Cloud Computing
Summer Term 2012

Björn Lohrmann
bjoern.lohrmann@tu-berlin.de
FG CIT

Project Assignment No. 2

DUE: 31.05.2012 23:59

The primary goal of this exercise is to gain experience in writing, deploying and monitoring an elastically scaling web application using Amazon Web Services

1.Prerequisites

- If you are unable to complete the assignment with the first Amazon AWS Grant Code your group has received, contact Björn Lohrmann via email to receive another one.
- Additional to the API and AMI tools from the previous assignment, you will also require the AS, CW and ELB tools to complete this assignment.
- Using the Amazon EC2 online documentation you can find out how to setup and use the tools.

2.Creating a Web Service AMI

First, write a Java server that offers a stateless web service which determines with absolute certainty whether a given 64bit number is prime or not. To test your service write a Java client that runs N threads which submit random 64bit numbers to the service in an endless loop. The client should accept an HTTP URL and the number of threads to run as command-line parameters.

Second, create an instance-store backed, 64bit Linux AMI that starts your web service automatically while booting.

Hints:

- In the additional materials you will find the following:
  - Java code that uses the Apache CXF framework to implement both a web service server and client. You only have to fill in the primality test algorithm and the client code to generate and submit random numbers, everything else is provided.
  - A Maven `pom.xml` that can be used to build your code. If you install Maven and run `mvn package` from the shell, it will produce a JAR file under `target/prime-service-0.1.jar` containing your compiled code.
  - Wrapper shell scripts `prime-client.sh` and `prime-server.sh` to start both client and server from a shell. The server script runs the web service as a background process so it can be invoked at boot time inside an EC2 instance. For the wrapper shell scripts to work, several jar files are required (read `lib/PUT_JARS_HERE`).
  - In Linux, services are usually started via scripts under `/etc/init.d/` The file `primeserver` contains a Ubuntu-Linux-compatible init.d-script. It

assumes that a system user called „primeserver" exists (you will have to create that user in your AMI) and that the server shell-script is placed under `/home/primeserver/prime-service/prime-server.sh`. For security reasons it will start the web service with normal user privileges, as running services with root privileges is a potential security risk.

- If you deploy everything correctly you should be able to start the service from a root-shell with `/etc/init.d/primeserver start`
- To make Ubuntu start your service at boot time you can use the command `update-rc.d primeserver defaults`

3. Creating an Elastically Scalable Web Service Deployment

Use the AS, CW and ELB tools to create an elastically scaling deployment of the primality test web service. Clients should submit their requests to a load balancer that distributes the requests among a pool of `m1.small` instances running the web service. When the average CPU utilization of the web service instances becomes >=90% your setup should automatically add another EC2 instance to process incoming requests. If the average CPU utilization drops to <= 50% it should remove EC2 instances. Scaling up and down the number of instances must be possible without any manual intervention. Optionally, you can also create your own metric (e.g. response time) and add/remove instances based on it. The minimum number of instances in your scaling group shall be one and the maximum shall be six. Make sure that between subsequent scale-up actions there is enough time for the instances to boot and be included by the load balancer.

Test that your setup can scale up and down using separate EC2 instances (outside of your scaling group), where you manually start web service clients. By varying the number of client threads you can adjust the request load. If you want to push your scaling group to the maximum, several EC2 instances with clients may be necessary.

Make sure to keep a listing of the AS/CW/ELB tool shell commands you executed, as they are part of your submission (see section „Submission Deliverables). Remember not to include any private information such as your access key or secret key in your submission (replace them with dummy strings).

As part of the project assignment you will give a live demonstration of your setup on 01.06.2012 12-14 in FR6043, so make sure you keep your AMI(s), scaling group and load balancer and other parts of the setup ready for the live demonstration. Hint: With `as-suspend-processes` and `as-resume-processes` you can suspend/resume the scaling behavior of your scaling groups.

4. Submission Deliverables

Your submission on ISIS should be a single .zip file containing the following:

- The Java code of your server and client from Section 2.
- A .txt or .pdf file containing...
  - the AWS shell commands you used in Section 3.
  - (optionally) a short description of the custom metric used in Section 3

To complete this assignment your group also has to give a live demonstration of your setup at 01.06.2012 12-14 in FR6043.