UNIVERSITY OF TARTU
Institute of Computer Science

(/)

Courses (/) » 2018/19 spring (/courses/index/2019/spring)
» Basics of Cloud Computing (MTAT.08.027) (/2019/cloud/spring)

Shefali Naresh Emmanuel ▼

(/user/lang/et?

userlang=et&redirect=%2F2019%2Fclou

Basics of Cloud Computing 2018/19 spring

Cus

- Main (https://courses.cs.ut.ee/2019/cloud/spring/Main/HomePage)
- Lectures (https://courses.cs.ut.ee/2019/cloud/spring/Main/Lectures)
- Practicals (https://courses.cs.ut.ee/2019/cloud/spring/Main/Practicals)
- Results (https://courses.cs.ut.ee/2019/cloud/spring/Main/Results)
- Submit Homework (https://courses.cs.ut.ee/2019/cloud/spring/Main/Homework)

Practice 8 - Cloud Functions

In this lab we will use the IBM Bluemix Cloud and take a look at the **Cloud Function** service, which can be considered a **Function as a Service** (FaaS) or **Serverless** platform and is based on the **Apache OpenWhisk** open source project.

In **FaaS** Coud Computing model your application consists of a set of functions or microservices. **Cloud Functions** are not constantly running in the background. They are executed only when specific user specified **events** are triggered (e.g. when new file is uploaded, new entry is added to the database, new message arrives, etc.).

You will make an account in **IBM Bluemix** cloud and set up multiple **Python Cloud Functions** which are automatically triggered on either web requests or database events. You will also use the **Cloudant** NoSQL service as the database for your **Cloud Functions**. **Cloudant** is based on the **CouchDB** open source NoSQL **JSON** document database.

Additional materials, tutorials and references

- IBM bluemix - https://www.ibm.com/cloud-computing/bluemix/node/4471 (https://www.ibm.com/cloud-computing/bluemix/node/4471)
- IBM Cloudant documentation - https://cloud.ibm.com/docs/services/Cloudant (https://cloud.ibm.com/docs/services/Cloudant)
- Cloudant Python tutorial - http://python-cloudant.readthedocs.io/en/latest/getting_started.html (http://python-cloudant.readthedocs.io/en/latest/getting_started.html)
- IBM Cloud Functions - https://console.bluemix.net/docs/openwhisk/index.html (https://console.bluemix.net/docs/openwhisk/index.html)
- Introduction to JSON - https://www.digitalocean.com/community/tutorials/an-introduction-to-json (https://www.digitalocean.com/community/tutorials/an-introduction-to-json)

Exercise 8.1. Creating an IBM Bluemix Account

- Sign up for a free IBM Bluemix account at https://www.ibm.com/cloud-computing/bluemix/open-source (https://www.ibm.com/cloud-computing/bluemix/open-source)
- Activate the account using the link sent by email and log in (verify spam folder or `Gmail` users

check `Promotions` tab).
- Familiarize yourself with the available Cloud services.

Exercise 8.2. Creating your first Cloud Function

- Go to the IBM Cloud Functions console at https://console.bluemix.net/openwhisk/ (https://console.bluemix.net/openwhisk/)
- Create a new Python Cloud Function
  - Click on `Start Creating` and `Create Action`
  - If you get and error `No Cloud Foundry Space` choose your `Region` (London), `Cloud Foundry ORG` (your username), `Cloud Foundry Space` (dev)
  - Assign a freely chosen `Action name` to the new action
  - Leave the package as *(Default Package)*
  - Choose `Python 3` as the `Runtime`
  - As a result, a simple *Hello World* function will be generated for you.
- Click `Invoke` to execute your function to test it.
  - This will execute your Function in the server without any arguments. You can specify default arguments (JSON document) by using `Change Input` link, which can be useful for testing your Cloud Functions.
- You should notice that the result of your function is in JSON format.
  - Input and output of IBM Cloud Functions is JSON by default
  - JSON can be described as a dictionary data structure in JavaScript notation.
  - In Python, JSON objects can also be manipulated as Python dictionaries.
  - If you have never used JSON before, then read the JSON tutorial here: https://www.digitalocean.com/community/tutorials/an-introduction-to-json (https://www.digitalocean.com/community/tutorials/an-introduction-to-json)
- Modify the function output message so it returns custom message for you instead of default "Hello World"

Exercise 8.2. Setting up Cloudant Database

We are going to use Cloudant NoSQL database in the following exercises, but first we need to set up a new database instance.

- Go to https://console.bluemix.net/catalog/services/cloudant-nosql-db (https://console.bluemix.net/catalog/services/cloudant-nosql-db) and create a **new (free) Lite `Cloudant` NoSQL database**.
  - `Service name` - choose freely
  - `Available authentication methods:` - `Use both legacy and IAM`
- The database will be accessible from your Bluemix Dashboard: https://console.bluemix.net/dashboard/apps (https://console.bluemix.net/dashboard/apps)
- After creating new `Cloudant database` wait few minutes till it finishes creating and then click o its name to open @Cloudant configuration options.
- Create new database credentials
  - Go to `Cloudant` -> `Service Credentials` -> `New credential` and assign names for your database credentials.
  - You can view the content of your credentials using the **View credentials**
  - NB! You will need these database credentials in the next exercise.
- Cloudant is a NoSQL document database. Entries in the database are JSON documents.
- Open the dashboard of your Cloudant database. Go to `Cloudant` -> `Manage –>` Launch Cloudant Dashboard@@
  - Create a new database: `labdb1`.
  - Create a new JSON document in your database:

        - Add two new fields `user` and `message` to the document.
        - The document should look something like this (Do not don't modify/overwrite the generated `"_id"` field):
            - ```
              {
                "_id": "...",
                "user": "Martin",
                "message": "Hello World!"
              }
              ```
    - Check the content of the created document.
        - `"_id"` and `"_rev"` fields will be automatically generated for each JSON document.

Exercise 8.3. Creating Cloud Function for posting documents to Cloudant database service

In this exercise, we will use a Cloud Function to create a simple web service for submitting data into the Cloudant database.

It is important to note that input to the Cloud Function will be the **content of the event** it was triggered on. In case of web requests, input will be the HTML request parameters (e.g. headers, body, html form fields). In case of Cloudent, it is the id of the database document and event type. When you execute the Function directly through the browser, input is usually an empty JSON document.

8.3.1 Create another Cloud Function
We will modify the function to create a new document in `labdb1` database every time the Function is executed.

- Add a new method for putting documents into your database:
    - First import Cloudant API
        - `from cloudant.client import Cloudant`
    - Then add a new `addDocToDB(new_doc, username, apikey)` method to your Cloud Function code
    - ```
      def addDocToDB(new_doc, username, apikey):
          databaseName = "labdb1"
          client = Cloudant.iam(username, apikey, connect=True)
          myDatabase = client[databaseName]
          return myDatabase.create_document(new_doc)
      ```
    - Specify database credentials as additional Cloud Function parameters:
        - Go to the **Parameters** page under your Function.
        - Add 2 new parameters with correct values based on your own database credentials.
            - `username` - Your Cloudant credentials username
            - `apikey` - Your Cloudant credentials Api Key
        - Example of how parameters should look like is provided here: parameters.png (https://courses.cs.ut.ee/MTAT.08.027/2019_spring/uploads/Main/parameters.png)
        - Now these parameters will be added to the input (document) of your Cloud Function and these values can be accessed from inside the function code.
        - This allows us to specify database credentials without hardcoding them into our function code.

8.3.2 Add a web endpoint to your cloud function.
This will make your function publicly accessible from the internet.

- Create a new Web action endpoint
  - Go to `Endpoints -> Enable as Web Action`
  - This will generate a public web URL for your function that can be accessed from anywhere in the web.
  - NB! You will need this URL in the next step.

8.3.3 Submitting data to the Cloud Function

Lets now create a local html page for submitting data to your Cloud Function

- Create a new html file in your computer:
- It should contain the following content:
  - ```
    <html>
      <body>
        <form id="form_1" method="post" action="https://CLOUD_FUNCTION_ENDPOINT">
          User: <input id="user" name="user"  size="30" type="text" value=""/> <br />
          Message: <textarea  rows="10" cols="30" id="message" name="message" type="text" value=""></textarea> <br />
          <input id="saveForm" type="submit" name="submit" value="Send message" />
        </form>
      </body>
    </html>
    ```
  - Replace `https://CLOUD_FUNCTION_ENDPOINT (https://CLOUD_FUNCTION_ENDPOINT)` with the real endpoint URL you previously created.
- As a result, you will have a HTML form which submits **user** and **message** fields through a HTML POST request to your web endpoint.
  - Save the html file in your computer and open it with a browser.

8.3.4 Modify the Cloud Function to create new documents and save them to the database

Modify Your Cloud Function to read the **user** and **message** fields sent from the HTML form and submit a new JSON document containing the values of these fields into the Cloudent database.

- Read the **user** and **message** field values from the input document:
- ```
  def main(param):
      user = ""
      if 'user' in param:
          user = param['user']

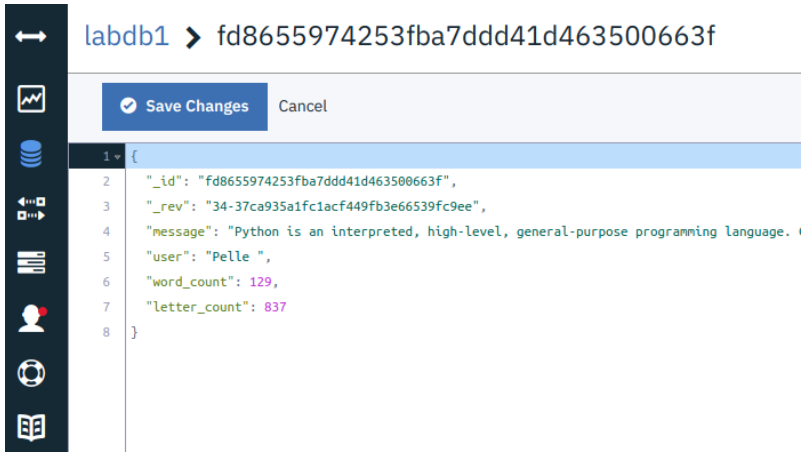
      message = ""
      if 'message' in param:
          message = param['message']
  ```
- Create a new JSON document that contains these two fields
  - `new_doc = {'message': message, 'user': user}`
- Use the `addDocToDB(new_doc, username, apikey)` method inside your functions `main(param)` method to add a new document to the database:
  - `modified_doc = addDocToDB(new_doc, param['username'], param['apikey'])`
  - username and apikey will be read from the additional parameters you specified for the Action.
- Return the document object at the end of `main` method for ease of debugging:
  - `return modified_doc`
- Save your Cloud Function
- Verify that a new document is created in a database every time a new value is submitted through the form.

Exercise 8.4. Creating a Cloud Function for automatically modifying new documents in the database

Cloud Functions can also be used to automate tasks. We will now create a Cloud Function that is automatically executed for every new document added to the database and which counts how many words and letters the message contained and adds this information as new fields into the the document

- Create a new Python Cloud Function.
  - Go to `Actions` -> `Create` -> `Create Action`
  - Assign `lab8dbTriggerFunction` as name to the new function
- Add a Cloudant trigger to your Function.
  - Go to `Connected Triggers` -> `Add Trigger` -> `Cloudant`
  - Assign `lab8dbTrigger` as the name of the trigger
  - Choose your database instance under `Cloudant Instance`
    - If you do not see the Cloudant instance in the list, you will have to configure its location manually by choosing `Input your own credentials` and then specifying `Username`, `Host`, `Database` and `iamApiKey`.
    - You will find correct values for those parameters from `Cloudant` -> `Service Credentials` -> `View credentials`. All values should be **without** quotation (`"`) marks.
    - Make sure the database is `labdb1`
  - Click **Create & Connect**
  - Now your Function will be automatically executed every time there is a new entry added to the `labdb1` database and it gets the **id** of the document as one of the `param` values as input.
- Modify the **lab8dbTriggerFunction** function to compute both count of letters and words in the document message field
  - Add import statements for Cloudant and time:
    - ```
      from datetime import datetime
      from cloudant.client import Cloudant
      ```
  - Add a new method to fetch the database object based on its id
    - ```
      def getDBdoc(doc_id, username, apikey):
          databaseName = "labdb1"
          client = Cloudant.iam(username, apikey, connect=True)
          myDatabase = client[databaseName]
          db_doc = myDatabase[doc_id]
          return  db_doc
      ```
    - Its arguments are document id, Cloudant username and Cloudant api key.
  - Modify the Cloud Function parameters to specify Cloudant `username` and `api key` (Just like in the **Exercise 8.3**)
  - Modify the `def main(param):` method to fetch the database object based on its id:
    - `doc = getDBdoc(param['id'], param['username'], param['apikey'])`
    - Id of the modified document will be passed as one of the fields inside the input `param` object when the Cloudant document modification even is Triggered.
    - However, param object does not contain the rest of the content of the document, which is why we have to use `getDBdoc` method to fetch it.
  - Calculate letter and word count based on the document message field (`doc['message']`).

- Modify the document by adding new attributes **word_count** and **letter_count** into it
  - `doc['word_count'] = ...`
- Save the modified document:
  - `doc.save()`
- Save your Cloud Function
- Test that the function works by creating new documents in `labdb1` database and verifying that **word_count** and **letter_count** attributes are automatically generated for each of them.
  - The result should look something like:



- Modify the function, so that it only computes **word_count** and **letter_count** once.
  - Our trigger function will be launched every time document is modified. To avoid recursive function triggering, make sure that **word_count** and **letter_count** are only computed once.
  - Simple way to achieve it is to check whether document already contains **word_count** field and stop the execution of the function if it does:
    - ```
      doc = getDBdoc(param)
      if 'word_count' in doc:
         return doc
      ```
    - When using IBM Cloud Functions command line interface, it is possible to specify triggers that are only launched on new document creation events.

Deliverables

- Source code of your Cloud Functions from **Exercise 8.3** and **Exercise 8.4**.
  - NB! Do not leave your IBM Cloudant database credentials inside your function code or screenshots!!
- Provide the URL to the Cloud Function web endpoint, which you created in **Exercise 8.3**.
- Screenshot of your Cloudant `labdb1` database view, which should display one of the open documents.

Task	lab 8
Current submission	ZIP (/course-2065/submissions/get/8/B91541_zip/B91541_4312_1) (15:48 03.04.2019)

If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
      ```

File Choose File no file selected

Comment https://eu-gb.functions.cloud.ibm.com/api/v1/web/emmanuelsn%40g.cofc.edu_dev/default/shefaliInAction.json

Submit