

Instance Installation Guide

Picking suitable instance to start

This is instruction for installing instance with the framework and MediaWiki to allow dynamical allocation and servers monitoring. This instruction will only use one instance to deploy all the necessary services. We are using Linux Ubuntu 11.04 (Natty) 32bit. List of daily builds of Ubuntu can be found from Ubuntu homepage: <http://cloud-images.ubuntu.com/releases/natty/release/>. User needs to have Amazon account, that is capable of starting new instances in the cloud, otherwise these instructions cannot be followed.

We selected out one of the images from the list (looking correct region and correct region). Because we are using c1.medium instance (various testing and experiments conducted have shown it to be most useful instance type), we need to use 32 bit architecture. Make sure where root of the image is stored, we are interested in instance, which have root stored on instance (otherwise we need to use S3 storage to mount the root).

Suitable instance to start with is with ID **ami-ffc01996**. There are several ways to start the instance. One is to use command line, but it needs to have package **ec2-api-tools** to execute command **ec2-run-instances**. Other option is to use Amazon AWS console to start and terminate instance, Ubuntu community gives also direct link to do so: <https://console.aws.amazon.com/ec2/home?region=us-east-1#launchAmi=ami-ffc01996>.

Run instance from command line with following command, also install needed tool to run it:

```
LOCAL$ sudo apt-get install ec2-api-tools
LOCAL$ ec2-run-instances ami-ffc01996 --instance-type m1.small --region us-east-1 --key ${EC2_KEYPAIR} -K ${EC2_PRIVATE_KEY} -C ${EC2_CERT}
RESERVATION      r-e5f36587  728681021428      default
INSTANCE   i-1bc7447d  ami-ffc01996      pending      key_pair      0
            m1.small      2012-05-13T13:46:29+0000      us-east-1b  aki-407d9529
            monitoring-disabled      instance-store
```

Notice, that we can use **m1.small** instance to install the necessary software as it will save little bit money for running the instance and installing the software. Also first tests are recommended to do with **m1.small** instance, as if there might be errors in the configuration, it will cost less to reconfigure the cloud again and run the experiments. If everything works as expected, it is possible to move to larger instance. **\${EC2_KEYPAIR}** is the name of the key-pair generated in the Amazon, not the key file. This will tell the instance, which public-private key to use to allow access to instance through Secure Shell.

If you have any problems to start the instance, follow these instructions:

<http://www.youtube.com/watch?v=rYJLIfVuSMY&lr=1> and <http://www.hongkiat.com/blog/amazon-s3-the-beginners-guide/>.

If the command is successful, it will return executed instance ID, this can be used to track, if instance is running or not. Using following command, you can have overview of your instances in the cloud:

```
LOCAL$ ec2-describe-instances -K ${EC2_PRIVATE_KEY} -C ${EC2_CERT}
```

```
RESERVATION      r-e5f36587  728681021428      default
INSTANCE  i-1bc7447d  ami-ffc01996      ec2-23-20-81-231.compute-1.amazonaws.com
            domU-12-31-39-0C-24-74.compute-1.internal  running      martti.v      0
            m1.small    2012-05-13T13:46:29+0000      us-east-1b  aki-407d9529
            monitoring-disabled      23.20.81.231      10.215.39.130
            instance-store
```

This will show state of the requested instance: pending, running, terminated etc. First the instance is pending, as the image is copied to physical host to start the virtual image by XEN. It takes some time, but should be done in 2..3 minutes. If the state changes to running, the instance gets private and public IP addresses. Using your private key, it is possible to connect with the running instance with following command (notice that public IP is given by DNS name and also with IP4: ec2-23-20-81-231.compute-1.amazonaws.com and 23.20.81.231):

```
LOCAL$ ssh -i /home/user/your_private_key.pem ubuntu@23.20.81.231
The authenticity of host '23.20.81.231 (23.20.81.231)' can't be established.
RSA key fingerprint is 11:3a:f0:e7:d2:1e:2e:5c:e2:##:##:##:##:##:##:##.
Are you sure you want to continue connecting (yes/no)?
```

First time it will ask if you want to add it to the known host list, answer yes. Ubuntu have user **ubuntu** to access the instance, but for other operating system, this differs. SciCloud is using as a user **root** to access the instance. If you are finally in the instance, you can type **\$ sudo -i** to gain root privileges and install necessary software.

If you do not want to define for **ec2-api-tools** keys and certifications each time a command is execute, you could make a file **/home/user/amazon/amazon**, that contains necessary information. The file consist following information:

```
EC2_KEY_DIR=$(dirname $(readlink -f ${BASH_SOURCE}))
export EC2_PRIVATE_KEY=${EC2_KEY_DIR}/your_private_key.pem
export EC2_CERT=${EC2_KEY_DIR}/your_cert_key.pem
export EC2_KEYPAIR=your_keypair_name
```

If you start shell, use **source /home/user/amazon/amazon** to store the variables in the environment defined above. Now you can call **ec2-api-tools** commands without need to define the keys.

Installing software on the instance

If you have successfully started the instance and have access it, now it is time to install necessary software for the framework. We want to install necessary services to run the MediaWiki application, this includes MySQL to store data in database, memcached to store cached pages in the memory, Apache with PHP to serve **HTTP GET** requests to MediaWiki and nginx for load balancing the requests.

To install necessary services, use aptitude tool under Ubuntu. Make sure you have root privileges (look above section, **sudo -i** will grant you the necessary permissions). But first, we need to update the list of entries in the repository.

```
$ add-apt-repository "deb http://archive.canonical.com/ lucid partner"
$ apt-get update
$ apt-get install mysql-server apache2 php5 php5-mysql php5-xcache memcached
mysql-server sysstat openjdk-6-jdk collectd nginx chkconfig ec2-api-tools
```

It should download ~300 MB of packages and should take some time. It will ask also user and password for MySQL, for just testing purpose, you could use **root** for both.

We will currently install nginx from repository, but in order to have Fair module, we need to build it from source. **chkconfig** package is used to disable service at the start up, as we do not want to activate all the services for the instance. Use following commands to stop services from the boot up:

```
$ chkconfig mysql off
$ chkconfig apache2 off
$ chkconfig memcached off
$ chkconfig nginx off
```

To check, if this has worked or not, you could restart instance with command **reboot**. After reboot, try following commands and see, if the services are running:

```
$ service mysql status
$ service apache2 status
$ service memcached status
$ service nginx status
```

For next step, you might want to start the services to allow installation of the MediaWiki into database and see, if everything is working. You should not start nginx, as otherwise port conflict happens (both are by default running on port 80).

```
$ service mysql start
$ service apache2 start
$ service memcached start
```

The framework will use command line to start the memcached with correct amount of memory as the default memory is set to be 64mb and is not enough for conducting large scale experiments. Make sure that from my.cnf row **bind-address** = **127.0.0.1** is commented out.

Installing MediaWiki

If all the services are installed and started again, it is time to install MediaWiki. You can download MediaWiki from <http://www.mediawiki.org/wiki/Download>. Apache default folder to serve the content for outside world is located at `/var/www`. Go there, download MediaWiki and unpack it with following commands:

```
$ cd /var/www
$ wget http://download.wikimedia.org/mediawiki/1.19/mediawiki-1.19.0.tar.gz
$ tar -xvf mediawiki-1.19.0.tar.gz
$ mv mediawiki-1.19.0/ mediawiki
$ chmod -R 777 /var/www/mediawiki/
```

We use simplified permissions in the file system, but it would be advised to use **chown www-data** and use **chmod 700** permissions in the MediaWiki configuration file, that it will be only accessible by Apache PHP.

Installation instruction can be find from http://www.mediawiki.org/wiki/Manual:Installation_guide . Access your instance from address <http://public.ip/mediawiki/>. It will warn you, that LocalSettings.php is missing and gives link to redirect you to installation part. Use following parameters to install the MediaWiki:

```
Database host: "localhost"
Database name: "wikidb"
Database table prefix: ""
Database user: "root"
Database password: "root"
Use the same account as for installation: "false"
New database user: "wikiuser"
New database password: "pass"
Create the account if it does not already exist: "true"
Storage engine: "innodb"
Database character set: "binary"
Name of wiki: "MediaWiki experiments"
Enable outbound e-mail: "false"
Settings for object caching: "Use Memcached (requires additional setup and configuration)"
Memcached servers: "localhost:11211"
```

If not defined above, user your own values for the form fields. If the installation is successful, it will download **LocalSettings.php** into your local computer through browser. This needs to be copied to instance folder `/var/www/mediawiki/`.

```
LOCAL$ scp -i /home/user/your_private_key.pem  
/home/user/localpath/LocalSettings.php ubuntu@23.20.81.231:/var/www/mediawiki/
```

Make sure you can copy file into the folder, otherwise it is not possible and cannot start MediaWiki page.

If everything works, you can go to <http://your.ip/mediawiki/> and it should redirect you to the main page. It will consist only one page, so selecting random page should still direct you to the same page.

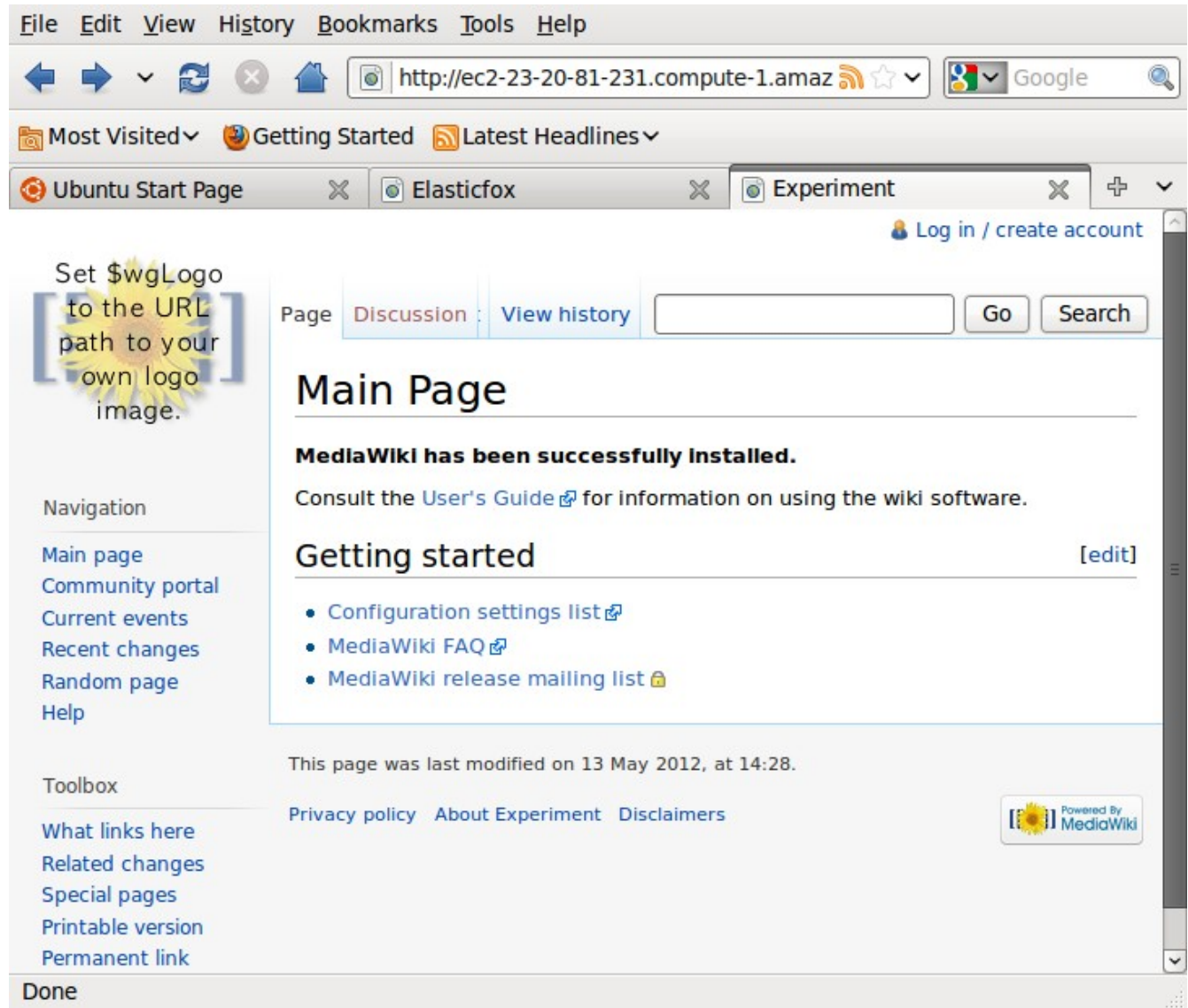


Figure 1. Successfully installed the MediaWiki 1.19.

Our experiments used older version of MediaWiki (1.16.5) and it seems, that newer MediaWiki versions are using more functionalities and are complex compared to previous versions, meaning that response time is slower.

Uploading Wikipedia dumps into MySQL database

Now we have successfully installed MediaWiki on our machine running in the cloud. It does not have necessary data as there is no point for requesting only example page (the size is small and complexity low, meaning the response time is very low). In order to get more data, we can use Wikipedia dumps to upload the data. Wikipedia makes backup copies of its databases in XML format, that can be downloaded from <http://download.wikimedia.org/enwiki>. It is in XML format if unpacked and should use program, that is capable of converting XML into MySQL queries. It can be copied from the DVD /project/mwdumper.jar. It seems that dumps used on this thesis are no longer valid anymore and are removed from the web: <http://dumps.wikimedia.org/enwiki/20110115/> is not accessible.

```
$ cd /mnt
$ chmod -R 777 /mnt
$ wget http://dumps.wikimedia.org/enwiki/20120403/enwiki-20120403-pages-articles1.xml-p000000010p000010000.bz2
$ wget math.ut.ee/~wazz/mwdumper.jar
$ java -jar mwdumper.jar --format=sql:1.5 enwiki-20120403-pages-articles1.xml-p000000010p000010000.bz2 | mysql -u root -p wikidb --password=root
1,000 pages (71.003/sec), 1,000 revs (71.003/sec)
2,000 pages (73.158/sec), 2,000 revs (73.158/sec)
3,000 pages (69.756/sec), 3,000 revs (69.756/sec)
4,000 pages (69.396/sec), 4,000 revs (69.396/sec)
5,000 pages (68.945/sec), 5,000 revs (68.945/sec)
6,000 pages (67.56/sec), 6,000 revs (67.56/sec)
6,343 pages (66.047/sec), 6,343 revs (66.047/sec)
```

This dump might be also be removed from Internet, then use latest dumps to download the content. Hopefully the format does not change much and it is possible to upload it with the same program.

To test, if the data is uploaded successfully, open up browser again and visit MediaWiki page you have just installed and select “Random page” to see, if new article opens.

If the web page opens up slowly (takes 10 seconds to open), it means memcached is not correctly configured and probably the service is not started. For any troubles, make sure that all the necessary services are running.

If you want, you can upload all the dumps into the database, but make sure that there is enough room to store the data or use S3 storage to store database information. To check free space, use **df** command:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1      9.9G  1.3G  8.1G  14% /
none            828M  116K  828M   1% /dev
none            833M    0  833M   0% /dev/shm
none            833M   56K  833M   1% /var/run
```

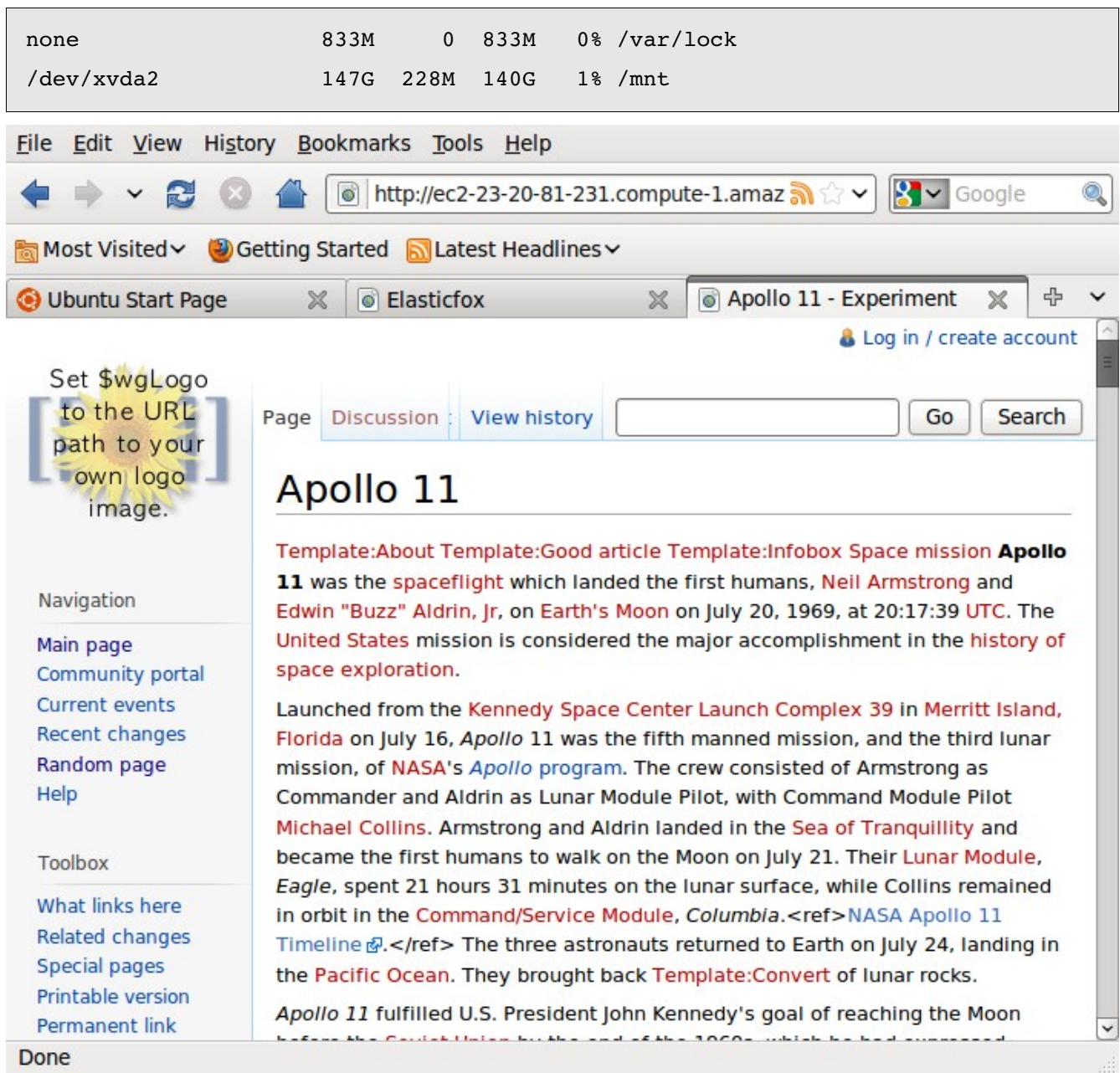



Figure 2. MediaWiki showing random page from database dump. Red links indicate missing content in the database.

Framework installation

Main part of the installation is now accomplished, site is accessible from the web, uses MySQL database and memcached to fetch data for visitors and has dump uploaded into database. Now we need to upload framework to the instance to allow dynamical allocation and monitoring servers while running the service in the cloud.

First we need to upload ServerStatus program to the cloud. It can be retrieved from DVD **/project/ServerStatus.jar**. If this is added to the instance, you have to add the program to start up script to allow starting the program while the instance has booted up.

```
$ cd /home/ubuntu
$ wget math.ut.ee/~wazz/ServerStatus.jar
$ nano /etc/rc.local
#!/bin/sh
sudo nohup java -jar /home/ubuntu/ServerStatus.jar &
exit 0
$ reboot
```

Reboot to make sure everything is working and service ServerStatus is started at the boot. You can check it by visiting page <http://your-ip:5555>.

You can also use following command to check, if ServerStatus is working:

```
$ echo "STATS" | /bin/netcat -q 2 127.0.0.1 5555
CPU: 92.57 0.54 0.00 0.00 2.56 4.33
MEM: 2187 798 71796 1741980 2713604 426188 2631648
HD: sda 22976815 62773520 0.00/s 0.00/s
NET: 665970122 207735342 665.970 207.735 0.000KB/s 0.000KB/s 0.000MB/s
0.000MB/s
LOAD: 0.58 0.50 0.50
PS: 196
APACHE: 4
SERVER: 0.000 1 1 48553 48552
VISITED: false
LAST: 1337027954463
DIFF: 15.340 sec
```

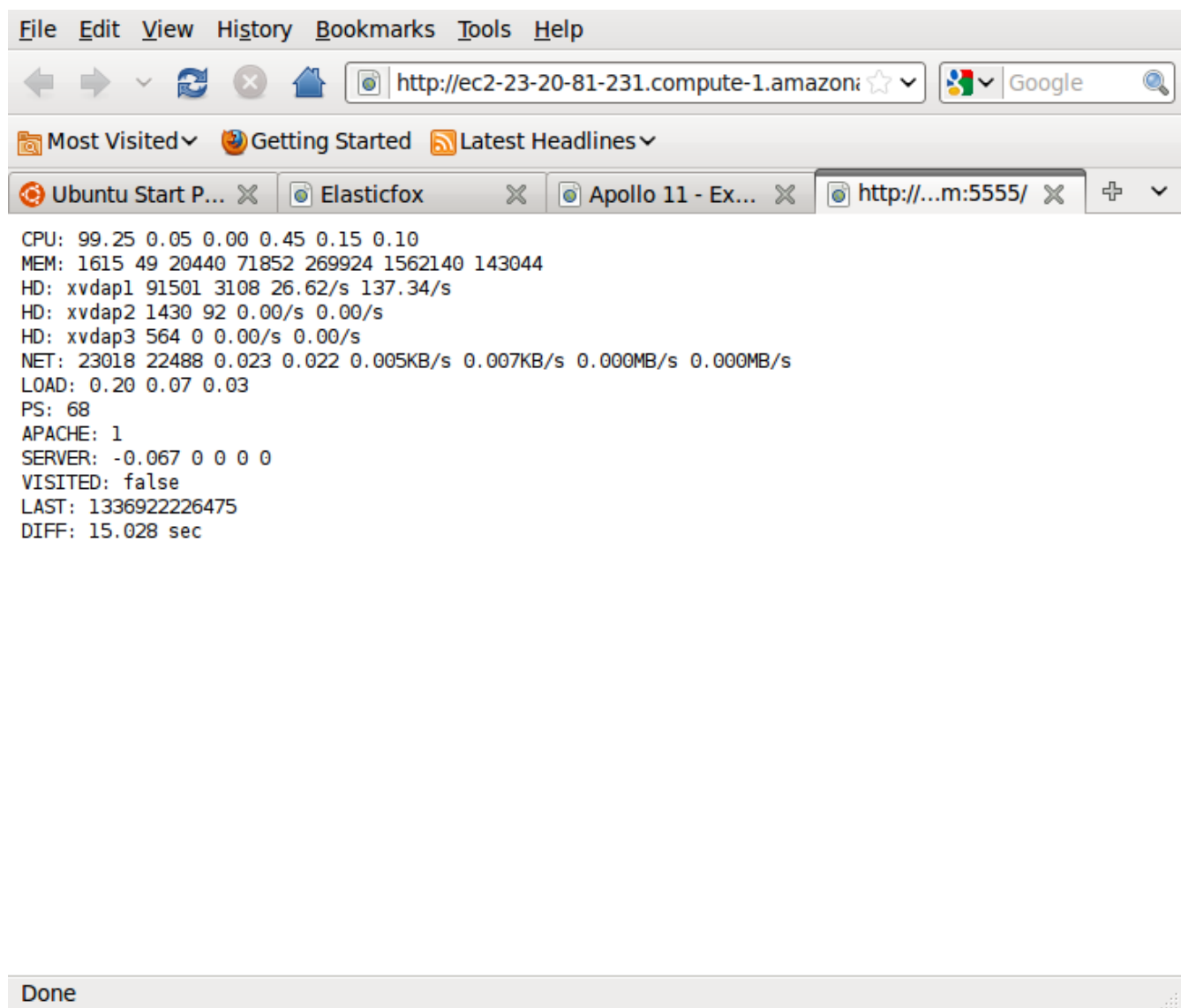



Figure 3. Showing ServerStatus collecting data

Bundling image together

This should be all that is needed for our framework. Now it is time to bundle the image together, so it would be possible to run several instances with the same configuration. Following are the commands necessary to run to create new image (we need our keys to be copied to the cloud in order to run these commands, use `/mnt` folder for bundling and key storing as this folder is not saved in the bundled image):

```
$ sudo apt-add-repository ppa:awstools-dev/awstools
$ sudo apt-get update
$ sudo apt-get install ec2-ami-tools
LOCAL$ scp -i your_key.pem your_key.pem your_cert.pem ubuntu@23.20.81.231:/mnt/
# BUNDLE THE IMAGE
$ ec2-bundle-vol -d /mnt -k /mnt/your_key.pem -c /mnt/your_cert.pem -u
728681021428 -r i386 -p ubuntu-11-nginx-sql-mediawiki
# UPLOAD TO THE CLOUD
ec2-upload-bundle -b ami_dsg -m /mnt/ubuntu-11-nginx-sql-mediawiki.manifest.xml
-a ${EC2_ACCESS_KEY} -s ${EC2_SECRET_KEY}
# REGISTER THE UPLOADED IMAGE
ec2-register ami_dsg/ubuntu-11-nginx-sql-mediawiki.manifest.xml -K
/mnt/your_key.pem -C /mnt/your_cert.pem -n ubuntu-11-nginx-sql-mediawiki
IMAGE      ami-325df95b
# IT WILL RETURN NEW INSTANCE ID, MARK IT UP SOMEWHERE, AS YOU ARE GOING TO
NEED IT LATER
```

You will retrieve ID of your instance, use this to launch new instances. Now we have everything done, if you want to make experiments, follow instructions below.

Our instance ID is **ami-325df95b**

Starting large scale experiment with CloudController

CloudController is used to take control of running instances and monitors the performance of the instance connecting with ServerStatus, which we previously have installed (make sure, that when new bundled image is started, the service at port 5555 is accessible through web browser).

If you want to dynamically allocate servers in the cloud, make sure that **ec2-api-tools** package is installed, as this will provide functionality to list already running instances, request new instances and remove running instances.

```
$ apt-get install ec2-api-tools
```

As we have installed collectd on the instance, it is possible to collect performance metrics from collectd also (we still can use ServerStatus information or compare the collected values). Use **/commands/performance/download_rrd_files/rrdcloud.py** script to download RRD files from the cloud (change configuration options in the file). It would be good idea to delete some of the metrics plugins under **/etc/collectd/collectd.conf** to reduce files needed to download to local computer.

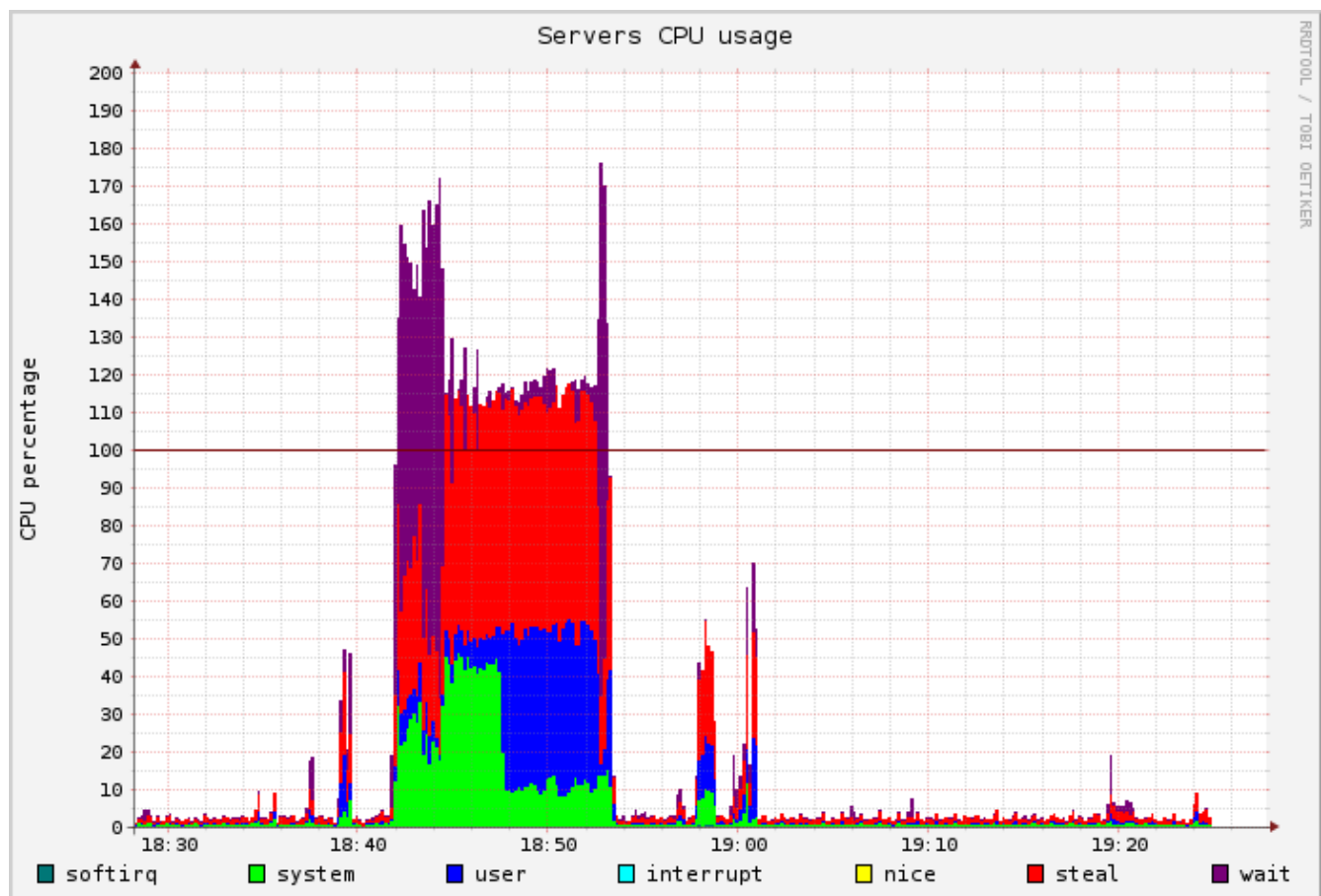


Figure 4. Showing CPU usage with collectd, while the image was bundled together, notice large **steal** usage.



Figure 5. Showing CPU usage with ServerStatus, while the image was bundled together.

Our new instance (which we bundled together and uploaded to cloud in previous step) ID is **ami-325df95b**. At minimum, at least 5 instances are needed to run the experiments. The instances have following roles: 1 Nginx, 1 MySQL, 1 memcached, 1 worker (BenchMark) and 1 Apache server. If you want to reduce cost of the experiments, it is possible to combine several service types together. We could combine nginx, MySQL, memcached and worker into one instance (as they are using least amount of CPU and can be run on the same machine) and Apache instances are running on different machines. This can affect service speed as if the first server starts to become saturated, it means all the services are saturated then (nginx, MySQL, memcached and BenchMark).

```
LOCAL$ ec2-run-instances ami-325df95b --instance-type m1.small -n 5 --region
us-east-1 --key ${EC2_KEYPAIR}
```

While the servers start to come online, select one instance as primary instance. Copy CloudController.tar.gz from **/project/** in to that instance.

Make sure, that you change **configuration.txt** file in the CloudController, as it will contain information, where to find the private key, which instance to use and what service time you have.

Configuration.txt file:

```
instance_ami_id ami-325df95b
# keypair name, not the file, when requesting new instance
key martti.v
# reduce time to increase and remove servers (in seconds)
change_servers 1800
# credential information, where is the key and cert file located, look above of
# the instruction where source /amazon/amazon was mentioned to see the content
# of the file
credential configuration/amazon/amazon
# allow adding and removing servers
use_autoconfiguration true
experiment_length 7200
private_key configuration/your_private_key.pem
```

```
servers_count_min 1
servers_count_max 3
```

Run following commands:

```
LOCAL$ scp -i /home/ubuntu/your_private.key ThesisCloudController.tar.gz
ubuntu@20.11.22.33:/mnt/
# OR
$ cd /mnt/
$ wget math.ut.ee/~wazz/ThesisCloudController.tar.gz
$ tar -xvf ThesisCloudController.tar.gz
$ java -jar CloudController.jar --configure -auto
```

It should automatically assign server roles into **servers.txt** and configure instances. Make sure that MySQL can be accessed from other instances (in **my.cnf**, comment out bind 127.0.0.1).

Connect with WORKER and start BenchMark (fill the cache first), CloudController configuration will automatically deploy necessary software there in **/mnt** folder. You can look roles from **servers.txt** file.

```
WORKER$ sudo -i
WORKER$ cd /mnt
WORKER$ java -jar BenchMark.jar --sut 10.210.47.209 --memcached 10.208.122.86
--prefix cache_ --min-rps 1 --max-rps 1 --url-file url.txt --curve-file
ramp_curve.txt --random-seed 0 --time 7200 --timeout 10000
```

It might happen, that the dumps are different and therefore the URLs there will output 404 errors. To make new URL file, connect with MySQL database:

```
MYSQL$ mysql -u root -p > output.txt
> use wikidb;
> select concat('/mediawiki/index.php?title=', page_title) from page where
page_is_redirect = '0' limit 100;
> exit
```

Use this new list in output.txt as **-url-file** in BenchMark.jar. Run it as long as for the last 60 seconds the cache hit ratio is at least 90% (if you have gone several times through the list, it means that the configuration is wrong and it does not cache the requests correctly). If it is over that, then you can proceed with the experiment. First start the CloudController in the first instance (where you copied all the things) and in the same time execute BenchMark:

```
WORKER$ cd /mnt
WORKER$ nohup java -jar BenchMark.jar --sut 10.210.47.209 --memcached
10.208.122.86 --prefix cache_ --min-rps 1 --max-rps 40 --url-file url.txt
--curve-file ramp_curve.txt --random-seed 1 --time 7200 --timeout 10000 >
worker_output.txt &
```

```
WORKRE$ tail -f worker_output.txt
CONTROLLER$ cd /mnt
CONTROLLER$ nohup java -jar CloudController.jar > output.txt &
```

If the experiment ends (after 2 hours), download or copy **HTTPServer.jar** and start server on port 8080 and mounted to **/mnt** folder to allow easy access for the results (it is also on DVD under **/project**).

```
$ cd /mnt
$ wget math.ut.ee/~wazz/HTTPServer.jar
$ nohup java -jar HTTPServer.jar 8080 10 /mnt true &
# and pack everything together
$ tar -cvzf output.tar.gz *.tx* *.ta*
```

Make sure, that you have opened up port 8080 from AWS console to access the instance. You should find out file **output.tar.gz** and can download it to local machine.

Example output of BenchMark.jar:

```
[1.004] Lo:1(1) Su:1 Fa:0 Ti:0 Av:160.000 LoT:1 SuT:1 FaT:0 TiT:0 AvT:160.000
[2.009] Lo:1(1) Su:0 Fa:0 Ti:0 Av:NaN LoT:2 SuT:1 FaT:0 TiT:0 AvT:160.000
[3.007] Lo:1(1) Su:1 Fa:0 Ti:0 Av:186.000 LoT:3 SuT:2 FaT:0 TiT:0 AvT:173.000
[4.002] Lo:1(1) Su:1 Fa:0 Ti:0 Av:225.000 LoT:4 SuT:3 FaT:0 TiT:0 AvT:190.333
[5.003] Lo:1(1) Su:1 Fa:0 Ti:0 Av:241.000 LoT:5 SuT:4 FaT:0 TiT:0 AvT:203.000
[6.002] Lo:1(1) Su:1 Fa:0 Ti:0 Av:206.000 LoT:6 SuT:5 FaT:0 TiT:0 AvT:203.600
[7.010] Lo:1(1) Su:1 Fa:0 Ti:0 Av:227.000 LoT:7 SuT:6 FaT:0 TiT:0 AvT:207.500
[8.003] Lo:1(1) Su:1 Fa:0 Ti:0 Av:171.000 LoT:8 SuT:7 FaT:0 TiT:0 AvT:202.286
[9.003] Lo:1(1) Su:2 Fa:0 Ti:0 Av:161.000 LoT:9 SuT:9 FaT:0 TiT:0 AvT:193.111
...
[30.006] Lo:1(1) Su:1 Fa:0 Ti:0 Av:163.000 LoT:30 SuT:29 FaT:0 TiT:0
AvT:190.552
[30] Memcached: 176/237 (74.26%) Total: 176/237 (74.26%)
[31.006] Lo:1(1) Su:1 Fa:0 Ti:0 Av:170.000 LoT:31 SuT:30 FaT:0 TiT:0
AvT:189.867
```

Example output of CloudController.jar:

```
Cloud controller started!
Loading configuration file
Loading server file
Found instance 10.210.47.209 NGINX
Found instance 10.192.167.80 SQL
```


Found instance 10.208.122.86 MEMCACHED

Found instance 10.208.177.189 WORKER

-> stats unknown 10.192.167.80 SQL true (CPU: 0.40, 0.00 0.01 0.05) RPS: -0.07
active: 0 idle: 0 mem_used: 52 mem_free: 1612 net_sent: 14807278 net_recv:
13550

72 reads: 0 writes: 0

Found instance 10.209.94.178 APACHE

Testing configuration

Setting up metrics algorithm to determine the count of servers needed

Checking additional Apache servers from the cloud

-> stats unknown 10.208.122.86 MEMCACHED true (CPU: 0.60, 0.04 0.26 0.28) RPS:
-0.07 active: 0 idle: 0 mem_used: 46 mem_free: 1619 net_sent: 436838 net_recv:
275674 reads: 0 writes: 0

-> stats unknown 10.208.177.189 WORKER true (CPU: 2.29, 0.00 0.01 0.05) RPS:
-0.07 active: 0 idle: 0 mem_used: 38 mem_free: 1626 net_sent: 341670 net_recv:
6036601 reads: 0 writes: 0

-> stats unknown 10.210.47.209 NGINX true (CPU: 0.45, 0.08 0.04 0.05) RPS:
-0.07 active: 0 idle: 0 mem_used: 51 mem_free: 1613 net_sent: 6568013 net_recv:
18200370 reads: 0 writes: 0

-> stats unknown 10.209.94.178 APACHE true (CPU: 0.10, 0.00 0.01 0.05) RPS:
0.07 active: 1 idle: 5 mem_used: 74 mem_free: 1590 net_sent: 2996836 net_recv:
15541457 reads: 0 writes: 0

Found already existing server NGINX with ip '10.210.47.209', id 'i-9b77f5fd'
and status RUNNING

Found already existing server SQL with ip '10.192.167.80', id 'i-9977f5ff' and
status RUNNING

Found already existing server MEMCACHED with ip '10.208.122.86', id 'i-
6774f601' and status RUNNING

Found already existing server WORKER with ip '10.208.177.189', id 'i-6574f603'
and status RUNNING

Found already existing server APACHE with ip '10.209.94.178', id 'i-6374f605'
and status RUNNING

-> nginx Reading: 0 Writing: 1 Waiting: 0

-> nginx cannot calculate arrival rate for the first measurment

-> nginx_minute -1.000 rps arr2: 159761 arr1: -1

-> nginx Reading: 0 Writing: 1 Waiting: 0

-> nginx cannot calculate arrival rate for the first measurment

-> nginx reset has been registered, using minute counter for arrival rates

-> nginx_hour -1.000 rps arr2: 159762 arr1: -1 (for hour)

Starting experiment for 2h 0m 0s

[30/7200] Gathering statistics

-> stats i-9b77f5fd 10.210.47.209 NGINX true (CPU: 0.60, 0.53 0.17 0.10) RPS:
-0.07 active: 0 idle: 0 mem_used: 57 mem_free: 1607 net_sent: 7049355 net_recv:

18682073 reads: 0 writes: 2872

-> stats i-6774f601 10.208.122.86 MEMCACHED true (CPU: 0.65, 0.02 0.22 0.27)
RPS: -0.07 active: 0 idle: 0 mem_used: 45 mem_free: 1619 net_sent: 544054
net_recv: 319400 reads: 0 writes: 2496

-> stats i-6574f603 10.208.177.189 WORKER true (CPU: 1.61, 0.00 0.01 0.05)
RPS: -0.07 active: 0 idle: 0 mem_used: 46 mem_free: 1619 net_sent: 376144
net_recv: 6381144 reads: 0 writes: 1504

-> stats i-9977f5ff 10.192.167.80 SQL true (CPU: 0.80, 0.00 0.01 0.05) RPS:
-0.07 active: 0 idle: 0 mem_used: 52 mem_free: 1612 net_sent: 20370486
net_recv: 1778751 reads: 0 writes: 3076

-> stats i-6374f605 10.209.94.178 APACHE true (CPU: 8.50, 0.00 0.01 0.05) RPS:
1.07 active: 1 idle: 5 mem_used: 74 mem_free: 1590 net_sent: 3842032 net_recv:
20375615 reads: 0 writes: 3192

-> ping APACHE 10.209.94.178 i-6374f605 0.415 0.462 0.544 0.054 ms
min/avg/max/mdev

-> ping MEMCACHED 10.208.122.86 i-6774f601 0.317 0.382 0.483 0.059 ms
min/avg/max/mdev

-> ping SQL 10.192.167.80 i-9977f5ff 0.376 0.408 0.430 0.023 ms
min/avg/max/mdev

-> ping NGINX 10.210.47.209 i-9b77f5fd 0.011 0.016 0.032 0.008 ms
min/avg/max/mdev

-> ping WORKER 10.208.177.189 i-6574f603 0.463 5.478 22.873 8.754 ms
min/avg/max/mdev

[60] Gathering arrival rate

-> nginx Reading: 0 Writing: 2 Waiting: 0

-> nginx_minute 1.150 rps arr2: 159830 arr1: 159761

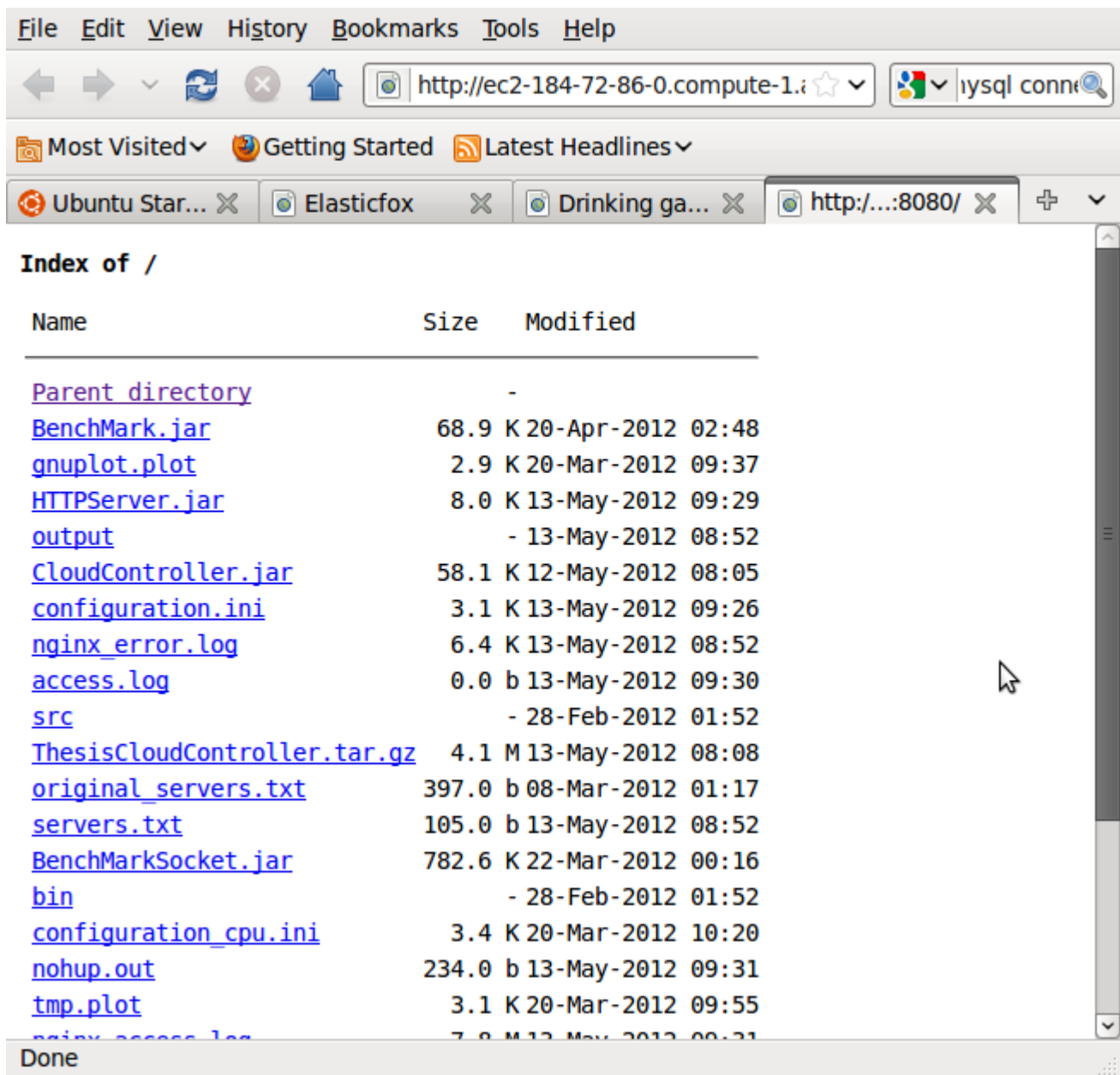


Figure 6. Showing files in /mnt folder, using HTTPServer program.

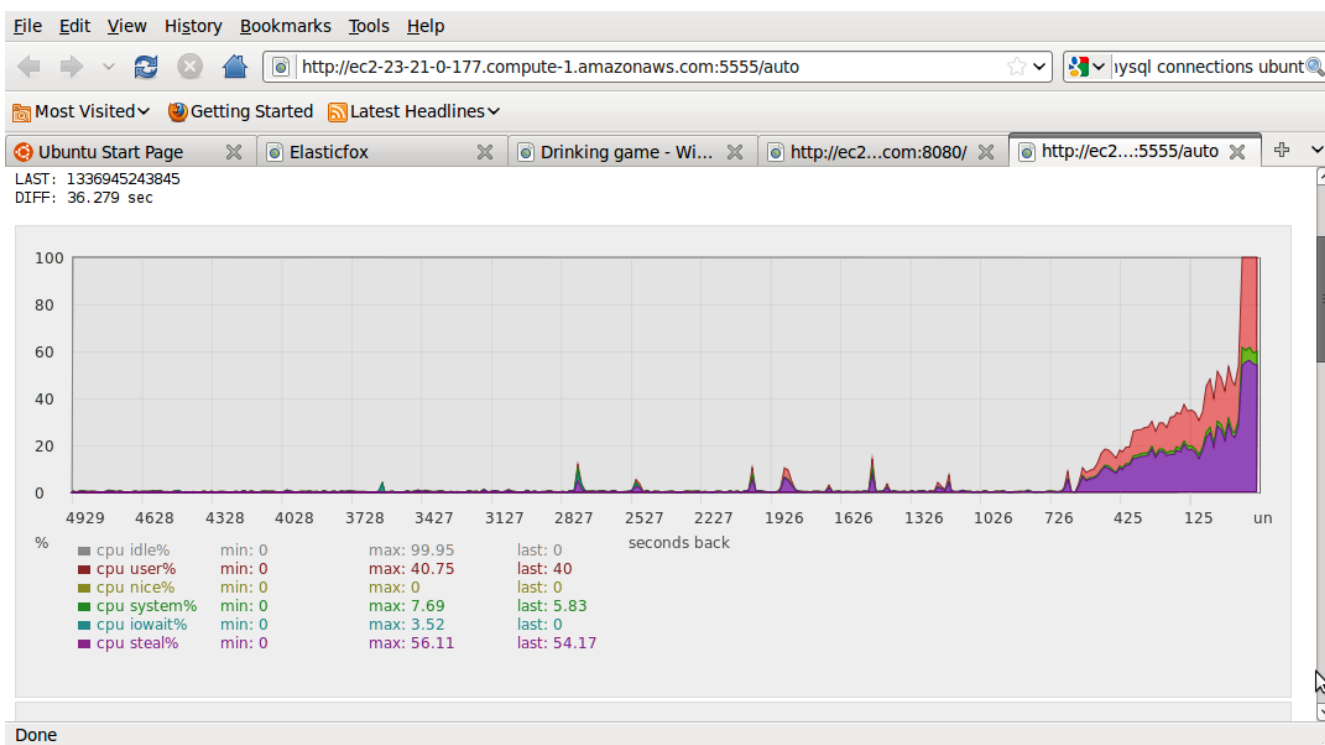


Figure 7. Showing CPU usage for **m1.small** instance. It is possible to see, that most of the CPU cycle is blocked by virtualization (purple)

Configuring server list manually

You can ignore auto configuration option (if for example **ec2-api-tools** is not installed) and configure the list manually. Only one memcached and MySQL server is allowed as other servers would sit in idle. A proper mechanism has to be made in order to divide the resources correctly and configure **LocalSettings.php** with master slave MySQL configuration and list of memcached servers. If you have done everything from previous steps, you can edit **servers.txt** in your favorable method.

Example of **servers.txt** content:

```
20.10.30.11 NGINX
20.11.31.21 SQL
20.21.32.42 MEMCACHED
20.21.33.45 WORKER
20.43.12.42 APACHE
20.32.53.43 APACHE
```

If this is done, you can call out CloudController to configure the servers and start proper services. Look **configuration/commands/** folder to see, what commands are executed or if you want to change the commands. These commands are also executed while new Apache server is added. It will allow change servers configuration on fly each time an experiment is started, e.g. copying new configuration file to each instance etc. It will allow faster configuration of the cloud from command line.

```
$ java -jar CloudController.jar --configure
```

This command will configure servers defined in the **servers.txt**. If the configuration is done, create a new terminal window and connect with all the WORKER instances and start BenchMark.jar (see from above, how it is executed or just run **java -jar BenchMark.jar** to see the help). Make sure that experiment length values are the same for BenchMark and CloudController (look **configuration.ini**).

Creating autoscale group for Amazon to use Auto Scale for dynamically allocating the servers

This is example how to set up autoscale group and dynamically allocate the servers.

```
# INSTALLING AUTO SCALE TOOLS
$ wget http://ec2-downloads.s3.amazonaws.com/AutoScaling-2011-01-01.zip
$ unzip AutoScaling-2011-01-01.zip
$ cd autoscaling/
$ export AWS_AUTO_SCALING_HOME=`pwd`
$ export PATH=${AWS_AUTO_SCALING_HOME}/bin:$PATH
$ source ../amazon/amazon
$ as-cmd

# HAD TO GENERATE NEW PRIVATE KEY
# LOOK https://forums.aws.amazon.com/thread.jspa?threadID=63965
$ openssl pkcs8 -topk8 -in yourkey.pem -nocrypt > yourkey_newforautoscale.pem
$ export EC2_PRIVATE_KEY=${EC2_KEY_DIR}/yourkey_newforautoscale.pem

# CREATE CONFIGURATION
$ as-create-launch-config yourLC --image-id ami-ffc01996 --instance-type m1.small

$ as-create-auto-scaling-group yourASG --launch-configuration yourLC --availability-zones us-east-1c --min-size 1 --max-size 5

$ as-create-or-update-trigger your-trigger --auto-scaling-group yourASG --namespace "AWS/EC2" --measure CPUUtilization --statistic Average --dimensions "AutoScalingGroupName=yourASG" --period 60 --lower-threshold 50 --upper-threshold 60 "--lower-breach-increment=-1" "--upper-breach-increment=1" --breach-duration 900
```

This will create auto scaling group that will monitor servers in the group to see if one of the thresholds have been exceeded. If average CPU usage goes over 60, it will add new server, if it goes below 50, it will remove server. To see instances in the auto scale group, use following command:

```
$ as-describe-auto-scaling-instances

INSTANCE  i-b0a690d2  yourASG  us-east-1c  InService  HEALTHY  yourLC
```

If you do not want to use it anymore, remove Auto Scale with following commands:

```
$ as-delete-trigger your-trigger --auto-scaling-group yourASG
# you need to change the running instance amount in order to start deleting the group , if you kill the instance, it will be automatically reassigned
$ as-update-auto-scaling-group yourASG --min-size 0 --max-size 0
$ as-delete-auto-scaling-group yourASG
$ as-delete-launch-config yourLC
```