Cloud Computing - Exercise 2
Amazon EC2 - Scalable Web Service

Florian Feigenbutz - 346141
Tim Strehlow - 316594
Till Rohrmann - 343756

May 30, 2012

1 Creating the AMI

We enhanced our custom AMI from exercise one by executing the following steps:

1. Use AMI created in exercise 1:

```
$ ec2-describe-images -o self

IMAGE ami-8f80bbfb 228315521052/AMI for Assignment 1
228315521052 available private
x86_64 machine aki-62695816instance-store
paravirtual xen
```

2. Start the instance to enhance the AMI

```
$ ec2-run-instances ami-8f80bbfb -k ubuntu-development
--region eu-west-1 --availability-zone eu-west-1a -t m1.small

INSTANCE i-cf7dd587 ami-8f80bbfb
pending ubuntu-development 0 m1.small
2012-05-30T16:39:34+0000 eu-west-1a aki-62695816
monitoring-disabled instance-store paravirtual
xen sg-2c54a75b default
```

3. Get instance address

```
$ ec2-describe-instances i-cf7dd587

INSTANCE i-cf7dd587 ami-8f80bbfb ec2-46-137-155-33.eu-west-1.compute.amazonaws.com
ip-10-59-53-98.eu-west-1.compute.internal running ubuntu-development
```

```
0 m1.small 2012-05-30T16:39:34+0000 eu-west-1a aki-62695816
monitoring-disabled 46.137.155.33 10.59.53.98 instance-store paravirtual
xensg-2c54a75b default
```

4. Connect to the machine

```
$ ssh -i [KEY] ubuntu@ec2-46-137-155-33.eu-west-1.compute.amazonaws.com
```

5. Install software

```
ubuntu@ip-10-59-53-98:~$ sudo apt-get install maven2 openjdk-7-jdk
```

6. Add user

```
ubuntu@ip-10-59-53-98:~$ sudo adduser primeserver
```

7. Copy files to primeserver user

```
$ scp -i [KEY] CC_SS12_Blatt2_additional_material.zip
ubuntu@ec2-46-137-155-33.eu-west-1.compute.amazonaws.com
```

8. Unzip files

```
ubuntu@ip-10-59-53-98:~$ sudo -u primeserver
  mv CC_SS12_Blatt2_additional_material.zip /home/primeserver
ubuntu@ip-10-59-53-98:~$ cd /home/primeserver
ubuntu@ip-10-59-53-98:~$ sudo -u primeserver
  unzip CC_SS12_Blatt2_additional_material.zip
```

9. Register web service for auto start

```
ubuntu@ip-10-59-53-98:~$ sudo update-rc.d primeserver defaults
```

10. Change region in AMI manifest

```
ubuntu@ip-10-59-53-98:~$ sudo ec2-migrate-manifest -c /tmp/cert.pem
-k /tmp/pk.pem -m /tmp/image.manifest.xml --region eu-west-1
-a ACCESS_KEY -s SECRET_ACCESS_KEY
```

11. Bundle image

```
ubuntu@ip-10-59-53-98:~$ ec2-bundle-vol -k /mnt/keys/pk.pem
  -u [USER_NUMBER] -c /mnt/keys/cert.pem
```

12. Upload AMI

```
ubuntu@ip-10-59-53-98:~$ ec2-upload-bundle --debug -b assignment2-ami
  -m /tmp/image.manifest.xml --access-key [ACCESS_KEY]
  --secret-key [SECRET_KEY] --url http://s3.amazonaws.com
```

13. Register AMI

```
ubuntu@ip-10-59-53-98:~$ ec2-register
  assignment2-ami/image.manifest.xml -n "AMI for Assignment 2"

IMAGE ami-2f86bc5b
```

2 Setting up Auto Scaling with Alarms and Load Balancing

Within the next steps we configured our web service to automatically scale up and down using a load balancer, alarms and auto scaling. Therefore we used the command line tools for AWS Elastic Load Balancing, AutoScaling and Cloud Watch. In order to avoid passing the access and secret key for each command we adapted the file `AWS_AUTO_SCALING_HOME`/credential-file-path.template and referenced it using the environment variable `AWS_CREDENTIAL_FILE`.

1. Set required environment variables for ELB and Cloud Watch

```
$ export AWS_ELB_URL=https://elasticloadbalancing.eu-west-1.amazonaws.com
$ export AWS_CLOUDWATCH_URL=https://monitoring.eu-west-1.amazonaws.com
```

2. Create ELB

```
$ elb-create-lb assignment2LB --availability-zones eu-west-1a
  --listener "protocol=http, lb-port=9000, instance-port=9000"

DNS_NAME  assignment2LB-184147699.eu-west-1.elb.amazonaws.com
```

3. Create Launch Configuration

```
$ as-create-launch-config assignment2LC --image-id ami-2f86bc5b
  --instance-type m1.small --key ubuntu-development

OK-Created launch config
```

4. Create Auto Scaling Group with at least 1 up to 6 Machines

```
$ as-create-auto-scaling-group assignment2ASG
  --launch-configuration assignment2LC  --availability-zones eu-west-1a
  --min-size 1 --max-size 6 --load-balancers assignment2LB

OK-Created AutoScalingGroup
```

5. Create Scaling Policy to Increase Number of Instances on High Load

```
$ as-put-scaling-policy assignment2UpScalePolicy
  --auto-scaling-group assignment2ASG --adjustment=1
  --type ChangeInCapacity --cooldown 300

arn:aws:autoscaling:eu-west-1:228315521052:scalingPolicy
  :28307e72-b97b-461c-b38c-331e8cd3bab4
  :autoScalingGroupName/assignment2ASG:policyName/assignment2UpScalePolicy
```

6. Create Watch Alarm for High CPU Load

```
$ mon-put-metric-alarm assignment2HighCPUAlarm
  --comparison-operator GreaterThanOrEqualToThreshold --evaluation-periods 1
  --metric-name CPUUtilization --namespace "AWS/EC2" --period 120
  --statistic Average --threshold 90
  --alarm-actions arn:aws:autoscaling:eu-west-1:228315521052:scalingPolicy
    :28307e72-b97b-461c-b38c-331e8cd3bab4:autoScalingGroupName/assignment2ASG:
    policyName/assignment2UpScalePolicy
  --dimensions "AutoScalingGroupName=assignment2ASG"

OK-Created Alarm
```

7. Create Scaling Policy to Reduce Number of Instances on Low Load

```
$ as-put-scaling-policy assignment2DownScalePolicy
  --auto-scaling-group assignment2ASG --adjustment=-1
  --type ChangeInCapacity --cooldown 300

arn:aws:autoscaling:eu-west-1:228315521052:scalingPolicy
  :2d9dad1a-e996-4dab-8cb4-c63346155b66
  :autoScalingGroupName/assignment2ASG:policyName/assignment2DownScalePolicy
```

8. Create Watch Alarm for Low CPU Load

```
$ mon-put-metric-alarm assignment2LowCPUAlarm
  --comparison-operator LessThanOrEqualToThreshold --evaluation-periods 1
  --metric-name CPUUtilization --namespace "AWS/EC2" --period 120
  --statistic Average --threshold 50
  --alarm-actions arn:aws:autoscaling:eu-west-1:228315521052
    :scalingPolicy:2d9dad1a-e996-4dab-8cb4-c63346155b66
    :autoScalingGroupName/assignment2ASG:policyName/assignment2DownScalePolicy
  --dimensions "AutoScalingGroupName=assignment2ASG"

OK-Created Alarm
```

3 Test the Web Service Scaling using EC2 Instances as Clients

Next we used another EC2 instance as a client to test the scalability of our infrastructure we set up within the last section.

1. Start the Instance to Enhance the AMI

```
$ ec2-run-instances ami-2f86bc5b -k ubuntu-development
  --region eu-west-1 --availability-zone eu-west-1a -t m1.small

RESERVATION r-8a1546c3 228315521052 default
INSTANCE i-0946ee41 ami-2f86bc5b pending
ubuntu-development 0 m1.small 2012-05-30T18:03:23+0000
eu-west-1a aki-62695816 monitoring-disabled
instance-store paravirtual xen sg-2c54a75b default
```

2. Get Instance Address

```
$ ec2-describe-instances i-0946ee41

RESERVATION r-8a1546c3 228315521052 default
INSTANCE i-0946ee41 ami-2f86bc5b ec2-176-34-95-58.eu-west-1.compute.amazonaws.com
ip-10-248-81-158.eu-west-1.compute.internal running ubuntu-development
0 m1.small 2012-05-30T18:03:23+0000 eu-west-1a aki-62695816
monitoring-disabled 176.34.95.58 10.248.81.158
instance-store paravirtual xensg-2c54a75b default
```

3. Connect to the Machine

```
$ ssh -i [KEY] ubuntu@ec2-176-34-95-58.eu-west-1.compute.amazonaws.com
```

4. Start the Client

```
ubuntu@ip-10-248-81-158:~$ ~/assignment2/prime-client.sh
  http://assignment2LB-184147699.eu-west-1.elb.amazonaws.com:9000/PrimeService 10
```

5. Periodically Check the Number of Assigned EC2 Instances

```
$ watch -n 30 elb-describe-instance-health assignment2LB

Every 30.0s: elb-describe-instance-health assignment2LB
Wed May 30 19:27:32 2012

INSTANCE_ID  i-49228a01  InService  N/A  N/A
INSTANCE_ID  i-5f208817  InService  N/A  N/A
```

```
INSTANCE_ID  i-3554fc7d  InService  N/A  N/A
INSTANCE_ID  i-932d85db  InService  N/A  N/A
INSTANCE_ID  i-532c841b  InService  N/A  N/A
INSTANCE_ID  i-4329810b  InService  N/A  N/A
```

6. Check Currently Active Instances

```
$ ec2-describe-instances -F instance-state-code=16

RESERVATION r-e40e5dad 228315521052 default
INSTANCE i-49228a01 ami-2f86bc5b ec2-79-125-66-78.eu-west-1.compute.amazonaws.com
...
```

5 Conclusion

With AWS Elastic Load Balancing, Auto Scaling Groups and Cloud Watch deploying a scalable and reliable infrastructure becomes quite comfortable. Such a setup can be configured in many parameters to fit the project needs in terms of scalability and performance. In our example we only used the CPU usage as a load indicator because our algorithm mostly uses CPU and therefore response times will nearly always equals the system machines' CPU load. Nevertheless one could imaging other scenarios in which it would be very helpful to track more then a single indicator such e.g. memory or I/O usage or even rely on indicators that correlate with the end users' experience of the system performance.