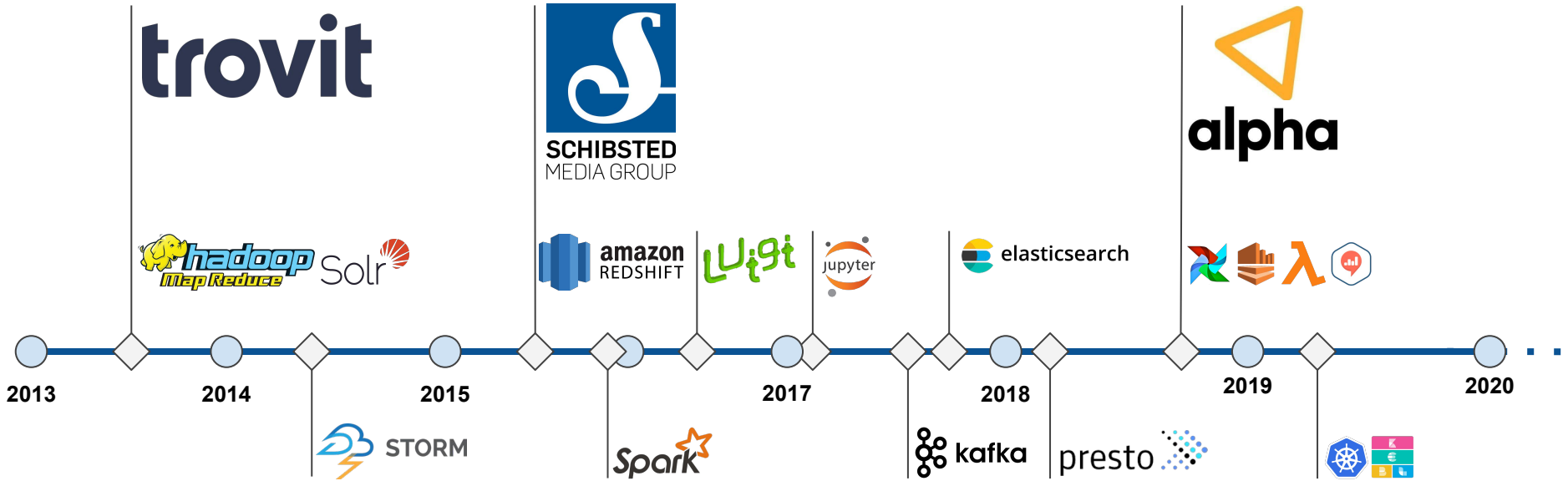
VEA: Validating, Evolving & Anonymizing Data in Real Time

Albert Franzi Cros, Data Engineer | Alpha Health

Slides available in:

bit.ly/afranzi-vea

About me


trovit
SCHIBSTED
MEDIA GROUP
alpha
hadoop
MapReduce
Solr
amazon
REDSHIFT
Luigi
jupyter
elasticsearch
2013
2014
2015
2017
2018
2019
2020
STORM
Spark
kafka
presto

VEA: Validating, Evolving & Anonymizing Data in Real Time



- Alpha Health Challenge
- Introducing VEA
- Data Validation
- Data Evolution & Anonymization
- Learnings

Alpha Health Challenge

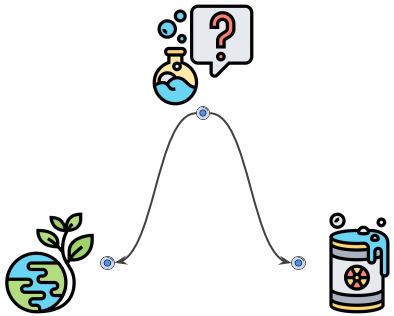
Introducing VEA

Data Validation

Data Evolution & Anonymization

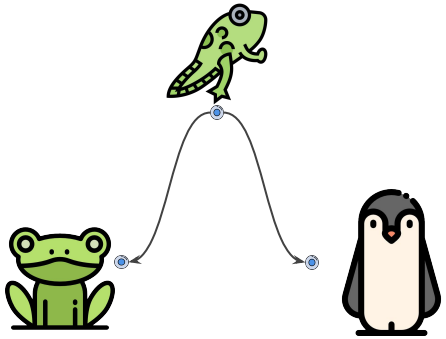
Learnings

Improve health quality



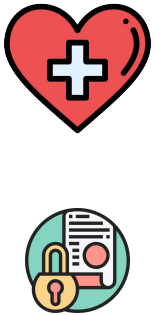
Good data quality

Prototyping



Data Evolve

Health records



Sensitive data

Alpha Health Challenge

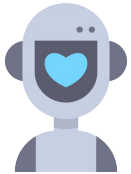
Introducing VEA

Data Validation

Data Evolution & Anonymization

Learnings

Introducing VEA



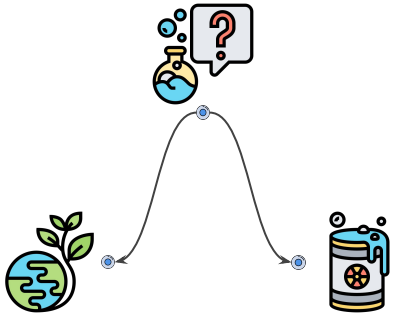
Validate

Evolve

Anonymize

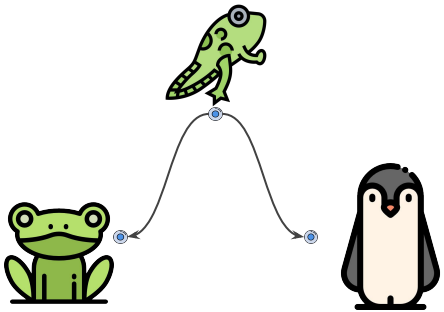
Introducing VEA

Improve health quality



Validate

Prototyping



Evolve

Health records

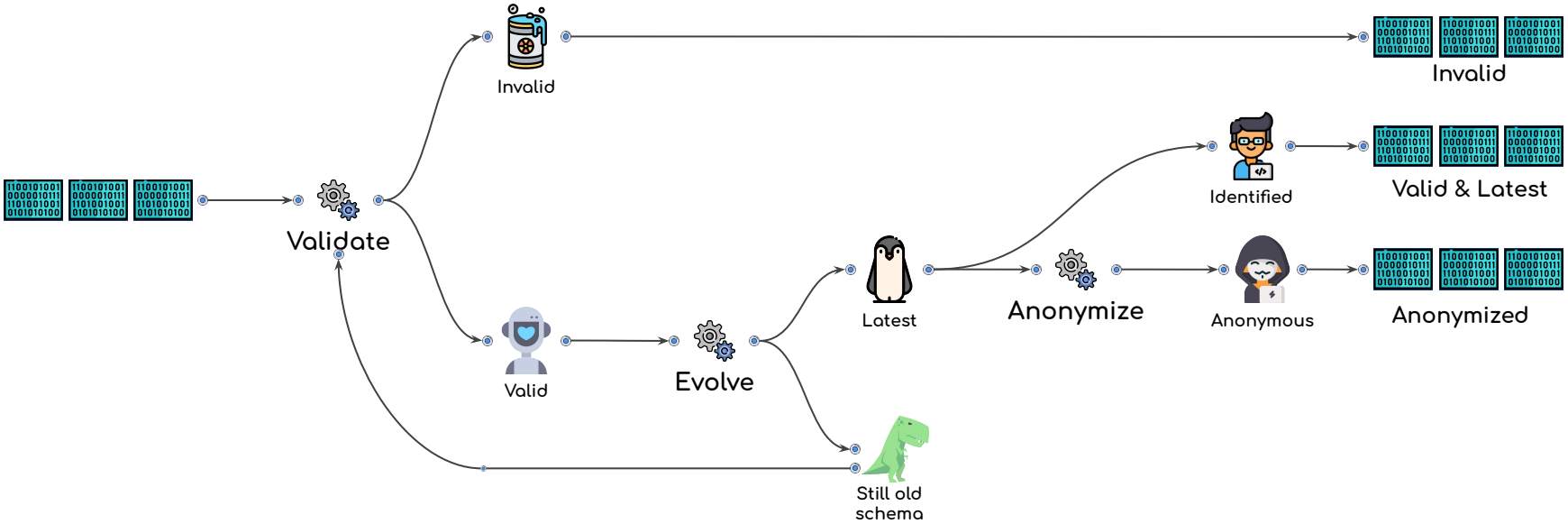
Anonymize

Introducing VEA Lambda

Introducing VEA Lambda

It's better to isolate wrong events than end up having a zombie data apocalypse where data cannot be consumed.

Introducing VEA Inside the λ



Introducing VEA Storage layers

Color as Privacy

Non-user data e.g. weather, aggregated stats, etc...

Anonymized user data.

Identified user data.

Raw Data, as it comes from the origin.

Retention periods per color

Access policy per color, role & user

Alpha Health Challenge

Introducing VEA

Data Validation

Data Evolution & Anonymization

Learnings

Data Validation Schemas

What is a Schema?

It's the DNA of the data it defines

Data Structure

Data Quality

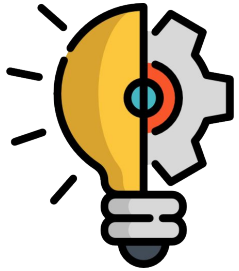
Data Format

Data Content

A proper schema helps us to have a **better understanding of our data.**

A clear understanding of our data allows us to create **better products for our users.**

Data Validation JSON schemas



In **Alpha Health**, we use the JSON-schema.org standard since it brings us the advantage of describing our existing data formats by providing a clear **human** and **machine-readable documentation**.

Validates data by using an **automated testing tool** (*i.e Github // everit-org // json-schema*) that guarantees the **quality** of the data ingested in our system.

Data Validation Schema model

base-event

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "/schemas/events/base-event/1.json",
  "description": "Base schema for all user-generated events (on device)",
  "properties": {
    "user": {
      "description": "User information",
      "$ref": "/schemas/objects/User/1.json"
    },
    "product": {
      "description": "Product information",
      "$ref": "/schemas/objects/Product/1.json"
    },
    "deploymentEnv": {
      "description": "Deployment environment in use",
      "enum": ["dev", "test", "stage", "prod"]
    },
    "createdAt": {
      "description": "Timestamp when the event was generate (following rfc 3339 format)",
      "type": "string",
      "format": "date-time"
    },
    "schema": {
      "description": "Name of the schema to validate against",
      "type": "string"
    },
    "source": {
      "description": "Source of the data point",
      "type": "string",
      "enum": ["analytics", "questionnaire", "sensor"]
    }
  },
  "required": ["source", "schema", "product", "deploymentEnv", "createdAt"],
  "type": "object"
}
```

Data Validation Schema model

base-device-event

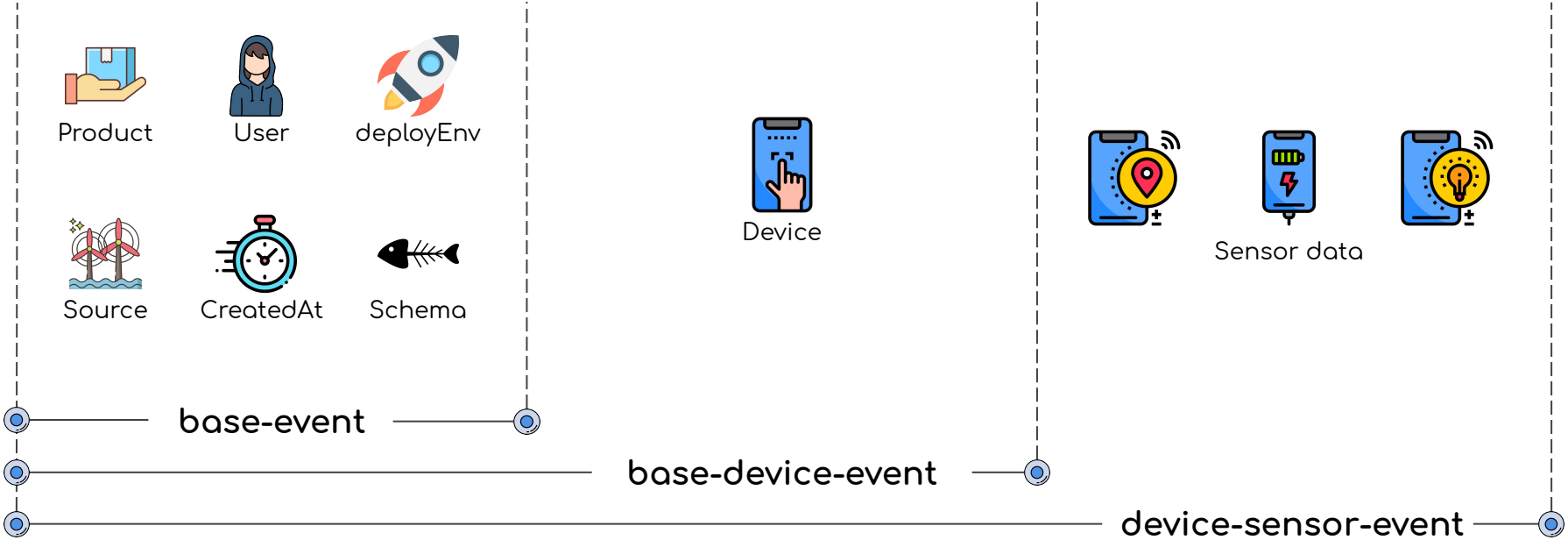
```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "/schemas/events/base-device-event/1.json",
  "additionalProperties": true,
  "allOf": [{"$ref": "/schemas/events/base-event/1.json"}],
  "description": "Base schema for all user-generated events (on device).",
  "properties": {
    "device": {
      "description": "Device information",
      "$ref": "/schemas/objects/Device/1.json"
    }
  },
  "required": ["device"]
}
```

Data Validation Schema model

device-sensor-event

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "/schemas/events/device-sensor-event/1.json",
  "allOf": [{ "$ref": "/schemas/events/base-device-event/1.json”}],
  "description": "User event including sensor data",
  "properties": {
    "data": { "oneOf": [
        {"$ref": "/schemas/objects/sensors/SensorAccelerometer/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorActivity/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorBattery/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorDevice/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorLight/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorMagnetometer/1.json"},
        ...
        {"$ref": "/schemas/objects/sensors/SensorPedometer/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorProximity/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorScreen/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorUnlock/1.json"},
        {"$ref": "/schemas/objects/sensors/SensorWalk/1.json"}
      ]}
  },
  "required": ["data", "device", "product", "user”]
}
```

Data Validation Schema inheritance



Data Validation JSON-Validator 🤖

```
def buildSchema(schema: JSONObject): Schema = {
  SchemaLoader.builder()
    .schemaJson(schema)
    .schemaClient(new ResourceSchemaClient)
    .draftV7Support()
    .useDefaults(true)
    .build()
    .load()
    .build()
}
```

Data Validation JSON-Validator 🤖

```
def validateEvent(schema: Schema, event: JSONObject): ValidationResult = {
  val validationListener: SchemaValidationListener = SchemaValidationListener()
  val validator: Validator = Validator
    .builder
    //.failEarly()
    .withListener(validationListener)
    .build()

  validator.performValidation(schema, event)

  val schemasReferenced: Seq[SchemaReferenced] = validationListener
    .schemasReferencedMatching

  ValidationResult(event, schemasReferenced)
}
```

Data Validation Validator Listener 🤖

github.com/everit-org/json-schema # ValidationListeners

#242 - PR done by Alpha Health to include the validation Listeners.

ValidationListeners can serve the purpose of resolving ambiguity about *how* does an instance JSON match (or does not match) against a schema. You can attach a ValidationListener implementation to the validator to receive event notifications about intermediate success/failure results.

Data Validation Validator Listener

```
class SchemaValidationListener() extends ValidationListener {
  val schemasReferencedMatching: ListBuffer[SchemaReferenced] = ListBuffer.empty

  override def schemaReferenced(event: SchemaReferencedEvent): Unit = {
    val subSchema: Schema = event.getReferredSchema
    val schemaReferenced = Option(subSchema.getId).getOrElse(subSchema.getSchemaLocation)
    val path = event.getPath
    val reference = SchemaReferenced(path, schemaReferenced)
    schemasReferencedMatching.append(reference)
  }

  override def combinedSchemaMatch(event: CombinedSchemaMatchEvent): Unit = {
    val subSchema: Schema = event.getSubSchema
    val path = event.getPath
    extractSchemaReferenced(subSchema).foreach { schemaId =>
        val reference = SchemaReferenced(path, schemaId)
        schemasReferencedMatching.append(reference)
    }
  }
}
```

Data Validation Validator Listener 🤖

```
val schemasReferenced: Seq[SchemaReferenced] = Seq(
      SchemaReferenced("#", "/schemas/events/base-event/1.json"),
      SchemaReferenced("#", "/schemas/events/base-device-event/1.json"),
      SchemaReferenced("#", "/schemas/events/device-sensor-event/1.json"),
      SchemaReferenced("#/data", "/schemas/objects/sensors/SensorWifi/1.json"),
      SchemaReferenced("#/data/scan/[0]", "/schemas/objects/sensors/WifiConnection/1.json"),
      SchemaReferenced("#/data/scan/[1]", "/schemas/objects/sensors/WifiConnection/1.json"),
      SchemaReferenced("#/device", "/schemas/objects/Device/1.json"),
      SchemaReferenced("#/product", "/schemas/objects/Product/3.json"),
      SchemaReferenced("#/user", "/schemas/objects/User/2.json")
)
```

Alpha Health Challenge

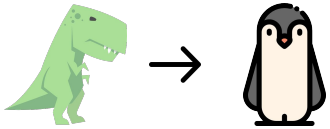
Introducing VEA

Data Validation

Data Evolution & Anonymization

Learnings

Data Evolution & Anonymization



"Evolving data allows us to keep up our development pace without worrying about older data versions."

"GDPR by design allows to keep up developing products on top of anonymized data without having nightmares with lawyers."

Data Evolution & Anonymization JSLT

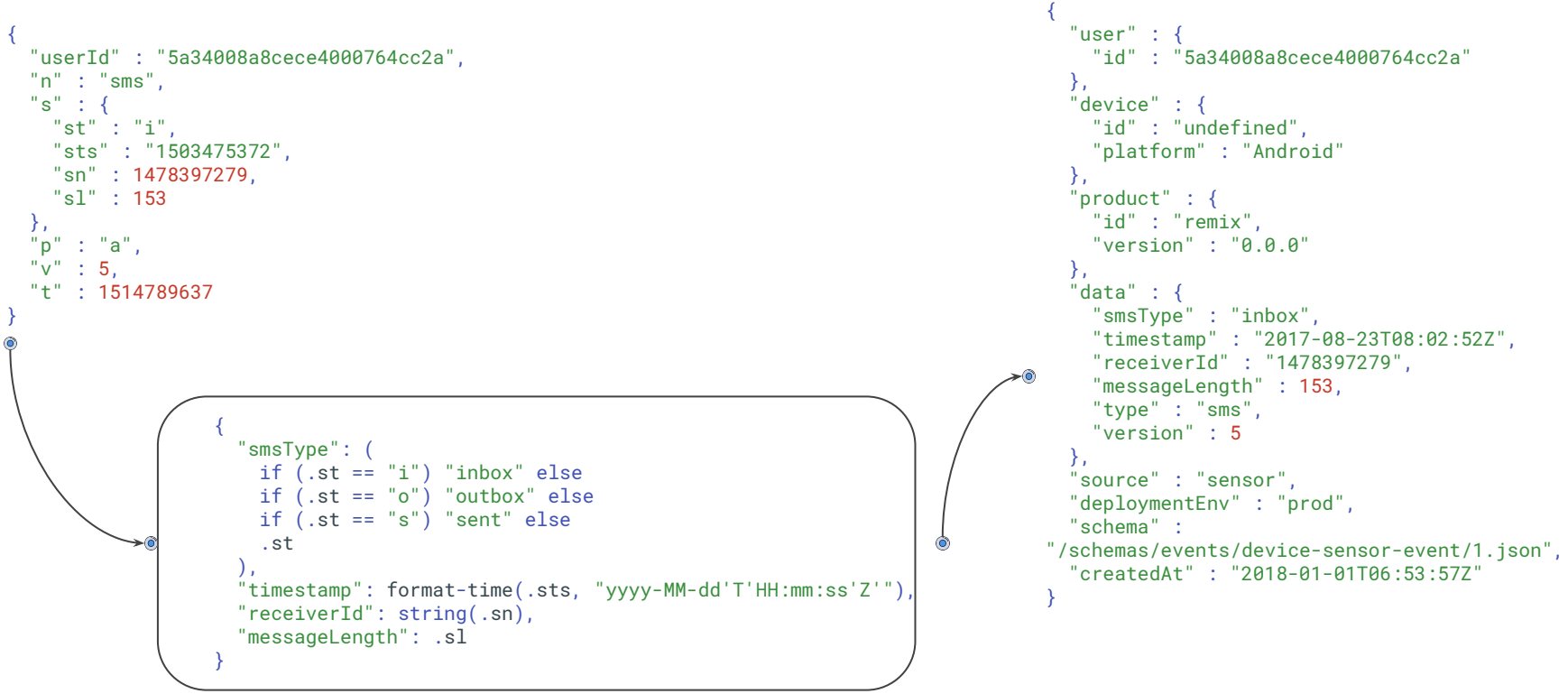
github.com/schibsted/jslt

JSLT is a complete query and transformation language for JSON inspired by jq, XPath, and XQuery.

JSLT can be used as:

- a query language to extract values from JSON (.foo.bar[0]),
- a filter/check language to test JSON objects (starts-with(.foo.bar[0], "http://")) ,
- a transformation language to convert between JSON formats.

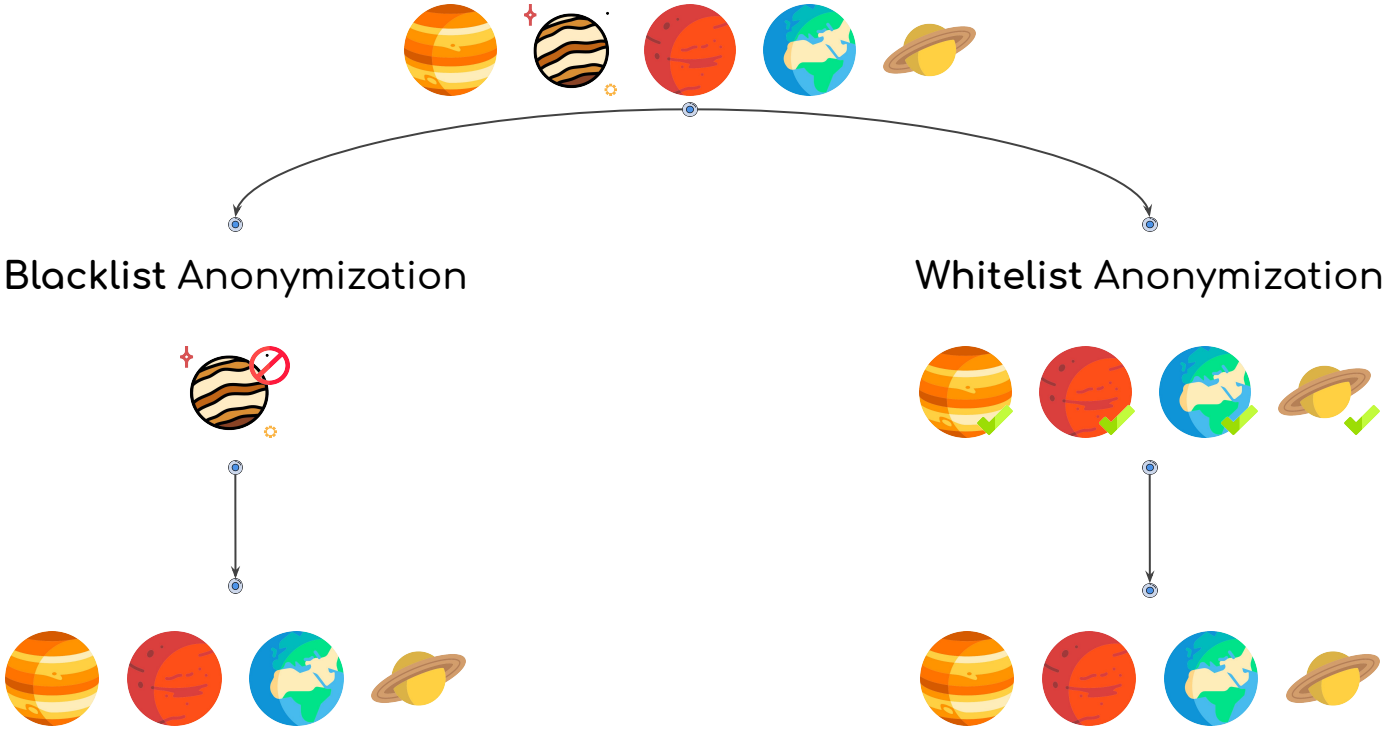
Data Evolution JSLT



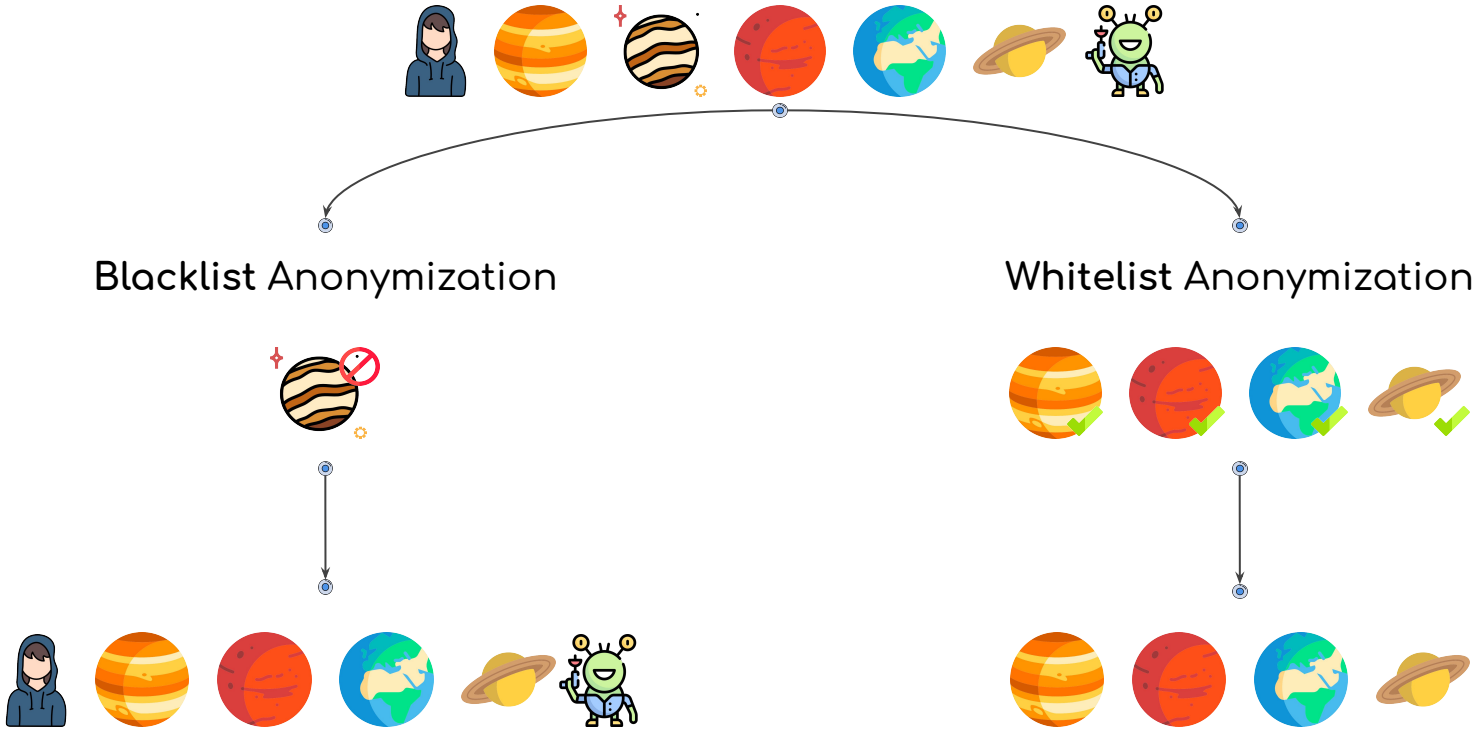
Data Evolution

```
def evolve(event: ValidationResult): EvolutionResult = {
  val schemasReferenced: Seq[SchemaReferenced] = event.schemasReferenced
  val json = event.json

  val schemasToEvolve = schemasReferenced
    .filter { case SchemaReferenced(_, schemaRef) => hasEvolution(schemaRef) }

  val eventEvolved = schemasToEvolve
    .foldLeft(json) {
      case (jsonEvent: JsonNode, SchemaReferenced(location, schemaRef)) =>
        val evolutionExpr = buildEvolutionExpr(location, schemaRef)
        val expr: Expression = Parser.compileString(evolutionExpr)
        expr(jsonEvent)
    }

  EvolutionResult(json,eventEvolved,schemasReferenced,schemasToEvolve)
}
```

Data Anonymization



Data Anonymization



Alpha Health Challenge

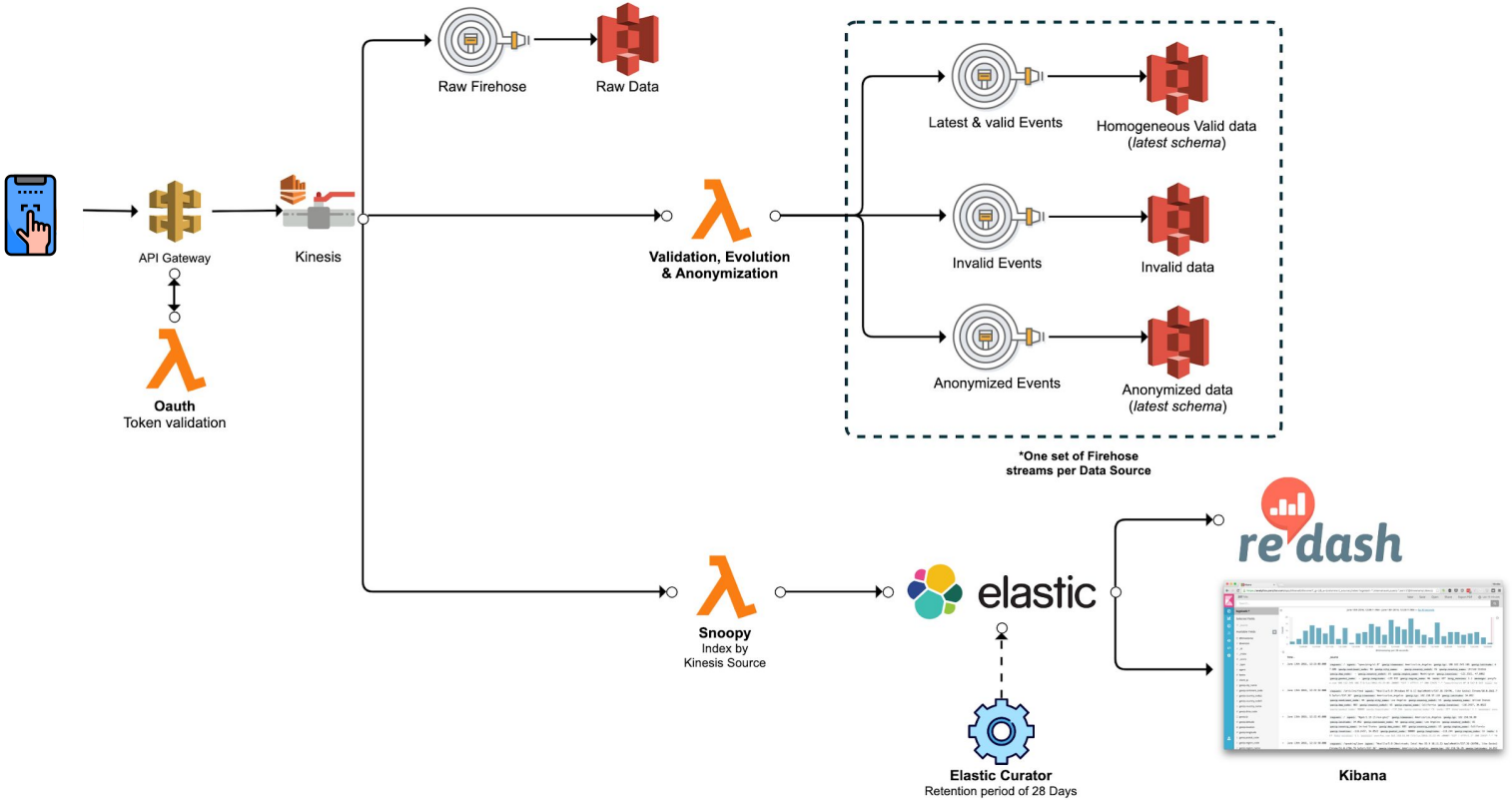
Introducing VEA

Data Validation

Data Evolution & Anonymization

Learnings

Learnings Infrastructure



Caching the Schemas - Google Guava # CachesExplained

```
import com.google.common.cache.{CacheBuilder, CacheLoader, LoadingCache}

lazy val schemaCache: LoadingCache[String, Schema] = CacheBuilder
  .newBuilder
  .maximumSize(MaximumCacheSize)
  .expireAfterAccess(CacheMinutes, TimeUnit.MINUTES)
  .build(new CacheLoader[String, Schema]() {
    override def load(schemaRef: String): Schema = {
      loadSchema(schemaRef)
    }
  })

  ...

  val schema: Schema = schemaCache.get(schemaRef)

  ...
```

Learnings Guidelines

JSON Guidelines

Must: Property names must be ASCII camelCase

Should: Define Maps Using **additionalProperties**

Should: Array names should be pluralized

Must: Boolean property values must not be null

Should: Null values should have their fields removed

Should: Empty array values should not be null

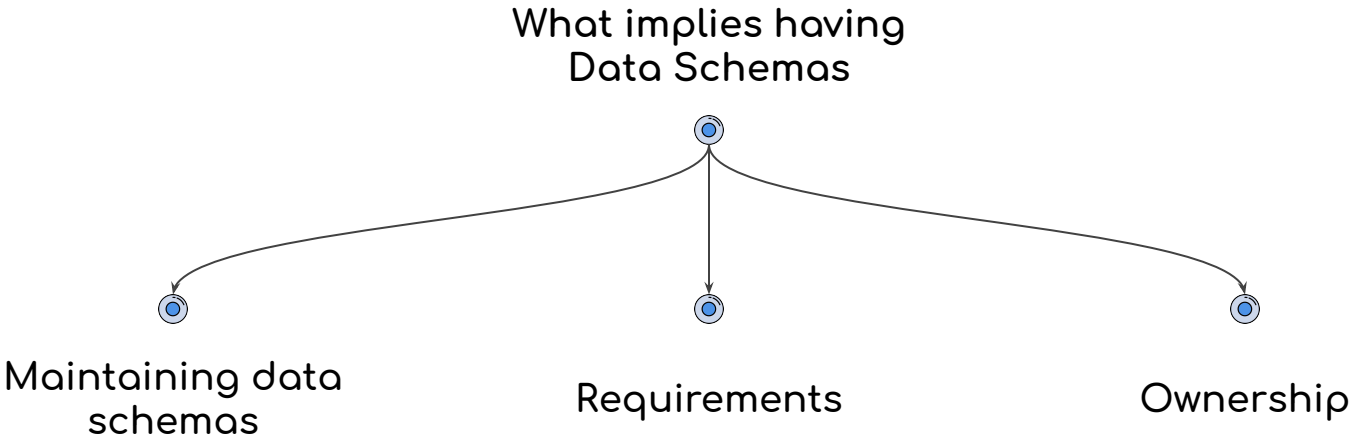
Should: Enumerations should be represented as Strings

Should: Date property values should conform to RFC 3339

May: Time durations and intervals could conform to ISO 8601

May: Standards could be used for Language, Country and Currency

zalando Based on opensource.zalando.com/restful-api-guidelines/#json-guidelines


What implies having
Data Schemas
Maintaining data
schemas
Requirements
Ownership

The team