

Azure Scalability Prescriptive Architecture using the Enzo Multitenant Framework

Many corporations and Independent Software Vendors considering cloud computing adoption face a similar challenge: how should an application be designed to support near linear scalability, and yet keep strong referential integrity to the underlying data?

This white paper offers a solution to this scalability challenge by introducing the nature of scalability in the cloud, tradeoffs, and the use of the Enzo Multitenant Framework to implement a transactional and flexible scalable architecture.

Blue Syntax Consulting specializes in the Microsoft Azure platform and provides architectural and development guidance to corporations leveraging cloud computing technologies.

Created by Blue Syntax Consulting

Published on 12/16/2010

For more information, contact Herve Roggero (hroggero@bluesyntax.net)

Version 1.0.4

Contents

- Introduction 3
- Design of a Multitenant Application 3
 - Scale Up versus Scale Out Designs..... 4
 - Abstracting Logical and Physical Data Access Models 5
 - Benefits of the Enzo Multitenant Framework 6
- Scenarios for Scale Out Architecture 7
 - Reports..... 7
 - Customer Federation 8
- Enzo Multitenant Framework Concepts 9
 - Data Domains and Attributes 9
 - Level of Abstraction 10
 - Configuration (routing, management, definitions)..... 11
 - Shard Definition 12
 - Fan-Out Queries..... 12
 - Schema Management 13
 - Elasticity and Time 13
 - Security 14
- Enzo Multitenant Framework Architecture 14
- About The Author 15
- About Blue Syntax Consulting..... 16

Introduction

Cloud computing, and more specifically the Microsoft Azure platform, offers new opportunities for businesses seeking scalability, high availability and a utility-based consumption model for their applications. As part of its infrastructure, Microsoft Azure places an emphasis on scale-out designs; indeed it is relatively easy to create and deploy web applications in Windows Azure. And with the push of a button, administrators can run a web application on 2 or more web servers.

However, scaling out OLTP (online transactional processing) databases has traditionally been difficult for many reasons, including the lack of transparent distributed database support and limited availability of management tools forcing developers to go through a difficult learning curve. Although the need for increased performance and scalability has always been prevalent, distributed data management has always been considered difficult to tackle without more support from database vendors.

In data warehousing however, distributed data stores along with optimized data access paths have shown that building a distributed data access layer is possible, and manageable. Indeed data warehouses have limited needs to update data and are primarily used to run reports and data analysis tools. The most prominent data warehousing systems leverage a shared nothing¹ data architecture minimizing data access latency.

While performance is always important, cloud-based applications place more emphasis on scalability. Certain cloud applications have a need to host data that belong to a specific customer (such as SaaS applications) or some other entity (such as a date range, or a geographic location); these applications are referred to as multitenant applications in this paper. Multitenant applications provide a shared presentation layer, a shared service layer for implementing business rules and caching, and one or more databases logically or physically separating customer data for increased privacy, performance and scalability. A successful multitenant design allows corporations to keep adding customers without having to worry about the various application layers.

The primary objective of scalability is to support business growth by optimizing resources and flexibility.

Planning to build a multitenant system that leverages distributed data stores can be complicated and requires careful planning. This white paper introduces you to a multitenant architecture leveraging the performance and scalability advantages of a scale out architecture using the Enzo Multitenant Framework.

Design of a Multitenant Application

Multitenant applications have specific objectives that favor scalability while keeping performance and availability as important parameters. The multitenant design proposed in this paper involves the use of SQL Azure as the data repository to leverage strong data integrity and transactional consistency.

¹ A shared nothing architecture is a distributed computed platform in which each machine runs a database; as a result each database has a dedicated set of CPU, memory and I/O footprint.

Scale Up versus Scale Out Designs

Before discussing various scale out architecture options, let's review a typical model which tends to become difficult to maintain as the customer base increases. It is based on the principle that all customer records should be stored in the same database, separated logically by a column storing a key, such as a customer ID. The logical separation is called a **data domain**, or **data federation**. Figure 1 shows a centralized database storing all customer records. The application is responsible for filtering the data correctly every time it fetches records. While this model works well, it becomes increasingly difficult to scale since tables have a tendency to store too many records. Performance tuning becomes very difficult over time and more frequent transaction bottlenecks (or locks) appear due to increased concurrency.

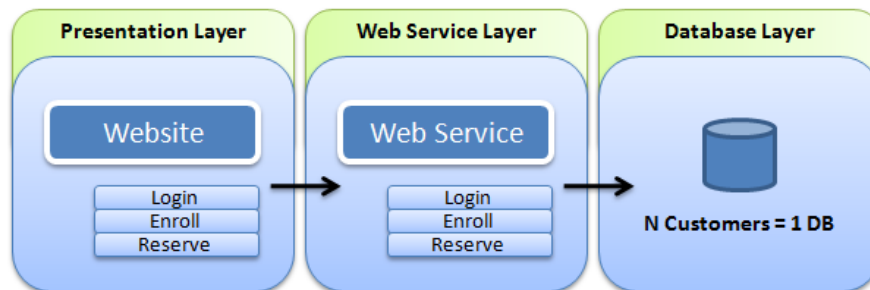


Figure 1 - Traditional Scale Up Architecture

A simple scale out design pattern involves the use of many databases; one per customer. This pattern is called a **Linear Shard** architecture since each database is independent from the other and each customer has its own database. The primary advantage of this implementation lies in the shared-nothing architecture implementation allowing each database to take full advantage of the hardware it is running on, along with full security isolation. In addition, this implementation does not rely on schemas, which simplifies certain management tasks. Figure 2 shows an example of a linear architecture.

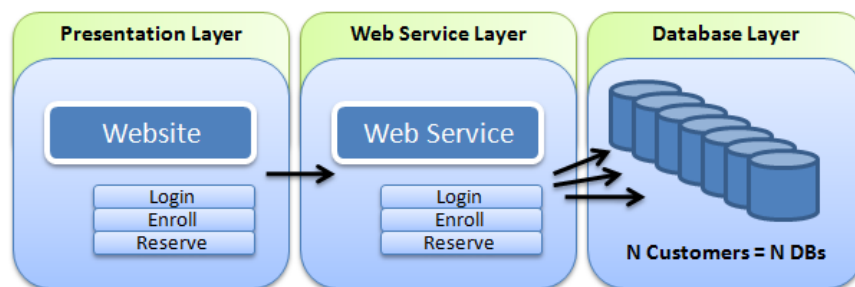


Figure 2 - Linear Shard Architecture

An **Expanded Shard**² database architecture is another possible design. In this approach, N customers are stored on a number of databases in which records are spread out almost evenly. The expanded shard architecture is a scale out model in which the databases making up the shard are not assigned to specific

² An expanded shard is a form of elasticity in which the number of databases forming the shard is greater than the number of data domains.

customers (or data domain). The ability to find records in this model depends on applying the proper filters when issuing database commands, much like in the scale up design discussed previously. Records are added to the shard using a round-robin mechanism ensuring even distribution of records in the shard. Also, querying the shard requires accessing one or more databases in parallel and aggregating the results in the web service layer. Figure 3 shows an example of this implementation.

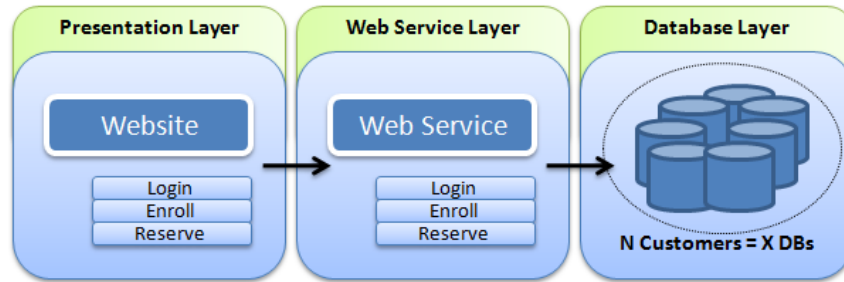


Figure 3 - Expanded Shard Architecture

As you may expect, it is also possible to build a design that mixes the linear and the expanded architecture, also referred to as a **Hybrid Scale Out model** as depicted in Figure 4. Pockets of expanded shards could be defined for a group of customers. For example, customers in Europe could be assigned to an expanded shard, while customers in South America could be assigned to another expanded shard. Shards could also be created based on timeframes, such as the archiving of records on a yearly basis.

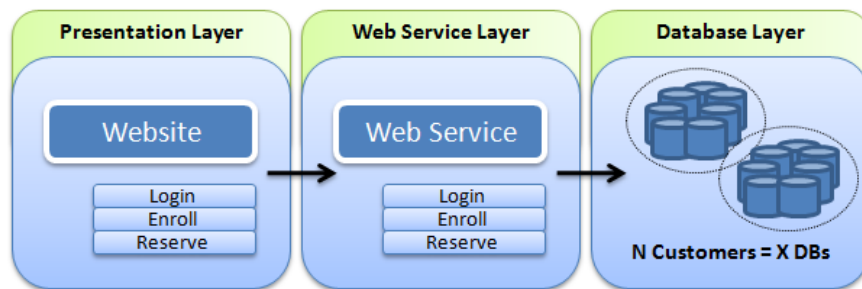


Figure 4 - Hybrid Scale Out Architecture

Finally, there is a special case called **Compressed Shards**³, in which the number of databases is less than the number of data domains, but greater than one (otherwise it would be a Scale Up architecture). To achieve this scale out model, certain customers are grouped in a single database but separated by schema. Each schema represents a logical database within a database and each schema is secured in a way that prevents other customers from ever seeing the other customers' data.

Abstracting Logical and Physical Data Access Models

As you can see in the previous examples, the scalability model chosen impacts the number of databases implemented, which impacts the application architecture, design, testing and implementation. Figure 5 provides a summary the various scale out design options discussed so far. The horizontal line represents

³ A compressed shard is a form of elasticity where there are fewer databases than data domains usually implemented using schema separation

the number of physical databases implemented, while the vertical line represents the number of data domains implemented (such as the number of customers, if Customer ID represents the data domain). The Scale Up Architecture follows the vertical line. When there are more physical databases than logical domains, an Expanded Shard is usually implemented. When there are fewer databases than logical domains, a Compressed Shard is typically implemented.

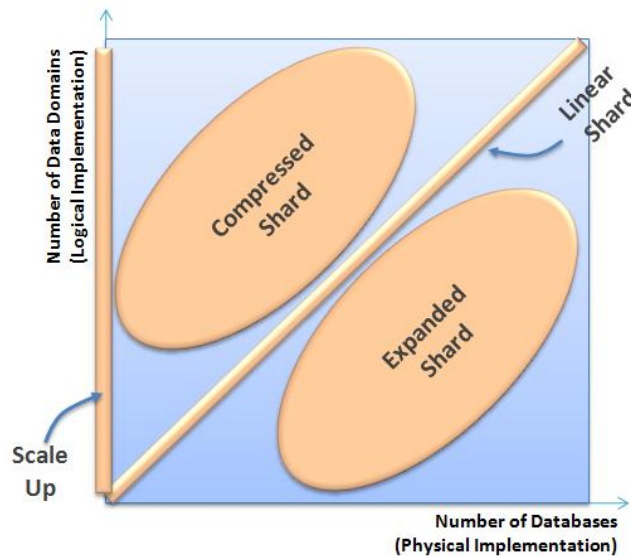


Figure 5 - Multitenant Design Options

In order to ensure a viable architecture, scale out designs should not rely on physical implementation knowledge, but instead access data using the logical implementation. Decoupling the application code from the physical implementation of the database layer creates an abstraction layer that gives businesses the option to reconfigure the physical layer without modifying the code.

Benefits of the Enzo Multitenant Framework

Some of the challenges inherent to designing a scale out architecture includes the need for flexibility as business needs change, and having the choice to implement quickly the type of scalability that best fits business demands. A new application may simply start using the scale up model, since it is probably the easiest to maintain at first. However as time goes by, a scale out model may make more sense. Switching scalability model over time can be extremely difficult and risky.

The Enzo Multitenant Framework helps you implement a flexible scalability model, and supports all the scalability patterns discussed so far, from scale up to an expanded shard model. The Enzo Multitenant Framework provides a metadata-driven database selection mechanism, allowing developers to specify a database, or group of databases, by attribute. Attributes are assigned to groups of databases and to each database optionally. The Enzo Multitenant Framework allows developers to specify operations against all databases, databases within a group, specific databases across multiple groups, or even a single database, based on attributes.

In order to provide maximum flexibility, developers should code their Data Access Layer (DAL) against the Logical Data Domains independently of the number of databases actually implemented.

The Enzo Multitenant Framework provides other advantages, such as multithreaded execution of database commands, queuing database commands for batch calls, optional use of round-robin when inserting records in shards and automatic refresh of configuration settings.

The Enzo Multitenant Framework provides the following advantages:

- Support for multiple scalability models
- Abstraction of database knowledge from application code through the use of attributes
- Automatic configuration caching and multithreading for optimum performance
- Management portal for configuring the scalability model
- Support for time-based elasticity implementation
- Leverages the underlying SQL Azure and SQL Server transactional capabilities

Scenarios for Scale Out Architecture

Scale out architectures can provide significant benefits, including performance improvements, scalability readiness and more. This section outlines two scenarios that benefit from a scale out architecture.

Reports

Interestingly enough, reports usually stress server resources and can be the cause for many performance issues found in ERP and CRM systems. While some reports run against OLAP databases, many still need to execute on OLTP databases. Indeed, operational reports have a tendency to read from data that is as accurate as possible, such as an occupancy report, and can create disk I/O, CPU and memory bottlenecks quickly. Historical reports too can have drastic performance implication on an OLTP database; however in many cases historical reports are executed on secondary databases on which no transaction is running. Nevertheless processing requirements can be so high that running a report on any database server can create critical bottlenecks.

Since most of the time reports create an I/O bottleneck first, spreading the data pulled by a report on as many independent disks as possible makes sense. In addition, since heavy I/O causes CPU bottlenecks, it makes further sense to spread the data over multiple physical database servers. As you can expect, spreading a large report that would normally create millions of I/O on a single database to 10 database servers would likely improve performance by a factor of up to 10 theoretically. In practice however the actual performance improvement depends on other infrastructure bottlenecks such as the maximum network bandwidth available and how many CPUs are installed on the machine running the report itself.

Tests conducted by Blue Syntax Consulting in controlled environments show that most reports can be executed two to three times faster when using a scale out architecture. The result is mostly due to the Enzo Multitenant Framework used by the report automatically taking advantage of multiple processors and returning data from disparate databases simultaneously. Figure 6 shows how the Enzo Multitenant

Framework platform consolidates information from multiple databases using a fan-out⁴ query approach, hence reducing the execution time of the report. The report can be configured to run against multiple databases in parallel transparently using the Enzo API provided with the Enzo Multitenant Framework.

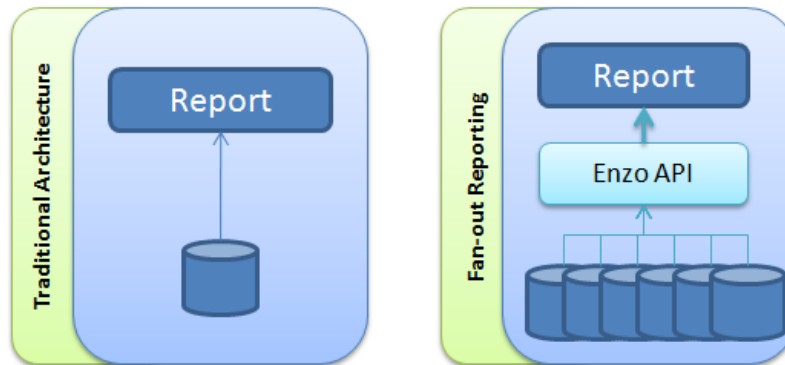


Figure 6: Reporting with the Enzo Multitenant Framework

Customer Federation

Companies that run hosted applications, such as SaaS (Software as a Service) vendors, have a choice to run their applications and databases in-house on their own servers or use a rented infrastructure such as Microsoft Azure. Regardless of the approach, SaaS vendors typically face the same decision point: should the design of their solution use a single database for all their customers, or use a database per customer? Which scalability model should be used? How should records be federated⁵?

The decision to implement a single or a distributed database model can have a significant impact in many areas, some of which include the ability to support a large number of databases, the lack of cross-database data integrity, the ability to upgrade one customer to the next release but not the others, the risk of having slightly different database schemas unintentionally and so on. Another important factor is the ability to scale. Indeed, a single database model has a tendency to introduce performance issues that are directly related to the decision of having a single database architecture. Furthermore, a single database architecture model requires corporations to plan for growth ahead of time and invest in the necessary infrastructure that will be necessary to support the growth of the business 3 to 5 years ahead.

In contrast, Cloud computing is about just-in-time scalability, up or down, giving you the ability to increase your infrastructure footprint on demand and pay for the increased footprint only when you use it. While the platform provided by Microsoft Azure gives you this flexibility for web services (such as Windows Azure), the database architecture is still in the hands of companies hosting their SaaS solution in the cloud. Scaling out the database tier can be achieved in multiple ways as discussed previously.

In this context, just-in-time scalability allows corporations to optimize their platform investments by minimizing, or avoiding large capital investments. The Enzo Multitenant Framework provides the necessary building blocks for implementing a flexible scalability model by providing the necessary

⁴ A fan-out query is a database command executed over multiple databases in parallel

⁵ Data federation is a term used to describe the logical or physical separation of data

programmatic and configuration support. Figure 7 shows a possible growth pattern in which a SaaS application is automatically made aware of new databases as they are added over time, without modifying the SaaS application, as new customers sign up for the service. Using the Enzo API provided by the framework developers can focus on developing application features without the need to worry about the final scalability model selected.

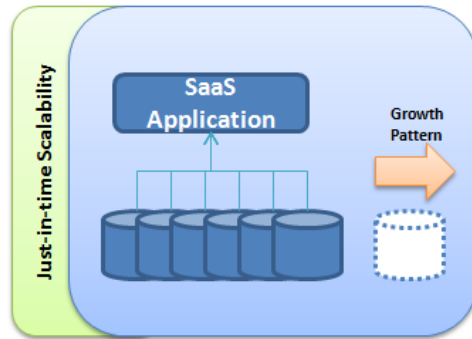


Figure 7: Federation Growth Pattern

Enzo Multitenant Framework Concepts

As discussed previously, building a distributed database topology provides unique opportunities for both performance and scalability. This section outlines the most important aspects of the Enzo Multitenant Framework that helps corporations achieve scalability.

Data Domains and Attributes

The premise behind distributed databases is the ability to logically and physically separate customer records to provide both improved performance and scalability. However in order to provide the mechanism needed to retrieve specific customer data, the separation needs to be performed using a key; for example a Customer ID may be used as the key that separates database records⁶.

There are however other separation models, such as the use of attributes. Using attributes, or meta-data, allows records to be separated by more than one dimension. For example, customer records could be identified using Customer ID and a State value (Florida, California...) or a Country property. A date could also be used as one of the attributes allowing records to be logically separated by timeframe, such as yearly sales or monthly support calls.

Distributing data by key and/or attribute allows the creation of one or more data domains that logically and/or physically distribute workloads. Similar to OLAP databases, federation keys and attributes allow slicing of data using predefined domains.

In Figure 8 the growth pattern is implemented using a time boundary, in which a new independent set of databases is built on a yearly basis, such as the shard defined in 2009 is unaware of the shard defined for 2010. Within the 2010 shard, 12 databases are created (one per month) over three sets of customer

⁶ SQL Azure will support the use of Federation Keys to implement this model

IDs. As a result, a data domain is defined by the year, a customer ID and a month. Using attributes, a developer could easily run a database statement within a specific data domain, or across months, customers, or even years.

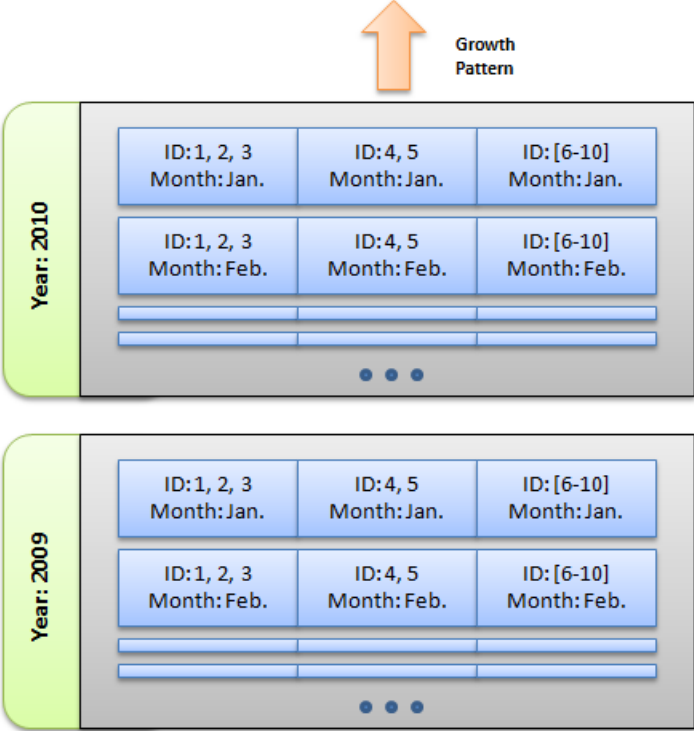


Figure 8: Federation Keys and Attributes

Level of Abstraction

Once the data has been sliced by data domains using federation keys and attributes, the next challenge is to provide simple programmatic constructs that allow developers to connect to the appropriate data domain, and to optionally read or change data from multiple domains. The level of abstraction provided to developers in accessing data in a distributed model is critical to its success.

For example, SQL Azure provides a federation key⁷, which could be a Customer ID for example. Knowing a Customer ID is sufficient for developers to leverage data federation in SQL Azure. All that is required for developers to build systems against multiple databases is to know which Customer ID to use ahead of making a database call by leveraging the USE FEDERATION command in T-SQL. The USE command in SQL Azure provides a programmatic construct to select the context of future queries which will execute against a slice of the data that belongs to the correct Customer ID.

In the Enzo Multitenant Framework, developers execute statements against one or more databases by specifying attribute values, such as a Customer ID, or using the default database if one has been defined. Developers do not need to know the connection string to the correct database; the Enzo Multitenant

⁷ At the time of this writing SQL Azure does not yet support data federation. However Microsoft announced support for data federation in future builds of SQL Azure.

Framework automatically selects the correct connection strings for a customer based on the attributes provided.

Figure 9 shows an example of a fetch operation performed against a set of shards (also called a Shard Cluster) in which the Customer ID of 4 was specified. The following pseudo-code shows how a developer would implement such logic using the Enzo Multitenant Framework:

```
SET SHARD ATTRIBUTE: CUSTOMER ID = 4  
FETCH ALL CUSTOMER_SALES GROUP BY YEAR
```

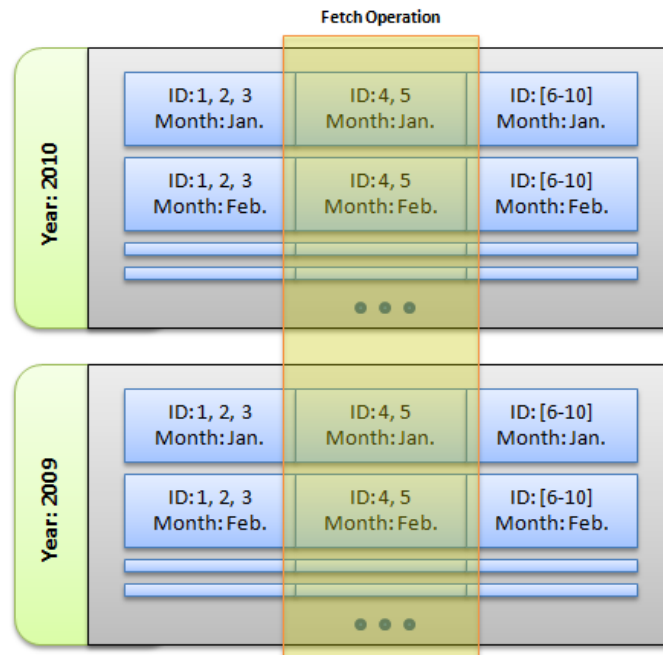


Figure 9: Attribute-driven data fetching

Configuration (routing, management, definitions)

Managing distributed databases is perhaps the most important task data architects will face since performance and scalability will be directly affected by the data domains selected. Once the data domains have been defined, data architects need to configure the multitenant system so that data can be accessed based on keys or attributes.

In SQL Azure, configuring the data federation environment is simple and only requires a federation key. Once the federation is defined, tables are created as part of the federation.

In the Enzo Multitenant Framework, data domains are created through a collection of shards (called a Shard Cluster), and a collection of databases in each shard. Unlike SQL Azure, a shard can be made of mixed databases, such as SQL Azure and on-premise SQL Server databases, creating additional opportunities for performance and scalability behaviors. Because a shard is defined as a collection of databases, the Enzo Multitenant Framework implements connection routing logic that automatically

selects the correct database(s) based on attributes. Figure 10 shows the management screen of the Enzo Multitenant Framework listing the databases that belong to one of the shards.

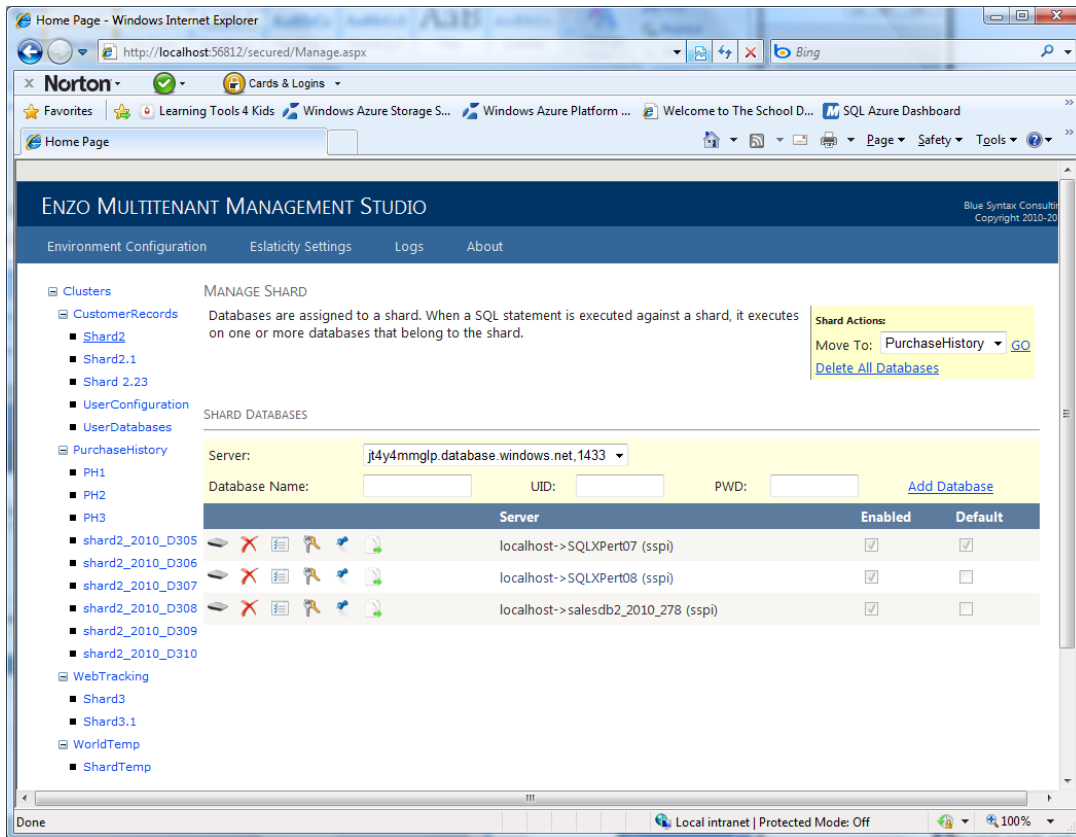


Figure 10: The Enzo Multitenant Framework Management Interface

Shard Definition

By definition a database shard is a collection of fragmented and loosely coupled data elements. Different data federation providers may define a shard differently. In SQL Azure for example, a shard is a data domain implemented in a single database, segregated by a federated key. In the Enzo Multitenant Framework, a shard is a collection of loosely coupled databases.

Fan-Out Queries

When data needs to be retrieved across databases, developers must create fan-out queries. The complexity of creating fan-out queries depends on the federated system. For example, both SQL Azure and the Enzo Multitenant Framework support fan-out queries. However by nature SQL Azure can only support fan-out queries within the boundaries of SQL Azure. The Enzo Multitenant Framework transparently extends support of fan-out queries beyond SQL Azure by also querying SQL Server databases.

The following code for example executes a SELECT statement against all the databases found in the default shard for which the COUNTRY attribute is "US". If this attribute is found on multiple databases,

including SQL Azure and local SQL Server, the fan-out processing engine of the **Enzo Shard API**⁸ will fetch all these records in parallel, using multi-threading logic, and return a single data set to be later displayed on a customer user interface.

```
ShardCluster.Shard.DBExecutionOption deo = new ShardCluster.Shard.DBExecutionOption();
deo.Attributes.Add("COUNTRY", "US");
deo.LoadOption = ShardCluster.Shard.DBLoadOptions.Default;
DataTable dt = sc.DefaultShard.ExecuteDataTable("SELECT * FROM CUSTOMER_HISTORY");
```

Schema Management

Perhaps one of the most challenging aspects of data federation is the assurance that every database used in the federation has the same object definition, or schema. Controlling schema variations can be challenging and is an important part of release management. At the moment none of the federation providers that allow scaling across databases offer strong schema management or synchronization.

In order to minimize schema differences over time, the Enzo Multitenant Framework offers the ability to create new databases based on a database template database using the new COPY operation in SQL Azure. This feature is available through the elasticity configuration settings and only applied to cloud databases.

Elasticity and Time

Elasticity refers to the ability of a federated system to automatically expand, based on capacity or time thresholds. Automated scalability can be very difficult to implement because developers may need to be aware of the elasticity parameters in order to take advantage of it.

The Enzo Multitenant Framework offers support for elasticity based on time parameters in SQL Azure. For example new databases can be added to a shard every month, which enables complex business scenarios that require automatic growth. In addition, the Enzo Multitenant Framework supports automatic shard creation as well along with automatic archiving.

As an example, a utility company may have a requirement to store phone conversation logs, which can grow very quickly over time. Reports on the other hand need to be able to quickly search through the database to run monthly statistics for example. Implementing an elastic shard for this scenario makes sense, because new databases would be created automatically on a monthly basis; in addition reports would be able to fan-out queries to fetch the data faster than in a single database footprint.

Figure 11 shows one of the configuration screens of the Enzo Multitenant Framework specifying various elasticity behaviors that can be used by shards.

⁸ The Enzo Shard API is a .NET library developers use to access distributed databases

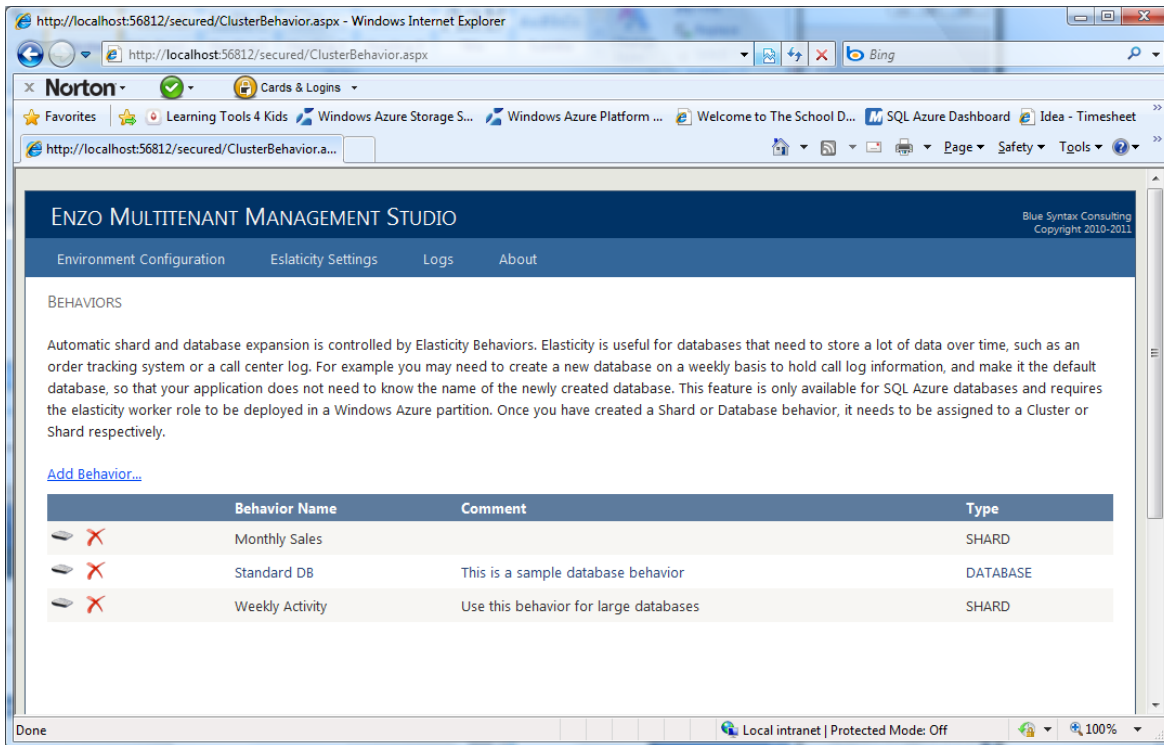


Figure 11: SQL Azure Elasticity with the Enzo Multitenant Framework

Security

Implementing data federation introduces certain security challenges, such as securing the database connection strings that make up a shard, and using a central auditing platform that can equally implement strong access monitoring.

Centralizing database connection strings can be a benefit, specifically when considering that most database connection strings are not encrypted in application configuration files. The Enzo Multitenant Framework stores passwords in an encrypted format in its configuration database. Applications do not need to be aware of all the databases that make up a shard, since the shard framework automatically fetches this information whenever necessary.

Enzo Multitenant Framework Architecture

The Enzo Multitenant Framework is a complete system allowing companies to scale out their databases easily by offering a federation-aware API, a management console and an elastic service providing automatic growth for SQL Azure databases. The framework is designed so that the API can automatically be made aware of configuration changes made in the management console, including changes to database connection strings and shard definitions.

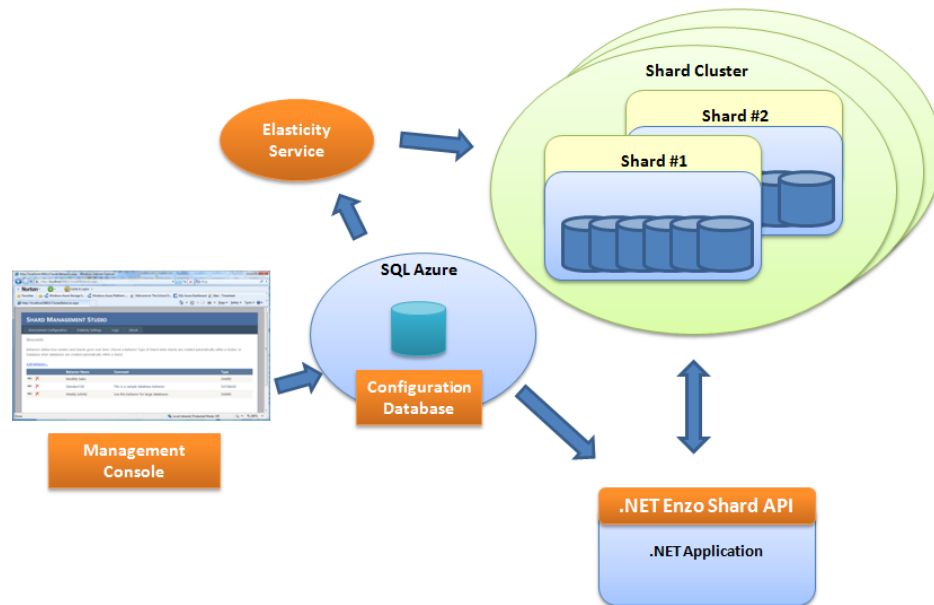


Figure 12: Enzo Multitenant Framework Architecture

Figure 12 shows the four main components of the Enzo Multitenant Framework. The management console is used to define shards, shard clusters, attributes on shards, elasticity settings and a few additional settings for the Enzo Shard API. The elasticity service is an optional component and runs in Windows Azure. It monitors the shard environment and automatically creates new shard and databases if configured to do so. The .NET Shard API provides the federated data access layer allowing developers to use data federation across multiple .NET projects including websites, windows applications, .NET reports and more. Finally the configuration database contains the actual information defining the data domains (shards) and attributes. The configuration database is installed in SQL Azure if the elasticity service is needed.

About The Author

Herve Roggero, co-author of Pro SQL Azure, is the founder of Pyn Logic, a company focusing on advanced SQL Server security and performance, and Blue Syntax Consulting, focusing on Azure consulting services and related technologies. Herve's experience includes software development, architecture, database administration and senior management with both global corporations and startup companies. Over the last 15 years, Herve has worked in the Education, Financial, Health Care, Management Consulting and Database Security sectors. In his last full time position, Herve was a Director of Software Development at a health care company in Florida. He has been freelancing as a .NET and SQL Server Architect since 2007. He holds multiple certifications, including an MCDBA, MCSE, MCSA. He also holds an MBA from Indiana University. Herve is heavily involved with the South Florida SQL Server community, speaks at multiple venues, and runs SQL Saturday events in South Florida.

About Blue Syntax Consulting

Blue Syntax Consulting provides high value-added services to customers by offering high level executive guidance on how to best leverage Cloud Computing technologies with the Microsoft Azure platform and low-level implementation services, including architecture guidance, development and testing. Blue Syntax Consulting develops leading edge technologies allowing corporations to adopt cloud computing sooner, including the Enzo Multitenant Framework and other solutions.

For more information, visit www.bluesyntax.net or contact hroggero@bluesyntax.net.