Cloud Computing Labs: Exploring a multi-cloud application

The primary purpose of this assignment is to get familiar with the typical PaaS example: Google AppEngine, understand the multi-cloud strategy, and experiment with a simple multi-cloud application.

The motivation of multi-cloud apps is to avoid the well-known problem of lockin and achieve the highest level of data reliability and availability for mission-critical tasks. In this small project, we will experiment with a toy example of storing application data (key-value pairs) in the two clouds: GAE's data store and Amazon DynamoDB. The task consists of three components. (1) GAE for the front-end web UI, (2) a microservice running in Amazon EC2 as an interface between GAE and DynamoDB, (3) an extensible API that stores data to multiple clouds (possibly extended to more clouds) and hides the details from the web UI.

Part 1: preparation

1.1 Review the lectures about using GAE. (1) Create your GAE account. Note that GAE has the free-tier quote for each account, which is sufficient for this project. (2) Download and install the google cloud SDK to your linux machine. Follow this instruction and remember to install the python related components google-cloud-sdk-app-engine-python and google-cloud-sdk-app-engine-python-extras. You will need to create an empty application at GAE as the "gcloud init" command will ask you which application id to use. Check the console page https://console.cloud.google.com/ to create the application. (3) Download this revised version of the GuestBook example. Go through the whole process of locally creating and debugging a web application and uploading to the GAE application you created (check the lecture slides). You should test-run the application with the local environment. You can use the normal browser or "lynx" (install it by "apt install lynx") as the commandline web browser if you work with terminals only.

To start the web server, go to the application directory that has app.yaml and type

```
dev_appserver.py app.yaml
```

The default port will be 8080. You can open the web page at a browser with the url "localhost:8080" or at a terminal with "lynx localhost:8080". If your application gets an internal error, you don't need to restart the web server. After debugging your code, refresh the browser to see whether your revised code works. Finally, deploy your application with

```
gcloud app deploy app.yaml
```

Now, answer the following questions:

Question 1.1 Revise the application to show "Greeting id [gid] saved at [time] says: [content]" instead, where [gid] is replaced with the greeting id, [time] with the greeting time, and [content] with the greeting content. Note that all the fields are already defined in the greeting data model. Debug locally and deploy the app to GAE. Briefly describe how you revise the code and post the URL to your app. You don't need to post the code.

1.2 Get familiar with the DynamoDB by reading the tutorial. You should work with the AWS setup you have done in Project 2. If you were not able to finish Project 2, please come to office hours. Boto3 provides nice support for accessing DynamoDB with python program. You can check the skeleton code and experiment with the API.

Now, answer the following questions:

Question 1.2 Based on the skeleton code, please use the APIs and methods to (1) create a table "greetings" that has the three fields "gid", "date", and "content". The method "create_table" is (2) Make sure you can read/insert/delete greetings. (3) Post your code snippets for (1) and (2) here.

Question 1.3 How much time did you spend on these tasks? (hours)

Question 1.4 How useful is this task to your understanding of the GAE development and DynamoDB? (very useful, somewhat useful, not useful)

Part 2: A guestbook application working with multiple cloud storages.

You will implement a simple guestbook web application that uses the GAE for the web interface, and stores data on two cloud data storages: GAE datastore and Amazon DynamoDB to achieve better availability and reliability. Your application should be extensible, allowing more cloud data storages to be easily added in the future.

The target web app is a simple modification of the GAE guestbook example. We have known how the app works. Users can append greetings and the web portal will store the greetings to the datastore and show all the recorded ones. In general, the basic webapp2-based applications can be used outside of GAE, except for the APIs specific to GAE, such as the GAE datastore APIs (i.e., BigTable as the back-end implementation). To avoid data lock-in and have redundant data storages for better data safety and possibly better concurrent data sharing, we will design the following framework that unifies all data storage activities and interfaces with different service providers. First, let's integrate DynamoDB. As GAE cannot use the boto3 library directly (experiment with it by yourself), we want to develop a RESTful microservice running in a EC2 instance that serves as the interface to the DynamoDB. The GAE application will use the url fetch API to access the microservice.

Your first task will be implementing the microservice, test it, and deploy it to an EC2 instance. You will be using the python flask framework for the implementation, which is clear and simple to learn. The movie database example and also this blog post are very good sources for understanding how a flask web service is implemented. This skeleton code for the greetings data service has been provided for you to implement the microservice. Please check the details in the skeleton code.

Question 2.1 Post your microservice code here, and briefly describe how you test your services.

Question 2.2 Now use the GAE url fetch API to access the microservice to store/retrieve greetings. Briefly describe which part of the original application needs to be modified. No need to post code here.

Question 2.3 Modify the existing design of your GAE application to provide the flexibility for incorporating multiple cloud data stores. Specifically, when a new greeting is entered, it will be saved to both GAE datastore and DynamoDB. You will

implement a new "data model" interface, as provided by this skeleton code, which can be inherited by both GAE datastore based methods (the GAEGreeting class) and the DynamoDB microservice (the DynamoGreeting class), and possibly any new data store method. Your GAE main program will instantiate and use the UnifiedGreeting class instead. Post the code in datamodel.py here, and briefly describe how you modify the GAE main program.

Question 2.4 How much time did you spend on this part? (hours)

Final Survey Questions

Question 3.1 Your level of interest in this lab exercise (high, average, low);

Question 3.2 How challenging is this lab exercise? (high, average, low);

Question 3.3 How valuable is this lab as a part of the course (high, average, low);

Question 3.4 Are the supporting materials and lectures helpful for you to finish the project? (very helpful, somewhat helpful, not helpful);

Quertion 3.5 Do you feel confident on applying the skills learned in the lab to handle other similar problems with the multi-cloud strategy? (high, average, low)

Deliverables

Turn in (1) the code and answers for Questions 1.1-1.2 and 2.1-2.3 in one PDF file. No need to rephrase the questions. For coding problems, you should copy&paste your code in text if requested. Do not post screenshots of your code or answers, unless you are asked to do so. (2) All survey questions go to the second PDF. (3) all source code ziped in one file. Submit these three files. Without following the instructions, you will get 0 immediately.

Make sure that you have terminated all EC2 instances after finishing your work!

This page, first created: Nov 2019