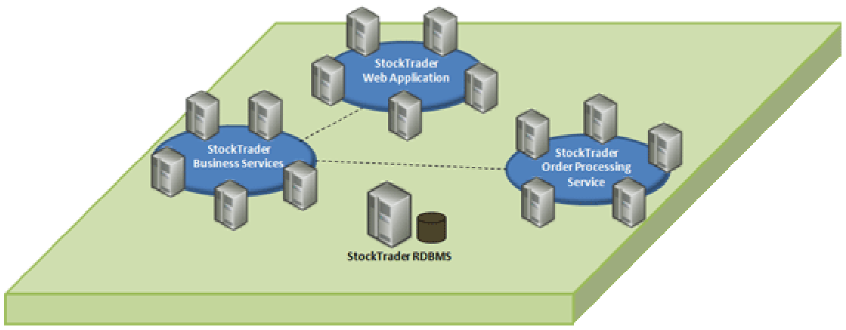
StockTrader 5.0 Technical Documentation

An End-to-End Service-Oriented Sample Application Illustrating Windows Azure Platform Cloud Migration and Integration

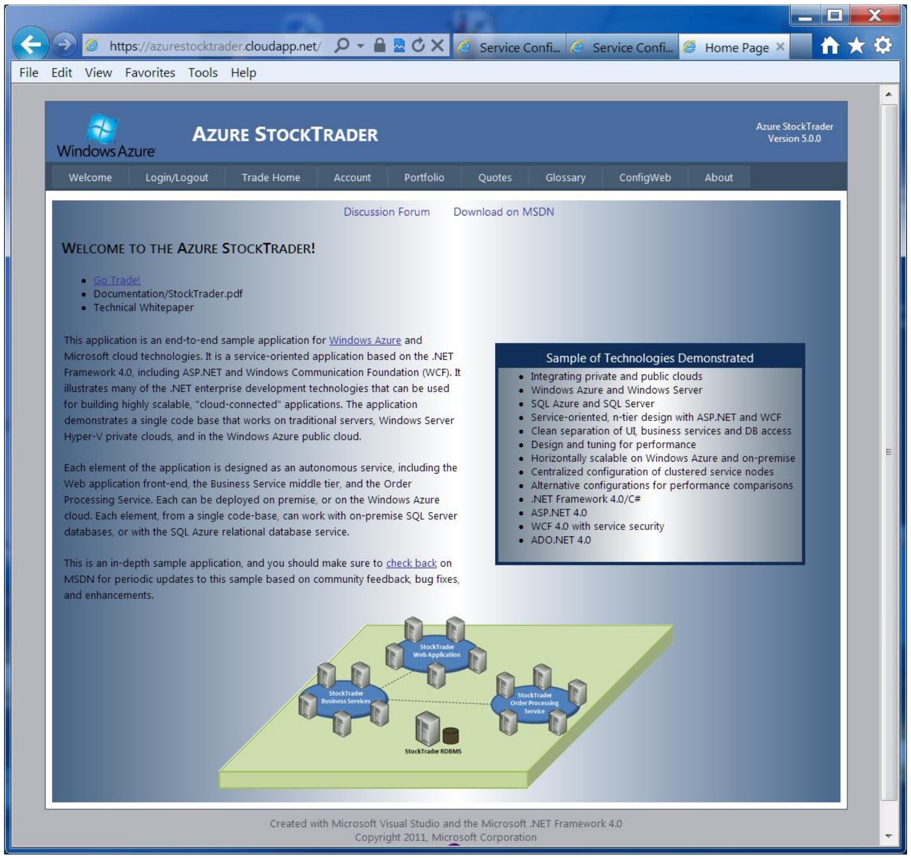
6/9/2011


StockTrader Web Application
StockTrader Business Services
StockTrader Order Processing Service
StockTrader RDBMS

Contents

Introduction

StockTrader 5 is an end-to-end sample application based on an online stock-trading scenario. The latest version of the application illustrates the migration of an on-premise application to the Windows Azure Platform. The application is running live on Windows Azure at https://azurestocktrader.cloudapp.net. The sample application illustrates the use of ASP.NET and Windows Communication Foundation (WCF) technologies in an end-to-end service-oriented architecture running in the cloud. As such, the application illustrates many best-practice programming practices for the Windows Azure Platform, SQL Azure, ASP.NET and WCF including the use of an n-tier, service-oriented design. Originally designed as a performance sample and downloadable benchmark kit for on-premise applications, the application now also illustrates best-practice programming for building high-performance and horizontally scalable applications for the cloud. The migration of the existing sample application from an existing on-premise application to the Windows Azure Platform was extremely straightforward, and this paper discusses the variety of new public cloud and hybrid deployment topologies the Windows Azure Platform supports.



WELCOME TO THE AZURE STOCKTRADER!

- Go Trade!
- Documentation/StockTrader.pdf
- Technical Whitepaper

This application is an end-to-end sample application for Windows Azure and Microsoft cloud technologies. It is a service-oriented application based on the .NET Framework 4.0, including ASP.NET and Windows Communication Foundation (WCF). It illustrates many of the .NET enterprise development technologies that can be used for building highly scalable, "cloud-connected" applications. The application demonstrates a single code base that works on traditional servers, Windows Server Hyper-V private clouds, and in the Windows Azure public cloud.

Each element of the application is designed as an autonomous service, including the Web application front-end, the Business Service middle tier, and the Order Processing Service. Each can be deployed on premise, or on the Windows Azure cloud. Each element, from a single code-base, can work with on-premise SQL Server databases, or with the SQL Azure relational database service.

This is an in-depth sample application, and you should make sure to check back on MSDN for periodic updates to this sample based on community feedback, bug fixes, and enhancements.

Sample of Technologies Demonstrated

- Integrating private and public clouds
- Windows Azure and Windows Server
- SQL Azure and SQL Server
- Service-oriented, n-tier design with ASP.NET and WCF
- Clean separation of UI, business services and DB access
- Design and tuning for performance
- Horizontally scalable on Windows Azure and on-premise
- Centralized configuration of clustered service nodes
- Alternative configurations for performance comparisons
- .NET Framework 4.0/C#
- ASP.NET 4.0
- WCF 4.0 with service security
- ADO.NET 4.0

Created with Microsoft Visual Studio and the Microsoft .NET Framework 4.0
Copyright 2011, Microsoft Corporation

Figure 1: Azure StockTrader 5.0 Home Page

Azure StockTrader: Migration from Existing On-Premise Application

The application was previously created as a benchmark and performance-oriented sample application for Microsoft .NET and Microsoft SQL Server. The new StockTrader 5 illustrates how an existing on-premise application can be easily migrated to the Windows Azure Platform. All tiers of the application have been migrated and are now running as cloud-connected applications and services on Windows Azure. Each tier is deployed as an independent hosted service, and each can be deployed to geographically disperse Azure service domains:

- Web user interface (ASP.NET)
- Business Services (Windows Communication Foundation middle tier service)
- Order Processor Service (Windows Communication Foundation transaction processing tier)
- Database tier (Microsoft SQL Azure)

The migration was straight-forward from a development perspective. In fact, **no lines** of business logic code had to be changed, and **every existing SQL Server query** worked without changes when moving to SQL Azure. Less than 50 new lines of code were added to the existing data tier to account for the Internet-hosted nature of SQL Azure. These lines of code are contained in a common method all database access utilizes, and deal with the possibility of transient connections over the Internet. In addition to the main business database for the application, each tier of the original application utilized a SQL Server database to store and access its configuration data. Each of these databases were also migrated to SQL Azure—in all 4 complete schemas (DDL) and data manipulation (DML) layers migrated with ease. The following tables illustrate the migration changes, as well as some additional optimizations that were done for the cloud-hosted implementation of each service tier.

StockTrader Tier	Original Lines of Code (SLOC)	Changed or Added Lines for Azure Hosting
Trade Web Application	**2,945**	**0**
Business Services	3,626	0
Data Access	1,616	48
Order Processor	626	0
Total	**8,813**	**48**

Table 1. Lines of Code.

Redesigned Lines for StockTrader 5 and Reason for Redesign	Lines of Code	Reason for Re-design
Trade Web Application	**40**	Added asynchronous page processing in two key pages to improve latency (multiple web service calls made on parallel threads from server)

Trade Web Application HTML	~6,000	Completely redesigned to be cross-browser (MSIE, FireFox, Google Chrome, Apple Safari) and render well in mobile devices (Windows Phone 7, Apple IPhone, Google Android devices).
Business Services	21	Consolidated certain web service calls to make less chatty
Data Access	10	Consolidated certain SQL queries
Order Processor	0	NA
Total	**71 lines not counting HTML redesign**	

Table 2. Optimizations for each tier.

In summary, within a matter of days all tiers of this .NET application were migrated to Windows Azure, hooked up and working as a consolidated service-oriented application with backing SQL Azure databases. Of course the HTML redesign took longer, as did testing in multiple browsers and mobile devices—frankly something we were not concerned about in the previous on-premise version. Thinking through the security of each tier and changing the WCF services to implement message/transport security across each tier took a few more days. This is discussed in a later section, but the .NET Windows Communication Foundation (WCF) also made this straightforward—and had no impact on any existing code.

The end result of the migration is an application that runs, **with a single code base**, on premise or hosted on Windows Azure as a cloud application accessible over the Internet. The new version, as a cloud application, implements the new levels of security, and a new cross-browser user interface that is accessible from PCs, tablets and any mobile device such as Windows Phone 7, Android phones, and Apple IPhones. Visual Studio 2010 and Microsoft Expression were used for the development and migration, including the development and integration of the new user interface. The new user interfaces leverages ASP.NET features such as Master Pages and data-bound Web controls. To ensure a fast experience for Internet users, new functionality was added such as a few SQL query optimizations and asynchronous ASP.NET page operations where it made sense.

The StockTrader Smart Client Desktop Application

In addition to the Web user interface (ASP.NET), the sample also includes a Windows Presentation Framework desktop client. We added a few lines of configuration to this application, which is now able to utilize either on-premise services, or the StockTrader Business Service layer running on Windows Azure. No logic changes were necessary here, other than adding the selection so the user can easily point to the BSL on Azure, and securely connect from the smart client

Migration Notes

Not all applications will move quite so easily to the cloud, but is amazing how this .NET application that incorporates so many Microsoft .NET enterprise platform technologies migrated so easily to Windows Azure and SQL Azure. Some key factors that contributed to the ease-of-migration:

- **Server-based application**. Since the application was already web-based with backing business services, there was a clean mapping to Azure Worker and Web roles.
- **Good object-oriented design**. The existing architecture helped a lot. The previous on-premise version, for example, cleanly separates UI from business logic from data access. All data access channels through a common code-block that is already optimized for performance. So instead of having to call the new transient connection method (the one place we did add code for SQL Azure) in hundreds of places throughout all the application tiers, only one class and a handful of common methods had to be changed—and once changed in this class, the entire application then was using the new logic automatically.
- **Service oriented design**. While today only the StockTrader 5 Web application uses the backing business services (which in turn uses the order processor service), now that these are hosted in the cloud, other applications can potentially integrate these services—from anywhere in the world. So, the original SOA design also naturally fits with a cloud model. For example, the original smart-client Windows Presentation Foundation desktop client can now simply make service calls to the Azure-hosted business service layer.
- **SQL Azure.** Since it's a true multi-tenant cloud implementation of SQL Server, all of the StockTrader data schema and queries just moved[1] right up to the cloud database. In fact, this was the first step of the migration process. Also, once migrated, we were able to run any application tier locally, behind the corporate firewall, but access SQL Azure over a very secure, encrypted communication channel. So, we can run any tier of the application on-premise, yet fully leverage SQL Azure as a relational database service in the cloud.
- **Visual Studio and the Windows Azure Platform with .NET**. As a .NET application, we could count on all the full .NET Framework features already being present on Azure, including:
  - The Common Language Runtime (already optimized for server applications)
  - ADO.NET and all data access technologies used
  - Full multi-threading; Azure instances are full Windows Server VMs;
  - Full Windows Server including IIS 7.5. All OS development elements from a platform perspective are already on the base Azure images—at least for this application. There were **absolutely zero additional elements** that needed to be installed/configured on the Azure images, other than deploying the application itself by way of point-and-click or automated scripts. Scale out is simple point and click or a configuration file update. Developers that might need to install additional application libraries (such as third party libraries), can either package them directly in their application, or use fully automated Windows PowerShell scripts to automatically install them on the startup of an Azure instance.
- **Visual Studio 2010 and the Windows Azure SDK**. These of course were integral for the migration for development, testing, packaging and deployment. Installing the Windows Azure

[1] In fact, on the initial deployment of the application, there were some thoughts that many of the queries and data access logic would need changing. This was simply not the case. The full application, with all existing SQL queries, worked on the first deployment against SQL Azure.

SDK and tools for Visual Studio took less than one minute on the development machines. Key features that really helped:

- The local Azure testing model. The application does not leverage Azure storage (rather, it used SQL Azure); but even if it did, it's quite unique that you can run a full local test environment, with full step-through debugging, that emulates Azure on the desktop. This Azure test environment launches automatically from Visual Studio, and emulates Azure compute and storage; so there is no need to cycle through cloud deployments with each application unit test cycle.
- The fact the application is designed on .NET. This greatly added to the overall productivity achieved; and the ultimate performance and stability of the application.

Instant Benefits of the Azure Migration

It is important to note the many benefits of running this application in the Azure cloud, because there are many.

Worldwide Web Access and Mobile Device Access

First of all, the application is now accessible from anywhere in the world with Web access, with no local setup and configuration required. In fact, the full Web interface works quite well on all major desktop browsers, and all major mobile devices.

Instant Highly Available SQL Server RDBMs

When developing, testing and running the on-premise version, four different databases need to be created and installed, potentially on four different provisioned machines, each with a local SQL Server installation. With SQL Azure, even if running/testing the logic on premise, no physical machines need to be provisioned, setup with SQL Server, and maintained. For the deployed database, to get extreme throughput (which the application can achieve based on its design), a large amount of time and money has been spent in the past on high-end database setups with mid-range storage options such as SANs arrays ad redundant disks with appropriate optimizations on the controllers and RAID-level used. When deploying to SQL Azure, this is all automatic; the SQL Server instance running your database is already optimized, and running on very fast storage. Even better, SQL Azure is automatically giving you high availability at the data tier, since it automatically runs three online replicas of your database; so if any instance goes down, failover to a backup replica happens automatically. Setting up such failover clusters on premise, from a hardware, networking and software standpoint, can literally take weeks. And the hardware can be quite expensive. On SQL Azure all of this is automatic.

No More Database Hardware or Database Software Maintenance

With the SQL Azure implementation, there is no need to worry about hardware maintenance, hardware support costs and the like. Additionally, as a truly multi-tenant database service, all SQL Server patching and upgrades are taken care of automatically in the data centers. And, with the online high availability—with no downtime! The cost savings potential is simply enormous. And no one on the development team ever spent any time setting up hardware, installing SQL Server, etc. All of this time could be focused on the development of the actual application.

Instant Scale Out of Application

When running on Windows Azure, scale out is a simple matter of updating a configuration file. Azure automatically provisions a new guest OS, and deploys and starts the application. The Internet endpoint is automatically load balanced against running instances by the Azure Fabric Controller. So, unlike some cloud environments, there is no need to provision/setup load balancers and NAT devices. At the same time, you can establish internal-only endpoints if you need them—for individual instances to privately communicate with each other. The Configuration Service that StockTrader utilizes, for example, make us of these internal endpoints to coordinate configuration changes across potentially dozens of live running instances of the application—all within seconds without any application downtime.

Ultimately scale out means Windows Azure is making it very easy to add capacity to service more users, faster. Adding this capacity is done in minutes, without having to buy, provision, or setup any new hardware, load balancers, or networks.

Rolling Upgrades of Application for High Availability

When the development team is ready to deploy an updated implementation of the application (perhaps with new functionality or bug fixes); Windows Azure automatically performs a rolling upgrade, so the application remains available to online users. Instances are automatically taken out of the load balancer (Azure Fabric Controller) on a rolling basis, upgraded to the new version, and then re-activated in the fabric to start processing requests again. The same is true when the base Windows Azure OS is patched or upgraded by Microsoft within the data centers (this happens automatically on a periodic basis).

Automatic Hardware, Operating System and Application Server Provisioning

As with SQL Azure, when deploying the business application (for StockTrader 5, the Web tier, and the service tiers), there is no need to buy, setup and maintain any hardware or operating system software. OS maintenance is automatic as well, with security patching, service packs and the like automatically performed in the data center, with no application downtime.

Geographically Disperse Deployments

Today, there are six different Azure data centers across the world. Deploying a service or application to another region, or multiple regions, is simple point and click. For example, the StockTrader Web application might run in North America, but the Business Service tier and Order Processor tier could be deployed to Europe or Asia. This might make sense, for example, in hybrid scenarios to move the application execution closer to the on-premise resources in hybrid cloud scenarios, to reduce latency. When running on Windows Azure your application is automatically leveraging one of the largest, fastest international networks in the world—the same network that runs Hotmail, MSN, Xbox Live, Windows Live and the like. So you are getting an enormous Internet pipe. In addition, although the StockTrader 5 sample does not utilize it, Azure includes a high-performance Content Delivery Network (CDN). Web content, large download files (such a video/media, .pdf files, etc.) are automatically front-end cached across the world for very fast local access by users. This alone can save an organization a tremendous amount of money versus maintaining their own geographically disperse source networks and caching.

StockTrader and Hybrid Cloud Scenarios

The new application runs with one code base on-premise (Windows Server/SQL Server) and in the cloud (Windows Azure/SQL Azure). As such, it presents a good opportunity to highlight hybrid cloud scenarios: some tiers of the application run on-premise, while others run in the cloud. Any combination of on-premise services and cloud services is possible with the application, based on its service-oriented design. Several key hybrid scenarios are illustrated, however.

- **Cloud-based Web user interface to on-premise services and on-premise SQL Server**. This is relevant for organizations that want to maintain their databases on-premise within their own data centers, yet build and deploy cloud-based applications that use the private-cloud database, with the front-end application(s) available on the Internet via Windows Azure.
- **Internal applications accessing the business services and transaction processing in the cloud via Windows Azure and SQL Azure**. For StockTrader 5, the Web application and/or the desktop WPF smart client application can run within the corporate network, behind the corporate firewall; yet access the Business Service tier and hence Order Processor service which are running in the Azure cloud. No code changes are required. Any number of different applications, deployed anywhere (from corporate networks to mobile devices), can leverage the service tiers of StockTrader 5, with proper secure authentication.
- **Internal applications accessing SQL Azure as a *cloud-based relational database service***. During the installation of StockTrader 5.0, users are asked for the name of a SQL Server database that will host the StockTrader business database and the configuration repositories for the services. Users can either point to an on-premise SQL Server instance (including SQL Express); or they can point to a SQL Azure database server. The code, again, is exactly the same, but when choosing a SQL Azure database, the local applications utilize SQL Azure databases through the Internet, via secure connections. In this way, no database software, setup and maintenance is necessary; yet the application logic remains internal and only accessible by users within the corporate network.

In the cases above, specific design choices were made regarding the WCF technologies implemented and the associated WCF security model implemented in the application. However, the Windows Azure platform presents many different choices to cover a wide range of customer scenarios: from Windows Azure Connect (VPN-based integration) to Azure App Fabric Access Control and the App Fabric Service Bus, to secure Web Services via WCF.

Cloud-based Web user interface to on-premise services and on-premise SQL Server

When running in the cloud, the existing StockTrader Web interface is utilizing a SQL Azure database for authentication—this database stores registered user information, and encrypted (salted hash) passwords these users setup when registering. The service tiers in StockTrader 5 are currently designed for exclusive access from the Azure StockTrader Web tier. To secure the services in this scenario, we implemented WCF security at both the transport and message level. This requires no extra code based on the WCF programming model, but does involve proper configuration. Namely, we setup a binding using *Transport With Message Credentials* security. This means communication is encrypted at the transport level—for example https/SSL or TLS over TCP. It also means client credentials are required before the service will accept requests.

StockTrader 5 can leverage both username credentials or certificate client credentials with mutual certificate exchange via X.509 digital certificates. When running with certificate client credentials, there is only one certificate in the world that the business service tier will accept—the one we created for the StockTrader Web application. This is an X.509 certificate (password protected, 2048 bit encryption) that the service must receive from the client channel, or the service operation request will be automatically rejected. The same is true of the Order Processor Service. Currently, only the Business Service Tier can gain access and utilize this service. Username credentials can also be used, of course, and in fact StockTrader 5 as a sample ships with this configuration also. In either case, other clients can be enabled for access using either X.509 certificates or any authentication mechanism utilizing a username/password combination.

For our hybrid scenario, where the Web application is running in the cloud, we just extended this same mechanism to our private cloud data center. This data center runs Forefront Threat Management Gateway (TMG), and is secured as is any typical corporate data center with firewalls and DMZs. Our SQL Server database runs behind the firewall, within the secure private network, as does the Order Processor Service, which also uses this database.

We are utilizing Microsoft Forefront to publish a single https-based endpoint for the StockTrader 5 Business Service tier, which runs also on our private, protected network. Forefront is automatically routing requests to a cluster of load balanced business service tiers running across multiple physical machines and multiple Hyper-V virtual machines hosted on these servers. This provides scale out within this private cloud environment. To securely integrate with these on premise resources, we can simply route the Azure StockTrader Web tier (ASP.NET Web application) to a different endpoint---this time our Forefront TMG published endpoint. All communication is via https, and again using client credentials (either certificate or username, depending on how we configure it). With WCF, there is complete separation of the programming logic from the network binding/security model used. You can also host a service that simultaneously publishes multiple endpoints, perhaps on different networks—for example, we might be using different security for internal users vs. those coming in from the Internet and the Azure-hosted public Web application.

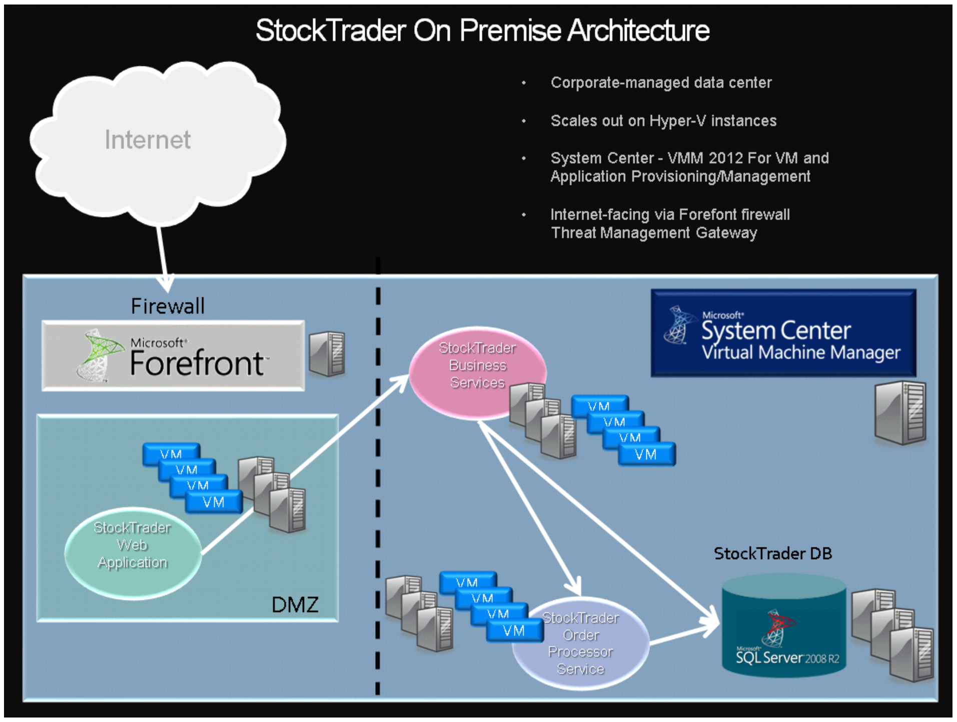

StockTrader On Premise Architecture
Internet
Corporate-managed data center
Scales out on Hyper-V instances
System Center - VMM 2012 For VM and Application Provisioning/Management
Internet-facing via Forefront firewall Threat Management Gateway
Firewall
Microsoft Forefront
Microsoft System Center Virtual Machine Manager
StockTrader Business Services
VM
StockTrader Web Application
DMZ
StockTrader DB
StockTrader Order Processor Service
SQL Server 2008 R2

Figure 2. On-Premise services and private SQL Server 2008 R2 Database, virtualized on Hyper-V.

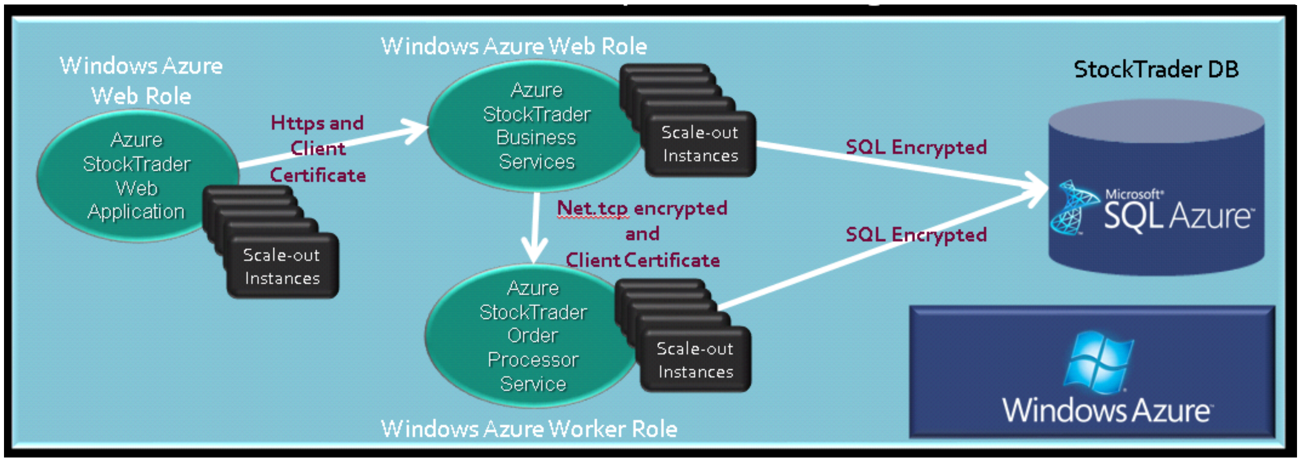

Windows Azure Web Role
Azure StockTrader Web Application
Scale-out Instances
Https and Client Certificate
Windows Azure Web Role
Azure StockTrader Business Services
Scale-out Instances
Net.tcp encrypted and Client Certificate
Azure StockTrader Order Processor Service
Scale-out Instances
Windows Azure Worker Role
SQL Encrypted
SQL Encrypted
StockTrader DB
Microsoft SQL Azure
Windows Azure

Figure 3. The Azure implementation with all tiers running in the public cloud.

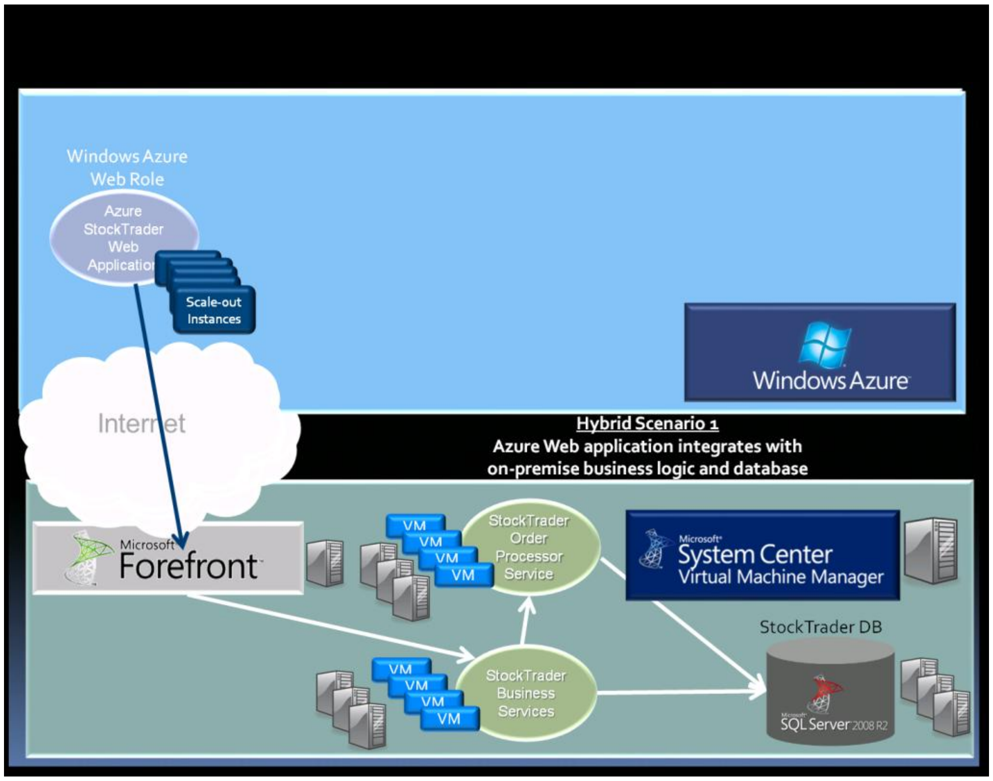

Windows Azure Web Role
Azure StockTrader Web Application
Scale-out Instances
Windows Azure
Internet
Hybrid Scenario 1
Azure Web application integrates with on-premise business logic and database
Microsoft Forefront
VM
StockTrader Order Processor Service
Microsoft System Center Virtual Machine Manager
StockTrader DB
StockTrader Business Services
SQL Server 2008 R2

Figure 4. Hyrbrid public-private cloud. Communication from the Web is over secure sockets (ssl), and authentication is either via a client X.509 certificate, or username credentials coming in from the Web application.

Internal application to Azure-hosted services and SQL Azure

This scenario is the opposite of the previous scenario.

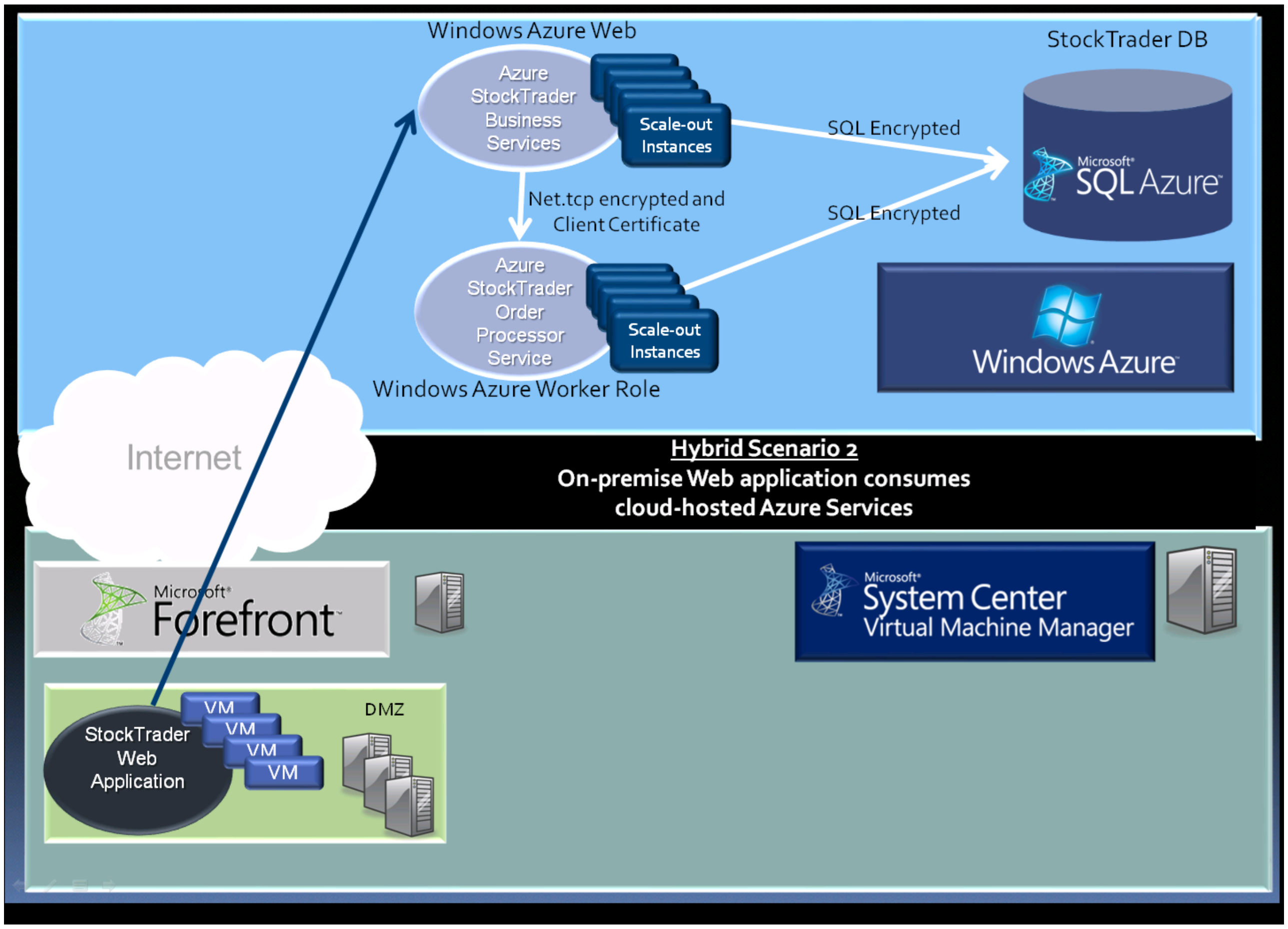

Windows Azure Web
StockTrader DB
Azure StockTrader Business Services
Scale-out Instances
SQL Encrypted
Microsoft SQL Azure
Net.tcp encrypted and Client Certificate
SQL Encrypted
Azure StockTrader Order Processor Service
Scale-out Instances
Windows Azure
Windows Azure Worker Role
Internet
Hybrid Scenario 2
On-premise Web application consumes cloud-hosted Azure Services
Microsoft Forefront
Microsoft System Center Virtual Machine Manager
DMZ
StockTrader Web Application
VM
VM
VM
VM

Figure 5. The reverse scenario, with the client application running on-premise, accessing cloud-services running on Windows Azure/SQL Azure. Note this picture shows the internal Web application accessing Azure-hosted services. The StockTrader Smart Client desktop application can also easily be configured for this scenario.

Technologies Incorporated into StockTrader 5

The following technologies are illustrated in the StockTrader 5 benchmark application:

- Windows Azure Web and Worker Roles
- SQL Azure (including high-performance development pattern via ADO.NET 4.0)
- Azure AppFabric Distributed Caching
- .NET Framework 4.0 including and ADO.NET, ASP.NET and WCF
- Interoperability between .NET and commercial Java application servers based on WCF and industry-standard Web Services, including WS-* Advanced Web Services.
- Database connectivity to both SQL Server and SQL Azure with a single code base.
- Implementing high-performance ASP.NET Web applications with a logical n-tier, service-oriented enterprise design pattern.
- Implementing high performance WCF services.

- Implementing multiple service bindings to support different network transports, message encoding formats, and security using WCF.
- Scale-out across Azure instances or on-premise virtualized environments, again with a single code base.

StockTrader 5 Configuration Overview

There are four main components to the StockTrader 5 application, which is a service-oriented, composite application. These components include:

1. The StockTrader Web Application user interface
2. The StockTrader Business Services
3. The StockTrader Order Processor Service
4. The StockTrader backing database

The application can be configured to run the Web application and the two primary services (Business Services and Order Processing) within a single monolithic application (a single process hosts all three elements), or to remotely activate the Business Services and/or the Order Processor Service based on the use of Windows Communication Foundation. The configuration is managed through a Configuration Management Service with backing SQL Server configuration repositories for each element of the application. While the StockTrader 5 uses the configuration management system and the source code for this system is included with the StockTrader 5 download, the configuration management system itself is separate from the application, and is designed to be re-usable across any application that might benefit from implementing it. This paper focuses on the core StockTrader 5 application; see the separate whitepaper entitled *Guide to Implementing the Configuration Service 5.0 for .NET Applications and Services* for a more thorough technical overview of the configuration management service itself, beyond the shorter overview provided below.

Brief Overview of the StockTrader 5 Configuration Management Service

As an end-to-end sample application, StockTrader 5 is rich with different configuration settings and deployment options. For example, the application can be configured to run services in-process, or to remotely access services using WCF. Once services are activated remotely, however, a best-practice design is to ensure they remain 'black-boxes' and completely autonomous from any applications or other services that may use them. Just because the StockTrader Business Services are used by the StockTrader Web application, for example, does not mean they will not be used by other applications as well. Also, these remote services might be hosted in the cloud, deployed in different data centers/geographic locations, deployed on-premise, or any hybrid combination.

Each of these cases presents challenges in how to centrally manage and configure a distributed, composite application made up of different autonomous and remotely activated services. Such challenges are introduced based on the distributed nature of this application. The configuration management implementation is designed to overcome these challenges, and ensure the application not only performs well, but also can be more easily managed, including across hybrid cloud environments.

StockTrader 5 uses two distinct services that can be remotely activated:

- The Business Services, which the Web application calls
- The Order Processing Service, which is called in turn by the Business Service layer (when configured for asynchronous order processing)

By abandoning assumptions about where services are located, where they are hosted or how they are hosted, greater flexibility is achieved in terms of supporting interoperability between services and different possible physical deployment topologies. So a key to understanding StockTrader 5 is to understand that all three elements (Web Application, Business Services, Order Processor Service) are designed to be autonomous. The StockTrader 5 Web application uses and relies on Business Services, but has no awareness of the implementation details of the Business Services layer itself—such as its (optional) use of a separate Order Processing Service—or how it manages/uses its own configuration settings.

So configuring such a distributed system, where elements are running on different servers, possibly not even in the same data center, developed by different teams, possibly clustered across replicated servers, would be much more difficult without some way to centrally manage the configuration of the remote services. At the same time, we want to ensure each service remains autonomous; hence **each service must be responsible for its own configuration**. The IT Administrator, however, should have a way to view and alter the overall composite application configuration in an integrated way. This is achieved via the Configuration Management Service, a re-usable system implemented in shared libraries and based on WCF for configuration exchanges between services and clustered nodes.

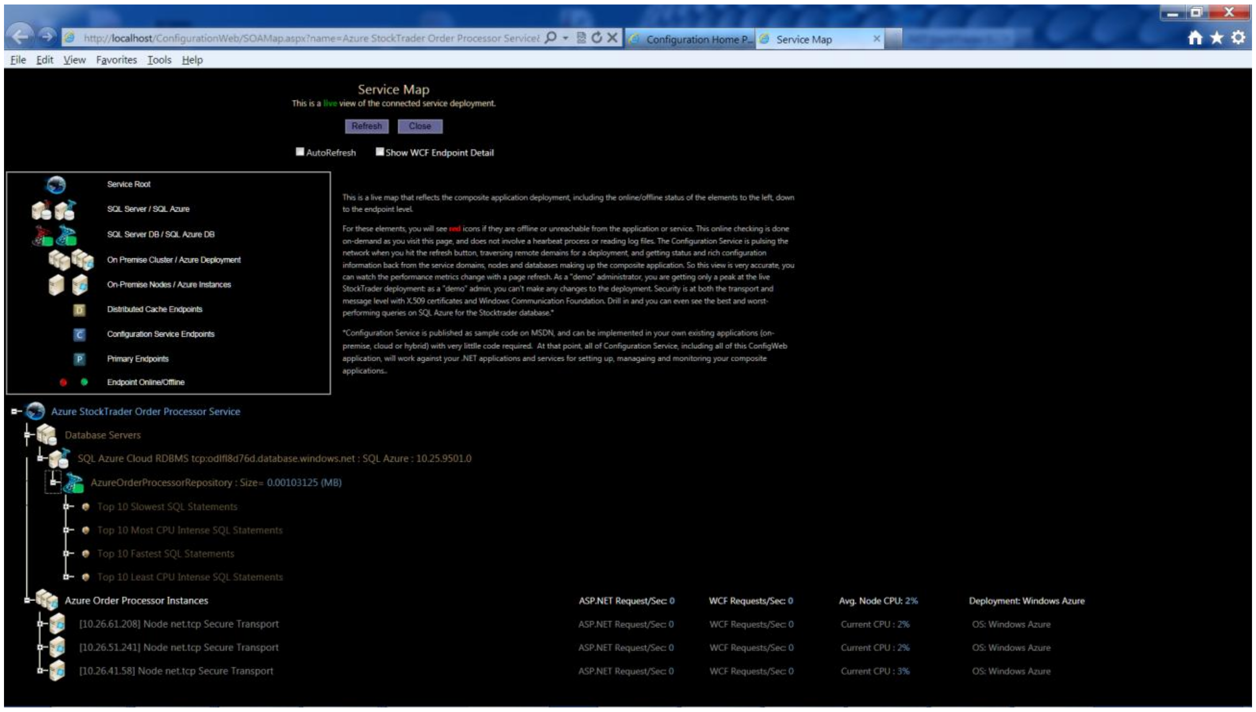
With StockTrader 5, the configuration system is accessed and used simply by logging into the ConfigWeb application. ConfigWeb is an ASP.NET Web application that presents a way to centrally view, manage and configure the overall system via the Configuration Service. These pages are generic: they can work with any application that implements the configuration service—they are not StockTrader-specific. ConfigWeb is also being hosted on Windows Azure, and developers can use it to browse the live StockTrader 5 deployment on Azure and its configuration as a pseudo-admin (no changes allowed as it's a live application).

Dynamic Clustering

One idea incorporated into the sample application is the idea of ***virtualizing*** service endpoints across any number of clustered nodes. This neatly applies to achieving seamless scale-out across Azure instances, or Hyper-V on-premise virtualized environments. This is also a core concept implemented in the Configuration Management System. Instead of manually/statically setting up and configuring clusters as with most application servers, StockTrader 5 services are automatically virtualized in that the administrator can simply start up new instances on different servers (or add Azure instances), and still centrally manage configuration data. This is possible because each service node points to the same central configuration repository—which is also its cluster management system. Hence, Business Services, for example, is 'clustered' for load balancing/scalability and application level failover rather easily: simply start the service on two different machines or add another Azure instance.

Health Monitoring

The ASP.NET configuration menu pages within the StockTrader sample application allows the administrator to view the various servers participating in a deployment, and see which servers are online or offline, as well as view endpoint status for service endpoints exposed by the host. New with StockTrader 5, the administrator can also view Database information in the Service Map page.


http://localhost/ConfigurationWeb/SOAMap.aspx?name=Azure StockTrader Order Processor Service
Configuration Home P...
Service Map
File Edit View Favorites Tools Help
Refresh
Close
AutoRefresh
Show WCF Endpoint Detail
Service Root
SQL Server / SQL Azure
SQL Server DB / SQL Azure DB
On Premise Cluster / Azure Deployment
On-Premise Nodes / Azure Instances
Distributed Cache Endpoints
Configuration Service Endpoints
Primary Endpoints
Endpoint Online/Offline

Service Map

This is a live view of the connected service deployment.

This is a live map that reflects the composite application deployment, including the online/offline status of the elements to the left, down to the endpoint level.

For these elements, you will see red icons if they are offline or unreachable from the application or service. This online checking is done on-demand as you visit this page, and does not involve a hearbeat process or reading log files. The Configuration Service is pulsing the network when you hit the refresh button, traversing remote demains for a deployment, and getting status and rich configuration information back from the service domains, nodes and databases making up the composite application. So this view is very accurate, you can watch the performance metrics change with a page refresh. As a "demo" administrator, you are getting only a peak at the live StockTrader deployment: as a "demo" admin, you can't make any changes to the deployment. Security is at both the transport and message level with X.509 certificates and Windows Communication Foundation. Drill in and you can even see the best and worst-performing queries on SQL Azure for the Stocktrader database.*

*Configuration Service is published as sample code on MSDN, and can be implemented in your own existing applications (on-premise, cloud or hybrid) with very little code required. At that point, all of Configuration Service, including all of this ConfigWeb application, will work against your .NET applications and services for setting up, managaing and monitoring your composite applications..

```
Azure StockTrader Order Processor Service
  Database Servers
    SQL Azure Cloud RDBMS tcp:odlfl8d76d.database.windows.net : SQL Azure : 10.25.9501.0
      AzureOrderProcessorRepository : Size= 0.00103125 (MB)
        Top 10 Slowest SQL Statements
        Top 10 Most CPU Intense SQL Statements
        Top 10 Fastest SQL Statements
        Top 10 Least CPU Intense SQL Statements
  Azure Order Processor Instances                       ASP.NET Request/Sec: 0   WCF Requests/Sec: 0   Avg. Node CPU: 2%   Deployment: Windows Azure
    [10.26.61.208] Node net.tcp Secure Transport        ASP.NET Request/Sec: 0   WCF Requests/Sec: 0   Current CPU : 2%    OS: Windows Azure
    [10.26.51.241] Node net.tcp Secure Transport        ASP.NET Request/Sec: 0   WCF Requests/Sec: 0   Current CPU : 2%    OS: Windows Azure
    [10.26.41.58] Node net.tcp Secure Transport         ASP.NET Request/Sec: 0   WCF Requests/Sec: 0   Current CPU : 3%    OS: Windows Azure
```

Figure 6. The Configuration Service Service Map View.

StockTrader 5 Application Design

The core StockTrader design is based on an enterprise, n-tier design pattern with full logical separation of the application into three distinct layers:

1. Web Application Layer (UI), utilizing ASP.NET and Web Forms
2. Middle layer Business Services Layer (BSL), utilizing stateless C# classes
3. Data Access Layer (DAL), utilizing ADO.NET and stateless C# classes

In addition, C# classes are used to model data records as mapped from the RDBMS database tables. The model classes are passed between all layers in the application allowing complete separation of the database implementation from both the business services and user interface layers.

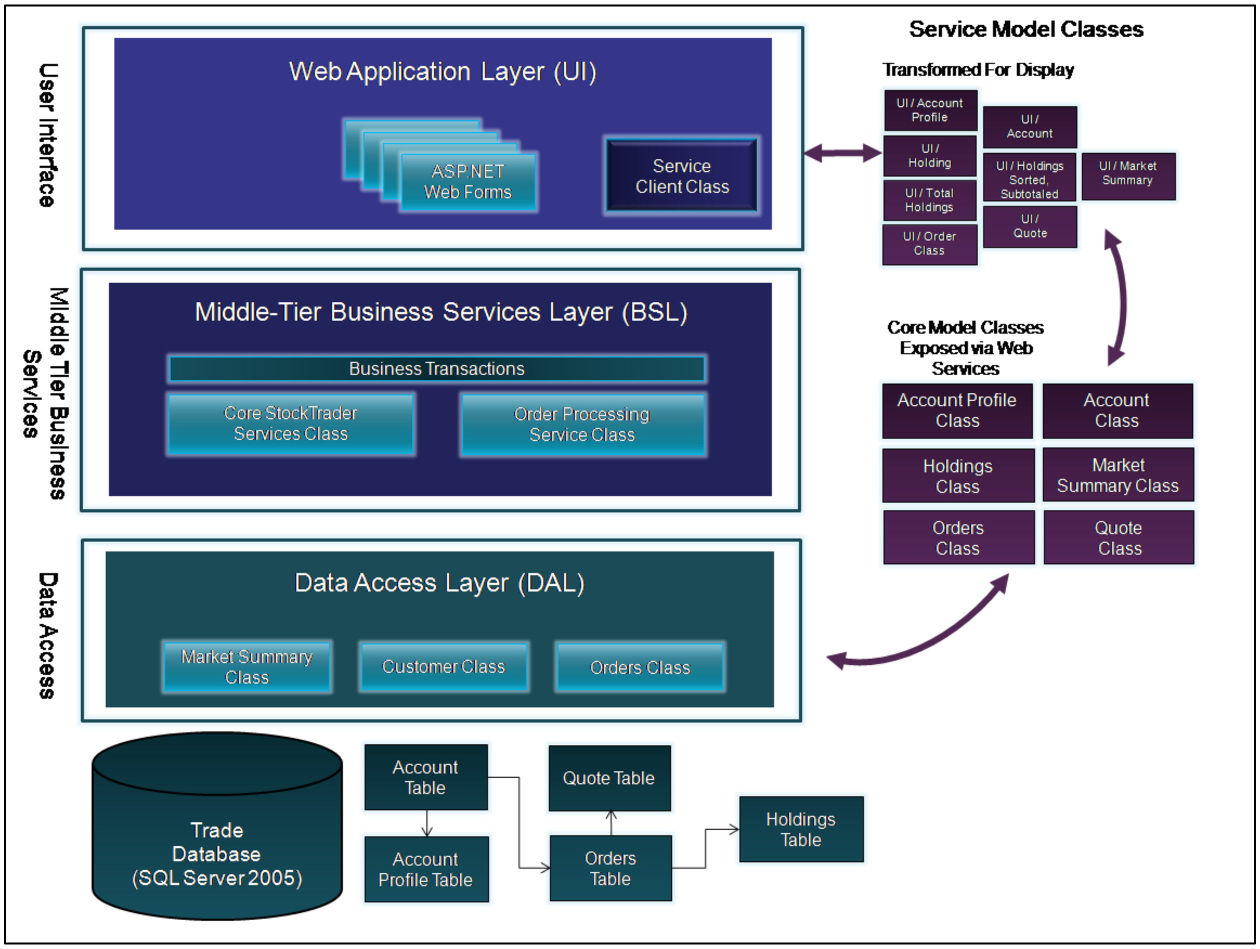


Figure 7: StockTrader 5 Logical n-tier Design

Based on this logical layering, several advantages are realized:

- The implementation logic of the business services can be changed without making any changes to the other layers.
- Database-specific implementation details are hidden to the Web UI Layer and the Business Services Layer. Hence, additional data access assemblies (DAL classes) can easily be added to communicate with other types of databases (e.g. DB2, Oracle). This can be done without making any changes to the Web/UI layer or the Business Services Layer.
- Services can be co-located or remotely invoked by the Web layer; enabling more flexible deployment options. For example, The ASP.NET Web servers do not need to have any database connectivity or access, since they only invoke methods in the Business Services Layer.
- The design is horizontally scalable out of the box across load-balanced server clusters using the built-in StockTrader Configuration System[2], Windows Azure, or hardware load-balancing techniques.
- Different programmers can more easily work as team, each focusing on a specific layer.

[2] See the separate paper *Using the Configuration Service 5.0 Visual Studio Template.*

StockTrader 5 Configuration Options

The application can be easily configured to use different .NET technologies which are incorporated into the design. Because all StockTrader 5.0 tiers implement the Configuration Service, the application can be re-configured easily via ConfigWeb. Detailed guidance for changing the remote modes and security between tiers is documented in the separate paper titled *.NET StockTrader 5.0 Installation and Configuration*. This document has step by step instructions.

The StockTrader 5 application has two primary settings that relate to configuration of different remote (or non-remote) modes, and the security applied to the service endpoints: **Access Mode** and **Order Mode**. The Access Mode setting determines how the StockTrader 5 Web application invokes the backend Business Services tier. The Order Mode setting determines whether orders are processed in-process/synchronously or remotely/asynchronously, and over which messaging format and specific transport protocol. These settings also help automate the configuration with respect to which host to communicate with, for example, the on-premise Business Services; or the Azure Business Services. The basic settings, already configured in the service configuration repositories for each tier are:

StockTrader 5 Sample Access Mode Settings

Setting	Description
In Process Activation	The Web application invokes the business service layer via a direct, in-memory mode with no remote service calls. In this setting, the Web application and business service layer run as a single, standalone ASP.NET Web application and the Web Layer and Business Service Layer cannot be divided across a network.
Http Web Service	The Web application invokes the service layer via WCF Web Services hosted in either Web or Worker roles. It uses a wsHttp2007 binding and Text-XML encoding. In this setting, the Web layer and Service layer can be distributed across a network, and any Web Service client (.NET or J2EE) can utilize the backend StockTrader WCF services.
Net.tcp Web Service	The Web application invokes the service layer via WCF Web Services hosted in a Web or Worker Azure Role. It uses the TCP/IP transport and binary encoding, and is another example of hosting services with WCF.
Net.tcp /Http Web Service with Transport Security	This includes either SSL/Https for message integrity, as well as TLS/TCP. All communication is encrypted.
Net.tcp /Http Web Service with Transport Security with Message Credentials, using User Name client credentials	This is an optional mode that uses X.509 digital certificates with username client credentials to secure the Business Service layer from unauthorized access. To configure this mode, read the document *StockTrader 5 Installation and Configuration,* as by default the Business Service will not be configured for this mode for on-premise installations. The Azure projects are configured by default to use this secure mode.

Net.tcp /Http Web Service with Transport Security with Message Credentials, using X.509 Client Certificate credentials	This is an optional mode that uses X.509 digital certificates on the service, as well as client-side certificate that is required for authentication to the service operations. To configure this mode, read the document *StockTrader 5 Installation and Configuration.*

StockTrader 5 Order Mode Settings

Setting	Description
In Process Activation	Orders are processed synchronously by the order processing service, and the Order Processing Service is run in-process with Business Services---so no remote calls are made for order processing.
Asynchronous via MSMQ / Durable Queue	Orders are placed asynchronously via WCF with an MSMQ binding. The WCF service is fully transacted, with orders placed into a transacted, durable message queue with assured message delivery. The StockTrader order placement service invokes a separate order processing service via WCF. In this mode, the order processing service can be distributed across a network and run on a separate, dedicated application server/server cluster from the main BSL layer. This is an example of loose coupling, since the order processing service does not have to be online for users to actually place trade orders (stock buys and sells). When the order processing service is brought online, it will automatically process any orders in the queue. Additionally, since the order processing service is based on WCF, the developer does not program any MSMQ-specific logic; rather they simply program to a standard WCF service, and use an MSMQ binding for that service—WCF takes care of all the rest. The *StockTrader 5 Installation and Configuration* document has step-by-step instructions for re-configuring to this mode.
Asynchronous via Net.tcp or Http	Orders are placed asynchronously via a WCF with a TCP/IP or Http transport binding. Order processing is asynchronous and the order processing service can be distributed across the network and run on a separate, dedicated application server/cluster.
Net.tcp /Http Web Service with Transport Security with Message Credentials, using User Name client credentials	This is an optional mode that uses X.509 digital certificates with username client credentials to secure the Order Processor Service layer from unauthorized access. To configure this mode, read the document *StockTrader 5 Installation and Configuration,* as by default the Order Processor Service will not be configured for this mode for on-premise installations.

	The Azure projects are configured by default to use this secure mode.
Net.tcp /Http Web Service with Transport Security with Message Credentials, using X.509 Client Certificate credentials	This is an optional mode that uses X.509 digital certificates on the service, as well as client-side certificate that is required for authentication to the service operations. To configure this mode, read the document *StockTrader 5 Installation and Configuration.*

Conclusion

The StockTrader 5 sample application explores several different alternative architectures for a service-oriented application that can be deployed on-premise or on Windows Azure. It illustrates the flexibility of .NET to support many different deployment topologies. The application also illustrates many best-practice programming practices for .NET and WCF including the use of an n-tier, service-oriented design. Please refer to the benchmarking whitepaper on the StockTrader MSDN site for performance/throughput benchmark comparisons based on the application and the various modes it supports.