Polyglot Storage

Lab Document

Overview

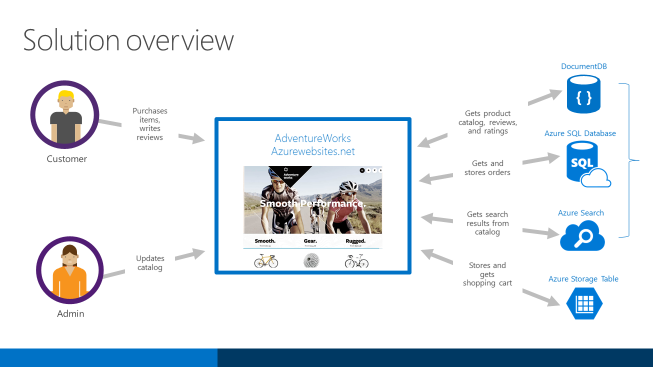
In this scenario based HOL, you will learn how to build a ‘polyglot persistence’ data pattern that is common in modern cloud hosted applications. Requirements of modern applications, such as, greater scale and availability, have driven the industry to begin using a much broader range of technologies for storing data within an application. Microsoft Azure provides a range of storage technologies that support these architectures and this HOL provides an example of the use of these in the well understood scenario of e-Commerce. With data services in Microsoft Azure, you can quickly design, deploy, and manage highly-available apps that scale without downtime and that enable you to rapidly respond to security threats. Features built into services like Azure SQL Database, Azure Search, and Azure DocumentDB help your apps scale smartly, run faster, and stay available and secure.

In this HOL you will see a browser based e-commerce application running under the LCA approved sample company name ‘AdventureWorks’. It has been created to demonstrate functionality provided by the following data storage technologies (SQL Database, DocumentDB, Search, Table Storage). In a real application, decisions will need to be made as to where data is stored. In this HOL we wish to highlight; how using multiple Azure data service technologies allows you to take a modern approach to data in your applications.

Note- The website (which gets built out of this HOL) is not intended to be a fully functioning site. It is not designed to be a reference e-commerce implementation nor a starting point for a customer’s implementation of an e-commerce site on Azure, rather it will provide the following functionality in order to demonstrate the selected storage technologies.

In the course of this lab, you will gain greater familiarity with Azure SQL Database, DocumentDB, Azure Search and Table Storage through performing the following tasks:

* Familiarize yourself with one of the tenant-company’s websites and its Azure SQL Database backend.
* Create a new database using the Azure portal.
* Configure and implement vertical scaling by increasing the capacity of a database.
* Use Azure SQL Database auditing features to track down an erroneous deletion from a database.
* Use Azure SQL Database point-in-time restore to correct the deletion (Optional)
* Configure and implement Azure SQL Database geographic disaster recovery to prevent large-scale data loss.
* Locate data using Azure Search.
* Modernize and create an iterative experience using DocumentDB.



|  |  |
| --- | --- |
| **Demo Story Elements** | **Completed (Yes/No)** |
| **Scenario 0: Azure Search – Index Definition**  Discuss how we maintain a product index within Azure Search. Contained within this index, we store everything to show you the products on this home page. Setting up a new index and getting started is simple. |  |
| **Scenario 1: Azure Search – Indexing Documents**  Discuss how we index documents to Azure Search, and how we can define indexers to do this for us. |  |
| **Scenario 2: Azure Search – Search Suggestions**  Discuss how we use out of the box Azure Search to provide search suggestions to the user. |  |
| **Scenario 3: Azure Search – Highlights**  Discuss we use highlights in Azure Search to provide highlighting of search terms. |  |
| **Scenario 4: Azure Search - Facets**  Discuss we use Facets provided by Azure Search to provide us with refiner items. |  |
| **Scenario 5: Azure Search – Scoring Profiles**  Discuss we can use scoring profiles to manipulate the priority of search results. |  |
| **Scenario 6: DocumentDB**  Discuss how we can rapidly produce new applications using DocumentDB. |  |
| **Scenario 7: DocumentDB – Schema Free Query**  Discuss how DocumentDB is well suited to query across schema free attributes. |  |
| **Scenario 8: DocumentDB – Stored Procedures / UDF**  Discuss how we can use stored procedures / UDF as part of our POST request to get the results we need. |  |
| **Scenario 9: DocumentDB – Nested Documents / Consistency**  Discuss how DocumentDB supports tunable consistency. |  |
| **Scenario 10: DocumentDB - Triggers**  Discuss how we can leverage triggers to automatically execute on events. |  |
| **Scenario 11: DocumentDB - Concurrency**  Discuss DocumentDB can handle common concurrency issues. |  |
| **Scenario 12: DocumentDB – Scale Points**  Discuss how DocumentDB can easily be elastically scaled. |  |
| **Scenario 13: DocumentDB - Transactions**  Discuss how DocumentDB provides transaction processing over multiple documents. |  |
| **Scenario 14: Table Storage**  Discuss Azure Table storage. |  |
| **Scenario 15: Azure SQL – Transaction Data Store**  Discuss how we can use Azure SQL to store transactional records. |  |
| **Scenario 16: Azure SQL – Predictable Performance**  Discuss how we can easily upscale or downscale the performance of Azure SQL. |  |
| **Scenario 17: Azure SQL – Database Recovery**  Discuss how Azure SQL automatically stores Point in Time restore points, and how we can leverage them to undo mistakes. |  |
| **Scenario 18: Azure SQL – Disaster Recovery**  Discuss how we can use Active geo-replication to handle potential catastrophic failures. |  |
| **Scenario 19: Azure SQL - Auditing**  Discuss how we can store the audit log to an Azure Storage account. |  |
| **Scenario 20: Azure SQL - Vertical Scaling**  Discuss how the website is anticipating a big sale and how that might require some increased capacity. Simulate some load using the mass order generator, and then show the spike in resource utilization. From the portal, show how easy it is to scale up by changing the size of an instance (for example, from B1 to S3) and how quickly the change is implemented. Re-run the mass order generator and show how the increased capacity has significantly improved the site’s ability to handle a spike in sales. |  |
| **Scenario 21: Power BI – One Click Updates and drilldown**  Discuss how Power BI responds to updates in the underlying Azure DocumentDB, SQL Database, or any other source it is connected to, and the interactive nature of PowerBI visualizations. |  |
| **Scenario 22: Power BI – Add DocumentDB collection and create a visualization**  Discuss how easy it is to add new DocumentDB collections to PowerBI and join them to the data model. PowerBI can extract data from a DocumentDB collection at any level of the hierarchy. When the data is extracted, it is represented as a dataset in PowerBI which can then be joined to other datasets, regardless of the source of the data. |  |
| **Scenario 23: Power BI – Upload report to the Power BI app and create a dashboard** Discuss how Power BI reports are consumed in the cloud, and how to upload a report |  |
| **Conclusion/Summary** |  |
| **TOTAL** |  |

| Screen | Click Steps |
| --- | --- |
|  | **DEMO SETUP** |
|  | Open Browser to http://INMMDDYYYY.azurewebsites.net (INMMDDYYYY is what you used in the deployment. IN is your initals and MMDDYYYY is the date). |
|  | * Select Melissa MacBeth as the customer by   + clicking on the person Icon   + selecting Melissa MacBeth |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 0: Azure Search – Index Definition** | |
|  | 1. Open browser http://INMMDDYYYY.azurewebsites.net |  |
|  | 1. Open a new browser window and navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Discover/AzureSearchSchema/Products 2. Click Viewer in the top left hand corner 3. Expand Out Fields Collection | *This is a JSON that defines the schema of our search index for Products.* |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Api/Discover/Index | *Let’s see how we create the index.*  *This is a JSON representation of the Index object that is created in the AdventureWorks code. This is then passed into the Azure Search Client.* |
|  | | *Build Index creates the index object as seen above. This is then passed into CreateAsync.*  *This index is created by using some custom attributes on our models, which then builds up the index object to pass to Azure Search.*  [*https://msdn.microsoft.com/en-us/library/azure/microsoft.azure.search.searchserviceclient.aspx*](https://msdn.microsoft.com/en-us/library/azure/microsoft.azure.search.searchserviceclient.aspx)  *Source Code: MSCorp.AdventureWorks.Core->AzureSearchClient.cs->Line 80* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 1: Azure Search – Indexing Documents** | |
|  | 1. Navigate to the Search Box. |  |
|  | 1. Type Mountain into the search box and press enter | *You will notice that there currently are no products being displayed on the website.*  *We need to index some documents to the search index. There are two ways to do this.* |
|  | | *We can upload documents as and when we need to, we are passing an object that represents the document and then indexing it through the Azure Search Client.*  *You can either use a full model (class) which matches your index type. Or you can instantiate a Document object which allows you to build up the document with key value pairs. (This allows for partial updates where you only need to update one field).*  *To do an update. You would call indexClient.Documents.ReplaceAsync. The key must be provided in the object.*  *But there is a more automated way, where no code is required.*  *Source Code: MSCorp.AdventureWorks.Core->AzureSearchClient.cs->Line 200* |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Api/Discover/Indexer | *We can create an indexer which is scheduled task that will look at a datasource and every x minutes and index them into the Search Index.*  *To create the Indexer. You must first define a datasource, and then define an indexer that uses that data source.*  *This returns a JSON that represents the DataSource and Indexer objects used in AdventureWorks for the products indexer.* |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Indexer/Create 2. Navigate to http://INMMDDYYYY.azurewebsites.net/Indexer/Run | *Let’s create the Indexer by running this command. First we are creating it. Then we are setting it to run.*  *Note: This is creating a predefined indexer which will link the existing DocumentDB database for products with the existing azure search service.* |
|  | | *While we wait for the Indexer to run, let’s observe how we created the data source and indexer.*  *BuildDataSource creates a DataSource object (as shown in step 3). This is a datasource that looks at the DocumentDB product database.*  *BuildIndexer creates an Indexer object (as shown in step 3).*  *After a couple of minutes, we should start getting results (shown in Scenario 2).*  *Source Code: MSCorp.AdventureWorks.Core->AzureSearchClient.cs->Line 114* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 2: Azure Search – Search Suggestions** | |
|  | 1. Navigate to the search box. |  |
|  | 1. Type ‘Mountian’ as a misspelling of ‘Mountain’ into the search box. (Don’t press enter yet) | *We use out of the box Azure Search to provide search* ***Suggestions*** *to the user. Suggestions is the term Azure Search uses to retrieve suggestions based on partial search input. They can only be set for fields of type Edm.String or Collection(Edm.String)*  *One of the options for Search suggestions is fuzzy which takes a Boolean parameter.*  *When fuzzy is turned on the Search service will find suggestions even if there's a substituted or missing character in the search text.*  *Worth noting that while this provides a better experience, in some scenarios it comes at a performance cost as fuzzy suggestion searches are slower and consume more resources.* |
|  | 1. Open a new browser window and navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Search/Suggest/Mountian - *Notice that we are sending the misspelling of mountain to the Suggest endpoint of the service.* 2. Expand out the element 0 of the Results collection. | *See that we are getting the correct spelling of mountain when passing in the incorrect spelling.* |
|  | | *Let's see how this suggest was executed.*  *Notice UseFuzzyMatching = true. This is what allows misspellings to still be found.*  *Source Code: MSCorp.AdventureWorks.Web->ApiSearchControllert.cs->Line 48*  *Source Code: MSCorp.AdventureWorks.Core->AzureSearchClient.cs->Line 300* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 3: Azure Search – Highlights** | |
|  | 1. Navigate to the search box. |  |
|  | 1. Type Mountain into the search box (Don’t press enter yet). | *Another one of the options for Search suggestions is highlight.*  *Highlight is an optional parameter, when populated requires a comma separated list of field names. These fields will be the ones which are highlights depending on the search text passed in.* |

|  |  |  |
| --- | --- | --- |
|  | 1. Open a new browser window and navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Search/comments/Mountain | *Let’s have a look at the resulting JSON we get back as a result of the search highlighting. You’ll see any entries which have sub text in italics like the text “This is the best* ***Mountain*** *bike fictitious money can buy.”*  *Looking at the raw JSON you see a text attribute within a value collection with the value “This is the best <em>mountain</em> bike that fictitious money can buy.” This identifies the highlighted text to be displayed.*  *(Note that the em tags are not visible in the ‘viewer’, so look at the raw text).* |
|  | | *Let's see how this search was executed.*  *Note that HightlightFields consisted of the Description field which lets Azure Search know that matches in the Description column should be bolded.*  *You can set your own bold tags by using the HighlightPreTag and HighlightPostTag properties of the SearchParameters.*  *Source Code: MSCorp.AdventureWorks.Web->ApiSearchControllert.cs->Line 92*  *Source Code: MSCorp.AdventureWorks.Core->AzureSearchClient.cs->Line 325* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 4: Azure Search – Facets** | |
|  | 1. Navigate to the search box. 2. Type ‘Mountain’ into the search box and press **Enter**. | *Azure search service provides us both the Search results and the refiner lists you see on the left hand side of the screen. Notice that we get other products which contain the word ‘Mountain’, these can be filtered using Azure Search Facets.* |
|  | 1. Select these Facets (Filters)    * Black    * 40” | *As we select and un-select facets, we are going back to Azure Search service for the facet filtering including the text search passed (in this case Mountain).*  *We do this by filtering against the facets using the word values and piped filters*  *E.g. facet=color,values:Black|Silver|White* |
|  | 1. Open a new browser window and navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Search/Products/Mountain | *Here you can see the Facets collection which provides us with our “groups” of filters that we can use to refine the results. (We provided the facet collections we wanted in Scenario 2.)* |
|  | | *Let's see how we can use Facets with Azure Search.*  *See that we have provided a Filter with this search. In this case we are asking to match where the product category is equal to the one passed into the function.*  *Source Code: MSCorp.AdventureWorks.Web->ApiSearchControllert.cs->Line 176* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 5: Azure Search – Scoring Profiles** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Search/Mountain(or type mountain in the search box and click enter) 2. Click a product image which is not at the top of a list. (but has mountain written in the name) 3. Click Edit Product Details. | *This will open up a new browser window from here to review the product definition. When looking at product detail page you are viewing the full product representation which is stored in DocumentDB* |
|  | 1. Update the product priority to be significantly larger than it is right now and click save. **Note**: the priority scoring profile has been setup for values between 1 and 10,000. | *As the product is updated we persist the changes into product storage and then update the representation within Azure Search.* |
|  | 1. Navigate back to the original browser window with the search results and refresh the page. | *By updating the products priority we have altered the order in which the products are returned.* ***Note****: This may take 5 minutes. Or we can force it straight away by running http://INMMDDYYYY.azurewebsites.net/Indexer/Run* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 6: DocumentDB** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Product/BK-M18B | *In AdventureWorks we have two product stores, DocumentDB which is the master product store and the product index from within Azure Search. The product information which you see on this product details page is surfaced directly from DocumentDB and passed to the browser for display.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 7: DocumentDB – Schema Free Query** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Product/BK-M18B 2. View the specifications of the product. | *Each product has some level of specifications which will be unique to each product. E.g. Bike “Frame Material : Carbon Fiber” VS Water bottle “Material : Eastman Tritan copolyester”* |
|  | 1. Open a new browser window and navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Search/Product/BK-M18B 2. Expand out the Product Attributes Collection. | *These specifications show the schema-free nature of the DocumentDB, this is not particularly special but next we will talk about query over schema-free data which really separates DocumentDB out from the competition.* |
|  | 1. Go back to the original page - http://INMMDDYYYY.azurewebsites.net/Product/BK-M18B 2. Click on the text ‘mountain biking’ under the specification ‘recommended for’ | *In other areas of the application, we’ve used Azure Search service for predictable searching across known product attributes E.g. name, description, category, and manufacturer. DocumentDB is well suited to query across schema free attributes. DocumentDB provides a* ***Rich query over schema-free design*** *which allows us to query over these product specifications. Specifically:*   * *Schema-free design allows for agile development and continuous iteration* * *Differentiated querying through no limitation to the number of ways you can query your data because there is no forced, pre-defined set of indices* * *Query hierarchical JSON data through a succinct query grammar, including transforms and inline JavaScript* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 8: DocumentDB – Stored Procedures / UDF** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Product/BK-R50B | *Notice that you are still seeing the USD price as the application still thinks we are ‘Melissa MacBeth’ based out of the US.* |
|  | 1. Click on the person icon. 2. Select Jeff Wang. | *Notice that the page refreshes and the price has been updated with a NZ Dollar price.* |

|  |  |  |
| --- | --- | --- |
|  | 1. Open a new browser window and navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Search/Product/BK-R50B 2. Expand out the Prices Collection | *In the DocumentDB, we store a Foreign exchange document collection. When we return product detail we join to the exchange collection and calculate for each currency using a server side stored procedure. We then let the client decide which one to use (based on the selected customers default currency).*  *Stored Procedures are named procedures with user defined signatures.* |
|  | | *Let see how we can add a stored procedure to DocumentDB.*  *CollectionUri is defined by the UriFactory, which is a way of getting Uris to relevant objects in the DocumentDB.*  *e.g.*  *UriFactory.CreateDocumentCollectionUri(databaseName, collectionName);*  *This technique is used to create other objects such as:*   * *Triggers* * *User Defined Functions* * *Permissions* * *Documents* * *Users*   *Some of these are covered later on.*  *Source Code: MSCorp.AdventureWorks.Core->DocumentDbProductRepository.cs->Line 62* |
|  | | *Let’s see the stored procedure in action. (written in JavaScript).*  *Let’s have a look at the Stored Procedure which loads up the product and joins off onto the foreign exchange rate document collection.*  *The LoadProductDetails stored procedure is responsible for loading up a single product document based on a given product code.*  *Our Stored Procedure is doing a few different things above and beyond pulling from the DocumentDB:*   * *Fetching the related attachments for the product* * *Calculating the foreign exchange rate* * *Calculating the discounted price based on the currency*   *Looking at the set of stored procedures you will also see ‘SearchForProductByAttribute’ which is the mechanism we used in the previous step to query across the schema free attributes.*  *Let’s look at how the foreign exchange is added. Firstly, we load the product document up based on the product code from the product document collection passing it a callback to run once loaded*    *Next we read the rate card for the product currency from the Rate documents within the same collection. We give it a callback once completed which calls through to generateResponse.*  *GenerateResponse will go through each rate and go through all the exchange rates found and calculate the discounted price.*    *Looking back at the JSON you see that the response has a Collection of Prices which is populated above. We actually calculate the discounted price in the same way based on the product discount and foreign exchange rate.*  *Source Code: MSCorp.AdventureWorks.Core->DocumentDbProductRepository.cs->Line 123* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 10: DocumentDB – Nested Documents / Consistency** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Product/BK-R50B 2. Add a comment underneath the existing review and click add. | *The product review document holds the original review and any related comments are stored as sub documents. This makes up the hierarchy.* |
|  | 1. Navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Search/ProductReview/BK-R50B | *Looking at the JSON you see that the responses collection has a set of comments displaying the nested document structure.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 9: DocumentDB – Triggers** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Product/BK-R50B 2. Enter a new review with the word “faster” in it and click Add | *Now that we have written a new review we need a way to update the review search index so that we can see results in the search suggestion drop down.*  *To achieve this and not influence the time to save the comment we write the comment to a journaling document using a DocumentDB post trigger.*  ***Note****: We could have also used an indexer here, by using this method, we could have changed the review to fit our index, or even had the capability to post it to other systems where the review would be useful.* |
|  | | *Let's look at this trigger.*  *We are creating a new snippet based on the review attributes and inserting it into a document collection. This is a separate collection which stores product review ready to be processed.*  *Triggers are special user defined functions that can be configured to be executed automatically on certain events like the creation or deletion of a document. Triggers come in two types - pre-triggers and post-triggers. Triggers are similar to stored procedures, but the methods should have no input parameters and not return an output response.*  *Triggers get executed with POST, PUT and DELETE requests on documents corresponding to create, update and delete operations respectively. Pre-triggers execute before the actual request and an exception thrown from a pre-trigger aborts the entire transaction. Post-triggers execute within the transaction of the request.*  *Source Code: MSCorp.AdventureWorks.Core->DocumentDbProductReviewRepository.cs->Line 90* |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Admin/Sync | *This will invoke a separate process which picks up any unprocessed journal documents as a result of the trigger process and then pushes them to Azure Search for indexing and surfacing to the search bar.* |
|  | 1. Navigate to the search box. 2. Type faster into the search box. | *You should now see the text ‘faster’ appear as a sub line in the search results box.* |
|  | **Scenario 11: DocumentDB – Concurrency** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Product/BK-M18B 2. Click Edit Product Details twice opening them in separate tabs. This is to mimic Mary and her colleague’s version of the edit page. | *Let’s see what happens when two people update the same record at the same time.* |
|  | 1. In the first tab update the ‘**description**’ and click ‘**save’.** | *The save has been successful as denoted by the green drop down.* |
|  | 1. In the second tab make the same edit and click ‘**save**’. | *The second save has failed as denoted by the error drop down.*  *DocumentDB support the ETag attribute. We can pass this in when saving the document to conditionally succeed if the ETag matches the current version.*  *Using the.net libraries this is easily achieved using an access condition with the Etag, this is then passed into the request options for the replace document async call.*  *AccessCondition accessCondition = new AccessCondition { Condition = product.ETag, Type = AccessConditionType.IfMatch };*  *RequestOptions requestOptions = new RequestOptions { AccessCondition = accessCondition };*  *ResourceResponse<Document> updateResponse = await Client.ReplaceDocumentAsync(product, requestOptions)*  *You’ll notice similarities between the use of ETAG here and within blob storage. This is yet another indicator of the ease of building with familiar tools within DocumentDB.* |
|  | | *Let's see how we used the Etag to handle concurrency conflicts.*  *We pass in the access conditions, stating the etag must match. If it doesn’t it will throw an exception for us so we know when concurrency has been breached.*  *Source Code: MSCorp.AdventureWorks.Core->DocumentDbProductRepository.cs->Line 313* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 12: DocumentDB – Scale Points** | |
|  | 1. Click on the person icon. | *One option for dealing with scale is at design time. In this instance when the AdventureWorks data model was created we made a design time decision to guard from customer expansion by splitting out the customer collections.*  *We have a three separate collections for the customer. We determine from the customer details which collection they should be stored in. From within the application layer when you click on the person icon we manually manage the query across all the customer collections and roll them up into a single response.* |
|  | 1. Navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Customer/GetAllCustomers 2. Open up the first and last customer in the list | *DocumentDB will allow you to* ***Elastically Scale*** *your document tenant**based on the anticipation of load or reaction to load. This is achieved through the Azure management portal.*   * + - *Scale up or scale down storage and throughput via stackable units that allow you to predictably control costs, this can also be achieved using power shell*     - *Storage and throughput scale linearly with cost* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 13: DocumentDB – Transactions** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Admin/Product/Discount 2. Select Recommended For in the first drop down. 3. Select Mountain Biking in the second drop down. | *Now that we have found all the products which we want to update let’s give all the products a 15% discount.* |
|  | 1. Type 15 into the set discount. 2. Click Apply Discount (Wait for confirmation). | *Under the covers we are updating all the product documents with that discounted price within a single ACID transaction. DocumentDB provides a full-featured NoSQL document database service including out of the box transactional processing over multiple documents through application defined j*  *JavaScript / stored procedures.* |
|  | 1. Right click any product in the list and **Open in a new tab.** 2. Click the Edit Product Details. 3. Save the product and go back to original discount tab. (Do not refresh). |  |
|  | 1. Click Apply Discount (wait for confirmation). | *See that the update discount has failed. Because we edited one of the products in the set, it was unable to be updated as it was out of date.*  *We have leveraged the ETag on the product, using a concurrency check we have demonstrated one scenario where we wanted to abort the underlying transaction.*  *We achieve this behavior through further use of stored procedures.*  *Stored procedures, triggers and UDFs are modelled after the concepts supported by RDBMS. All JavaScript logic is executed within an ambient ACID transaction with snapshot isolation. During the course of its execution, if the JavaScript throws an exception, then the entire transaction is aborted.* |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Api/Discover/StoredProcedures/ProductCollection 2. Scroll to procedure ApplyDiscountToProducts. | *This is the stored procedure responsible for apply discounts and checking the Etag.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Table Storage** | |
|  | **Scenario 14: Table Storage** | |
|  | 1. Make sure you are signed in as Jeff Wang. 2. Navigate to http://INMMDDYYYY.azurewebsites.net/Product/BK-M18B 3. Click Add to Cart. |  |
|  | 1. Click on the cart icon. |  |
|  | 1. Navigate to http://jsonviewer.stack.hu/#http://INMMDDYYYY.azurewebsites.net/Api/Cart/108 | *As you can see by the representation of what is in the shopping cart, we are storing an un-schematized JSON representation of the order. If the order schema changes in any we don’t need to consider changing anything about table storage. All information pertaining to the information is stored in a single document, so no links or hierarchy are required.*  *We store the shopping cart based on the customer who has added the order to the cart. All the information is stored in a single document so no links or hierarchy is required.*  *Table storage is cheap and accessible as a simple RESTful service. The shopping cart is a canonical use of key value pair - simple to retrieve based on the customer key and not required after the users session is completed.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Azure SQL** | |
|  | **Scenario 15: Azure SQL – Transaction Data Store** | |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/Checkout 2. Click Make Purchase | *As part of making the purchase we take what is in the shopping cart and push it into an Azure SQL database as the transactional store.* |
|  | 1. Open up SSMS with the connection to the database. | *Let’s head over to the database and validate that we see a new order in the database.*  *Run the following SQL looking at the order and lines.* |
|  | 1. Execute these queries   SELECT \*  FROM [dbo].[Orders]  ORDER BY Id DESC  SELECT \*  FROM [dbo].[OrderLines]  ORDER BY Order\_ID DESC | *By using Azure SQL, we remove virtually all infrastructure maintenance with SQL Database which provides automatic software patching as part of the service. We are also able to leverage SQL Server skills across on-premises and cloud environments with a familiar relational foundation and T-SQL functions.*  *These two aspects of Azure SQL make it both familiar and self-managed which have enabled AdventureWorks as a company to drive fast time-to-market and unprecedented efficiencies with near-zero maintenance service.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 17: Azure SQL – Predictable Performance and Vertical Scaling** | |
|  | 1. Login to the Azure portal at <https://portal.azure.com/> with credentials which has access to the database hosting the application. 2. Click **Browse** and select **SQL databases.** 3. Select the **AdventureWorks***,* Click on **Monitoring** in the Database blade. | *As you can see from the monitoring on the AdventureWorks database, at the moment with the current load we are in no way stressing the database. The database is configured on the premium service tier. This is exactly where we want to be with the sale event starting this weekend. But it was a different story last year.* |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/images/LastYearMonitoring.jpg | *Here are the metrics from last year’s Black Friday sale event. We were caught out with the amount of traffic generated from so many customers placing orders.*  *We can see that we were reaching to the 80% for Log writes per second and the CPU percentage. Based on these telemetry results we can clearly see our database was getting close to the upper threshold limit of the service tier, we had to quickly react and upgrade to a higher performance level to maintain great performance and thus a great customer experience.*  *This was a good problem to have, but this year we are prepared in advance as we have already scaled our database service tier up to premium.* |
|  | 1. Close the tab and navigate back to the open portal window, Right click on the graph and select edit chart. | *The Azure Portal has built in dashboard views showing telemetry data. This allows us to get an understanding of what’s been happening on our databases over time. Note as developers we also have the ability to programmatically obtain this.*  *We can easily include or exclude metrics to plot on the graph and change the charts time range. Options for telemetry are Storage, CPU percentage, Physical Data reads, Log writes per second, blocked by firewall, deadlocks, failed connections and successful connections.* |
|  | 1. Click on Monitoring Graph. | *Notice when I click on the monitoring graph we have the ability to add an alert. We can configure alerts on performance metrics. For example, if you expect the workload on your database to grow, you can chose to configure an email alert whenever your database reaches 80% on any of the log writes percentage. You can use this as an early warning to figure out when you might have to switch to the next higher service tier. Alternatively you could use it to monitor when it falls below a certain percentage to downgrade the service tier.*  *Let's do an experiment.* |
| C:\Dev\HOL-AdventureWorks\TimeTakenOnS3.PNG | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/MassOrdering 2. Click Generate and Save Mass Order Data (and wait). | *This will generate 1,000 orders simulating a big sale event with big usage spikes.*  *You will see the time it took for this process.* |
| C:\Dev\HOL-AdventureWorks\DTUPercentageonS3.PNG | 1. Go back to the monitoring graph. | *Note: This may take 5-10 minutes to update.*  *Notice that DTU percentage is at 20% (your results may differ slightly).*  *We will compare the results when we scale down in Scenario 19.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 17: Azure SQL – Database Recovery** | |
|  | 1. Sign in to the [Azure Management Portal](https://manage.windowsazure.com) using your Microsoft account. 2. From the **Databases** list, select the database you want to restore - *Only the new service tiers Basic, Standard and Premium service tiers support Point in Time.* 3. At the bottom of the page in the command bar, click **Restore**. This launches the **Specify Restore Settings** dialog box. 4. Select the **Restore Point.** 5. Specify a new database name in the **Database Name** field. 6. Click create to submit the restore request. | *Point in Time Restore is designed to recover a database to a specific point in time within the backup retention period supported by the service tier of the database. Restoring creates a new database with the same service tier that was in use at the chosen restore point and the lowest performance level supported by that tier. For example, if you restore a database which was set to Premium at the specified restore point in time, the new database will also be a Premium database.*  *The Restore Point is used to specify the point in time to which your database should be restored. The restore point must be within the retention period supported by the database service tier and the Point in Time setting must be set to ON.*  *The backup retention period is a maximum of seven days for Basic, 14 days for Standard, and 35 days for Premium. Requests to restore to a point in time outside of the supported retention period for the service tier will fail.*  *Restore operations may take a long time to complete. You can monitor the status of the restore operation on the Databases list. Databases that are being restored are visible in the Databases list once the restore is in progress. The row is populated with the database settings and the status of the restore operation which is displayed as Restoring until the operation is completed.* |
|  | 1. Navigate to the SQL Server. 2. On the side tab, scroll down to deleted databases. | *We can also restore a deleted database. This list includes a list of****Restorable Deleted Databases*** *available to restore.**Similarly to above, the restore point must be within an appropriate retention period supported by the service tier.*  *Point in Time Restore enables restoring a database to a point in time in the past within the retention period for the given service tier. However, the retention period for the service tier may not be long enough to meet business needs. In this case, consider creating an archived database using methods such as automated export through Azure.*  *Point in Time restore offers self-service control over data restoration from available backup data, putting the power to restore in the hands of customers in the case of a human or programmatic data deletion scenario.*  *Using Azure there are other options for database recovery depending on the tier you are using. We’ve already mentioned Automated Export above, but there is also Geo restore, standard Geo replication and a database copy and manual restore.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 18: Azure SQL – Disaster Recovery** | |
|  | 1. Sign in to the [Azure Management Portal](https://manage.windowsazure.com) using your Microsoft account. 2. Select a database from the **Databases** list. This opens the **Database Dashboard** or the **Quick Start** page. 3. Select the **Geo-Replication** tab for the database. | *Active Geo-Replication is currently only supported for databases in the Premium service tier.* |
|  | 1. To create a new active secondary, click on a **target region**. | *Secondary databases are continuous copies which are asynchronous replicas of the primary. Standard databases offer one offline secondary. Premium databases offer up to four readable secondaries.*  *This launches the Add Secondary dialog which allows you to specify the Secondary type, and Target Server settings.  In this case we want a readable copy so that members of the AdventureWorks staff can report against.* |
| 1. Select Secondary Type : Readable 2. Reuse an existing server which you know the credentials to or create a new server (recording the username and password) and click the tick. 3. Wait five mins. | *The list of servers shown in the Target Server drop down box is the list of servers that are enabled for Active Geo-Replication in the same subscription as the primary and have available quota.*  *If the target server selected is in the same region than the primary database, you will see a warning message advising you that the primary and secondary are in the same regions. Microsoft recommends hosting the primary and active secondary in different regions for disaster recovery purposes. The active secondary will have the same name as the primary database, they must reside on different server than the primary. The “Readable” option we identified that the database will be copied to a secondary and be readable. The other offline option is a copy for disaster recovery. An offline secondary database is created in a different predetermined region than your primary database.*  *The secondary database only becomes available to client when the data center hosting the primary database fails.*  *In the context of AdventureWorks, we don’t get the additional benefit of being able to read from the secondaries for reporting, so we opted for the online version.* |
|  | 1. Refresh Database list in the portal. | *Now you can see another database in the portal with the same name. If you open it, you will see that it says it is the secondary.* |
|  | 1. Navigate to http://INMMDDYYYY.azurewebsites.net and make another purchase. | *Let’s check the replication by placing a new order. We expect that the primary transactional database will contain the order which we have just placed.* |
|  | 1. Open up SSMS from Scenario 14 and rerun the query against the primary database. | *See the newly added order.* |
|  | 1. Add a connection to the new database/server setup during the geo replication step and run the same query as above. | *See the newly added order in the secondary geo replicated copy of the data.*  *As you can see, being able to setup an active geo replicated copy of your data is very simple.*  *The setup of the Azure database which we have manually achieved here can also be achieved through the use of PowerShell.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 19: Azure SQL – Auditing** | |
|  | 1. Sign in to the [Azure Management Portal](https://manage.windowsazure.com) using your Microsoft account. 2. Select a database from the **Databases** list. 3. Click Auditing and Threat Detection. | *This will enable the auditing and launch the auditing configuration blade.* |
|  | 1. In the auditing configuration blade, select the Azure storage account where logs will be saved. | *If we had several audited databases, we would use the same storage account for all audited databases to get the most out of the preconfigured reports templates.*  *Audit logs are aggregated in a single Azure Store Table named AuditLogs in the Azure storage account.* |
|  | 1. Under Auditing Event, click All, then click OK. | *Depending on the requirements for auditing, we can configure what type of events are audited. Access to data, Schema changes (DDL), data changes (DML), Accounts, roles, and permissions (DCL), Security exceptions.*  *We can also save this configuration as default. This means that these settings would apply to all future databases on the same server, and any that don't already have auditing set up. You can override the settings later for each database by following these same steps.* |
|  | 1. Click Save. |  |
|  | 1. Click Explore. 2. Click Open in Excel. | *To use the template on your audit logs, you need Excel 2013 or later, and Power Query. The template here has fictional sample data in it which we can use to show you the powerful capabilities.*  *Let’s have a look through some of the sheets.*  ***Orientation****: Documentation links and brief description of the visible interactive reports*  ***Anomalies****: Events that usually should not appear in normal operation. Double-click a Count value for the list of events*  ***Drill Down****: Ability to slice to the desired specific set of rows. Double-click a Count value for the list of events*  ***Event Type Distribution****: Distribution of Audit Logs Event Types by several dimensions: Database, Month --> Day, Weekday, Hour in the Day*  ***Event Time Analysis****: Analysis over time by Event Type. The two vertically adjacent charts enable correlating events with different scale of appearance. Double-click a count value for the list of events*  *With the Database Auditing capability, Microsoft Azure SQL Database offers a tool to streamline compliance-related activities, gain knowledge about what is happening in your database, and identify trends, discrepancies, and anomalies. These can serve to enhance business visibility, or potentially indicate business concerns or suspected security violations.* |

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | **Scenario 20: Azure SQL - Vertical Scaling** | |
|  | 1. Login into the Azure Portal. 2. Select SQL Databases and select the AdventureWorks database. 3. Click on the pricing tier tab. |  |
|  | 1. Click on Basic then click Select. | *This will downgrade our database to five DTUs.* |
| C:\Dev\HOL-AdventureWorks\TimeTakenOnB1.PNG | 1. Navigate to http://INMMDDYYYY.azurewebsites.net/MassOrdering 2. Click Generate and Save Mass Order Data and wait. | *This will generate 1,000 orders, simulating a big sale event with big usage spikes.*  *You will see the time it took for this process.* |
| C:\Dev\HOL-AdventureWorks\DTUPercentageonB1.PNG | 1. Go back to the monitoring graph (Database Page). | *Note: This may take 5-10 minutes to update.*  *Notice that DTU percentage is at 98%. Your results may differ slightly.*  *This has drastically reduced the performance of the SQL Server. Luckily with Azure, it is easy to vertically scale the services on an adhoc or full time basis.* |