Contents

[Cosmos DB ( Document DB/Mongo DB) 1](#_Toc50952461)

[Create an account 2](#_Toc50952462)

[Cosmos DB API 3](#_Toc50952463)

[Core (SQL) API 3](#_Toc50952464)

[MongoDB API 3](#_Toc50952465)

[Cassandra API 3](#_Toc50952466)

[Azure Table API 4](#_Toc50952467)

[Gremlin (graph) API 4](#_Toc50952468)

[Replicate Data Globally 4](#_Toc50952469)

[Consistency levels 5](#_Toc50952470)

[Strong 5](#_Toc50952471)

[Bounded staleness 5](#_Toc50952472)

[Session 5](#_Toc50952473)

[Consistent prefix 6](#_Toc50952474)

[Eventual 6](#_Toc50952475)

[IP firewall in Azure Cosmos DB 6](#_Toc50952476)

[MS Learn 6](#_Toc50952477)

[Azure SQL 6](#_Toc50952478)

[Azure SQL Database 6](#_Toc50952479)

[Deployment models 7](#_Toc50952480)

[Purchasing Model: 7](#_Toc50952481)

[Service tiers 8](#_Toc50952482)

[Azure SQL Managed Instance 8](#_Toc50952483)

[SQL Server on Azure VM 8](#_Toc50952484)

[Data encryption 9](#_Toc50952485)

# Cosmos DB ( Document DB/Mongo DB)

<https://www.youtube.com/watch?v=R_Fi59j6BMo>

## Create an account

<https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-manage-database-account>

On the **Create Azure Cosmos DB Account** page, enter the basic settings for the new Azure Cosmos account.

| **Setting** | **Value** | **Description** |
| --- | --- | --- |
| Subscription | Subscription name | Select the Azure subscription that you want to use for this Azure Cosmos account. |
| Resource Group | Resource group name | Select a resource group, or select **Create new**, then enter a unique name for the new resource group. |
| Account Name | A unique name | Enter a name to identify your Azure Cosmos account. Because documents.azure.com is appended to the name that you provide to create your URI, use a unique name.  The name can only contain lowercase letters, numbers, and the hyphen (-) character. It must be between 3-44 characters in length. |
| API | The type of account to create | Select **Core (SQL)** to create a document database and query by using SQL syntax.  The API determines the type of account to create. Azure Cosmos DB provides five APIs: Core (SQL) and MongoDB for document data, Gremlin for graph data, Azure Table, and Cassandra. Currently, you must create a separate account for each API. |
| Capacity mode | Provisioned throughput or Serverless | Select **Provisioned throughput** to create an account in [provisioned throughput](https://docs.microsoft.com/en-us/azure/cosmos-db/set-throughput) mode. Select **Serverless** to create an account in [serverless](https://docs.microsoft.com/en-us/azure/cosmos-db/serverless) mode.  **Note**: Serverless is currently available for Core (SQL) API accounts only. |
| Apply Free Tier Discount | Apply or Do not apply | With Azure Cosmos DB free tier, you will get the first 400 RU/s and 5 GB of storage for free in an account. Learn more about [free tier](https://azure.microsoft.com/pricing/details/cosmos-db/). |
| Location | The region closest to your users | Select a geographic location to host your Azure Cosmos DB account. Use the location that is closest to your users to give them the fastest access to the data. |
| Account Type | Production or Non-Production | Select **Production** if the account will be used for a production workload. Select **Non-Production** if the account will be used for non-production, e.g. development, testing, QA, or staging. This is an Azure resource tag setting that tunes the Portal experience but does not affect the underlying Azure Cosmos DB account. You can change this value anytime. |

## Cosmos DB API

<https://docs.microsoft.com/en-us/learn/modules/choose-api-for-cosmos-db/>

### Core (SQL) API

Core (SQL) is the default API for Azure Cosmos DB, which provides you with a view of your data that resembles a traditional NoSQL document store. You can query the hierarchical JSON documents with a SQL-like language. Core (SQL) uses JavaScript's type system, expression evaluation, and function invocation.

For developers who have experience with the SQL query language, Core (SQL) provides several familiar SQL statements and clauses; for example:

SELECT

FROM

WHERE

BETWEEN

COUNT

SUM

MIN

MAX

ORDER BY

### MongoDB API

Azure Cosmos DB's API for MongoDB supports the MongoDB wire protocol. This API allows existing MongoDB client SDKs, drivers, and tools to interact with the data transparently, as if they are running against an actual MongoDB database. The data is stored in document format, which is the same as using Core (SQL). Azure Cosmos DB's API for MongoDB is currently compatible with 3.2 version of the MongoDB wire protocol.

### Cassandra API

Azure Cosmos DB's support for the Cassandra API makes it possible to query data by using the Cassandra Query Language (CQL), and your data will appear to be a partitioned row store. Just like the MongoDB API, any clients or tools should be able to connect transparently to Azure Cosmos DB; only your connection settings should need to be updated. Cosmos DB's Cassandra API currently supports version 4 of the CQL wire protocol.

### Azure Table API

Azure Cosmos DB's Azure Table API provides support for applications that are written for Azure Table Storage that need premium capabilities like global distribution, high availability, scalable throughput. The original Table API only allows for indexing on the Partition and Row keys; there are no secondary indexes. Storing table data in Cosmos DB automatically indexes all the properties, and requires no index management.

Querying is accomplished by using OData and LINQ queries in code, and the original REST API for GET operations

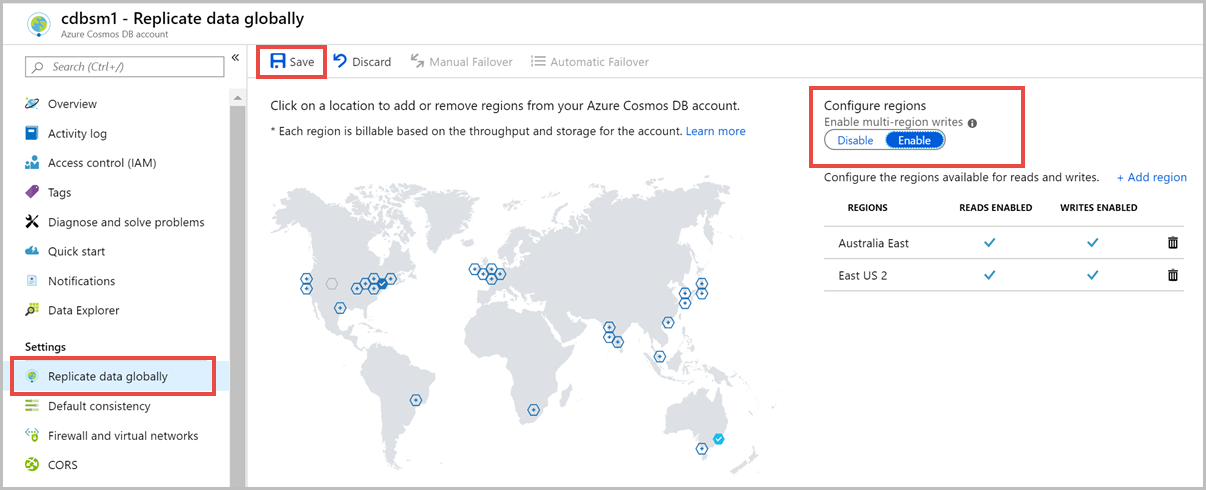
### Gremlin (graph) API

Choosing Gremlin as the API provides a graph-based view over the data. Remember that at the lowest level, all data in any Azure Cosmos DB is stored in an ARS format. A graph-based view on the database means data is either a vertex (which is an individual item in the database), or an edge (which is a relationship between items in the database).

You typically use a traversal language to query a graph database, and Azure Cosmos DB supports Apache Tinkerpop's Gremlin language.

## Replicate Data Globally

Open the **Replicate Data Globally** tab and select **Enable** to enable multi-region writes. After you enable multi-region writes, all the read regions that you currently have on the account will become read and write regions.



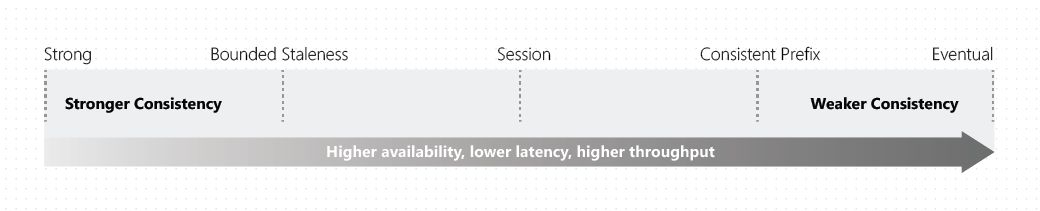
One can also select **Automatic Failover** from the page’s tab menu option.

## Consistency levels

<https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels>

Most commercially available distributed databases ask developers to choose between the two extreme consistency models: **strong consistency** and **eventual consistency**.

Each level provides [availability and performance tradeoffs](https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels-tradeoffs) and are backed by SLAs.



### Strong

The reads are guaranteed to return the most recent committed version of an item. A client never sees an uncommitted or partial write. Users are always guaranteed to read the latest committed write.

### Bounded staleness

The reads are guaranteed to honor the consistent-prefix guarantee. when you choose bounded staleness, the "staleness" can be configured in two ways:

* The number of versions (K) of the item
* The time interval (T) by which the reads might lag behind the writes

Bounded staleness is frequently chosen by globally distributed applications that expect low write latencies but require total global order guarantee. Bounded staleness is great for applications featuring group collaboration and sharing, stock ticker, publish-subscribe/queueing etc.

### Session

Within a single client session reads are guaranteed to honor the consistent-prefix, monotonic reads, monotonic writes, read-your-writes, and write-follows-reads guarantees. This assumes a single "writer" session or sharing the session token for multiple writers.

### Consistent prefix

Updates that are returned contain some prefix of all the updates, with no gaps. Consistent prefix consistency level guarantees that reads never see out-of-order writes.

If writes were performed in the order A, B, C, then a client sees either A, A,B, or A,B,C, but never out-of-order permutations like A,C or B,A,C. Consistent Prefix provides write latencies, availability, and read throughput comparable to that of eventual consistency, but also provides the order guarantees that suit the needs of scenarios where order is important.

### Eventual

There's no ordering guarantee for reads. In the absence of any further writes, the replicas eventually converge.

Eventual consistency is the weakest form of consistency because a client may read the values that are older than the ones it had read before. Eventual consistency is ideal where the application does not require any ordering guarantees. Examples include count of Retweets, Likes, or non-threaded comments.

## IP firewall in Azure Cosmos DB

By default, your Azure Cosmos account is accessible from internet, as long as the request is accompanied by a valid authorization token.

If IP firewall and VNET Access Control List (ACLs) are not set up, the Azure Cosmos account can be accessed with the authorization token. After the IP firewall or VNET ACLs or both are set up on the Azure Cosmos account, only requests originating from the sources you have specified (and with the authorization token) get valid responses.

## MS Learn

<https://docs.microsoft.com/en-us/learn/modules/choose-api-for-cosmos-db/>

<https://github.com/Azure-Samples/azure-cosmos-db-sample-data>

# Azure SQL

<https://docs.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview>

<https://www.youtube.com/watch?v=BgvEOkcR0Wk>

## Azure SQL Database

<https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview>

Azure SQL Database is a fully managed platform as a service (PaaS) database engine that handles most of the database management functions such as upgrading, patching, backups, and monitoring without user involvement. Azure SQL Database is always running on the latest stable version of the SQL Server database engine and patched OS with 99.99% availability.

SQL Database enables you to easily define and scale performance within two different purchasing models: a [vCore-based purchasing model](https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tiers-vcore) and a [DTU-based purchasing model](https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tiers-dtu).

### Deployment models

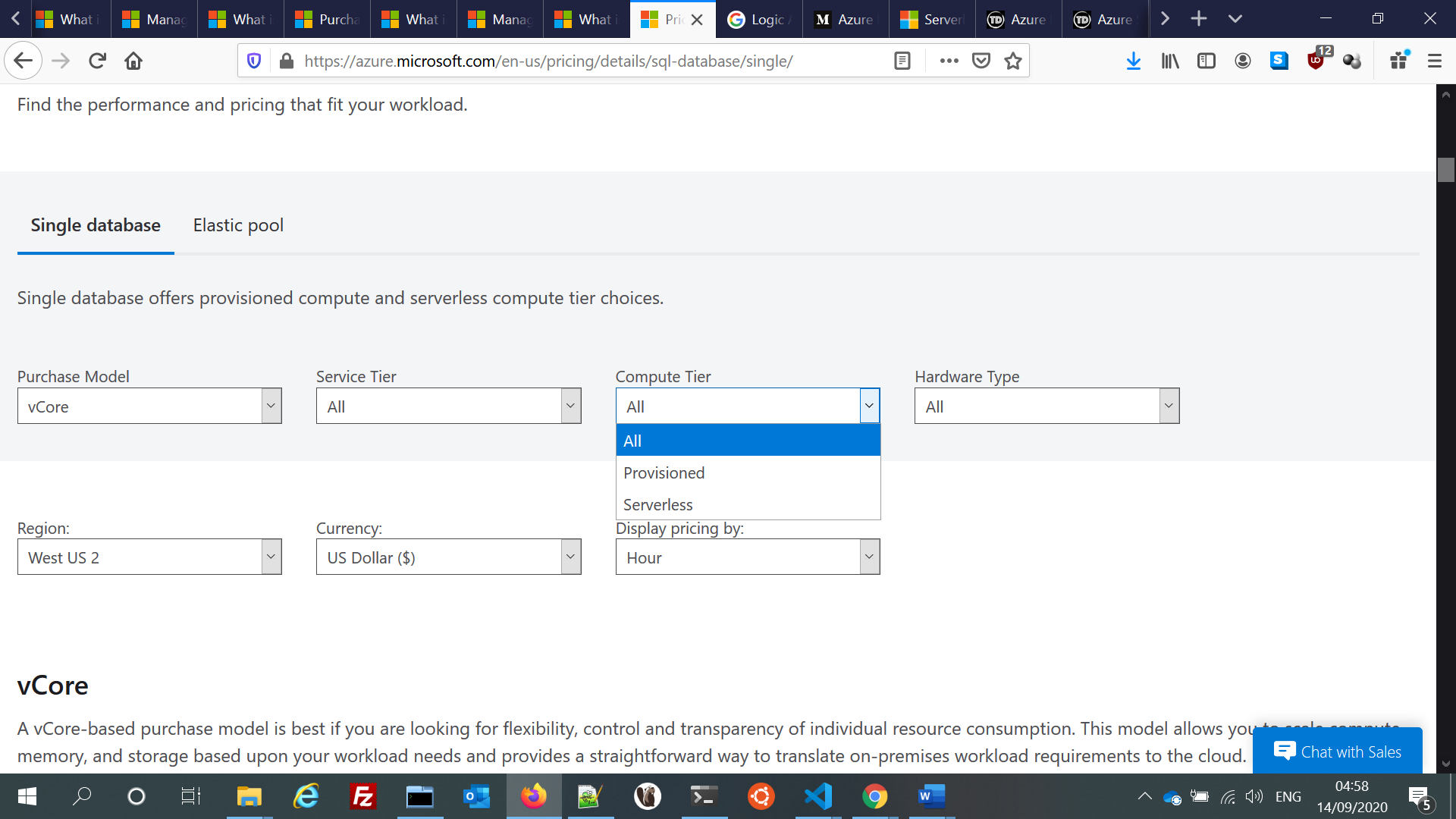
Azure SQL Database provides the following deployment options for a database:

* [Single database](https://docs.microsoft.com/en-us/azure/azure-sql/database/single-database-overview) represents a fully managed, isolated database. You might use this option if you have modern cloud applications and microservices that need a single reliable data source. A single database is similar to a [contained database](https://docs.microsoft.com/en-us/sql/relational-databases/databases/contained-databases?toc=/azure/sql-database/toc.json) in the [SQL Server database engine](https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?toc=/azure/sql-database/toc.json).
* [Elastic pool](https://docs.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview) is a collection of [single databases](https://docs.microsoft.com/en-us/azure/azure-sql/database/single-database-overview) with a shared set of resources, such as CPU or memory. Single databases can be moved into and out of an elastic pool.

### Purchasing Model:

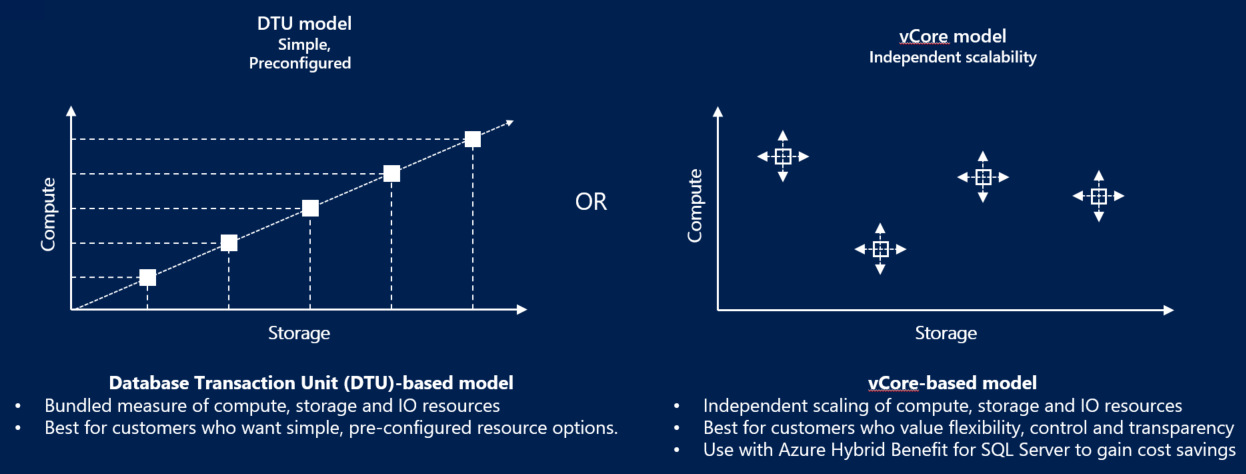
<https://azure.microsoft.com/en-us/pricing/details/sql-database/single/>

* [Virtual core (vCore)-based purchasing model](https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tiers-vcore) (**recommended**). This purchasing model provides a choice between a provisioned compute tier and a serverless compute tier. With the provisioned compute tier, you choose the exact amount of compute resources that are always provisioned for your workload. With the serverless compute tier, you specify the autoscaling of the compute resources over a configurable compute range. With this compute tier, you can also automatically pause and resume the database based on workload activity. The vCore unit price per unit of time is lower in the provisioned compute tier than it is in the serverless compute tier.
* [Database transaction unit (DTU)-based purchasing model](https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tiers-dtu). This purchasing model provides bundled compute and storage packages balanced for common workloads.



Another demarcation comes for Compute Tier where Serverless can also be an option

The [serverless model](https://docs.microsoft.com/en-us/azure/azure-sql/database/serverless-tier-overview) automatically scales compute based on workload demand, and bills for the amount of compute used per second. The serverless compute tier also automatically pauses databases during inactive periods when only storage is billed, and automatically resumes databases when activity returns.



### Service tiers

Azure SQL Database offers three service tiers that are designed for different types of applications:

* [General Purpose/Standard](https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tier-general-purpose) service tier designed for common workloads. It offers budget-oriented balanced compute and storage options.
* [Business Critical/Premium](https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tier-business-critical) service tier designed for OLTP applications with high transaction rate and lowest-latency I/O. It offers the highest resilience to failures by using several isolated replicas.
* [Hyperscale](https://docs.microsoft.com/en-us/azure/azure-sql/database/service-tier-hyperscale) service tier designed for very large OLTP database and the ability to autoscale storage and scale compute fluidly.

## Azure SQL Managed Instance

<https://docs.microsoft.com/en-us/azure/azure-sql/managed-instance/sql-managed-instance-paas-overview>

SQL Managed Instance is a collection of system and user databases with a shared set of resources that is lift-and-shift ready.

* Best for new applications or existing on-premises applications that want to use the latest stable SQL Server features and that are migrated to the cloud with minimal changes.
* SQL Managed Instance supports database migration from on-premises with minimal to no database change. This option provides all of the PaaS benefits of Azure SQL Database but adds capabilities that were previously only available in SQL Server VMs.

## SQL Server on Azure VM

<https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-server-on-azure-vm-iaas-what-is-overview>

[SQL Server on Azure VM](https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-server-on-azure-vm-iaas-what-is-overview) falls into the industry category Infrastructure-as-a-Service (IaaS) and allows you to run SQL Server inside a fully managed virtual machine (VM) in Azure.

Best for migrations and applications requiring OS-level access. SQL virtual machines in Azure are lift-and-shift ready for existing applications that require fast migration to the cloud with minimal changes or no changes.

Rapid development and test scenarios when you do not want to buy on-premises non-production SQL Server hardware.

| **Azure SQL Database** | **Azure SQL Managed Instance** | **SQL Server on Azure VM** |
| --- | --- | --- |
| Supports most on-premises database-level capabilities. The most commonly used SQL Server features are available. 99.995% availability guaranteed. Built-in backups, patching, recovery. Latest stable Database Engine version. Ability to assign necessary resources (CPU/storage) to individual databases. Built-in advanced intelligence and security. Online change of resources (CPU/storage). | Supports almost all on-premises instance-level and database-level capabilities. High compatibility with SQL Server. 99.99% availability guaranteed. Built-in backups, patching, recovery. Latest stable Database Engine version. Easy migration from SQL Server. Private IP address within Azure Virtual Network. Built-in advanced intelligence and security. Online change of resources (CPU/storage). | You have full control over the SQL Server engine. Supports all on-premises capabilities. Up to 99.99% availability. Full parity with the matching version of on-premises SQL Server. Fixed, well-known Database Engine version. Easy migration from SQL Server. Private IP address within Azure Virtual Network. You have the ability to deploy application or services on the host where SQL Server is placed. |
| Migration from SQL Server might be challenging. Some SQL Server features are not available. No guaranteed exact maintenance time (but nearly transparent). Compatibility with the SQL Server version can be achieved only using database compatibility levels. Private IP address support with [Azure Private Link](https://docs.microsoft.com/en-us/azure/azure-sql/database/private-endpoint-overview). | There is still some minimal number of SQL Server features that are not available. No guaranteed exact maintenance time (but nearly transparent). Compatibility with the SQL Server version can be achieved only using database compatibility levels. | You need to manage your backups and patches. You need to implement your own High-Availability solution. There is a downtime while changing the resources(CPU/storage) |
| Databases of up to 100 TB. | Up to 8 TB. | SQL Server instances with up to 256 TB of storage. The instance can support as many databases as needed. |
| On-premises application can access data in Azure SQL Database. | [Native virtual network implementation](https://docs.microsoft.com/en-us/azure/azure-sql/managed-instance/vnet-existing-add-subnet) and connectivity to your on-premises environment using Azure Express Route or VPN Gateway. | With SQL virtual machines, you can have applications that run partly in the cloud and partly on-premises. For example, you can extend your on-premises network and Active Directory Domain to the cloud via [Azure Virtual Network](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview). For more information on hybrid cloud solutions, see [Extending on-premises data solutions to the cloud](https://docs.microsoft.com/en-us/azure/architecture/data-guide/scenarios/hybrid-on-premises-and-cloud). |

### Data encryption

SQL Database helps secure your data by providing encryption. For data in motion, it uses [transport layer security](https://support.microsoft.com/kb/3135244). For data at rest, it uses [transparent data encryption](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption-azure-sql). For data in use, it uses [Always Encrypted](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine).

<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-always-encrypted>

Always encrypted is correct, as this ensures encryption happens during transport and that the encryption keys are never revealed to the database engine. Transparent Data Encryption will not suffice as this is database-level encryption, which means data is protected at rest.

<https://docs.microsoft.com/en-us/azure/sql-database/transparent-data-encryption-azure-sql?tabs=azure-portal>