

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

Hups, ostoskoriisi unohtui jotain! ☆

DNA 16.02
vast.ott. minä ▾

Jos viesti ei näy oikein, avaa [selaimes](#)

DNA Vaihdohyvitys Löydä lähin DNA kauppa Tarjoukset



Hups, ostoskoriisi unohtui jotakin!

Ei kuitenkaan syytä huoleen, sillä pääset takaisin ostoksille yhdellä klikkauksella.

Unohdit ainakin tämän:

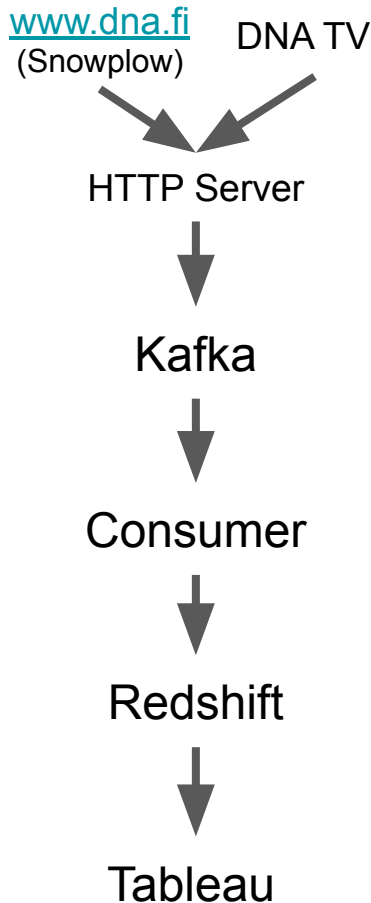


Samsung Galaxy J6, Kulta

Past!
Kun tilaat laitteen tai liittymän DNA Nettikaupasta, saat sen aina ilman toimitusmaksuja, 14 päivän palautusoikeudella.

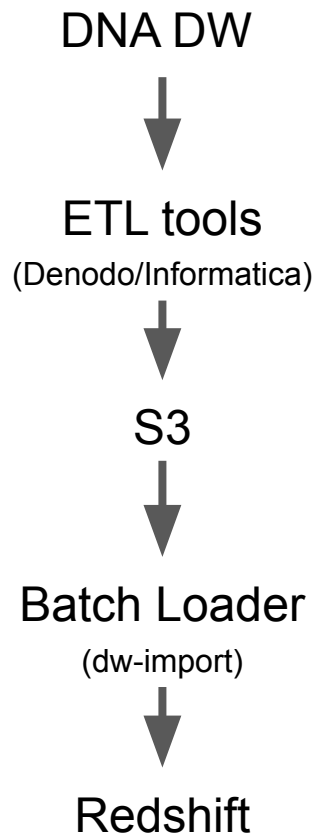
Event pipeline

- 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:

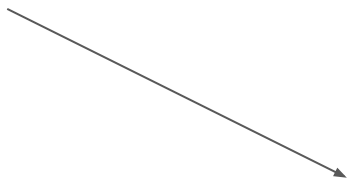
```
:superuser-statements - Run statements that require superuser rights onto test database
                        (like 'grant usage on language...')
:db-name - Use persistent db, creating it if not present, but not dropped at end of fixture
:schema-snapshot - Map with keys
  :version schema version 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



Jarno Kartela

@JKartela

Following



now starting to move recommendation engine to clojure w/ flambo in hopes of easier data wrangling & simplicity #clojure ping @KimmoKoskinen

10:42 AM - 27 Feb 2017

2 Likes



3



2



Kimmo Koskinen @KimmoKoskinen · 27 Feb 2017



Replying to @JKartela

Wow! wait wait wait, I'm still with the kids for one month :) #Clojure + #DataScience



1



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

```
λ dna-dev ~ lein new ecs-task example-poller
Generating fresh 'lein new' ecs-task project.
λ dna-dev ~ tree example-poller
example-poller
├── Dockerfile
├── README.md
├── build.sh
├── deploy.sh
├── deployment
├── ansible
│   └── install.yml
├── terraform
│   ├── main.tf
│   ├── roles.tf
│   ├── setup.tf
│   ├── templates
│   │   └── config.edn.tpl
│   ├── templates.tf
│   └── variables.tf
├── vars
│   └── prod.tfvars
├── dev
│   └── dev.clj
├── project.clj
├── src
│   ├── example_poller
│   │   ├── core.clj
│   │   └── persistence.clj
├── test
│   └── example_poller
│       └── core_test.clj
└── 10 directories, 17 files
```


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)

The screenshot displays the DNA Kiikari web application. At the top, the header shows the DNA logo and the name 'DNA Kiikari'. Below the header, the main content area is divided into several sections:

- Asiakastiedot** (Customer Information): Displays the name 'Matias Kainulainen', address '29, mies, [redacted], 02600 ESPOO', and contact details 'VISIO [redacted] matias.kainulainen@[redacted] 040 [redacted]'.
- Saatavuus** (Availability): Shows the address '02600 ESPOO' and details about the service, including 'Lähin DNA Kauppa: DNA Kauppa Espoo, Sello, Leppävaarankatu 3-9, 02600 ESPOO'. It also lists 'Kiinteä laajakaista: (dna.fi)' and 'Taloyhtiölaajakaista: Ethernet, kiinteistökaapeli: kytetty'.
- Puhutuotteet** (Products): A section for products, currently showing 'Ei tuotteita.' (No products).
- Laajakaista** (Broadband): Details about the broadband service, including 'DNA TV Netillä Welhokaista (taloyhtiö, ethernet)' and 'Kampanja: DNA TV Netillä Welhokaista 100M (taloyhtiö, eth) (01.10.2017 - 30.09.2019)'.
- TV**: Information about the TV service, including 'DNA TV' and 'Ei aktivointipäivää'.
- Muut** (Other): A section for other services, including 'Tietoturva kuntoon DNA Turvapaketillä' and 'DNA TV-hubi'.

The interface is clean and modern, with a white background and blue accents. It uses a grid layout to organize the information.

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! :)