ETL IN SCALA

MAX EPELBAUM

NORTHEAST SCALA 2019

GITHUB LINK: HTTPS://GITHUB.COM/EPPELS/NESCALA-ETL

AFTER THIS TALK YOU SHOULD...

- Understand the pattern for performing ETL that has worked pretty well at our firm
- Appreciate the Scala features that power this framework
- Be able to actually use this pattern and extend it for your needs if you think it would help your ETL use cases

CONTEXT

- I switched into a space where I started doing projects that had important ETL needs
- The existing solutions for ETL in that space were antiquated, error prone, and untestable
- Knew of some existing tools and patterns, but they were either not suitable for my use cases or too expensive to set up internally
- Knew I could stand something up quickly with Scala

APPROACH

- Key requirements
  - Connect to necessary input and destination data stores
  - Output in destination store must adhere to internal domain model
  - Make it cheap to do integration testing outside of production environment

- Started concrete and abstracted as new use cases came up
  - Initially one application that went SqlServer -> SqlServer
  - Now four separate apps extend the framework and have adapters for our internal non-sql cloud based database, S3, CSV files on network, Sftp, and REST endpoints

ARCHITECTURE

- We decided to name the framework **TowTruck**
- The **ITowTruckApp** implements the main method by running a given **TowTruckRunMode**
- A **TowTruckRunMode** is a template for specifying what **Jobs** to run
- A **Job** is a singular ETL task
- The **TowTruckJob** trait is abstracted over any data source
- **JobTemplates** extend **TowTruckJob** and provide enriched interfaces for particular data source types
- **Concrete Jobs** extend JobTemplates

APPS AND RUNMODES

- Abstracted over the **App** to handle different ETL use cases that need to be built and deployed as separate executables
- Within an app have different **RunModes** to handle wanting to run different collections of **jobs** at different times
  - Example: Run all jobs in the morning, but in afternoon only run the subset of the jobs that handle faster moving data for better runtime performance

JOBS AND JOB TEMPLATES

- The abstract **TowTruckJob** handles the running of any job
  - Ensures that failure in one job doesn't impact others
  - Separates the **extract** from the **load**, allowing checks in between
  - Not directly extended with a concrete job
- **Concrete Jobs** extend **JobTemplates**
  - **JobTemplates** are specific to types of data stores and implement functionality specific to those stores for you
  - This is also where additional type safety is introduced
- Most active development is in the creation of new **job templates** and **concrete jobs**

CODE WALKTHROUGH

- Code hosted on: https://github.com/eppels/nescala-etl.

- Feel free to clone and follow along

- If you prefer writing code over watching me talk through it, the readme includes some ideas for how to extend the demo

MISSING FROM THE DEMO

- Patterns for abstracting over the input data sources
- Full set of job templates
- In code unit tests and how integration tests would work

CONTACT INFO FOR QUESTIONS

- This presentation and all the code are on GitHub
  - https://github.com/eppels/nescala-etl
- Email Max.Epelbaum@bwater.com