Problem Description

Design A Google Analytic like Backend System. We need to provide Google Analytic like services to our customers. Pls provide a high level solution design for the backend system. Feel free to choose any open source tools as you want.

Criteria

Based on the problem given, the system must be able to handle these features

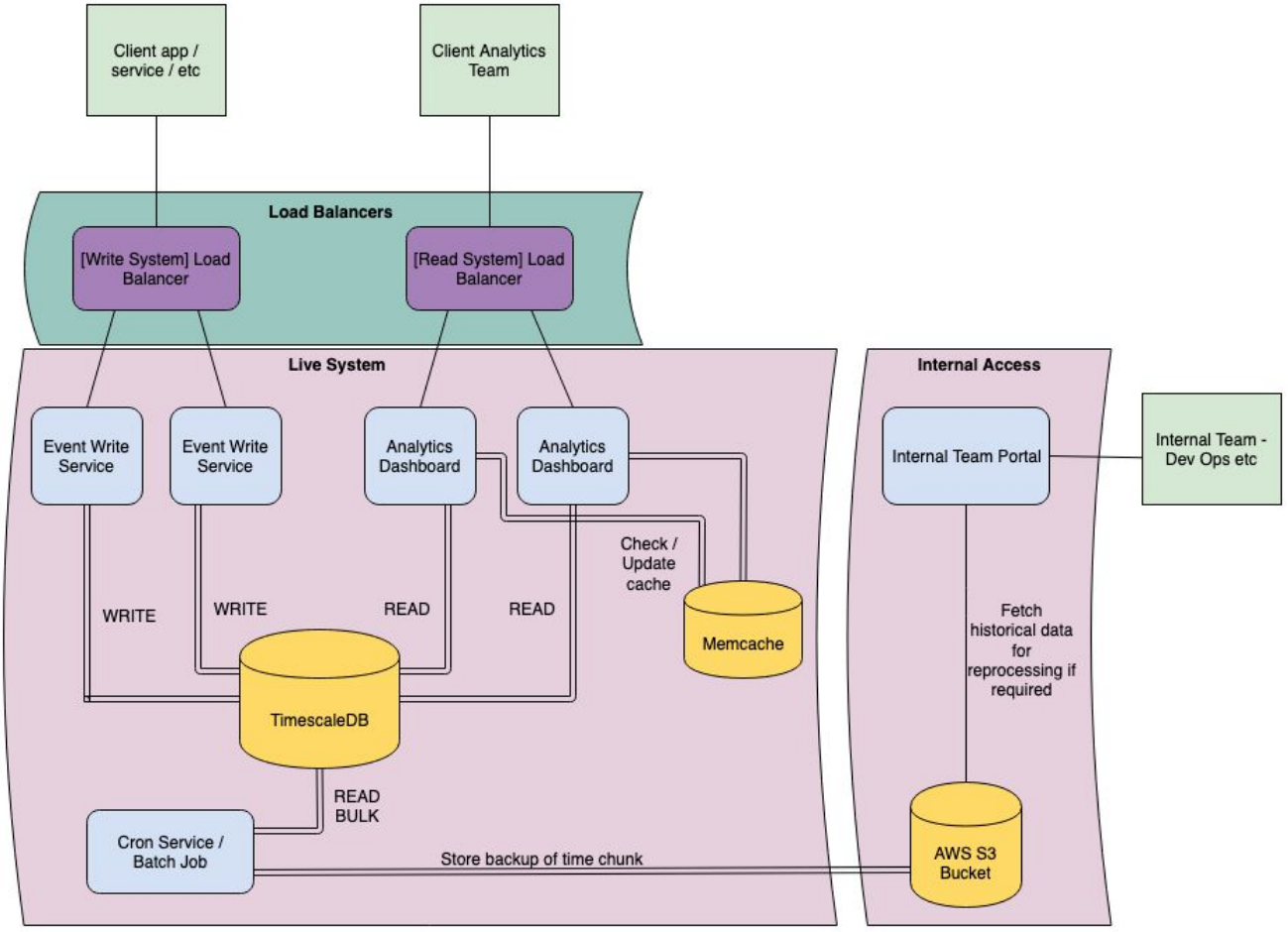
- Many writes per second
- Many reads per second
- Customer receives metrics within one hour of real time
- Minimum downtime

Solution

Introduction

Based on the problem description, I have created a system diagram to explain my solution. In the next section, I will show the system diagram, then use the following sections to explain the solution, why tools were used, and how they target the problem criteria.

System Diagram



Choice of Database

Firstly, due to predicted high rate of traffic, I have chosen TimescaleDB. It is specifically built to handle time series data (i.e. writes, rather than updates), which would suit this scenario with the forecasted billions of write events per day. As such, when implementing this we should ensure each new analytic is written as a new row, with no row updates. I believe this will help address the ‘handle large write volume’ element of the problem description.

Writing Analytics Events

I have chosen to separate the ‘read’ and ‘write’ aspects of the system as much as possible. I have placed a separate load balancer for all write events, with a setup of 1-n microservices instances behind it for writing events to the DB. With this setup, we should be able to scale with varying load, etc. I will talk about scaling more in the autoscaling section.

Based on the problem description, I am envisioning the ‘Event Write Service’ to be quite simple instances (input validation, logging, writing to database), so they seem very suitable for a microservices style deployment. Depending on the load, we can possibly free up additional load by having the Event Write Service spawn a new thread (for e.g. Java implementation) to process the event, and just return a http response immediately. This should minimize the time a network event takes, and therefore increases the amount of events we can support simultaneously.

Reading Analytics Events

For reading events, I have chosen to utilize a combination of the backend DB (TimesaleDB) as well as Memcache to reduce stress on the DB. When a client requests metrics through the dashboard, the dashboard backend can first check if that particular time series data exists in Memcache. If it exists there, it can return that data. If the data does not exist (or if it is over 30 minutes old), it can fetch it from Timescale DB, update memcache, then return the data.

Like with the write services, I have chosen to keep the dashboard (i.e read) calls behind its own load balancer. This should give us more flexibility and ultimately uptime - if the write services fail, then the client is still able to get read metrics.

Reproduce Historical Data

For recording historical data, I have chosen to create a separate application to be used for reading events in bulk, and preserving copies in a file data store. For my implementation I referenced writing to an Amazon S3 bucket, but other cloud data stores may be sufficient.

The batch job should pull time series data from TimescaleDB at fixed intervals (I will initially suggest 3 minutes, but could be tweaked) and write the output file to S3. For implementation of the batch job I will suggest Java’s qwartz library (as that is what I am familiar with), or something like Spring Batch.

In this way, historical data can be accessed directly from the cloud without needing to interfere with the Live database. The problem description states that the system should ‘*have the ability to reprocess historical data in case of bugs in the processing logic*’, and hence I am assuming that is an internal team who will need access to this data. Hence, I suggest an internal portal / site should be built to enable certain peoples to access this data. The internal portal will allow searching by time parameters.

Autoscaling / Deployments

The problem requires that the system be able to ‘run with minimum downtime’. I believe autoscaling and the way we manage deployments to the system will both touch on this requirement. Firstly, I believe we can scale deployments of our microservices depending on load. I know AWS supports autoscaling, but I am not mandating any particular service here. If one particular instance is malfunctioning, we can use the load balancers to switch traffic to a new instance and bring down / restart / investigate the broken instance. We can use a simple health-check system to monitor whether a particular instance is green or red, and controls terminations / restarts that way. Kubernetes and AWS both have support for health checks so we can make use of that

Secondly, we can of course utilize the load balancers for any new deployments we make - deploy the new service, perform some tests on it, then allow the load balancer to send traffic to it.

Further Options

One possible further consideration that may need to be made is around data cleanup. As we are expecting billions of writes events per day, we are going to have to deal with very large amounts of data. We may consider only allowing metrics searches within one year of the current date, and moving data out of the Timescale DB to S3 / Glacier / Redshift etc.