


# Distributed transactions across cloud databases

03/12/2019 • 8 minutes to read •  +4

## In this article

[Common scenarios](#)

[Installation and migration](#)

[Development experience](#)

[.NET installation for Azure Cloud Services](#)

[Transactions across multiple servers](#)

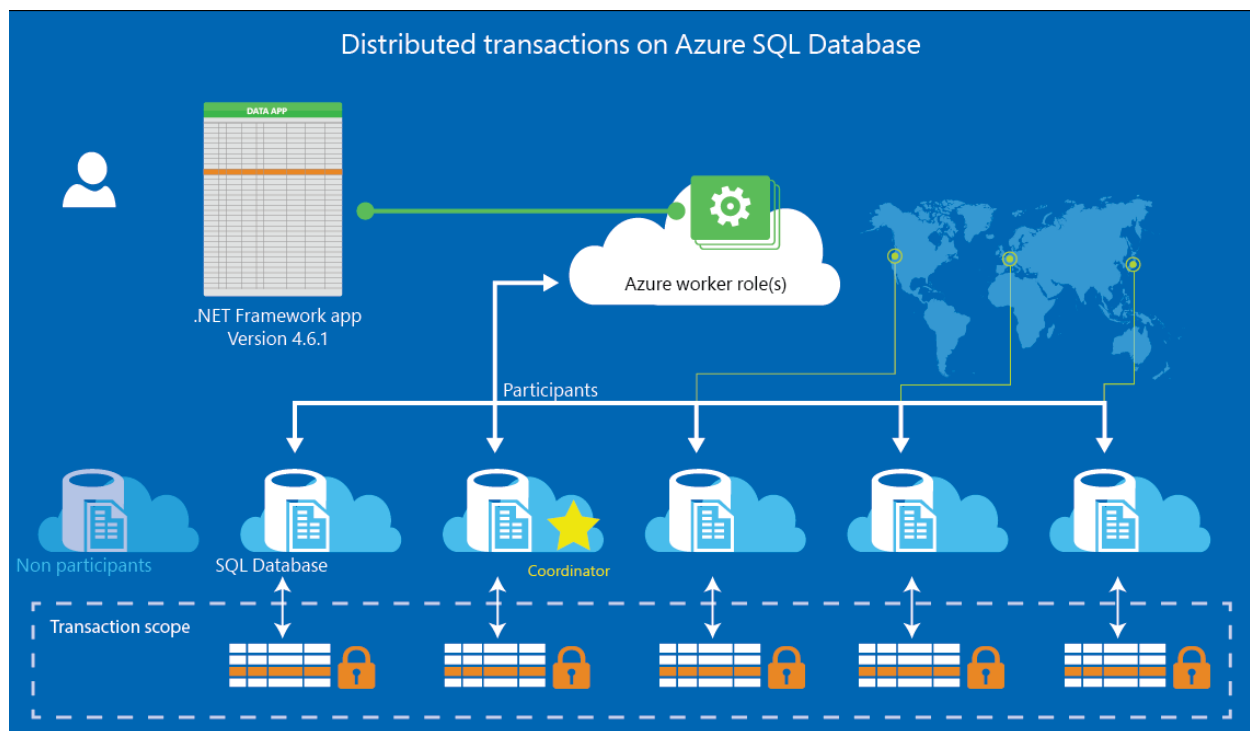
[Monitoring transaction status](#)

[Limitations](#)

[Next steps](#)

Elastic database transactions for Azure SQL Database (SQL DB) allow you to run transactions that span several databases in SQL DB. Elastic database transactions for SQL DB are available for .NET applications using ADO .NET and integrate with the familiar programming experience using the [System.Transaction](#) classes. To get the library, see [.NET Framework 4.6.1 \(Web Installer\)](#).

On premises, such a scenario usually required running Microsoft Distributed Transaction Coordinator (MSDTC). Since MSDTC is not available for Platform-as-a-Service application in Azure, the ability to coordinate distributed transactions has now been directly integrated into SQL DB. Applications can connect to any SQL Database to launch distributed transactions, and one of the databases will transparently coordinate the distributed transaction, as shown in the following figure.



## Common scenarios

Elastic database transactions for SQL DB enable applications to make atomic changes to data stored in several different SQL Databases. The preview focuses on client-side development experiences in C# and .NET. A server-side experience using T-SQL is planned for a later time.

Elastic database transactions targets the following scenarios:

- **Multi-database applications in Azure:** With this scenario, data is vertically partitioned across several databases in SQL DB such that different kinds of data reside on different databases. Some operations require changes to data which is kept in two or more databases. The application uses elastic database transactions to coordinate the changes across databases and ensure atomicity.
- **Sharded database applications in Azure:** With this scenario, the data tier uses the [Elastic Database client library](#) or self-sharding to horizontally partition the data across many databases in SQL DB. One prominent use case is the need to perform atomic changes for a sharded multi-tenant application when changes span tenants. Think for instance of a transfer from one tenant to another, both residing on different databases. A second case is fine-grained sharding to accommodate capacity needs for a large tenant which in turn typically implies that some atomic operations needs to stretch across several databases used for the same tenant. A third case is atomic updates to reference data that are replicated across databases. Atomic, transacted, operations along these lines can now be coordinated across several databases using the preview. Elastic database transactions use two-phase commit to ensure transaction atomicity across databases. It is a good fit for transactions that involve less than 100 databases at a time within a single

transaction. These limits are not enforced, but one should expect performance and success rates for elastic database transactions to suffer when exceeding these limits.

## Installation and migration

The capabilities for elastic database transactions in SQL DB are provided through updates to the .NET libraries System.Data.dll and System.Transactions.dll. The DLLs ensure that two-phase commit is used where necessary to ensure atomicity. To start developing applications using elastic database transactions, install [.NET Framework 4.6.1](#) or a later version. When running on an earlier version of the .NET framework, transactions will fail to promote to a distributed transaction and an exception will be raised.

After installation, you can use the distributed transaction APIs in System.Transactions with connections to SQL DB. If you have existing MSDTC applications using these APIs, simply rebuild your existing applications for .NET 4.6 after installing the 4.6.1 Framework. If your projects target .NET 4.6, they will automatically use the updated DLLs from the new Framework version and distributed transaction API calls in combination with connections to SQL DB will now succeed.

Remember that elastic database transactions do not require installing MSDTC. Instead, elastic database transactions are directly managed by and within SQL DB. This significantly simplifies cloud scenarios since a deployment of MSDTC is not necessary to use distributed transactions with SQL DB. Section 4 explains in more detail how to deploy elastic database transactions and the required .NET framework together with your cloud applications to Azure.

## Development experience

### Multi-database applications

The following sample code uses the familiar programming experience with .NET System.Transactions. The TransactionScope class establishes an ambient transaction in .NET. (An “ambient transaction” is one that lives in the current thread.) All connections opened within the TransactionScope participate in the transaction. If different databases participate, the transaction is automatically elevated to a distributed transaction. The outcome of the transaction is controlled by setting the scope to complete to indicate a commit.

```

using (var scope = new TransactionScope())
{
    using (var conn1 = new SqlConnection(connStrDb1))
    {
        conn1.Open();
        SqlCommand cmd1 = conn1.CreateCommand();
        cmd1.CommandText = string.Format("insert into T1 values(1)");
        cmd1.ExecuteNonQuery();
    }

    using (var conn2 = new SqlConnection(connStrDb2))
    {
        conn2.Open();
        var cmd2 = conn2.CreateCommand();
        cmd2.CommandText = string.Format("insert into T2 values(2)");
        cmd2.ExecuteNonQuery();
    }

    scope.Complete();
}

```

## Sharded database applications

Elastic database transactions for SQL DB also support coordinating distributed transactions where you use the `OpenConnectionForKey` method of the elastic database client library to open connections for a scaled out data tier. Consider cases where you need to guarantee transactional consistency for changes across several different sharding key values. Connections to the shards hosting the different sharding key values are brokered using `OpenConnectionForKey`. In the general case, the connections can be to different shards such that ensuring transactional guarantees requires a distributed transaction. The following code sample illustrates this approach. It assumes that a variable called `shardmap` is used to represent a shard map from the elastic database client library:

 Copy

```

using (var scope = new TransactionScope())
{
    using (var conn1 = shardmap.OpenConnectionForKey(tenantId1,
credentialsStr))
    {
        conn1.Open();
        SqlCommand cmd1 = conn1.CreateCommand();
        cmd1.CommandText = string.Format("insert into T1 values(1)");
        cmd1.ExecuteNonQuery();
    }

    using (var conn2 = shardmap.OpenConnectionForKey(tenantId2,

```

```

credentialsStr))
{
    conn2.Open();
    var cmd2 = conn2.CreateCommand();
    cmd2.CommandText = string.Format("insert into T1 values(2)");
    cmd2.ExecuteNonQuery();
}

scope.Complete();
}

```

## .NET installation for Azure Cloud Services

Azure provides several offerings to host .NET applications. A comparison of the different offerings is available in [Azure App Service, Cloud Services, and Virtual Machines comparison](#). If the guest OS of the offering is smaller than .NET 4.6.1 required for elastic transactions, you need to upgrade the guest OS to 4.6.1.

For Azure App Services, upgrades to the guest OS are currently not supported. For Azure Virtual Machines, simply log into the VM and run the installer for the latest .NET framework. For Azure Cloud Services, you need to include the installation of a newer .NET version into the startup tasks of your deployment. The concepts and steps are documented in [Install .NET on a Cloud Service Role](#).

Note that the installer for .NET 4.6.1 may require more temporary storage during the bootstrapping process on Azure cloud services than the installer for .NET 4.6. To ensure a successful installation, you need to increase temporary storage for your Azure cloud service in your ServiceDefinition.csdef file in the LocalResources section and the environment settings of your startup task, as shown in the following sample:

 Copy

```

<LocalResources>
...
    <LocalStorage name="TEMP" sizeInMB="5000" cleanOnRoleRecycle="false" />
    <LocalStorage name="TMP" sizeInMB="5000" cleanOnRoleRecycle="false" />
</LocalResources>
<Startup>
    <Task commandLine="install.cmd" executionContext="elevated"
taskType="simple">
        <Environment>
            ...
                <Variable name="TEMP">
                    <RoleInstanceValue
xpath="/RoleEnvironment/CurrentInstance/LocalResources/LocalResource[@name='
TEMP']/@path" />
                </Variable>
                <Variable name="TMP">

```

```
<RoleInstanceValue
xpath="/RoleEnvironment/CurrentInstance/LocalResources/LocalResource[@name='
TMP']/@path" />
</Variable>
</Environment>
</Task>
</Startup>
```

## Transactions across multiple servers

### 📌 Note

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

### 📌 Important

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

Elastic database transactions are supported across different SQL Database servers in Azure SQL Database. When transactions cross SQL Database server boundaries, the participating servers first need to be entered into a mutual communication relationship. Once the communication relationship has been established, any database in any of the two servers can participate in elastic transactions with databases from the other server. With transactions spanning more than two SQL Database servers, a communication relationship needs to be in place for any pair of SQL Database servers.

Use the following PowerShell cmdlets to manage cross-server communication relationships for elastic database transactions:

- **New-AzSqlServerCommunicationLink:** Use this cmdlet to create a new communication relationship between two SQL Database servers in Azure SQL Database. The relationship is symmetric which means both servers can initiate transactions with the other server.

- **Get-AzSqlServerCommunicationLink:** Use this cmdlet to retrieve existing communication relationships and their properties.
- **Remove-AzSqlServerCommunicationLink:** Use this cmdlet to remove an existing communication relationship.

## Monitoring transaction status

Use Dynamic Management Views (DMVs) in SQL DB to monitor status and progress of your ongoing elastic database transactions. All DMVs related to transactions are relevant for distributed transactions in SQL DB. You can find the corresponding list of DMVs here: [Transaction Related Dynamic Management Views and Functions \(Transact-SQL\)](#).

These DMVs are particularly useful:

- **sys.dm\_tran\_active\_transactions:** Lists currently active transactions and their status. The UOW (Unit Of Work) column can identify the different child transactions that belong to the same distributed transaction. All transactions within the same distributed transaction carry the same UOW value. See the [DMV documentation](#) for more information.
- **sys.dm\_tran\_database\_transactions:** Provides additional information about transactions, such as placement of the transaction in the log. See the [DMV documentation](#) for more information.
- **sys.dm\_tran\_locks:** Provides information about the locks that are currently held by ongoing transactions. See the [DMV documentation](#) for more information.

## Limitations

The following limitations currently apply to elastic database transactions in SQL DB:

- Only transactions across databases in SQL DB are supported. Other [X/Open XA](#) resource providers and databases outside of SQL DB cannot participate in elastic database transactions. That means that elastic database transactions cannot stretch across on premises SQL Server and Azure SQL Database. For distributed transactions on premises, continue to use MSDTC.
- Only client-coordinated transactions from a .NET application are supported. Server-side support for T-SQL such as BEGIN DISTRIBUTED TRANSACTION is planned, but not yet available.
- Transactions across WCF services are not supported. For example, you have a WCF service method that executes a transaction. Enclosing the call within a transaction scope will fail as a [System.ServiceModel.ProtocolException](#).

# Next steps

For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

---

Is this page helpful?

 Yes  No

---