



Deployment

Jaume Canals

Spring 2019

Contents

1	Introduction.....	3
2	Create an Azure Account	3
3	Create nucoris Cosmos DB	3
4	Create nucoris Key Vault	5
5	Create nucoris App Service	5
6	Grant to the App Service Access to the Key Vault	5
7	Service Bus	6
8	Cosmos DB Change Feed Setup	8
9	Application Event Loopback.....	10
10	Authentication	11

1 Introduction

nucoris is the exploratory prototype of a cloud-native healthcare information system (HIS) that I have developed and used as a learning platform during a sabbatical leave.

Here I explain how to deploy it to Azure.

Note: in the instructions below I've used "nucoris" when creating names for several Azure resources. Since some resources must have names that are unique across Azure, you may have to replace "nucoris" with a different name, maybe a combination of "nucoris" + your initials.

2 Create an Azure Account

If you haven't yet an Azure account, you can create a free one:

<https://azure.microsoft.com/en-us/free/>

3 Create nucoris Cosmos DB

- 1) **Create a Cosmos DB account.** Follow the instructions from <https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-manage-database-account>.
 - a. As resource group, use "nucoris"
 - b. As account name, use "nucoris"

c. As API, choose SQL

[Basics](#) [Network](#) [Tags](#) [Review + create](#)

Azure Cosmos DB is a fully managed globally distributed, multi-model database service, transparently replicating your data across any number of Azure regions. You can elastically scale throughput and storage, and take advantage of fast, single-digit-millisecond data access using the API of your choice backed by 99.999 SLA. [learn more](#)

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription

* Resource Group [Create new](#)

INSTANCE DETAILS

* Account Name [documents.azure.com](#)

* API

* Location

Geo-Redundancy

Multi-region Writes

- 2) **Create nucoris DB and collection.** Follow the instructions from <https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-create-container#portal-sql>, keeping in mind that “container” is synonymous to “collection” in our case:
 - a. Specify a new database with id “**nucorisDb**” (this name is mandatory; it’s the name assumed by the application)
 - b. Specify a new collection with id “**nucorisCol**” (this name is mandatory; it’s the name assumed by the application)
 - c. Specify as partition key “**/partitionKey**” (this path is mandatory; it’s the key used by the application)
 - d. Specify as throughput “400”
- 3) **Apply nucoris indexing policy to nucorisCol.** Follow the instructions from <https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-manage-indexing-policy#manage-indexing-using-azure-portal>
 - a. Copy and paste the settings from [nucoris GitHub repository](#):
`...\src\nucoris.persistence\DbObjects\indexing-policy-nucoris.json`
- 4) **Add persistence stored procedure.**
 - a. Copy and paste from [nucoris GitHub repository](#):
`...\src\nucoris.persistence\DbObjects\spPersistDocuments.js`

- b. Use “**spPersistDocuments**” as procedure id.
- 5) **Prime sample reference data.** This prototype does not include views for managing reference data, you will have to add it manually (it could be automated, but it’s so few items that I didn’t find it worth the effort):
 - a. Copy and paste, one by one, the items in [nucoris GitHub repository](#):
 - i. ... \src\nucoris.persistence\DbObjects\sample administration frequencies.txt
 - ii. ... \src\nucoris.persistence\DbObjects\sample medications.txt
 - iii. ... \src\nucoris.persistence\DbObjects\sample users.txt

4 Create nucoris Key Vault

- 1) **Create a key vault.** Follow the instructions from <https://docs.microsoft.com/en-us/azure/key-vault/quick-create-portal>
 - a. Name the vault “nucoris”
 - b. Use as resource group “nucoris”
- 2) Add as secret **Cosmos DB endpoint.** Follow the instructions from <https://docs.microsoft.com/en-us/azure/key-vault/quick-create-portal#add-a-secret-to-key-vault>:
 - a. *Name*: “nucorisDbEndpoint”
 - b. *Value*: take the endpoint from the Cosmos DB resource.
- 3) Add as secret **Cosmos DB authorization key**:
 - a. *Name*: “nucorisDbAuthKey”
 - b. *Value*: take the authorization key from the Cosmos DB resource.

5 Create nucoris App Service

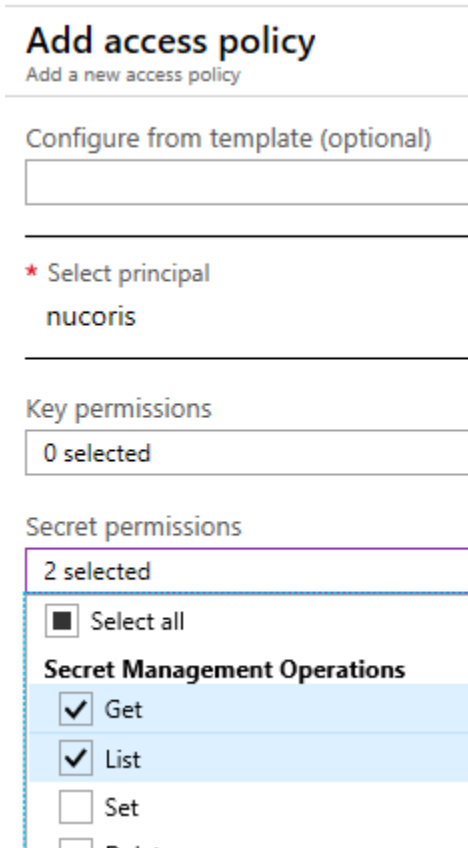
You can either create first the App Service and then publish the application from Visual Studio (or you online repository), or create the App Service while publishing, following instructions from <https://docs.microsoft.com/en-us/azure/app-service/app-service-web-get-started-dotnet>.

It’s very easy, so I will not add any additional explanation, just keep in mind that I’ve used “nucoris” in my case, with the cheapest plan (free) available in West Europe.

6 Grant to the App Service Access to the Key Vault

- 1) The method `GetDatabaseSettings` in ... \src\nucoris.webapp\Startup.cs assume the name of the key vault is “nucoris”. You may need to change it.

- 2) In Azure Portal, go to “nucoris” Key Vault
- 3) Go to *Settings->Access Policies*
- 4) Click *Add New*
- 5) Select “nucoris” AppService as Principal, and assign to it secret permissions to *Get* and *List*:



Add access policy
Add a new access policy

Configure from template (optional)

★ Select principal
nucoris

Key permissions
0 selected

Secret permissions
2 selected

☐ Select all

Secret Management Operations

☒ Get

☒ List

☐ Set

☐ Delete

- 6) Click *OK*

7 Service Bus

- 1) Create a Service Bus namespace, following the instructions from <https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-create-namespace-portal>
 - a. Name is “nucorisSB”.
 - b. Important: nucoris uses *topics*, so currently you need to choose the “Standard” pricing tier.

- 2) Add a topic named “application”. You can use the following settings:

Create topic

Service Bus

*

Name

Application

Max topic size

1 GB

Message time to live

Days

Hours

Minutes

Seconds

0

1

0

0

☒

Enable duplicate detection

Duplicate detection window

Days

Hours

Minutes

Seconds

0

0

0

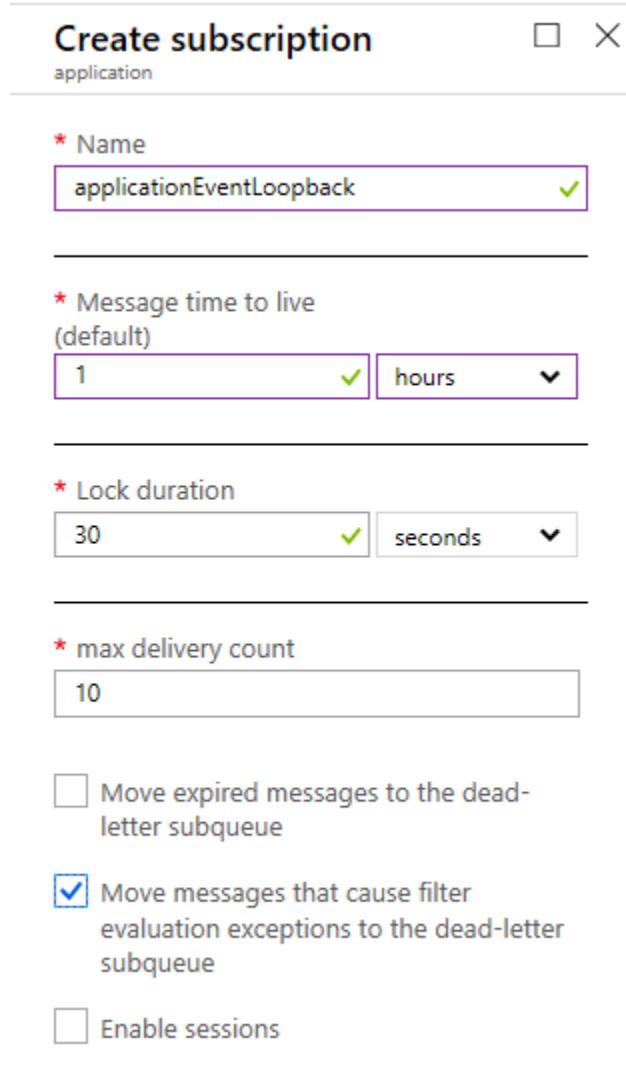
30

☐

Enable partitioning

- 3) Add a subscription to the “application” topic that nucoris will use to listen to events to maintain the *V_ActiveOrdersView* materialized view:
- Name it “applicationEventLoopback”.

- b. Do not enable sessions for it (*Note: we would like to, but sessions are currently not supported in functions' Service Bus trigger*).



Create subscription ☐

application

* Name
applicationEventLoopback ✓

* Message time to live (default)
1 ✓ hours ▼

* Lock duration
30 ✓ seconds ▼

* max delivery count
10

☐ Move expired messages to the dead-letter subqueue

☒ Move messages that cause filter evaluation exceptions to the dead-letter subqueue

☐ Enable sessions

- 4) To peek messages you can use the Service Bus Explorer tool from <https://github.com/paolosavatori/ServiceBusExplorer>
- a. You can install it with Chocolatey. Once installed you can search for it typing "servicebusexplorer" in Windows search.

8 Cosmos DB Change Feed Setup

- 1) Create a new Function App called "nucorisFunc"
- 2) Create a new Function called "nucorisDbFeed"
 - a. As trigger use "Cosmos DB"
 - b. Install any extension it requires.

- c. You will be asked to designate the name of the app setting containing the connection string, collection name, database name, lease collection name, etc. Fortunately, Azure helps with the setup of the connection string (just click *new* pick the Cosmos DB resource name) and the lease collection. The end result should be:

New Function

Name:

nucorisDbFeed

Azure Cosmos DB trigger

Azure Cosmos DB account connection ⓘ *new* *show value*

nucoris_DOCUMENTDB

Collection name ⓘ

nucorisCol

Create lease collection if it does not exist ⓘ

☒

Database name ⓘ

nucorisDb

Collection name for leases ⓘ

leases

Create Cancel

- d. The default template suggested by Azure is:

```
run.csx Save Run
1 #r "Microsoft.Azure.DocumentDB.Core"
2 using System;
3 using System.Collections.Generic;
4 using Microsoft.Azure.Documents;
5
6 public static void Run(IReadOnlyList<Document> input, ILogger log)
7 {
8     if (input != null && input.Count > 0)
9     {
10         log.LogInformation("Documents modified " + input.Count);
11         log.LogInformation("First document Id " + input[0].Id);
12     }
13 }
14
```

- e. Save it for the time being.

- 3) Add an output binding to “Azure Service Bus”.
 - a. Install any extension Azure requires.
 - b. Create a new service bus connection name. Accept the name Azure suggests.
 - c. Name the output parameter “messages”
 - d. Name the topic “application”

Message type ⓘ
 Service Bus Topic

Message parameter name ⓘ
 messages
☐ Use function return value

Save Cancel

Service Bus connection ⓘ [show value](#)
 nucorisSB_RootManageSharedAccessKey_SERVICE [new](#)

Topic name ⓘ
 application

- 4) The final code for the function is:

run.csx

Save

▶ Run

```

1  #r "Microsoft.Azure.DocumentDB.Core"
2  #r "..\bin\Microsoft.Azure.ServiceBus.dll"
3
4  using System;
5  using System.Collections.Generic;
6  using Microsoft.Azure.Documents;
7  using Microsoft.Azure.ServiceBus;
8
9
10 public static void Run(IReadOnlyList<Document> input, ILogger log, ICollector<Message> messages)
11 {
12     if (input != null && input.Count > 0)
13     {
14         log.LogInformation("Documents modified " + input.Count);
15
16         foreach (var doc in input)
17         {
18             var docType = doc.GetProperty<string>("docType");
19             log.LogInformation($"Document with id {doc.Id} has docType='{docType}'");
20             if (docType == "Event")
21             {
22                 var partitionKey = doc.GetProperty<string>("partitionKey");
23                 var message = new Message(System.Text.Encoding.UTF8.GetBytes(doc.ToString()));
24                 message.SessionId = partitionKey;
25                 messages.Add(message);
26                 log.LogInformation($"Added Event document {doc.Id} to service bus 'application'");
27             }
28         }
29     }
30 }
31

```

- 5) The function code is also stored for convenience under *nucoris.webapp/AzureFunctions*.

9 Application Event Loopback

- 1) In the same Function App called “nucorisFunc”, create a new Function called “nucorisSendEventBackToApplication”
 - a. As trigger use “Service Bus Topic”

- b. Install any extension it requires.
- c. Set it up as follows:

Message type ⓘ <div>Service Bus Topic</div>	Message parameter name ⓘ <div>message</div>
Service Bus connection ⓘ show value <div>nucorisSB_RootManageSharedAccessKey_SERVICE new</div>	Topic name ⓘ <div>application</div>
Subscription name ⓘ <div>applicationEventLoopback</div>	

2) The final code for the function is:

```

1 #r "..\bin\Microsoft.Azure.ServiceBus.dll"
2
3 using System;
4 using System.Net.Http;
5 using System.Threading.Tasks;
6 using Microsoft.Azure.ServiceBus;
7
8 // Create a single, static HttpClient, following advice from:
9 // https://docs.microsoft.com/en-us/azure/azure-functions/manage-connections
10 static HttpClient _httpClient = new HttpClient();
11
12 // This function listens to events in the application bus and sends them back to the application.
13 // This allows the application to process incoming user requests quickly and defer handling of non-urgent events, if need be.
14 // See nucoris Persistence document for further information.
15 public static async Task Run(Message message, ILogger log)
16 {
17     log.LogInformation($"Received service bus message with id {message.MessageId} and label {message.Label}");
18
19     if (message.Label == "Event")
20     {
21         var byteContent = new ByteArrayContent(message.Body);
22         byteContent.Headers.ContentType = new System.Net.Http.Headers.MediaTypeHeaderValue("application/json");
23
24         var result = await _httpClient.PostAsync("https://nucoris.azurewebsites.net/api/applicationEvents", byteContent);
25
26         if (result.IsSuccessStatusCode)
27         {
28             log.LogInformation($"Successfully processed service bus message {message.MessageId} with result {result.StatusCode.ToString()}");
29         }
30         else
31         {
32             string errorMsg = await result.Content.ReadAsStringAsync();
33             log.LogError($"[{result.StatusCode}] when processing service bus message {message.MessageId}. Error: {errorMsg}");
34         }
35     }
36 }
37

```

3) The function code is also stored for convenience under *nucoris.webapp/AzureFunctions*.

10 Authentication

The application can be run with no authentication, which will default all users to “USER, Guest”. This is how it has been published to Azure, for ease of access.

If you want to run **nucoris** locally with authentication that allows anyone with a Microsoft account to connect:

- 1) In Azure Portal, go to *Azure Active Directory->App Registrations*
- 2) Click *New Registration*.
- 3) Enter “nucoris” as name, and as *Sign-on url*, for testing, <https://localhost:5001>

4) Once created, edit its *Redirect URIs* configuration as follows:





Redirect URIs

The URIs that we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. Also referred to as reply URLs. [Learn more about adding support for web, mobile and desktop clients](#)

TYPE	REDIRECT URI
Web	https://localhost:5001/signin-oidc
Web	e.g. https://myapp.com/auth

Suggested Redirect URIs for public clients (mobile, desktop)

If you are using the Microsoft Authentication Library (MSAL) or the Active Directory Authentication Library (ADAL) to build applications for desktop or mobile devices, you may select from the suggested Redirect URIs below or enter a custom redirect URI above. For more information, refer to the library documentation.

- ☐ msalb3b0f735-55ff-4f99-9a24-f5cb55d57d5d://auth (MSAL only) 
- ☐ urn:ietf:wg:oauth:2.0:oob 
- ☐ https://login.microsoftonline.com/common/oauth2/nativeclient 
- ☐ https://login.live.com/oauth20_desktop.srf (LiveSDK) 

Advanced settings

Logout URL 

Implicit grant

Allows an application to request a token directly from the authorization endpoint. Recommended only if the application has a single page architecture (SPA), has no backend components, or invokes a Web API via JavaScript.

To enable the implicit grant flow, select the tokens you would like to be issued by the authorization endpoint:

- ☐ Access tokens
- ☒ ID tokens

5) Verify manually in the Manifest that:

"signInAudience": "AzureADandPersonalMicrosoftAccount",

6) Open **nucoris.webapp appsettings.json** and:

- a. Set nucoris/AuthenticationMode to "AzureAd"
- b. Edit the AzureAd section:
 - i. Entry "Domain": set it to the value of the "identifierUri" entry in the manifest.
 - ii. Entry "ClientId": set it to the application id value of the app registration.