# Walkthrough Azure

## Ziel

StockTicker hosten

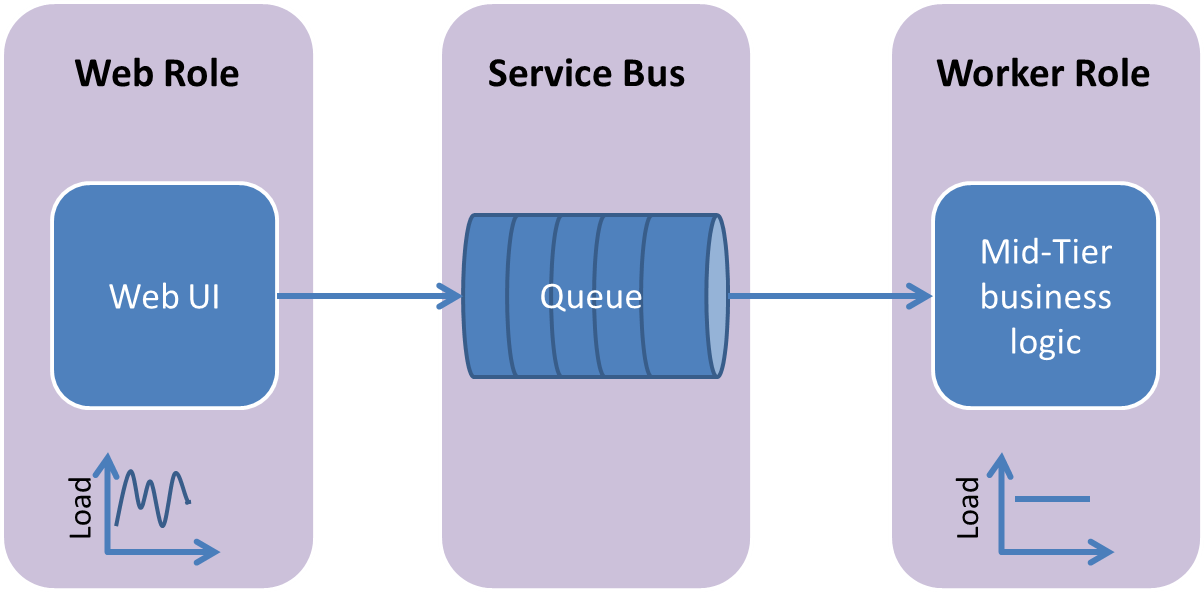
MultiTier Apllication using ServiceBus Queues

## Scenario Overview: Inter-Role Communication

To submit an order for processing, the front end UI component, running in the web role, needs to interact with the middle tier logic running in the worker role. This example uses Service Bus brokered messaging for the communication between the tiers.

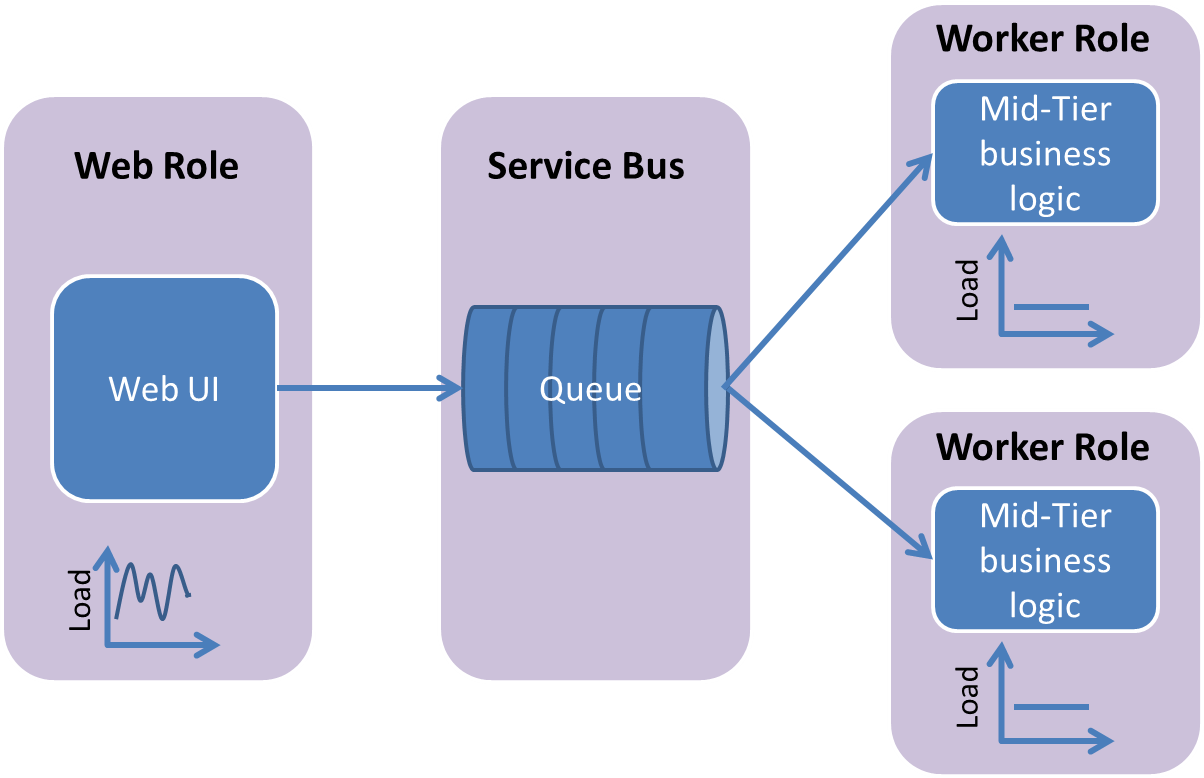
Using brokered messaging between the web and middle tiers decouples the two components. In contrast to direct messaging (that is, TCP or HTTP), the web tier does not connect to the middle tier directly; instead it pushes units of work, as messages, into the Service Bus, which reliably retains them until the middle tier is ready to consume and process them.

The Service Bus provides two entities to support brokered messaging, queues and topics. With queues, each message sent to the queue is consumed by a single receiver. Topics support the publish/subscribe pattern in which each published message is made available to each subscription registered with the topic. Each subscription logically maintains its own queue of messages. Subscriptions can also be configured with filter rules that restrict the set of messages passed to the subscription queue to those that match the filter. This example uses Service Bus queues.



This communication mechanism has several advantages over direct messaging, namely:

* **Temporal decoupling.** With the asynchronous messaging pattern, producers and consumers need not be online at the same time. Service Busreliably stores messages until the consuming party is ready to receive them. This allows the components of the distributed application to be disconnected, either voluntarily, for example, for maintenance, or due to a component crash, without impacting the system as a whole. Furthermore, the consuming application may only need to come online during certain times of the day.
* **Load leveling**. In many applications, system load varies over time whereas the processing time required for each unit of work is typically constant. Intermediating message producers and consumers with a queue means that the consuming application (the worker) only needs to be provisioned to accommodate average load rather than peak load. The depth of the queue will grow and contract as the incoming load varies. This directly saves money in terms of the amount of infrastructure required to service the application load.
* **Load balancing.** As load increases, more worker processes can be added to read from the queue. Each message is processed by only one of the worker processes. Furthermore, this pull-based load balancing allows for optimum utilization of the worker machines even if the worker machines differ in terms of processing power as they will pull messages at their own maximum rate. This pattern is often termed the competing consumer pattern.



CI mit Teamcity

Anforderung: Deploy early, deploy often!

* Continuous Deployment mittels scripts von TeamCity aus

WebRole für ASP.NET MVC Anwendung

WorkerRole für WCF Service

TableStorage für Datenhaltung

News von WebService (comdirect, swissquote), provided by WorkerRole (WebRole unabhängig von NewsProvider)

Userverwaltung OAuth via MVC 4

Persistenz

* Portfolios
* Alerts

## Preconditions

* VisualStudio 2012
* ASP.NET MVC 4
* Web Platform Installer 4.0
* Optional: ReSharper 7
* GitHub repository
  + Be a collaborator of repository!
  + Clone repo
    - Copy address of repo
    - Git Extensions🡪 GitHub🡪Fork/Clone Repository🡪Search for repo 🡪Clone
    - Pull with AutoStash!
  + Test push to repo
* TeamCity Project anlegen
  + Git support
    - Git Extensions mit MSysGit installieren auf BuildServer
  + NuGet support
    - Nuget step for package management
  + Azure support
    - Install Azure Authoring Tools <https://www.windowsazure.com/en-us/develop/net/common-tasks/continuous-delivery/>
    - Copy C:\Program Files (x86)\MSBuild\Microsoft\VisualStudio\v10.0\WebApplications and Windows Azure Tools(v11.0 for Visual Studio 2012) to BuildServer

## Install Azure SDK

1. Website <http://www.windowsazure.com/en-us/develop/net/>

## Create MVC 4 WebRole

1. Start DevStudio with AdminRights
2. New Project -> Cloud -> .net framework4 -> Windows Azure Cloud Service -> StockTickerMeetsAzure -> MVC4 WebRole -> umbenennen ->ok
3. F5->ok
4. Set Solution to manage NuGet packages automatically
5. Add source files to git
   1. exclude packages, obj, bin, \*.user, \*.suo, \*.log
   2. include packages\repositories.config

## Initial Deployment

Build a Package using MSBuild Commands

1. Right click on role
2. Publish…
3. The first time you publish, you have to download credentials via the provided link.
4. Click Sign in to download credentials
5. Sign in with LiveID
6. Save download as file to desired directory
7. Use “Import” Button to browse to downloaded publish settings file (don’t use Chrome)
8. Choose subscription
9. Create cloud service with name of application and select location near to you
10. Accept common settings with a click to next…
11. Click Publish

## Continuous Deployment

Beware: Use only lowercase letters in azure names in powershell commands!

1. TeamCity BuildServer
   * 1. Install PowerShell cmdlets <https://www.windowsazure.com/en-us/manage/downloads/>
     2. Run "C:\Install\WindowsAzurePowerShell.3f.3f.3fnew.exe"
2. Start Windows Azure PowerShell using the Start menu (with Admin Rights). If you start in this way, the Windows Azure PowerShell cmdlets will be loaded.
3. At the PowerShell prompt, verify that the PowerShell cmdlets are loaded by typing the partial command Get-Azure and then pressing tab for statement completion.   
   You should see various Windows Azure PowerShell commands.
4. Verify that you can execute powershell scripts by   
   Set-ExecutionPolicy unrestricted (needs Admin Rights)
5. Verify that you can connect to your Windows Azure subscription by importing your subscription information from the .publishsettings file. (filename without blanks!)
   1. Import-AzurePublishSettingsFile c:\scripts\WindowsAzure\default.publishsettings
   2. Then give the command  
      Get-AzureSubscription
   3. This will display information about your subscription. Verify that everything is correct.
6. When using RemoteDesktop in Solution, set your certificate from solution file ServiceConfiguration.Cloud.cscfg thumbprint:  
   Add-AzureCertificate -serviceName stocktickermeetsazure -certToDeploy (get-item cert:\CurrentUser\MY\<THUMBPRINT from ServiceConfiguration.Cloud.cscfg >)