

This is ok (up)

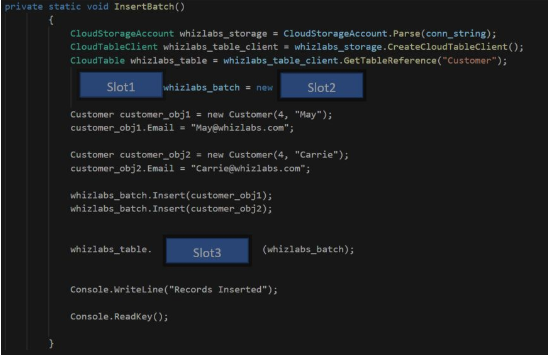
This is nok (down):

* Create SearchINDEXClient
* Create IndexBatch
* Call Documents.Index of SearchIndexClient, passing IndexBatch as parameter

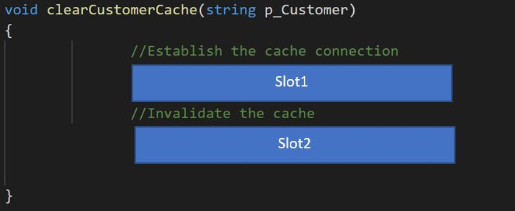
OK: Object, which is used to create indexes in Azure Search service: **SearchServiceClient**, **SearchCredentials**

Create Kubernetes cluster:

* az group create
* az aks create
* az aks get-credentials
* kubectl apply



Slot 1: TableBatchOperation  
Slot 2: TableBatchOperation  
Slot 3: ExecuteBatch



Redis cache  
Slot 1: IDatabase cache = Connection.GetDatabase();  
Slot 2: cache.KeyDelete(p\_Customer);

Create Dockerfile, with: application skillcertlabs.dll runs at startup, run a powershell script (both files are in the same directory as Dockerfile)

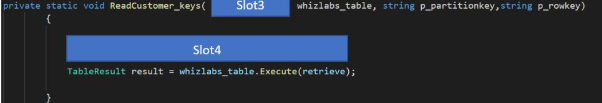
FROM microsoft/dotnet:2.-aspnetcore-runtime  
ENTRYPOINT[„dotnet”, „skillcertlabs.dll”]  
RUN powershell „skillcertlabsscript.ps1”

Querying from Azure storage (table Customers, partitioned by column firstname)  
Get all „Dave”s:

TableQuery.GenerateFilterCondition(„PartitionKey”, QueryComparions.Equal, „Dave”)

Table Storage:

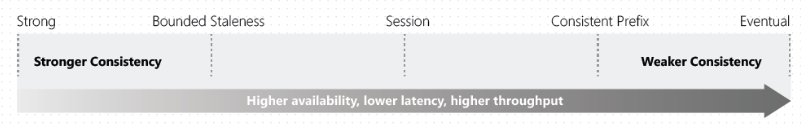
* What is the parition key: The field with which we want to load balance the data
* What is the row key: Must have value



Slot 3: CloudTable  
Slot 4: TableOperation retrieve = TableOperation.Retrieve(p\_partitionKey, p\_rowKey)

CosmosDB for hospital patients, requirements:

1. Status of patient must be the most recent (even if multiple users in different locations update the same patient’s records) 🡪 STRONG CONSISTENCY
2. Health of patient is recorded by one module, here it needs to be unsured that the data must be either the current version, or the previous version 🡪 BOUNDED STALENESS
3. If patient discharged, all charges to be processed, final bill processed 🡪 EVENTUAL CONSISTENCY



1. **Strong:** Reads return the most recent version (pricey)
2. **Bounded staleness:** Reads consistent with a **preconfigured lag**
   1. Lag can consist of a number of the most recent (K) versions or time interval (T)
3. **Session:** client session scope, best balance between strong consistency and performance by eventual consistency (best where writes occur in the context of a single user)
4. **Consistent Prefix:** Always read in the same order as I write (no guarantee that I can read all the data)
   1. If I write A, B, C then I can read: A or A,B or A,B,C (but never A,C or B,A,C)
5. **Eventual Consistency:** no guarantee for order

Azure Batch Service: create compute nodes 🡪 BatchClient.PoolOperations.CreatePool()  
Azure Batch Service: submit a job 🡪 JobOperations.CreateJob() + CloudJob.CommitAsync(Ienumerable, CancellationToken)

I have a Batch Service’s code, it should put its output to Storage account: **DO NOT get access keys of storage account and submit them to batch job, NOR use CORS (...) 🡪 use SAS (shared access signature)**

WebApp has D1 subscription, needs scaling

* Configure web app to use Standard App Service Plan **(we need this. No Shared subscription is enough, but Premium is too much)**
* Enable autoscaling
* Configure scale condition
* Add scale rule

Order CDN

1. User requests image from CDN URL, the DNS routes the request to the best performing Point of Presence location
2. If no edge server in the Point of Presence has an imagine in the cache, it will request it from the origin server
3. The origin server will return an image to the Edge server in the Point of Presence (it’ll cache the image and return the image to the user)
4. Subsequent s may redirected to the same Point of Presence

Logic apps: edit B2B workflow, what to use 🡪 Enterprise Integration Pack  
Logic apps: edit definitions in JSON, what to use 🡪 Code View editor  
Logic apps: visually add functionality, what to use 🡪 Logic Apps Designer

Web jobs:

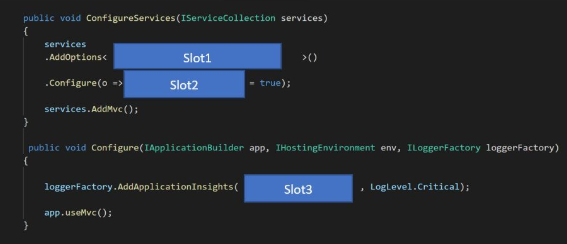
* Triggered: manually/on schedule, runs on single instance that Azure selects for load balancing (no remote debugging)
* Continous: starts immediately after creation (endless loop), runs on all instances that the web app runs on (can be restricted to a single instance) (yes remote debugging)

Multi factor authentication for Azure AD:

* Needs Azure AD Premium
* Azure Ad 🡪 create new conditional access policy

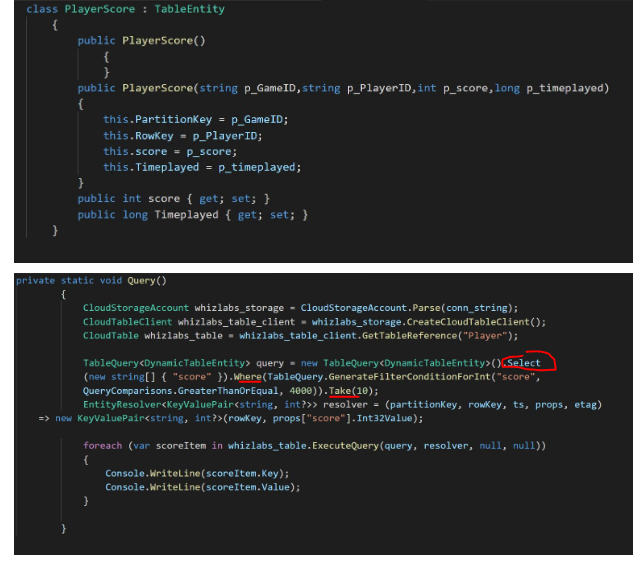
Authentication:

* To communicate with MS Support: **Microsoft.Support/\***
* Useful:
  + Get-AzRoleDefinition –Name „Reader” | ConverTo-Json Out-File c:\sample.json
  + Set-AzRoleDefinition –Role $role
  + (there’s no update)

We are adding app insights logging (**make sure that log messages can be correlated to events tracked by App Insights**):  


Slot1: ApplicationInsightsLoggerOptions  
Slot2: IncludeEventId  
Slot3: app.ApplicationServices

This is valid:

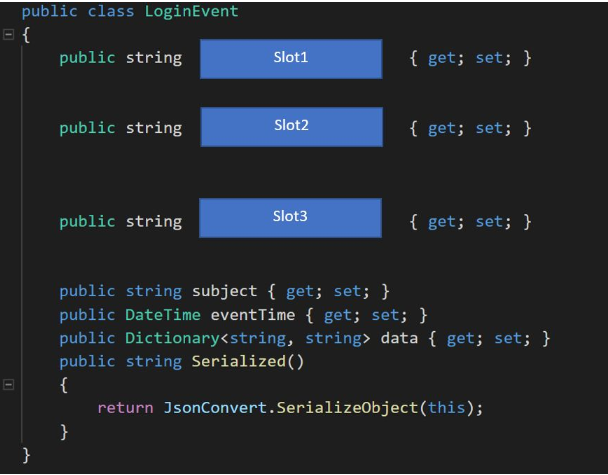


Todo: refresh on Azure AD + Event Grid (topics vs subscriptions)

* Eg: sign in and sign outs need their separate topics
* SqlFilter
* CorrelationFilter
* True/FalseFilter
* No Filter: <https://docs.microsoft.com/en-us/azure/service-bus-messaging/topic-filters>

Recheck questions 77-

Parsing event data from Event grid:



Slot1: id  
Slot2: topic  
Slot3: eventType

We want a Web App to scale on demand: **Application Insights metric**

What to store in Redis cache: **Session state** (not HttpContext.Items, ViewState or TempData)

* Cache aside: load data from DB only when needed
* Content caching: static content
* User session caching: shopping carts, etc (before they used to use cookies, but they can grow, have to be validated, etc 🡪 use cookie as the key to the cache/db)
* Job and messauge queuing
* Distributed transaction

We have a SQL DB, with a column „SSN” to which external company should have no access: **use Enable AlwaysOn encryption**

* Shit answers include:
  + set column encryption as disabled,
  + Assign users to public fixed database role
  + Store column encryption keys in system catalogue view of the DB (nope, should be always in KeyVault)

We want a VM to be encrypted with keys from KeyVault:

1. New-AzVm
2. Get-AzKeyVault
3. Set-AzVmDiskEncryptionExtension

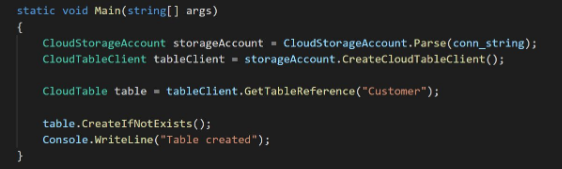
We have a Web App for 4 customers (they need it on separate, individual instances)

* Ability to automatically scale on demand
* Use deployment slots for staging
* Separate, isolated network
* **Use isolated app service plan**

Where to store client certificates: **HTTP request header** (not client cookie, http message body or url query string)  
What encoding to use? **Base64**



Slot1: NotificationHubClient  
Slot2: NotificationHubClient  
Slot3: CreateClientFromConnectionString  
Slot4: SendWindowsNativeNotificationAsync



Can this create table Customer? **Yes, if connection string is provided**Can this work with CosmosDB? **Yes**Can this work with Azure Storage? **Yes 🡪 TableAPI is both for Azure Storage and CosmosDB**

We are caching with Azure API Management:

* A set-variable policy to store the detected user identity: **Inbound policy**
* A cache-lookup-value policy: **Inbound policy**
* A cache-store-value policy: **Outbound policy**
* A find-and-replace policy to update the response body with the user profile info: **Inbound policy**

Azure Search with RegEx? **Change QueryType to Full Lucence search**Azure Search to organize results by counts for name-value pairs? **Facets**