Cloud Computing
Builing a scalable web application

Jérôme Navez - Xavier Pérignon

December 2016

Introduction

AWS offers multiples cloud services that allow us to rapidly create and deploy scalable applications. So we used them to build a scalable SaaS, a web application.
In this report, we will describe this application that allows the user to create an event and add pictures to this event. He can also share the link of this event to his friends so that this can also add pictures and download the archive(s) of the event.
First of all, we will describe the infrastructure set up and then, we will talk about the evolutions and limitations of our web app.

1 Installation

You can find the application here:
https://github.com/gg0512/PictureEvent—Cloud-Project-2
To run the application, you must have *awscli* and *awsebcli* installed. *awscli* must be configured.
Simply execute
./install.sh
in the root of the project.
Normally, the installation takes approximately 20 to 30 minutes. Sometimes, the CloudFront take more time to be fully deployed.
When the application is fully deployed and that the CDN are ready, the application automatically launch to the default browser.

To unistall the application, execute
./aws_clean.sh
in the root of the project.

2 Infrastructure and architecture

To build this application, we used these components:
- An Elastic Beanstalk and everything that comes with (Load Balancer, EC2,...)
- Three S3 buckets
- A standard SQS queue
- CloudFront
- Tree types of Lambda functions

2.1 Description of the components

See the figure 1 in the Appendix representing all these components.

2.1.1 Elastic Beanstalk

We use an *Elastic Beanstalk* to host a Django server. It allows having a scalable server for the different HTTP requests and the background computations. The background computations are the suppression of the bad images from the zip uploaded (if this is a zip) and the upload of each image to the S3.
So there is a verification of the files to be sure that only image files will be processed.
Because Elastic Beanstalk is stateless, we must be sure that the files are saved in the S3 before sending the HTTP response to the user. The files on the instances can be lost if the instance is shutting down, so when the user receive his HTTP response, we are sure that the files are no longer in the instance but on the S3.

2.1.2 S3 buckets

We choose to use 3 different buckets for 3 different uses:

- The *S3static* to store the CSS of the website. This bucket is set up like a static website hosting. So the resources can be directly accessed with a URL without using *Elastic Beanstalk*.
- The *S3images* to store the images uploaded by the users via the Django app. This can only be accessed by *Elastic Beanstalk* and the *Lambda functions*. This bucket is split into two folders: One for the *rawimages* (images not resized) and another for the *resized* images. The nomenclature for these two folders will be *S3images/raw* and *S3images/resized*.
- The *S3archives* to store the archives of the events. Only *Elastic Beanstalk* and the *Lambda functions* can put objects inside but everyone can get the archives. This bucket is set up like a static website hosting to let the users getting their archives without using *Elastic Beanstalk*.

2.1.3 Standard SQS queue

Each time that an image has been sent to *S3images/resized*, a message containing the key of the image is added to the *SQS queue*. This queue is the link between *S3images/resized* and the lambda function. We used an SQS queue to resolve concurrency problems. When using *S3images/resized* as a trigger for the *Lambda function* (More information about the Lambda in the next section), every time that an image is added and resized, a Lambda function is launched. So that if two images are uploaded at the same time for the same event, two lambdas will be launch and will pull, modify and put the archive of the event. If these steps are processed in this order:

- Lambda1 pull archive
- Lambda2 pull archive
- Lambda1 modify archive
- Lambda2 modify archive
- Lambda1 put archive
- Lambda2 put archive

The modification of the Lambda1 will be lost and an image will not be in the final archive.
To solve this, we use an *SQS* that store the events of a put inside *S3images/resized* and the *Lambda functions* will listen into this *SQS*. The next section will this explain more precisely.

2.1.4 Lambda functions

There is three type of *Lambda function* in our app: the *resizer*, the *zippers* and the *manager*.

Resizer There is one resizer.
The function of the resizer is to resize the raw images. This function is triggered at each Object created in *S3images/raw*. The resized images are then stored in *S3images/resized*.

Zippers There are multiples zippers.
A zipper is triggered each minute by a *CloudWatch Event Scheduler*. It pulls some messages from the *SQS queue* and add the associated images to the archive of the event. To prevent a timeout of the function, we have to limit the pull with some parameters. The information about these parameters are given in the *Evaluation and limitations* section.

The Lambda process these images event by event by downloading the current archive, adding the images from S3 to the archive and uploading the archive to S3.
If the archive is too heavy, it creates another archive that will be modified by the next images and the previous archive will not be modified anymore. The archive is considered too heavy if there is a risk that the $'/tmp'$ directory of the lambda function is full by adding a file to this archive. This directory is limited to $512Mo$. This trick is repeated each time that the current archive is too heavy. So a big event where a lot of people add their images can have multiples archives.
Because of there can be multiple zippers running at the same time, we check that the archive pulled from the S3 is the same that the archive overwrote when uploading the archive in the S3 to avoid concurrency problems.

Manager There is one manager.
Because of there is a limitation of the number of messages pulled at the same time by a *zipper*, we have had some zippers if needed.
The manager is triggered every minute. It adjusts the number of zippers running.
It adds a zipper function if the number of messages remaining in the SQS is too big and removes a zipper if it is no longer useful because the number of messages in the SQS is lower.
But there will always be at least one zipper running.

Why using multiple zippers ? A lambda function is limited with 512 MB of storage in the /tmp folder. Because of this limitation, the manager adds more zipper instead of more Cloudwatch trigger to the same zipper. If we add more Cloudwatch trigger to the same zipper, all the concurrent executions of the zipper will share the same /tmp folder and there will be not enough place if the total storage size needed by all the executions is more that 512Mo. So even if the computation power of a Lambda function is elastic, its storage is not and we wanted to avoid any problems with that.
Another reason is because pulling messages from a queue take a lot of time, so each zipper can only pull a maximum of 10 messages each minute to avoid a timeout by pulling all the messages. Using multiple zippers avoid being limited by only 10 images processed each minute.

2.1.5 CloudFront

To make the static resources and the archives easily reachable across the world we use $CloudFront$ as a CDN. The little inconvenient is that two users from two different places in the world will not have the same archives at the same time. It takes some time to distribute the updated files across the world. But at the end, the files will be the same.

To have a better view of the interaction between these components, see the sequence diagrams in the Appendix.
There is also some screenshots of the application.

3 Evaluation and limitations

3.1 Stress tests

3.1.1 Test 1

Uploading a zip with 1000 images of the dimension 1x1.
This results with a timeout in the browser of the client. The timeout is due to the big amount of images to be uploaded one by one to $S3images/raw$. Even if the client gets a timeout, the instance of *Elastic Beanstalk* continue to upload the files. This is a situation that we want to avoid because the user thinks that his images will not be processed. We added a limitation of maximum 200 images in a single zip for the user.

3.1.2 Test 2

We used a typical zip as an example of upload. This zip contains 47 images that must be resized and has a size of 99 MB. Each image has the dimensions 3264 x 2448 with an average size of 2,1 MB.
We made a script using curl to simulate a client sending a file to the application. At each upload we print the response time of the request. This script will run inside of t2.micro instances that we call *bots*.

As an initial situation, there is only one instance running in the *Elastic Beanstalk*.
To simulate the peak of traffic, we added 5 independent bots sending a zip each minute during 20 minutes, so there are 5 zips added each minute.

Nomenclature: t0 is the initial time, t42 is the initial time + 42 minutes

At t0, the first 5 zips are sent to *Elastic Beanstalk*. The average response time after the upload is 50 seconds. A first extra zipper is created, from this time, an extra zipper will be added each minute. The resizer react well and there is already 235 messages in the SQS.
At t1, the average response time is a little bit longer, 98 seconds.

t2: one more EC2 instance is being launched for the *Elastic Beanstalk*, at t6, this instance is completely running. So there is two instance running in the *Elastic Beanstalk*. The respond time change: for some *bots*, an average of 5 seconds for 3 bots and 60 seconds for the two others. This difference will continue but progressively mitigate. We will observe this type of difference each time that a EC2 instance is added in the *Elastic Beanstalk*.

t9: there is 1071 messages in the SQS and 10 zippers. Another instance is being launched in the *Elastic Beanstalk*.

t13: there is 3 instances running and 1198 messages in the queue.

t15: the number max of remaining messages in the queue is reached: 1225. This number begin to decrease but the manager still create one zipper each minute.

t17: a 4th instance is launched inside *Elastic Beanstalk*.

t20: the bots stop sending zips, they have done their job.

t21: there is 4 instances running in the *Elastic Beanstalk*.

t31: the total number of zipper created is 30. No more zipper is added.

t32: the SQS is empty, the manager start removing one zipper each minute.

End of the peak: the second instance shut down at t39, the third at t47 and the fourth at t52. At t62, there is no more extra zipper, we return to a state without activity.

Conclusion: 4 instances where running a the same time and 30 extra zippers were created. The maximum of memory took by the zippers is 408MB and 191 MB for the resizer. The manager always take 32MB. These numbers will be usefull for the limitations.

See the Appendices section to have more information about the instances of *Elastic Beanstalk* during the test.

3.2 Test 3

Uploading the same zip of the test 2 each minute with the network of the Intel room.
Using an EC2 instance to send a file to *Elastic Beanstalk* doesn't really represent an upload in normal condition. The bandwidth between the Intel room and AWS is lower that between two AWS components.
If the bandwidth is lower, it means that the time to upload will grow up and the instance of *Elastic Beanstalk* will be busy during more time.
This test result to a response time that continue growing each minute. The initial instance is overloaded and 3 other extra instances are deployed.

3.3 Scaling

There is there nodes of computation which have to deal with the traffic load in the application: The *Elastic Beanstalk*, the *resizer* and the *zippers*.

Elastic Beanstalk The Elastic Beanstalk is automatically scaled with the Elastic Balancer. This is a horizontal scaling because the Elastic Balancer add more instances of the same type (t2.micro). The speed of the scaling is not very fast because of the time needed to launch an extra server.

Resizer The lambda functions are also automatically scaled by AWS. If the load is too important, there will be more resources available for the function. This is a vertical scaling.

Zippers The number of *zippers* increase if the *manager* detect that there are too many messages to process. This is an automatic scaling and a horizontal scaling. The scaling is not very fast (see next section).

3.4 Components limitations

This section only treat the technical limitations of the components.

3.4.1 Elastic Beanstalk

The elasticity of *Elastic Beanstalk* is limited by the maximum number of instances running at the same time allowed by the configurations. We kept the default limitation of maximum 4 instances. This is the first limitation of the application. A weakness of the *Elastic Beanstalk* is the fact that it will list the content of the *S3images* before uploading an image to this bucket in order to avoid overwriting a file with the same name. If the *S3images* contain a lot of images, this can take a few time.

3.4.2 SQS queue

The only limit of the queue is 120,000 inflight messages. This means that if we reach this limit, the zippers will not be able to process new messages.
Let's imagine the case where each zipper process 50 messages, the maximum (see next section): so to reach this limit, there must be minimum 2400 zippers running at the same time.

The maximum number of Lambda function running at the same time is 100. So this limit of 120,000 inflight messages will never be reached and can be ignored.

3.4.3 Lambda functions

Zippers A *zipper* cannot pull more than 10 messages. This limitation is approximate, so if the queue has 20 messages, it's possible that the zipper pulls only 8 messages. 10 is just a maximum limit. Pulling messages from a queue is an operation that cost a lot of time. So we prefer that more zippers pull a maximum of 10 messages each than one zipper that pulls all the messages.
According to that, the manager add a new zipper if the number of pending messages in the queue is more than 30 and remove a zipper if the number of messages is less than 10.

After the test 2, we put the memory size of the zipper to 512MB.

Resizer An invocation of *resizer* can fail if there are too many invocations at the same time. The *resizer* use */tmp* folder which has a maximum capacity of 512 MB. If this folder is filled too much, an invocation can fail and an image will not be processed from $S3images/raw$ to $S3images/resized$.
We use ImageMagick to resize the images. This software use a lot of memory, so if multiple images are resizing at the same time, there is a risk that the call to ImageMagick fails. After the stress test 2, we fixed the memory to 256.

Manager The way that the *manager* adds and removes nodes is linear, it will only add the extra *zipper* one by one. So if there is a peak of traffic, the efficient number of $zippers$ can take some time to be reached. The manager is limited to 128 MB of memory but this is not a limitation because it will never go upper.

3.5 Operational Costs

3.5.1 Assumptions

- If applicable, the components are running in the region Oregon.
- The users are from the USA, Canada and Europe.
- In average, there is 50 images uploaded each minute.
- In average, each image has a raw size of 2MB and a resized size of 1MB.
- In average, one new event is created each minutes.
- In average, each event is downloaded 5 times at the end when all the users have added their images.
- The account used to create the deploy the application is no longer on the free tier.

3.5.2 Resulting scale

According to the test 2 and 3, we can suppose that there will be 4 EC2 instance in the Elastic Beanstalk and 6 zippers.

3.5.3 Cost

Elastic Beanstalk The price is \$0.012 per hour with a t2.micro EC2. There is 4 instances, so the price is calculated like that:
$0.012 * 4 * 24 * 30 = \$ 34.56/month$
The Load Balancer cost \$ 0.025/hour:
$0.025 * 24 * 30 = \$ 18/month$
And \$ 0.008/GB of data processed. I will only take the images uploaded in account. The HTML documents are negligible.
$0.002 * 50 * 60 * 24 * 30 * 0.008 = \$ 34,56/month$

S3 storage Each images is replicated 3 times: once in $S3images/raw$, once in $S3images/resized$ (but resized) and once in $S3archives$ (resized in a zip). So each image take 2 + 1 + 1 MB of storage. The total number of images per month is:
$50 * 60 * 24 * 30 = 2,160,000$
Each image take 4MB of storage, each month, the size used per mouth in S3 is:
$4MB * 2,160,000 = 8,640GB$
The cost of one GB is 0.023, so the total cost for the first month is:
$8,640 * 0.023 = \$ 198.72/month$

Resizer The resizer runs during 8 seconds (80 * 100 ms) each minute and have 512MB of memory. The number of invocation per month is:
$50 * 60 * 24 * 30 = 2,160,000$
The cost of the invocations is:
$2.160 * 0.2 = \$ 0.432/month$
The total computation time multiplied by the cost for 512MB is:
$2,160,000 * 80 * 0.000000834 = \$ 144.12/month$

Zippers The zippers runs during 35 seconds (350 * 100 ms) each minute and have 512MB of memory. The number of invocation per month is:
$6 * 60 * 24 * 30 = 259,200$
The cost of the invocations is:
$0.2592 * 0.2 = \$ 0.05184/month$
The total computation time multiplied by the cost for 512MB is:
$259,200 * 350 * 0.000000834 = \$ 75.66/month$

Manager The manager runs during 0.8 seconds (8 * 100 ms) each minute and have 128MB of memory. The number of invocation per month is:
$1 * 60 * 24 * 30 = 43,200$
The cost of the invocations is:
$0.0432 * 0.2 = \$ 0.00864/month$
The total computation time multiplied by the cost for 128MB is:
$43,200 * 8 * 0.000000208 = \$ 0.0000718848/month$

CloudFront The average size of the final archive of an event is 250MB ($1MB * 50 \ images * 5 \ users$). The five users download the archive when the archive is completely generated with all the images, so in average, one archive is downloaded 5 times each 5 minutes (or 1 time each minute).
The total size downloaded from CloudFront per month is:
$0.25 * 60 \ minutes * 24 \ hours * 30 \ days = 10.8 \ TB$
The prize for the first 10TB is 0.085 per GB and it will be 0.08 for the 0.8 TB left:
$0.085 * 10,000 + 0.08 * 800 = \$ 914/month$

SQS Each minute, there are 50 put, 50 pull and 50 delete by the *zippers* and 1 pull by the *manager*. The number of requests per mounth is:
$151 * 60 * 24 * 30 = 6,523,200$
The cost is $ 0.40 per 1 million Requests:
$6.523 * 0.4 = \$ 2.61/mounth$

In total, the price of the first month will be **$ 1,388.16/month**

3.6 Replication needed ?

The only interaction between the user and the application is with *Elastic Beanstalk*. *Elastic Beanstalk* only takes the uploaded files and send some HTML documents. The resources like the CSS or the background images are stored into bucket S3 (*S3static*) and are distributed via a CloudFront CDN. The archives are also distributed via a CDN.
The only replication that can be useful is for the *Elastic Beanstalk* to increase the bandwidth when the user upload a file.
The others components are connected between them in AWS so there is no need to replicate them.
Using the CDN CloudFront is already a replication of the archives and the static files (CSS and background images of the site)

3.7 Failure

Elastic Beanstalk The *Elastic Beanstalk* send the HTTP response to the user only if the file has been stored on the S3, so if the instance crash or is shutting down by the Elastic Balancer before sending on the S3, the user don't receive the response.

S3images If the *S3images* suddenly became unreachable, *Elastic Beanstalk* will not be able to send the files to the bucket and same as before, the user doesn't receive the confirmation.

Resizer As explained earlier, if an invocation of *resizer* fail, there will be an image lost.

SQS There is a limitation of 120,000 inflight messages. If this limit is reached, the zippers will not be able to pull messages from the SQS, but no information will be lost.

Zipper About the zipper Lambda functions: the zippers pull some messages from the queue and modify the archive and only then delete the message. So the messages are deleted only if the processing went well. If the Lambda fails, the messages will not be deleted and will be released by the visibility timeout. Another zipper will be able to pull them later. So no information will be lost, even if a zipper is destroyed during a computation.

S3archives If the *S3archives* became unreachable, the Lambda zippers will not be able to update the archives, so the Lambda functions will fail. The number of SQS messages will just grow up.

During all the process, each time that a file must be added to an archive or a bucket, there is a verification that the file doesn't overwrite another file with the same name. So the name of the files is modified if needed.

Conclusion

AWS allows us to use many services in the cloud to build scalable applications. This is important to keep in mind the cost and limitations of the application deployed.

References

https://aws.amazon.com/lambda/faqs/
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/limits-queues.html
https://docs.aws.amazon.com/lambda/latest/dg/limits.html
https://aws.amazon.com/lambda/pricing/
https://stackoverflow.com/questions/26832358/imagemagick-memory-usage
https://aws.amazon.com/sqs/pricing/

Appendices

Architecture

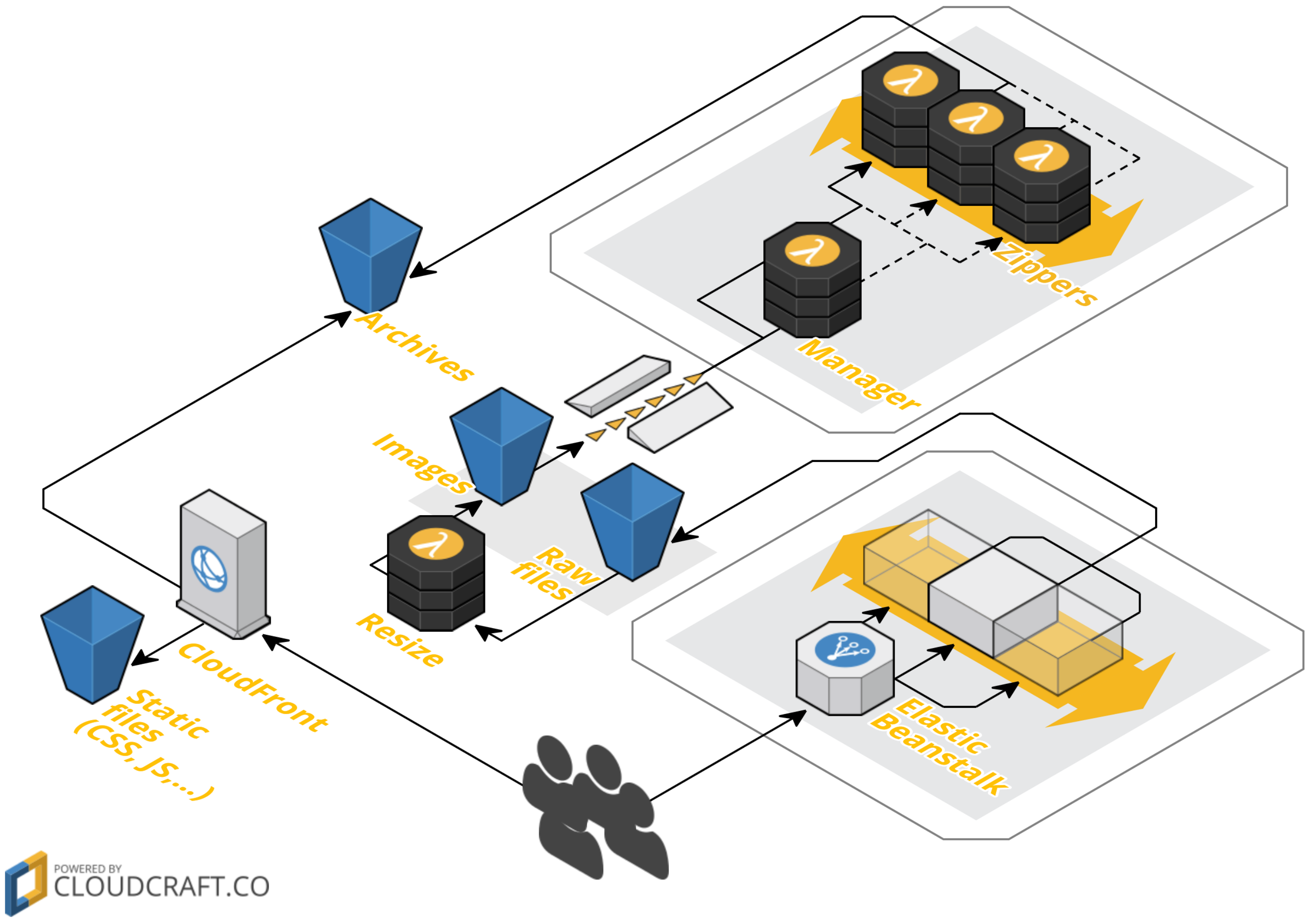


Figure 1: Architecture of the infrastructure

Sequence diagrams

Client upload and download

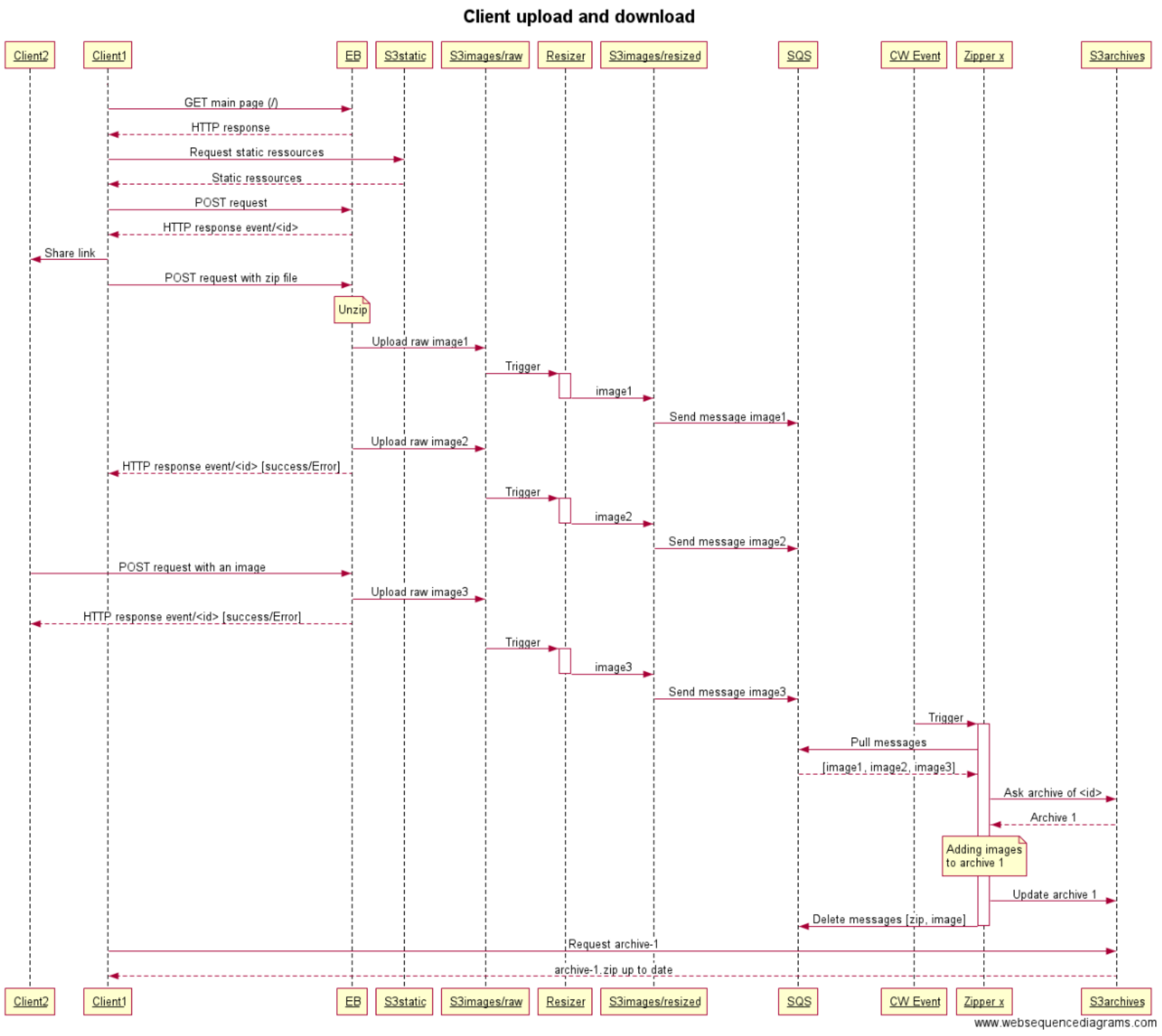


Figure 2: Client upload and download

Client1 share the link of a new event to Client2. Client1 upload a zip and Client2 upload a single image. Then, Client1 download the archive.

Manager

Here, Zipper x represent the existence of a lambda function.

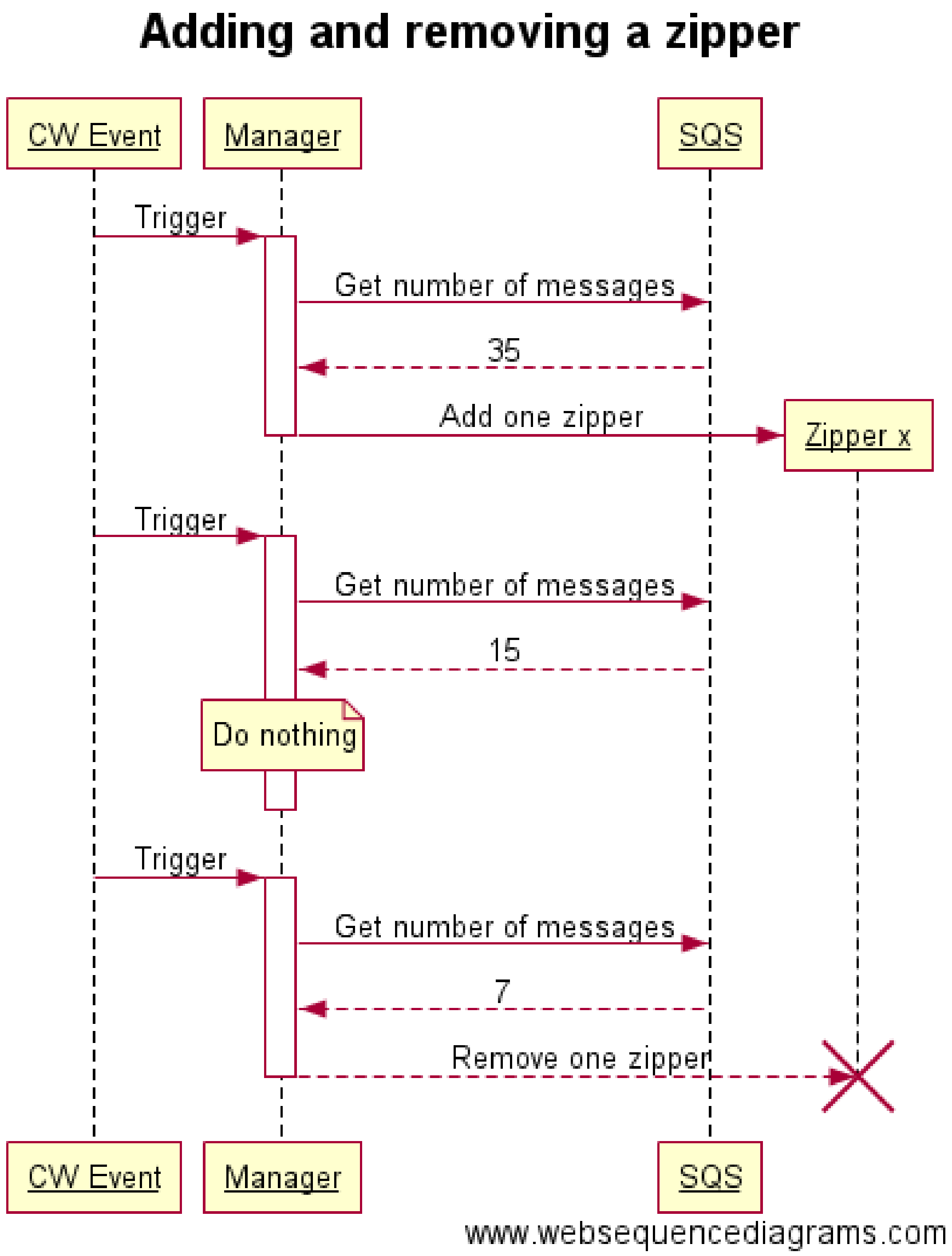


Figure 3: Adding and removing a zipper

Test 2: EC2 instances graphs

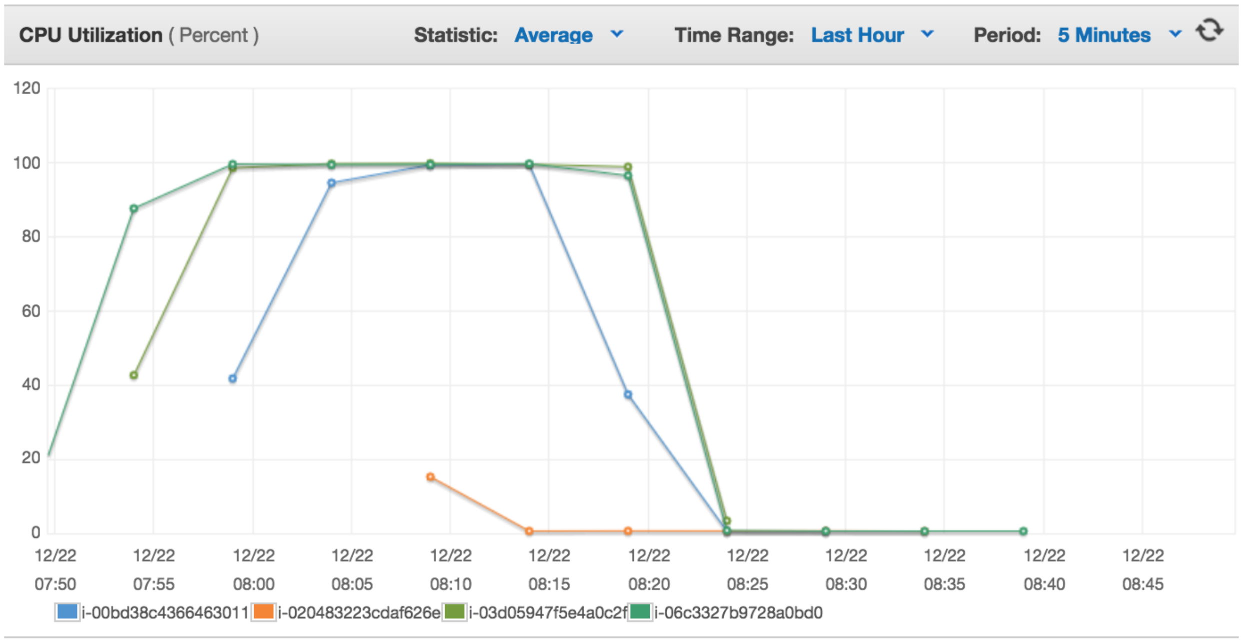
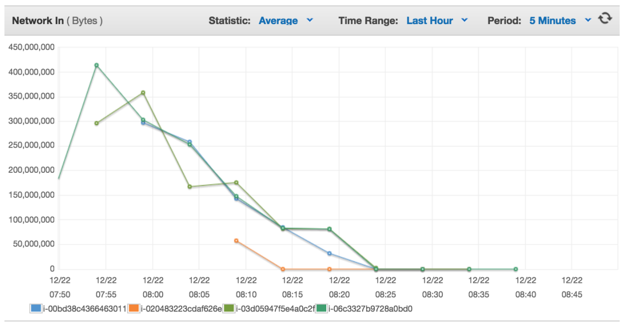
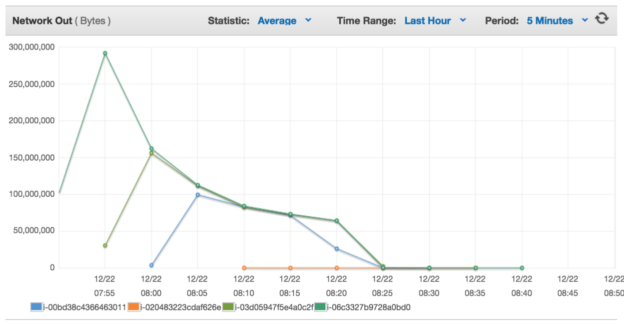

CPU Utilization (Percent) Statistic: Average Time Range: Last Hour Period: 5 Minutes

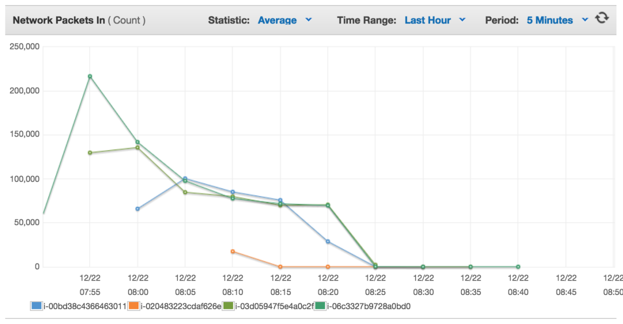
Figure 4: CPU utilization


Network In (Bytes) Statistic: Average Time Range: Last Hour Period: 5 Minutes

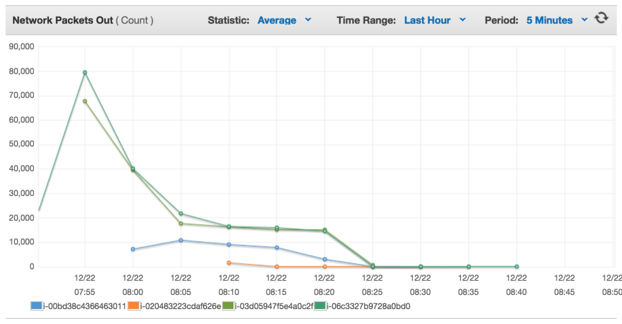
(a) Network in


Network Out (Bytes) Statistic: Average Time Range: Last Hour Period: 5 Minutes

(b) Network out


Network Packets In (Count) Statistic: Average Time Range: Last Hour Period: 5 Minutes

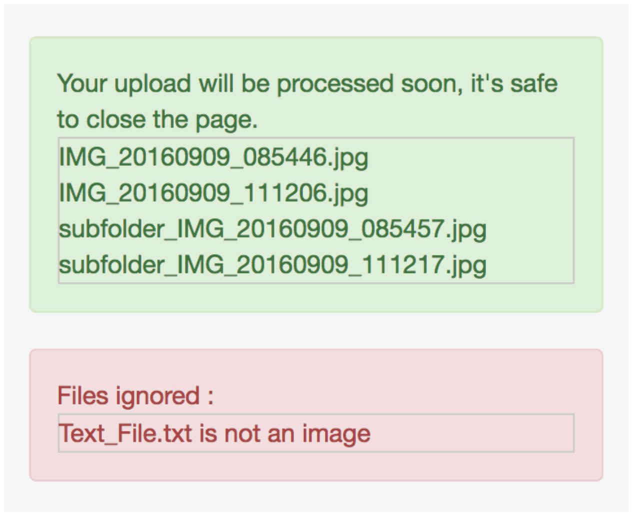
(a) Network packets in


Network Packets Out (Count) Statistic: Average Time Range: Last Hour Period: 5 Minutes

(b) Network packets out

Other screenshots

File verification



Your upload will be processed soon, it's safe to close the page.

IMG_20160909_085446.jpg
IMG_20160909_111206.jpg
subfolder_IMG_20160909_085457.jpg
subfolder_IMG_20160909_111217.jpg

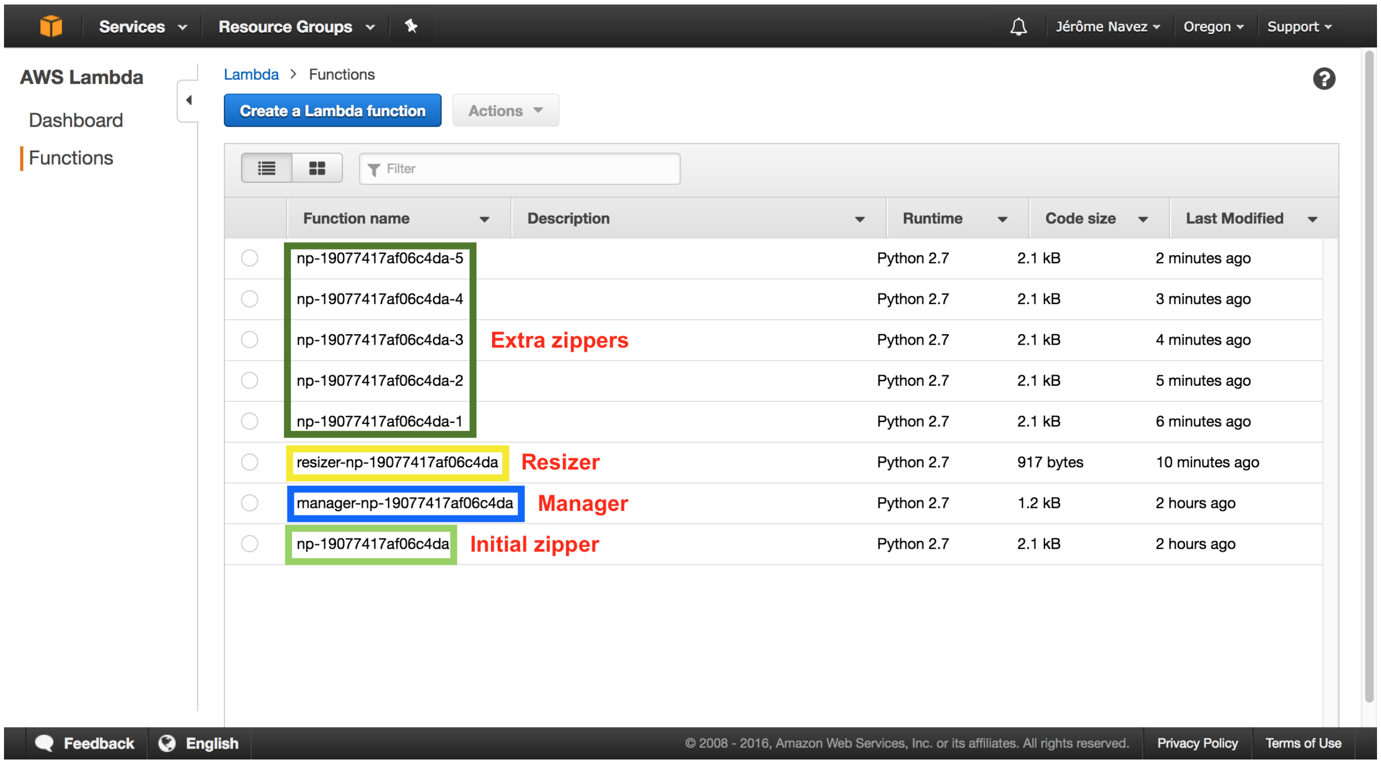
Files ignored :

Text_File.txt is not an image

Figure 7: Files ignored after analyse*

*The verification is not made just by looking to the extension of the name, but by looking to the file itself.

Extra zippers



Function name	Description	Runtime	Code size	Last Modified
np-19077417af06c4da-5		Python 2.7	2.1 kB	2 minutes ago
np-19077417af06c4da-4		Python 2.7	2.1 kB	3 minutes ago
np-19077417af06c4da-3		Python 2.7	2.1 kB	4 minutes ago
np-19077417af06c4da-2		Python 2.7	2.1 kB	5 minutes ago
np-19077417af06c4da-1		Python 2.7	2.1 kB	6 minutes ago
resizer-np-19077417af06c4da		Python 2.7	917 bytes	10 minutes ago
manager-np-19077417af06c4da		Python 2.7	1.2 kB	2 hours ago
np-19077417af06c4da		Python 2.7	2.1 kB	2 hours ago

Figure 8: The extra zippers created by the manager to support the load