Migration Spark to Beam with hexagonal architecture and DDD

By Mazlum TOSUN

Group Bees
About me

Mazlum TOSUN

- Head of data and co-founder at Group Bees
- Tech lead GCP and data
- Passionate about Google Cloud, data, craft and functional programming
- Fan

https://github.com/tosun-si
https://twitter.com/MazlumTosun3
https://www.linkedin.com/in/mazlum-tosun-900b1812/
Context

- My previous customer worked on GCP and had Spark/Dataproc batch jobs
- There was some issues with Spark jobs (Spark streaming on bucket and memory usage)
- Have the need to develop custom code connectors for GCP resources
- Our customer wanted to change batch jobs to streaming
The strategy was to be cloud native
Spark structured streaming was not compatible with Pub Sub
The team wanted to do a POC on Apache Beam and Dataflow
The team used to work with JVM languages (Scala, Java)
Firstly I did a POC for a datamart with SCIO (Scala wrapper on Beam by Spotify)

Why?

- Because the team used to work with Spark Scala

Interesting choice and it produced good code but finally the team decided using native Beam with Java sdk

The goal was to be near to the native SDK to be more confident, learn native code and have more documentation on the Web

Beam Java instead of Python
I was in charge to propose a Beam boilerplate code and architecture to the team.

The Spark code was mostly oriented with inheritance design and without code decoupling.

Our use cases had many transformations and business rules.

I proposed an hexagonal architecture and domain driven design.
Architecture

https://blog.octo.com/hexagonal-architecture-three-principles-and-an-implementation-example/
Advantages of hexagonal architecture and DDD

- Isolation of business domain part
- Isolation of infrastructure and technical part
- Better handling of code complexity (domain and infra layers separated)
- Code decoupling between domain and infrastructure part
- If technical part evolves, there is no impact on domain part
- The responsibilities are clear and the code can evolve easily
- Domain part can be easily tested separately with mocks on infra part
- Tests can be done with domain + infra
Dependency injection

- The dependency injection is a concept allowing the code decoupling with contracts and interfaces
- The IOC meaning delegation of object instantiation to a dedicated framework
- The concern of instantiation is not in the applicative code but separated to the framework (connection between interface and implementations)
Dependency injection

- There are many popular libraries or frameworks in the Java community:
  - Spring
  - CDI
  - Giuce
  - Dagger 2

- Some explanations for each of them
Dependency injection

- The choice was Dagger 2:
  - Dependency injection done at compile time
  - Existing Maven plugin for Dagger 2
  - Better performance
  - Maintained by Google
  - Flexible system with modules and components
Feedback after the migration

Feedback for developers used to work with Spark/Scala

Pros

- Beam is simple for JVM devs: only PCollections and transformations
- Can easily separate a composition of transformations with PTransforms
- Better support for streaming and same code between batch and streaming
- The compatibility with GCP is full, native IO, cloud logging...
- Dataflow runner autoscaling, metrics and monitoring allows devs to be more focus on the code logic
- Serveless and no cluster to manage
Feedback after the migration

**Cons**

- Beam Java is more verbose than Spark Scala
- With a bad use of lambda expression, the code can be less readable
Beam DDD code demo real application
Links to example projects

https://github.com/tosun-si/teams-league-java-ddd-beam-summit

https://github.com/tosun-si/teams-league-python-ddd-beam-summit
Thank you :)