

Azure SQL Database – Vertical Scaling

Level-300 Demonstration

Script

Version 1.0

08/21/2015

Prepared for: Darmadi Komo

Prepared by: Prowess Consulting

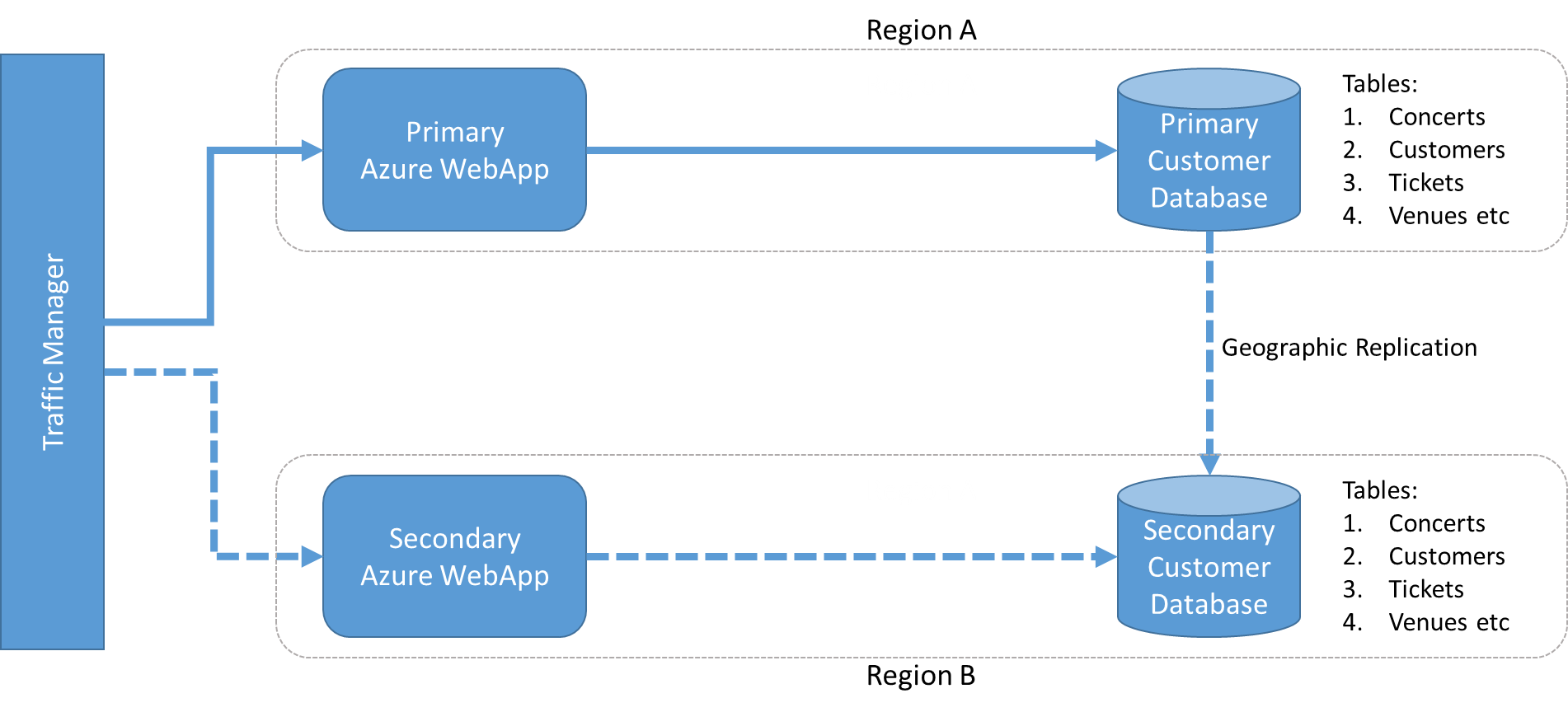
Overview

This demonstration explores vertical scaling in Microsoft Azure SQL Database in the context of a software-as-a-service (SaaS) provider, Wingtip Tickets, which provides ticketing software to artists and groups. This demonstration centers on the tenant Julie and the Plantes (a fictitious pop-music tenant).

Other Tenants that will be discussed in future labs, will include the following:

* The Archie Boyle Band (a fictitious rock-music tenant)
* Walla Walla Symphony (a fictitious classical-music tenant)

## Demo Architecture



**Figure 1** Overall architecture of the demo components

|  |  |  |
| --- | --- | --- |
| Screen | Click Steps | Demo Script |
|  | **Section 0: Lab Setup** |  |
|  | Because this demonstration uses the Azure public cloud, all server names must be unique. To help ensure this, the New-WTTEnvironment PowerShell script that you used to configure this demo automatically generated unique server names based on the environment name that you supplied during configuration. This demo script uses the following server names, to illustrate an example where the environment was setup as follows:  New-WTTEnvironment –WTTEnvironmentApplicationName mbjulieandtheplantes   * *mbjulieandtheplantesprimary.database.windows.net* (Primary Database Server) * *mbjulieandtheplantessecondary.database.windows.net* (Secondary Database Server) * *mbjulieandtheplantesprimary.azurewebsites.net* (Primary WebApp) * *mbjulieandtheplantessecondary.azurewebsites.net* (Secondary WebApp) * [*mbjulieandtheplantes.trafficmanager.net*](http://mbchristophanhalt.trafficmanager.net/)(Traffic Manager Profile) * *mbjulieandtheplantes* (Azure Storage Account that contains: Audit Events Table) * *mbjulieandtheplantes* (Resource Group)   Your particular server names will have your initials prepended to them. For example, *Primary WebApp* might be *ghjulieandtheplantesprimary.azurewebsites.net*. |  |
|  | If this is your first time setting up the environment, skip steps 1 and 2.   1. In **portal.azure.com**, make sure the *Primary Database Server/Customer1* database is set to **Basic SKU** and **Geo Replication** is disabled. (In this example, it is *mbjulieandtheplantesprimary.database.windows.net/Customer1*.) 2. Makes sure Customer1 DB doesn’t exist on the *Secondary Database Server*. |  |

|  |  |  |
| --- | --- | --- |
|  | **Section 1: Vertical Scaling** |  |
|  | One of the benefits of Azure SQL Database is the ability to scale up and down as needed, with zero impact to uptime.  Julie and the Plantes is anticipating sell-out crowds for upcoming events, and the website will require increased capacity to handle demand. In this demonstration we will:   * Simulate the usage load using the load generator and show the spike in resource utilization with the current size of the database instance. * See how easy it is to scale-up by changing the size of an instance (in this case, from Basic to S0) and how quickly the change is implemented. * Re-run the load generator, and show how the increased capacity has significantly improved the sites ability to handle a spike in ticket sales.   One of the benefits of Azure SQL Database is the ability to scale up and down as needed, with zero impact to uptime. | |
|  | To demonstrate a task where an event has just gone on sale and many users purchase tickets at the same time, we have a load generator that simulates this, so as to create a spike.   1. To start the Load Generator, open the LoadGenerator folder and double-click **LoadGenerator.exe**. 2. Insert the Primary Database Server. (In this example *mbjulieandtheplantesprimary.database.windows.net*.) | To demonstrate a scenario where an event has just gone on sale, and many users are purchasing tickets at the same time, we will use a load generator to create a usage spike. |
|  | 1. Select the first **Event** from the list which will correspond to ConcertId 1, then select **Ticket Level**, and **Customer**. 2. For the purposes of this lab, the same event used in the previous section was chosen. 3. In the **Tickets To Purchase** field, enter **10000**. 4. In the **Bulk Purchase** field, enter **80**. 5. Click **Start**. | As you can see, we just simulated a real-world task where 10,000 tickets were purchased, and the Database was able to handle a rate of around 365 tickets per second. |
|  | 1. Switch to the Azure Portal, expand the SQL Databases blade, and select **Customer1** on the *Primary Database Server* (in this example *mbjulieandtheplantesprimary.database.windows.net*.) | Within a minute or so, we should see a spike in the Resource Utilization chart. CPU usage spiked over 80% percent which would obviously cause a problem if there were multiple events that went on sale at the same time.  This database was initially configured using the Basic pricing tier, which we just saw was not enough capacity to handle an even larger spike, or multiple concurrent events.  Basic would not be used for a real-world ticket purchasing website, but is used in this lab scenario primarily to make it easy to show a spike in CPU utilization and to demonstrate how easy it is to scale up on demand. |
|  | 1. In the **Customer1 Database** summary pane, scroll down to Pricing tier. 2. Click **Pricing tier**. | Azure SQL Database makes it easy to scale up as needed with the click of a button. |
|  | 1. Click **S0 Standard**. 2. Click **Select**. | Within a few seconds, the Basic pricing tier is upgraded to a S0, and the allotted capacity has effectively doubled from 5 database throughput units (DTUs) to 10 DTUs. DTUs are based on a simple, blended measure of CPU, memory, reads, and writes, and they provide a convenient means of describing the relative capacity of the performance levels of different Azure SQL Database SKU pricing tiers.  Now that we’ve upgraded the capacity, let’s see how our application scales with the same load. |
|  | 1. Return to the Load Generator. 2. Insert the Primary Database Server (In this example, *mbjulieandtheplantesprimary.database.windows.net.*) 3. Select the first **Event** from the list which will correspond to ConcertId 1, and then select **Ticket Level**,and **Customer**. 4. For the purposes of this lab, choose the same event as used previously. 5. In the **Tickets To Purchase** field, enter **10000**. 6. In the **Bulk Purchase** field, enter **80**. 7. Click **Start**. | As you can see, we just simulated another real-world task where 10,000 tickets were purchased, but the tickets per second results look pretty much the same. |
|  | 1. Switch back to the Azure portal. 2. Make sure that you have the **Summary** pane open for the Customer1 Database on the *Primary Database Server* (in this example *mbjulieandtheplantesprimary.database.windows.net*). 3. Within a minute or so, you should notice a spike in the Resource Utilization chart from the LoadGenerator. Notice this time the database CPU usage spiked to around 40 percent, which demonstrates that the added capacity is already providing a benefit. | As you can see, we just simulated another real-world scenario where 2,000 tickets were purchased, and this time the database CPU usage spiked to around 40 percent, which demonstrates that the added capacity is already providing a benefit.  Beyond scaling up individual databases, you will next see how to let Azure SQL Database help customers dynamically scale groups of databases to meet changes in demand, rather than have to manage this manually. |