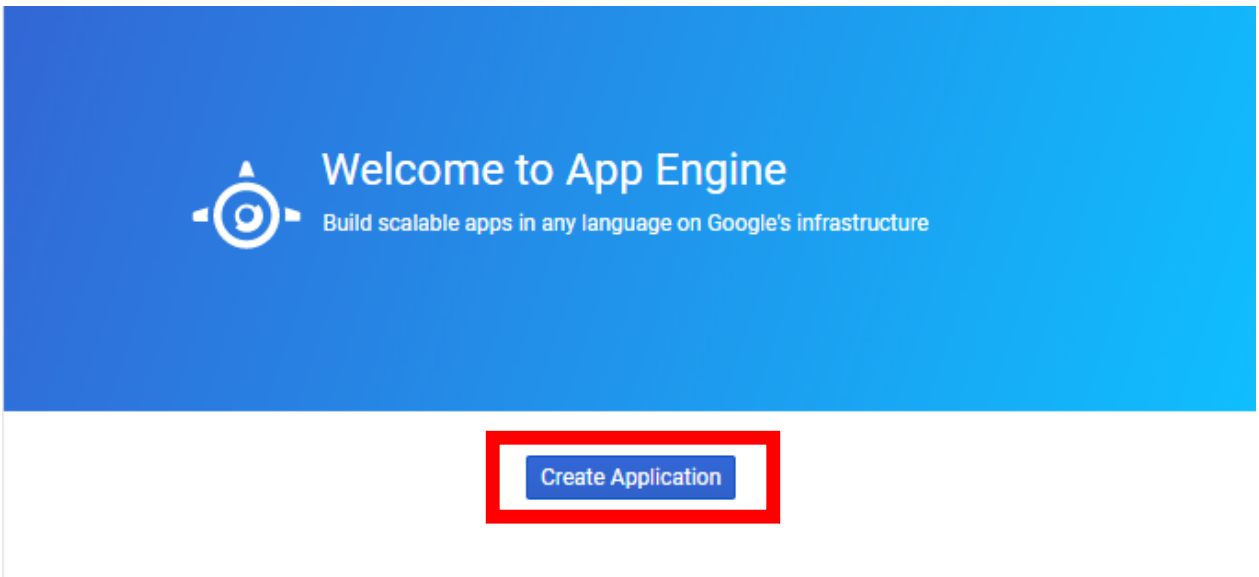
GCP Lab 2

LAB Overview

This lab introduces you how you can integrate several Google Cloud Platform services. It will use Compute Engine, App Engine, App Engine cron, Cloud Pub/Sub, Cloud Function and Cloud Storage.
You will build solution that produces event on App Engine, sends them to Cloud Pub/Sub, then they are retrieved by Cloud Function and saved into Cloud Storage.

Task 1: Enabling App Engine API in GCP portal.

1. Go to GCP Portal:
   https://console.cloud.google.com
   And sign in using you Your Gmail credential.
2. From navigation bar go to App Engine and select Dashboard. You should be prompted with following:



3. Click **Create Application** and in Region page set:

- **Region**: europe-west

Click **Create**.

4. In the next step select:

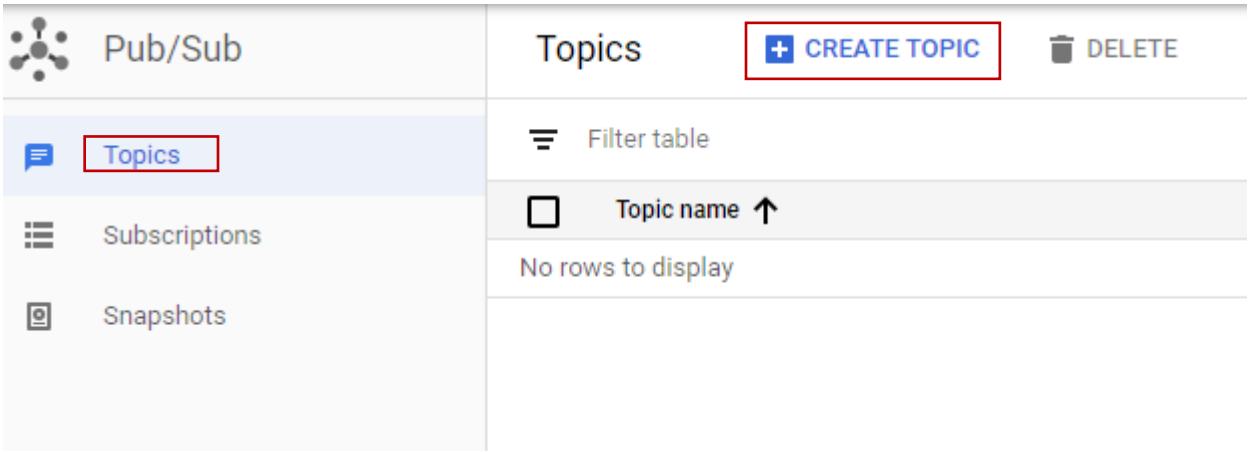
   - **Language**: .NET
   - **Environment**: Flexible

   Click **Next**.
   In the last page click **I'll DO THIS LATER**.

5. Your App Engine service is now up and ready for applications deployment.

Task 2: Creating Cloud Pub/Sub topic.

1. From navigation bar go to **Big Data** section and select **Pub/Sub**. API for Cloud Pub/Sub will be enabled automatically so please wait.
2. When API is enabled then on **Topic** page click **Create Topic** button:



3. On the pop-up window fill fields:

    - **Name**: studentXXtopic
    - **Encryption**: Google-managed key

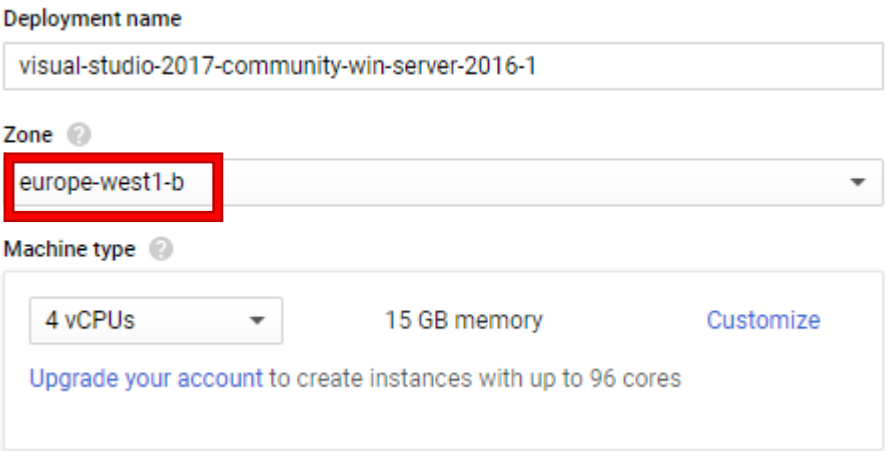
    And click on **Create Topic**.

Remember to write down the name of the created Topic.

Task 3: Create virtual machine from which the web application will be deployed to App Engine

1. From navigation bar go to **Compute Engine** and select **VM instances**.
2. If Compute Engine API was not started you need to wait for it.
3. From the left panel select **Marketplace**.
4. Inside Marketplace find in search **Visual Studio 2017 Community on Hardened Windows Server 2016** and select it.
5. In the next page click buton **Launch on Compute Engine**.
6. In the Virtual Machine creation window change **Zone** to one from europe-west region:

    - **Zone:** europe-west1-b
    - Leave the rest configuration as it is



Click **Deploy** button.

7. After virtual machine is deployed navigate to **Compute Engine** > **VM Instances**.
8. Select newly created instance and click **Edit**.
9. Slide down and in the **Firewalls** configuration part mark **Allow HTTP traffic**.
10. Click **Save** and after modification finishes go back to **VM Instances** page.
11. In the list of instances click on the arrow in the **Connect** column (next to **RDP**).
12. Select **Set Windows password**:

    - **Username:** your-gmail-account

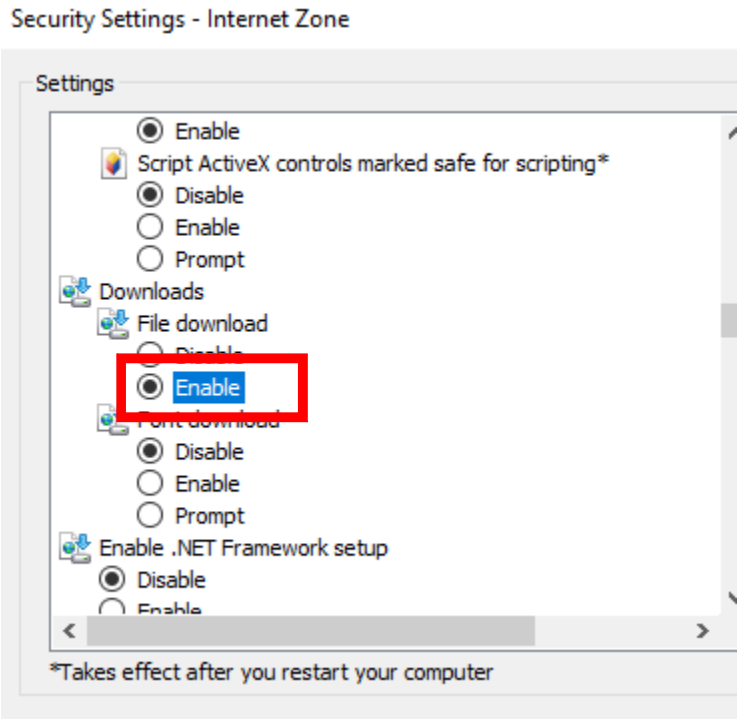
    Click **Set** button.

13. Copy generated password and login to VM through RDP using **your-gmail-acount** and VM IP from **External IP** column.
    a. **<u>If there is problem with connecting to VM follow step</u>**:
        i. Navigate to **VPC network** > **Firewall rules**
        ii. Click **Create Firewall rule**
            1. **Name**: rdp
            2. **Priority**: 1000
            3. **Targets**: All instances in the network
            4. **Source filter**: IP ranges
            5. **Source IP ranges**: 0.0.0.0/0
            6. **Specified protocols and ports**: tcp - 3389

    Click **Create**.

14. On the virtual machine open **Internet Explorer**.

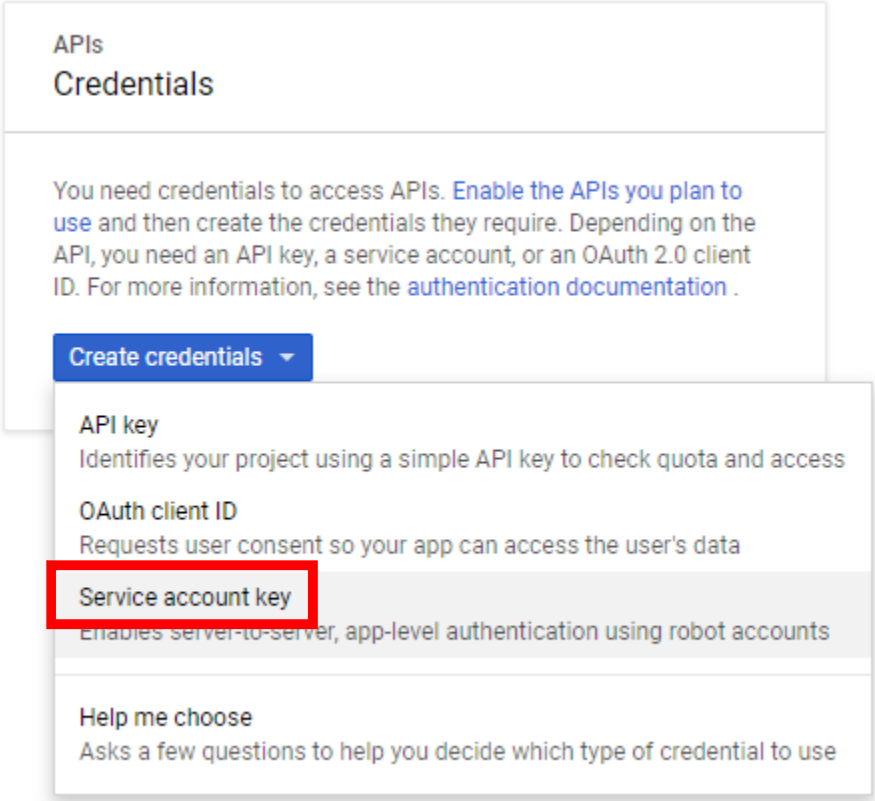
15. Open **Menu** > **Tools** > **Internet Options** > **Security** tab > **Custom Level**
16. Inside **Custom level** configuration window find **Download** configuration and set it to **Enable**.



17. After that download .Net Core SDK from link: https://dotnet.microsoft.com/download/thank-you/dotnet-sdk-2.1.701-windows-x64-installer
18. Then run installer and install SDK.
19. Open command line as administrator and run **dotnet --version** to verfiy installation - it should return **2.1.701**

Task 4: Setting serice account credentials for Virtual Machine

1. From the navigation bar in GCP console go to **APIs & Services** > **Credentials**.
2. Click **Create credentials** button and select **Service account key**



3. In the Create service account key page select:

    - **Service account:** App Engine default service account
    - **Key type**: JSON

    Click **Create** and save created key.

4. Go back to VM and copy created JSON credential file there.
5. Open command line as administrator and run command with path to copied file:
    a. gcloud auth activate-service-account --key-file "%PATH%\<FILE-NAME>.json"

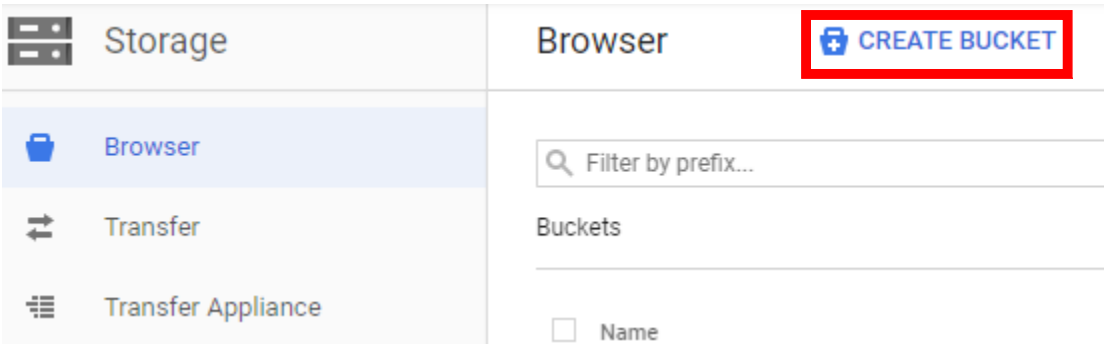
Task 5: Deploying web application to App Engine

1. On the VM in command line run:
    a. gcloud components update (next windows should appear, type **Y**)
2. With command line go to C:\ and create **dev** catalog and go in it.
    a. git clone https://github.com/lasekg/aspnetcorepubsub.git
3. Open aspnetcorepubsub/Pubsub catalog and edit **appsettings.json** file, change values corresponding to your project ID and previously created topic name (to get your Project ID navigate to **IAM & admin** > **Settings**):
    a. **<YOUR-PROJECT-ID>** - Project ID
    b. **<YOUR-TOPIC-NAME>** - Topic name
4. In the VM in command line navigate to source root folder:
    a. cd aspnetcorepubsub/Pubsub
5. Inside the folder do following:
    a. Run: dotnet restore
    b. Run: dotnet publish
    c. Run: gcloud app deploy bin/Debug/netcoreapp2.1/publish/app.yaml --project=**<YOUR-PROJECT-ID>** (if the command fails run it again)
        i. Confirm enablind App Engine API
        ii. Confirm deployment
6. Verify application deployment under:
    a. https://<YOUR-PROJECT-ID.appspot.com/
    b. https://<YOUR-PROJECT-ID.appspot.com/home/messages (for generating few messages to Cloud Pub/Sub manually)
7. Wait till deployment ends and application responds, then deploy the cron (from the same folder as application):
    a. gcloud app deploy bin/Debug/netcoreapp2.1/publish/cron.yaml --project=**<YOUR-PROJECT-ID>**
8. Verify if cron was deployed properly by navigating in portal to **App Engine** > **Cron jobs**

Task 6: Creating Bucket in Cloud Storage

1. Navigate to section **Storage** > **Storage** in GCP portal.

2. Click **Create bucket** button and in the next steps fill:

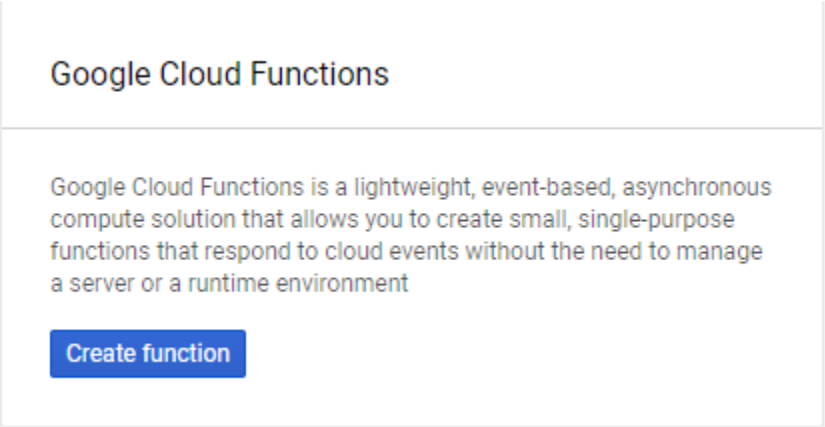


a. **Name your bucket**: studentXXbucket
b. **Choose a default storage class**: Regional
c. **Location**: europe-west1
d. **Access control model**: Set object-level and bucket-level permissions
e. **Encryption:** Google-managed key

Click **Create**.

Task 7: Creating Cloud Function for processing messages

1. Navigate to section **Compute** > **Cloud Function** in GCP portal.
2. After Cloud Function API starts click **Create function**:



3. Fill function configuration:
   a. **Name**: function-1
   b. **Memory allocated**: 256 MB

c. **Trigger**: Cloud Pub/Sub
d. **Topic**: <YOUR-TOPIC-NAME>
e. **Source code**: Inline editor
f. **Runtime**: Node.js 8
g. **Code preview**: paste code with correct bucket name in index.js:

```
const {Storage} = require('@google-cloud/storage');
var fs = require("fs");

exports.processPubSub = (event, context) => {

  const storage = new Storage();
  var bucket = storage.bucket('<YOUR-BUCKET-NAME>');

  const pubsubMessage = event.data;
  var data = Buffer.from(pubsubMessage, 'base64').toString();
  console.log('Received message:', data);

  var now = new Date();
  var datePrefix = now.getFullYear() + "-"+ now.getMonth() + "-"
+ now.getDate() + "-" + now.getHours() + "-" + now.getMinutes() +
"-" + now.getSeconds();
  var logFileName = datePrefix + ".log";
  var logFilePath = "/tmp/" + logFileName;

  fs.writeFile(logFilePath, data, (err) => {
    if (err) console.log(err);
    console.log("Successfully written message data to log
file.");
  });

  bucket.upload(logFilePath, function(err, file) {
    if (err) console.log(err);
    console.log("Successfully uploaded log file in to storage");
  });
};
```

h. **Code preview:** paste code in packacge.json**:**

```
{
  "name": "sample-pubsub",
  "version": "0.0.1",
  "dependencies": {
    "@google-cloud/pubsub": "^0.18.0",
```

```
    "@google-cloud/storage": "3.0.2"
  }
}
```

i. **Function to execute:** processPubSub
j. Click on Environment variables, networking, timeouts and more
k. **Region**: europe-west1

Click **Create**.

Task 8: Verify if data appear in Cloud Storage Bucket