```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "fb054efb-c141-45dd-9274-17d34933da90",
   "metadata": {},
   "source": [
    "# Assignment0 : Function as a Service (FaaS)\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "ea1a76b3-60bc-42ec-84a6-c44abc041d4d",
   "metadata": {},
   "source": [
    "## Description\n",
    "The objective of this assignment is to gain proficiency in building, deploying, and running code in a cloud environment using an event-driven serverless architecture. You will be setting up a cloud environment and creating, deploying, and testing a function-as-a-service on a cloud platform.\n",
    "\n",
    "To complete this assignment, you will begin by running and testing a Python program on your local machine and verifying the output. Next, you will proceed to deploy and test the same function on a cloud platform utilizing serverless services, and compare the results with those obtained from local execution. Please adhere to the provided instructions to finish the assignment."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "467b6084-5991-404a-95f7-941f9b18ea1b",
   "metadata": {},
   "source": [
    "## Run and test a python program on a local machine\n",
    "For simplicity, a python script has been provided for you in this assignment. The Python script example reads a paragraph of text, counts the number of occurrences of each word, and dumps the result in a JSON format.
"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "4f406f13-4fbd-4705-b493-8c807959c4e4",
```

```
   "metadata": {},
   "outputs": [],
   "source": [
    "import json\n",
    "from collections import Counter\n",
    "\n",
    "\n",
    "def word_count(paragraph):\n",
    "    \n",
    "    # Remove punctuation from input paragraph and convert to
lowercase\n",
    "    paragraph = paragraph.lower().replace('.', '').replace(',',
'')\n",
    "    \n",
    "    # Split the paragraph into a list of words\n",
    "    words = paragraph.split()\n",
    "\n",
    "    # Count the number of occurrences of each word\n",
    "   # word_counts = dict(Counter(words))\n",
    "    word_count = Counter(words)\n",
    "\n",
    "    # Sort the words in descending order of frequency\n",
    "    sorted_word_count = dict(sorted(word_count.items(), key=lambda
item: item[1], reverse=True))\n",
    "    print (sorted_word_count)\n",
    "    return sorted_word_count\n",
    "    "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "35e99626-7f06-4ad6-ba74-21ad1506924b",
   "metadata": {},
   "source": [
    "You can call this function from your local machine with a paragraph
string as follows:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "5c179a0f-5602-407b-bdda-9640389b7981",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
```

```
     "output_type": "stream",
     "text": [
      "{'a': 6, 'and': 6, 'to': 5, 'chatgpt': 3, 'language': 3, 'with': 3,
'knowledge': 3, 'in': 3, 'can': 2, 'of': 2, 'natural': 2, 'processing': 2,
'looking': 2, 'the': 2, 'or': 2, 'is': 1, 'cutting-edge': 1, 'model': 1,
'that': 1, 'perform': 1, 'wide': 1, 'range': 1, 'tasks': 1, 'remarkable':
1, 'accuracy': 1, 'fluency': 1, 'from': 1, 'answering': 1, 'general': 1,
'questions': 1, 'assisting': 1, 'research': 1, 'generating': 1,
'human-like': 1, 'text': 1, 'even': 1, 'engaging': 1, 'creative': 1,
'writing': 1, \"chatgpt's\": 1, 'capabilities': 1, 'are': 1, 'truly': 1,
'impressive': 1, 'its': 1, 'vast': 1, 'base': 1, 'ability': 1, 'learn': 1,
'adapt': 1, 'over': 1, 'time': 1, 'serve': 1, 'as': 1, 'valuable': 1,
'resource': 1, 'for': 1, 'anyone': 1, 'harness': 1, 'power': 1, 'their': 1,
'work': 1, 'personal': 1, 'projects': 1, 'whether': 1, \"you're\": 1,
'researcher': 1, 'writer': 1, 'business': 1, 'professional': 1, 'simply':
1, 'someone': 1, 'engage': 1, 'interesting': 1, 'conversations': 1, 'has':
1, 'skills': 1, 'help': 1, 'you': 1, 'achieve': 1, 'your': 1, 'goals':
1}\n",
      "the most frequent word is  a  and its frequency is  6\n"
     ]
    }
   ],
   "source": [
    "\n",
    "# Define the input paragraph\n",
    "paragraph = \"\"\"\n",
    "ChatGPT is a cutting-edge language model that can perform a wide range
of natural language processing tasks with remarkable accuracy and fluency.
From answering general knowledge questions and assisting with research, to
generating human-like text and even engaging in creative writing, ChatGPT's
capabilities are truly impressive. With its vast knowledge base and ability
to learn and adapt over time, ChatGPT can serve as a valuable resource for
anyone looking to harness the power of natural language processing in their
work or personal projects. Whether you're a researcher, a writer, a
business professional, or simply someone looking to engage in interesting
conversations, ChatGPT has the skills and knowledge to help you achieve
your goals.\"\"\"\n",
    "\n",
    "\n",
    "result = word_count(paragraph)\n",
    "\n",
    "print(\"the most frequent word is \", max(result, key=result.get), \"
and its frequency is \", result[max(result, key=result.get)])\n",
    " "
   ]
  },
```

```
  {
   "cell_type": "markdown",
   "id": "09993990-0d0a-4cff-8631-33032169cd62",
   "metadata": {},
   "source": [
    " - What are three most frequent words? \n",
    " - What are their frequency?\n",
    "\n",
    "\n",
    " "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "7c49cc93-2257-4b94-84de-559c007554ae",
   "metadata": {},
   "source": [
    "\n",
    "\n",
    " "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2e41d2c1-f18d-4aa9-bffc-aca592b195d2",
   "metadata": {},
   "source": [
    "\n",
    "## How to go serverless?\n",
    "Serverless computing is a cloud computing model that allows developers to build and run applications without having to manage servers or infrastructure. In a serverless architecture, the cloud provider is responsible for managing the underlying infrastructure, including servers, storage, and networking. Developers only need to write code for their applications and upload it to the serverless platform, which automatically handles the deployment, scaling, and management of the code. To deploy the above function as a serverless function, you need to walk through the following steps:\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "9500ce10-ab6b-467a-981f-fd3cf16184bc",
   "metadata": {},
   "source": [
    "### 1- Choose a Cloud platform: \n",
```

```
    "You have the option to select from the leading cloud providers: Amazon Web Services, Microsoft Azure, or Google Cloud Platform. Depending on your choice, you will be working with a serverless computing service like AWS Lambda, Azure Functions, or Google Cloud Functions to fulfill the requirements of this assignment.\n",
    " \n",
    "\n",
    "### 2- Sign up for an account: \n",
    " - Set up an account in one cloud platform (e.g., AWS). \n",
    " - Log in to your account (e.g., AWS management console) and navigate to FaaS service (e.g., Lambda)\n",
    "\n",
    "### 3- Create a new function: \n",
    " Click on the \"Create Function\" button and choose \"Author from scratch\". You can create a function in your preferred programming language, such as Python, Node.js or Java. Alternatively, you can use the above python script that has been provided for you. Give your function a name, select Python 3.8 as the runtime, and choose an appropriate execution role. \n",
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "a8f2b4f1-b928-4737-b703-3e38f44893b2",
   "metadata": {},
   "source": [
    "\n",
    "### 4- Justify the function code\n",
    " You need to apply slight modification to the function through event definition. \n",
    " - When you deploy a function to a serverless platform like AWS Lambda, the platform handles the invocation of the function by passing an event and a context object as arguments to the function. The event provides information about the trigger that caused the function to be invoked, such as an HTTP request, a message from a queue or an input data. The context object provides information about the runtime environment, such as the AWS region, function name, and time remaining before the function times out.In the context of our example function, the event represents the input paragraph string that the function processes. The context object isn't used in this particular function, but it's included in the function signature because it's a required argument for AWS Lambda functions.\n",
    " \n",
    " - In serverless computing, a serverless function typically sends its response back to the client or service that invoked it by returning a JSON object that includes a statusCode and a body. The statusCode indicates the
```

```
status of the response, such as whether the request was successful or not, and the body contains the response data in JSON format. In the example function, the sorted_word_count dictionary contains the word frequencies that the function has computed. To return this data to the client that invoked the function, we use the json.dumps() method to convert the dictionary to a JSON-formatted string and include it as the value of the body key in the response object. We set the statusCode to 200 to indicate that the request was successful.\n",
    "\n",
    "\n",
    " "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "4d4fd2db-5ff9-4a67-b884-2dd604262e8e",
   "metadata": {},
   "outputs": [],
   "source": [
    "import json\n",
    "from collections import Counter\n",
    "\n",
    "def lambda_handler(event, context):\n",
    "    def word_count(paragraph):\n",
    "        # Remove punctuation from input paragraph and convert to lowercase\n",
    "        paragraph = paragraph.lower().replace('.', '').replace(',', '')\n",
    "\n",
    "        # Split the paragraph into a list of words\n",
    "        words = paragraph.split()\n",
    "\n",
    "        # Count the number of occurrences of each word\n",
    "        word_count = Counter(words)\n",
    "\n",
    "        # Sort the words in descending order of frequency\n",
    "        sorted_word_count = dict(sorted(word_count.items(), key=lambda item: item[1], reverse=True))\n",
    "        print (sorted_word_count)\n",
    "        return sorted_word_count\n",
    "\n",
    "    sorted_word_count = word_count(event['paragraph'])\n",
    "\n",
    "    return {\n",
    "        \"statusCode\": 200,\n",
```

```
    "        \"body\": str(sorted_word_count)\n",
    "    }\n",
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "bf74ca6d-871b-4308-9ce2-6082897db076",
   "metadata": {},
   "source": [
    "\n",
    "In the function code editor, copy and paste the word_count function definition we defined earlier. \n",
    "This version of the function sorts the frequency of words in descending order using the sorted() method with a key function that sorts by the second element of each tuple in the dictionary. The resulting dictionary is then converted to a JSON object using the json.dumps() method and returned in the response body."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "76307846",
   "metadata": {},
   "source": [
    "### 5- Define the input event\n",
    "You can invoke your Lambda function directly by creating a JSON event object and passing it as an argument to the function.\n",
    "\n",
    "To create a test event:\n",
    "1. Click on the <b> Test </b> Tab in the top left corner (navigate to the test page)\n",
    "2. Copy and paste the JSON code below into \"Event JSON\" section\n",
    "3. Press save"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "66aa27e4-3501-41a6-a362-e7f86ccfc8aa",
   "metadata": {},
   "outputs": [],
   "source": [
    "{\n",
    "    \"paragraph\": \"ChatGPT is a cutting-edge language model that can perform a wide range of natural language processing tasks with remarkable
```

accuracy and fluency. From answering general knowledge questions and assisting with research, to generating human-like text and even engaging in creative writing, ChatGPT's capabilities are truly impressive. With its vast knowledge base and ability to learn and adapt over time, ChatGPT can serve as a valuable resource for anyone looking to harness the power of natural language processing in their work or personal projects. Whether you're a researcher, a writer, a business professional, or simply someone looking to engage in interesting conversations, ChatGPT has the skills and knowledge to help you achieve your goals.\"\n",
    "}"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "15362836-1fd8-475f-a3dd-708da4ad58b4",
   "metadata": {},
   "source": [
    "In this example, we define a JSON object with a \"paragraph\" key that contains the input paragraph to be analyzed. We then call the 'word_count' function and pass this event object as the first argument and None as the second argument to simulate an empty context object."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "90a96e69-8d2c-4370-b995-bf98f873f2fb",
   "metadata": {},
   "source": [
    "### 6- Deploy and test the function\n",
    " You can now test the function by invoking it with test data as an event.\n",
    "1. Click on the Code Tab in the top left corner (navigate to the code page)\n",
    "2. Click on the \"Deploy\" button to ensure lambda is deployed\n",
    "3. Click on the test button to see the results of the function"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "328ae9c9",
   "metadata": {},
   "source": [
    "### 7- Serverless Word Count with AWS Lambda and S3 Triggers\n",
    "\n",
    " In this step, you need to create and deploy a serverless function for a Word Count application, triggered by S3 bucket file upload event. Once

```
triggered, it should parse the uploaded file to get the word count. \n",
    "\n",
    "\n",
    "1. **Navigate to S3 bucket console**\n",
    "    - Search and navigate to the S3 bucket console. <br/><br/>\n",
    "\n",
    "2. **Create a new bucket**\n",
    "    - Click on \"Create Bucket\".\n",
    "    - Provide a bucket name.\n",
    "    - Select region as \"ca-central-1\".\n",
    "    - Turn off \"Block All Public Access\" and acknowledge the
warning.\n",
    "    - Click on \"Create Bucket\".<br/><br/>\n",
    "\n",
    "3. **Edit Bucket Permissions**\n",
    "    - Now you can see your bucket in the list of buckets.\n",
    "    - Click on the bucket name.\n",
    "    - Then click the \"Permissions\" tab.\n",
    "    - Edit the bucket policy and include the following (replace
`your_bucket_name` with your bucket name):<br/><br/>\n",
    "\n",
    "```json\n",
    "{\n",
    "    \"Version\": \"2012-10-17\",\n",
    "    \"Statement\": [\n",
    "        {\n",
    "            \"Sid\": \"AllowPublicRead\",\n",
    "            \"Effect\": \"Allow\",\n",
    "            \"Principal\": {\n",
    "                \"AWS\": \"*\"\n",
    "            },\n",
    "            \"Action\": \"s3:GetObject\",\n",
    "            \"Resource\": \"arn:aws:s3:::your_bucket_name/*\"\n",
    "        }\n",
    "    ]\n",
    "}\n",
    "```\n",
    "\n",
    "4. **Add Trigger to Lambda Function**\n",
    "    - Go to the Lambda dashboard (of the function you previously
made), and click on \"Add Trigger\".\n",
    "    - Select S3 from the list of triggers.\n",
    "    - Choose the bucket name from the dropdown list.\n",
    "    - Select the event type as \"Object Created (All)\" and click on
\"Add\".<br/><br/>\n",
    "5. **Specify File Type for Trigger**\n",
```

```
    "    - Add `.txt` file in the optional suffix box to trigger the Lambda function only when a text file is uploaded to the S3 bucket.\n",
    "    - Now click \"Add\".<br/><br/>\n",
    "\n",
    "6. **Code to Upload Files to S3 Bucket**\n",
    "   - Create a `.env` file in the same directory as this notebook and add the following variables to it:<br/><br/>\n",
    "\n",
    "```bash\n",
    "        AWS_ACCESS_KEY_ID=your_access_key_id\n",
    "        AWS_SECRET_ACCESS_KEY=your_secret_access_key\n",
    "        AWS_SESSION_TOKEN=your_session_token\n",
    "        BUCKET_NAME=your_bucket_name\n",
    "```\n",
    "   - Create a txt file called `upload.txt` in the same directory as this notebook and add some text to it (copy 'paragraph' from above)\n",
    "\n",
    "7. **Upload File to S3 Bucket**\n",
    "   - Now running the following code will upload a file to the S3 bucket (you may need to install boto3 package and restart the notebook kernel).\n",
    "\n",
    "Please note that you should replace `your_bucket_name`, `your_access_key_id`, `your_secret_access_key`, and `your_session_token` with your actual AWS credentials and bucket name. "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "daaf2b8e",
   "metadata": {},
   "outputs": [],
   "source": [
    "import logging\n",
    "import boto3\n",
    "from botocore.exceptions import ClientError\n",
    "import os\n",
    "from dotenv import load_dotenv\n",
    "\n",
    "load_dotenv(\".env\")\n",
    "\n",
    "for key, value in os.environ.items():\n",
    "    print(f\"{key}: {value}\")\n",
    "\n",
    "def upload_file(file_name, bucket, object_name=None):\n",
```

```
    "    # Verify env file is setup correctly\n",
    "    print(\"AWS Access Key:\", os.getenv(\"AWS_ACCESS_KEY_ID\"))\n",
    "    \n",
    "    \"\"\"Upload a file to an S3 bucket\n",
    "\n",
    "    :param file_name: File to upload\n",
    "    :param bucket: Bucket to upload to\n",
    "    :param object_name: S3 object name. If not specified then
file_name is used\n",
    "    :return: True if file was uploaded, else False\n",
    "    \"\"\"\n",
    "    # If S3 object_name was not specified, use file_name\n",
    "    if object_name is None:\n",
    "        object_name = os.path.basename(file_name)\n",
    "    # Upload the file\n",
    "    s3_client = boto3.client('s3',
aws_access_key_id=os.getenv(\"AWS_ACCESS_KEY_ID\"),
aws_secret_access_key=os.getenv(\"AWS_SECRET_ACCESS_KEY\"),
aws_session_token=os.getenv(\"AWS_SESSION_TOKEN\"))\n",
    "    try:\n",
    "        response = s3_client.upload_file(file_name, bucket,
object_name)\n",
    "        print(response)\n",
    "    except ClientError as e:\n",
    "        logging.error(e)\n",
    "        return False\n",
    "    return True\n",
    "# upload_file('upload.txt', \"testbucketcpsc436\" , 'upload.txt')\n",
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "081a3f39",
   "metadata": {},
   "source": [
    "Update the serverless function to parse the uploaded .txt file when
called by the S3 bucket event :"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "c9bfd0dd",
   "metadata": {},
   "outputs": [],
```

```
   "source": [
    "import json\n",
    "from collections import Counter\n",
    "import boto3\n",
    "\n",
    "def lambda_handler(event, context):\n",
    "    def word_count(paragraph):\n",
    "        # Remove punctuation from input paragraph and convert to
lowercase\n",
    "        paragraph = paragraph.lower().replace('.', '').replace(',',
'')\n",
    "\n",
    "        # Split the paragraph into a list of words\n",
    "        words = paragraph.split()\n",
    "\n",
    "        # Count the number of occurrences of each word\n",
    "        word_count = Counter(words)\n",
    "\n",
    "        # Sort the words in descending order of frequency\n",
    "        sorted_word_count = dict(sorted(word_count.items(), key=lambda
item: item[1], reverse=True))\n",
    "        print (sorted_word_count)\n",
    "        return sorted_word_count\n",
    "\n",
    "    s3 = boto3.client('s3')\n",
    "    \n",
    "    # Get bucket name and file key from the S3 event\n",
    "    bucket_name = event['Records'][0]['s3']['bucket']['name']\n",
    "    file_key = event['Records'][0]['s3']['object']['key']\n",
    "    \n",
    "    # Get the file object from S3\n",
    "    file_obj = s3.get_object(Bucket=bucket_name, Key=file_key)\n",
    "    \n",
    "    # Read the content of the file\n",
    "    file_content = file_obj['Body'].read().decode('utf-8')\n",
    "    \n",
    "    print(f'Content of the file {file_key} from bucket
{bucket_name}:')\n",
    "\n",
    "    sorted_word_count = word_count(file_content)\n",
    "\n",
    "    return {\n",
    "        \"statusCode\": 200,\n",
    "        \"body\": str(sorted_word_count)\n",
    "    }\n",
    "\n",
```

```
    "    "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "e1a86a8c",
   "metadata": {},
   "source": [
    "\n",
    "After you run the file upload function, you can see the uploaded file in the S3 bucket. This will also trigger the Lambda function and you can see the output in the Lambda console by navigating to the function and clicking on \"Monitor\"->\"logging\"->\"View logs in CloudWatch\"."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "b1ac0fe1",
   "metadata": {},
   "source": [
    "## 8 Create an API Gateway\n",
    "In this step, you will create an API Gateway so you can send various paragraphs to be processed by the lambda functions. \n",
    "\n",
    "An API Gateway serves as a centralized entry point for managing, securing, and optimizing APIs. Its main purposes include routing requests to backend services, enforcing security measures, providing monitoring and analytics, transforming data formats, implementing caching, load balancing, handling cross-cutting concerns, and managing API versions. It simplifies API management, enhances security, and improves overall system performance.\n",
    "\n",
    "0. **Create a New Serverless Function**\n",
    "    - Navigate to Lambda console\n",
    "    - Create a new serverless function called \"WordCountRestAPI\"\n",
    "    - Copy the following code into the serverless function\n",
    "    <br/><br/>"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "f34829ba",
   "metadata": {},
   "outputs": [],
   "source": [
```

```
    "import json\n",
    "from collections import Counter\n",
    "\n",
    "def lambda_handler(event, context):\n",
    "    \n",
    "    def word_count(paragraph):\n",
    "        # Remove punctuation from input paragraph and convert to 
lowercase\n",
    "        paragraph = paragraph.lower().replace('.', '').replace(',', 
'')\n",
    "\n",
    "        # Split the paragraph into a list of words\n",
    "        words = paragraph.split()\n",
    "\n",
    "        # Count the number of occurrences of each word\n",
    "        word_count = Counter(words)\n",
    "\n",
    "        # Sort the words in descending order of frequency\n",
    "        sorted_word_count = dict(sorted(word_count.items(), key=lambda 
item: item[1], reverse=True))\n",
    "        print (sorted_word_count)\n",
    "        return sorted_word_count\n",
    "    \n",
    "    paragraph = event['body'][\"paragraph\"]\n",
    "        \n",
    "    sorted_word_count = word_count(paragraph)\n",
    "        \n",
    "\n",
    "    return {\n",
    "        \"statusCode\": 200,\n",
    "        \"body\": str(sorted_word_count)\n",
    "    }\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "cc7ef34e",
   "metadata": {},
   "source": [
    "\n",
    "1. **Navigate to API Gateway console**\n",
    "    - Navigate to API Gateway console.\n",
    "    <br/><br/>\n",
    "    \n",
    "2. **Create a new API Gateway**\n",
    "    - Click on \"Create API\"\n",
```

```
    "    - Scroll down to REST API and Press \"Build\"\n",
    "    - Choose \"New API\" and name your API\n",
    "    - Click Create API<br/><br/>\n",
    "\n",
    "3. **Create A Resource**\n",
    "    - Click \"Create resource\"\n",
    "    - Make your Resource Name \"test\" and click \"Create
Resource\"<br/><br/>\n",
    "\n",
    "4. **Create a Method**\n",
    "    - Click \"Create method\"\n",
    "    - Choose Method type \"Post\"\n",
    "    - Choose Lambda function\n",
    "    - Choose \"ca-central-1\"\n",
    "    - Start typing \"WordCountRestAPI\" (lambda you created in the
previous step) in the input box and choose that one\n",
    "    - Click \"Create method\"<br/><br/>\n",
    "\n",
    "\n",
    "5. **Deploy API**\n",
    "    - Press \"Deploy API\" in the upper right hand corner\n",
    "    - In the Stage dropdown select \"new stage\"\n",
    "    - Name the new stage\n",
    "    - Press Deploy<br/><br/>\n",
    "\n",
    "6. **Navigate to Stages**\n",
    "    - On the left hand side of the page navigate to \"Stages\"\n",
    "    - Once you have deployed the api you should see an Invoke URL\n",
    "    - <b>Copy the Invoke URL</b>\n",
    "    - You will be using this invoke URL to query the API<br/><br/>\n",
    "\n",
    "7. **Run the following python script to hit the API endpoint**\n",
    "    - Replace the current the invoke_url_and_resource_path variable
with:<br/><br/>\n",
    "        - concatenated(invoke url + resource name) <br/><br/>\n",
    "            - eg. invoke_url    =
...ca-central-1.amazonaws.com/stage_1\n",
    "            - eg. resource name = test\n",
    "            - eg. FINAL_URL =
...ca-central-1.amazonaws.com/stage_1/<b>test</b><br/><br/>\n",
    "\n",
    "8. <b>If you run into an \"missing token error\":</b>\n",
    "    - ensure that your resource name is located at the end of your
URL: \n",
    "        - eg. invoke_url    =
...ca-central-1.amazonaws.com/stage_1\n",
```

```
    "        - eg. resource name = test\n",
    "    \n",
    "        - eg. FINAL_URL =
...ca-central-1.amazonaws.com/stage_1/<b>test</b>\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "9c9f6a09",
   "metadata": {},
   "outputs": [],
   "source": [
    "import requests\n",
    "import json\n",
    "\n",
    "# REPLACE with your INVOKE URL and RESOURCE PATH\n",
    "invoke_url_and_resource_path =
\"https://fwefasd.execute-api.us-east-2.amazonaws.com/deploy/test\"\n",
    "\n",
    "# JSON body to be sent in the POST request\n",
    "data = {\n",
    "  \"body\": {\n",
    "    \"paragraph\": \"We were both young when I first saw you I close
my eyes and the flashback starts I'm standin' there On a balcony in summer
air See the lights, see the party, the ball gowns See you make your way
through the crowd And say, Hello Little did I know That you were Romeo, you
were throwin' pebbles And my daddy said, Stay away from Juliet And I was
cryin' on the staircase Beggin' you, Please don't go,  and I said Romeo,
take me somewhere we can be alone I'll be waiting, all there's left to do
is run You'll be the prince and I'll be the princess It's a love story,
baby, just say, Yes So I sneak out to the garden to see you We keep quiet,
'cause we're dead if they knew So close your eyes Escape this town for a
little while, oh oh 'Cause you were Romeo, I was a scarlet letter And my
daddy said, Stay away from Juliet But you were everything to me I was
beggin' you, Please don't go,  and I said Romeo, take me somewhere we can
be alone I'll be waiting, all there's left to do is run You'll be the
prince and I'll be the princess It's a love story, baby, just say, Yes
Romeo, save me, they're tryna tell me how to feel This love is difficult,
but it's real Don't be afraid, we'll make it out of this mess It's a love
story, baby, just say, Yes Oh, oh I got tired of waiting Wonderin' if you
were ever comin' around My faith in you was fading When I met you on the
outskirts of town, and I said Romeo, save me, I've been feeling so alone I
keep waiting for you, but you never come Is this in my head? I don't know
what to think He knelt to the ground and pulled out a ring And said, Marry
me, Juliet You'll never have to be alone I love you and that's all I really
```

```
know I talked to your dad, go pick out a white dress It's a love story, baby, just say, Yes Oh, oh, oh Oh, oh, oh, oh 'Cause we were both young when I first saw you\",\n",
    "  }\n",
    "}\n",
    "\n",
    "# Convert the Python dictionary to a JSON string\n",
    "json_data = json.dumps(data)\n",
    "\n",
    "# Specify the headers if needed (e.g., content type as JSON)\n",
    "headers = {\n",
    "    \"Content-Type\": \"application/json\"\n",
    "}\n",
    "\n",
    "# Send the POST request\n",
    "response = requests.post(invoke_url_and_resource_path, data=json_data, headers=headers)\n",
    "\n",
    "# Print the response\n",
    "print(\"Response Code:\", response.status_code)\n",
    "print(\"Response Body:\", response.text)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "23151af6",
   "metadata": {},
   "source": [
    "# 9 - Create and Deploy your own code\n",
    "You learnt how to create a Function as a Service (FaaS) through steps 1-8. Now, you need to choose/develop a simple code and attempt to deploy it as a FaaS.\n",
    "\n",
    "To deploy your code as a FaaS, you need to modify it to conform to the platform's requirements.\n",
    "\n",
    "To trigger the execution of your function, you must specify an event. The event could be anything that the FaaS platform supports, such as a new file upload, a new database entry, or an HTTP request. Once the event is detected, the platform will invoke your function, passing it the necessary input parameters. The function will then execute and return the output to the platform, which can be further processed or returned to the user.\n",
    "\n",
    " - What modification is required in your code?\n",
    " - What could be defined as an event to trigger your function execution?\n",
```

```
    " - Did you face any dependency problem?"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "00a7619d",
   "metadata": {},
   "source": [
    "### Deliverables:\n",
    "\n",
    "In a word document please show:\n",
    "- Step 6 - 7:\n",
    "    - Pictures of \"Execution Result\" of word_count serverless function. \n",
    "    - The brief explanation of how you configured the event to trigger your lambda function. \n",
    "- Step 8:\n",
    "    - A Picture of your API gateway Resources Page and Stages Page\n",
    "    - A Picture of your serverless function\n",
    "    - A Picture of the \"Execution result\" of running your serverless function\n",
    "    - Your url (invoke_url_and_resource_path) that you used to process words with your api gateway for TA testing\n",
    "- Step 9:\n",
    "    - The serverless function code you created for step 9.\n",
    "    - The brief explanation of how you configured the event to trigger your lambda function\n",
    "    - The event code\n",
    "    - The \"Execution result\" of running your serverless function"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "ca605b10",
   "metadata": {},
   "source": [
    "### Good Luck!"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
```

```
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.13"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```