Scenario 1: Logging

In this scenario, you are tasked with creating a logging server for any number of other arbitrary pieces of technologies.

Your logs should have some common fields, but support any number of customizeable fields for an individual log entry. You should be able to effectively query them based on any of these fields.

How would you store your log entries? How would you allow users to submit log entries? How would you allow them to query log entries? How would you allow them to see their log entries? What would be your web server?

Solution:

Personally, I will use MERN for this scenario 1.

Log entries will stored in MongoDB as documents(each entries) & sub-documents; If the storage is too big, such as 300GB, we should use Apache hadoop for distribution storage.

Submit entries should use the MongoDB insertOne() fuction, as for user interface there will be a submit table.

Query entries using findOne() function in MongoDB, also provide with user interface. For big data sets, using ES(ElasticSearch) is better at performance.

After search function returns json, we provide the result to users in UI using Express.

Web server would use node server. If we want to be quick, redis should be included.

Scenario 2: Expense Reports

In this scenario, you are tasked with making an expense reporting web application.

Users should be able to submit expenses, which are always of the same data structure: `id`, `user`, `isReimbursed`, `reimbursedBy`, `submittedOn`, `paidOn`, and `amount`.

When an expense is reimbursed you will generate a PDF and email it to the user who submitted the expense.

How would you store your expenses? What web server would you choose, and why? How would you handle the emails? How would you handle the PDF generation? How are you going to handle all the templating for the web application?

Solution：

I think it’s better to use XAMP model;

Expense details are close related, therefore storing expenses details in MySQL, a relation database, is better;

Web server will be Apache.

Emails will be handled by SMTP service;

PDF generation will use LaTeX to generate PDF report;

Templating: there are some templating model, such as handlebar.

Scenario 3: A Twitter Streaming Safety Service

In this scenario, you are tasked with creating a service for your local Police Department that keeps track of Tweets within your area and scans for keywords to trigger an investigation.

This application comes with several parts:

- An online website to CRUD combinations of keywords to add to your trigger. For example, it would alert when a tweet contains the words (`fight` **or** `drugs`) AND (`SmallTown USA HS` or `SMUHS`).
- An email alerting system to alert different officers depending on the contents of the Tweet, who tweeted it, etc.
- A text alert system to inform officers for critical triggers (triggers that meet a combination that is marked as extremely important to note).
- A historical database to view possible incidents (tweets that triggered an alert) and to mark its investigation status.
- A historical log of *all* tweets to retroactively search through.
- A streaming, online incident report. This would allow you to see tweets as they are parsed and see their threat level. This updates in real time.
- A long term storage of all the media used by any tweets in your area (pictures, snapshots of the URL, etc).

Which Twitter API do you use? How would you build this so its expandable to beyond your local precinct? What would you do to make sure that this system is constantly stable? What would be your web server technology? What databases would you use for triggers? For the historical log of tweets? How would you handle the real time, streaming incident report? How would you handle storing all the media that you have to store as well? What web server technology would you use?

Solution:

Twitter API: there is an Enterprise Search API which allows to search; one is for recent 30 days, second one is for Full-archive search. So as request I will choose the second API;

Web Server: Redis, which is faster;

Database: Apache Hadoop; quick and reliable, distributed storage.

Historical log of tweets:Full-archive search of Twitter API, which can search all the historical records

Storage: store these data in the hadoop, may also using label is better ?

Web Server Technology: Elastic Search(for quick search in the database), and SMTP service(for sending emails)

Scenario 4: A Mildly Interesting Mobile Application

In this scenario, you are tasked with creating the web server side for a mobile application where people take pictures of mildly interesting things and upload them. The mobile application allows users to see mildly interesting pictures in their geographical location.

Users must have an account to use this service. Your backend will effectively amount to an API and a storage solution for CRUD users, CRUD 'interesting events', as well as an administrative dashboard for managing content.

How would you handle the geospatial nature of your data? How would you store images, both for long term, cheap storage and for short term, fast retrieval? What would you write your API in? What would be your database?

Solution:

Geospatial nature: will stored as an related label or attributes with related photo.

Storage: Long-term:Store in Cloud Drive or some database; long-term and permanent;

Short-term: Store in memory or local; Which will be quicker for retrieving the data.

API: using C/C++, which can perform as what I want, with high performance of the operation;

Database: Using cloud service such as AWS, or using MongoDB/ MySQL;

Because this case doesn’t need too much storage space and calculating;(In my opinion)