Azure App Services

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.
These compute resources are analogous to the *server farm* in conventional web hosting.
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)

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.
One or more apps can be configured to run on the same computing resources (or in the same App Service plan).
Whatever apps you put into this App Service plan run on these compute resources as defined by your 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**, run apps 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. App receives CPU minutes on a shared VM instance and 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. When the app runs, it 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. The App Service plan is the scale unit of the App Service apps. 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.
- **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.

In a nutshell: Shared compute resources - Isolate Compute only - Isolate Compute and Network.

Azure App Service brings together everything you need to create websites, mobile backends, and web APIs for any platform or device.

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. Within App Service, Hybrid Connections can be used to access application resources in other networks. It provides access from your app to an application endpoint. Each Hybrid Connection correlates to a single TCP host and port combination. This means that the Hybrid Connection endpoint can be on any operating system and any application, provided you are accessing a TCP listening port. The Hybrid Connections feature does not know or care what the application protocol is, or what you are accessing. It is simply providing network access.

Each tier also provides a specific subset of App Service features. These features include custom domains and SSL certificates, autoscaling, deployment slots, backups, Traffic Manager integration, and more. The higher the tier, the more features are available.

Windows Base:

	FREE **Try for free**	**SHARED** **Environment for dev/test**	**BASIC** **Dedicated environment for dev/test**	**STANDARD** **Run production workloads**	**PREMIUM** **Enhanced performance and scale**	**ISOLATED** **High-Performance, Security and Isolation**
Web, mobile, or API apps	10	100	Unlimited	Unlimited	Unlimited	Unlimited
Disk space	1 GB	1 GB	10 GB	50 GB	250 GB	1 TB
Maximum instances	–	–	Up to 3	Up to 10	Up to 30**	Up to 100*

Custom domain	–	Supported	Supported	Supported	Supported	Supported
Auto Scale	–	–	–	Supported	Supported	Supported
VPN hybrid connectivity	–	–	–	Supported	Supported	Supported
Network Isolation						Supported
Price per hour	Free	$0.013	$0.075	$0.10	$0.20	$0.40

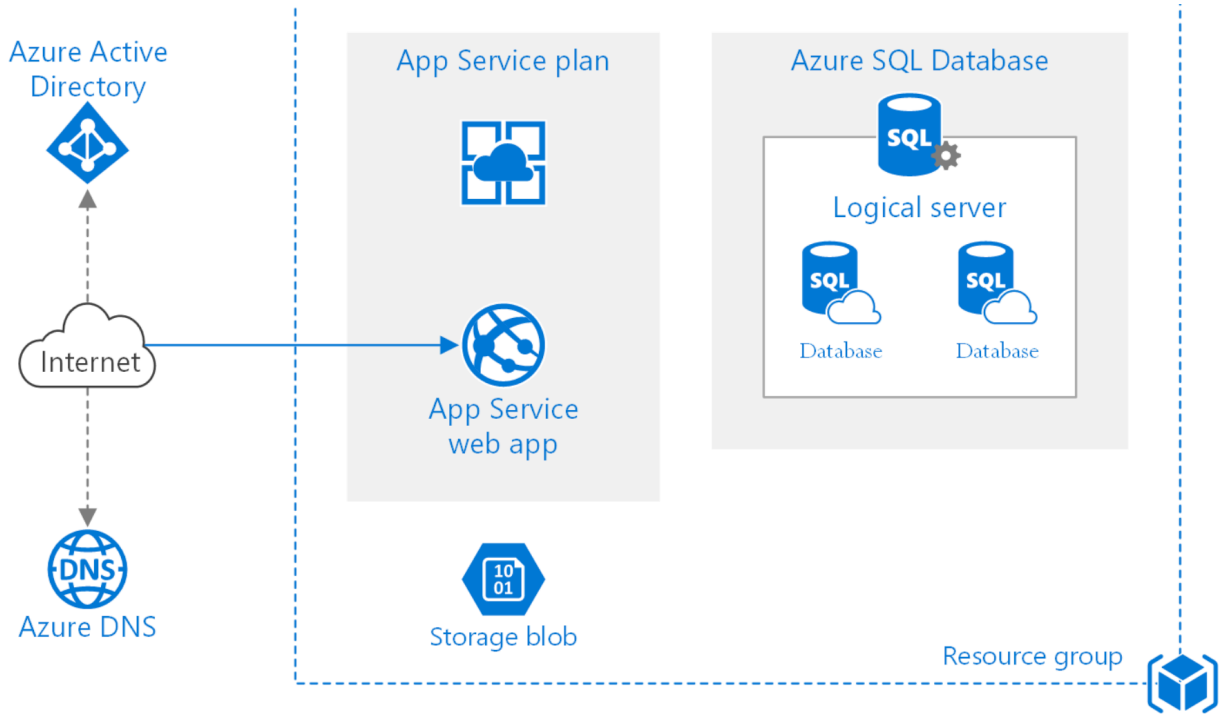
Linux Base:

	FREE **Try for free**	**BASIC** **Dedicated environment for dev/ test**	**STANDARD** **Run production workloads**	**PREMIUM** **Enhanced performance and scale**	**ISOLATED** **High-Performance, Security and Isolation**
Web, mobile, or API apps	10	Unlimited	Unlimited	Unlimited	Unlimited
Disk space	1 GB	10 GB	50 GB	250 GB	1 TB
Maximum instances	–	Up to 3	Up to 10	Up to 30**	Up to 100*
Custom domain	–	Supported	Supported	Supported	Supported
Auto Scale	–	–	Supported	Supported	Supported
VPN hybrid connectivity	–	–	Supported	Supported	Supported
Network Isolation					Supported
Price per hour	Free	Promotional Price **$0.018**	$0.095	$0.111	$0.38

	RAM 1 GM, Storage 1 GB, Shared CPU (60)				

Type-1

Run a basic web application in Azure



The architecture has the following components:

- **Resource group**. A resource group is a logical container for Azure resources.
- **App Service app**. Azure App Service is a fully managed platform for creating and deploying cloud applications.
- **App Service plan**. An App Service plan provides the managed virtual machines (VMs) that host your app. All apps associated with a plan run on the same VM instances.
- **Deployment slots**. A deployment slot lets you stage a deployment and then swap it with the production deployment. That way, you avoid deploying directly into production.
- **IP address**. The App Service app has a public IP address and a domain name. The domain name is a subdomain of azurewebsites.net, such as contoso.azurewebsites.net.
- **Azure DNS**. Azure DNS is a hosting service for DNS domains, providing name resolution using Microsoft Azure infrastructure. By hosting your domains in

Azure, you can manage your DNS records using the same credentials, APIs, tools, and billing as your other Azure services. To use a custom domain name (such as contoso.com) create DNS records that map the custom domain name to the IP address.

- **Azure SQL Database**. SQL Database is a relational database-as-a-service in the cloud. SQL Database shares its code base with the Microsoft SQL Server database engine. Depending on your application requirements, you can also use Azure Database for MySQL or Azure Database for PostgreSQL. These are fully managed database services, based on the open-source MySQL Server and Postgres database engines, respectively.
- **Logical server**. In Azure SQL Database, a logical server hosts your databases. You can create multiple databases per logical server.
- **Azure Storage**. Create an Azure storage account with a blob container to store diagnostic logs.
- **Azure Active Directory (Azure AD)**. Use Azure AD or another identity provider for authentication.

Recommendations

Your requirements might differ from the architecture described here. Use the recommendations in this section as a starting point.

App Service plan
Use the Standard or Premium tiers, because they support scale-out, autoscale, and secure sockets layer (SSL). Each tier supports several *instance sizes* that differ by number of cores and memory. You can change the tier or instance size after you create a plan.

SQL Database
Use the V12 version of SQL Database. SQL Database supports Basic, Standard, and Premium service tiers, with multiple performance levels within each tier measured in Database Transaction Units (DTUs). Perform capacity planning and choose a tier and performance level that meets your requirements.

Region
Provision the App Service plan and the SQL Database in the same region to minimize network latency. Generally, choose the region closest to your users. The resource group also has a region, which specifies where deployment metadata is stored. Put the resource group and its resources in the same region. This can improve availability during deployment.

Scalability considerations

A major benefit of Azure App Service is the ability to scale your application based on load. Here are some considerations to keep in mind when planning to scale your application.

Scaling the App Service app

There are two ways to scale an App Service app:

- *Scale up*, which means changing the instance size. The instance size determines the memory, number of cores, and storage on each VM instance. You can scale up manually by changing the instance size or the plan tier.
- *Scale out*, which means adding instances to handle increased load. Each pricing tier has a maximum number of instances.
  You can scale out manually by changing the instance count, or use autoscaling to have Azure automatically add or remove instances based on a schedule and/or performance metrics. Each scale operation happens quickly—typically within seconds.
  To enable autoscaling, create an autoscale *profile* that defines the minimum and maximum number of instances. Profiles can be scheduled. For example, you might create separate profiles for weekdays and weekends. Optionally, a profile contains rules for when to add or remove instances. (Example: Add two instances if CPU usage is above 70% for 5 minutes.)

Recommendations for scaling a web app:

- As much as possible, avoid scaling up and down, because it may trigger an application restart. Instead, select a tier and size that meet your performance requirements under typical load and then scale out the instances to handle changes in traffic volume.
- Enable autoscaling. If your application has a predictable, regular workload, create profiles to schedule the instance counts ahead of time. If the workload is not predictable, use rule-based autoscaling to react to changes in load as they occur. You can combine both approaches.
- CPU usage is generally a good metric for autoscale rules. However, you should load test your application, identify potential bottlenecks, and base your autoscale rules on that data.
- Autoscale rules include a *cool-down* period, which is the interval to wait after a scale action has completed before starting a new scale action. The cool-down period lets the system stabilize before scaling again. Set a shorter cool-down period for adding instances, and a longer cool-down period for removing instances. For example, set 5 minutes to add an instance, but 60 minutes to remove an instance. It's better to add new instances quickly under heavy load to handle the additional traffic, and then gradually scale back.

Scaling SQL Database

If you need a higher service tier or performance level for SQL Database, you can scale up individual databases with no application downtime.

Availability considerations

At the time of writing, the service level agreement (SLA) for App Service is 99.95% and the SLA for SQL Database is 99.99% for Basic, Standard, and Premium tiers.

Backups
In the event of data loss, SQL Database provides point-in-time restore and geo-restore. These features are available in all tiers and are automatically enabled. You don't need to schedule or manage the backups.

- Use point-in-time restore to recover from human error by returning the database to an earlier point in time.
- Use geo-restore to recover from a service outage by restoring a database from a geo-redundant backup.

App Service provides a backup and restore feature for your application files. However, be aware that the backed-up files include app settings in plain text and these may include secrets, such as connection strings. Avoid using the App Service backup feature to back up your SQL databases because it exports the database to a SQL BACPAC file, consuming DTUs. Instead, use SQL Database point-in-time restore described above.

Deployment

Deployment involves two steps:

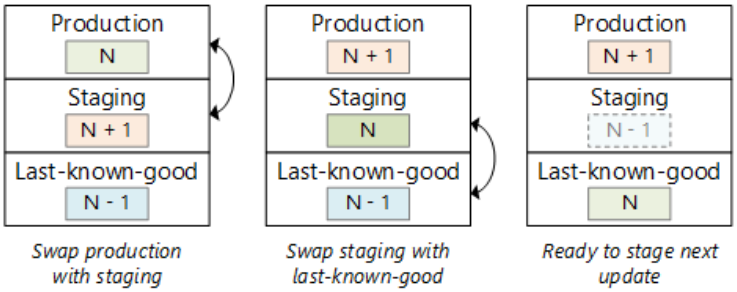
1. Provisioning the Azure resources. We recommend that you use Azure Resource Manager templates for this step. Templates make it easier to automate deployments via PowerShell or the Azure CLI.
2. Deploying the application (code, binaries, and content files). You have several options, including deploying from a local Git repository, using Visual Studio, or continuous deployment from cloud-based source control.

An App Service app always has one deployment slot named production, which represents the live production site. We recommend creating a staging slot for deploying updates. The benefits of using a staging slot include:

- You can verify the deployment succeeded, before swapping it into production.
- Deploying to a staging slot ensures that all instances are warmed up before being swapped into production. Many applications have a significant warmup and cold-start time.

We also recommend creating a third slot to hold the last-known-good deployment. After you swap staging and production, move the previous production deployment

(which is now in staging) into the last-known-good slot. That way, if you discover a problem later, you can quickly revert to the last-known-good version.



If you revert to a previous version, make sure any database schema changes are backward compatible.
Don't use slots on your production deployment for testing because all apps within the same App Service plan share the same VM instances. For example, load tests might degrade the live production site. Instead, create separate App Service plans for production and test. By putting test deployments into a separate plan, you isolate them from the production version.

Authentication

We recommend authenticating through an identity provider (IDP), such as Azure AD, Facebook, Google, or Twitter. Use OAuth 2 or OpenID Connect (OIDC) for the authentication flow. Azure AD provides functionality to manage users and groups, create application roles, integrate your on-premises identities, and consume backend services such as Office 365 and Skype for Business.
Avoid having the application manage user logins and credentials directly, as it creates a potential attack surface. At a minimum, you would need to have email confirmation, password recovery, and multi-factor authentication; validate password strength; and store password hashes securely. The large identity providers handle all of those things for you, and are constantly monitoring and improving their security practices.

Consider using App Service authentication to implement the OAuth/OIDC authentication flow. The benefits of App Service authentication include:

- Easy to configure.
- No code is required for simple authentication scenarios.
- Supports delegated authorization using OAuth access tokens to consume resources on behalf of the user.
- Provides a built-in token cache.

Some limitations of App Service authentication:

- Limited customization options.
- Delegated authorization is restricted to one backend resource per login

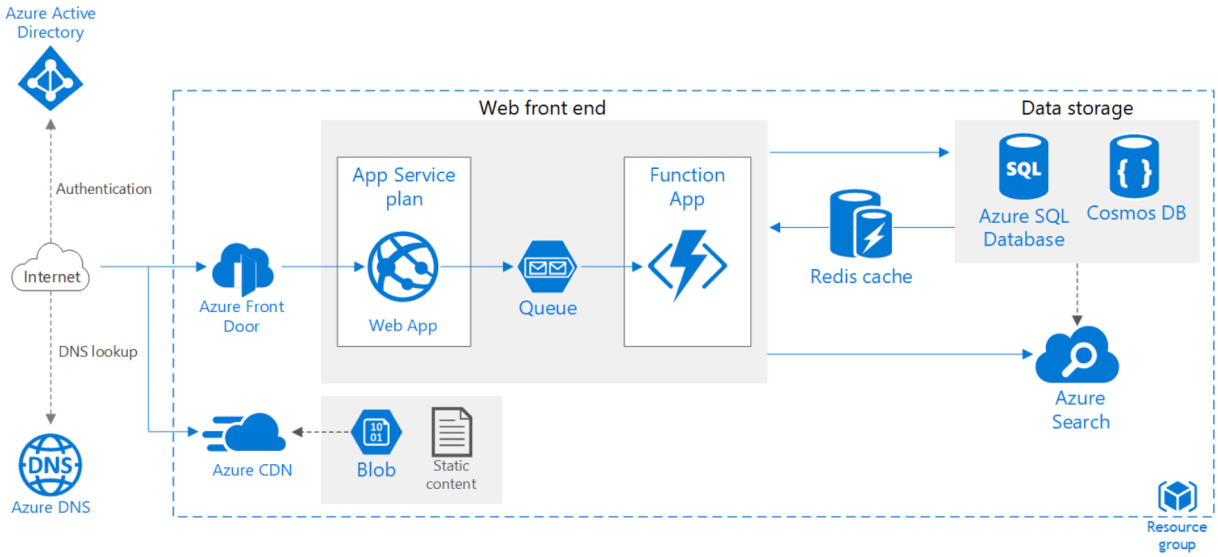
session.

- If you use more than one IDP, there is no built-in mechanism for home realm discovery.
- For multi-tenant scenarios, the application must implement the logic to validate the token issuer.

Type-2

Improve scalability in an Azure web application

This reference architecture shows proven practices for improving scalability and performance in an Azure App Service web application.



This architecture builds on the one shown in Basic web application. It includes the following components:

- **Web app**. A typical modern application might include both a website and one or more RESTful web APIs. A web API might be consumed by browser clients through AJAX, by native client applications, or by server-side applications. For considerations on designing web APIs.
- **Front Door**. Front Door is a layer 7 load balancer. In this architecture, it routes HTTP requests to the web front end. Front Door also provides a web application firewall (WAF) that protects the application from common exploits and vulnerabilities.
- **Function App**. Use Function Apps to run background tasks. Functions are invoked by a trigger, such as a timer event or a message being placed on queue. For long-running stateful tasks, use Durable Functions.

- **Queue**. In the architecture shown here, the application queues background tasks by putting a message onto an Azure Queue storage queue. The message triggers a function app. Alternatively, you can use Service Bus queues. For a comparison.
- **Cache**. Store semi-static data in Azure Cache for Redis.
- **CDN**. Use Azure Content Delivery Network (CDN) to cache publicly available content for lower latency and faster delivery of content.
- **Data storage**. Use Azure SQL Database for relational data. For non-relational data, consider Cosmos DB.
- **Azure Search**. Use Azure Search to add search functionality such as search suggestions, fuzzy search, and language-specific search. Azure Search is typically used in conjunction with another data store, especially if the primary data store requires strict consistency. In this approach, store authoritative data in the other data store and the search index in Azure Search. Azure Search can also be used to consolidate a single search index from multiple data stores.
- **Azure DNS**. Azure DNS is a hosting service for DNS domains, providing name resolution using Microsoft Azure infrastructure. By hosting your domains in Azure, you can manage your DNS records using the same credentials, APIs, tools, and billing as your other Azure services.

Recommendations

Your requirements might differ from the architecture described here. Use the recommendations in this section as a starting point.

App Service apps

We recommend creating the web application and the web API as separate App Service apps. This design lets you run them in separate App Service plans so they can be scaled independently. If you don't need that level of scalability initially, you can deploy the apps into the same plan and move them into separate plans later if necessary.

Cache

You can improve performance and scalability by using Azure Cache for Redis to cache some data. Consider using Azure Cache for Redis for:

- Semi-static transaction data.
- Session state.
- HTML output. This can be useful in applications that render complex HTML output.

CDN

Use Azure CDN to cache static content. The main benefit of a CDN is to reduce latency for users, because content is cached at an edge server that is geographically close to the user. CDN can also reduce load on the application, because that traffic is not being handled by the application.
If your app consists mostly of static pages, consider using CDN to cache the entire app. Otherwise, put static content such as images, CSS, and HTML files, into Azure Storage and use CDN to cache those files.

Storage

Modern applications often process large amounts of data. In order to scale for the cloud, it's important to choose the right storage type. Here are some baseline recommendations.

What you want to store	Example	Recommended storage
Files	Images, documents, PDFs	Azure Blob Storage
Key/Value pairs	User profile data looked up by user ID	Azure Table storage
Short messages intended to trigger further processing	Order requests	Azure Queue storage, Service Bus queue, or Service Bus topic
Non-relational data with a flexible schema requiring basic querying	Product catalog	Document database, such as Azure Cosmos DB, MongoDB, or Apache CouchDB
Relational data requiring richer query support, strict schema, and/or strong consistency	Product inventory	Azure SQL Database

Scalability considerations

A major benefit of Azure App Service is the ability to scale your application based on load. Here are some considerations to keep in mind when planning to scale your application.

App Service app

If your solution includes several App Service apps, consider deploying them to separate App Service plans. This approach enables you to scale them independently because they run on separate instances.

Similarly, consider putting a function app into its own plan so that background tasks don't run on the same instances that handle HTTP requests. If background tasks run intermittently, consider using a consumption plan, which is billed based on the number of executions, rather than hourly.

SQL Database
Increase scalability of a SQL database by *sharding* the database. Sharding refers to partitioning the database horizontally. Sharding allows you to scale out the database horizontally using Elastic Database tools. Potential benefits of sharding include:

- Better transaction throughput.
- Queries can run faster over a subset of the data.

Azure Front Door
Front Door can perform SSL offload and also reduces the total number of TCP connections with the backend web app. This improves scalability because the web app manages a smaller volume of SSL handshakes and TCP connections. These performance gains apply even if you forward the requests to the web app as HTTPS, due to the high level of connection reuse.

Azure Search
Azure Search removes the overhead of performing complex data searches from the primary data store, and it can scale to handle load.

Security considerations

This section lists security considerations that are specific to the Azure services described in this article. It's not a complete list of security best practices for web applications.

Restrict incoming traffic
Configure the application to accept traffic only from Front Door. This ensures that all traffic goes through the WAF before reaching the app.

Cross-Origin Resource Sharing (CORS)
If you create a website and web API as separate apps, the website cannot make client-side AJAX calls to the API unless you enable CORS.
App Services has built-in support for CORS, without needing to write any application code. See Consume an API app from JavaScript using CORS. Add the website to the list of allowed origins for the API.

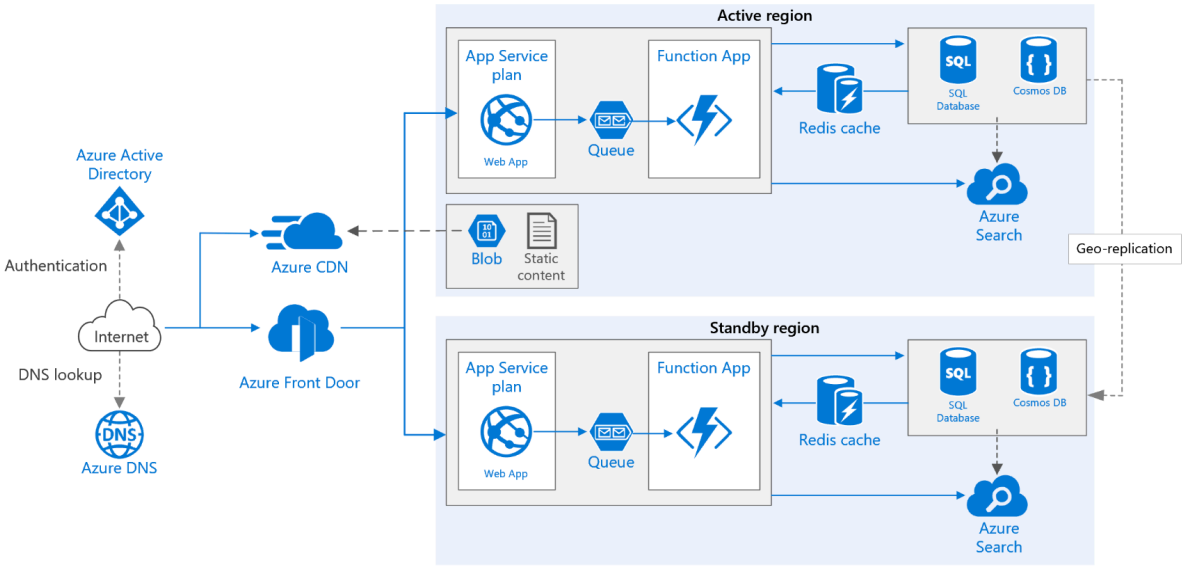
SQL Database encryption
Use Transparent Data Encryption if you need to encrypt data at rest in the

database. This feature performs real-time encryption and decryption of an entire database (including backups and transaction log files) and requires no changes to the application. Encryption does add some latency, so it's a good practice to separate the data that must be secure into its own database and enable encryption only for that database.

Type-3

Run a web application in multiple Azure regions for high availability

This reference architecture shows how to run an Azure App Service application in multiple regions to achieve high availability.



Architecture

This architecture builds on the one shown in Improve scalability in a web application. The main differences are:

- **Primary and secondary regions**. This architecture uses two regions to achieve higher availability. The application is deployed to each region. During normal operations, network traffic is routed to the primary region. If the primary region becomes unavailable, traffic is routed to the secondary region.
- **Front Door**. Front Door routes incoming requests to the primary region. If the application running that region becomes unavailable, Front Door fails over to the secondary region.
- **Geo-replication** of SQL Database and/or Cosmos DB.

A multi-region architecture can provide higher availability than deploying to a single region. If a regional outage affects the primary region, you can use Front Door to fail over to the secondary region. This architecture can also help if an individual subsystem of the application fails.

There are several general approaches to achieving high availability across regions:

- Active/passive with hot standby. Traffic goes to one region, while the other waits on hot standby. Hot standby means the VMs in the secondary region are allocated and running at all times.
- Active/passive with cold standby. Traffic goes to one region, while the other waits on cold standby. Cold standby means the VMs in the secondary region are not allocated until needed for failover. This approach costs less to run, but will generally take longer to come online during a failure.
- Active/active. Both regions are active, and requests are load balanced between them. If one region becomes unavailable, it is taken out of rotation.

This reference architecture focuses on active/passive with hot standby, using Front Door for failover.

Recommendations

Your requirements might differ from the architecture described here. Use the recommendations in this section as a starting point.

Regional pairing

Each Azure region is paired with another region within the same geography. In general, choose regions from the same regional pair (for example, East US 2 and Central US). Benefits of doing so include:

- If there is a broad outage, recovery of at least one region out of every pair is prioritized.
- Planned Azure system updates are rolled out to paired regions sequentially to minimize possible downtime.
- In most cases, regional pairs reside within the same geography to meet data residency requirements.

However, make sure that both regions support all of the Azure services needed for your application. See Services by region. For more information about regional pairs, see Business continuity and disaster recovery (BCDR): Azure Paired Regions.

Resource groups

Consider placing the primary region, secondary region, and Traffic Manager into separate resource groups. This lets you manage the resources deployed to each

region as a single collection.

Front Door configuration

Routing. Front Door supports several routing mechanisms. For the scenario described in this article, use *priority* routing. With this setting, Front Door sends all requests to the primary region unless the endpoint for that region becomes unreachable. At that point, it automatically fails over to the secondary region. Set the backend pool with different priority values, 1 for the active region and 2 or lower for the standby or passive region.

Health probe. Front Door uses an HTTP (or HTTPS) probe to monitor the availability of each back end. The probe gives Front Door a pass/fail test for failing over to the secondary region. It works by sending a request to a specified URL path. If it gets a non-200 response within a timeout period, the probe fails. You can configure the health probe frequency, number of samples required for evaluation, and the number of successful samples required for the backend to be marked as healthy. If Front Door marks the backend as degraded, it fails over to the other backend.

As a best practice, create a health probe path in your application backend that reports the overall health of the application. This health probe should check critical dependencies such as the App Service apps, storage queue, and SQL Database. Otherwise, the probe might report a healthy backend when critical parts of the application are actually failing. On the other hand, don't use the health probe to check lower priority services. For example, if an email service goes down the application can switch to a second provider or just send emails later.

SQL Database

Use Active Geo-Replication to create a readable secondary replica in a different region. You can have up to four readable secondary replicas. Fail over to a secondary database if your primary database fails or needs to be taken offline. Active Geo-Replication can be configured for any database in any elastic database pool.

Cosmos DB

Cosmos DB supports geo-replication across regions with multi-master (multiple write regions). Alternatively, you can designate one region as the writable region and the others as read-only replicas. If there is a regional outage, you can fail over by selecting another region to be the write region. The client SDK automatically sends write requests to the current write region, so you don't need to update the client configuration after a failover.

Storage

For Azure Storage, use read-access geo-redundant storage (RA-GRS). With RA-GRS storage, the data is replicated to a secondary region. You have read-only

access to the data in the secondary region through a separate endpoint. If there is a regional outage or disaster, the Azure Storage team might decide to perform a geo-failover to the secondary region. There is no customer action required for this failover.
For Queue storage, create a backup queue in the secondary region. During failover, the app can use the backup queue until the primary region becomes available again. That way, the application can still process new requests.

Availability considerations - Front Door

Front Door automatically fails over if the primary region becomes unavailable. When Front Door fails over, there is a period of time (usually about 20-60 seconds) when clients cannot reach the application. The duration is affected by the following factors:

- **Frequency of health probes**. The more frequent the health probes are sent, the faster Front Door can detect downtime or the backend coming back healthy.
- **Sample size configuration**. This configuration controls how many samples are required for the health probe to detect that the primary backend has become unreachable. If this value is too low, you could get false positives from intermittent issues.

Front Door is a possible failure point in the system. If the service fails, clients cannot access your application during the downtime. Review the Front Door service level agreement (SLA) and determine whether using Front Door alone meets your business requirements for high availability. If not, consider adding another traffic management solution as a fallback. If the Front Door service fails, change your canonical name (CNAME) records in DNS to point to the other traffic management service. This step must be performed manually, and your application will be unavailable until the DNS changes are propagated.

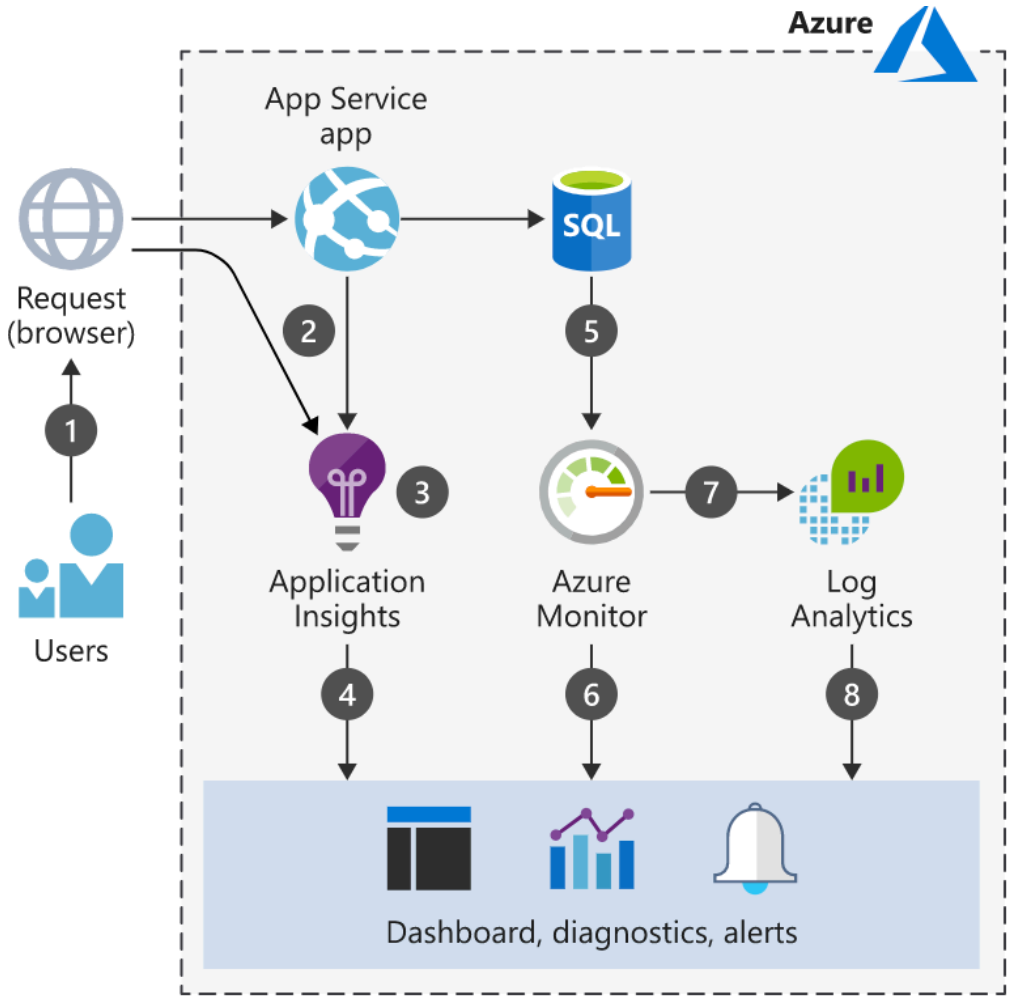
Web application monitoring on Azure

Azure platform as a service (PaaS) offerings manage compute resources for you and affect how you monitor deployments. Azure includes multiple monitoring services, each of which performs a specific role. Together, these services deliver a comprehensive solution for collecting, analyzing, and acting on telemetry from your applications and the Azure resources they consume.
This scenario addresses the monitoring services you can use and describes a dataflow model for use with multiple data sources. When it comes to monitoring, many tools and services work with Azure deployments. In this scenario, we choose readily available services precisely because they are easy to consume. Other monitoring options are discussed later in this article.

Relevant use cases

Other relevant use cases include:

- Instrumenting a web application for monitoring telemetry.
- Collecting front-end and back-end telemetry for an application deployed on Azure.
- Monitoring metrics and quotas associated with services on Azure.



This scenario uses a managed Azure environment to host an application and data tier. The data flows through the scenario as follows:

1. A user interacts with the application.
2. The browser and app service emit telemetry.
3. Application Insights collects and analyzes application health, performance, and usage data.
4. Developers and administrators can review health, performance, and usage information.

5. Azure SQL Database emits telemetry.
6. Azure Monitor collects and analyzes infrastructure metrics and quotas.
7. Log Analytics collects and analyzes logs and metrics.
8. Developers and administrators can review health, performance, and usage information.

Components

- Azure App Service is a PaaS service for building and hosting apps in managed virtual machines. The underlying compute infrastructures on which your apps run is managed for you. App Service provides monitoring of resource usage quotas and app metrics, logging of diagnostic information, and alerts based on metrics. Even better, you can use Application Insights to create availability tests for testing your application from different regions.
- Application Insights is an extensible Application Performance Management (APM) service for developers and supports multiple platforms. It monitors the application, detects application anomalies such as poor performance and failures, and sends telemetry to the Azure portal. Application Insights can also be used for logging, distributed tracing, and custom application metrics.
- Azure Monitor provides base-level infrastructure metrics and logs for most services in Azure. You can interact with the metrics in several ways, including charting them in Azure portal, accessing them through the REST API, or querying them using PowerShell or CLI. Azure Monitor also offers its data directly into Log Analytics and other services, where you can query and combine it with data from other sources on premises or in the cloud.
- Log Analytics helps correlate the usage and performance data collected by Application Insights with configuration and performance data across the Azure resources that support the app. This scenario uses the Azure Log Analytics agent to push SQL Server audit logs into Log Analytics. You can write queries and view data in the Log Analytics blade of the Azure portal.

Considerations

A recommended practice is adding Application Insights to your code during development using the Application Insights SDKs, and customizing per application. These open-source SDKs are available for most application frameworks. To enrich and control the data you collect, incorporate the use of the SDKs both for testing and production deployments into your development process. The main requirement is for the app to have a direct or indirect line of sight to the Applications Insights ingestion endpoint hosted with an Internet-facing address. You can then add telemetry or enrich an existing telemetry collection.
Runtime monitoring is another easy way to get started. The telemetry that is collected must be controlled through configuration files. For example, you can

include runtime methods that enable tools such as Application Insights Status Monitor to deploy the SDKs into the correct folder and add the right configurations to begin monitoring.
Like Application Insights, Log Analytics provides tools for analyzing data across sources, creating complex queries, and sending proactive alerts on specified conditions. You can also view telemetry in the Azure portal. Log Analytics adds value to existing monitoring services such as Azure Monitor and can also monitor on-premises environments.
Both Application Insights and Log Analytics use Azure Log Analytics Query Language. You can also use cross-resource queries to analyze the telemetry gathered by Application Insights and Log Analytics in a single query.
Azure Monitor, Application Insights, and Log Analytics all send alerts. For example, Azure Monitor alerts on platform-level metrics such as CPU utilization, while Application Insights alerts on application-level metrics such as server response time. Azure Monitor alerts on new events in the Azure Activity Log, while Log Analytics can issue alerts about metrics or event data for the services configured to use it. Unified alerts in Azure Monitor is a new, unified alerting experience in Azure that uses a different taxonomy.

Alternatives
This article describes conveniently available monitoring options with popular features, but you have many choices, including the option to create your own logging mechanisms. A recommended practice is to add monitoring services as you build out tiers in a solution. Here are some possible extensions and alternatives:

- Consolidate Azure Monitor and Application Insights metrics in Grafana using the Azure Monitor Data Source For Grafana.
- Data Dog features a connector for Azure Monitor
- Automate monitoring functions using Azure Automation.
- Add communication with ITSM solutions.
- Extend Log Analytics with a management solution.

Scalability and availability
This scenario focuses on PaaS solutions for monitoring in large part because they conveniently handle availability and scalability for you and are backed by service-level agreements (SLAs). For example, App Services provides a guaranteed SLA for its availability.
Application Insights has limits on how many requests can be processed per second. If you exceed the request limit, you may experience message throttling. To prevent throttling, implement filtering or sampling to reduce the data rate
High availability considerations for the app you run, however, are the developer's responsibility. For information about scale, for example, see the Scalability considerations section in the basic web application reference architecture. After

an app is deployed, you can set up tests to monitor its availability using Application Insights.

Security

Sensitive information and compliance requirements affect data collection, retention, and storage. Learn more about how Application Insights and Log Analytics handle telemetry.

The following security considerations may also apply:

- Develop a plan to handle personal information if developers are allowed to collect their own data or enrich existing telemetry.
- Consider data retention. For example, Application Insights retains telemetry data for 90 days. Archive data you want access to for longer periods using Microsoft Power BI, Continuous Export, or the REST API. Storage rates apply.
- Limit access to Azure resources to control access to data and who can view telemetry from a specific application. To help lock down access to monitoring telemetry.
- Consider whether to control read/write access in application code to prevent users from adding version or tag markers that limit data ingestion from the application. With Application Insights, there is no control over individual data items once they are sent to a resource, so if a user has access to any data, they have access to all data in an individual resource.
- Add governance mechanisms to enforce policy or cost controls over Azure resources if needed. For example, use Log Analytics for security-related monitoring such as policies and role-based access control, or use Azure Policy to create, assign and, manage policy definitions.
- To monitor potential security issues and get a central view of the security state of your Azure resources, consider using Azure Security Center.

Pricing

Monitoring charges can add up quickly, so consider pricing up front, understand what you are monitoring, and check the associated fees for each service. Azure Monitor provides basic metrics at no cost, while monitoring costs for Application Insights and Log Analytics are based on the amount of data ingested and the number of tests you run.

To help you get started, use the pricing calculator to estimate costs. To see how the pricing would change for your particular use case, change the various options to match your expected deployment.

Telemetry from Application Insights is sent to the Azure portal during debugging and after you have published your app. For testing purposes and to avoid charges, a limited volume of telemetry is instrumented. To add more indicators, you can raise the telemetry limit. For more granular control, see Sampling in Application Insights.

After deployment, you can watch a Live Metrics Stream of performance indicators.

This data is not stored — you are viewing real-time metrics — but the telemetry can be collected and analyzed later. There is no charge for Live Stream data. Log Analytics is billed per gigabyte (GB) of data ingested into the service. The first 5 GB of data ingested to the Azure Log Analytics service every month is offered free, and the data is retained at no charge for first 31 days in your Log Analytics workspace.

	FREE	SHARED	BASIC	STANDARD	PREMIUM	ISOLATED *	APP SERVICE LINUX
– Limits **							
Apps	10	100	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited
Disk space	1 GB	1 GB	10 GB	50 GB	250 GB	1 TB	
Max instances			Up to 3	Up to 10	Up to 30	Up to 100	
SLA			99.95%	99.95%	99.95%	99.95%	
Functions on App Service Plans *			✓	✓	✓	✓	
– App Deployment							
Continuous Deployment *	✓	✓	✓	✓	✓	✓ 3	✓
Deployment Slots				✓	✓	✓	✓
Docker (Containers)							✓ 1
– Development Tools							
Clone App					✓	✓	
Kudu	✓	✓	✓	✓	✓	✓ 3	✓ 2
Site Extensions	✓	✓	✓	✓	✓	✓ 3	
Testing in Production				✓	✓	✓	✓

– Diagnostics & Monitoring							
Log Stream	✓	✓	✓	✓	✓	✓[3]	
Process Explorer	✓	✓	✓	✓	✓	✓[3]	
– Networking							
Hybrid Connections *	✓	✓	✓	✓	✓	✓	
VNET Integration *				✓	✓	✓	

– Scale							
Auto-scale				✓	✓	✓	✓
Integrated Load Balancer		✓	✓	✓	✓	✓	✓
Traffic Manager [3]				✓	✓	✓	
– Settings							
64-bit			✓	✓	✓	✓	✓
App Service Advisor *			✓	✓	✓	✓	✓
Always On			✓	✓	✓	✓	
Authentication & Authorization	✓	✓	✓	✓	✓	✓	
Backup/Restore				✓	✓	✓	✓
Custom Domains		✓	✓	✓	✓	✓	✓
FTP/FTPS	✓	✓	✓	✓	✓	✓	✓
Local Cache				✓	✓	✓	

https://azure.microsoft.com/en-us/pricing/details/app-service/plans/