oyster.engineering /post/120026990383/data-at-oyster-part-i-infrastructure

Data at Oyster Part I: Infrastructure

5-6 minutes

Mengxi Lu

This is part one in a series of posts on building data-driven products at Oyster.

Our mission at Oyster is to connect readers with books they'll love. We do this by combining over 1 million books with great discovery features via social, editorial, and algorithmic recommendations to make it easy to browse and discover a great book.

Oyster has its own unique challenges with data processing. For example, we have user behavioral data coming from online and offline sessions, and book reading progress is hard to represent with immutable data structure. We also have multiple teams that rely on data analytics in their own ways. The product team, for instance, relies on event data to understand how readers interact with Oyster apps, while marketing needs data to track user growth and conversions.

In this blog post, we will share our approach to building a scalable and accessible data system to support these diverse needs.

The Backstory

The first version of the Oyster data platform was built on Amazon Elastic MapReduce. We ran our offline batch processing jobs using Scalding, and there were also multiple analytics jobs baked into the Django backend and managed by Celery.

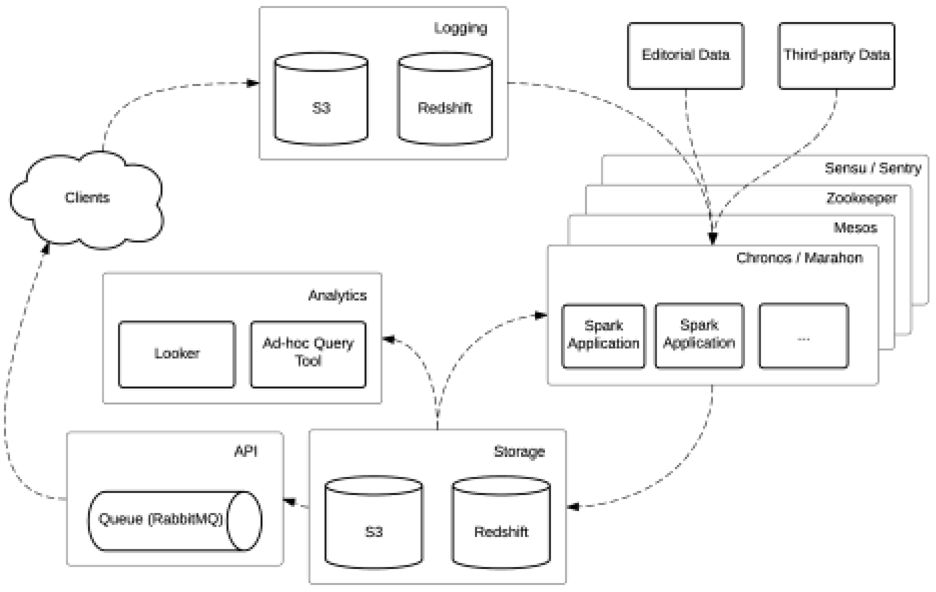
This worked fine in the early days, because it's very easy to scale up/down and monitor jobs in progress through the AWS web console. However, EMR also had multiple disadvantages:

- It gets expensive quickly as your data grows.
- Performance is unpredictable.
- Limited customizability. If your application is consuming data sources other than S3, or you need to use an unsupported Hadoop add-on, you end up spending a lot of time creating workarounds.

The New Stack

Oyster has grown rapidly since launching in 2012, and we started to hit a lot of limits with our original system. To support our growth, we migrated to a new data infrastructure with the following design decisions in mind:

- Reactive
- KISS. We want our data pipeline to start small and simple, but be able to scale over time.
- Easy access. We introduced business intelligence tools to enable non-developers to answer their own questions about our data.



The Datacenter Kernel: Mesos

Apache Mesos is a fault-tolerant cluster manager and the cornerstone of our elastic compute infrastructure. While Hadoop presents challenges in parallel scale-out, Mesos provides efficient resource isolation and sharing across distributed applications or frameworks. It bridges the gap between hardware and applications (or frameworks, in the context of Mesos), and solves the issue of elasticity and virtualization for large-scale deployments.

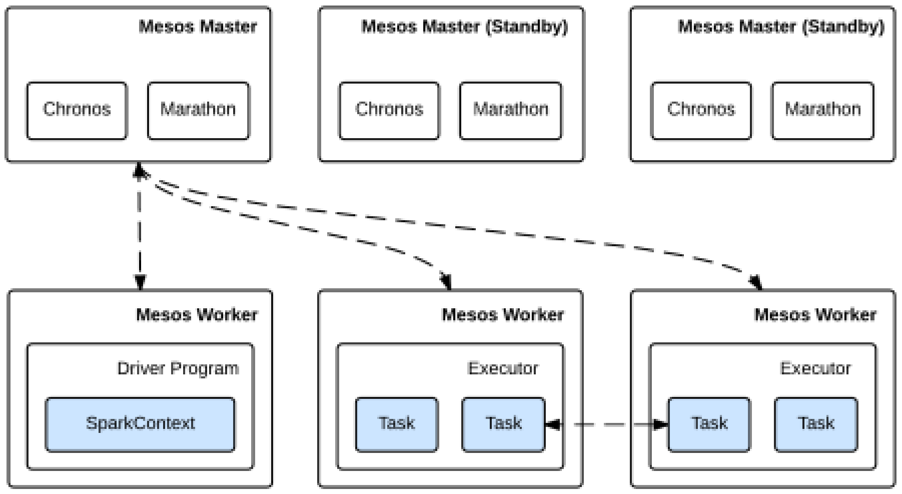
We have used Mesos in production for several months now and are very satisfied with its performance. We are able to dynamically provision and add new workers to the pool without changing any cluster configurations, and our developers have greater control over deployment, without worrying about resource allocation.

ETL Jobs as Data Applications

We love Spark here at Oyster because it offers a simple programming abstraction that provides powerful parallel computing and persistence capabilities. It can handle batch, interactive, and real-time data processing (Spark Streaming) in one framework. Spark also comes with MLlib, a library of efficient implementations of many popular algorithms directly built using Spark.

Spark/MLlib is our primary data processing engine. We use it for book recommendations, personalizing the Home experience, and many other ETL jobs. Thanks to Mesos, we don't need to maintain a dedicated Spark master/slave cluster. Instead, we encapsulate each ETL job into isolated Spark applications and run them on the Mesos cluster alongside other frameworks.

To achieve this, we package all dependencies of an ETL job into an uber jar using the sbt-assembly plugin, ship it to all Mesos workers, and then execute the ETL jobs just like regular Java applications. This pattern enables us to rapidly develop and deploy data applications. For datasets that need further downstream processing by other services, we serialize the output using Avro and use RabbitMQ to notify other systems that the data are ready to go (we will provide a detailed look into this notification system in the second part of the series).



For ETL job scheduling, we use Chronos, a distributed and fault-tolerant task scheduler which also runs on top of Mesos. Chronos uses Mesos' primitives for storing state and distributing work, and it provides a RESTful API over HTTP, meaning you can quickly delete all tasks for a job if it gets stuck.

Business Intelligence

Every day we collect and aggregate hundreds of metrics to understand our performance and how readers engage with Oyster. Looker helps us look across these data in a more meaningful way. We connect it to Redshift, our primary analytics data store, and use its expressive model language to create views for business operations, user engagement, growth/retention, and more. Different teams leverage these data and visualizations to make more informed product decisions.

At Oyster we are all passionate about data. If you have great ideas on building data infrastructure in a fast changing environment, come join us!