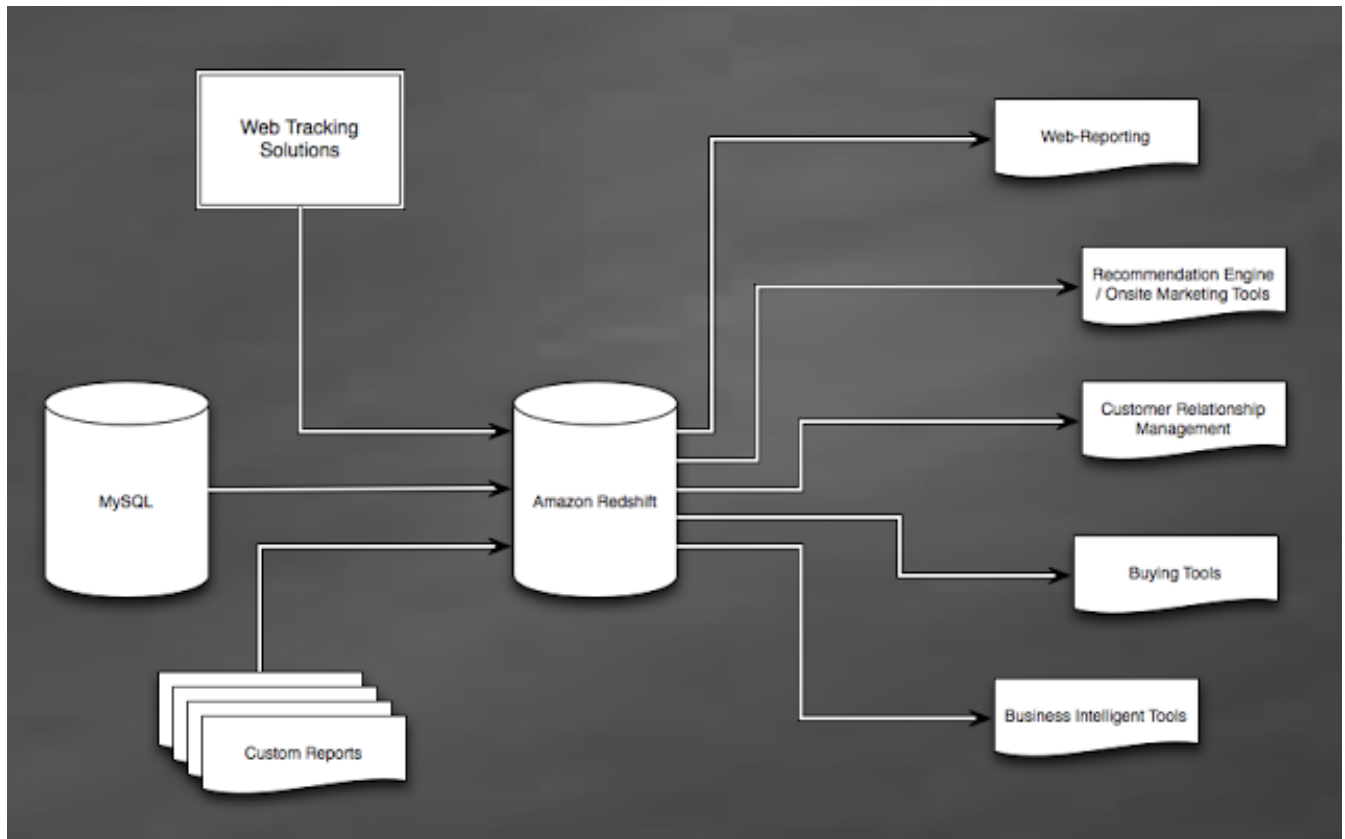


25th October 2013 Loading data (incrementally) into Amazon Redshift.

At Zalora, we (Data Science team) use [Amazon Redshift](http://aws.amazon.com/redshift/) [http://aws.amazon.com/redshift/] as our central data warehouse. Our Redshift cluster stores everything that you can think of, for example:

- Current and Historical Product Metadata
- Sale orders, sale items data.
- User on-site interactions, click-stream events, mailing events, offline events...



[http://1.bp.blogspot.com/-

uTVYKGqQMGM/UmlWg_AELqI/AAAAAAB5g/UM_x0VH9pCM/s1600/Screen+Shot+2013-10-25+at+1.18.25.png]

Amazon Redshift @ Zalora

As a result, one of the tasks we need to do is: To **load data from various FTP sources into Amazon Redshift everyday.**

As a side note, loading data into Redshift is as simple as it is, all you need is:

- An [Amazon S3](http://aws.amazon.com/s3/) [http://aws.amazon.com/s3/] Bucket that's at the same region as your Redshift Instance.
- Postgresql command line tool: [PSQL](http://www.postgresql.org/docs/8.0/static/app-psql.html) [http://www.postgresql.org/docs/8.0/static/app-psql.html] (sudo apt-get install postgresql-client)
- Redshift's [COPY](http://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html) [http://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html] command to import data from S3

However, for this particular task, there are 3 important criterias, in which, most of the time, only 2 are satisfied:

- **FAST:** **Data need to be consumed as FAST as possible.** It's generally ok to just load the entire table all over again everyday, if it's small enough.
- **SIMPLE:** There is no ETL tool for the task, thus we need to have an easily maintainable, understandable solution with minimal number of lines of code.
- **SAFE:** In short: **Fault Tolerance.** **What if a load fail one day?** **What if data does not arrive as**


expected? What if we have duplicated data load? 

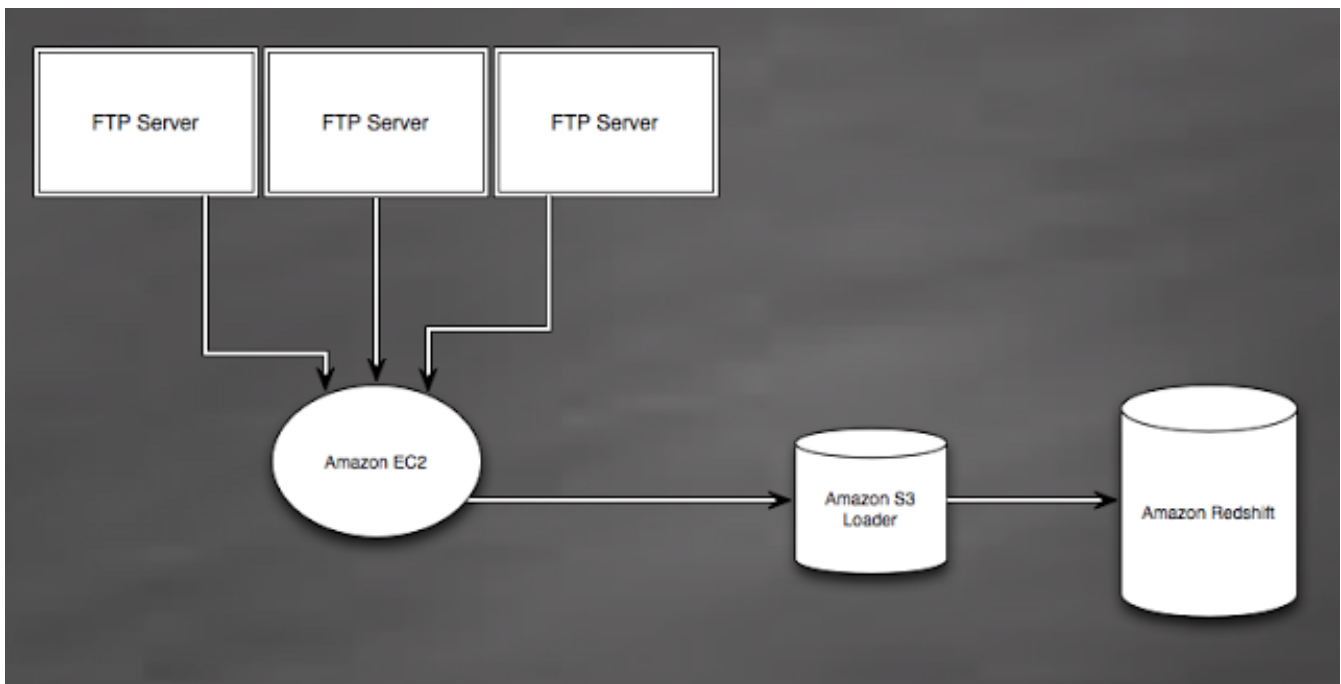
So, here is how this task was done:

- Programming Language of choice: Unix Shell Script.

The reason is simple, with all the available command-line tools:

- **LFTP** [<http://lftp.yar.ru/>] for a 1-liner FTP mirroring task.
- **S3cmd** [<http://s3tools.org/s3cmd>] for a 1-liner S3 mirroring / download / upload task.
- **PSQL** [<http://www.postgresql.org/docs/8.0/static/app-psql.html>] (note that currently, Redshift is based on PostgreSQL 8.0.2) for executing sql commands on Redshift.

- Infrastructure: A single **Amazon EC2** [<http://aws.amazon.com/ec2/>] instance (preferably EBS backed, in the same Region as Redshift Cluster) connecting to everything else: 



[<http://4.bp.blogspot.com/-JkfyvrJTeU/UmlULgOrfcl/AAAAAAAAAB5I/j7KAQkR4LTw/s1600/Screen+Shot+2013-10-25+at+1.08.36.png>]

Infrastructure Overview

- The task are functionally broken down into 5 steps: 

- LFTP mirroring from FTP Servers into EC2's EBS:

```
lftp -e "mirror -n $SOURCE $DEST;exit" -u $USER,$PASS $HOST
```

- Process data on EC2's EBS (Unzip, Rename if needed, Gzip all files)

```
unzip_and_gzip() { # $1: filename excluding ALL extensions.
    zipfile=$1.csv.zip
    gzipfile=$1.csv.gz
    if [ ! -f $gzipfile ]; then # not available then proceed
        unzip $1.csv.zip
        gzip $1.csv # gzip automatically remove csv file after done
    fi
}
```

```
}
}
```

- S3cmd sync Gzipped files from EC2's EBS to S3.

```
s3cmd --config=$CONF sync --skip-existing $SOURCE $DEST
```

- Drop and Re-Create Table Definition for each of the Data Source.

```
psql -h $PG_ENDPOINT -U $PG_USER -d $PG_DB -p $PG_PORT -c "
DROP TABLE $SCHEMA.$TABLE CASCADE;"
```

```
psql -h $PG_ENDPOINT -U $PG_USER -d $PG_DB -p $PG_PORT -c "
CREATE TABLE $SCHEMA.$TABLE
(col1 varchar(25),
 col2 date,
 col3 integer,
 col4 integer,
 col5 timestamp) DISTKEY (col1) SORTKEY (col2, col3);"
```

- COPY data from S3 into each Table on Redshift.

```
psql -h $PG_ENDPOINT -U $PG_USER -d $PG_DB -p $PG_PORT -c "
COPY $SCHEMA.$TABLE FROM '$S3'
CREDENTIALS 'aws_access_key_id=$ACCESSKEY;aws_secret_access_key=$SECRETKEY'
COMPUPDATE OFF
EMPTYASNULL
ACCEPTANYDATE
ACCEPTINVCHARS AS '^'
GZIP
TRUNCATECOLUMNS
FILLRECORD
DELIMITER '$DELIM'
REMOVEQUOTES
STATUPDATE ON
MAXERROR AS $MaxERROR;"
```

With these processes, "SIMPLE" and "SAFE" are more or less satisfied. All data are backed up on to S3 as well as EBS, **no data deletion along the process**. If needed, just fire off either LFTP, S3 sync or psql COPY command and data will be loaded into Redshift easily, otherwise, all tables will be complete re-created / fixed automatically the next day.

Everything seems easy enough, however, **here comes the problem: FAST**. As it turns out, some of the tables take more than 10 hours to load into Redshift, which means, re-creating the table simply doesn't work for us anymore, incrementally load is needed.

The main question is: **"How do we know which data is already loaded, which data is new and needs to be loaded into Redshift?"**

- Removing loaded data doesn't help, one of the requirements is to keep source data 100% untouched and retained on the cloud.
- Moving "loaded" data into other location seems ok, but it doesn't help either. LFTP and S3Sync will

still mirror everything fully back.

- Load data by date: data arrives to our FTP location can be delayed, loading it by date means we will have to deal with the problem of missing data, data duplication. Oh and and also, timezone problem.
- Store loaded data files and use them as a filter when loading into S3 or Redshift. This seems like a feasible solution, but programming wise, it is going to be more complicated and introduces unnecessary dependency: new location for storing load-history, proper data structure for load-history...

As a result, a simple "hack" is introduced:



- All gzipped files are truncated / replaced into / by empty gzipped files after loaded into Redshift.

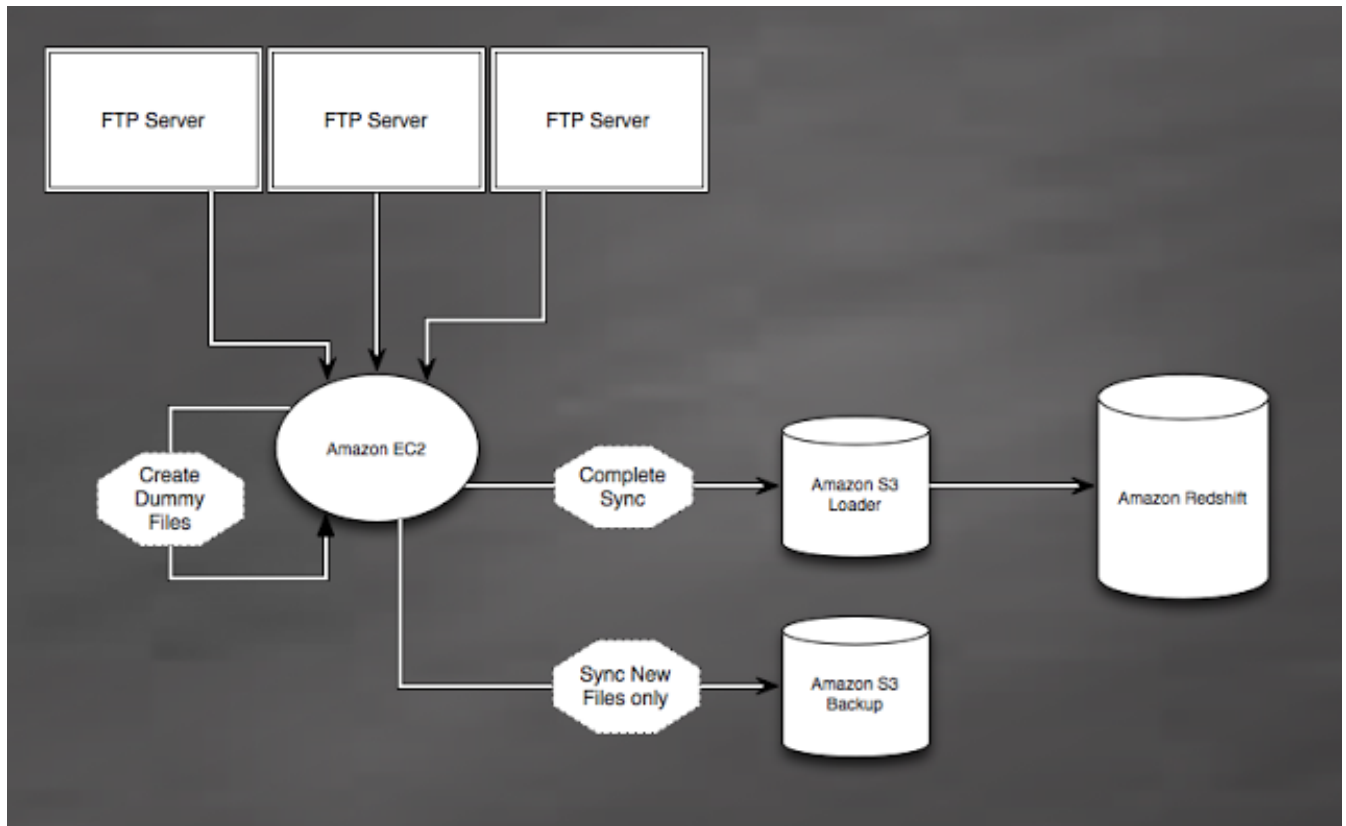
- Two S3 locations instead of just one, are used: one for incremental loading (refreshed everyday with all data, containing dummy gzipped files that were already loaded), one for backing up (only load in new data).

In the end:

- Original zipped files are retained.
- Load data can be done incrementally, regardless of the time when they were exported. As a result, a 10 hours loading process is now becoming 5 minutes process.
- 100% uptime since no "Dropping & Re-creating" task needed.
- Programming wise: nothing but a simple bash function is needed to be called after each loading job:

```
create_dummy_gz_file () {  
    rm -f $DATA/*.gz  
    zipped_files_list="$(ls $DATA/*.zip 2>/dev/null)"  
    for f in $zipped_files_list;  
    do  
        filename=`echo "$f" | cut -d'.' -f1` # anything before the dot '.'  
        touch "$filename".csv &  
    done  
    wait  
    gzip $Data/*.csv  
}
```

TL;DR:



[http://1.bp.blogspot.com/-IRJA8z9EJ_M/UmlUVJ1Gdnl/AAAAAAAAAB5Y/v5W6rgkg7is/s1600/Screen+Shot+2013-10-25+at+1.08.28.png]

Infrastructure Overview

1. Amazon EC2(s) with EBS.
2. Bash scripting with lftp, psql, s3cmd.
3. Functional design.
4. For incremental load: dummy gzipped files created after original files are loaded.

Posted 25th October 2013 by [Dat Le](#)

4

View comments



Sarvesh Gupta 24 February 2014 at 02:17

How would you insert a gz file into redshift because as far as I am reading we can't insert data using gz format.. In case you are able to – can you guide me how to do it? It is because I am also trying to insert the gz files gz using java library on linux into redshift – but unable to do so.. please help as it is urgent

[Reply](#)

[Replies](#)



Dat Le 24 February 2014 at 02:22

http://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html

Use GZIP option and you should be fine.



damodharan p 5 April 2014 at 19:42

hi dat le, how can i use this for a mysql db of with millions of rows and incremental update in the in redshift. This because file based any how sql dump wil get into singel file



Dat Le 5 April 2014 at 21:54

You dont. You re-create them every time. The best way to do it, is to create new table, copy your data over into the new table, remove old table and rename the new one into the one you need.
That way you get 99% uptime for your tables.

Reply

Enter your comment...

Comment as: Google Account ▼

PublishPreview