Case Study - ControlUp (July 2017)

https://www.controlup.com/

Field of occupation: Computer Monitoring

Company concept:

ControlUp provides inside for IT managers on their hypervisors, server farms and desktops in the organization.
ControlUp consist of two components:
1. ControlUp real-time component
   - On-prime component that helps you find and fix issues through in-OS guest view, real-time grid interface, and assisted troubleshooting.
   - Easily monitor and manage through the console's actionable dashboard and quickly resolve issues on hypervisors, virtual and physical servers, user sessions, application processes, and more.
2. ControlUp Insights

The big data analytic platform. Summarizes key metrics of the state of your monitored resources every 24 hours, highlighting key activity indicators, risk factors and performance stats.
The insights component can help and IT manager with question like sizing, system health for long period of time, user experience and application usage.

The situation before the project:

Only the real-time component was exists, no history and deep insights on the system.
The Insight component lavage the real-time component have to build a new product that will help the IT manager with different decision like: Do I need to buy more disks ? How many license the organization uses for software X ? Which application/process on my users desktop cause me problems?

Major challenges:

1. Uploading data from the customer data-centers (source real-time component) to AWS with little network disruption. A first snapshot of the runtime is uploaded, from this point on only changes are uploaded (not full record). This is done to minimize the effect of the new system on the customer site (network & minimal change on the real-time application)
2. Upload data is about computer (desktop/ hypervisors), session, processes. The data is split to 2 types:
   a. Static data (header) on the component like names, when started / end, etc. All static changes are saved (have history).
   b. Counters data like CPU, memory, IO, etc. We needed to generate a full record of all counters every 5 min even if no data was uploaded from the real-time component (no changes in the counters).
3. The real-time component data is partially, contains incorrect and in-consist data. The ETL needed to validate and fix this issues as well as generate the missing data.
   The connectivity of the entity (computer/session/process/etc) holds very important information for the system but this data might be not accurate and we need to verify and correct it.
   a. We might get or not an indication that the entity was started/created.
   b. We might get or not a keep-alive indication that the entity still running.
   c. We might get or not a stop indication that the entity is down.

   The connectivity of an entity define for the ETL process if we need to generate a 5 min counter record based on it.
4. The ETL is split to 2 sub-system:
   a. Process that prepare the data - read the uploaded partial data and rebuild a full events (filling missing events, fix data problems with the incoming data).
      After verify, fix and generate the raw events, we create aggregation data for all entities:
      i. Time-series 5 minutes aggregation events for 48 hours.

ii. Time-series 30 minutes aggregation events for 7 days.
iii. Time-series 1 hour aggregation events for 30 days.
iv. Time-series 1 day aggregation events for 1 year.

b. Serving Redshift Cluster for reporting tools
   i. Upload the generated events
   ii. Make sure that we are not saving too much history and by that manage the tables and cluster size.
   iii. Support a front end web application with hundreds of users.

5. The number of events generated by the system is based on the numbered of monitored machines on site.
   The biggest entity we monitor is the processes entity. We aggregate the process events into 2 types of objects: process & application (aggregation on the processes)
   a. In Windows each process is forking 2 process: 1 background process and 1 real process. The background process live for less than 1 second.
   b. In the aggregation tables we save all processes (include the background) for the 5 minutes time-series. For 30 minute, 1 hour and 1 day we filter the “insignificant” processes (background and non-important processes).
   c. Statistical information :
      i. Size of processes: More than 1B events per day.

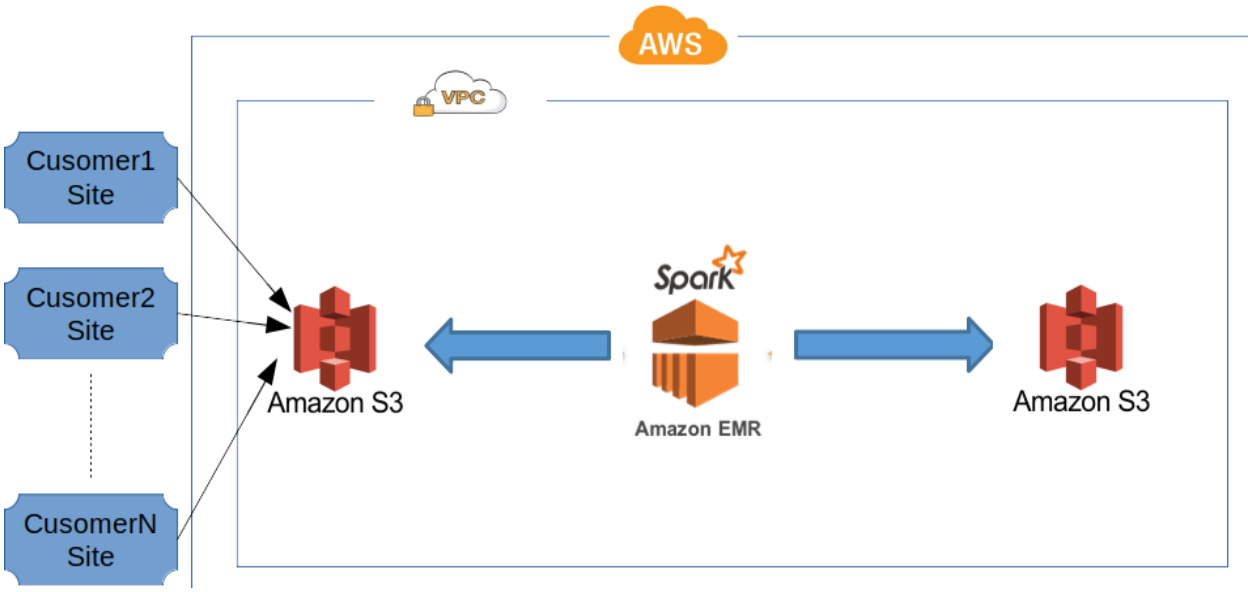
Process Time-Series	**Number of records**	**Number of Blocks**
5 min aggrigation (48 hours)	2518912084	235212
30 min aggrigation (7 days)	3314966261	258348
1 hour aggrigation (1 month)	7661702693	553496
1 day aggrigation (1 year)	11279814022	756065

ii. Size of application: More than 1B events per day.

Application Time-Series	**Number of records**	**Number of Blocks**
5 min aggrigation (48 hours)	1167399320	103875
30 min aggrigation (1 week)	1521859312	85567
1 hour aggrigation (1 month)	3217751836	176483
1 day aggrigation (1 year)	1079023582	102021

Solution:

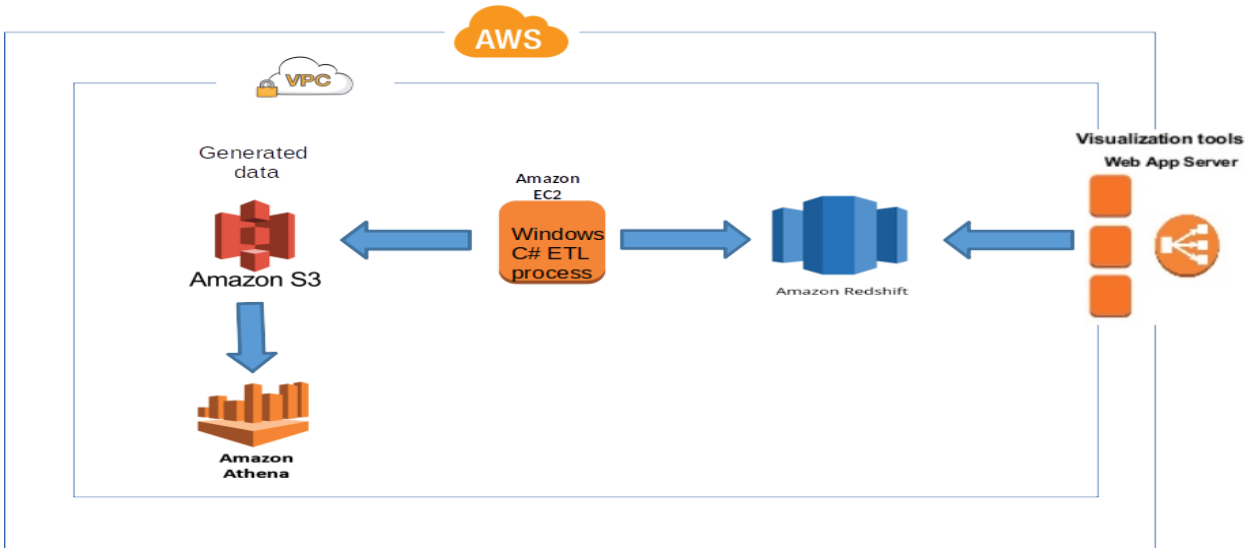
1. The process that prepare the data:
   The current solution is based on EMR (Java Spark) that fix, generate and prepare the data from the remote sites. All the heavy business logic and rules are located in this components.



a
is uploaded to S3 bucket. The data holds only the changes that occur (if any) for each reported entity.

- Each entity have 2 types of records:
  - Header metadata that holds descriptive data and non counters data.
  - Counters data
- The EMR spark job read the data from the S3 bucket, validate and fix the data, generated missing data.
  - Read incoming data from s3 bucket.
  - For the header metadata we save history changes for a few columns but for most of them we upsert the records.
  - For the counters data - Time-series data (every 5 minutes) we generate the missing records
  - Generate the 30 minutes/1 hour/1 day aggregation data.
  - Upload the generated data to s3 to be consume by the Redshift serve cluster.

2. The Redshift serve cluster
    - Ingest the generated data (from the EMR job) into Redshift. Build a solution that will minimize the needs to update old records.



– Split the data to 2 types
        - Active records – header record for data that is still alive on the system. For example: process is still running on the customer machine, machine that is still running, etc. We have 2 tables with a view over them that we flip every Ingest job. With this solution we disable the needs to update, delete and insert header records for every job.
        - Closed records – header record for data that the entity is not active. We hold data between 30 days to 1 year depends on the customer license.
        - A view that holds the close table + current active table.
    - For the counter data add the new data and manage the time we save the data depends on the customer license.
- All users are connected to this cluster:
    - Online reporting customer website. Response time depends on the complexity of the report (amount of tables + data we moving).
        - Response time for simple report: up to 5 sec.
        - Response time for medium report: up to 20 sec.
        - Response time for heavy report: up to 5 minute (we try to generate the report and send it via email).
    - Application that generate automatic reports.
    - In-house insight analytic data:
        - Part of the big tasks were moved to Athena.
        - Part internal inquiries are moved to Athena.
- Management scripts to handle: vacuum, analyze, automatic “partition tables” and view recreation.
    - Partition tables means that instead of one big table we follow that best practice on splitting it to smaller tables and a view(s). The application using the view(s).
    - Write a component that based on metadata data decide if we need to create a new partition table and rebuild the views to hold only the required historical data.
    - To manage the cluster size we needed a tool that will remove “old data” (unused data by the web-application depends on customer license). The big issue with removing (deleting) data is the vacuum full process. The solution was to save data more time that we need and run the vacuum (weekend job) on the partition table that is not the current table we add new data.
    - All data is saved also on S3 so the data-scientist and business analyst can query using Athena without overload the Redshift serve cluster.