

Auto scaling squid servers using Amazon Web Services

October 29th 2014

Version 0.93

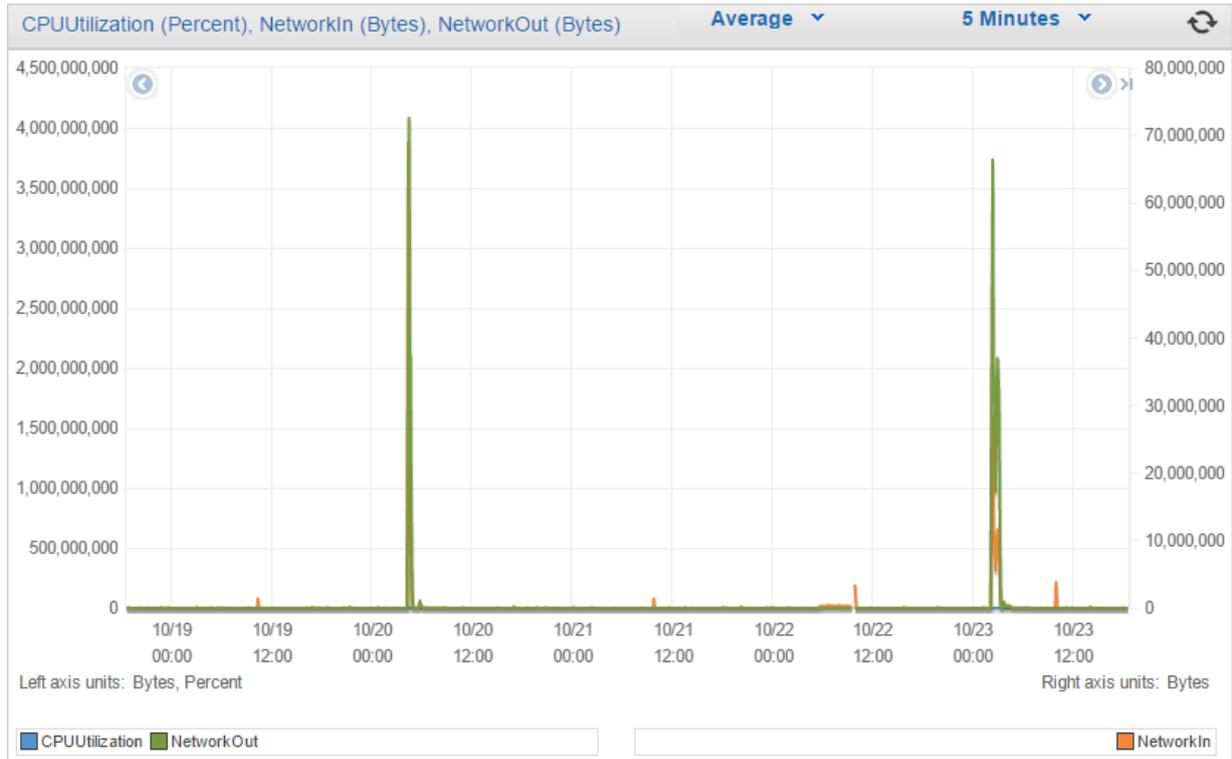
Claudio Pontili

1	Squid server load during jobs execution	2
1.1	Network OUT.....	2
1.1.1	First spike.....	3
1.1.2	Second spike.....	3
1.2	CPU.....	4
1.3	Network IN.....	4
1.4	Conclusions.....	5
2	Auto scalablign squid servers	6
2.1	Prices	7
2.2	Price for a real job execution.....	7
3	Cloudformation	10
3.1	Cloudformation and squid servers	10
4	Testing scalable squids.....	15
4.1	ELB+Single squid server	16
4.2	ELB+Multiple Squid Instances.....	18
4.3	Conclusions.....	19
5	Using cloudfront instead of squid servers	19
5.1	Price problem	19
5.2	problems.....	21
5.3	Conclusions.....	22

1 SQUID SERVER LOAD DURING JOBS EXECUTION

These is the load of a single squid server during the execution of about 1000 jobs inside AWS.

1.1 NETWORK OUT



AVERAGE OF 5 MINUTES - BYTES PER MINUTE

Two spikes:

Value:

4,083,881,062.8 (Bytes) about 520 Mbit/s

Time:

2014/10/19 23:30 CST

Metric:

NetworkOut

and

Value:

3,736,336,809.2 (Bytes) about 475 Mbit/s

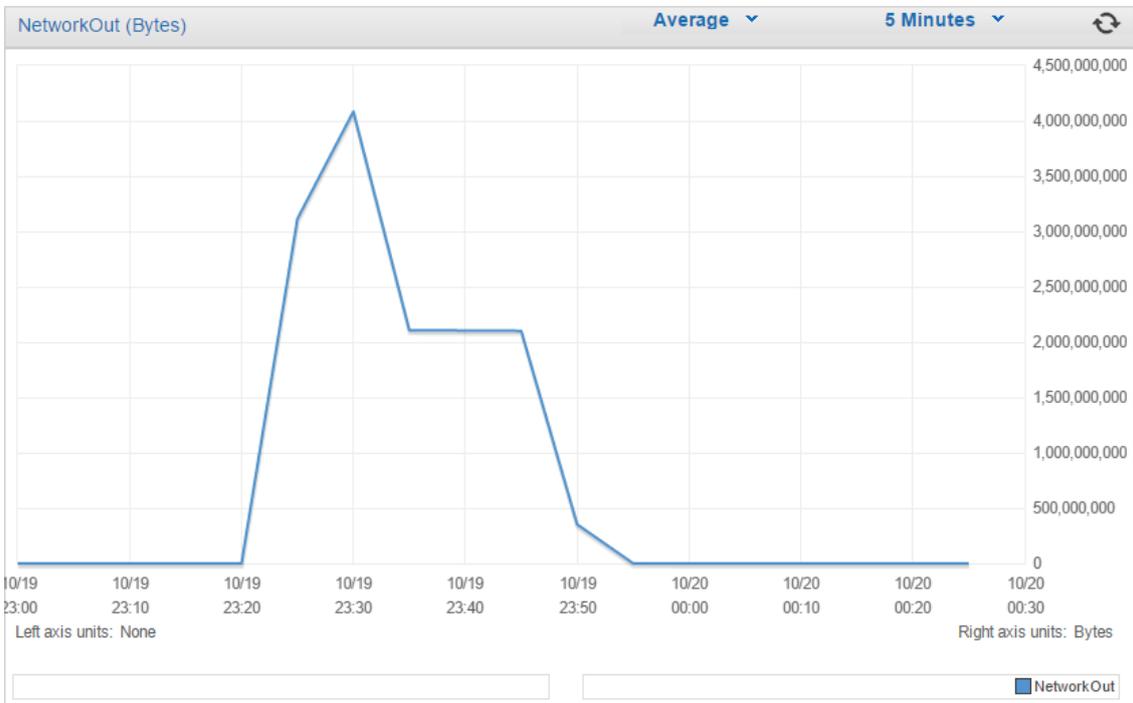
Time:

2014/10/22 21:25 CST

Metric:

NetworkOut

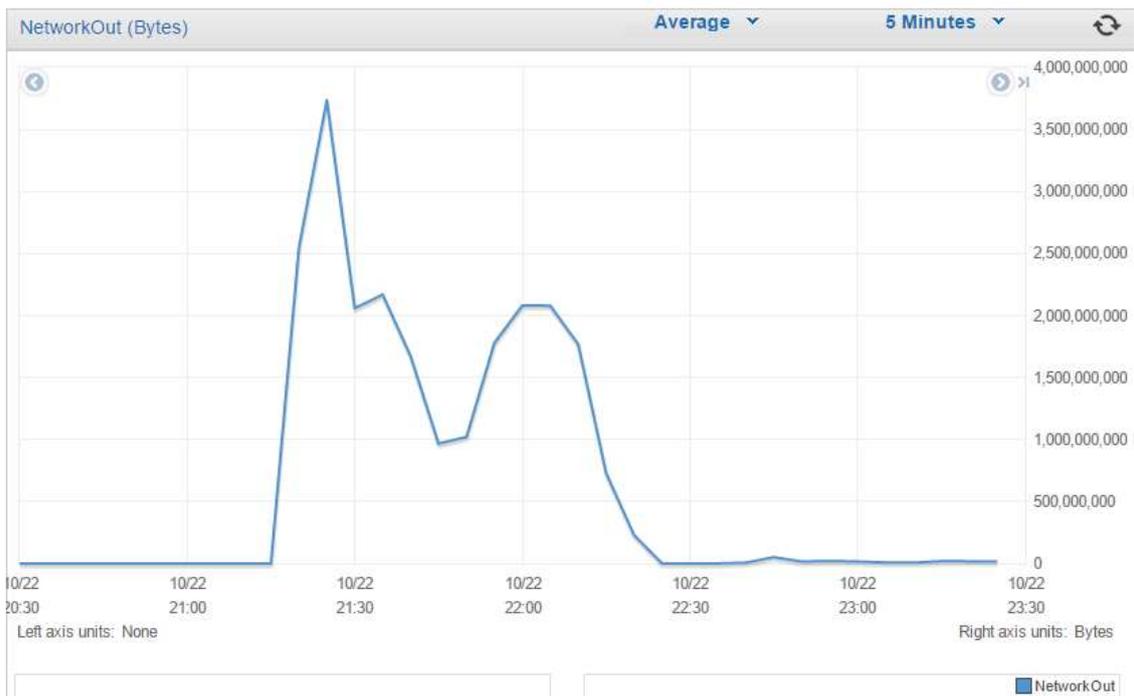
1.1.1 First spike



Started at 23:20 and ended at 23:55 CST Time about 30 minutes.

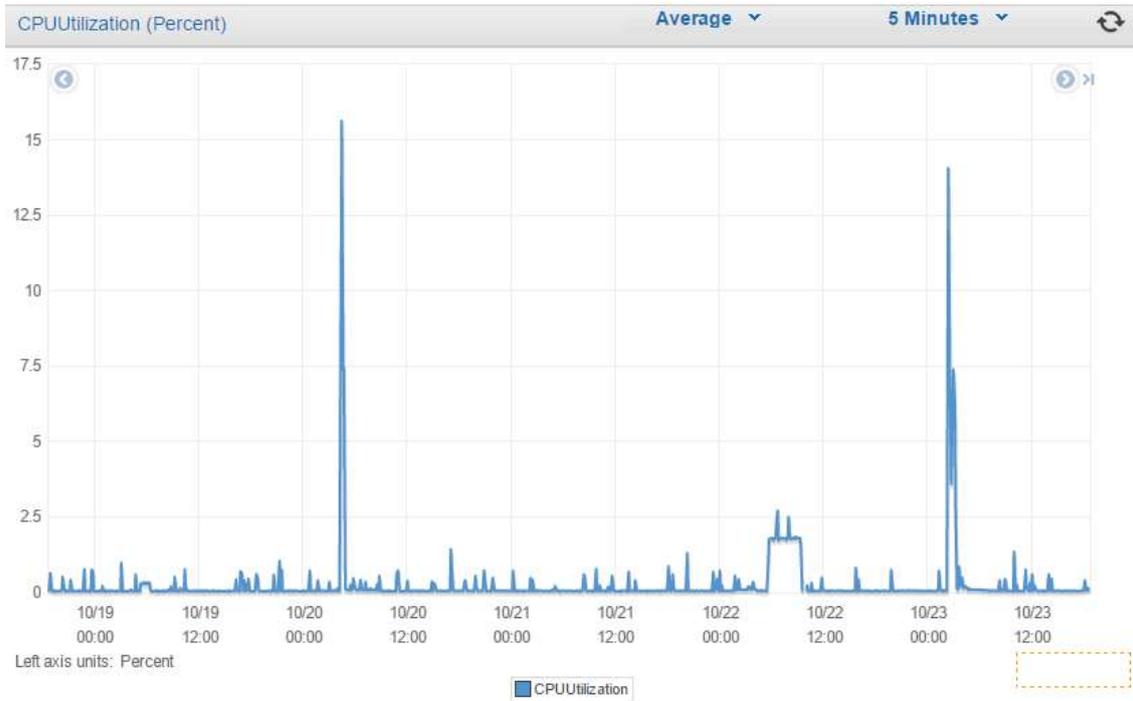
In about 10 minutes the squid server reached the maximum load, we don't know if 520 Mbit/s is the max throughput or there is bottleneck.

1.1.2 Second spike



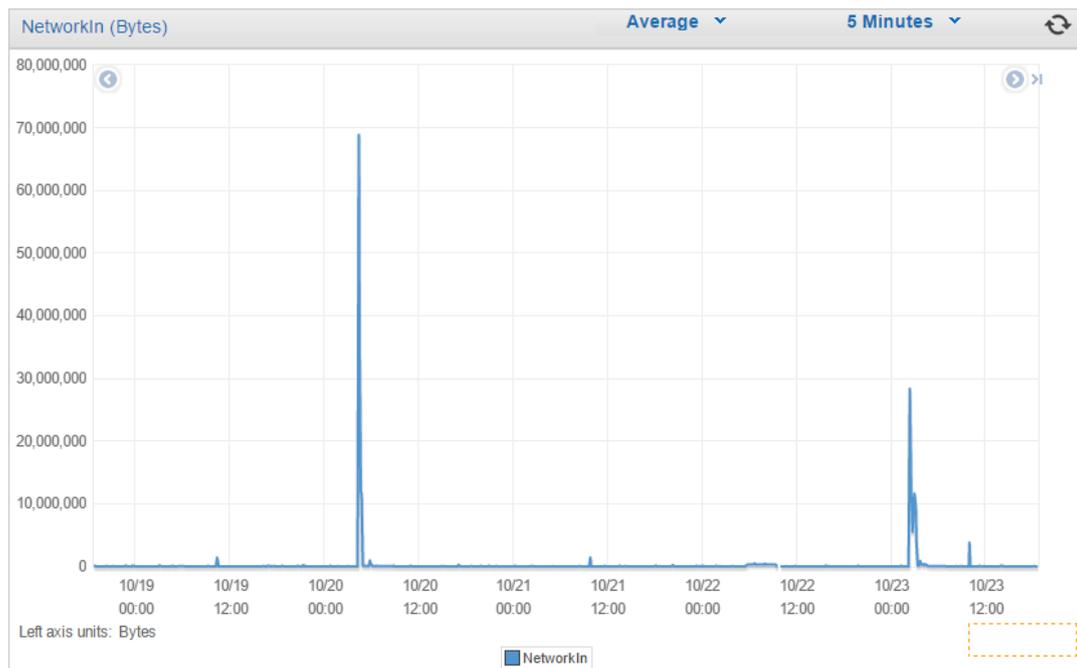
Started at 21:15 and ended at 22:25 CST Time about 1h10 minutes.

1.2 CPU



The CPU (m3.large) is not a bottleneck.

1.3 NETWORK IN



This is the value of the biggest spike in network IN so I think there is no bottleneck too.

Value:

68,871,411 (Bytes) about 9 Mbit/s

Time:

2014/10/20 04:25 UTC

Metric:

NetworkIn

1.4 CONCLUSIONS

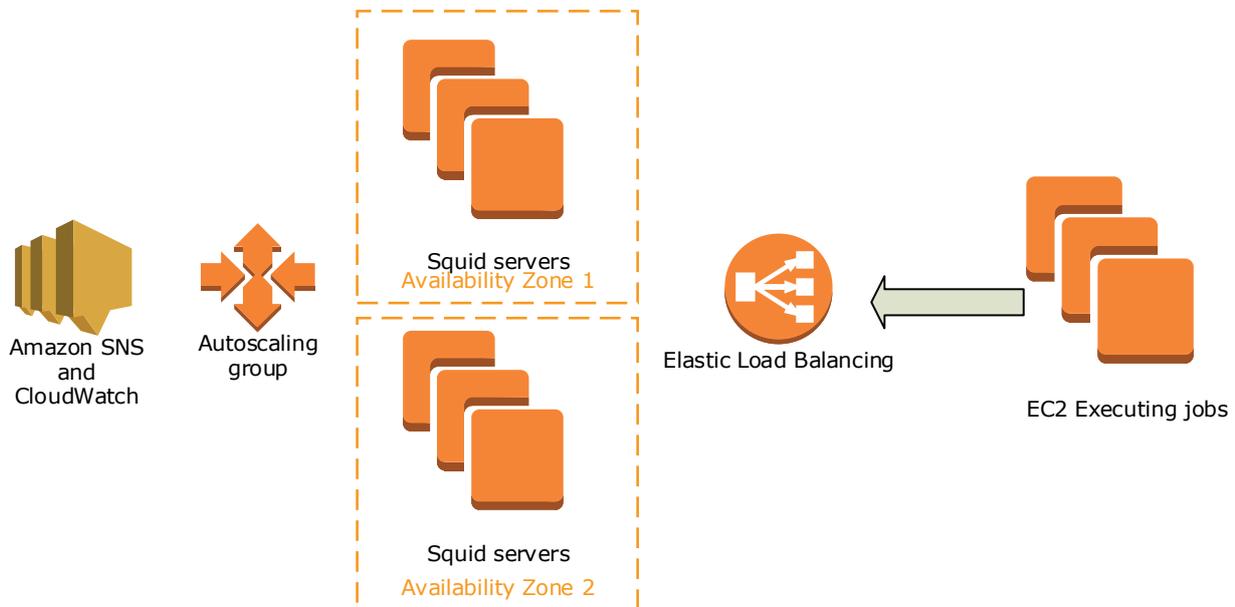
The bottleneck in a m3.large instance is the network OUT. The spikes are high; they reach the maximum in about 10 minutes.

However to scale up our squid server with an ASG (Auto scaling group) we need about 2/3 minutes so we have enough time to handle all the traffic.

Moreover we know in advance that there'll be a spike so we can pre-load the ELB (Elastic Load Balancer) calling AWS and pre-load the ASG adding one or two instance launching a simple script.

2 AUTO SCALABLING SQUID SERVERS

Using ASG (Auto Scaling Group), LC (Launch Control), ELB (Elastic Load Balancer), SNS (Simple notification service) we can create a scalable architecture for our squid servers.



This is a simple picture of what we can configure inside AWS.

The SNS monitors the current instances of squid servers and alert the ASG when the network traffic OUT is too high or too low.

When the traffic is too low the ASG terminates one squid server, we can choose different termination policy.

<http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AutoScalingBehavior.InstanceTermination.html>

When the traffic is too high, the ASG executes the LC and create a new instance. The instance is included in the ELB and the ELB checks if the instance is healthy executing a ping to the port 3128. The ELB keeps checking all the instances, if one of them became un-healthy, it will be terminated and replaced with a new one.

The ELB is configured as “internal”. It can accept traffic only from inside VPC (Virtual private cloud). Moreover the EC2 instances that are executing jobs must use a DNS name like that <http://SquidELBTest-539578840.us-west-2.elb.amazonaws.com:3128>. In fact an ELB is a combination of a Round-Robin DNS and load balancer. It uses RR DNS to scale up infinitely itself but after scaling up and down it behaves like a normal load balancer.

The next step should be to create a DNS CNAME record like “squidAWS.fnal.gov” because every time we create a new ELB the assigned DNS changes.

In this way, we do not need the cache discovery system (shoal) because the address of the squid server never change and we can easily reach a throughput of Gbits.

2.1 PRICES

There are two additional prices for this solution. ASG and LC are free. The price of SNS is very low. We must pay the ELB and the traffic of the EC2 instances:

- ELB billed per hour 0.025\$ and per GB processed 0.008\$
- The squid servers are distributed between different subnet and different AZs (availability zones) so we have to pay for the traffic IN or OUT 0.01\$ per GB

Looking to the definition of AZ:

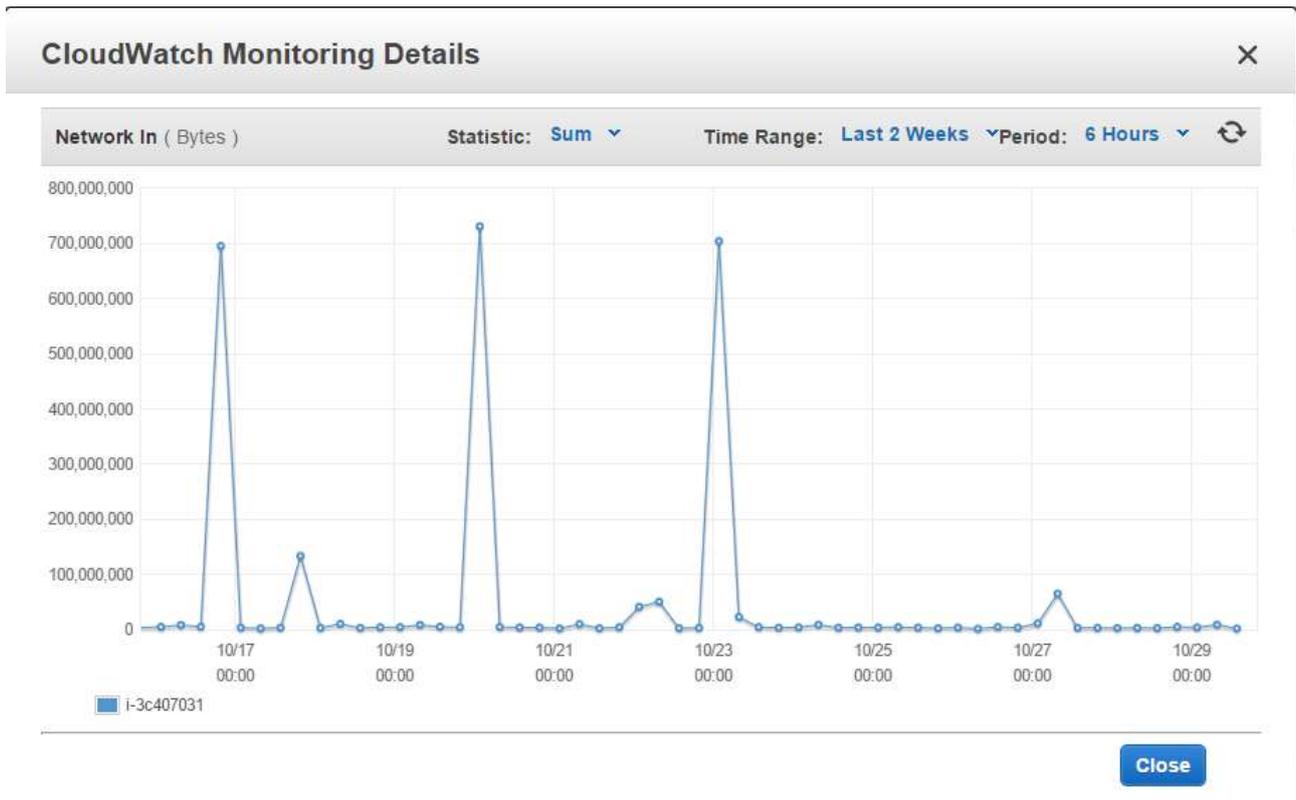
“An availability zone is a distinct location within a region. Each availability zone is isolated from failures in any other zone. By running multiple AMIs across several zones, you can protect your databases and applications from a single point of failure. Each zone has its own power sources and cooling and are designed to be insulated from floods and fires.”

We can understand that using all three AZs inside the Oregon Region is important. And last but not least if we run thousands of instances in different AZs it's less likely that AWS can stop us.

So the second price of 0.01 per GB could be avoided but it is NOT recommended.

2.2 PRICE FOR A REAL JOB EXECUTION





Using this metric extracted from AWS Console we can calculate the price for the scalable squid server solution if we would have used during the execution of 1000 jobs.

The total network out+in for one spike is about 120 GB. We can assume the worst scenario where each EC2 job instance communicate with a squid instance in another AZ, so we need to pay 0.01 per GB out and in.	$120 * 0.01 * 2 = 2.4$ USD
We can assume 2 hours of 2 EC2 m3.medium, but can be less because of scale down policy.	$0.07 * 4 = 0.28$ USD
2 hours of ELB	$0.025 * 2 = 0.050$ USD
Data processed by ELB	$0.008 * 120 = 0.96$ USD
EBS IO + EBS Space	Max 2 USD
	Total price 5.69 USD

We can spend only 6 USD because we have a cloudformation script and it's easy to destroy and create the infrastructure only when we need it.

I think that for less than 6 USD it's better not to use the squid server of the FermiLab and avoid spikes of traffic that can be easily handled inside AWS.

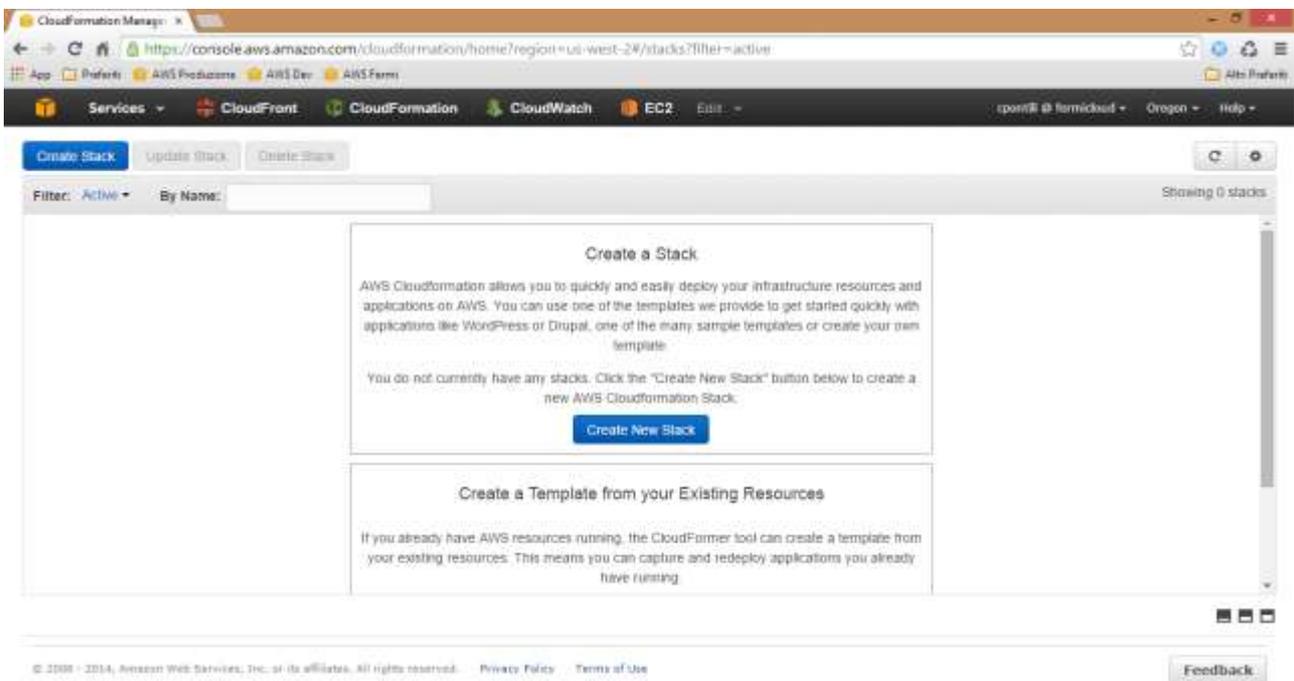
3 CLOUDFORMATION

AWS CloudFormation enables you to create and provision AWS infrastructure deployments predictably and repeatedly. It helps you leverage AWS products such as Amazon EC2, Amazon Elastic Block Store, Amazon SNS, Elastic Load Balancing, and Auto Scaling to build highly reliable, highly scalable, cost-effective applications without worrying about creating and configuring the underlying AWS infrastructure. AWS CloudFormation enables you to use a template file to create and delete a collection of resources together as a single unit (a stack).

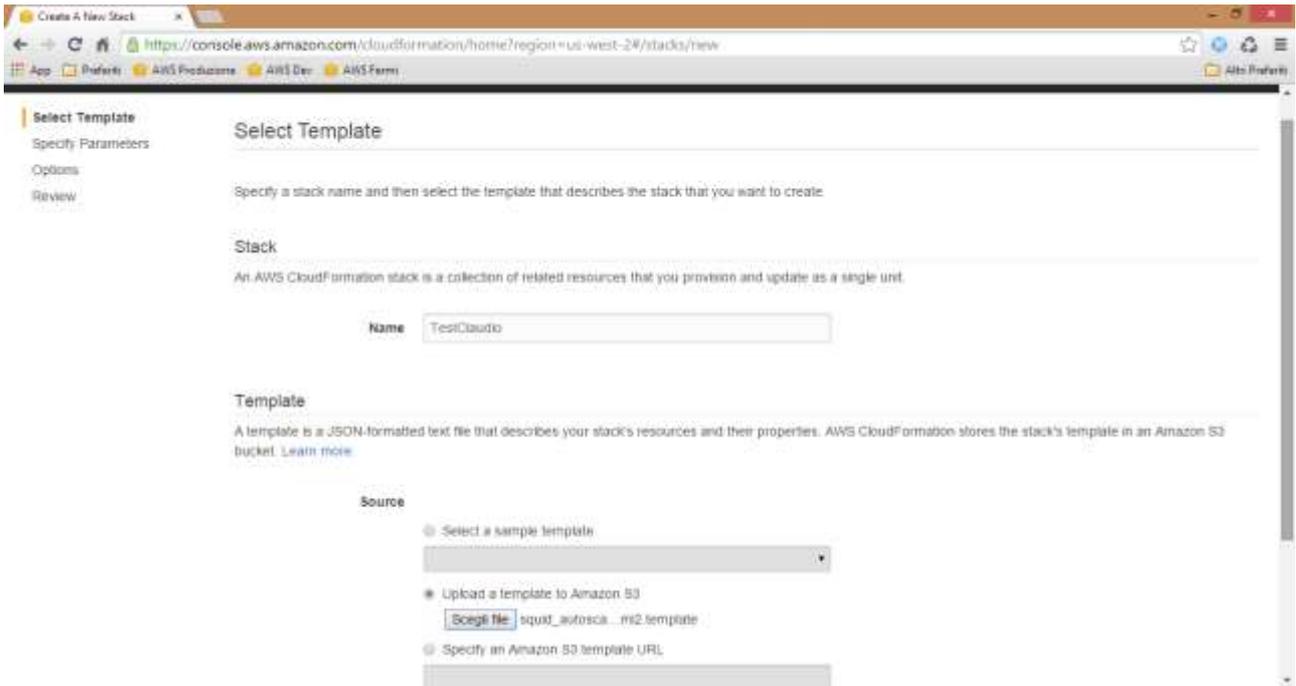
3.1 CLOUDFORMATION AND SQUID SERVERS

The squid servers that we are using inside AWS are used only when there are some job to compute.

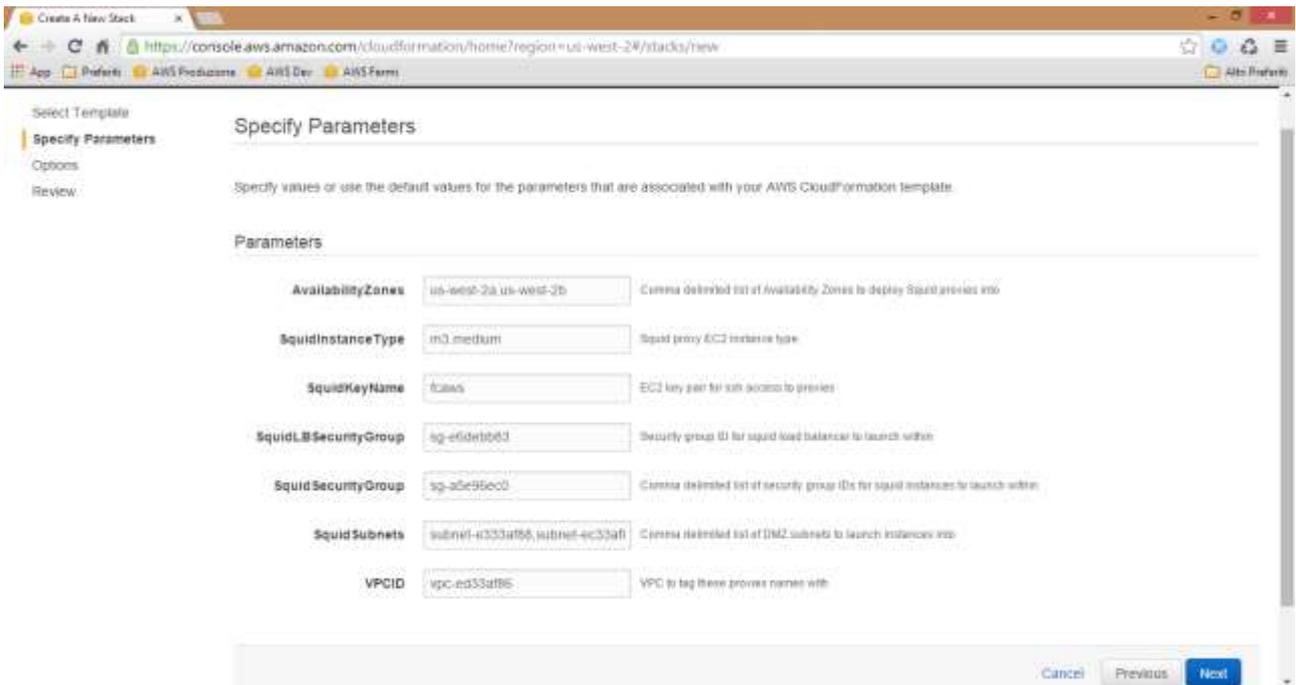
Using ASG+LC+ELB+VPC+SNS means that we need a lot of work to set up all the parameters correctly. So I've created a ClouFormation script in JSON format, to set up the entire infrastructure in minutes and destroy it with just one click.



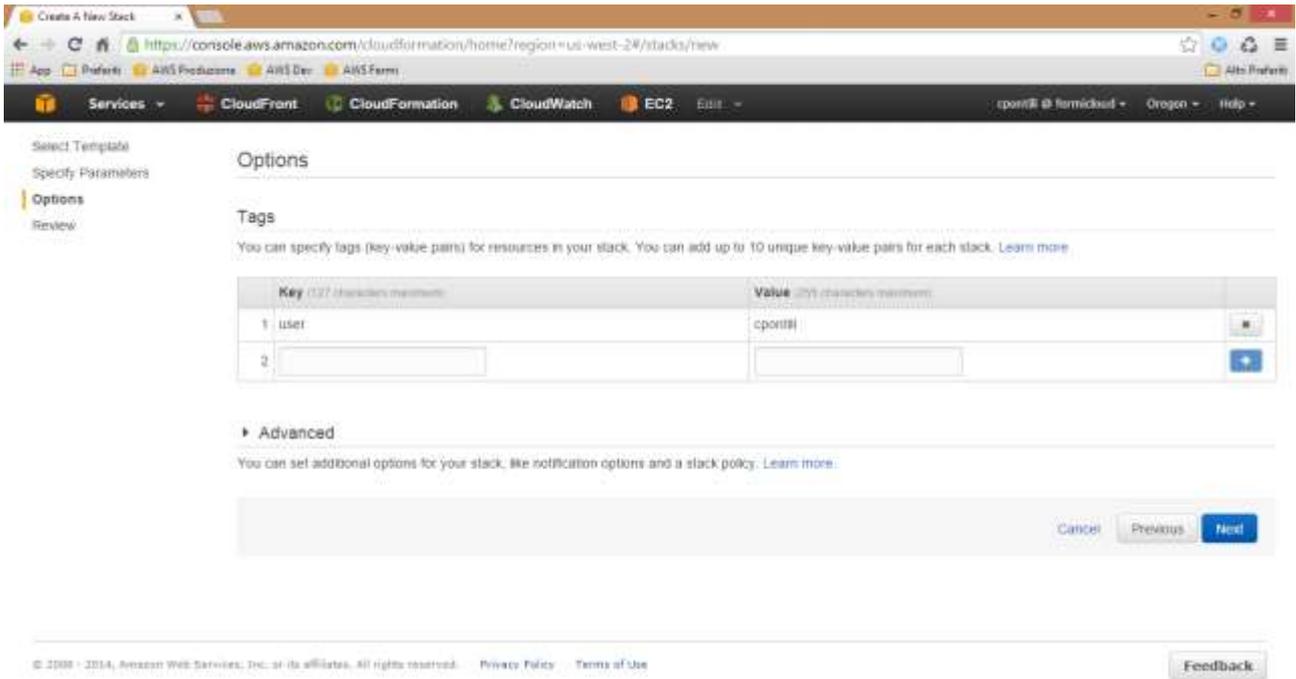
With the web console of AWS we can create a new stack



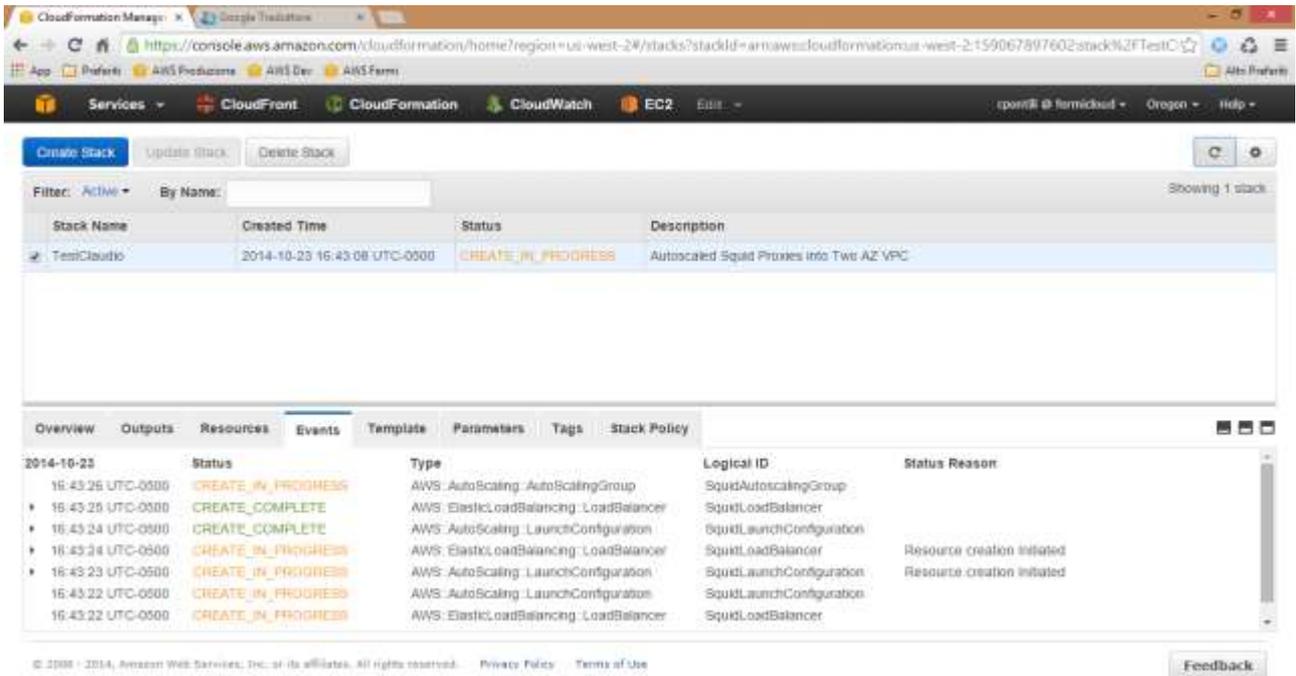
Insert a name and select a template



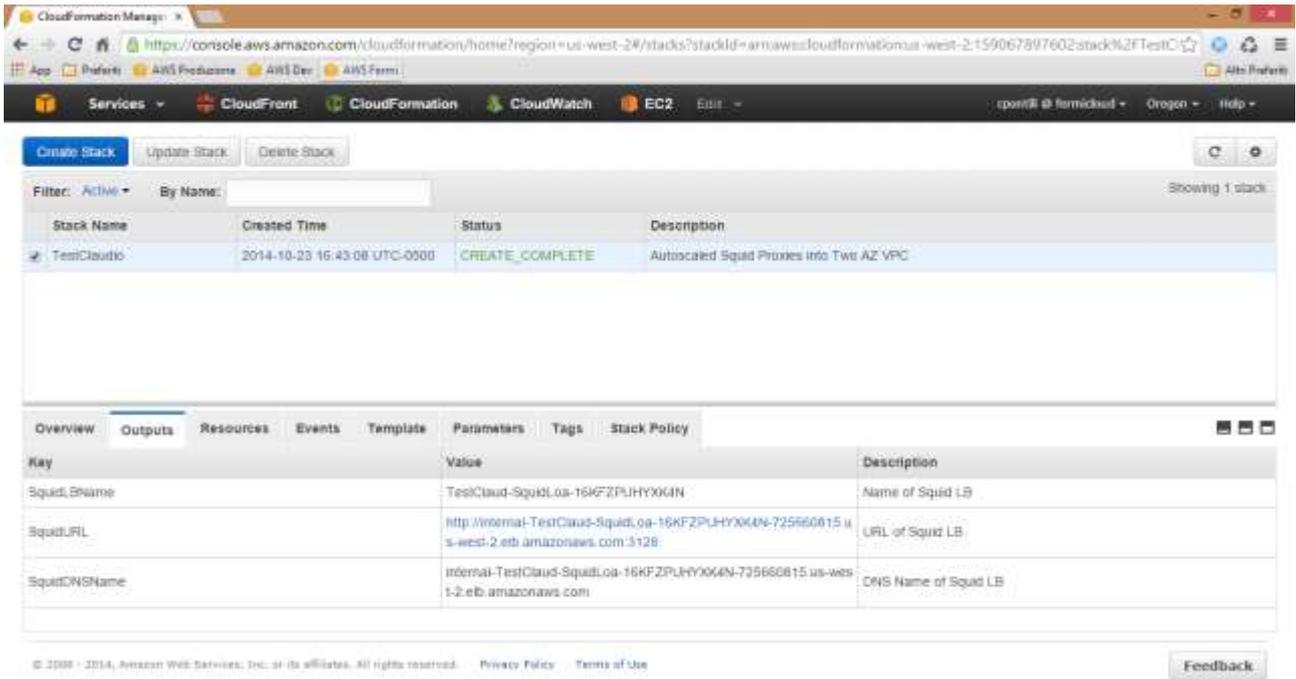
You can change some parameters like the instance type from medium to large, or the security group (firewall rules), etc.



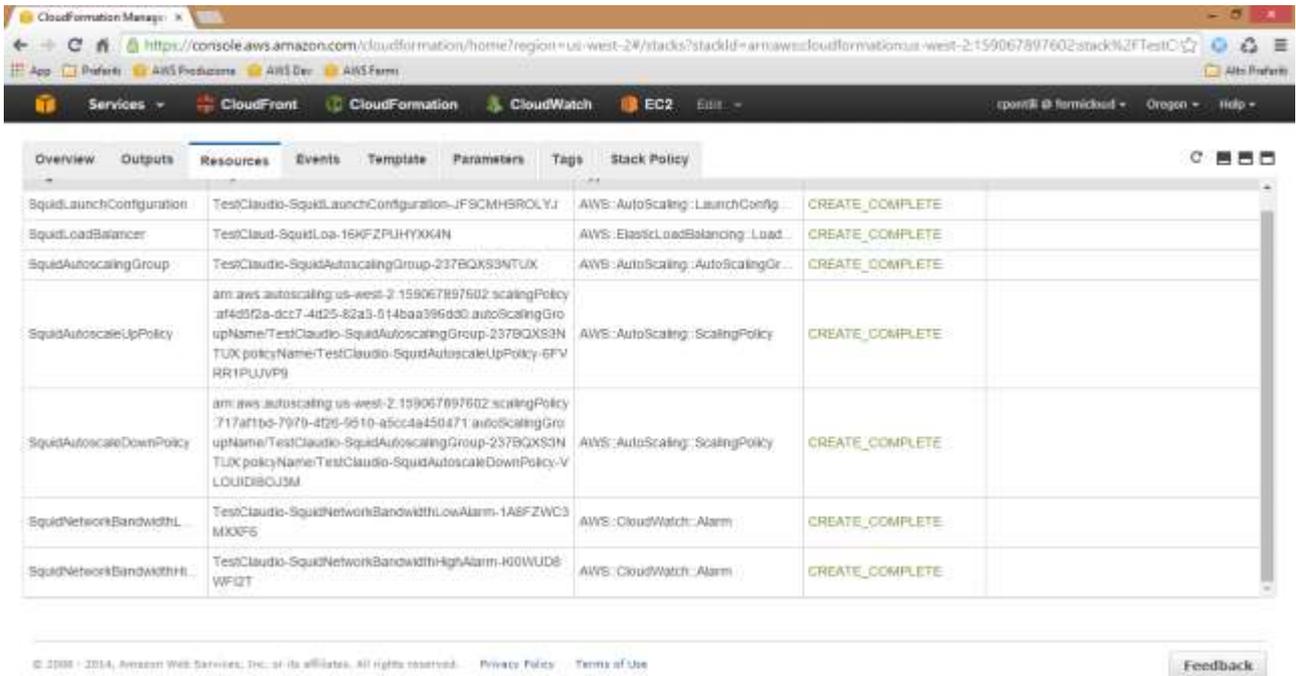
You can add a tag to every resource that we'll be created by the stack. Moreover Cloudformation add other tags to suggest that a resource is created with an automatic process.



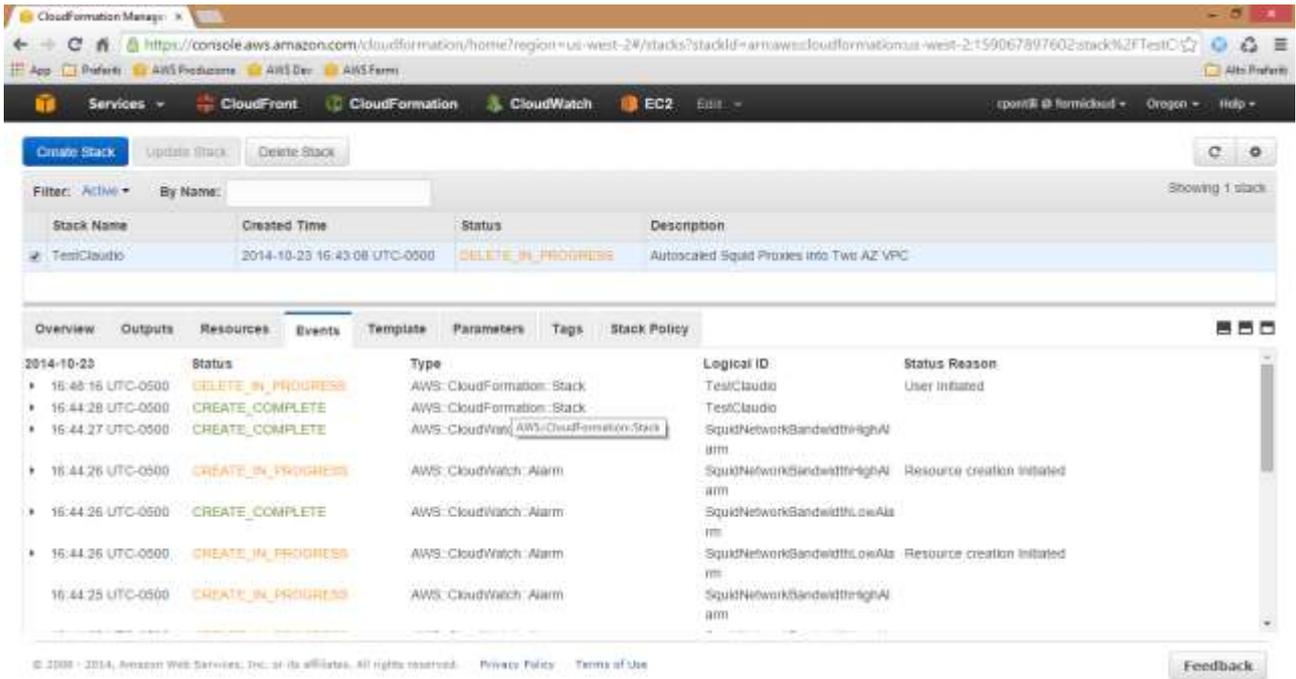
Create in progress



In about 1 minute and 30 seconds we have our infrastructure up and running. In the output tab you can find the address of the squid servers.



In the resources tab you can find all the resource created by the script.



Deleting the stack you'll delete all the resources created.

4 TESTING SCALABLE SQUIDS

Testing our infrastructure means to test the scalability of the ELB and the scalability of the squid servers together.

To do that I used Dave Dykstra's script executing it in some testing EC2 instances.

```
#!/bin/bash
export http_proxy=http://internal-SquidInternetELB-933666428.us-west-2.elb.amazonaws.com:3128
let I=0
RUNNING="running.`uname -n`"
trap "rm -f $RUNNING" 0
touch $RUNNING
let P=0
WGETSPERITER=15
while [ $P -lt 10 ]; do
  let P=$P+1
  while [ -f $RUNNING ]; do
    let I=$I+1
    echo "Iteration $P:$I"
    time bash -c 'n=0; while [ $n -lt "$WGETSPERITER" ]; do wget -qO/dev/null --header="X-Frontier-Id: dwdtest"
"http://cmsfrontier4.cern.ch:8000/dwdfiles/Frontier/type=frontier_file:1:DEFAULT&encoding=BLOB&p1=medfile.tgz"; let n=$n+1; done'
  done &
done
#rm -f $RUNNING
Wait
```

All the instances are m3.medium with:

- 1 vCPU High Frequency Intel Xeon E5-2670 v2 (Ivy Bridge).
- 3.75 GB of memory
- 1 root EBS magnetic storage 10 GB (Max throughput/volume 40 MBps)
- 1 additional EBS SSD storage 10 GB (Max throughput/volume 128 MBps)



Number of requests registered from the ELB



CPU Utilization of Squid Server

So the bottleneck is the CPU because the m3.medium instance has only 1 vCPU Intel Xeon E5-2670.

4.2 ELB+MULTIPLE SQUID INSTANCES

To test the scalability of the infrastructure I have followed these steps:

1. Launch the CloudFormation script
2. Launch two m3.medium instance (test client) and execute Dave's script
3. Wait 10-20 minutes and check the scale up
4. Launch another two m3.medium instance (test client) and execute Dave's script
5. Wait 20-30 minutes and check the maximum throughput
6. Terminate three m3.medium (test client) instance
7. Wait 10-20 minutes and check the current bandwidth
8. Terminate the last m3.medium (test client)
9. Wait 10 minutes and check the scale down process



The Network Out statistics show that everything works as expected.

Status	Description	Start Time	End Time
Successful	Terminating EC2 instance: i-ade628a7	2014 October 24 14:53:27 UTC-5	2014 October 24 14:54:12 UTC-5
Successful	Launching a new EC2 instance: i-dc50bad0	2014 October 24 13:47:25 UTC-5	2014 October 24 13:47:57 UTC-5
Successful	Launching a new EC2 instance: i-ade628a7	2014 October 24 13:33:24 UTC-5	2014 October 24 13:33:57 UTC-5

The Scaling History of the Auto Scaling Group show a Scaling up (Launching a new EC2 instance –idc50bad0) and a Scaling Down (Terminating EC2 instance i-ade628a7).

The maximum throughput reached with two m3.medium squid server is about 2,200,000,000*2 Bytes/min (560 Mbit/s).

4.3 CONCLUSIONS

Using a CloudFormation with Autoscaling is easy to deploy in a couple of minutes a group of squid servers that can easily reach a throughput of Gbits per seconds.

The scaling up process is pretty fast so we should have problem with traffic spikes.

Of course we should test the infrastructure with a more real traffic load before to use it in a production environment. We should tune the scaling up policy, the scaling down policy, choose the right instance size (m3.medium or m3.large?), and check if we can have a CPU or memory bottleneck.

5 USING CLOUDFRONT INSTEAD OF SQUID SERVERS

Amazon CloudFront is a content delivery web service. It integrates with other Amazon Web Services products to give developers and businesses an easy way to distribute content to end users with low latency, high data transfer speeds, and no minimum usage commitments.

In other words is a proxy server to speed up HTTP traffic distributing contents in Edge Location. Each AWS Region is an Edge Location and it scale up and down automatically.

Therefore, we could use it instead of Squid servers.

5.1 PRICE PROBLEM

Normally CloudFront is used to speed up web sites hosted inside AWS. The contents are cached in the Edge Locations in the entire world so the clients can have higher speed and the server can have lower load.

Of course, we want to use it in a different way, using just the Edge Location in Oregon Region caching CVMFS contents.

The prices are

- \$0.120 per GB for data from CloudFront to Internet
- \$0.020 per GB for data from Origin (CVMFS) to CloudFront
- \$0.0075 per 1000 HTTP requests

I got in contact with AWS Business support to ask about prices in our solution. I hoped that the traffic from Cloudfront to our EC2 Job Instance could be billed in a different way. The answer was no, it's billed as normal traffic from CloudFront to Internet.



Network out of squid server serving 1000 EC2 instances. Sum of 6 hours of traffic.

The total traffic of two spikes is about 200 GBytes.



Networkin of squid server serving 1000 EC2 instances. Sum of 6 hours of traffic.

The total traffic of two spikes is about 1.4 GB

The total prices should be:

- $200 * 0.12 = 24$ USD
- $0.02 * 1.4 = 0.028$ USD
- Assuming an average file size of 50 MB the HTTP requests are $((201.4 * 1024 * 1024) / 50) / 1000 * 0.0075 = 31$ USD

Total price is about 55 USD.

5.2 PROBLEMS

Amazon does not give information about what kind of server use in the background of CloudFront.

Talking with Dave I knew that you need a specific version of squid server. In fact, other versions of squid have some bugs.

```
claudio@ubuntu: /tmp

claudio@ubuntu: /tmp$ wget -dq d2ngj6ifp9h35h.cloudfront.net
Setting --quiet (quiet) to 1
DEBUG output created by Wget 1.15 on linux-gnu.

URI encoding = 'UTF-8'
Caching d2ngj6ifp9h35h.cloudfront.net => 54.230.32.230 216.137.39.70 216.137.39.99 54.230.32.191 54.230.35.112 54.230.32.172 54.230.35.237 54.230.32.143
Created socket 3.
Releasing 0x00000000000ec93e0 (new refcount 1).

---request begin---
GET / HTTP/1.1
User-Agent: Wget/1.15 (linux-gnu)
Accept: */*
Host: d2ngj6ifp9h35h.cloudfront.net
Connection: Keep-Alive

---request end---

---response begin---
HTTP/1.1 200 OK
Content-Type: text/html; charset=ISO-8859-1
Transfer-Encoding: chunked
Date: Wed, 22 Oct 2014 18:52:50 GMT
Expires: -1
Cache-Control: private, max-age=0
P3P: CP="This is not a P3P policy! See http://www.google.com/support/accounts/bi
n/answer.py?hl=en&answer=151657 for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic,p=0.01
X-Cache: Miss from cloudfront
Via: 1.1 723d8cb9111beaba0f68a4c77a05f60c.cloudfront.net (CloudFront)
X-Amz-Cf-Id: vGxYnVmotasHJ5WJ23hDGkd5nN-InLLQ8pTPhZvIRRp0CkYH7lGJoA==
Connection: Keep-Alive
```

Setting up a CloudFront Distribution to <http://cmsfrontier4.cern.ch:8000> and executing a WGET we can't find useful information in the VIA header about the server used.

Moreover, I was not able to execute Dave's Script correctly.

5.3 CONCLUSIONS

CloudFront could be perfect because it has a pay-per-GB pricing policy and it scales automatically.

However, the price is not so low it corresponds about 400 hours of m3.large instance and above all we could have problems using a different version of squid server.

We can go on testing CloudFront but it doesn't seem a good solution.