Hands-on : C2C Recommendation

Contacts :

damien.desesquelle@amaris.com
francois.chabannes@amaris.com

Project version: 2.0

Keywords: Azure Event Hub, CosmosDB, Text Embedding, Graph, Streaming data

Github template repository: https://github.com/amaris-bootcamp/C2C-article-recommendation-bootcamp

The repository C2C-article-recommendation contains the full solution and you shouldn't have access to it. You should clone the template and create your own repository C2C-article-recommendation-{yourname}

Important note: this hand-on is a collaborative project. The content and solution provided in this version are not perfect and you may find better ways to do things! Let us know if you brought improvements or explored new ways of doing things and we will use your Github branch to update the main branch of the solution repository that and will make a new version of this project.

You can already clone the template introduced above and create a new repository named C2C-article-recommendation-{yourname}. If you don't have the permissions to create your repository, contact us and we will do it for you.

Introduction and context

You work for a news company and you have been asked to develop a content to content recommendation system. The objective is that when users finish reading an article on our website, they are suggested similar articles to the one they just read. Hence, your objective is to develop a solution that computes the similarities between our news articles, and updates a graph database to store the relationships between news articles.

In this project, two micro-services will be developed. A first one will scrap the articles of a particular website. At the moment we have two scrapers for the BBC and Le Monde, feel free to use which ever you prefer. This micro-service will next send the articles to an Azure Event Hub. A second micro-service will be triggered when a news article arrive on the Azure Event Hub. This micro-service will read the streaming articles and will vectorize them. It will also compute the similarities between the incoming articles and the existing ones in the CosmosDB database and will insert the new articles in the graph database.

To orchestrate those two micro-services, we will first use Azure Functions (AZF), which is a serverless compute service that enables you to build, deploy and scale event-driven microservices.
At the end of this hands on, we are inviting you to explore how to set up a CI/CD pipeline using the Deployment Center of Azure Functions or even using Azure Kubernetes Service (AKS) instead. The next version of this hands on will address this in details.

Good to know

The goal of the content-based method is to develop a measure of similarity between items so that items that are similar to items that were liked or consumed by the target-user can be recommended.

The overall method is to:

- For each item, extract text information about the content (e.g.: item description, item name, name of the writer or producer in case it's a book or a movie, etc.). Depending on the use-case, we can vectorize the meta-data, the content itself, or both. In our use-case we will concatenate the meta-data and content, and vectorize it.
- Convert items into vectors by encoding the descriptive text using techniques such as Bag of Words, TF-IDF or embedding techniques such as word2vect, doc2vect or any other more sophisticated text embedding models.
- Compute the similarity between all the items using, for example cosine similarity.
- Recommend the N most similar items to the ones already used or liked by the user.

Advantages:

- When a new item is created (e.g.: a new article is published), it can directly be recommended to users if it shares many common features (key words, topics, entities…) with items that were previously consumed or liked by users. Hence, it is unaffected by the cold-start problem. Another advantage of the content-based recommendation is that it is much easier to explain than collaborative filtering. It is possible to know why a certain recommendation was made, because it uses feature models and does not try to infer hidden user preferences.

Drawbacks:

- However, they suffer from overspecialization when users begin to receive only recommendations similar to items they have already encountered and will not be recommended something new that they might like. The major drawback of content-based recommendation is the need for in-depth knowledge and description of the characteristics of the features in the items, which is not always easy to obtain (e.g.: video contents or retail products with no meta-data).

Relevant orders of magnitudes for a news company in Belgium:

- Number of articles in database: 300K – 400K articles. Articles don't have all the same lifetime (weather articles won't stay for as long as cultural articles).
- Number of new articles per day: 200 – 400.
- Number of customers: 3M subscribers.
- Similar articles have similarity values roughly around 0.5. Above 0.8 two items are very similar and could contain redundant information.

Scrapping the news articles

Two website scrapers have been developed : one for the BBC (bbc_scraper.py) and one for Le Monde (le_monde_scraper.py). Choose one and get familiar with how it works. Those scrapers both return a list of articles (dictionaries). Those dictionaries have keys:

- title
- publishedAt (publish date)
- section
- URL
- description
- content

Feel free to develop a new scraper with you want to work with another website, but its method scrape_articles() must return a list of dictionaries with keys shown above.

In the next steps, we will send those news articles to an Azure Event Hub.

Creating Resources

Your local work directory

After cloning the template, create your own repository as explained in the introduction. It's recommended to use Visual Studio Code for this project. You can also start from scratch without using the template.

Regarding Git, you should set up git in order to have a master and a development/feature branch.

Azure Functions

Azure Functions is a serverless compute service provided by Microsoft Azure that allows you to run event-triggered code in a serverless environment without the need to manage infrastructure. It lets you execute small, single-purpose functions in response to various events, such as HTTP requests, timers, or external triggers, making it suitable for building scalable and cost-effective applications.

Useful links:

- https://www.techtarget.com/searchcloudcomputing/definition/Microsoft-Azure-Functions#:~:text=Azure%20Functions%20is%20a%20serverless,and%20perform%20their%20own%20development.
- https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-python?pivots=python-mode-decorators

1 – Create a new Function App hands-on-C2C-{yourname} in the subscription E5-Business-AMAVLA using the resource group rg-datalab-hands and a python runtime stack. Our solution was made using a **Python 3.8** runtime, so unless you want to do the project without using the files provided and the requirements.txt, you should use Python 3.8 too. Make sure to enable public access, otherwise you won't be able to deploy the code of your Function App from VSC.

If you don't have access to this resource group, just let us know and we will quickly add you!

Azure Event Hub

Azure Event Hubs is a scalable and fully managed real-time data streaming platform provided by Microsoft Azure, designed to ingest, process, and analyze large volumes of event data from various sources, such as IoT devices, applications, and services. It enables you to build event-driven, distributed applications and allows seamless integration with other Azure services and third-party analytics tools.

Useful links:

- https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-create
- https://www.serverless360.com/azure-event-hub

1 – Go to the Azure Event Hub **event-hub-hands-on** and create a new Event Hub article-generation-{yourname}. We won't want to keep the articles on the Azure Event Hub for more than a day.

2 – What is the parameter partition count ? Taking into account that our only consumer will be an Azure Function, which we won't handle high volumes and that we want to minimize costs, what should be the value for that parameter ?

Azure CosmosDB

Azure Cosmos DB is a globally distributed, multi-model database service that enables you to build highly responsive and scalable applications with support for various data models like document, key-value, graph, and column-family.

Cosmos DB Gremlin is a component of Azure Cosmos DB that provides support for the Gremlin graph query language, allowing developers to work with and query graph data in Cosmos DB to model complex relationships and analyze interconnected data.

Useful links:

- https://learn.microsoft.com/en-us/azure/cosmos-db/gremlin/introduction

1 – Go to the CosmosDB account **cosmosdb-amavla-recommendation**.

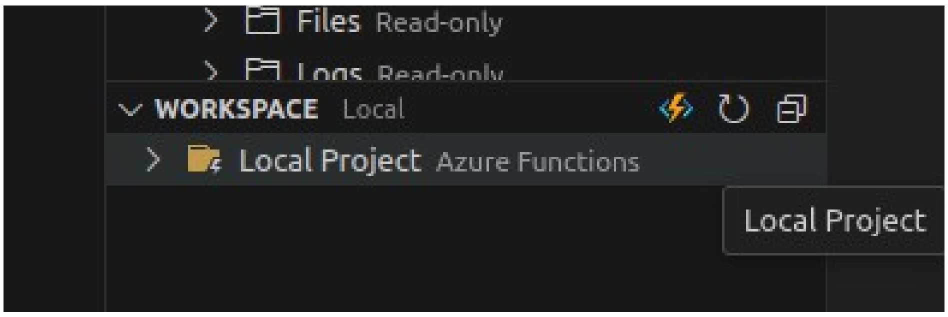
2 – Create a new database *news-recommendation-{yourname}*. Inside this database, create a new graph *news-articles*, When creating your graph, you will have to choose a partition key. A partition key must be one of the properties of the *article* vertices ('title', 'publishedAt', 'description', 'content'). It determines how data is distributed across physical partitions in CosmosDB. It's usually recommended to use a property that:

- has high cardinality
- is often used in *where* and *group by* in queries
- is not skewed or unevenly distributed

Sending news articles to Azure Event Hub with Azure Functions

1 – On your Azure Function App hands-on-C2C-{yourname} you can find on the tab Overview some documentation about how to create Azure Functions. It's recommended to use VS Code to develop and deploy Azure Functions. You will need to install the extension Azure Function on your VSC.

From VSC, once on the Azure tab, click on the trigger next to *Workspace* and deploy to function app (or create a new project, if you didn't clone the template and start from scratch). You will have to choose your Function App in the drop down menu and choose the Python interpreter - 3.8 in our case.



If you are facing issues when deploying in the way described above, another way to deploy your Function App to Azure is to run in the terminal of VSC:

```
> az login
> func azure functionapp publish {your function app name} --build remote
```

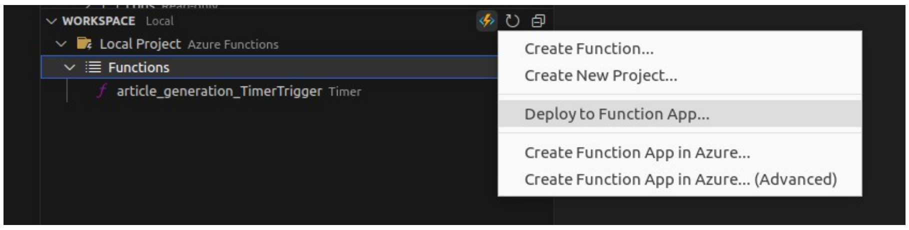
Tips:
- Make sure to run the build command from inside the Azure Functions folder.
- In case you have the error "Server Response: Run-From-Zip is set to a remote URL using WEBSITE_RUN_FROM_PACKAGE or WEBSITE_USE_ZIP app setting". Then you should go on your Function App > Settings > Configuration > and create a new Application setting "WEBSITE_RUN_FROM_PACKAGE" with value 0. it allows your function app to directly run the code from its deployed files rather than from a remote URL or ZIP package.

2 – If you are not using the template and are starting from scratch, create a new function within this project and select a timer trigger template and schedule it to run every 12 hours (for debugging purposes, you might want to use a faster scheduling at the beginning). In the subject, we named our function *article_generation_TimerTrigger*.

You should now be able to observe the new folder and files created in the folder Azure Functions:

- Azure Functions
    - host.json : Specifies configuration settings for the Azure Functions host, such as function timeout, host version, and extension bundle details. In our context it is used to specify how to process events from the Event Hub, including batch size, checkpoint frequency etc, to control event processing behavior.
    - requirements.txt : Lists the Python packages and their versions required by your Azure Function to ensure the correct environment is set up when deploying or running the function.
    - local.settings.json : Contains local development settings and connection strings, enabling you to configure environment-specific variables for testing and debugging your Azure Functions locally. In this requirement we typically added what was necessary to import the text embedding model.
    - article_generation_TimerTrigger
        - function.json : Defines the metadata and configuration for an individual Azure Function, including trigger bindings, input and output bindings, and other function-specific settings.
        - __init__.py : The Python that contains the code for this Azure Function.

For now, this project exists only locally and you will later deploy it to your Azure Function App by doing so:


WORKSPACE Local
Local Project Azure Functions
Functions
article_generation_TimerTrigger Timer
Create Function...
Create New Project...
Deploy to Function App...
Create Function App in Azure...
Create Function App in Azure... (Advanced)

Important note:

Over time, you'll find that deploying your Function App to Azure is time consuming when you're trying to debug your code. Instead, you can quickly build and execute your Azure Function App by running locally :

```
> func start –-build --verbose
```

You will have to write your Application Settings locally in the file local.settings.json so that the function as the credentials needed to access the different Azure Services.

3 – Using the resources in the folder Azure Functions/article_generation_TimerTrigger, update the files:

- init.py : you have to define the connection to the Event Hub in order to define a Event Hub Producer Client. Then you have to create a batch and send it to the Event Hub. You can use the python file clean_article.py to clean the content of the articles or you can make this cleaning yourself. For now we can hardcore the credentials to the resources until the first deployment but we will hide them later in the hands-on.

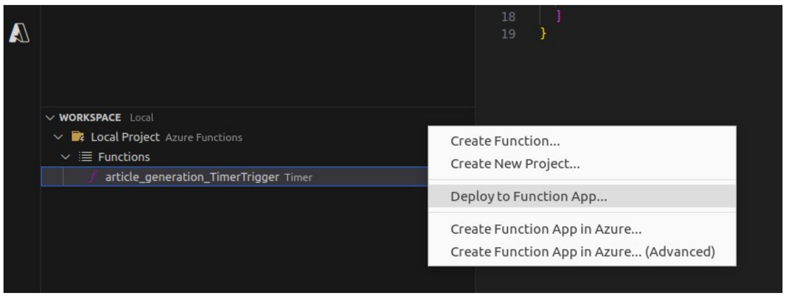
- function.json : our function will provide an output, without reading an input, hence we only have "direction" = "out". Update the connection settings for the Azure Event Hub that will receive the news articles. Careful : you are not supposed to hardcode variables in the function.json file. You must create those variables by declaring new Application Settings in the configuration tab of your Azure App Function. Once those application settings defined, you can use the variable names in your function.json file.

- requirements.txt : simply copy the one provided in the template. In AZF, the requirements.txt file is on the level of the Function App, not on the level of the function. Reminder: this requirements file work for Python 3.8.

- host.json : update the file so that the Azure Function runtime will:
    - Save its progress checkpoint after processing one event.
    - Process one event at a time.
    - Preload two events in advance for faster processing.

Tip : in the Azure Event Hub Namespace event-hub-hands-on, two shared access policies have been created : sender_policy and listen_policy. Use the sender_policy connection string for your function.json file (via the application settings).

4 – Deploy your function to your Azure Function App from VSC.


WORKSPACE Local
Local Project Azure Functions
Functions
article_generation_TimerTrigger Timer
Create Function...
Create New Project...
Deploy to Function App...
Create Function App in Azure...
Create Function App in Azure... (Advanced)

If the deployment is successful you should see your Time Trigger Function appear in *hands-on-C2C-yourname.*

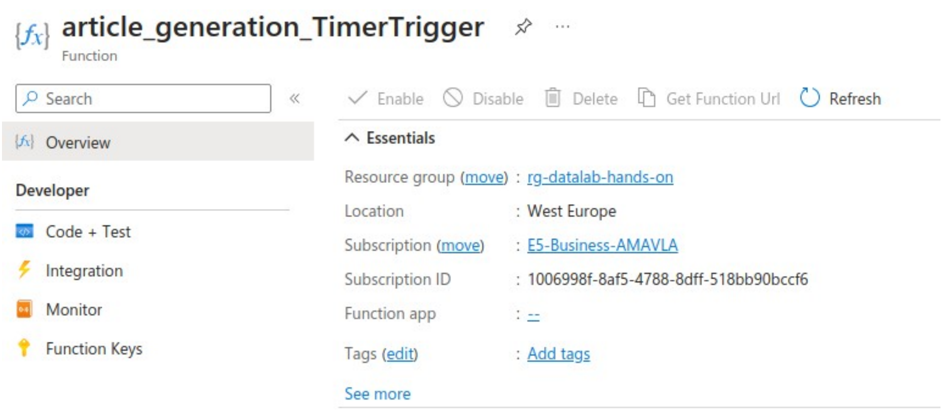
Once you've clicked on your function inside your Azure Function App, you can use the monitor tab to make sure the last runs were successful.
In case of errors, make sure the connection settings to the Azure Event Hub are correct.

Tips :

- Keep in mind that for debugging purpose, you can quickly build locally and test your code by running:
> func start --build --verbose

You will have to write your Application Settings locally in the file local.settings.json so that the function as the credentials needed to access the different Azure Services.

- Alternatively, if you want to do some debugging from the Azure portal, you can go on your Function App overview on Azure and click on your Azure Function:



Resource group (move)	: rg-datalab-hands-on
Location	: West Europe
Subscription (move)	: E5-Business-AMAVLA
Subscription ID	: 1006998f-8af5-4788-8dff-518bb90bccf6
Function app	: --
Tags (edit)	: Add tags

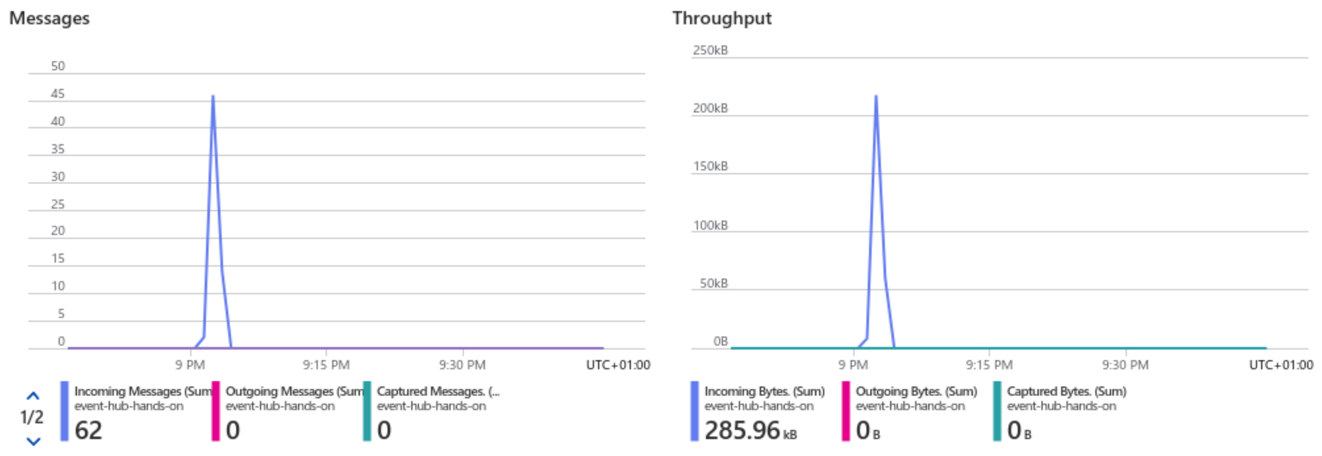
- In the tab Code + Test you will be able to trigger manually your function. You can also the live logs in the Logs tab.
- In the tab Monitor you can see status and logs of the last invocations.

So rather than wait for your scheduling to run your function, you can simply run it from the Code +Test tab and look at the logs in the Monitor tab.

- In case you get the error message "Deployment 'latest' not found" you should reset the publish profile on the overview tab of your Azure App Function.

5 – To avoid hardcoding the credentials of the Azure Event Hub in the init.py you can also call the variables declared in the Application Settings of your Azure Function App. You can call any variable set as application parameter in your Azure Function App by using os.environ.get(“myvar”).

6 – Make sure that the Azure Function is running like expected by looking at the logs (tab *monitor* in the Azure Function App) and by going on your Azure Event Hub. You should see messages arriving :



Incoming Messages (Sum) event-hub-hands-on 62
Outgoing Messages (Sum) event-hub-hands-on 0
Captured Messages. (... event-hub-hands-on 0
Incoming Bytes. (Sum) event-hub-hands-on 285.96 kB
Outgoing Bytes. (Sum) event-hub-hands-on 0 B
Captured Bytes. (Sum) event-hub-hands-on 0 B

At the moment, there are no outgoing messages, since we are not reading from this Azure Event Hub yet.

Update the CosmosDB database with Azure Functions

1 – If you are not using the template, you now have to create a new function from VSC (by clicking on the trigger sign next to Workspace, once on the Azure tab) and use a Azure Event Hub trigger template. We named our function article_similarity_EventHubTrigger. If you cloned the template you can directly update the files and deploy to the Function App.

2 – Using the code inside Azure Function/article_similarity_EventHubTrigger, update the code of this new Azure Function.

The overall purpose of this python script is:

- To receive an incoming article and extract its content into a dictionary in the *main()* function.
- To apply the function *insert_article()* to this new article converted to a dictionary. This function will:
    - Check that the arriving article is not already in the CosmosDB graph.
    - Query the recent articles, as we don't want to compute the similarity between all articles if the database gets big. Let's say we consider the last 200 articles.
    - Iterate through all those selected articles in the graph and compute their similarity with the arriving article. The similarity will be based on the concatenation of the title, description and content of the article. Insert the new article by specifying the similarity relationship between each articles. We will create a similarity relationship only if the similarity score is higher than 0.4.

Now you have to:

- Update the function.json and the beginning of init.py that reads the database names and connection strings, by updating the Application Settings.
- Complete the *main()* function so that you extract the incoming article and parse it into a dictionary.

  Notes :

    - The use of *async def* allows for non-blocking I/O operations, enabling efficient and concurrent handling of incoming articles. In short, the program works faster and doesn't ignore new articles while it's busy with others.
    - We are using *await insert_article().* The *await* keyword in this function is used to tell the program to wait for the *insert_article* operation to finish before moving on to the next event. It ensures that the program doesn't continue

processing events until the current article has been inserted into the graph database.

- Before the main() function, we are using nest_asyncio.apply(). It allows you to handle asynchronous event-driven operations, such as processing incoming articles concurrently, without interfering with the event loop used by the Azure SDK for Event Hubs. Without it, you would face issues with event handling and potential blocking of other tasks.

- The input parameter *events* is a list containing the events. The length of *events* can be set in the function.json file via the parameter maxBatchSize.

  - Small Batch:
    - Pros: Low latency, resource-efficient.
    - Cons: Increased processing overhead, higher network traffic.
  - Big Batch:
    - Pros: Efficient processing, reduced network overhead.
    - Cons: Higher latency, resource-intensive.

- Complete the function compute_similarity in the file compute_similarity.py. The similarity should be based on the concatenation of the titles, descriptions and content of the two input articles. The model used is from the HuggingFace platform. All details can be found : https://huggingface.co/tasks/sentence-similarity

- Complete the function *insert_articles().*

  - Create a Gremlin client *gremlin_client* and connect to the Cosmos DB graph.

  - Using gremlin_client.submit(), check that the incoming article is not already in the database. In our case we will use the URL of an article as its identifier.

  - Execute a gremlin query that create a vertex for the new article with the properties: title, publishedAt, section, URL, description, content.

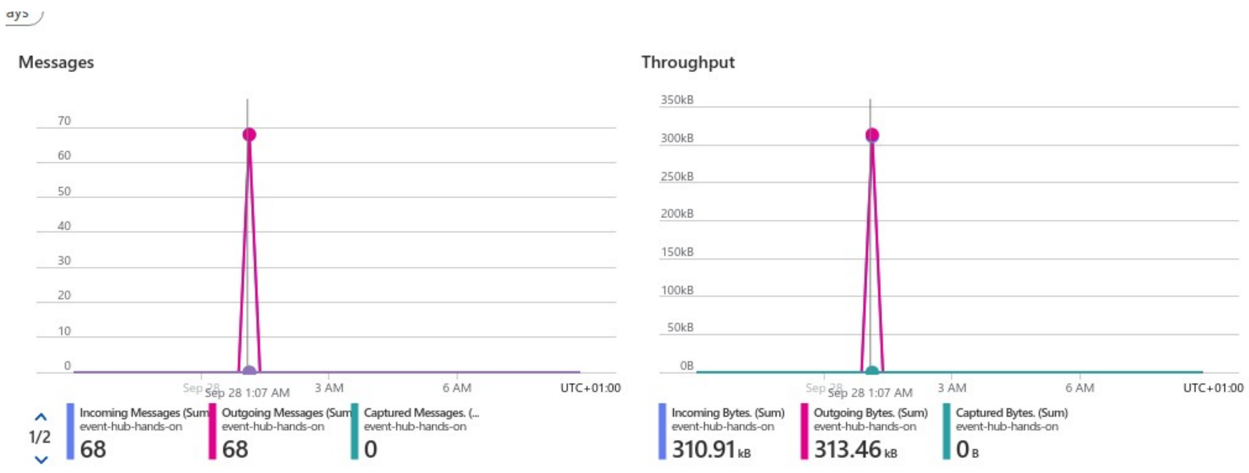
  - Submit a gremlin query that fetch the most recent 200 articles. You can extract all the URLs of those article into a list.

  - Iterate through this list of article URLs. For each article URL:
    - Compute the similarity between this article and new article.
    - If the similarity value is superior than the predefined threshold (~0.4), run a gremlin query to create a new similarity relationship between the two articles.

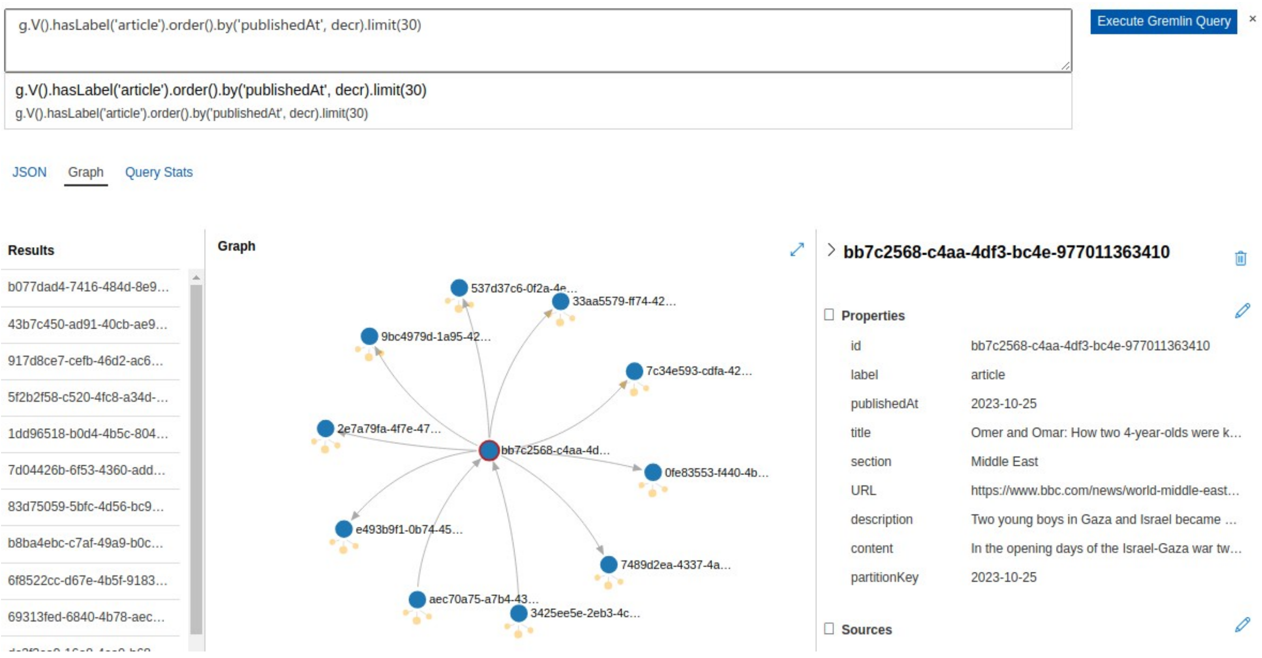
Tip:

- As explained for the first Azure Function, and easy way to debug your code is to manually trigger the first Azure Function by going in the Code + Test tab. Simultaneously, you can go on the Monitor tab of your event-hub-trigger Azure Function and look at the live logs.

3 – Make sure that there are now outgoing messages leaving your Azure Event Hub. Go to your CosmosDB graph and make sure you are successfully inserting articles.



The first articles won't have similarities relationships as you won't have enough history yet. After a few runs of the function article_similarity_EventHubTrigger, you should start seeing similarity relationships being created between articles:



```
g.V().hasLabel('article').order().by('publishedAt', decr).limit(30)
```

bb7c2568-c4aa-4df3-bc4e-977011363410

Properties

id	bb7c2568-c4aa-4df3-bc4e-977011363410
label	article
publishedAt	2023-10-25
title	Omer and Omar: How two 4-year-olds were k...
section	Middle East
URL	https://www.bbc.com/news/world-middle-east...
description	Two young boys in Gaza and Israel became ...
content	In the opening days of the Israel-Gaza war tw...
partitionKey	2023-10-25

Sources

4 – Once you get a few days of history, get familiar with the graph database and the gremlin querying language by querying:

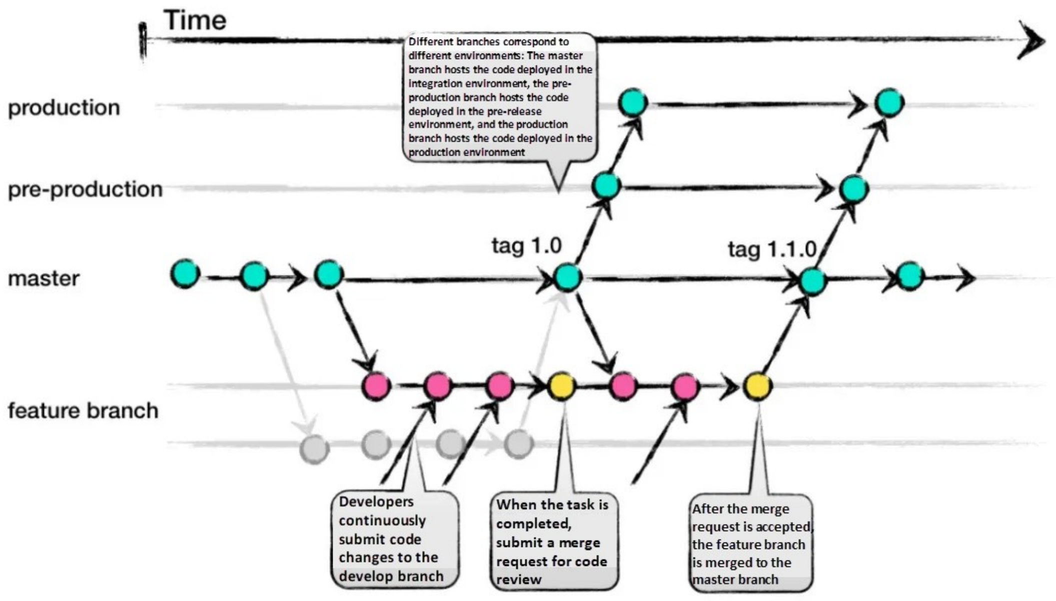
- All the articles with "war" in the description.

- The count of articles per day.
- All the articles linked by a similarity relationship that is higher than 0.7.
- The count of articles per section.

CI / CD pipelines in Azure Dev Ops and Azure Functions

Continuous Integration (CI) is the practice of automatically integrating code changes from multiple developers into a shared repository, typically using feature branches, to detect and resolve integration issues early. Continuous Deployment (CD) is the process of automating the deployment of code to production after successful testing and integration, ensuring that code changes are quickly and safely delivered to end-users.

In a “real” situation, we would use the following branch strategy for CI/CD:


Time
Different branches correspond to different environments: The master branch hosts the code deployed in the integration environment, the pre-production branch hosts the code deployed in the pre-release environment, and the production branch hosts the code deployed in the production environment
production
pre-production
master
tag 1.0
tag 1.1.0
feature branch
Developers continuously submit code changes to the develop branch
When the task is completed, submit a merge request for code review
After the merge request is accepted, the feature branch is merged to the master branch

Because we have limited resources in this hands-on, we can’t duplicate every single Azure resource into a pre-production and a production resource. Hence, we will use our master branch as our production branch.

Normally, the master branch should be protected to avoid that someone directly push a broken version of the code on this production branch. Because we have a free

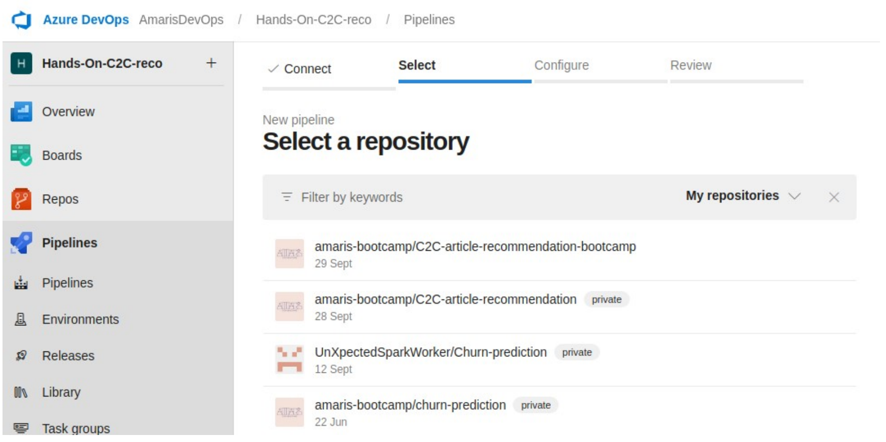
GitHub subscription we can't protect the branch is this hands-on but keep in mind that we should. From now on -if it wasn't the case before- only work on develop branch and update the master branch only by making Pull Requests.

Continuous Integration (CI) pipeline

In this Continuous Integration section, we want to test the code when there is a pull request from the development branch to the master branch. In other words, we want to make sure that the new version of the code is not going to fail in production. If the tests fail, the pull request will fail and the master branch will not be updated.

1 – In the folder /tests, you can find two python files. Those files perform tests on the functions used in our two Azure Functions, using unittest.
Go through the code of those simple tests and make sure that they work correctly locally. You can also develop more tests if you want.

2 – Go to our C2C recommendation project on our Azure Dev Ops organization https://dev.azure.com/AmarisDevOps/Hands-On-C2C-reco/, and go to the Pipelines tab. Create a new pipeline and choose the GitHub as code location. Make sure to choose your repository.

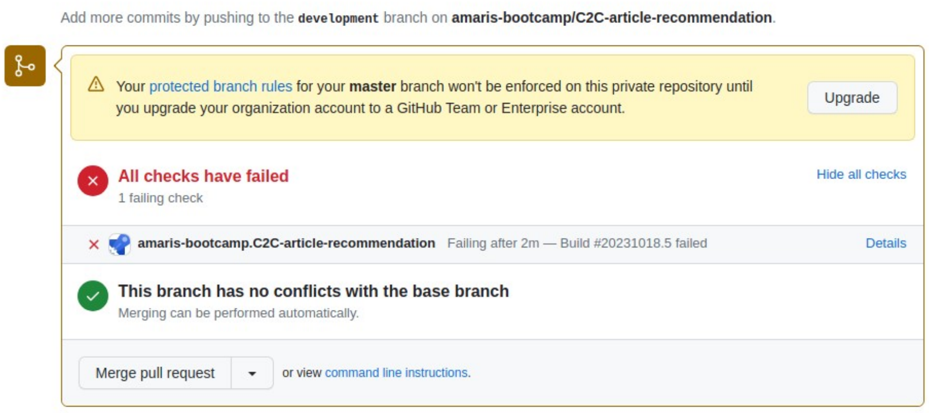


Choose the default "Starter Pipeline" template. Adapt this file so that

- The packages in the requirements.txt can be installed
- Run the tests inside the /test folder
- Making sure this pipeline runs prior any Pull Request on master.

You can use the file azure-CI-pipeline.yml as a template.

Once your yml file configured, go your local development branch, make some changes that would cause the tests to fail, and ensure that initiating a Pull Request on your GitHub repository results in failure like shown below:


Add more commits by pushing to the `development` branch on **amaris-bootcamp/C2C-article-recommendation**.

Your protected branch rules for your **master** branch won't be enforced on this private repository until you upgrade your organization account to a GitHub Team or Enterprise account.

All checks have failed
1 failing check

amaris-bootcamp.C2C-article-recommendation Failing after 2m — Build #20231018.5 failed

This branch has no conflicts with the base branch
Merging can be performed automatically.

Continuous Deployment (CD) pipeline

1 – Go back to our C2C recommendation project on our Azure Dev Ops organization https://dev.azure.com/AmarisDevOps/Hands-On-C2C-reco/, and go to the Pipelines tab. Again, create a new pipeline and choose the GitHub as code location. Make sure to choose your repository. For template, you can use the same “Starter Pipeline” or the “Python Function App to Linux on Azure”. This second template will already give you a specific template for deploying your Azure Function App.

2 – Configure this yml file so that the Function App is deployed every time there is a Pull Request on the Master branch. You can use the file azure-CD-pipeline.yml as a template if you need help.

Check that after a Pull Request, the code of your Azure Function App in Azure is automatically updated (you can look at the Activity log tab of your App Function).

Improving the deployment of the project

You are reaching the end of the guided path ! Time for you to explore on your own with more autonomy. Please let us know of your progress and we will update this pdf and our solution branch based on your findings.

Potential next steps for this project are related to the deployment of the solution:

1 – Improve the current CI/CD approach on Azure DevOps.

2 – Instead of using Azure Functions, use Azure Kubernetes Service (AKS), create two microservices for the article generation and article insertion in the graph database and create CI/CD pipeline.

Go to our project on our Azure Dev Ops organization : https://dev.azure.com/AmarisDevOps/Hands-On-C2C-reco/

Go to Project Settings (bottom left of the screen) and add your github repository if it’s not already here