Project 1: IaaS—Amazon Web Services

CSE 546: Cloud Computing

1. Summary

In the first project, we will build an elastic application that can automatically scale up and down on demand and cost-effectively by using cloud resources. Specifically, we will build this application using the cloud resources from Amazon which is the most widely used IaaS provider and offers a variety of different compute, storage, and message cloud services. Although our application will be quite simple, the technologies and techniques that we use will be useful to create full-blown cloud applications.

2. Description

This application is called **CloudPi**, which uses cloud resources to compute the digits of Pi. It follows the typical three-tier architecture: 1) the web tier services the requests for computing Pi; 2) the application tier performs the computation; and 3) the data tier stores the computing result. The application tier needs to automatically scales up and down based on the demand of the requests. Specifically, your implementation of CloudPi should follow the requirements below.

Requirements:

1) The **web tier** should be hosted on a single EC2 instance. It should be able to service HTTP requests on a static public IP address. Assume that the public IP is *A.B.C.D*, an Internet user makes a request to your web tier using the URL http://A.B.C.D/cloudpi.php?input=x, where *x* is an integer value which is the input of the compute Pi program. The web tier will pass on this input to the application tier, which computes Pi and passes back the output. The web tier should then return the output to the user in a plain HTML page.

   Note: It is important that your web tier supports the URL above, because the workload generator that I will use to test your implementation will issue requests using only this URL.

2) The **application tier** should be hosted on EC2 instances. Because the computation of Pi is a highly CPU intensive task, we want to run it on a separate tier, instead of as part of the web tier, so that it is able to scale dynamically. An application tier instance receives the input from the web tier—the *x* value in the HTTP request that the web tier receives, and executes the compute Pi program using this input.

   The compute Pi program named **pifft** will be provided to you. It is a C program that uses Fast Fourier Transform (FFT) to calculate the digits of pi. The pifft program takes a single command-line argument which is the path of the input file. The input file is a plain text file containing only an integer which is the number of FFT iterations that pifft should run. The more iterations that it runs, the higher number of digits that it can produce for pi. At the end of the run, the program prints out all the computed digits to the standard output.

   The application tier needs to execute the pifft program using the input that it receives from the web tier as the input to the pifft program. For example, if the web tier receives a request http://A.B.C.D/cloudpi.php?input=65536, the application tier should run pifft with 65536 as the input—the number of FFT iterations.

In order to process multiple compute Pi requests concurrently, the application tier should automatically scale up by launching more instances. Each application tier instance is identical to the others, but each processes a different request given by the web tier. When an application tier instance becomes idle because of the drop in request demand, it should be terminated immediately.

Note: Because we have limited resources from the free tier, you should 1) use no more than ten instances for the application tier; and 2) terminating an application tier instance, instead of stopping it, when it is not needed any more.

3) The **data tier** uses S3 to store the results of the Pi computations persistently. For each compute Pi request that the web tier receives, the data tier needs to store the input—the number of FFT iterations, and the output—the digits of Pi, as a pair of strings. The data tier should store the input-output pairs for all the requests that the web tier has received and the application tier has computed accordingly.

You need to use an AWS SDK to implement CloudPi, particularly the application tier and the data tier. The AWS Java SDK is the recommended one for the project. If you prefer using a different language supported by AWS, you need to get the instructor's approval.

In addition, you need to implement several simple SHELL scripts using the AWS Linux CLI for starting and stopping the application and examining its status.

1) ***start.sh*** starts your web tier and application tier instances and initializes your S3 storage. Note that the start.sh script should start only one instance of the application tier, as the others application tier instances should be started on demand.
2) ***stop.sh*** stops all your instances. Note that your instances should be all stopped when you are not using them and when you submit your implementation for grading. The data generated by the current run of CloudPi should be kept in S3.
3) ***list_instances.sh*** list all your instances' IDs and their current CPU usages, one instance per line.
4) ***list_data.sh*** list all the input-output pairs stored in S3, one input-output pair per line. Note that it should list the results from only the current run of CloudPi.

3. Submission

You need to submit your implementation of CloudPi by March 14th 15:9:26. No late submissions will be accepted. You submission should be **a single zip file** that is named by the full names of your team members and that contains the following folders and files.

Requirements:

1) A folder named "**web-tier**" for the source code of your web tier program. If you use any makefile or build file, include it as well. Include a plain text README file to explain how to deploy and run your web tier program. Do not include any program that is not developed by you. Do not include any binary files (executables, objects, classes, etc.)
2) A folder named "**app-tier**" for the source code of your application tier program. If you use any makefile or build file, include it as well. Include a plain text README file to explain how to deploy and run your application tier program. Do not include any program that is not developed by you. Do not include any binary files (executables, objects, classes, etc.)

3) A folder named "**misc**" for any other source code that you implement for CloudPi. If you use any makefile or build file, include it as well. Include a plain text README file to explain how to deploy and run your web tier program. Do not include any program that is not developed by you. Do not include any binary files (executables, objects, classes, etc.)
4) A folder named "**scripts**" that contains your four SHELL scripts.
5) A plain text file named "**credentials**" that contains the access key for running your programs.
6) A plain text **README** file that lists your group member names, the public IP of your web tier, and any additional information that can help the instructor understand your code and give you points.

Failure to follow the above submission instructions will cause penalty to your grade.

4. Policies

1) Late submissions will absolutely not be graded (unless you have verifiable proof of emergency). It is much better to submit partial work on time and get partial credit for your work than to submit late for no credit.
2) Each group needs to work independently on this exercise. We encourage high-level discussions among groups to help each other understand the concepts and principles. However, code-level discussion is prohibited and plagiarism will directly lead to failure of this course. We will use anti-plagiarism tools to detect violations of this policy.