Fellows who have gone through the data engineering program have told us that coming up with a project idea to work on at Insight is one of the most challenging parts of the program. We've put together this document to help guide you in choosing a project idea.

We've outlined several project seeds in this document that we hope will help put you on a good path toward coming up with your Insight project idea.

Each project seed

- Will help you start thinking about project ideas
- Highlights why the technology or idea may appeal to hiring managers

A project seed should provide enough information to get you 85% of the way to a solid Insight project idea. The rest of the project idea should come from you, your unique take on the problem and additional research and discussion you conduct between now and when project ideas are finalized in Week 1 (week of Jan. 13).

In each project seed below, we'll discuss

- **Overview**: What's interesting or challenging about the topics covered by this project seed
- **Engineering challenge(s)**: Highlights of technical challenges you might encounter that could be interesting and exhibit your problem solving and engineering abilities
- **Possible approaches**: Ideas on how to go from seed to project
- **Functional highlights**: Sample business use case where this project seed could be interesting (Keep in mind we expect you to bring other ideas or examples of how a business might use what you will build rather than relying on our example)
- **Presentation highlights**: Tips on how to present your work
- **Additional readings/resources:** Find some industry blogs that inspire us and may provide you additional information on the project seed
- **Related Insight projects**: Some project seeds may list past Insight Fellow projects that have worked on topics in the same vein as what is described in these seeds. Please use these examples only as inspiration. We (and hiring managers) want you to bring your creativity and thoughtfulness to your project idea.

Enjoy!
Insight Data Engineering Team

Table of Contents

Building a real-time data analytics platform involving schema evolution

Overview

One popular trend in data engineering is the rise of real-time analytics. Using tools like Apache Kafka and Pulsar, companies are striving to build platforms that can explore business intelligence data immediately. At the same time, “real-time” also means handling changing data sources and structures as they evolve. Imagine you are collecting online shopping “users” data with a schema of

```
{"ID", "Name", "Address"}
```

After a while, you realize you need to add attributes “email” and “age”, so you have:

```
{"ID", "Name", "Address", "email", "age"}
```

Another while later you decide to divide “Address” into “Street”, “Zipcode” and “State”:

```
{"ID", "Name", "Street", "Zipcode", "State", "email", "age"}
```

The simplest way to proceed is to stop the application, perform the needed schema changes and then restart. But in the era of digital intelligence, every second counts. Additionally, providing full compatibility of different data formats also is important. Data Engineers want to decouple data processing components and ensure that all data coming into the system is encoded to support backward compatibility (new code can read old data) and forward compatibility (old code can read new data).

Schema serialization tools like Avro, Protobuff, and Thrift can be used to solve these problems. For example, Confluent Schema Registry is a platform based on Kafka and Avro that supports the evolution of schemas based on multiple compatibility settings. It’s a very good starting point for designing a robust real-time data analytics platform.

Engineering challenges

The first challenge is to design your real-time analytics platform in order to reach different compatibility levels. Take Kafka’s “Producer” and “Consumer” as an example, “forward” compatibility means that when Kafka producers have a newer version of a data schema, consumers will be able to still process data under the new schema.

To achieve this, under the “forward” compatibility contract, newer schemas can only delete fields or add optional fields. Combined with your dataset and business requirements, what kind of

compatibility will your system need? Is this compatibility stable? With that compatibility level in mind, how would you design your schema? Another challenge can the management of schema evolution including manual/automatic schema registration, compatibility checks, or changing compatibility modes to meet your application needs.

Possible approaches

Start with building a real-time data analytics platform where data is streaming in and undergoes some sort of computation or data transformation. From there, consider how if the data coming in were to change (e.g., new fields added, deleted or modified), how might your platform support backward and forward compatibility with no downtime. You can find a previous Insight project dealing with real-time streaming and then think about if you can redesign the data pipeline with a robust schema to achieve this, or find your own use case.

Functional highlights

There are a lot of real-time analytics examples including social media/tech trends, online transaction records ingesting, online advertising, real-time events check-ins analytics and log info ingesting.

Below are two use cases that can help your brainstorming, but the best story will come from your own exploration of various datasets and business insights.

1. Design a logging system for a busy and growing online shopping website. Maybe it starts with a limited set of features (e.g., customers, orders) but expands over time to include more data, such as email campaigns and multiple shipping addresses.Your task is to write all kinds of order-related activities to a database, such as ordering, shipping, reviewing, returning, customer complaints, refunds, etc.
2. Pick a city with public complaints or incident data (some cities publish this data as 911/311 calls), build a real time analytics platform to answer questions like data aggregation by categories/zip codes.

Whatever use case you choose, it should feature instances where schema management would be highly beneficial, and whatever data platform you build should support schema evolution and reach some level of compatibility.

Presentation highlights

Telling a good story will start with an interesting use case that requires sophisticated schema management, and features a distributed data streaming platform with decoupled data processing components that allow for schema evolution and potentially, rolling deployments if you have time. This will allow you to showcase your business sense and understanding of big data processing trends. Designing a robust schema to reach different levels of compatibility will also show your data engineering skills.

Additional readings/resources

- Schemas, Contracts, and Compatibility
- Introduction to Schemas in Apache Kafka with the Confluent Schema Registry

Related Insight projects

- TaxiOptimizer
- AdMorphous

Tracking evolution of data at large scale

Overview

As the demand for (relational) databases has grown and complexity of data processing increased, so has the need for better fault tolerance. Fault tolerance in terms of better server resilience and crash recovery at the infrastructure level exists. But that still doesn't save us from an accidental query that may delete data from a table/database. While it's possible to recover from such scenarios, it's not an easy path. For example in MySQL, the point-time-recovery involves quite a few steps, which also requires frequent and expensive backups. These factors make it complicated to easily recover a subset of deleted data. Is there a better solution?

Managed services like AWS Aurora have features like point-in-time recovery.

This project is also an opportunity to provide visibility to how much individual data-points have evolved -- effectively creating an audit trail of how data evolves over time -- often described as data lineage. This will be of interest to many industrial applications.

Engineering challenges

Each and every update applied to a database table should be captured in great detail. This should allow you to playback the table-status.

Other engineering challenges would be around the cost this additional overhead might carry, such as impact to database performance and when the added cost of this feature would outweigh the benefit.

Also, to really stand out, you'll want to put thought into what fields to track and how to interpret how the data evolves over time and what that means (e.g., if you were working for a social media company and tracking a customer profile table, your platform might be able to identify how frequently customers are changing their user details)

Possible approaches

Use Cassandra to capture every value evolution of a data-point. This in conjunction with Presto would let you query specific value for any key at any point-in-time.

An alternative approach could be to use CockroachDB to capture the same. With some clever data-modeling and data retrieval techniques, point-in-time data should be feasible.

Functional highlights

A data-science team would be interested to detect patterns in data evolution for each and every data point. This isn't possible with any existing systems.

Presentation highlights

Projects that come out of this seed would focus on building a system that enables the capture and recovery of any point-in-time. You can start off with a state captured in a dataset. Apply meaningful updates to these datasets and then apply them through this new pipeline.

Additional reading / resources

- https://medium.com/netflix-techblog/dblog-a-generic-change-data-capture-framework-69351fb9099b
    - Recently published blog that is very relevant to this project seed
- https://www.infoq.com/news/2019/11/data-synchronization-netflix/

Scaling Time Series Data Storage

Overview

Recent technology advancements have improved the efficiency of collecting, storing and analyzing time series data, spurring an increased appetite to consume this data. However this explosion of time series data can overwhelm most architectures not designed with this in mind. As video streaming services increase in popularity, more scalable architectures are needed.

Engineering challenges

People watch hundreds of millions of hours of content on video streaming services every day. Each member provides several data points while viewing a title and they are stored as viewing records. Streaming platforms analyze the viewing data and provide real time accurate bookmarks and personalized recommendations.

Viewing history data increases along three dimensions:

- As time progresses, more viewing data is stored for each member
- As member count grows, viewing data is stored for more members
- As member monthly viewing hours increase, more viewing data is stored for each member

Building scalable architectures can address challenges of cost and performance. Two possible approaches to scaling time series storage for streaming data are outlined below. Engineering challenges include parallelism, compression, and improving the data model.

Possible approaches

- **Live and Compressed Storage Approach:** Implement a caching system to make recent time series data highly available while archiving older data.

  Live or Recent Viewing History: Small number of recent viewing records with frequent updates. The data is stored in uncompressed form as in the caching layer.

  Compressed or Archival Viewing History: Large number of older viewing records with rare updates. The data is compressed to reduce storage footprint. Compressed viewing history is stored in a single column per row key.

  Below are diagrams depicting a sample architecture that Netflix implements internally. Some of the technologies Netflix has released to open source, such as EVCache. However, be aware that the Readme for EVCache is barebones and there are few resources to help in the installation and configuration. As a result, we don't recommend

you use it given the short timeframe you will have to complete your Insight project. Other tools to investigate include memcached and Redis.

If you are focusing on building a caching solution, zero in on how to implement that first and foremost rather than installing and configuring all of the same technologies that Netflix has in its ecosystem, because if you do the later, you will not finish your project in time and may not be able to present it to any companies because your project is incomplete.

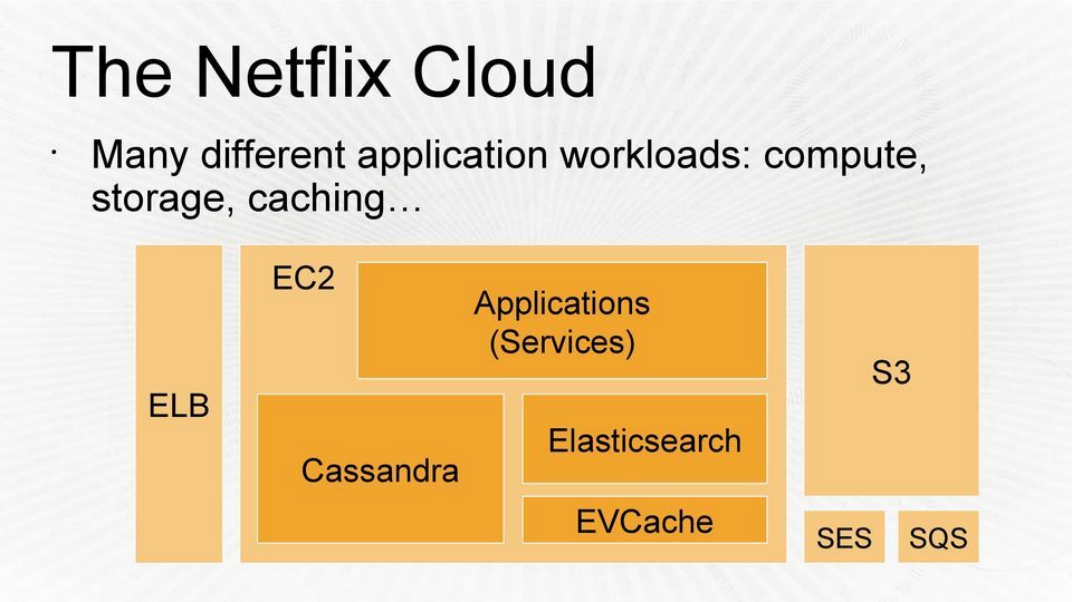


Diagram: System could be built using Memcached, Cassandra, and S3 as Netflix has done but be aware this architecture is well out of the scope of a 3-week Insight project

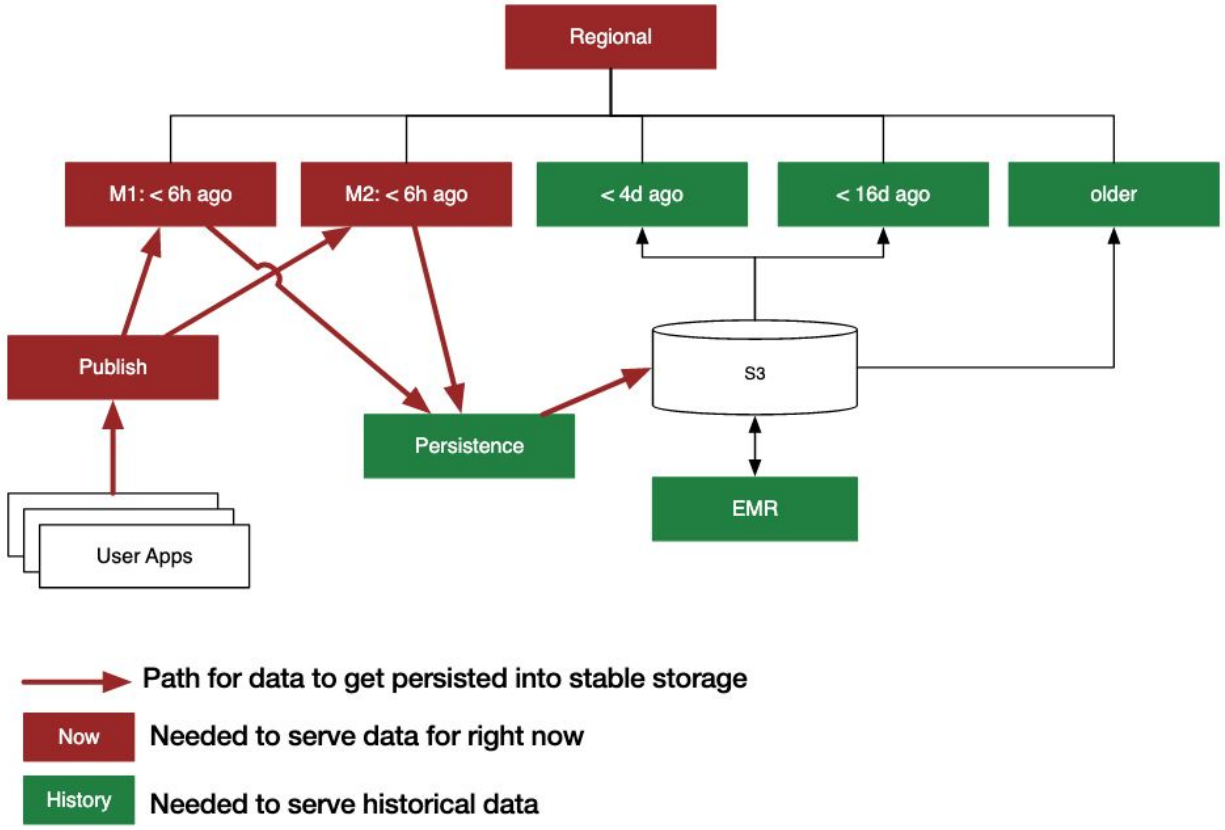


Diagram: Data can transition between live and compressed data based on data freshness.

Functional highlights

Implement a scalable system that can meet the requirements for a massive dataset, tackling a unique engineering challenge:

1. Parallelism - large read/write operations can reduce performance. Implement chunking to shard user data across nodes (Netflix describes how they did it with Cassandra here)
2. Compression - create a high availability cache to store recent data and store compressed older data separately.
3. Improved data model - read the blog post 1 and blog post 2 for ideas about how streaming services are currently implemented and design a way to improve performance or reduce storage footprint.

Presentation highlights

As organizations have increasingly large datasets (some of our partners work with data at the petabyte scale), it's important to learn not just how to use common tools but how to architect a system that is scalable and reliable, while considering cost.

Additional readings/resources

- Scaling time series data storage, Part 1: https://medium.com/netflix-techblog/scaling-time-series-data-storage-part-i-ec2b6d44ba39
- Scaling time series data storage, Part 2: https://medium.com/netflix-techblog/scaling-time-series-data-storage-part-ii-d67939655586
- Distributed in-memory data store: https://medium.com/netflix-techblog/announcing-evcache-distributed-in-memory-datastore-for-cloud-c26a698c27f7
- Cache warming: Agility for a stateful service: https://medium.com/netflix-techblog/cache-warming-agility-for-a-stateful-service-2d3b1da82642

Building a database API

Overview

For companies that have a database of record (i.e., single source of truth), they often have to balance who can access the database, for what purposes and how often. One set of database queries can have unintended consequences for another. For instance, hitting the database for internal purposes (e.g., running analytic reports or pulling data) could affect performance and have other unintended impacts to the database for customer-facing queries (e.g., customer updating their profile or making a purchase) occuring at the same time.

To allow application developers, analysts and other users access to the same data but with checks that ensure certain queries are not negatively impacting database performance, accidentally overwriting data, changing the expected meaning of certain fields or causing other unintended consequences for other users, data engineers are often asked to create a layer of abstraction around a database. They do this by creating an API that users can use to access a database or set of databases. The API then internally identifies performance bottlenecks and tries to alleviate them or provides some sort of protection around unintended consequences.

Engineering challenges

Making tradeoffs and design choices are an important part of designing and building an API. Consider your use case first and then the engineering challenges will become more apparent.

For instance, if your project idea will involve a large volume of database requests (e.g., many queries or thousands of users), the engineering challenge may be around juggling all of those requests and optimizing performance. If your project idea involves modifying highly private or sensitive data, then the engineering challenge may be controlling access to certain tables or data updates.

Another consideration is the API design. Would the API use a REST architecture or would a work queue and/or publish-subscribe messaging queue offering, such as RabbitMQ, be a better fit?

Possible approaches

Start with a dataset that you know well, create a schema for how it'd be represented in a database, such as MySQL or Postgres and then load a small dataset into a relational database. Now consider access patterns to how more data would be inserted, existing data updated or deleted and queries to select data using a certain criteria. It'll be important for you to identify what company or set of users might add/delete/update/select such a database and how often they'd want to do that work.

If you want to optimize for performance, consider which operations could be done in bulk, would caching or having read replicas of the database alleviate bottlenecks, what about scheduling a batch job during off-peak times? If your dataset is fairly modest and you don't expect the query volume (often, expressed as QPS or queries per second) to be very high, consider providing more access controls (e.g., only certain users can update or view this data) or data quality checks (e.g., address fields should follow a certain format).

Once you've nailed down your use cases, consider whether you can get by with a RESTful API framework such as Flask or Django, or if a messaging queue, such as RabbitMQ, would be also needed . The latter typically would be used if you'd expect a long line of requests to the database, and therefore, would want to "queue" those requests and handle them asynchronously. But what tradeoffs are you making by handling data updates asynchronously -- would there be a possibility of data corruption, race conditions or unacceptable time lags?

Functional highlights

This project would work best if you already have a dataset or schema that you have worked with in the past, and a project idea around how a business or set of users might want to add, update, select or delete on a frequent basis. Then, consider how you ensure high volume of queries per second or rules that prevent "bad" data from being added or updated.

To make this project seed your "own," think through data that you've worked with in the past or use cases where many people were accessing the same database (e.g., customers, data analysts and web servers hitting the same database at the same time).

Presentation highlights

Backend engineering teams are always building or supporting APIs for their teams. Engineers with experience building and supporting an API are in high demand. Showcasing your ability to build an API and then discussing the tradeoffs you made in designing your service demonstrates a valued skill and would allow you to communicate some of the trade-offs you made in the process.

If you are focusing on optimizing the performance of your API platform, then you also should be prepared to simulate many queries coming in during a short-time and how your platform handles it.

Additional readings/resources

- Pinterest engineering blog: https://medium.com/pinterest-engineering/using-kafka-to-throttle-qps-on-mysql-shards-in-bulk-write-apis-a326ae0f1ac1

Cataloging a Data Lake for Exploration

Overview

Many companies generate data from a variety of events throughout their architecture meaning data is stored across a number of locations and formats including relational databases, NoSQL databases, and logs. As more data is collected at a single company, the data landscape can become increasingly fragmented, complex, and siloed. Build a data platform that can interoperate across these data sets as one “single” data warehouse so that it is possible to discover, process and manage.

Engineering challenges

Accessing each section of the dispersed data will require some centralized schema or processing engine. Additionally, each separate data store may be very large. Some enterprise companies commonly ingest several terabytes (TB) or even petabytes (PB) of data for storage.

Building an abstraction layer between the data and information about the data (i.e., metadata, such as source, data format, date added, date modified) may be needed, allowing you manage what happens each time a new data store is added, removed or schema altered. This might require you to automate the process of registering a new schema or data source based on certain triggers or conditions, such as periodically scheduled events or when records match known datastore formats.

Possible approaches

Store several large datasets in a variety of formats across just as many computing clusters or servers and

1. Introduce a common abstraction layer, where users can register datastores, define the schema, and store metadata about each datastore. Consider using a graph database such as GraphQL or Neo4j for this layer to map relationships across datasets, tables, or other attributes. Alternatively, Apache Atlas could be used as the “metadata engine” for this layer.
2. Build a simple web-app dashboard to display the abstraction layer’s relationships and statistics on metadata and schemas. See the presentation highlights for ideas on what to display.
3. Once a common abstraction layer is established, Introduce schema and metadata versioning.

Functional highlights

To fully utilize disparate data resources, the data needs to be processed, explored, and analyzed by users who may not necessarily own or be aware of the data. A live catalogue of available data may make data exploration possible. It can facilitate tracking the metadata changes for a specific column or offer the ability to view table size trends over time. Being able to ask what the metadata looked like at a point in the past is important for auditing, debugging, and also useful for reprocessing and roll-back use cases.

Presentation highlights

1. Format metadata statistics into a dashboard running real-time updates on metadata such as the number of records or schema changes.
2. Track queries in each datastore and highlight frequently accessed data.
3. Demonstrate how, by using the dashboard, a user might learn about data because they are new to a company, the data is new, or they do not own the data.
4. Demonstrate how the catalogue manager accesses the common abstraction layer to the additional datastore.

Additional readings/resources

1. Atlas is Data Governance and Metadata framework for Hadoop
   https://atlas.apache.org/#/
2. Metacat: Making Big Data Discoverable and Meaningful at Netflix
   https://medium.com/netflix-techblog/metacat-making-big-data-discoverable-and-meaningful-at-netflix-56fb36a53520
3. Open Sourcing Amundsen: A Data Discovery And Metadata Platform
   https://eng.lyft.com/open-sourcing-amundsen-a-data-discovery-and-metadata-platform-2282bb436234

Unified real-time and historical analytics platform

Overview

More and more companies are ingesting and processing data in real time. Traditionally, this data will be processed in real time but also stored in a database/data warehouse for historical analysis. In this project seed, we will discuss a new trend that eliminates the need for a database while still enabling historical analysis.

Building a platform that can process tens of thousands of events per second is not a small feat and typically starts with a so-called *ingestion framework*. You can think of an ingestion framework as a message hub: all incoming data will first be recorded here before it will be further processed. Two popular open source ingestion frameworks include Apache Kafka and Apache Pulsar.

This project seed focuses on a new feature in Apache Pulsar: tiered storage. Traditionally, data in Kafka and Pulsar has a retention period, after which it is deleted (this is necessary to avoid running out of disc space on the machines running Kafka/Pulsar). Pulsar enables you to move old messages to external data stores (such as S3) to avoid this issue. In addition to moving it to external storage, it also allows you to query this data using its SQL engine (Pulsar SQL).
With these new features, you can use Pulsar for both real-time and historical analysis and eliminate the need for a separate database/data warehouse.

Engineering challenges

1. A key feature for this project is tiered storage. It is important to fully understand how it works and what parameters an engineer can and must tweak to make it run efficiently.
2. You can visualize your data streams and your data processing workflow in what is called a processing topology. Defining a processing topology that will serve both real-time and historical analysis will be challenging. Here is a good resource for processing topologies (it focuses on Kafka but translates almost 1-1 to Pulsar).

Possible approaches

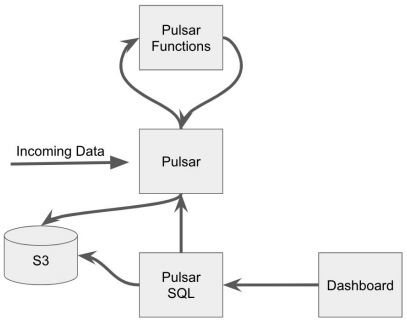
1. You could build your entire platform around Pulsar. Set up a Pulsar cluster and start ingesting your raw data. Use Pulsar Functions to process this data in real-time: clean raw data and compute real-time analytics. You can then re-ingest your results into Pulsar.

   Now, set up tiered storage with S3 as the external store and define when old data gets moved from Pulsar (which uses Bookkeeper internally to store data) to S3.

You can then set up a Pulsar SQL cluster to query the data that is both stored in Bookkeeper and S3 simultaneously. This will allow you to compute historical data analysis.

As a last step, build a dashboard to visualize your results for demonstrating your project.

Here is what your architecture could look like:



2. Rather than using Pulsar's internal processing and query frameworks, you could use a separate processing engine like Spark. In this case, after setting up Pulsar, you should set up a Spark cluster and use the Spark-Pulsar connector (tutorial here) to connect both frameworks.

   You can then use Spark Streaming instead of Pulsar Functions to perform streaming processing. Spark Streaming can reingest its output into Pulsar.

   For batch computations and running queries against historical data, you can use Spark/SparkSQL to process all of the data.

   This approach is significantly more complicated than the first one and should only be followed if the use case demands it and you already have prior experience with Spark Streaming. Be sure to chat with your Program Director and get approval for this route if you intend to take it.

Functional highlights

It is important to find an interesting use case that demands both real-time and historical data analysis. It may be a good idea to start with an interesting streaming problem and then determine if there may also be interesting insights in the historic data.

Presentation highlights

1. Finding an interesting use-case that demands both real-time and historical data analysis will be important to convey the key advantages of a combined stream and batch platform.

2. Highlighting how you address key challenges (finding the right processing topology, effective partitioning for both streaming and historical data) will allow you to showcase the technical skills learned in this project.

Additional readings/resources

- Apache Pulsar as One Storage System for Both Real-time and Historical Data Analysis
    - This blog is from a company founded by core members of the Apache Pulsar team - take their remarks about Kafka with a grain of salt.
- Stream processing with Pulsar Functions.
- It is also on Kafka's roadmap to support tiered storage.
- PulsarSQL internally uses Presto, a popular SQL engine developed by Facebook.

Related Insight Projects

- Xiaolei Zhu

Identifying code similarity at scale

Overview

Identifying data or code that has been duplicated or copied from one source to another can be a rather challenging and time-consuming process. Programs, such as Moss, are available to detect code similarity, however, it's slow and resource intensive. Building a platform that optimizes the process through distributed computing or filters that can narrow the number of comparisons can be an interesting challenge. Note that we are not asking you to develop an algorithm for detecting similarity -- that would be something a Data Science Fellow might focus on exclusively during his or her three week Insight project. Rather this seed proposes taking an already existing similarity algorithm that has been proven to get the right results on a subset of data and show that you can apply it to a large dataset and improve its performance or reliability via prefiltering or distributed computing techniques.

Engineering challenges

Identifying data or code duplication can be a complex problem, and engineers are always looking for ways to optimize the process. For instance, there may be an approximate solution or a quick way of pre-filtering or narrowing the subset of data or code you then pass to a code plagiarism detector or NLP model. While identifying text or code similarity is one category of projects, if you have experience with images or audio files, feel free to use that as your dataset.

Possible approaches

1. Identify code base or repository of similar data
2. Create a hashing algorithm, such as a locality sensitive hashing algorithm that can help narrow your data and number of comparisons you end up making
3. Measure how good your hashing algorithm is and validate the accuracy of the algorithm you use and ensure that it doesn't degrade when scaling.
4. An alternative to focusing on prefiltering your data is to work on scaling the computation such that if there's an error on one of the machines doing the work goes down, a backup or another instance can pick up where the work was left off rather than having to manually re-run the entire job from the start.
5. A third alternative would be to distribute the workload across multiple machines. This would essentially boil down to doing simple cluster management in conjunction with Apache Airflow or another scheduling tool. Create a Directed Acyclical Graph of several similarity detection jobs and start executing them. A stretch goal would be to dynamically scale the number of workers, which would require service discovery.

Functional highlights

Kaggle and other type of competitions often depend on participants submitting unique code solutions and need this to happen on a quick and automated way. Text, image and audio companies also want to ensure that they are protecting their intellectual property.

Presentation highlights

Many Fellows have found success in outlining their problem solving skills and ability to think outside the box in when designing performant and fault tolerant pipelines. Also, while you may not have created the similarity algorithm that you've used, hiring managers, which sometimes includes data scientists will expect you understand how the algorithm works and have validated its results.

Related Insight projects:

1. AskedAgain by Kellie Lu
2. CopyCatch by Natalie Lebedova

Making a high-volume Apache Airflow architecture fault-tolerant

Overview

Apache Airflow, which is in use at companies such as Meetup, Airbnb and Wayfair, is typically deployed on one machine but for high-volume jobs, it may make sense to have multiple machines running different workflows. But if you are running multiple Airflow jobs, you need to designate one machine to act as the scheduler. But if that machine goes down, you may have to run the entire workflow again instead of picking up where it left off. Design and build a better alternative.

Engineering challenges

Building a distributed Airflow model will require you to make design choices, such as possibly maintaining a cache to keep info on ongoing workflows so that if a scheduler goes down, information about the workflows that have successfully completed or are in process are preserved. Else, perhaps you'll chose to keep a second scheduler on hand so that if the first one goes down, the second one can take over.

Possible approaches

1. Identify high-volume batch jobs that need to be run on a very frequent basis
2. Build an external database that can maintain information on completed workflows and in-progress one
3. Instead of one scheduler, have two schedulers running and keep them in sync using a heartbeat. If the first one goes down, switch to the second one
4. Use service discovery to dynamically change the number of workers that are running workflows at one time
5. Build monitoring and alerting services that senses when a job crashes and sends the relevant information (e.g., Spark job on worker X failed in task Y)

Functional highlights

Projects that come out of this seed will require the running of multiple and complex batch processing jobs. Identify a business use case with high-volume data needs that may benefit from a distributed Airflow architecture.

Presentation highlights

Airflow is a very popular tool and learning more about it in detail will allow you to showcase how you can be productive on Day 1 at many companies. Additionally, developing a novel approach

to a common problem in industry showcases your engineering skills. Thoroughly discussing the trade-offs and design choices made will be key to a successful presentation.

Productionizing a machine-learning platform

Overview

Machine learning models are great but there are many complexities around making them scale, managing different versions and then measuring their effectiveness. Build a platform where numerous client machines are submitting data that needs some processing before being run through a pre-trained model.

Engineering challenges

There are many architectural and engineering considerations to take into account and hitting all of them would take much more than what a three-week Insight project will entail so the key will be to make sure you have a minimum viable product and then iterate on it to build out additional functionality.

Possible approaches

Say your platform revolves around identifying objects in an image using a pre-trained model. Because you'll want to focus on your engineering challenges rather than training your model, pick a pre-trained model for your pipeline.

1. Ingest file locations of images in real-time from multiple sources
2. Pre-process the images using a stream processing engine to resize the image, or as a stretch goal normalize it perhaps using a package such as OpenCV
3. If you have time or if you want to skip the pre-processing step, concentrate on ways you can pass data to a pre-trained model that is running on multiple TensorFlow instances. Manage those instances using a load balancer. As a stretch goal, provide fault tolerance via cluster management, autoscaling and perhaps using Kubernetes or other container orchestration
4. Instead of (No. 3: focusing on managing the TensorFlow instances), turn your attention on how to maintain multiple versions of a model and track which version was used on what data. A stretch goal would be to automate this version control process perhaps by building a continuous integration/continuous deployment pipeline.
5. If you do use a machine learning model, it’s best to use a pre-trained one to cut down on the focus you would need to devote to training your model before you could use it in your project. But you still might want to gauge how good your model results are. Provide a way to collect user feedback on the results of your model and run analytics on how well the model is performing -- provide a way to flag to data scientists and let them know it’s time to tweak the model or do another round of training

Functional highlights

Keep in mind that the crux of this project is not the machine learning but the architectural decisions that you make to allow the machine learning models to run optimally at scale.

Presentation highlights

Any presentation that tackles machine learning also opens itself up to intense questioning by data scientist. Be sure that you are ready to address any questions about the model you choose, including the internals of how it was trained, its accuracy as you were able to measure it and tradeoffs you made.

Related Insight projects:

1. ElastiK Nearest Neighbors - by Alex Klibisz

Ingesting and joining of multiple large and diverse datasets

Overview

Joining multiple large datasets that are used to answer business questions is a common requirement at many companies. But it's not enough to obtain data and store it into a database. Companies are looking for a bit more sophistication.

Engineering challenges

Real data is often messy -- real messy. Figuring out ways to detangle and then validating the resulting data are correct can be a challenge. Also identifying a schema that can represent the data now and down the road if the producer of the data changes what the fields mean is often crucial to a data engineer's job. Finally, identifying the right tools for the job that can reliably get the data where it needs to be when it needs to get there also is highly desirable.

Possible approaches

1. Ingest multiple sets of messy data and create a schema that allows the data to be joined and would be flexible should the data or their fields change.
2. Include support for schema changes with the use of protocol buffers.
3. Use tools, such as Presto (distributed SQL Query engine popular at Facebook and other organizations) or Apache Drill, which is inspired by Google Dremel, to query the data.
4. Build tools to allow analysts to slice and dice the data in multiple flexible ways, which could also include the addition of a data warehouse, such as AWS Redshift
5. Incorporate a scheduler, such as Apache Airflow or Luigi, to ensure that batch jobs run on time and are successful. **If you are building a batch processing platform (i.e., projects inspired by this seed), this step is required.**

Functional highlights

Identifying appropriately large datasets and how to join them to answer questions will be key.

Presentation highlights

Finding the right dataset, identifying questions that can be asked and answering them will allow you to showcase your creativity, background and business sense. The details of the work, such as cleaning and validating data as part of a batch job and putting rules in place to account for

errors in the data or if external systems are inaccessible, also will highlight your attention to detail as a data engineer.

Related Insight projects:

1. Unified-Blockchains - by Dragos Velicanu

Working on an Open Source Ticket

Overview

When companies decide to use open source frameworks, they often need to make adjustments or improvements to suit their needs. Sometimes, companies merge these changes to move the open source project further as a whole and be an active member of the community.

Technical highlights

Contributing to an open source framework allows you to dive deep into a popular technology and become a domain expert. It also allows you to get experience in the process of working with the open source community and working with them to see your pull request get merged to the master branch. Finally, by pairing your project to a realistic use case, you demonstrate your ability to translate between business scenarios and technical concepts.

Possible approaches

1. Find an interesting open source ticket for a popular framework in the DE realm. It is important that the ticket is both suitable for an Insight project in scope and topic.
    a. Example: Kafka Data Loss Scenario
2. Find a business case relevant to the open source ticket. In order to showcase your work on the ticket, consider building a data pipeline. For instance, for the example above, build a simple ingestion platform using Kafka and simulate the data loss scenario. Building such a pipeline also allows you to think about a Plan B in case you struggle with the open source ticket. In the example mentioned above, this could mean that you have a strategy to resolve the data loss scenario in your use case without modifying Kafka's codebase.
3. Work on the open source ticket and submit a pull request. Try to engage with the community and be sure to follow software engineering best practices in your use of git, JIRA and the likes.
4. Deploy your modified framework in your example pipeline and demonstrate the new feature. In the example from above, deploy your own version of Kafka and demonstrate that the data loss scenario is now resolved.
5. Merge the pull request and become an official contributor to a popular open source framework.

How to evaluate open source tickets

Looking at open tickets can be overwhelming - we recommend you use the following questions as guidelines in evaluating an open source ticket:

- What is a good business use case for this ticket?
- Is the ticket related to a bug or a feature?
    - If it is a bug, is it reproducible? If it is not, you will not be able to effectively demo your contribution.
    - If it is a feature, is it related to Data Engineering?
        - A bad example would be a ticket related to improving error messages / logging. While crucial, this would most likely not allow you to showcase data engineering skills
- Is there a lot of activity regarding the ticket? You ideally want a ticket that many people are excited about, but no one is assigned to yet:
    - How many people are following it?
    - Is it assigned?
    - How long is it open for?

Functional highlights

Most companies use modified versions of popular open source frameworks or build new frameworks from ground up. If you can demonstrate that you solved a relevant open issue, companies will get a strong sense on your engineering and problem solving skills.

Presentation highlights

Identify a strong business case for your contribution - what company would need the feature you added or would suffer under the bug that you fixed? Explain your approach, your biggest technical challenge and your trade-offs clearly to showcase your ability to communicate complex matters. Lastly, convey your ability to work on a large project and work with the community by submitting a pull request and actively cooperating with the community.

Related Insight projects:

1. Fresh Taste, slides - by Ben St. Clair
2. Quadtree kNN - by Dan Blazevski
3. SparkSQL in trouble, slides - Bastian Haase