Scaling out with Azure SQL Database

Article • 06/05/2023

Applies to: ✓ Azure SQL Database

You can easily scale out databases in Azure SQL Database using the **Elastic Database** tools. These tools and features let you use the database resources of **Azure SQL Database** to create solutions for transactional workloads, and especially Software as a Service (SaaS) applications. Elastic Database features are composed of the:

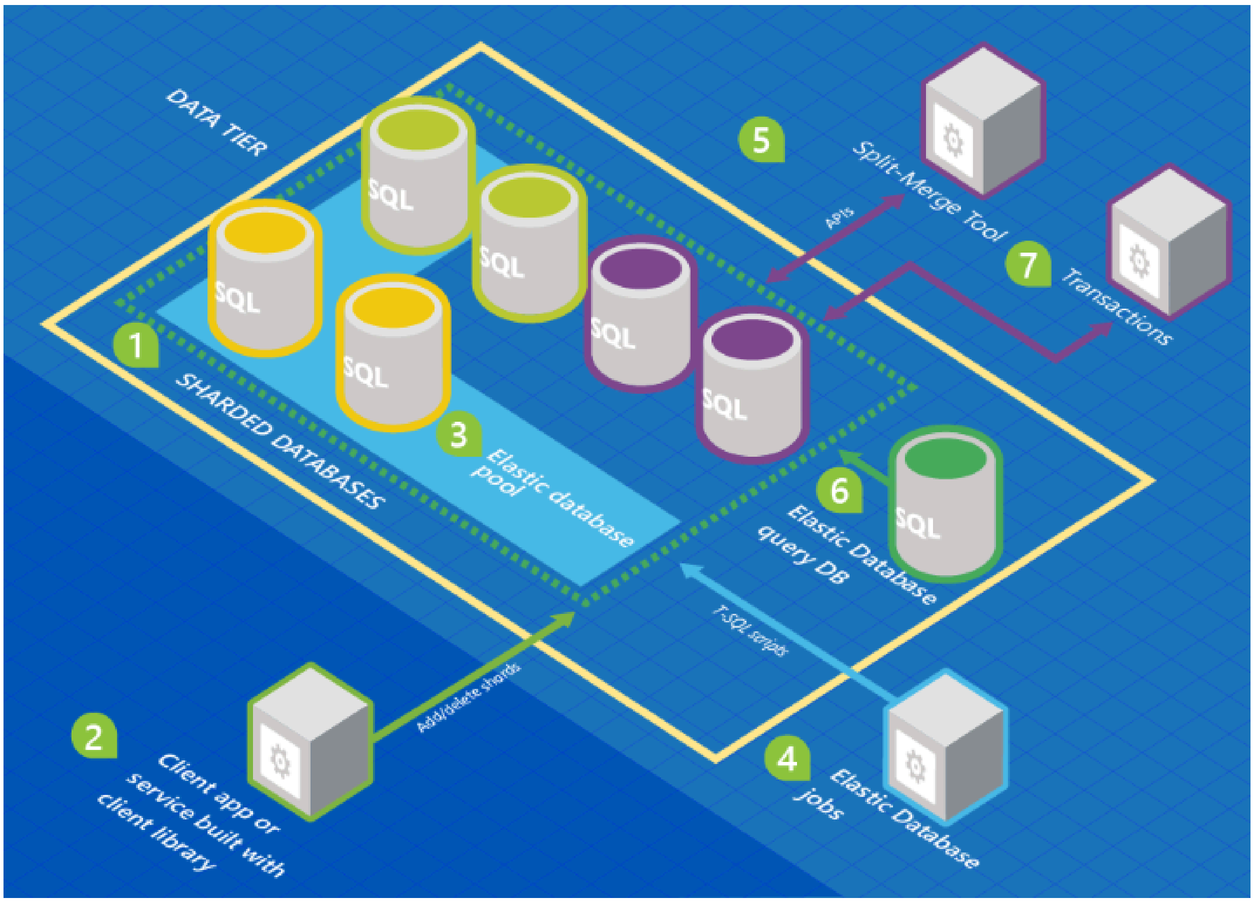
- Elastic Database client library: The client library is a feature that allows you to create and maintain sharded databases. See Get started with Elastic Database tools.
- Elastic Database split-merge tool: moves data between sharded databases. This tool is useful for moving data from a multi-tenant database to a single-tenant database (or vice-versa). See Elastic database Split-Merge tool tutorial.
- Elastic jobs: Use jobs to manage large numbers of databases in Azure SQL Database. Easily perform administrative operations such as schema changes, credentials management, reference data updates, performance data collection, or tenant (customer) telemetry collection using jobs.
- Elastic Database query (preview): Enables you to run a Transact-SQL query that spans multiple databases. This enables connection to reporting tools such as Excel, Power BI, Tableau, etc.
- Elastic transactions: This feature allows you to run transactions that span several databases. Elastic database transactions are available for .NET applications using ADO .NET and integrate with the familiar programming experience using the System.Transaction classes.

The following graphic shows an architecture that includes the **Elastic Database features** in relation to a collection of databases.

In this graphic, colors of the database represent schemas. Databases with the same color share the same schema.

1. A set of **SQL databases** is hosted on Azure using sharding architecture.
2. The **Elastic Database client library** is used to manage a shard set.
3. A subset of the databases is put into an **elastic pool**. (See What is a pool?).
4. An **Elastic Database job** runs scheduled or ad hoc T-SQL scripts against all databases.
5. The **split-merge tool** is used to move data from one shard to another.

6. The **Elastic Database query** allows you to write a query that spans all databases in the shard set.
7. **Elastic transactions** allow you to run transactions that span several databases.



Why use the tools?

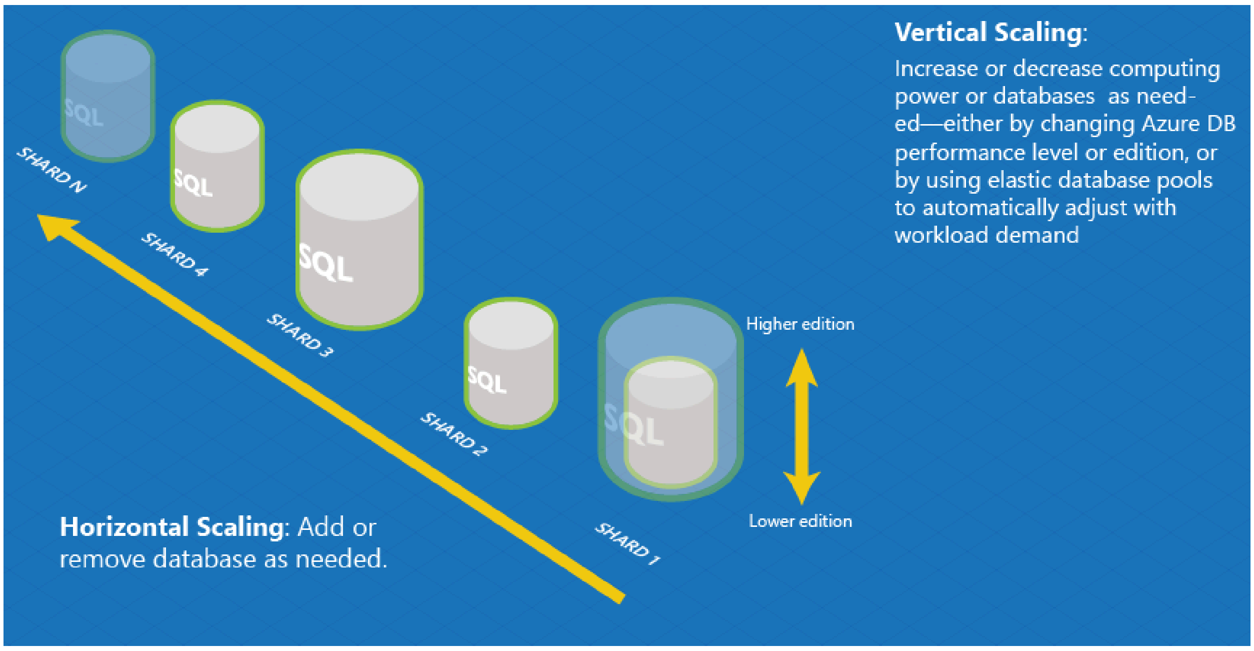
Achieving elasticity and scale for cloud applications has been straightforward for VMs and blob storage - simply add or subtract units, or increase power. But it has remained a challenge for stateful data processing in relational databases. Challenges emerged in these scenarios:

- Growing and shrinking capacity for the relational database part of your workload.
- Managing hotspots that may arise affecting a specific subset of data - such as a busy end-customer (tenant).

Traditionally, scenarios like these have been addressed by investing in larger-scale servers to support the application. However, this option is limited in the cloud where all processing happens on predefined commodity hardware. Instead, distributing data and processing across many identically structured databases (a scale-out pattern known as "sharding") provides an alternative to traditional scale-up approaches both in terms of cost and elasticity.

Horizontal and vertical scaling

The following figure shows the horizontal and vertical dimensions of scaling, which are the basic ways the elastic databases can be scaled.


Vertical Scaling:
Increase or decrease computing power or databases as needed—either by changing Azure DB performance level or edition, or by using elastic database pools to automatically adjust with workload demand
SQL
SHARD N
SHARD 4
SHARD 3
SHARD 2
SHARD 1
Higher edition
Lower edition
Horizontal Scaling: Add or remove database as needed.

Horizontal scaling refers to adding or removing databases in order to adjust capacity or overall performance, also called "scaling out". Sharding, in which data is partitioned across a collection of identically structured databases, is a common way to implement horizontal scaling.

Vertical scaling refers to increasing or decreasing the compute size of an individual database, also known as "scaling up."

Most cloud-scale database applications use a combination of these two strategies. For example, a Software as a Service application may use horizontal scaling to provision new end-customers and vertical scaling to allow each end-customer's database to grow or shrink resources as needed by the workload.

- Horizontal scaling is managed using the Elastic Database client library.
- Vertical scaling is accomplished using Azure PowerShell cmdlets to change the service tier, or by placing databases in an elastic pool.

Sharding

Sharding is a technique to distribute large amounts of identically structured data across a number of independent databases. It is especially popular with cloud developers creating Software as a Service (SAAS) offerings for end customers or businesses. These end customers are often referred to as "tenants". Sharding may be required for any number of reasons:

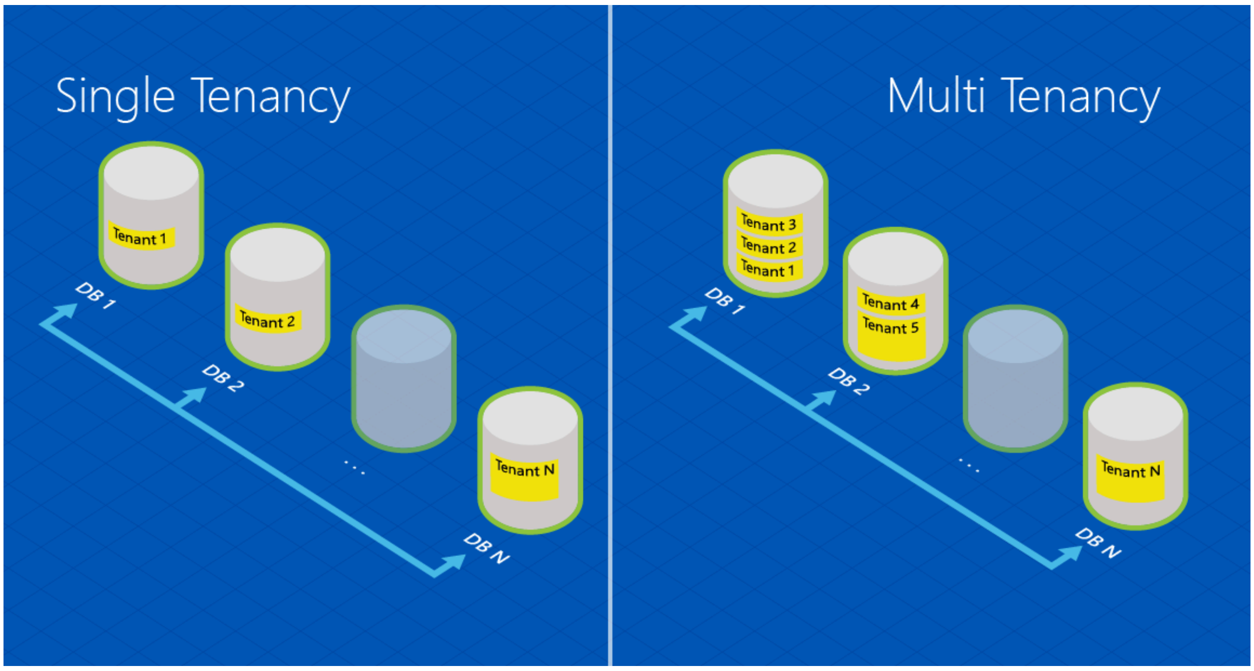
- The total amount of data is too large to fit within the constraints of an individual database
- The transaction throughput of the overall workload exceeds the capabilities of an individual database
- Tenants may require physical isolation from each other, so separate databases are needed for each tenant
- Different sections of a database may need to reside in different geographies for compliance, performance, or geopolitical reasons.

In other scenarios, such as ingestion of data from distributed devices, sharding can be used to fill a set of databases that are organized temporally. For example, a separate database can be dedicated to each day or week. In that case, the sharding key can be an integer representing the date (present in all rows of the sharded tables) and queries retrieving information for a date range must be routed by the application to the subset of databases covering the range in question.

Sharding works best when every transaction in an application can be restricted to a single value of a sharding key. That ensures that all transactions are local to a specific database.

Multi-tenant and single-tenant

Some applications use the simplest approach of creating a separate database for each tenant. This approach is the **single tenant sharding pattern** that provides isolation, backup/restore ability, and resource scaling at the granularity of the tenant. With single tenant sharding, each database is associated with a specific tenant ID value (or customer key value), but that key need not always be present in the data itself. It is the application's responsibility to route each request to the appropriate database - and the client library can simplify this.



Others scenarios pack multiple tenants together into databases, rather than isolating them into separate databases. This pattern is a typical **multi-tenant sharding pattern** - and it may be driven by the fact that an application manages large numbers of small tenants. In multi-tenant sharding, the rows in the database tables are all designed to carry a key identifying the tenant ID or sharding key. Again, the application tier is responsible for routing a tenant's request to the appropriate database, and this can be supported by the elastic database client library. In addition, row-level security can be used to filter which rows each tenant can access - for details, see Multi-tenant applications with elastic database tools and row-level security. Redistributing data among databases may be needed with the multi-tenant sharding pattern, and is facilitated by the elastic database split-merge tool. To learn more about design patterns for SaaS applications using elastic pools, see Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database.

Move data from multiple to single-tenancy databases

When creating a SaaS application, it is typical to offer prospective customers a trial version of the software. In this case, it is cost-effective to use a multi-tenant database for the data. However, when a prospect becomes a customer, a single-tenant database is better since it provides better performance. If the customer had created data during the trial period, use the split-merge tool to move the data from the multi-tenant to the new single-tenant database.

ⓘ Note

Restoring from multi-tenant databases to a single tenant is not possible.

Next steps

For a sample app that demonstrates the client library, see Get started with Elastic Database tools.

To convert existing databases to use the tools, see Migrate existing databases to scale out.

To see the specifics of the elastic pool, see Price and performance considerations for an elastic pool, or create a new pool with elastic pools.

Additional resources

Not using elastic database tools yet? Check out our Getting Started Guide. For questions, contact us on the Microsoft Q&A question page for SQL Database and for feature requests, add new ideas or vote for existing ideas in the SQL Database feedback forum.