex relationship.

1. **What is unmanaged disk?**

The legacy way of deploying a virtual machine (VM) was as follows:

1. **Create a storage account**. This is a way to carve out a slice of storage from Azure. This was used to store the virtual hard-disk (VHD) of the VM, and is used as a disk.
2. **Create a Virtual machine**. The OS disk is placed in the azure storage account.
3. Add data disks. They store data related to performance, support, or management. These data disks are also kept in a storage account.

This means that there is a two-step process for creating a new virtual hard disk. This way of provisioning VHD is known as **un-managed VHD**.

For best performance, reliability, scalability and access control, the recommendation is to use Azure Managed Disks for most virtual machine configurations. Unmanaged disks are used if you need to support certain classic scenarios or want to manage disk VHDs in your own storage account.

https://www.petri.com/introduction-azure-vm-managed-disks

1. **What is managed disk?**

An Azure **managed disk** is a virtual hard disk (**VHD**), just like **unmanaged disk**. However, with **managed disk**, Microsoft handles the storage on the user’s behalf. Users do not need to create storage accounts for virtual machine disks. Azure does that for the users. Azure managed disks are stored as page blobs, which are a random IO storage object in Azure.

The advantages of Managed disk over unmanaged one are:

* Reduced Management
* Increased reliability
* More security
* Easier scaling out
* Pricing
* Snapshots and
* Back-up

1. **What are the types of VHDs in Azure?**

There are two types of VHDs in Azure:

* **Image**: A **VHD** that is a template for the creation of a new Azure VM. As a template, it does not have settings such as a machine name, administrative user, and so on.
* **Disk**: A possibly bootable **VHD** that can be used as a mountable disk for a VM. There are two types of disks: an OS disk and a data disk.

All durable disks (the OS disk and data disks), are

* fixed-format **VHDs**
* backed by page blobs in Azure Storage i.e. stored as page blobs in Azure Storage
* used by the virtual machine to store durable data.

Therefore, the disks inherit the benefits of blob storage: high availability, durability, and geo-redundancy options. Blob storage provides a mechanism by which data can be stored safely for use by the VM. The disks can be mounted as drives on the VM. The Azure platform will hold an infinite lease on the page blob to prevent accidental deletion of the page blob containing the VHD, the related container, or the storage account.

Apart from an OS disk and addition data disk, there’s also a **temporary disk**. This is the physical disk that’s inside the chassis of the server. In case of any unfortunate event with the VM, the data in the temporary disk is also lost.

1. **What is managed disk snapshot and image?**

A managed disk **snapshot** is a read-only full copy of a managed disk (not Virtual Machines) that is stored as a standard managed disk by default. With snapshots, you can back up your managed disks at any point in time. These snapshots exist independent of the source disk and can be used to create new managed disks. They are billed based on the used size. For example, if you create a snapshot of a managed disk with provisioned capacity of 64 GiB and actual used data size of 10 GiB, that snapshot is billed only for the used data size of 10 GiB.

Managed disks also support creating a managed custom image. You can create an image from your custom VHD in a storage account or directly from a generalized (sysprepped) VM. This process captures a single image. This image contains all managed disks associated with a VM, including both the OS and data disks. This managed custom image enables creating hundreds of VMs using your custom image without the need to copy or manage any storage accounts.

The difference snapshot and image is that the snapshot is for only one disk whereas an image takes all the disks attached to the VM. If a VM has just one OS Disk, the either snapshot or image will do. However, if the disk increases then taking an image should be considered.

1. **What is Storage redundancy?**

The data in your Microsoft Azure storage account is always replicated to ensure durability and high availability. Azure Storage copies your data so that it is protected from planned and unplanned events, including transient hardware failures, network or power outages, and massive natural disasters. You can choose to replicate your data within the same data center, across zonal data centers within the same region, or across geographically separated regions.

There are four choices for redundancy: Locally Redundant Storage (**LRS**), Geo-Redundant Storage (**GRS**), Read-Access Geo-Redundant storage (**RA-GRS**) and Zone Redundant Storage (**ZRS**).

1. **What is Subnet and Subnet masking?**

A subnetwork or subnet is a logical partitioning of an IP network. In azure, subnets within same VNet can communicate with each other. The computers belonging to same subnet are addressed with an identical most-significant bit group. This results in the logical division of an IP address into two parts: Network number (or routing prefix) and host identifier (rest identifier).

For an example, the address 198.51.100.10 and 198.51.100.50 might or might not belong to same subnet. However, to conform say that, we need either of the two additional information i.e. the **subnet mask** or the **Classless Inter-Domain Routine** (**CIDR**).

In IPv4, a network is characterized by its **subnet mask** or net-mask, which is a Bit-mask that when applied by a **bitwise AND** operation to any IP address in the network, yields the routing prefix. So, 255.255.255.0 is the subnet mask for 198.51.100.x.

198.51.100.10 => 11000000.00110011.01100100.00001010

198.51.100.50 => 110000900.00110011.01100100.00110010

So, we can say that 198.51.100.10 and 198.51.100.50, belongs to

* + - * + same subnet if the mask is **255.255.255.192/26** (11111111.11111111.11111111.**11**000000)
        + different subnet if the mask is **255.255.255.240/28**

(11111111.11111111.11111111.**1111**0000)

In CIDR format, the routing prefix can be expressed as the first network address followed by a slash character.

For example, 198.58.100.0/x. where x is the position of left most ‘1’ bit.

So, 198.58.100.0/23 translates to the range of 198.58.100 to 198.58.101.255. A total of 512 host can be accommodated within this subnet. The network mask will be 255.255.254.0/23.

1. **What are the benefits of Sub-netting?**

There are five Sub-netting benefits.

1. Improve network performance and speed
   * + - A single broadcasting packet reaches out to every device. Large number of connected device negatively impacts the performance of the switches.
       - Spamming every device leads to spamming irrelevant devices, which can stain a network’s capacity, causing it to collapse.
       - Sub-netting keeps the information within the subnet. This allows other subnets to maximize their speed and effectiveness.
       - Sub-netting divides the network’s broadcast domains, thus enabling better control traffic flow and increased network performance.
2. Reduce traffic congestion: Sub-netting ensures that traffic destined for a device within a subnet stays in that subnet, which reduces congestion. Through strategic placement of subnets, you can help reduce your network’s load and more efficiently route traffic.
3. Boost network security
4. Control network growth and,
5. Ease administration
6. **What is Azure Virtual network?**

Azure **Virtual Network** enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks. A virtual network is scoped to a single region; however, multiple virtual networks from different regions can be connected together using **Virtual Network Peering**.

Azure virtual network provides following capabilities

* Isolation and segmentation
* Communicate with Internet
* Communicate with Azure resources
* Communicate with on-premises resources
* Filtering network traffic
* Route network traffic, and
* Connect to other virtual network

1. **How the communication with the internet happens in Azure virtual network?**

All resources in a virtual network can communicate outbound to the internet, by default. However, if we want to communicate inbound to a resource, via the internet, then it is done by assigning a public IP address or a public Load Balancer. We can also use public IP or public Load Balancer to manage our outbound connections.

A deployment in Azure can communicate with endpoints outside Azure, in the public IP address space. When an instance initiates an outbound flow to a destination in the public IP address space, Azure dynamically maps the private IP address to a public IP address. After this mapping is created, return traffic for this outbound originated flow can also reach the private IP address where the flow originated.

If you don't want a VM to communicate with endpoints outside Azure in public IP address space, you can use network security groups (NSGs) to block access as needed.

The default outbound behavior of VM is applicable if there is no standard load balancer applied. Otherwise we need to define the outbound connection behavior.

1. **How the communication between the azure resources happens in Azure virtual network?**

Azure resources communicate securely with each other in one of the following ways

* **Through a virtual network**: You can deploy VMs, and several other types of Azure resources to a virtual network, such as Azure App Service Environments, the Azure Kubernetes Service (AKS), and Azure Virtual Machine Scale Sets.
* **Through a virtual network service endpoint**: Extend your virtual network private address space and the identity of your virtual network to Azure service resources, such as Azure Storage accounts and Azure SQL databases, over a direct connection. Service endpoints allow you to secure your critical Azure service resources to only a virtual network.

1. **What are the various connectivity available in Azure?**

To connect to on-premises computers and networks, Azure offers following options:

* **Point-to-Site virtual private network (VPN)**: This is established between a VNet and a single computer in other network. Each of the computer has to establish the connectivity with the VNet individually. The communication happens via encrypted tunnel over the internet. Good for initial setups and the situations where changes to the existing network is not required.
* **Site-to-Site VPN**: (S2S) Established between on-premises and an Azure VPN gateway that is deployed in a virtual network. This connection requires a gateway to be installed. This type of connection enables any authorized on-premises resource to access a virtual network. The communication happens via encrypted tunnel over the internet.
* **Azure Express routes**: **ExpressRoute** lets you extend your on-premises networks into the Microsoft cloud over a private connection facilitated by a connectivity provider. This connection is private. Traffic does not go over the public internet, unlike P2P and S2S. This allows **ExpressRoute** connections to offer more reliability, faster speeds, lower latencies, and higher security than typical connections over the Internet.
* **Hybrid Connection**: Hybrid Connections is both a service in Azure and a feature in Azure App Service. As a service, it has uses and capabilities beyond those that are used in App Service. It allows application deployed in Azure to access applications on premises. And, unlike VPN, it is free.

To connect two Azure **Virtual** **networks**, following are the options:

* **Site-to-Site**: It is for the complicated network configuration. The local network gateways are created manually. The local network gateway for each VNet treats the other VNet as a local site. This lets you specify additional address space for the local network gateway in order to route traffic. If the address space for a VNet changes, you need to update the corresponding local network gateway to reflect the change. It does not automatically update.
* **VNet-to-VNet**: An easy way to connect two **VNets**. This is similar to S2S. This connectivity requires gateway to be installed in both the **VNets**. The difference between the S2S and V2V lies in the way the local network gateway is configured. When you create a VNet-to-VNet connection, you do not see the local network gateway address space, unlike S2S. It is automatically created and populated. If you update the address space for one VNet, the other VNet automatically knows to route to the updated address space. Creating a VNet-to-VNet connection is typically faster and easier than creating a Site-to-Site connection between **VNets**.
* **VNet peering**: enables to seamlessly connect Azure virtual networks. Once peered, the virtual networks appear as one, for connectivity purpose. The traffic between virtual machines in the peered virtual networks is routed through the Microsoft backbone infrastructure, much like traffic is routed between virtual machines in the same virtual network, through private IP addresses only. Azure supports:
  + VNet peering - connecting VNets within the same Azure region
  + Global VNet peering - connecting VNets across Azure regions

VNet peering does not need a gateway unlike VNet2VNet. The two virtual machines can communicate at the speed of their NIC (around 25GBPS) where as VNet2VNet works at a maximum speed of 100 Mbps.

1. **How filtering Traffic network works in Azure?**

A network traffic between different subnets can be filtered using either or both of the options:

* **Security group**s: Network security group and application security group can contain multiple inbound and outbound security rules that enables to filter the traffic from and to resources by source and destination IP address, port, and protocol.
* **Network virtual appliances**: it is essentially a VM that performs a network function such as firewall, WAN optimization and other network functions.

1. **What is Traffic Manager?**

Azure **Traffic Manager** is a DNS-based traffic load balancer that enables you to distribute traffic optimally to services across global Azure regions, while providing high availability and responsiveness.

**Traffic Manager** uses DNS to direct client requests to the most appropriate service endpoint based on a traffic-routing method and the health of the endpoints. An endpoint is any Internet-facing service hosted inside or outside of Azure. Traffic Manager provides a range of traffic-routing methods and endpoint monitoring options to suit different application needs and automatic failover models. Traffic Manager is resilient to failure, including the failure of an entire Azure region.

Traffic manager offers following features:

* **Increase application availability**: monitors the endpoint and provides automatic failover when endpoint is down.
* **Improve application performance**: directs the traffic towards the lowest network latency across the world.
* **Perform service maintenance**: without downtime directs the traffic to alternative endpoint while the maintenance is in progress
* **Combine hybrid applications**: supports external non-azure endpoints, enabling it to be used with hybrid cloud and on-premises deployments.

1. **What are the traffic routing methods available in traffic manager?**

The following traffic routing methods are available in Traffic Manager:

* **Priority**: Select Priority when you want to use a primary service endpoint for all traffic, and provide backups in case the primary or the backup endpoints are unavailable.
* **Weighted**: Select Weighted when you want to distribute traffic across a set of endpoints, either evenly or according to weights, which you define.
* **Performance**: Select Performance when you have endpoints in different geographic locations and you want end users to use the "closest" endpoint in terms of the lowest network latency.
* **Geographic**: Select Geographic so that users are directed to specific endpoints (Azure, External, or Nested) based on which geographic location their DNS query originates from. This empowers Traffic Manager customers to enable scenarios where knowing a user’s geographic region and routing them based on that is important. Examples include complying with data sovereignty mandates, localization of content & user experience and measuring traffic from different regions.
* **Multi-value**: Select Multi-value for Traffic Manager profiles that can only have IPv4/IPv6 addresses as endpoints. When a query is received for this profile, all healthy endpoints are returned.
* **Subnet**: Select Subnet traffic-routing method to map sets of end-user IP address ranges to a specific endpoint within a Traffic Manager profile. When a request is received, the endpoint returned will be the one mapped for that request’s source IP address.

1. **What is Traffic manager endpoint monitoring?**

Azure **Traffic Manager** includes built-in endpoint monitoring and automatic endpoint failover. This feature helps you deliver high-availability applications that are resilient to endpoint failure, including Azure region failures.

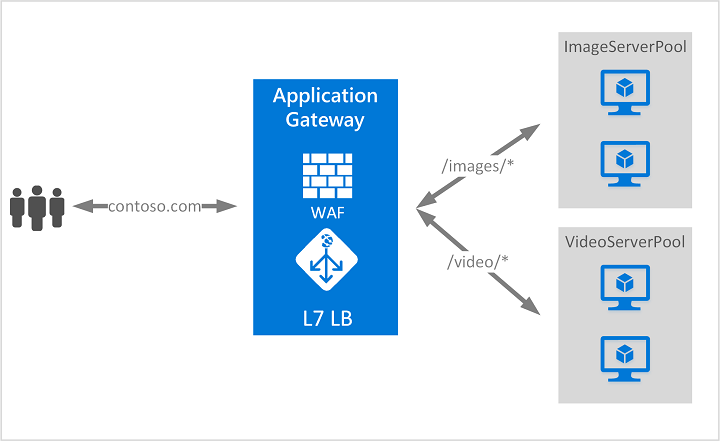
To configure endpoint monitoring, you must specify the following settings on your Traffic Manager profile:

* **Protocol**. Choose HTTP, HTTPS, or TCP as the protocol that Traffic Manager uses when probing your endpoint to check its health. HTTPS monitoring does not verify whether your SSL certificate is valid--it only checks that the certificate is present.
* **Port**. Choose the port used for the request.
* **Path**. This configuration setting is valid only for the HTTP and HTTPS protocols, for which specifying the path setting is required. Providing this setting for the TCP monitoring protocol results in an error. For HTTP and HTTPS protocol, give the relative path and the name of the webpage or the file that the monitoring accesses. A forward slash (/) is a valid entry for the relative path. This value implies that the file is in the root directory (default).
* **Custom header settings** This configuration setting helps you add specific HTTP headers to the health checks that Traffic Manager sends to endpoints under a profile. The custom headers can be specified at a profile level to be applicable for all endpoints in that profile and / or at an endpoint level applicable only to that endpoint. You can use custom headers for having health checks to endpoints in a multi-tenant environment be routed correctly to their destination by specifying a host header. You can also use this setting by adding unique headers that can be used to identify Traffic Manager originated HTTP(S) requests and processes them differently.
* **Expected status code ranges** This setting allows you to specify multiple success code ranges in the format 200-299, 301-301. If these status codes are received as response from an endpoint when a health check is initiated, Traffic Manager marks those endpoints as healthy. You can specify a maximum of 8 status code ranges. This setting is applicable only to HTTP and HTTPS protocol and to all endpoints. This setting is at the Traffic Manager profile level and by default the value 200 is defined as the success status code.
* **Probing interval**. This value specifies how often an endpoint is checked for its health from a Traffic Manager probing agent. You can specify two values here: 30 seconds (normal probing) and 10 seconds (fast probing). If no values are provided, the profile sets to a default value of 30 seconds. Visit the [Traffic Manager Pricing](https://azure.microsoft.com/pricing/details/traffic-manager) page to learn more about fast probing pricing.
* **Tolerated number of failures**. This value specifies how many failures a Traffic Manager probing agent tolerates before marking that endpoint as unhealthy. Its value can range between 0 and 9. A value of 0 means a single monitoring failure can cause that endpoint to be marked as unhealthy. If no value is specified, it uses the default value of 3.
* **Probe timeout**. This property specifies the amount of time the Traffic Manager probing agent should wait before considering that check a failure when a health check probe is sent to the endpoint. If the Probing Interval is set to 30 seconds, then you can set the Timeout value between 5 and 10 seconds. If no value is specified, it uses a default value of 10 seconds. If the Probing Interval is set to 10 seconds, then you can set the Timeout value between 5 and 9 seconds. If no Timeout value is specified, it uses a default value of 9 seconds.

1. **What is Azure application gateway?**

Azure **Application Gateway** is a web traffic load balancer that enables you to manage traffic to your web applications. Traditional load balancers operate at the transport layer (OSI layer 4 - TCP and UDP) and route traffic based on source IP address and port, to a destination IP address and port.

But with the Application Gateway you can be even more specific. For example, you can route traffic based on the incoming URL. So if /images is in the incoming URL, you can route traffic to a specific set of servers (known as a pool) configured for images. If “**/video**” is in the URL, that traffic is routed to another pool optimized for videos.



This type of routing is known as application layer (OSI layer 7) load balancing. Azure Application Gateway can do URL-based routing and more.

1. **Load-balancer Vs. Traffic manager Vs. Application gateway?**

**Traffic manager**: is kept right at the top and works only as an intelligent DNS routing. This essentially works at DNS level and uses DNS Responses to redirect end user traffic to **globally** distributed end points.

**Usage**: If you have resources at different locations and depending the end user’s closest latency you wish to direct the traffic to an end point, you should use Traffic Manager.

**Application Gateway:** This works at Application Layer (Layer 7). It acts as a reverse proxy service. This terminates the client connection and forwards request to back endpoints.

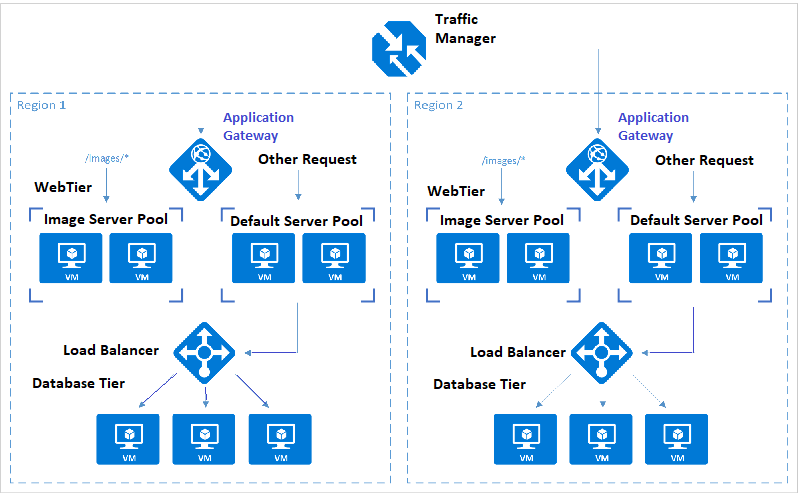
**Usage**: Also called as Application delivery controller, this is ideally used if you want to redirect to an endpoint depending on application. For e.g. The application developer has decided that any URLs that match the pattern /images/\* are served from a dedicated pool of VMs that are different from the rest of the web farm. We will see a diagram to help understanding this concept below.

**Load Balancing:** This works at Transport Layer (Layer 4). This provides network level distribution but essentially only with the same Azure Data Centre.

There are 2 types of Load balancing:

* Internet facing LB
* Internal LB

**Usage**: If you have couple of VM’s in the same Data Center and you wish to provide appropriate load balancing between them.



1. **What is Network Security Group (NSG)?**

Microsoft created **NSGs** to provide a flexible method for defining the access rules allowing traffic into and out of a VM in a VNet—or even an entire subnet in the VNet. When a Windows Server with a public IP address is created in the portal, an NSG is created that blocks all inbound Internet traffic except RDP on port 3389. Similarly, for a Linux VM with a public IP address, the default **NSG** created blocks all inbound traffic from the Internet except SSH on port 22. You have to specifically open any other ports you want open, including HTTP and HTTPS. If you do nothing, you are protected by default. The same set of rules can be applied to a single VM or multiple VMs. You can also apply an NSG to a subnet, which applies it to all of the VMs in that subnet.

When you create a VM, by default it’s going to ask you to create a **Network Security Group (NSG)**. You don’t have to create one. You can create your VMs in Azure and network them together without an NSG. However, if a VM has a public IP address, it is hosted on the public Internet, making it subject to attack. This means there is nothing protecting your VMs except the internal Windows firewall.

For example, let’s say you have four VMs running front-end applications. These connect to eight back-end servers that consist of web services and database servers. You could create one NSG that says “allow access to/from the public Internet, and access to the back end” and apply that to all the front-end VMs. Then, you can create another NSG that says “allow access from these four front-end servers, and allow access to the internal Azure services, but don’t allow access to the public Internet” and apply it to the back-end servers. The back-end servers will not be accessible from the public Internet. Note that NSGs are actually applied to a NIC attached to a VM (rather than the VM itself). If a VM has multiple NICs, the NSG needs to be applied separately to each NIC.

If we later add two front-end servers to our resources, we can simply assign them to the same NSG as the other front-end servers and add the new servers to the allowed servers for the back end. This allows you to implement changes with no updates to the running VMs themselves.

# *PAAS*

1. **What is an App Service?**

The App service is a service that hosts one of the five kinds of applications:

* Web Apps
* Mobile Apps
* Logic Apps
* API Aps, and
* Function apps

App Service not only adds the power of Microsoft Azure to your application, such as security, load balancing, auto-scaling, and automated management. You can also take advantage of its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources, package management, staging environments, custom domain, and SSL certificates.

With App Service, you pay for the Azure compute resources you use. The compute resources you use is determined by the App Service plan that you run your apps on.

1. **What is App Service Plan?**

In App Service, an app runs in an ***App Service plan***. An App Service plan defines a set of compute resources for a web app to run. One or more apps can be configured to run on the same computing resources (or in the same App Service plan).

When you create an App Service plan in a certain region (for example, West Europe), a set of compute resources is created for that plan in that region. Whatever apps you put into this App Service plan run on these compute resources as defined by your App Service plan. Each App Service plan defines:

* Region (West US, East US, etc.)
* Number of VM instances
* Size of VM instances (Small, Medium, Large)
* Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, Isolated)

1. **What are the various pricing tiers in App Service Plan?**

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**: **Free** and **Shared**, the two base tiers, runs an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources cannot scale out.
* **Dedicated compute**: The **Basic**, **Standard**, **Premium**, and **PremiumV2** 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 on top of compute isolation to your apps. It provides the maximum scale-out capabilities.

1. **What is Web-App?**

A Web App is a web application that is hosted in an App Service. The App Service is the managed service in Azure that enables you to deploy a web application and make it available to your customers on the Internet in a very short amount of time. As noted above, you don’t directly support the VMs on which your web app runs; they are managed for you. In fact, you don’t have access to those underlying VMs.

1. **What is Mobile App?**

Azure can also help you when you are creating mobile applications. You can host a backend for your mobile app in [Azure App Services Mobile Apps](https://azure.microsoft.com/services/app-service/mobile/). You can easily connect to this backend using the SDKs for Azure Mobile Apps that are available for IOS, Android, Windows, Xamarin.IOS, Xamarin.Android and Xamarin.Forms.

The mobile backend provides you with some unique benefits. One of them is the ability to do offline sync. This enables a user to continue working with the app if he is offline and sync data back to the backend when he comes online again. Another capability is push notifications. This allows you to send notifications about your app to the user’s device. Additionally, Mobile Apps has all of the same capabilities that Web Apps has, like auto-scaling, and high availability.

1. **What is Logic-App?**

Azure App Service Logic Apps are different from Web Apps and Mobile Apps in that you don’t host an application in it, but orchestrate business logic with it. Think of Logic Apps as a way to automate a business process by just configuring it.

A Logic App is started by a Trigger. This can be a time (every 15 minutes) or an outside source, like a new message on a queue. The Trigger passes values into the workflow (like the contents of the queue message), that can be used throughout the Logic App. The rest of the flow of the Logic App consists out of calling [Connectors](https://docs.microsoft.com/azure/connectors/apis-list), which are APIs to third party services, like Office365 or Twitter or your own APIs.

Logic Apps scale automatically and you only pay for them when they run. This is sometimes called “server less” because it means that you can completely focus on your application or logic and not on the underlying infrastructure.

1. **What are Azure Functions?**

[Azure App Services Function Apps](https://azure.microsoft.com/services/functions/) can host one or more Azure Functions. You use Azure Functions to host small applications, like background jobs or a Microservices that only runs for a short period of time.

Azure Functions can be triggered by configurable timers, like on a schedule (every 15 minutes) or by an external service, like when a new Blob is added to Azure Blob Storage. When triggered, the code in the Azure Function can use the value from the trigger, like the Blob that was added. You can also add output bindings to an Azure Function to output a value to an external service, without writing any plumbing. This could, for instance, be a Blob Storage output where you just return a Blob without having to write code to connect to Azure Storage.

Just like Logic Apps, Azure Functions are “**serverless**”, because they scale automatically and you only pay for them when they run.

Azure functions are based on **WebJobs** SDK. They provide most of the functionality already available in the **WebJobs**, but with extra capabilities.

In terms of triggers, in addition to those already available for WebJobs (e.g. Service Bus, Storage Queues, Storage Blobs, CRON schedules, WebHooks, EventHub, and File Cloud Storage providers), Azure Functions can be triggered as APIs. And HTTP calls don't require kudu credentials, but can be authenticated via Azure AD and third-party identity providers.

The most significant and cool advantages brought by Functions is the alternative of having a Dynamic Service Plan with a "Serverless" model, in which we don't need to manage VM instances or scaling; it's all managed for us. Additionally, by not having dedicated instances, we only pay for the resources we actually use.

There are no custom configuration in Azure functions.

1. **What are triggers supported in Azure Functions?**

Functions provides templates to get you started with key scenarios, including the following:

* **HTTPTrigger** - Trigger the execution of your code by using an HTTP request.
* **TimerTrigger** - Execute cleanup or other batch tasks on a predefined schedule.
* **CosmosDBTrigger** - Process Azure Cosmos DB documents when they are added or updated in collections in a NoSQL database.
* **BlobTrigger** - Process Azure Storage blobs when they are added to containers. You might use this function for image resizing.
* **QueueTrigger** - Respond to messages as they arrive in an Azure Storage queue.
* **EventGridTrigger** - Respond to events delivered to a subscription in Azure Event Grid. Supports a subscription-based model for receiving events, which includes filtering. A good solution for building event-based architectures.
* **EventHubTrigger** - Respond to events delivered to an Azure Event Hub. Particularly useful in application instrumentation, user experience or workflow processing, and internet-of-things (IoT) scenarios.
* **ServiceBusQueueTrigger** - Connect your code to other Azure services or on-premises services by listening to message queues.
* **ServiceBusTopicTrigger** - Connect your code to other Azure services or on-premises services by subscribing to topics.

Azure Functions supports triggers, which are ways to start execution of your code, and bindings, which are ways to simplify coding for input and output data.

Ref: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview>

1. **How much does Functions app cost?**

Azure Functions has two kinds of pricing plans. **Consumption Plan** and **App Service Plan**.

**Consumption plan** - When you're using a Consumption plan, instances of the Azure Functions host are dynamically added and removed based on the number of incoming events. This Serverless plan scales automatically, and you're charged for compute resources only when your functions are running. On a Consumption plan, a function execution times out after a configurable period of time.

Billing is based on number of executions, execution time, and memory used. Billing is aggregated across all functions within a function app.

The Consumption plan is the default hosting plan and offers the following benefits:

* Pay only when your functions are running.
* Scale out automatically, even during periods of high load.

**App Service plan** - In the dedicated App Service plan, your function apps run on dedicated VMs on Basic, Standard, Premium, and Isolated SKUs, which is the same as other App Service apps. Dedicated VMs are allocated to your function app, which means the functions host can be always running. App Service plans support Linux.

Consider an App Service plan in the following cases:

* You have existing, underutilized VMs that are already running other App Service instances.
* Your function apps run continuously, or nearly continuously. In this case, an App Service Plan can be more cost-effective.
* You need more CPU or memory options than what is provided on the Consumption plan.
* Your code needs to run longer than the maximum execution time allowed on the Consumption plan, which is up to 10 minutes.
* You require features that are only available on an App Service plan, such as support for App Service Environment, VNET/VPN connectivity, and larger VM sizes.
* You want to run your function app on Linux, or you want to provide a custom image on which to run your functions.

A VM decouples cost from number of executions, execution time, and memory used. As a result, you won't pay more than the cost of the VM instance that you allocate.

**Always On**

If you run on an App Service plan, you should enable the Always on setting so that your function app runs correctly. On an App Service plan, the functions runtime goes idle after a few minutes of inactivity, so only HTTP triggers will "wake up" your functions. Always on is available only on an App Service plan. On a Consumption plan, the platform activates function apps automatically.

1. **What are durable Functions?**

Durable functions are extensions of Azure functions, that let you write stateful functions in a serverless environment. The extension manages state, checkpoints, and restarts for you. Durable functions can be used for stateful orchestration of function execution.

A durable function is made up of different Azure functions. There are three types of durable functions:

* **Activity Functions**: these are the basic unit of work in a durable function orchestration. Activity functions are the functions and tasks that are orchestrated in the process. For example, you might create a durable function to process an order. The tasks involve checking the inventory, charging the customer, and creating a shipment. Each task would be an activity function.
* **Orchestrator Functions**: These functions describe how actions are executed and the order in which actions are executed. Orchestrator functions describe the orchestration in code. An orchestration can have many different types of actions, including [activity functions](https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-types-features-overview#activity-functions), [sub-orchestrations](https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-types-features-overview#sub-orchestrations), [waiting for external events](https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-types-features-overview#external-events), and [timers](https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-types-features-overview#durable-timers)
* **Client Function**: are the triggered functions that create new instances of an orchestration. Client functions are the entry point for creating an instance of a Durable Functions orchestration. You can trigger a client function from any source (HTTP, queue, event stream). You can write a client function in any language that the app supports.

1. **What are Web-Jobs?**

Another way to run background tasks is to run them in Azure **WebJobs**. **WebJobs** are part of App Services and run inside an App Service like a Web App or a Mobile App. You can write and host code in WebJobs that gets started by a trigger, like a timer (every 15 minutes) or an external service, like a new message in a queue.

One of the more common requirements for web applications is to be able to run background tasks. Batch processing, scheduled tasks, and long-running processes are all common in modern applications. The issue with running these on IIS is that it consumes one of the precious threads dedicated to serving content and the process may be interrupted by an app pool recycle. There are, of course, some tricks to push off app pool recycles until your task has completed, but ideally we’d like to run the task outside of IIS. **WebJobs** were created to provide the capability to run the task outside of IIS. Developers can call out to a **WebJob** via a messaging system such as Storage Queues or Azure Service Bus and have it complete the task while the main application continues on.

The advantages to such a system are numerous:

* Frees up IIS threads
* Can easily be run on a separate machine to avoid scalability issues
* Offers a higher degree of resilience to app pool recycles

WebJobs work similarly to Azure Functions in that they run small pieces of code that can be triggered by outside sources that don’t require any plumbing code to set up.

They are different from Azure Functions in that you need to scale them manually. You pay for the App Service that hosts your Web Job, which is a monthly fee, regardless if the WebJob runs or not.

1. **Web Jobs Vs. Azure functions?**

The significant difference between **WebJobs** and Azure **Functions** is the pricing.

**WebJobs** require you to create and manage an Azure App Service (Web App) and the underlying App Service Plan (a.k.a. server farm). If you want your **WebJob** to run continuously, you need at least one instance on a Basic App Service Plan to support “Always On”. For **WebJobs** you always need to pay for at least one VM Instance (as PaaS) regardless of this being used or idle. For **WebJobs**, the [App Service Plan Pricing](https://azure.microsoft.com/en-us/pricing/details/app-service/) applies. However, you can always deploy more than one App Service on one App Service Plan. If you have larger loads or load peaks and you need auto-scaling, then you would require at least a Standard App Service Plan.

Conversely, with **Azure Functions** and the [Dynamic Service Plan](https://azure.microsoft.com/en-us/documentation/articles/functions-scale/) (**Serverless**), the creation and management of a VM Instances and configuring scaling is all abstracted now. We can write functions without caring about server instances and get the benefits of a [**Serverless**](http://martinfowler.com/articles/serverless.html) architecture. Functions scale out automatic and dynamically as load increases, and scale down if decreases. Scaling up or down is performed based on the traffic, which depends on the configured triggers.

With **functions**, you get billed only for the [resources you actually use](https://azure.microsoft.com/en-us/pricing/details/functions/). The cost is calculated by the number of executions, memory size, and execution time measure as Gigabyte Seconds. If you have background processes which don’t require a dedicated instance and you only want to pay for the compute resources in use, then a dynamic plan would make a lot of sense.

It’s worth noting that if you already have an **App Service Plan**, which you are already managing and paying for, and has resources available, you can deploy your **Functions** on it and avoid extra costs.

One point to consider with the *Dynamic Service Plan* (**Serverless** model) is that as you don’t control which instances are hosting your Azure Functions, there might be a [cold-startup overhead](https://msdn.microsoft.com/en-us/library/cc656914(v=vs.110).aspx). This wouldn’t be the case for **Functions** running on your own App Service Plan (server farm) or **WebJobs** running as continuous on an “*Always On*” Web App where you have “dedicated” instances and can benefit from having your components loaded in memory.

1. **What are the Pros and Cons of Serverless?**

Serverless is a development approach that replaces long-running virtual machines with compute power that comes into existence on demand and disappears immediately after use.

Despite the name, there certainly are servers involved in running your application. It’s just that your cloud service provider, whether it’s AWS, Azure, or Google Cloud Platform, manages these servers, and they’re not always running.

Rather, you configure events, such as API requests or file uploads, that trigger your serverless function to execute. And when that action is complete, the server goes idle until another action is requested, and you are not billed for the idle time.

**Pros of serverless**:

1. Pay as you use. Pay for the short time when the server is active.
2. Scalability/Elasticity. Automatically scale up to accommodate many concurrent users.
3. Manageability. Spend less time on provisioning infrastructure, capacity, avoiding downtime.
4. Market faster. Reduce development time and get your products to market faster.
5. Leverages Micro-service. Build smaller, loosely-coupled parts of the software.
6. Independent of hosting location. High availability. No affinity to particular availability zones.

**Cons of serverless**:

1. Cold start. Latency involved in executing tasks.
2. Vendor lock-in. Moving to another Cloud service provider (CSP) might need code changes.
3. Not good for long running apps. Gaming or analysis apps of very large datasets wont fit good as serverless functions have time limits (5 mins).
4. Tough to build Complex app. Needs a lot of coordination and manage dependencies between all of the serverless components.
5. **What is App Service Environment (ASE)?**

The Azure **App Service Environment** is an Azure App Service feature that provides a fully isolated and dedicated environment for securely running App Service apps at high scale. It is a Private PAAS environment in the Cloud.

The ASE is a capability that allows us to host our apps in Network isolation and provides many scenarios not available in the multi-tenant App service.

This capability can host your:

* Windows web apps
* Linux web apps
* Docker containers
* Mobile apps
* Functions

App Service environments (**ASEs**) are appropriate for application workloads that require:

* Very high scale.
* Isolation and secure network access.
* High memory utilization.

Customers can create multiple **ASEs** within a single Azure region or across multiple Azure regions. This flexibility makes **ASEs** ideal for horizontally scaling stateless application tiers in support of high RPS workloads.

**ASEs** are isolated to running only a single customer's applications and are always deployed into a virtual network. Customers have fine-grained control over inbound and outbound application network traffic. Applications can establish high-speed secure connections over VPNs to on-premises corporate resources.

1. **Normal App service(Multi-tenant) Vs. ASE?**

Azure offer three different app service deployment model, namely, Multi-tenant App Service (normal app service), App service Environment and Azure Stack.

Although, Multi-tenant are secure and isolated, however, we cannot lock it in our network.

**ASE**, on the other hand, is a deployment of Azure App service directly into the subnet in customer’s virtual network. This gives benefits of hosting a line of business applications on a private IP address or to scale to greater sizes because we have greater access to virtual machines.

Azure stack is a deployment of the App Service into our on-premise azure stack deployment.

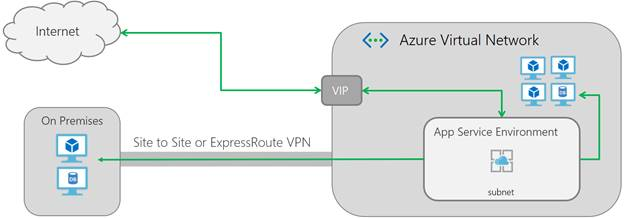
So, basically the difference between Multi-tenant app service and **ASE** is that **ASE** is a *private* PAAS environment in cloud. Multi-tenant is useful when you have an application that you can host in a public facing internet service. **ASE** is useful when we want the endpoints under tight control or we want to lock down the network access using NSG or internal load balancer.

1. **What is External ASE and ILB ASE?**

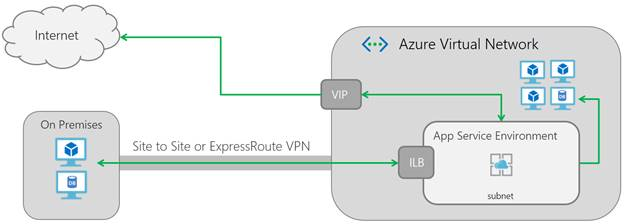
There are two deployment types for an App Service Environment: **External** ASE and **ILB** ASE.

**External** ASE exposes the ASE-hosted apps on an internet-accessible IP address., whereas ILB ASE exposes the ASE hosted apps on an IP address inside our Virtual Network(**VNet**). This is done using **Internal Load Balancer**, which has public IP.

All ASEs, **External**, and **ILB**, have a public VIP that is used for inbound management traffic and as the **“*from”*** address when making calls from the **ASE** to the internet. The calls from an **ASE** that go to the internet leave the VNet through the VIP assigned for the **ASE**. The public IP of this VIP is the source IP for all calls from the ASE that go to the internet. If the apps in your ASE make calls to resources in your VNet or across a VPN, the source IP is one of the IPs in the subnet used by your ASE. Because the **ASE** is within the VNet, it can also access resources within the VNet without any additional configuration. If the VNet is connected to your on-premises network, apps in your ASE also have access to resources there without additional configuration.



An external ASE



An ILB ASE

1. **What is Enterprise Application Integration?**

***Enterprise Application Integration*** is an approach to provide the interoperability between multiple disparate systems that make up a typical Enterprise infrastructure.

Enterprise architectures, by their nature, tend to consist of many systems and applications, which provide the various services the company relies upon, to conduct their day to day business.  A single organization might use separate systems, either developed in-house or licensed from a third party vendor, to manage their supply chain, customer relationships, employee information, and business logic.  This modularization is often desirable.  In theory, breaking the task of running a business into multiple smaller functionalities allows for easy implementation of the best and newest technological advancements in each area, and quick adaptation to changing business needs.

However, to gain the benefits of this kind of distributed, modular system, an organization must implement technologies that deal with the problems presented by this architecture:

* Interoperability: the various components of the infrastructure may use different operating systems, data formats, and languages, preventing connection via a standard interface.
* Data integration: in order for a modular, distributed system to be functional, a standard method of handling the flow of data between applications and systems to enforce consistency across the database is crucial.
* Robustness, Stability, and Scalability: Because they are the glue that holds together a modular infrastructure, integration solutions must be highly robust, stable, and scalable.

1. **What are the ways to implement EAI?**

There are three popular ways to implement EAI

* **Point to Point**: A unique connector component is implemented for each pair of applications or system that communicate. Complexity grows exponentially.
* **Broker Model**: A central integration engine, residing in the middle of the network, and providing all message transformation, routing, and any other inter-application functionality. All communication between applications must flow through the hub, allowing the hub to maintain data concurrency for the entire network. Can become a single point of failure for the network. Can become a bottleneck for messages.
* **Enterprise service bus**: The bus architecture sought to lessen the burden of functionality placed on a single component by distributing some of the integration tasks to other parts of the network.

1. **What is Enterprise service bus?**

An ESB implements a communication system between mutually interacting software applications in a service oriented architecture (SOA).

The primary duties of an ESB are:

* Route messages between services
* Monitor and control routing of message exchange between services
* Resolve contention between communicating service components
* Control deployment and versioning of services
* Marshal use of redundant services
* Provide commodity services like event handling, data transformation and mapping, message and event queuing and sequencing, security or [exception handling](https://en.wikipedia.org/wiki/Exception_handling), protocol conversion and enforcing proper quality of communication service

1. **What are Azure Service Bus, Service Bus Queue and, Service Bus Topics?**

Microsoft **Azure Service Bus** is a fully managed enterprise [integration](http://azure.com/integration) message broker. Service Bus is most commonly used to decouple applications and services from each other, and is a reliable and secure platform for asynchronous data and state transfer. Data is transferred between different applications and services using *messages*. A message is in binary format, which can contain JSON, XML, or just text.

Some common messaging scenarios are:

* Messaging: transfer business data, such as sales or purchase orders, journals, or inventory movements.
* Decouple applications: improve reliability and scalability of applications and services (client and service do not have to be online at the same time).
* Topics and subscriptions: enable 1:*n* relationships between publishers and subscribers.
* Message sessions: implement workflows that require message ordering or message deferral.

**Queues**: enable you to store messages until the receiving application is available to receive and process them. Messages in queues are ordered and time-stamped on arrival. Once accepted, the message is held safely in redundant storage. Messages are delivered in *pull* mode, which delivers messages on request.

**Topics**: While a queue is often used for point-to-point communication, topics are useful in publish/subscribe scenarios. Topics can have multiple, independent subscriptions. A subscriber to a topic can receive a copy of each message sent to that topic. Subscriptions are named entities, which are durably created but can optionally expire or auto-delete.

1. **What is Azure service fabric?**

Azure **Service Fabric** represents the next generation hosting environment for cloud-scale solutions. It is a deployment management system. Service Fabric makes it much easier to deploy and manage highly scalable, available, and reliable services. Applications designed as micro-services are well suited to run on **Service Fabric**, primarily due to Service Fabric’s runtime and lifecycle management capabilities—providing comprehensive capabilities for failover, leader election, state management, live upgrades with rollback, and automatic scale-up and scale-down.

It creates a versioned deployable app for you to send out to a connected network of system resources. The servers in the system don’t have names. So, instead of sending your application to a specific machine, you are just sending it to the cluster. It manages where to place these services in order to evenly distribute hotspots across available resources. Service Fabric also has managed deployment rollback should a service fail to initialize. Service communication, distribution patterns, scale patterns, service discovery – all are built right in.

It supports all kind of stacks, communication protocol and technology. It can be run in Linux environment. It can be used in on-premises data centers, any other cloud provider infrastructure, apart from Azure. Azure service fabric can also host containers.

1. **What is Event hub?**

Azure **Event Hubs** is a highly scalable managed service capable of ingesting millions of events per second, enabling you to capture, process, and analyze massive amounts of data originating from connected devices (often IoT scenarios) and applications. You can think of Event Hubs as a gateway, or entry point, for an event processing pipeline. Data is collected into an Event Hub, then transformed and stored. You have control over what data transformations and storage are needed.

The programmatic interface for Event Hubs is **AMQP** (**Advanced Message Queuing Protocol**) or **HTTP(S)**, making it very easy for a wide range of clients to publish event data to Event Hubs. To support the need for massive scale, Event Hubs uses a partitioning pattern to scale the load internally. Receiving messages from an Event Hub is handled via consumer groups. Consumer groups are responsible for knowing from which partition to read and maintaining a view (state, position in the stream, etc.) of the Event Hub.

It is important not to confuse **Event Hubs** with **Azure Service Bus queues** or **topics**. While the two are similar in that they are both messaging systems, Event Hubs is designed specifically for handling message events at high scale. It does not implement some of the messaging capabilities of Service Bus queues and topics, such as dead lettering, filters (property based routing), and various message retrieval, delivery, and scale semantics. Service Bus is better suited for per-message needs, while Event Hubs is better suited for event streaming needs.

1. **What is Notification hubs?**

While Event Hubs allow you to take in millions of events per second, Azure **Notification Hubs** send data in the other direction—they enable you to send push notifications to mobile devices from any backend, whether in the cloud or on-premises. With a single API call, you can target individual users or entire audience segments of millions of users across all of their devices.

Push notifications are challenging. In general, the app developer still has to do much of the work to implement even common push notification scenarios, like sending notifications to a specific group of customers. To make them work, you have to build infrastructure that is complicated and, in most cases, unrelated to the business logic for the app.

**Notification Hubs** remove that complexity, eliminating the need for you to manage the challenges of push notifications. Notification Hubs are cross-platform—they can be used to support Windows, iOS, Android, and Windows Phone apps; they reduce the amount of push-specific code you have to put in your backend. They are fully scalable, allowing you to send notifications to millions of devices with a single API call.

All of the functionality of a push infrastructure is implemented in **Notification Hubs** for you. The devices only have to register their PNS handles, and the backend can send messages to customers without worrying about the platform the customers are using.

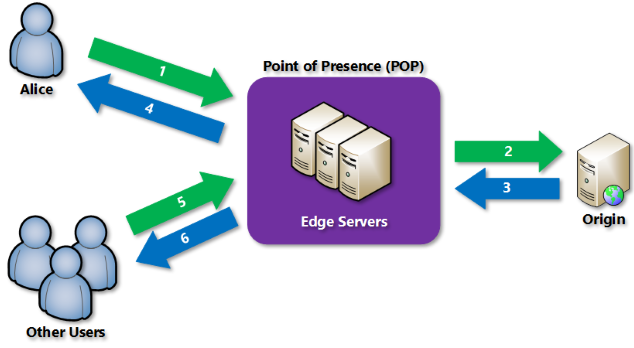
1. **What is Azure Redis Cache?**

Redis is an open source, in-memory data structure store often used as a cache, database, or message broker. Azure Redis Cache is based on the popular open source Redis. The difference is that Azure manages Redis for you, saving you the trouble of spinning up a VM and installing and managing Redis yourself while still giving you a secure and dedicated Redis cache that can be accessed from any application within Azure. You can provision a Redis cache using the Azure portal.

1. **What is Azure CDN?**

Azure **Content Delivery Network** (CDN) is a global **CDN** solution for delivering high-bandwidth content. It can be hosted in Azure or any other location. With Azure **CDN**, you can cache static objects loaded from Azure Blob storage, a web application, or any publicly accessible web server, by using the closest point of presence (POP) server, to minimize latency. Azure **CDN** can also accelerate dynamic content, which cannot be cached, by leveraging various network and routing optimizations.

1. **How CDN works?**
2. A user (Alice) requests a file (also called an asset) by using a URL with a special domain name, such as *<endpoint name>*.azureedge.net. This name can be an endpoint hostname or a custom domain. The DNS routes the request to the best performing POP location, which is usually the POP that is geographically closest to the user.
3. If no edge servers in the POP have the file in their cache, the POP requests the file from the origin server. The origin server can be an Azure Web App, Azure Cloud Service, Azure Storage account, or any publicly accessible web server.
4. The origin server returns the file to an edge server in the POP.



1. An edge server in the POP caches the file and returns the file to the original requestor (Alice). The file remains cached on the edge server in the POP until the time-to-live (TTL) specified by its HTTP headers expires. If the origin server didn't specify a TTL, the default TTL is seven days.
2. Additional users can then request the same file by using the same URL that Alice used, and can also be directed to the same POP.
3. If the TTL for the file hasn't expired, the POP edge server returns the file directly from the cache. This process results in a faster, more responsive user experience.
4. **What are the Pros and Cons of Container?**

A container is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, and settings.

Containers solve the problem of running software when it has been moved from one computing environment by essentially isolating it from its environment. For instance, containers allow you to move software from development to staging and from staging to production, and have it run reliably regardless of the differences of all the environments.

**Pros of container**

1. Portability. The main draw of a container is that you can combine the application and all of its dependencies into a neat little package and run it anywhere. This provides an unprecedented level of flexibility and portability, and allows you to stay cloud vendor-agnostic.
2. Control of the domain. Manage all policies, resources, security, and determine how the application is deployed and behaves.
3. Supports large complex applications: applications can also be as large and complex as you need them to be, as there are no memory or time limitations like there are with serverless.

**Cons of container**

1. Weaker isolation and security. Containers share the kernel, other components of the host operating system, and they have root access. This means that containers are less isolated from each other than virtual machines, and if there is a vulnerability in the kernel it can jeopardize the security of the other containers as well.
2. Time to setup and manage. containers take much more work to set up and manage. Every time you make a change to your codebase, you’ll need to package the container and ensure all of the containers communicate with each other properly before deploying into production. You’ll also need to keep containers’ operating systems up to date with frequent security fixes and other patches. And you have to figure out which containers go on which servers. All of this can slow down your development process.
3. Because containers need a long-running hosting location, they are more expensive to run than serverless functions. With serverless, you only pay when servers execute your function. With containers, you have to pay for server usage even if they’re sitting idle.
4. As an application grows, more and more containers are added. And these containers are highly dispersed, scattered, and constantly changing, thus making monitoring a nightmare.
5. **Serverless vs. Container, what to choose when?**

**Containers are best suited for:**

* Large, complex and long running applications.
* Migrating Monolithic legacy applications.

**Serverless are best suited for:**

* Small task based application, with short running time
* Applications which do not always need to be running continuously.
* Applications for which auto-scaling is needed.

**Comparison**

* Scalability: Serverless can auto-scale while containers has to be pre-defined quantity
* Cost: Serverless are consumption based charged while containers are always up and running which is chargeable.
* Maintenance: Serverless are managed by CSP while containers are managed by developers.
* Time of deployment: Serverless takes milliseconds to deploy while containers takes seconds.

# *OAuth*

1. **What is OAuth?**

**OAuth** is a protocol that allows a user to grant limited access to their resources residing at one site, to the another site, without having to expose users credentials.

**OAuth** allows notifying a resource provider (e.g. Facebook) that the resource owner (e.g. you) grants permission to a third-party (e.g. a Facebook Application) access to their information (e.g. the list of your friends).

The **OAuth** authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

In OAuth, there are following roles:

* **Resource Owner**: the entity that can grant access to a protected resource. Typically this is the end-user.
* **Resource Serv**er: the server hosting the protected resources. This is the API you want to access.
* **Client**: the app requesting access to a protected resource on behalf of the Resource Owner.
* **Authorization Server**: the server that authenticates the Resource Owner, and issues Access Tokens after getting proper authorization. Also known as Secure Token Service (STS), or Id Provider.

1. **What is Access Token and Id Token?**

An **Access Token** is a string representing the granted permissions. It is in the **JSON Web Token** (**JWT**) format. A **JWT** contains 3 parts: Header, Payload and a Signature, all separated by dot.

The *Header* contains the meta data about the token type, and the cryptographic algorithm to encrypt the contents.

The *Payload* contains a set of claims, which are the statement about the permissions that should be allowed, and other information like audience, expiration time, roles, scopes etc.

The *Signature* is used to validate that the token is trustworthy and has not been tampered with.

An **ID Token** are sent to the client application as part of an **OpenID connect** flow. They can be sent alongside or instead of an **access token**, and are used by the client to authenticate the *user*. The **ID Token** should be used to validate that a user is who they claim to be and get additional useful information about them - it shouldn't be used for authorization in place of an **Access Token**. Like **Access Token**, it is a JWT format.

The payload of **ID Token** differs from the **access token**. For example, there is no scope claim in **ID Token**.

1. **Define Scope**

A **Scope** is a mechanism in OAuth 2.0 to limit an application's access to a user's account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the **access token** issued to the application will be limited to the scopes granted.

The OAuth spec allows the authorization server or user to modify the scopes granted to the application compared to what is requested, although there are not many examples of services doing this in practice.

OAuth does not define any particular values for scopes, since it is highly dependent on the service's internal architecture and needs.

1. **What are the ways to use Scopes?**

When an app requests permission to access a resource through an authorization server, it uses the scope parameter to specify what access it needs, and the authorization server uses the scope parameter to respond with the access that was actually granted (if the granted access was different from what was requested).

Generally, you use scopes in three ways:

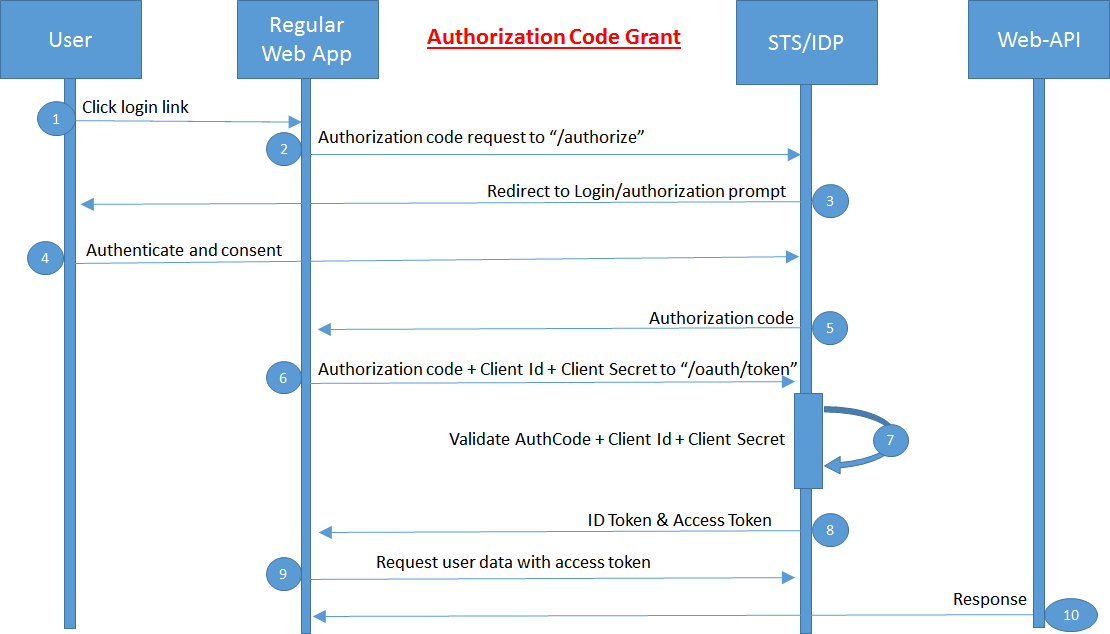
* From an [application](https://auth0.com/docs/applications), to verify the identity of a user and get basic profile information about the user, such as their email or picture. Each scope returns a set of user attributes, which are called ***claims***. In this scenario, the scopes, available to you include those implemented by the **OpenID Connect** (**OIDC**) protocol. For details, see [*OpenID Connect Scopes*](https://auth0.com/docs/scopes/current/oidc-scopes).
* In an [API](https://auth0.com/docs/apis), to implement access control. In this case, you need to define custom scopes for your API and then identify these scopes so that calling applications can use them. For details, see [*API Scopes*](https://auth0.com/docs/scopes/current/api-scopes).
* From an application, to call an API that has implemented its own custom scopes. In this case, you need to know which custom scopes are defined for the API you are calling. For an example of calling a custom API from an application, see [*Sample Use Cases: Scopes and Claims*](https://auth0.com/docs/scopes/current/sample-use-cases#request-custom-API-access)*.*

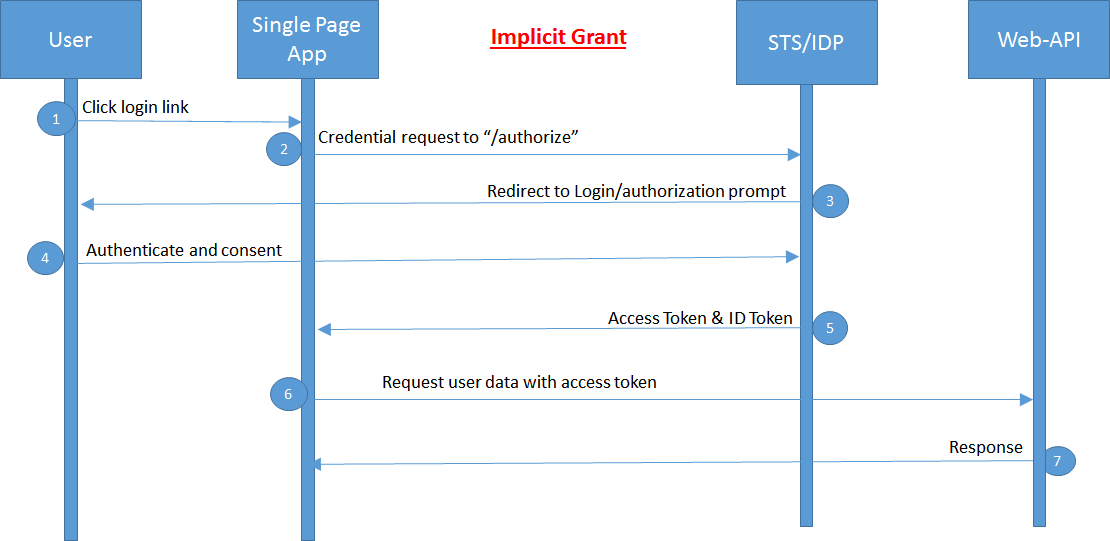
1. **What are the various Authorization grant types?**

There are majorly 2 flows to get access token. These flows are called grant types.

1. **Authorization code flow**: In this Authorization code(AuthCode) is exchanged for ID token and Access Token. This flow is suitable for the server side application where the Auth-Code and the client secret are not exposed to outside world, which is not possible in SPA applications.
2. **Implicit flow**: In this the ID token and Access tokens are generated and passed to the client app, without involving AuthCode. The AuthCode might be generated by the STS however it is not exposed to client applications. This flow is best suited for the SPA applications where client secrets cannot be shared. Client-Secret an important information which along with AuthCode, has to be passed to the STS, in order to get the Access token and ID token.

There are other flow types like Authorization code flow with Proof Key for Code Exchange (PKCE), Client Credential flow etc.





1. **What is Open Id Connect and how it is different than OAuth?**
2. **What parameters are needed to get Auth-code?**
3. **How an application is registered with STS?**

# *Azure DevOps*

1. **What are the approaches for Infrastructure as code?**

There are in general two approaches one can adopt to implement Infrastructure as Code and Configuration as Code: Declarative and Imperative.

**Declarative** (functional). The declarative approach states *what* the final state should be. When run, the script or definition will initialize or configure the machine to have the finished state that was declared, without defining *how* that final state should be achieved.

**Imperative** (procedural). In the imperative approach, the script states the *how* for the final state of the machine by executing the steps to get to the finished state. It defines what the final state needs to be, but also includes how to achieve that final state. It also can include coding concepts such as for loops, and matrices.

The declarative approach abstracts away the methodology of how a state is achieved. As such, it can be easier to read and understand what is being done. It also makes it easier to write and define. Declarative approaches also separate out the final desired state, and the coding required to achieve that state. Thus, it does not force you to use a particular approach, which allows for optimization where possible.

1. **What is Idempotence?**

**Idempotence** is a mathematical term that can be used in the context of Infrastructure as Code and Configuration as Code. It is the ability to apply one or more operations against a resource, resulting in the same outcome.

For example, if you run a script on a system it should have the same outcome regardless of the number of times you execute the script. It should not error out, or perform duplicate actions regardless of the environment’s starting state.

In essence, if you apply a deployment to a set of resources 1,000 times, you should end up with the same result after each application of the script or template.

1. **What is a Technical debt?**

**Technical debt** is a set of problems in a development effort that make progress on customer value inefficient. We think of our scripts, templates, and definition files as code; technical debt undermines productivity by making this code fragile, hard to understand, time-consuming to change, and difficult to validate. This in turn creates unplanned work that blocks progress.

Technical debt saps an organization’s strength because of high customer-support costs. Eventually, some combination of these issues produces larger problems.

Technical debt is insidious, meaning it starts small and grows over time as a result of rushed changes, and lack of context and discipline. It can seemingly materialize out of nowhere (even for a project regarded as clean), because of changes in project circumstances. For example, a product produced for one particular market might be considered for international release. This instantly creates debt related to its ability to localize. The technical debt introduced to the application will have repercussions later on, and will need to be addressed at some stage.

Technical debt includes anything the team must do to deploy production quality code and keep it running in production. Examples of technical debt can be:

* Bugs
* Performance issues
* Operational issues
* Accessibility
* Manual updates or configurations not implemented using infrastructure or configuration as code methodologies, such as version control
* Changes made ‘on-the-fly’, or directly to an application, without using DevOps methodologies
* Switching to technologies or versions not accounted for in your development process
* Updates to platform or services that you were not aware of or have not accounted for.

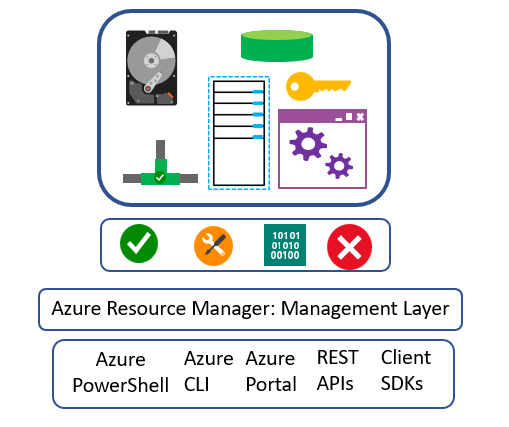
1. **What is Configurational drift?**

**Configuration drift** is the process of a set of resources changing over time from their original deployment state. This can be because of changes made by people, processes, or programs and can be a done manually or via automation.

Eventually, an environment may become a *snowflake*. A *snowflake* is a unique configuration that cannot be reproduced automatically, snowflakes are typically a result of configuration drift. Inconsistency among environments may lead to issues during deployment. With *snowflakes*, the infrastructure administration and maintenance invariably involves manual processes, which can be hard to track and prone to human error. The more an environment drifts from its original state, the more likely it is for an application to encounter issues. The greater the degree of configuration drift, the longer it takes to troubleshoot and rectify issues.

1. **What is Azure Resource Manager?**

**Azure Resource Manager** is a management layer in which a resource group and all the resources within it are created, configured, managed, and deleted. It provides a consistent management layer that allows you automate deployment and configuration of resources, using different automation and scripting tools such as Microsoft Azure PowerShell, Azure Command-Line Interface (Azure CLI), Microsoft Azure portal, REST API, and client SDKs.



With **Azure Resource Manager**, you can deploy Application resources. You also can update, manage, and delete all the resources for your solution in a single, coordinated operation.

**Azure Resource Manager (ARM)** has several components, one of which is *resource provider*. *Resource providers* offer a set of resources and operations for working with an Azure service, which are made available through Azure Resource Manager. Some common resource providers are:

* Microsoft.Compute, which supplies the virtual machine resource.
* Microsoft.Storage, which supplies the storage account resource.
* Microsoft.Web, which supplies resources related to web apps.

1. **What is ARM Template and its components?**

An **ARM template** precisely defines all the Resource Manager resources in a deployment. You can deploy a Resource Manager template into a resource group as a single operation.

Resource Manager templates are declarative in nature and written in JSON format. By using a template, you can repeatedly deploy your solution in development, test, and production type environments throughout its lifecycle, and have confidence that your resources are deployed in a consistent state.

**ARM template** can contain sections that are expressed using JSON notation, but are not related to the JSON language itself. These sections are:

* $schema



* contentVersion
* parameters
* variables
* functions
* resources and
* outputs

**Parameters**:

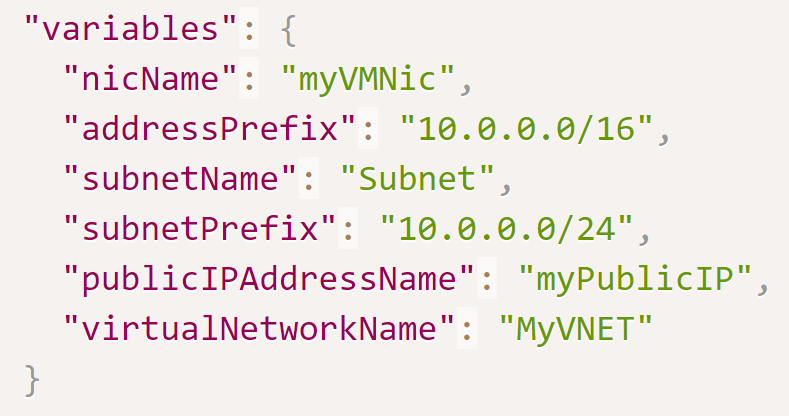
This section is where you specify which values are configurable when the template runs. For example, you might allow users of your template to specify a username, password, or domain name.



Here's an example that illustrates two parameters: one for a virtual machine's (VM's) username, and one for its password.

**Variables:**

This section is where you define values that are used throughout the template. Variables can help make your templates easier to maintain. For example, you might define a storage account name one time as a variable, and then use that variable throughout the template. If the storage account name changes, you need to only update the variable once.



Here's an example that illustrates a few variables that describe networking features for a VM.

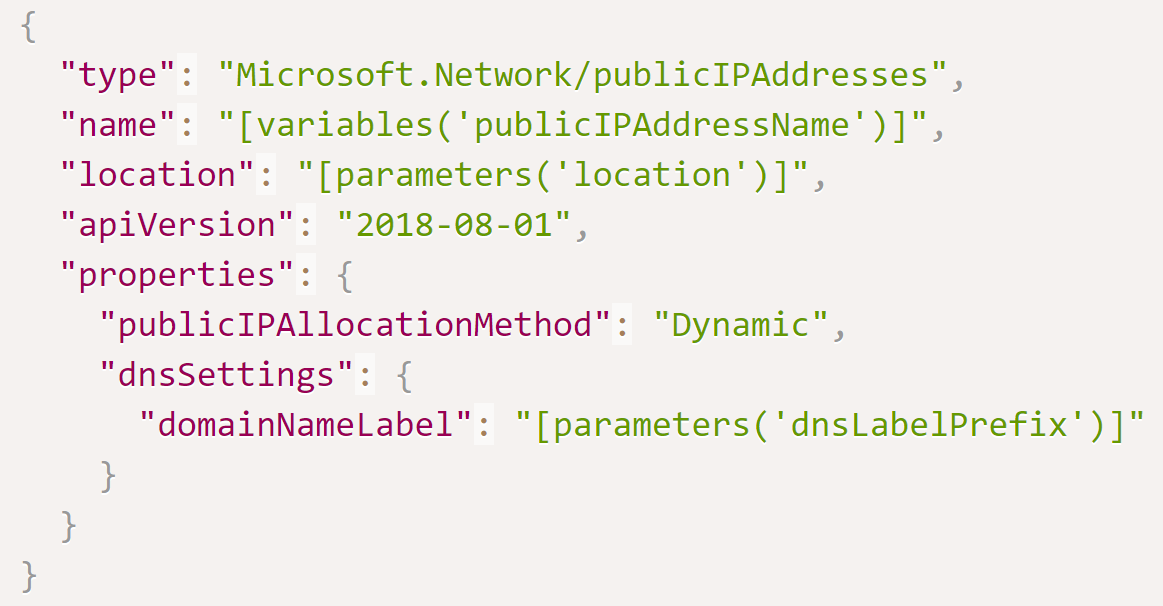
**Functions**

This section is where you define procedures that you don't want to repeat throughout the template. Similar to variables, functions can help make your templates easier to maintain.



Here's an example that creates a function to create a unique name, for use when creating resources that have globally unique naming requirements.

**Resources**



This section is where you define the Azure resources that make up your deployment.

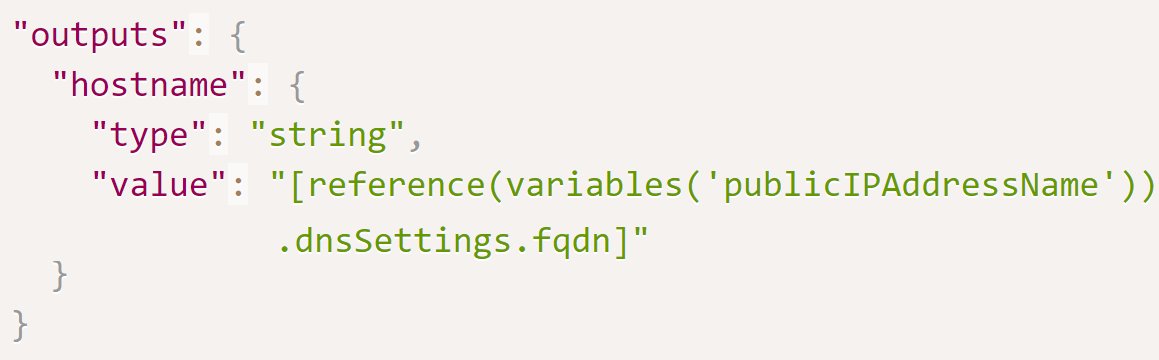
Here's an example that creates a public IP address resource.

Here, the type of resource is *Microsoft.Network/publicIPAddresses*. The **name** is read from the variables section, and the **location**, or *Azure region*, is read from the **parameters** section.

Because resource types can change over time, apiVersion refers to the version of the resource type you want to use. As resource types evolve, you can modify your templates to work with the latest features.

**Outputs**

This section is where you define any information you'd like to receive when the template runs. For example, you might want to receive your VM's IP address or fully qualified domain name (FQDN), information you do not know until the deployment runs.



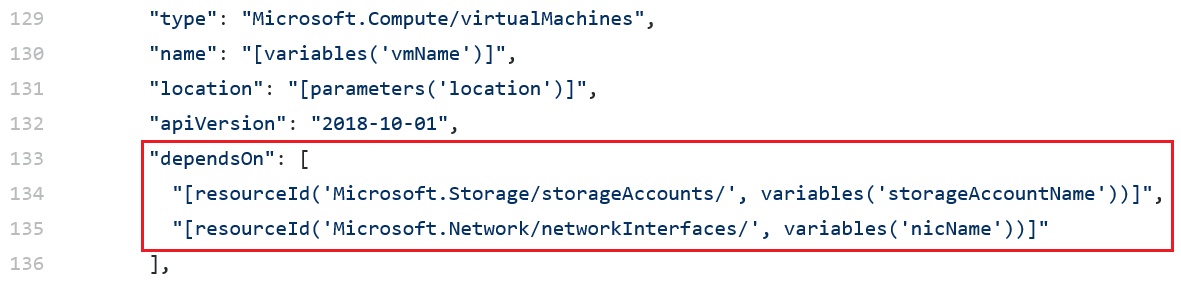
Here's an example that illustrates an output named **hostname**. The FQDN value is read from the VM's public IP address settings.

1. **How deployment dependencies are handled?**

For any given resource, other resources might need to exist before you can deploy the resource. For example, a SQL server must exist before attempting to deploy a SQL database. You can define this relationship by marking one resource as dependent on the other resource. You define a dependency with the **dependsOn** element, or by using the **reference** function.

Resource Manager evaluates the dependencies between resources, and deploys them in their dependent order. When resources aren't dependent on each other, Resource Manager deploys them in parallel. You only need to define dependencies for resources that are deployed in the same template.

Within your template, the **dependsOn** element enables you to define one resource as a dependent on one or more resources. Its value can be a comma-separated list of resource names.



1. **What are the deployment mode when templates are used?**

When deploying your resources using templates, you have three options available to you:

* **Validate**. This option compiles the templates, validates the deployment, ensures the template is functional (such as no circular dependencies), and the syntax is correct.
* **Incremental mode (default)**. This option only deploys whatever is defined in the template. It does not remove or modify any resources that are not defined in the template. For example, if you have deployed a VM via template, then renamed the VM in the template, the first VM deployed will still remain after the template is run again. This is the default mode.
* **Complete mode**: Resource Manager deletes resources that exist in the resource group, but aren't specified in the template. For example, only resources defined in the template will be present in the resource group after the template deploys. As best practice, use this mode for production environments where possible, to try to achieve idempotency in your deployment templates.

1. **What is CSE?**

With **Azure Custom Script Extension**(**CSE**) you can download and execute scripts on Azure virtual machines. This extension is useful for post deployment configuration, software installation, or any other configuration / management task. Scripts can be downloaded from Azure storage or GitHub, or provided to the Azure portal at extension run time.

There are multiple ways to execute a PowerShell script on a Windows Virtual machine in Azure.

1. PowerShell Remoting
2. Desired State Configuration script resource
3. Custom Script Extension

Whenever you want to configure something or update *virtual machine* (windows or Linux) like configure IIS , update firewall , create some folder, copy files etc.,  you need to take **RDP** (windows) or **SSH** (Linux) into *virtual machine*. Taking RDP or SSH access is not feasible in every situation. Your Virtual Network / Network Security Group (NSG) which contains list of security Rules can block you from taking direct access of your *Virtual Machine*. While doing automation of Virtual Machine/s Deployment you can update your VM using CSE. In your ARM template you can add new resource (section) **CSE** , this will get execute once your virtual machine is in ‘Running’ state. This Custom Script Extension is useful for post deployment configuration, software installation, or any other configuration / management task etc. This script file or zip (which include multiple scripts) can be downloaded from Azure storage, GitHub or any location which is available publicly via URL. CSE is integrated part of Azure Resource Manager Template, PowerShell, Azure Portal and REST API.

1. **How to check Status of CSE?**

Once you are ARM template is executed successfully you can check status of your CSE in Portal. Go to Virtual machine  -> Settings Extension tab. Here you will see all list of deployed extensions with its version, status, Type etc.

1. **How a resource can be deployed using Resource Templates?**

A resource can be deployed with Resource Manager templates using following methods:

* Azure Portal
* Azure Command Line Interface (CLI)
* PowerShell and
* REST-API

1. **How will you create VM using ARM template?**
2. **on-premise AD account vs. Azure AD?**
3. **What are Runbooks?**

<https://azure-overview.com/>

<https://stackify.com/service-fabric-misconceptions/>

<https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops>

# *.Net Core*

1. **What are the advantages of ASP.Net Core over ASP.Net?**

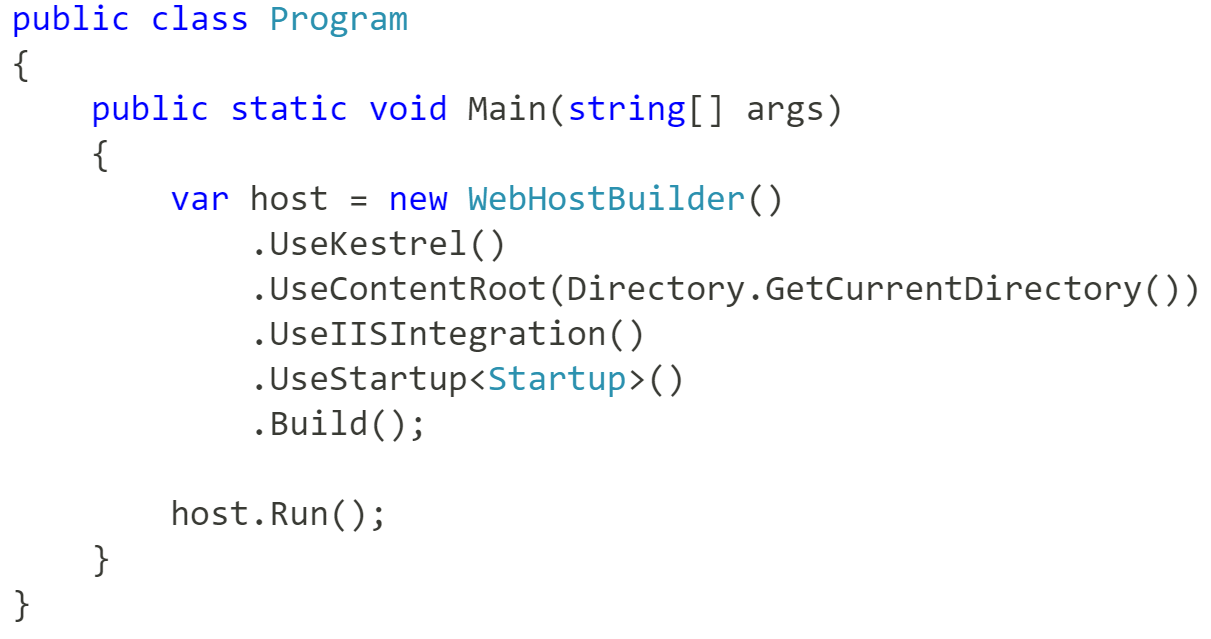
ASP.Net Core is a lightweight, open source, cross-platform framework for building modern cloud based web-apps on Windows, Linux or Mac-OS. It has built-in dependency injection. It has high performance than the ASP.Net. It can be hosted on Kestrel, IIS, HTTP.Sys, Nginx, Apache, Docker.

1. **Why Program class?**

An ASP.NET Core application is basically a console application that hosts a web server (Kestrel). Knowing this, there’s no surprise to find a Program class with the classic Main method we see on traditional console applications. This means we can do pretty much anything as we would on a console application before running the server, mainly things that we don’t want to do in the context of the ASP.NET Core application per se.

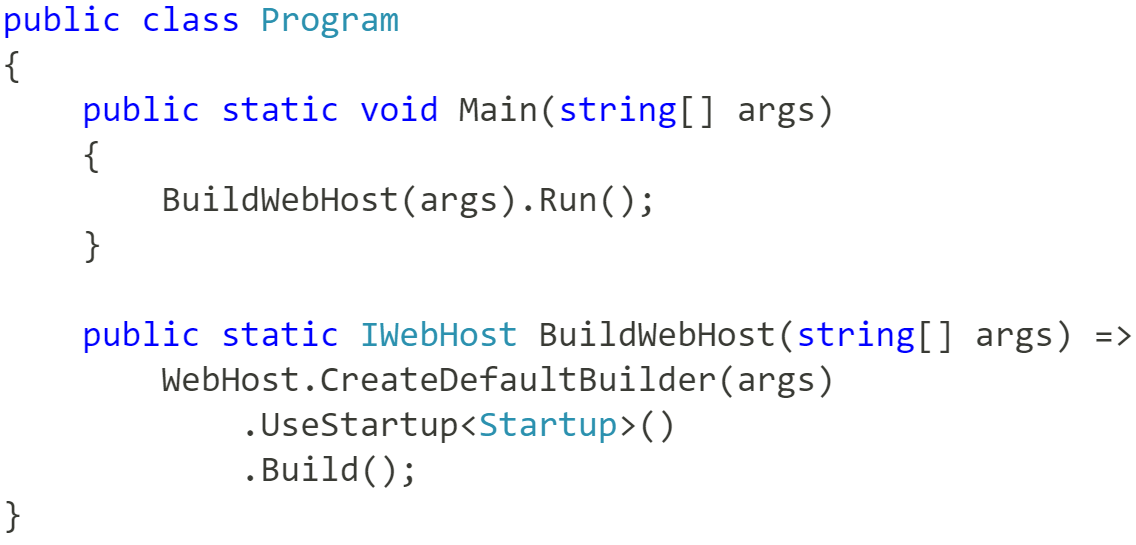
The other use of Program class is to create webhost that will listen to http requests. More specifically to create host. For web apps, webhost is created while for other type of applications, a generic host is created.

Here in the adjacent image, the host is set up with



* + - * A web server (Kestrel)
      * The content directory
      * IIS Integration
      * Startup class

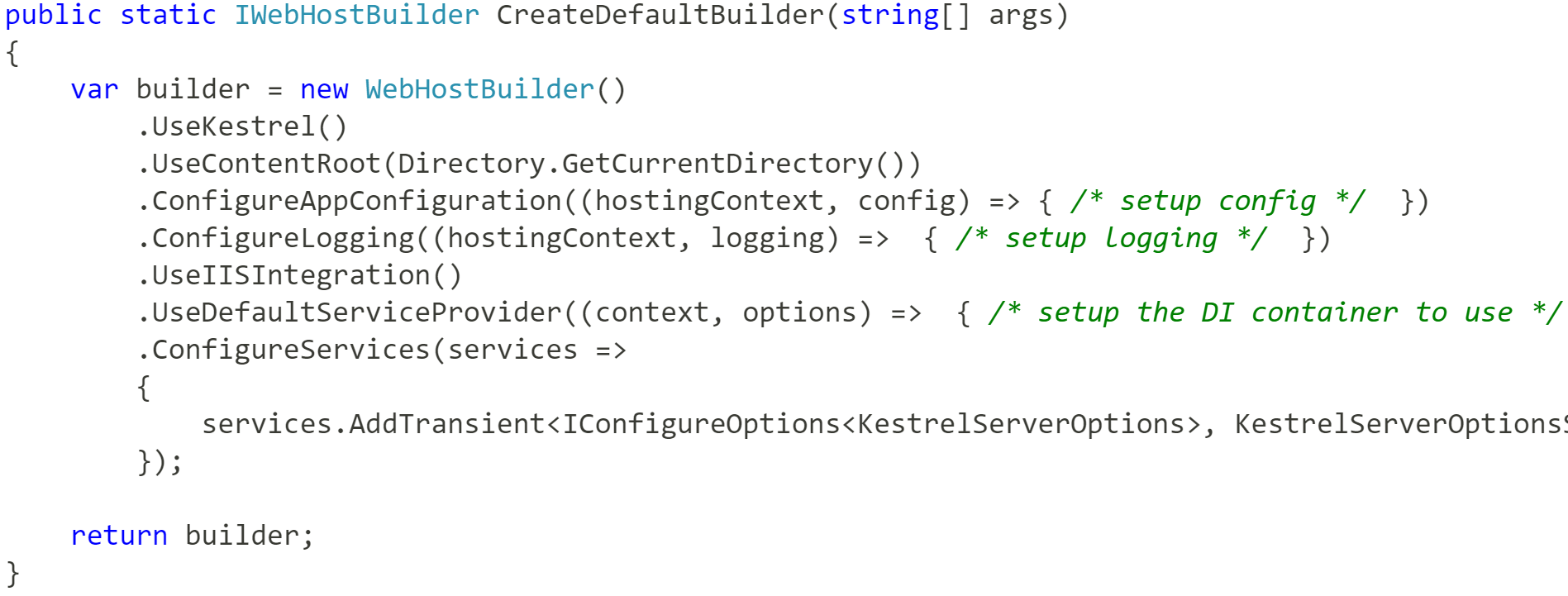
And finally it is build and Run. This was in ASP.Net **Core** **1.x**



In ASP.Net **Core 2.0**, the Program file looks something like the adjacent image.

The method, *CreateDefaultBuilder*, abstracts the inner details, as shown in the adjacent image. The implementation can be found in GitHub.

The ***UseStartup*** method tells the **WebHostBuilder** about the class that should be used for initialization purposes. Doing this is optional, as we can call ***Configure*** and ***ConfigureServices*** directly on the **WebHostBuilder**.



1. **Why Startup class?**

The startup class contains some initialization code for the ASP.Net Core application. The Startup class includes two methods: ConfigureServices and Configure.

* + - * ConfigureServices (optional): to register Services (reusable components) via Dependency Injection
      * Configure: To create app’s request processing pipeline.



<https://andrewlock.net/exploring-program-and-startup-in-asp-net-core-2-preview1-2/>

<https://blog.codingmilitia.com/2018/10/27/aspnet-004-from-zero-to-overkill-the-program-and-startup-classes>

# *Database Architecture*

1. **What are database cluster?**

Database Clustering is the process of combining more than one servers or instances connecting a single database. Sometimes one server may not be adequate to manage the amount of data or the number of requests, that is when a Data Cluster is needed. Database clustering, SQL server clustering, and SQL clustering are closely associated with SQL is the language used to manage the database information.

Clustering is when you have a group of machines (nodes) hosting the same database schema on the same database software with some form of data exchange between these machines. From outside of the cluster, these machines are seen as a single unit containing a union of the data that is spread across the nodes in the cluster. When your application accesses a cluster, the request is ultimately routed to a single node in the cluster for read or write operation.

The main reasons for database clustering are its advantages a server receives:

1. **Data redundancy**: Each node has exactly same data as all other nodes. If one node fails, the data is available in other nodes.
2. **Load balancing**: distributing workload to different computers in the cluster. More users can be supported. High spikes can be handled.
3. **High availability**: defined as the amount of time a database is available. In cluster, even if a server is down, the database will be still available.
4. **Monitoring and automation**: Cluster have advantage that allows to automate a lot of processes of a database, at the same time it permits to setup rules to warn potential issues.
5. **In How many ways, replication can be done in database cluster?**

Replication defines the method by which a set of servers remain synchronized without having to share the storage being able to be geographically disperse. It is a form of clustering where all nodes in the cluster have the same/identical schema and data. There are two main ways of going about it.

1. **Master-Master (or multi-master) replication**: Any server can update the database. It is usually taken care of by a different module within the database (or a whole different software running on top of them in some cases).

Downside is that it is very hard to do well, and some systems lose ACID properties when in this mode of replication.

Upside is that it is flexible and you can support the failure of any server while still having the database updated.

1. **master-slave replication**: There is only a single copy of authoritative data, which is the pushed to the slave servers.

Downside is that it is less fault tolerant, if the master dies, there are no further changes in the slaves.

Upside is that it is easier to do than multi-master and it usually preserve ACID properties.

The replication is useful in following scenarios:

1. **To have high availability of data**. Even if a node goes down, the data is still available from other nodes in the cluster.
2. **Separate nodes for write and read**. Data is replicated from the write cluster to all read clusters. This ensures that under high transaction volumes, the read operations are not getting delayed by write operations and vice-versa.
3. **Multiple nodes for read operations and for write operations**. This ensures that the read and the write operations are load balanced across multiple nodes resulting in higher scalability and data throughput.
4. **What is database partitioning?**

Partitioning are essentially the same cluster topology. All nodes in the cluster have identical schema, however the data is divided across nodes such that each node has only a subset of the data. No two nodes will have the same data. **Sharding** has its use in the following scenarios:

1. Each database node has an upper limit to the amount of data it can store. This limited is usually due to hardware configuration. For example, a node that has a 4TB hard disk attached, can store up to a maximum of 4TB of data. If the total volume of data exceeds this limit, it has to be **sharded**.
2. Suppose your dataset has 500 million records (rows) for a given table. In a replicated setup, when you fire a query against this table, it is executed in a single node for all 500 million records. In a sharded setup, the query would be split-up across shards and executed. This would be faster since a. the amount of records to query in each shard would be much lesser than 500 million and b. the queries across shards will be executed in parallel.
3. **What is Materialized view?**

A materialized view is a database object that contains the results of a query. They are similar to regular views, in that they are a logical view of your data (based on a select statement), however, the underlying query result set has been saved to a table. The upside of this is that when you query a materialized view, you are querying a table, which may also be indexed.

The good thing about **materialized view** is that the joins are resolved only once when the materialized view have been created, which saves the subsequent query time. The Downside though is that the data you get back from the **materialized view** is only as up to date as the last time the materialized view has been refreshed.

Materialized views can be set to refresh manually, on a set schedule, or *based on the database detecting a change in data from one of the underlying tables*. Materialized views can be incrementally updated by combining them with materialized view logs, which act as change data capture sources on the underlying tables.

# *Hands-On*

1. **How to create and deploy ARM template**

Step 1. Create a blank template

Step 2. Create a Resource Group

Step 3. Deploy the blank template

Step 4. Add resource and parameter

Step 5. Add Function

**Step 1.**

Create a blank template file in VS Code and name it as AzureDeploy.json. Paste the below code.

{

"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"resources": []

}

**Step 2**.

When you deploy a template, you specify a resource group that will contain the resources. Before running the deployment command, create the resource group with either Azure CLI or Azure PowerShell.

New-AzResourceGroup `

-Name premTrialResGrp `

-Location "Central US"

**Step 3**.

To deploy the template, use either Azure CLI or Azure PowerShell. Use the resource group you created. Give a name to the deployment so you can easily identify it in the deployment history.

New-AzResourceGroupDeployment `

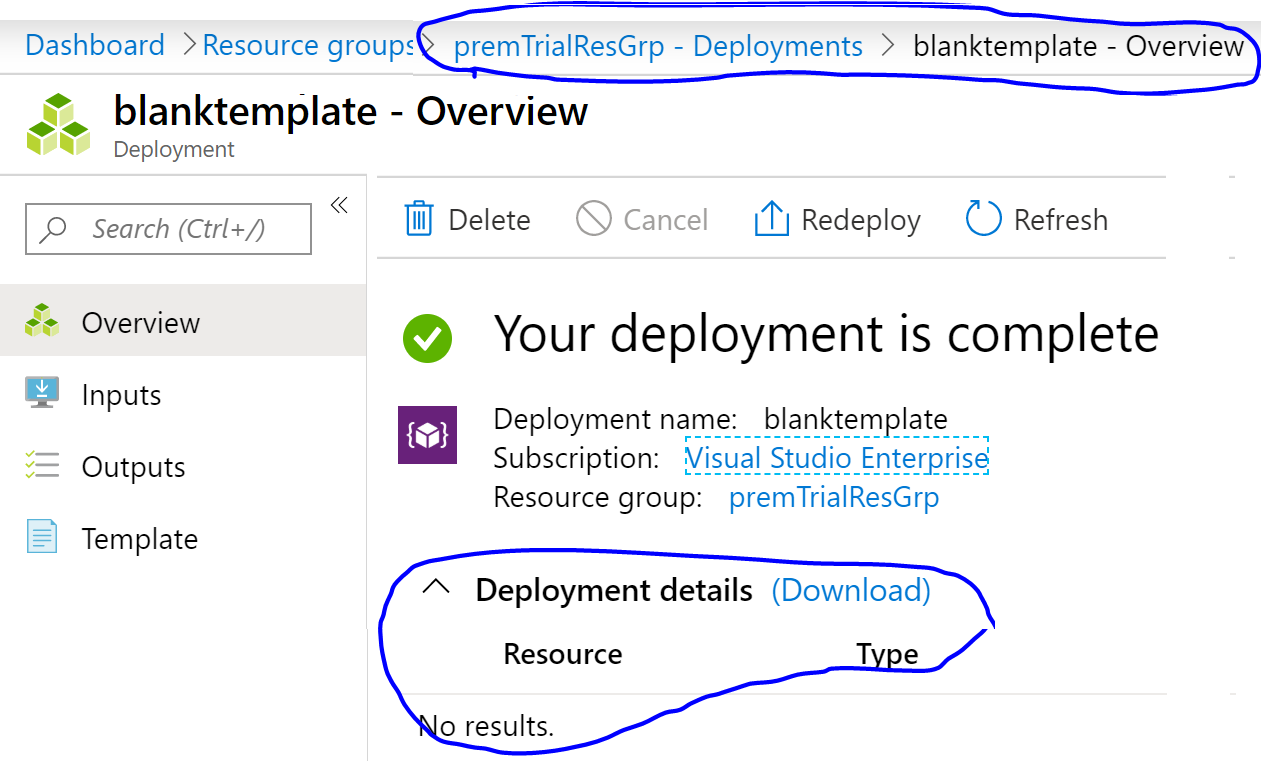
-Name blanktemplate `

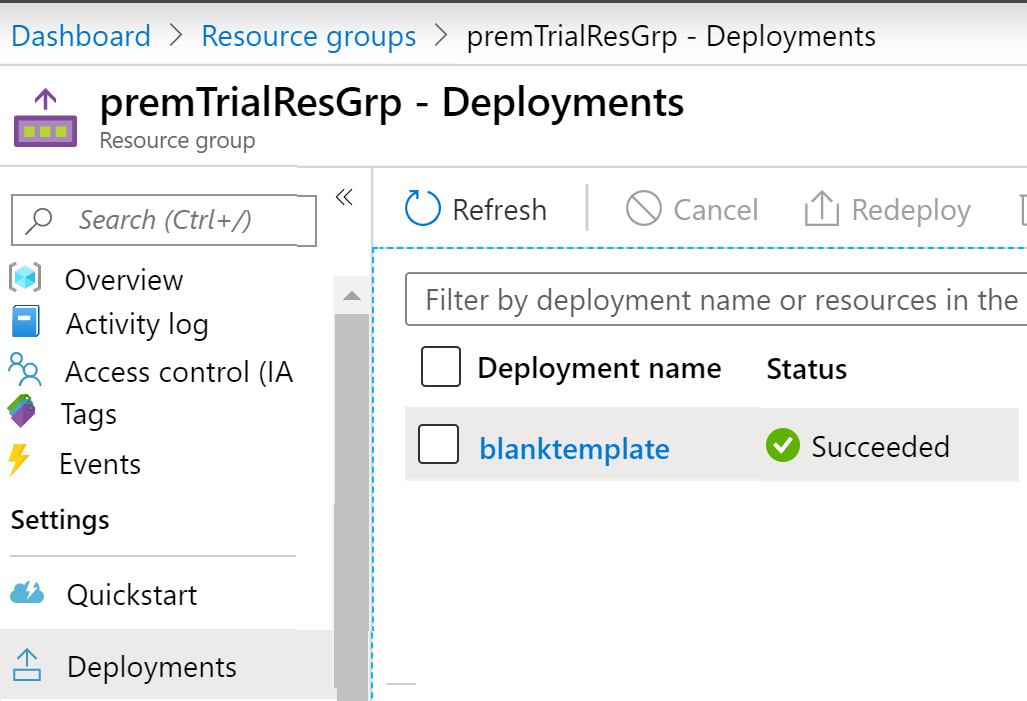
-ResourceGroupName premTrialResGrp `

-TemplateFile $templateFile

The cmdLet, **New**-**AzResourceGroupDeployment**, adds a deployment to an existing resource group. This includes the resources that the deployment requires. An Azure resource group is a collection of Azure resources that are deployed as a unit. A resource group deployment uses a template to add resources to a resource group and publishes them so that they are available in Azure. To add resources to a resource group without using a template, use the **New-AzResource** cmdlet. To add a resource group deployment, specify the name of an existing resource group and a resource group template.

When the above code is executed, under the **Resource Group** ***premTrialResGrp***, a new deployment entry is created, called ***blanktemplate***.





**Step 4**.

We need to add resources, like storage account, to the resource group. We can hard code the value however for every deployment we need to change the name. So, to overcome that we need parameter. The below code snippet, in yellow, adds the parameter functionality.

(File: AzureDeploy.json)

{

"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storageName": {

"type": "string",

"minLength": 3,

"maxLength": 24

}

},

"resources": [

{

"type": "Microsoft.Storage/storageAccounts",

"apiVersion": "2019-04-01",

"name": "[parameters('storageName')]",

"location": "eastus",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

]

}

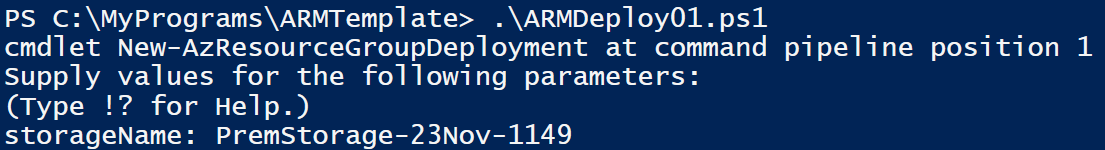
(File: ARMDeploy.ps1)

$templateFile = "C:\MyPrograms\ARMTemplate\azureDeploy.json"

New-AzResourceGroupDeployment `

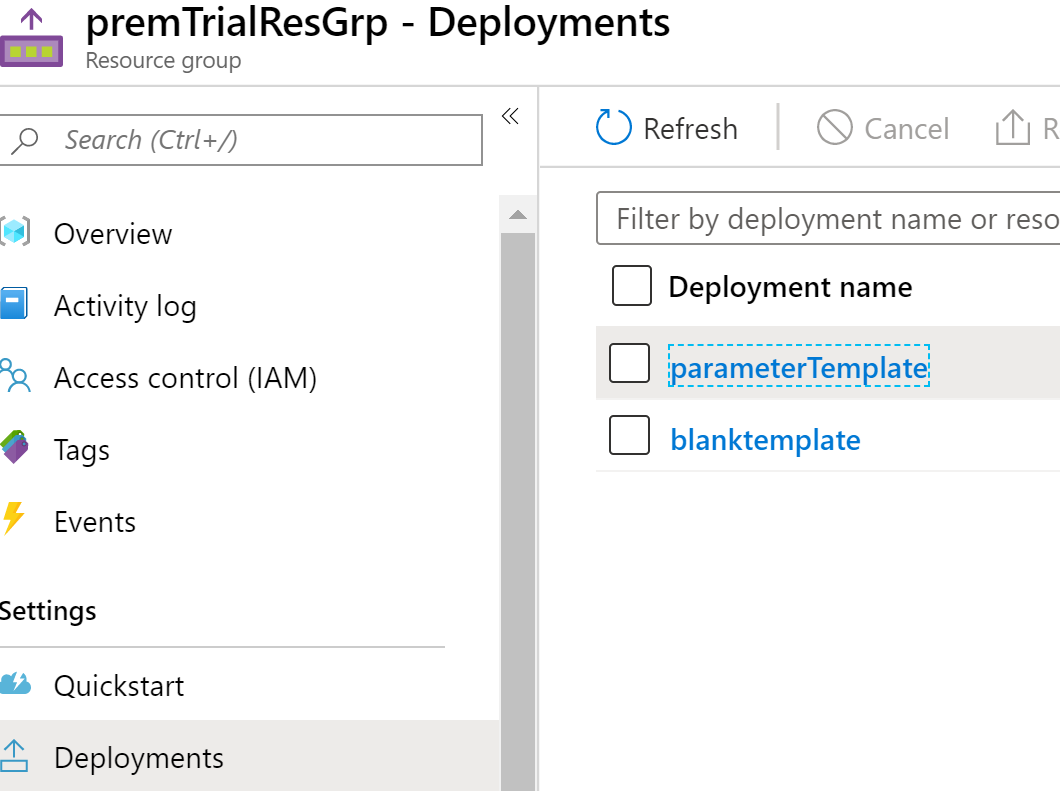
-Name parameterTemplate `

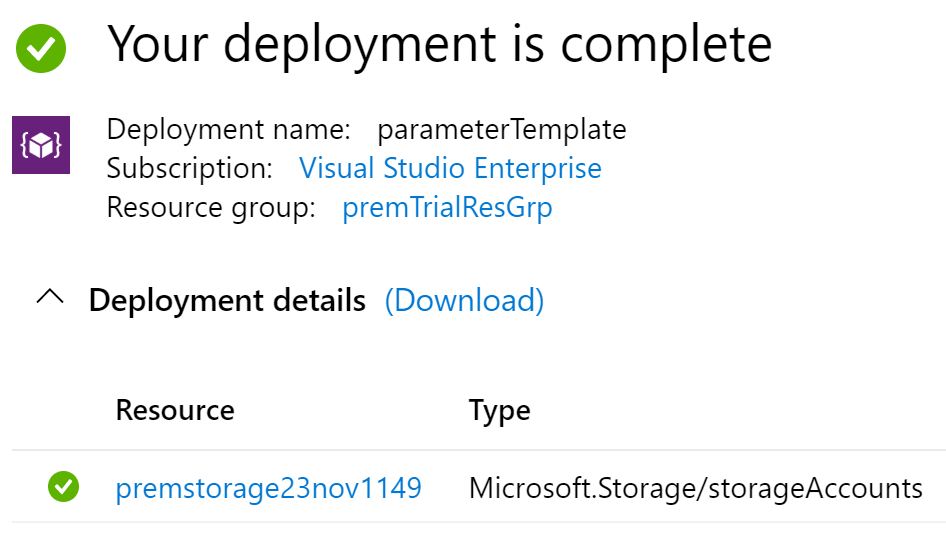
-ResourceGroupName premTrialResGrp `



-TemplateFile $templateFile

After the execution of the command, the resource group now has two deployments: blanktemplate and parameterTemplate. Under parameterTemplate deployment, a storage account with the name premstorage23Nov1149 is created.





The parameter values can be passed via any external file as well.

**Understand resource updates**

In the previous section, you deployed a storage account with the same name that you had created earlier. You may be wondering how the resource is affected by the redeployment.

If the resource already exists and no change is detected in the properties, no action is taken. If the resource already exists and a property has changed, the resource is updated. If the resource doesn't exist, it's created.

This way of handling updates means your template can include all of the resources you need for an Azure solution. You can safely redeploy the template and know that resources are changed or created only when needed. For example, if you have added files to your storage account, you can redeploy the storage account without losing those files.

**Step 5**.

You use functions to dynamically construct values. In addition to these system-provided template functions, you can also create user-defined functions. The below code snippet, in yellow, defines a simple function that appends a constant string before the name of storage account. Just like previously, the storage account name will be prompted. Once supplied, the value “prem” will be suffixed to it.

{

"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storageName": {

"type": "string",

"minLength": 3,

"maxLength": 24

}

},

"functions": [{

"namespace": "premNameSpace",

"members": {

"getName": {

"parameters": [

{

"name": "id",

"type": "string"

}

],

"output": {

"type": "string",

"value": "[concat('prem', toLower(parameters('id')))]"

}

}

}

}],

"resources": [

{

"type": "Microsoft.Storage/storageAccounts",

"apiVersion": "2019-04-01",

"name": "[premNamespace.getName(parameters('storageName'))]",

"location": "eastus",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

],

"outputs": {

"storageEndpoint": {

"type": "object",

"value": "[reference(premNameSpace.getName(parameters('storageName'))).primaryEndpoints]"

}

}

}

In the similar way, we can add variables in variable section.

**Step 6**.

You use outputs when you need a value from a deployed resource. You can use outputs to return values from the template. For example, it might be helpful to get the endpoints for your new storage account. The above code in green, returns the endpoint of the storage account created.

