**Explore Azure App Service**

**Azure App Service**

* Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends.
* You can develop in your favorite programming language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python.
* Applications run and scale with ease on both Windows and Linux-based environments.

**Examine Azure App Service plans**

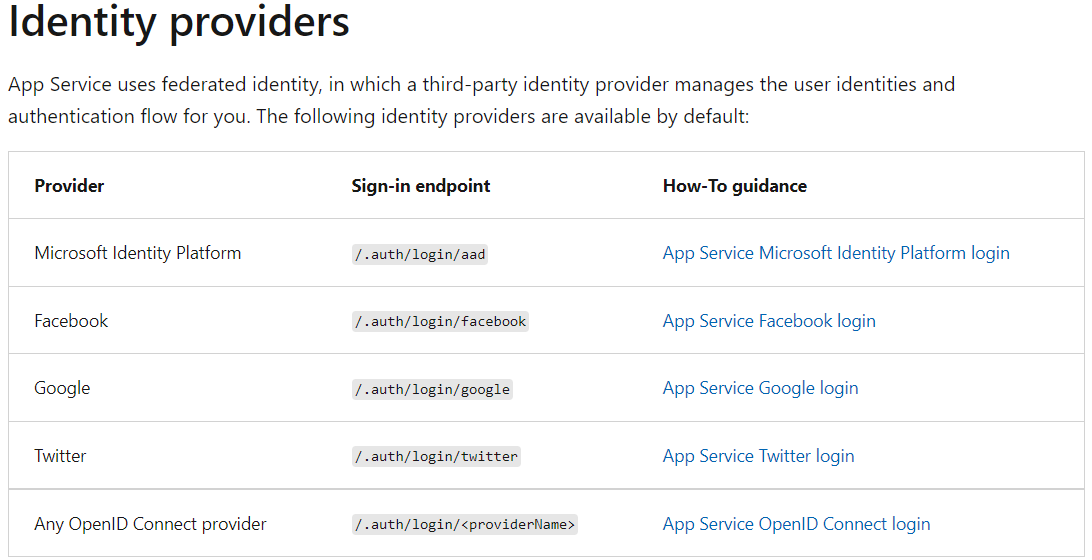
* In App Service, an app (Web Apps, API Apps, or Mobile Apps) always runs in an App Service plan.
* An App Service plan defines a set of compute resources for a web app to run. One or more apps can be configured to run on the same computing resources (or in the same App Service plan). In addition, Azure Functions also has the option of running in an 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, PremiumV3, Isolated)
* The pricing tier of an App Service plan determines what App Service features you get and how much you pay for the plan. There are a few categories of pricing tiers:
  + **Shared compute:** Both Free and Shared share the resource pools of your apps with the apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources can't scale out.
  + **Dedicated compute:** The Basic, Standard, Premium, PremiumV2, and PremiumV3 tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances are available to you for scale-out.
  + **Isolated**: This tier runs dedicated Azure VMs on dedicated Azure Virtual Networks. It provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.
  + **Consumption**: This tier is only available to function apps. It scales the functions dynamically depending on workload.
* **How does my app run and scale**
  + An app runs on all the VM instances configured in the App Service plan.
  + If multiple apps are in the same App Service plan, they all share the same VM instances.
  + If you have multiple deployment slots for an app, all deployment slots also run on the same VM instances.
  + If you enable diagnostic logs, perform backups, or run WebJobs, they also use CPU cycles and memory on these VM instances.
  + If the plan is configured to run five VM instances, then all apps in the plan run on all five instances. If the plan is configured for autoscaling, then all apps in the plan are scaled out together based on the autoscale settings.
* **What if my app needs more capabilities or features?**
  + If your app is in the same App Service plan with other apps, you may want to improve the app's performance by isolating the compute resources. You can do it by moving the app into a separate App Service plan.
  + Isolate your app into a new App Service plan when:
    - The app is resource-intensive.
    - You want to scale the app independently from the other apps in the existing plan.
    - The app needs resource in a different geographical region.

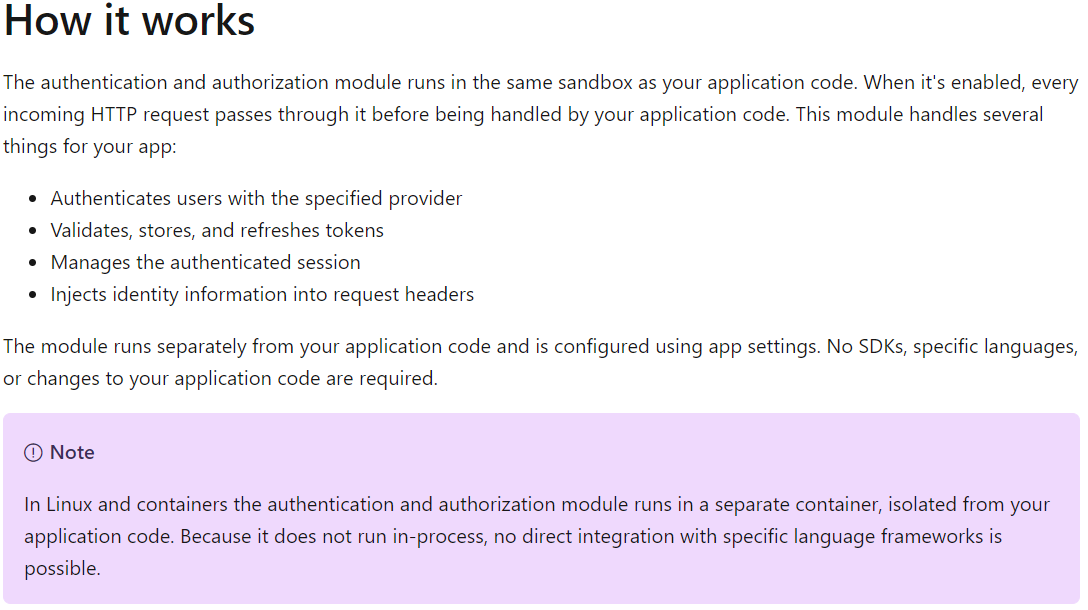
**Deploy to App Service**

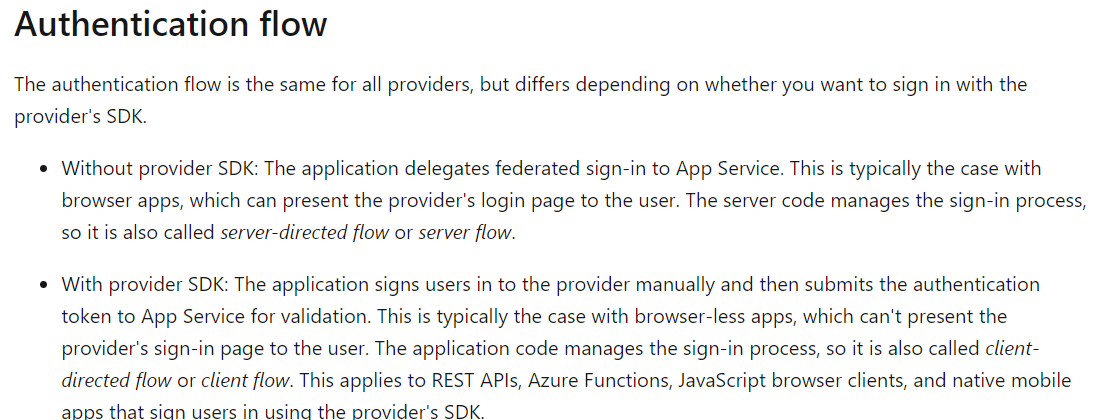
* App Service supports both automated and manual deployment.
* **Automated deployment**
  + Automated deployment, or continuous integration, is a process used to push out new features and bug fixes in a fast and repetitive pattern with minimal impact on end users.
  + Azure supports automated deployment directly from several sources. The following options are available:
    - **Azure DevOps:** You can push your code to Azure DevOps, build your code in the cloud, run the tests, generate a release from the code, and finally, push your code to an Azure Web App.
    - **GitHub:** Azure supports automated deployment directly from GitHub. When you connect your GitHub repository to Azure for automated deployment, any changes you push to your production branch on GitHub will be automatically deployed for you.
    - **Bitbucket:** With its similarities to GitHub, you can configure an automated deployment with Bitbucket.
* **Manual deployment**
  + There are a few options that you can use to manually push your code to Azure:
    - **Git:** App Service web apps feature a Git URL that you can add as a remote repository. Pushing to the remote repository will deploy your app.
    - **CLI:** webapp up is a feature of the az command-line interface that packages your app and deploys it. Unlike other deployment methods, az webapp up can create a new App Service web app for you if you haven't already created one.
    - **Zip deploy:** Use curl or a similar HTTP utility to send a ZIP of your application files to App Service.
    - **FTP/S:** FTP or FTPS is a traditional way of pushing your code to many hosting environments, including App Service.
* **Use deployment slots**
  + Whenever possible, use deployment slots when deploying a new production build. When using a Standard App Service Plan tier or better, you can deploy your app to a staging environment and then swap your staging and production slots. The swap operation warms up the necessary worker instances to match your production scale, thus eliminating downtime.

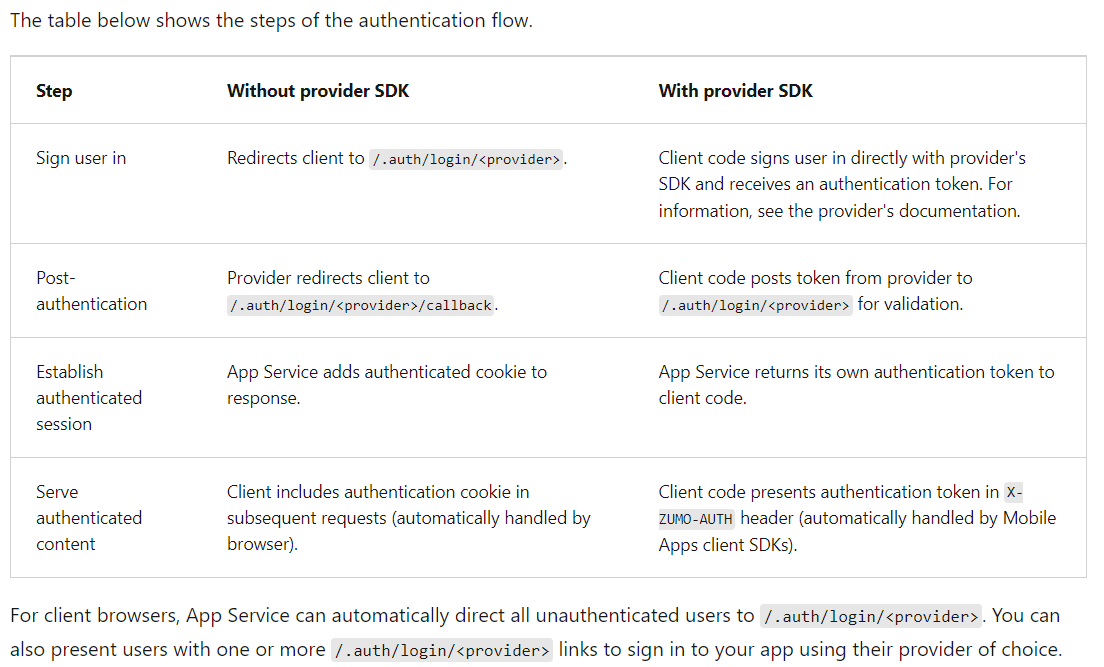
**Explore authentication and authorization in App Service**

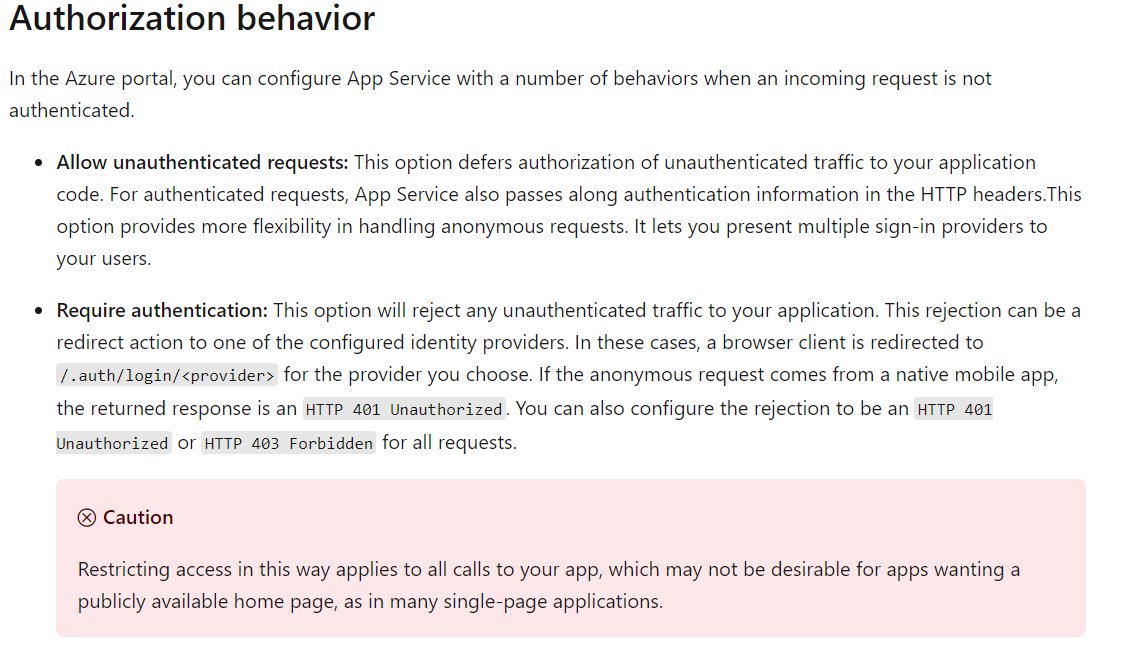
* Azure App Service provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal or no code in your web app, API, and mobile back end, and also Azure Functions.
* If you need more flexibility than App Service provides, you can also write your own utilities.
* Azure App Service allows you to integrate a variety of auth capabilities into your web app or API without implementing them yourself.
* It’s built directly into the platform and doesn’t require any particular language, SDK, security expertise, or even any code to utilize.
* You can integrate with multiple login providers. For example, Azure AD, Facebook, Google, Twitter.











**Discover App Service networking features**

* By default, apps hosted in App Service are accessible directly through the internet and can reach only internet-hosted endpoints. But for many applications, you need to control the inbound and outbound network traffic.
* There are two main deployment types for Azure App Service.
  + The multitenant public service hosts App Service plans in the Free, Shared, Basic, Standard, Premium, PremiumV2, and PremiumV3 pricing SKUs.
  + There is also the single-tenant App Service Environment (ASE) hosts Isolated SKU App Service plans directly in your Azure virtual network.

**Configure application settings**

* In App Service, app settings are variables passed as environment variables to the application code.
* For Linux apps and custom containers, App Service passes app settings to the container using the --env flag to set the environment variable in the container.
* For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in <appSettings> in Web.config or appsettings.json, but the values in App Service override the ones in Web.config or appsettings.json.
* You can keep development settings (for example, local MySQL password) in Web.config or appsettings.json, but production secrets (for example, Azure MySQL database password) safe in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.
* App settings are always encrypted when stored (encrypted-at-rest).

**Add log messages in code**

* ASP.NET
  + System.Diagnostics.Trace.TraceError("If you're seeing this, something bad happened");
* ASP.NET Core
  + By default, ASP.NET Core uses the Microsoft.Extensions.Logging.AzureAppServices logging provider.

**Access log files**

* For logs stored in the App Service file system, the easiest way is to download the ZIP file in the browser at:
  + Linux/container apps: https://<app-name>.scm.azurewebsites.net/api/logs/docker/zip
  + Windows apps: https://<app-name>.scm.azurewebsites.net/api/dump
* For Linux/container apps, the ZIP file contains console output logs for both the docker host and the docker container. For a scaled-out app, the ZIP file contains one set of logs for each instance.
* In the App Service file system, these log files are the contents of the /home/LogFiles directory.

**Webspace**

* A certificate uploaded into an app is stored in a deployment unit that is bound to the app service plan's resource group and region combination (internally called a webspace).
* This makes the certificate accessible to other apps in the same resource group and region combination.

**Private certificate requirements**

* The free **App Service managed certificate** and the **App Service certificate** already satisfy the requirements of App Service.
* If you want to use a private certificate in App Service, your certificate must meet the following requirements:
  + Exported as a password-protected PFX file, encrypted using triple DES.
  + Contains private key at least 2048 bits long
  + Contains all intermediate certificates in the certificate chain
  + To secure a custom domain in a TLS binding, the certificate has additional requirements:
    - Contains an Extended Key Usage for server authentication (OID = 1.3.6.1.5.5.7.3.1)
    - Signed by a trusted certificate authority

**Creating a free managed certificate**

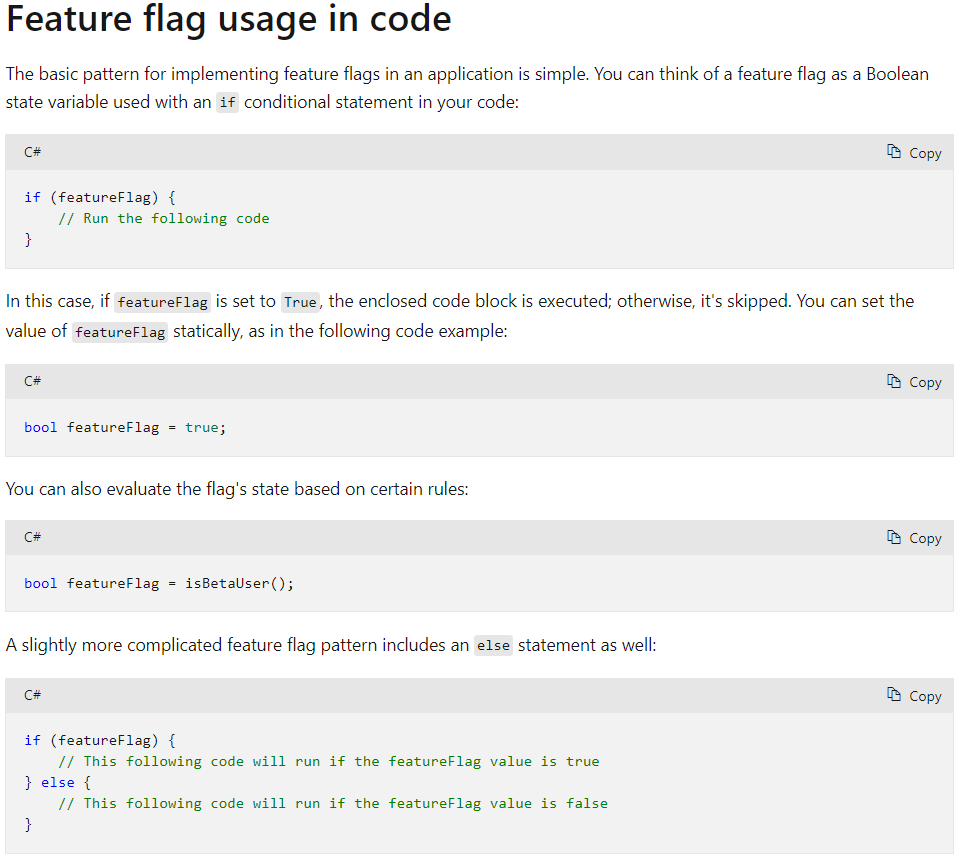
* To create custom TLS/SSL bindings or enable client certificates for your App Service app, your App Service plan must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier.
* Custom SSL is not supported in the **F1** or **D1** tier.
* The free App Service managed certificate is a turn-key solution for securing your custom DNS name in App Service.
* It's a TLS/SSL server certificate that's fully managed by App Service and renewed continuously and automatically in six-month increments, 45 days before expiration.
* You create the certificate and bind it to a custom domain, and let App Service do the rest.
* The free certificate comes with the following limitations:
  + Does not support wildcard certificates.
  + Does not support usage as a client certificate by certificate thumbprint.
  + Is not exportable.
  + Is not supported on App Service Environment (ASE).
  + Is not supported with root domains that are integrated with Traffic Manager.
  + If a certificate is for a CNAME-mapped domain, the CNAME must be mapped directly to <app-name>.azurewebsites.net.

**Import an App Service Certificate**

* If you purchase an App Service Certificate from Azure, Azure manages the following tasks:
  + Takes care of the purchase process from **GoDaddy**.
  + Performs domain verification of the certificate.
  + Maintains the certificate in Azure Key Vault.
  + Manages certificate renewal.
  + Synchronize the certificate automatically with the imported copies in App Service apps.

**Manage app features (Feature Flags, Feature Toggles, Feature Switches)**

* Feature management is a modern software-development practice that decouples feature release from code deployment and enables quick changes to feature availability on demand.
* It uses a technique called feature flags (also known as feature toggles, feature switches, and so on) to dynamically administer a feature's lifecycle.
* Here are several new terms related to feature management:
  + **Feature flag:**
    - A feature flag is a variable with a binary state of on or off.
    - The feature flag also has an associated code block.
    - The state of the feature flag triggers whether the code block runs or not.
  + **Feature manager:**
    - A feature manager is an application package that handles the lifecycle of all the feature flags in an application.
    - The feature manager typically provides additional functionality, such as caching feature flags and updating their states.
  + **Filter**:
    - A filter is a rule for evaluating the state of a feature flag.
    - A user group, a device or browser type, a geographic location, and a time window are all examples of what a filter can represent.
* An effective implementation of feature management consists of at least two components working in concert:
  + An application that makes use of feature flags.
  + A separate repository that stores the feature flags and their current states.



**Feature flag declaration**

* Each feature flag has two parts: a name and a list of one or more filters that are used to evaluate if a feature's state is on (that is, when its value is True).
* A filter defines a use case for when a feature should be turned on.
* When a feature flag has multiple filters, the filter list is traversed in order until one of the filters determines the feature should be enabled. At that point, the feature flag is on, and any remaining filter results are skipped. If no filter indicates the feature should be enabled, the feature flag is off.
* The feature manager supports appsettings.json as a configuration source for feature flags. The following example shows how to set up feature flags in a JSON file:



**Examine autoscale factors**

**What is autoscaling**

* Autoscaling is a cloud system or process that adjusts available resources based on the current demand. Autoscaling performs scaling in and out, as opposed to scaling up and down.
* Autoscaling can be triggered according to a schedule, or by assessing whether the system is running short on resources.
* For example, autoscaling could be triggered if CPU utilization grows, memory occupancy increases, the number of incoming requests to a service appears to be surging, or some combination of factors.

**Azure App Service Autoscaling**

* Autoscaling in Azure App Service monitors the resource metrics of a web app as it runs.
* It detects situations where additional resources are required to handle an increasing workload, and ensures those resources are available before the system becomes overloaded.
* Autoscaling can also deallocate resources when the workload has diminished.
* Autoscaling responds to changes in the environment by adding or removing web servers and balancing the load between them. Autoscaling doesn't have any effect on the CPU power, memory, or storage capacity of the web servers powering the app, it only changes the number of web servers.

**Autoscaling rules**

* Autoscaling makes its decisions based on rules that you define.
* A rule specifies the threshold for a metric, and triggers an autoscale event when this threshold is crossed.
* Autoscaling can also deallocate resources when the workload has diminished.

**When should you consider autoscaling?**

1. **Autoscaling provides elasticity for your services.**
   * It's a suitable solution when hosting any application when you can't easily predict the workload in advance, or when the workload is likely to vary by date or time. For example, you might expect increased/reduced activity for a business app during holidays.
2. **Autoscaling improves availability and fault tolerance.**
3. **Autoscaling works by adding or removing web servers.**
4. **Autoscaling isn't the best approach to handling long-term growth.**
   * You might have a web app that starts with a small number of users, but increases in popularity over time. Autoscaling has an overhead associated with monitoring resources and determining whether to trigger a scaling event. In this scenario, if you can anticipate the rate of growth, manually scaling the system over time may be a more cost effective approach.
5. **The number of instances of a service is also a factor.**

**Identify autoscale factors**

* You can configure autoscaling to detect when to scale in and out according to a combination of factors, based on resource usage.
* You can also configure autoscaling to occur according to a schedule.

**App Service Plan**

* Not all App Service Plan pricing tiers support autoscaling.
* Autoscaling is a feature of the App Service Plan used by the web app. When the web app scales out, Azure starts new instances of the hardware defined by the App Service Plan to the app.
* To prevent runaway autoscaling, an App Service Plan has an instance limit. Plans in more expensive pricing tiers have a higher limit. Autoscaling cannot create more instances than this limit.

**Autoscale conditions**

* You indicate how to autoscale by creating autoscale conditions. Azure provides two options for autoscaling:
* **Scale based on a metric**, such as the length of the disk queue, or the number of HTTP requests awaiting processing.
* **Scale to a specific instance count according to a schedule**.
  + For example, you can arrange to scale out at a particular time of day, or on a specific date or day of the week. You also specify an end date, and the system will scale back in at this time.
* Scaling to a specific instance count only enables you to scale out to a defined number of instances. If you need to scale out incrementally, you can combine metric and schedule-based autoscaling in the same autoscale condition. So, you could arrange for the system to scale out if the number of HTTP requests exceeds some threshold, but only between certain hours of the day.
* You can create multiple autoscale conditions to handle different schedules and metrics. Azure will autoscale your service when any of these conditions apply.
* An App Service Plan also has a default condition that will be used if none of the other conditions are applicable. This condition is always active and doesn't have a schedule.

**Metrics for autoscale rules**

* Autoscaling by metric requires that you define one or more autoscale rules. An autoscale rule specifies a metric to monitor, and how autoscaling should respond when this metric crosses a defined threshold.
* The metrics you can monitor for a web app are:
  + **CPU Percentage.** This metric is an indication of the CPU utilization across all instances. A high value shows that instances are becoming CPU-bound, which could cause delays in processing client requests.
  + **Memory Percentage**. This metric captures the memory occupancy of the application across all instances. A high value indicates that free memory could be running low, and could cause one or more instances to fail.
  + **Disk Queue Length.** This metric is a measure of the number of outstanding I/O requests across all instances. A high value means that disk contention could be occurring.
  + **Http Queue Length.** This metric shows how many client requests are waiting for processing by the web app. If this number is large, client requests might fail with HTTP 408 (Timeout) errors.
  + **Data In**. This metric is the number of bytes received across all instances.
  + **Data Out.** This metric is the number of bytes sent by all instances.
  + You can also scale based on metrics for other Azure services. For example, if the web app processes requests received from a Service Bus Queue, you might want to spin up additional instances of a web app if the number of items held in an Azure Service Bus Queue exceeds a critical length.

**How an autoscale rule analyzes metrics**

* Autoscaling works by analyzing trends in metric values over time across all instances. Analysis is a multi-step process.
* **In the first step,** 
  + an autoscale rule aggregates the values retrieved for a metric for all instances across a period of time known as the time grain. Each metric has its own intrinsic time grain, but in most cases this period is 1 minute. The aggregated value is known as the time aggregation. The options available are Average, Minimum, Maximum, Sum, Last, and Count.
  + An interval of one minute is a very short interval in which to determine whether any change in metric is long-lasting enough to make autoscaling worthwhile.
* **So, an autoscale rule performs a second step** that performs a further aggregation of the value calculated by the time aggregation over a longer, user-specified period, known as the Duration. The minimum Duration is 5 minutes. If the Duration is set to 10 minutes for example, the autoscale rule will aggregate the 10 values calculated for the time grain.
* The aggregation calculation for the Duration can be different from that of the time grain. For example, if the time aggregation is Average and the statistic gathered is CPU Percentage across a one-minute time grain, each minute the average CPU percentage utilization across all instances for that minute will be calculated. If the time grain statistic is set to Maximum, and the Duration of the rule is set to 10 minutes, the maximum of the 10 average values for the CPU percentage utilization will be used to determine whether the rule threshold has been crossed.

**Autoscale actions**

* An autoscale action uses an operator (such as less than, greater than, equal to, and so on) to determine how to react to the threshold.
* An autoscale action can be
  + **scale-out** 
    - A scale-out action increases the number of instances
    - Scale-out actions typically use the greater than operator to compare the metric value to the threshold.
  + **scale-in.**
    - A scale-in action reduces the instance count.
    - Scale-in actions tend to compare the metric value to the threshold with the less than operator.
* An autoscale action can also set the instance count to a specific level, rather than incrementing or decrementing the number available.
* **An autoscale action has a cool down period, specified in minutes.** 
  + During this interval, the scale rule won't be triggered again.
  + This is to allow the system to stabilize between autoscale events.
  + Remember that it takes time to start up or shut down instances, and so any metrics gathered might not show any significant changes for several minutes.
  + The minimum cool down period is five minutes.

**Combining autoscale rules**

* A single autoscale condition can contain several autoscale rules (for example, a scale-out rule and the corresponding scale-in rule).
* However, the autoscale rules in an autoscale condition don't have to be directly related.
* You could define the following four rules in the same autoscale condition:
  + If the HTTP queue length exceeds 10, scale out by 1
  + If the CPU utilization exceeds 70%, scale out by 1
  + If the HTTP queue length is zero, scale in by 1
  + If the CPU utilization drops below 50%, scale in by 1

**Route traffic in App Service**

* By default, all client requests to the app's production URL (http://<app\_name>.azurewebsites.net) are routed to the production slot.
* You can route a portion of the traffic to another slot. This feature is useful if you need user feedback for a new update, but you're not ready to release it to production.
* You can swap deployment slots on your app's Deployment slots page and the Overview page.
* Configure auto swap
  + Auto swap streamlines Azure DevOps scenarios where you want to deploy your app continuously with zero cold starts and zero downtime for customers of the app.
  + When auto swap is enabled from a slot into production, every time you push your code changes to that slot, App Service automatically swaps the app into production after it's warmed up in the source slot.
* Route production traffic automatically
  + After a client is automatically routed to a specific slot, it's "pinned" to that slot for the life of that client session. On the client browser, you can see which slot your session is pinned to by looking at the **x-ms-routing-name** cookie in your HTTP headers.
  + A request that's routed to the "staging" slot has the cookie **x-ms-routing-name=staging**.
  + A request that's routed to the production slot has the cookie **x-ms-routing-name=self**.

