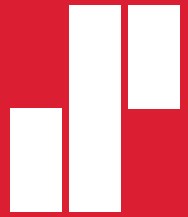


Clojure in Analytics pipeline at DNA

Kimmo Koskinen - Solita, Juha Niiranen - DNA
Clojure Meetup 2019-04-29

SOLITA

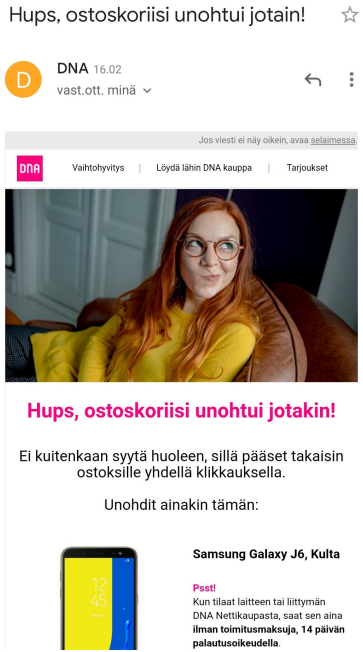
Mandatory table of contents

- Project background
- Few chosen topics
  - Event pipeline
  - Extract Transform Load (ETL)
  - Apache Spark and Clojure
  - Utilities
  - Kiikari
  - Evolution of the platform
- Notes on future
  - Closing

Project background: Analytics and ML at DNA

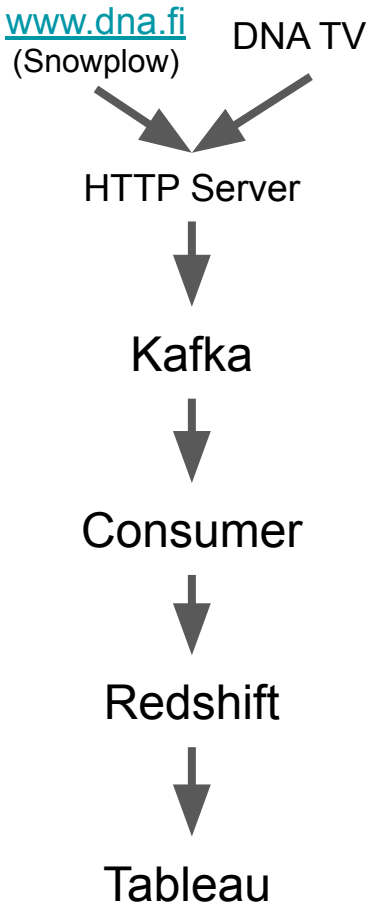
- Started in summer of 2015
  - Part of a bigger program
- Vision: Unified customer experience
  - Data all over the company
  - Not centralized view of the customer
  - No personalized customer experience
- Strategy
  - Build a platform for centralized customer view
- This team
  - Builds and maintains a platform for ML based analytics
  - 4-5 Data Scientists + PO
  - 3 Data Engineers (strong software dev focus)

GrandOne winner in 2017



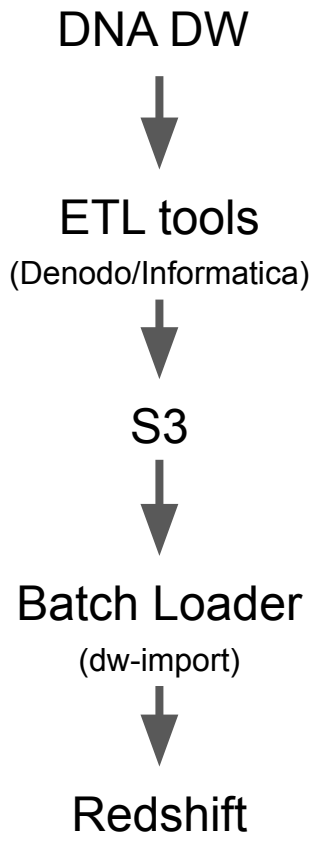
Event pipeline

- Collect user event data in real time for analytics
- Kafka producer
  - Component, Ring/Compojure, Duct
    - Kudos to James Reeves
- Kafka consumer
  - clj-kafka
    - Transducers for data transform
  - overtone/at-at background tasks
    - Copy data to Redshift
    - Post-processing SQL scripts
  - migratus for migrations



ETL: Redshift batch loads

- Initially a AWS Lambda
  - uswitch/lambada
- Until 5 minute timeout started to be a problem
  - Move to EC2 as uberjar
    - Duct, Component, Amazonica
  - S3 events to SQS
  - Scheduled polling of SQS queue
- SQL scripts run after load (ELT)



ETL: AWS Lambda, runtime for data ingestion

- Poll
  - Triggered by Cloudwatch schedule
  - External S3, SOAP, HTTP APIs, SFTP
  - XML, JSON, CSV
- Receive
  - Triggered by S3 event
  - Use SQS when workload management is needed
  - JSON, XML, Parquet tarballs
- ETL FTW!
  - Fetch/Receive data
  - Transform – Clojure fits here well
  - Load – Write to S3, issue load to Redshift

ETL: AWS ECS, when Lambda timeouts

- Mainly pollers
  - Scheduled via Cloudwatch
  - Initially couple of EC2s, now on Fargate
- Similar otherwise
  - Extract, Transform, Load
  - Regular data transform in Clojure code
    - Some clojure.spec for validation

ETL: Redshift, the beast

- How to test ETL?
- Cannot run the DB locally
- SQL dialect specific for parallelism
- Interaction with S3 is the core
  - Mocking this out would not capture Joins in SQL

Extract & Transform

```
UNLOAD (‘sql with bunch of joins’)
TO ‘s3://.../prefix’
IAM_ROLE ‘...’
```

Load

```
COPY my_table
FROM ‘s3://.../prefix’
IAM_ROLE ‘...’
opts
```

ETL: Testing with Redshift

- Coverage
  - Real Database
  - Real S3
- Via library: clojure.test fixture
  - Create a temporary database for duration of test
  - Initialize with selected tables

ETL: Testing inside Redshift

```
(defn with-redshift-db
  "Fixture function that runs test in a temporary database.
  Binds commons-test.core/*test-db* (jdbc-spec with active connection)
  for duration of the test to a temporary database.

  Use like this:

  (use-fixtures :once with-redshift-db)
  (deftest read-table
    (jdbc/query commons-test.core/*test-db* [\"select 1\"]))

  Optionally takes options map with keys:

  :superuser-statements - Run statements that require superuser rights onto test database
                          (like 'grant usage on language...')
  :db-name - Use persistent db, creating it if not present, but not dropped at end of fixture
  :schema-snapshot - Map with keys
    :version schema version identifier or :latest (defaults to latest)
    :tables set of table names (i.e. #{\"events\"}) for populating database with these tables
    :env environment of the schema, defaults to :prod"
  ([fun]
   (with-redshift-db {} fun))
  ([{:keys [superuser-statements db-name schema-snapshot]} fun]))
```

Apache Spark and Clojure

- First Machine Learning application: Personalized recommendations
- Apache Spark was chosen
  - Resilient Distributed Datasets (RDD)
  - Spark SQL
  - MLlib
  - Collaborative Filtering: Alternating Least Squares (ALS)
- Initially version written in Scala
  - Using RDD based MLlib
  - Scala meh :)
- Idea: Spark SQL + Clojure!
  - Lean on Java API

Project Owner approves!

Me on paternity leave,
too much interesting stuff
going on at work

Apache Spark and Clojure: Status back then

- Flambo, Sparkling
  - 1.5 Spark
    - We want 2.x
  - Based on RDD
    - We want Spark SQL API
- Powderkeg
  - Nice REPL support, but on 1.5, lacks Spark SQL

Apache Spark and Clojure: Dataset wrapper

- Apply Clojure functions to Datasets

```
(defn map
  "Returns a new Dataset that contains the result of applying `func`
  to each element."
  [^Dataset dataset func ^ExpressionEncoder enc]
  (let [mf (reify MapFunction (call [this value] (func value)))]
    (.map dataset mf enc)))
```

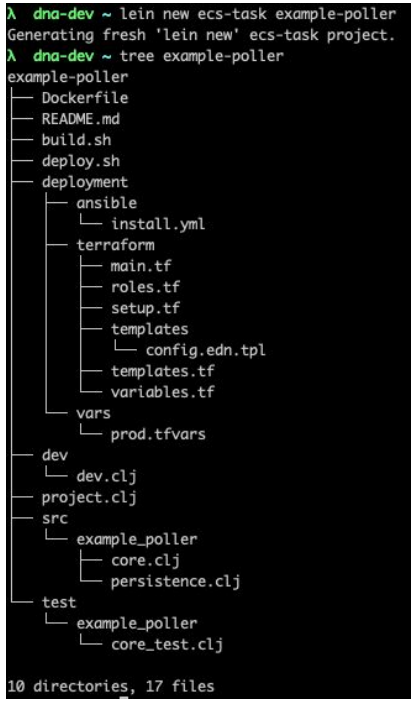
Apache Spark and Clojure

- Parallel Redshift writes via s3

```
(defn write-redshift-results [recommendations actuals storage topic conf]
  (let [accum (-> recommendations .sparkSession .sparkContext .longAccumulator)
        ...
        results (make-redshift-results recommendations actuals topic) ]
    (persistence/redshift-truncate (:redshift conf) table-name)
    (-> results
        (ds/foreach-partition
         (fn [rows]
           (let [key (str key-prefix "/" (UUID/randomUUID) ".json.gz")
                 fname (str "tmp/" key)]
             (persistence/write-rows-to-json-gz (map #(row->map % cols) rows) fname accum)
             (persistence/upload-to-s3 bucket key fname conf)))))
    (persistence/redshift-copy (str bucket "/" key-prefix) table-name conf)
    (timbre/debugf "Wrote %s rows to Redshift for topic %s."
                   (.value accum)
                   (:name topic))))
```

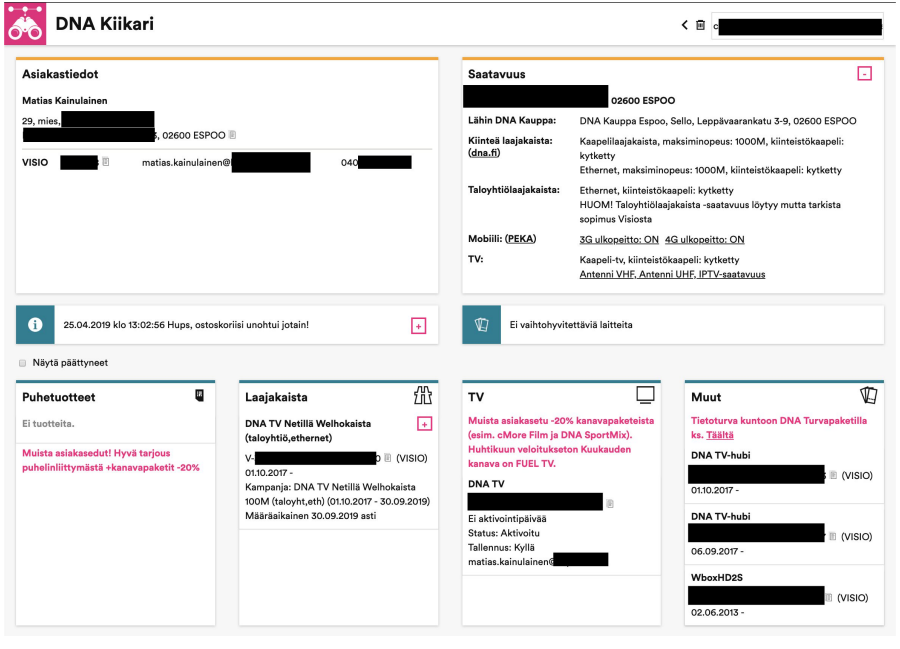
Utilities: Leiningen templates

- lein new ml-model
  - Generate scaffolding for AWS Batch task (Terraform, Ansible, Dockerfile, etc.)
- lein new lambda
  - Cloudwatch scheduled Lambda task
- lein new ecs-task
  - Cloudwatch scheduled ECS task



Kiikari

- Customer service application
  - Full view of the customer
  - Recommendations and personalized tips based on machine learning models and product data
- Clojure + Clojurescript, Full stack
  - Reagent/carry
  - Duct (0.8, component)
  - sharetribe/aws-sig4 (Elasticsearch)
  - reloaded.repl, duct/figwheel-component
    - Start dev via single `lein repl`
    - Also start local IDP (SAML)
- Elasticsearch bulk loader
  - core.async, throttler (HTTP 429)



Evolution

- Kafka -> Kinesis
  - Kinesis Data + Firehose Delivery Streams
    - Transformations in Lambda
- Ansible -> Terraform + Ansible
- Default VPC -> Corporate Transit VPC

Closing: Future

- MXNet
  - Clojure bindings
- Real-time recommendations
  - Feedback on how recommendations work
  - Adapt during session

Closing extra: Numbers (November 2018)

- 12 git repositories
  - 11 257 commits (to main repo master branch)
  - 595 pull requests closed (in main repo)
- 69 Jenkins jobs (excluding deploy jobs)
- 3 AWS environments
  - 26 EC2 instances, 20 ECS tasks, 22 Lambda functions, ~1.3TB data in Redshift, a lot of other AWS services (in production environment)
- ~25K LOC in Clojure, ~2K LOC in ClojureScript, ~10K LOC in Python, ~2K LOC in R
- ~50K LOC in SQL
- ~20K LOC in YAML (Ansible), ~12K LOC in HCL (Terraform)

Thanks!

PS. we are hiring a senior developer to our team! :)