Divolte Clickstream Collection

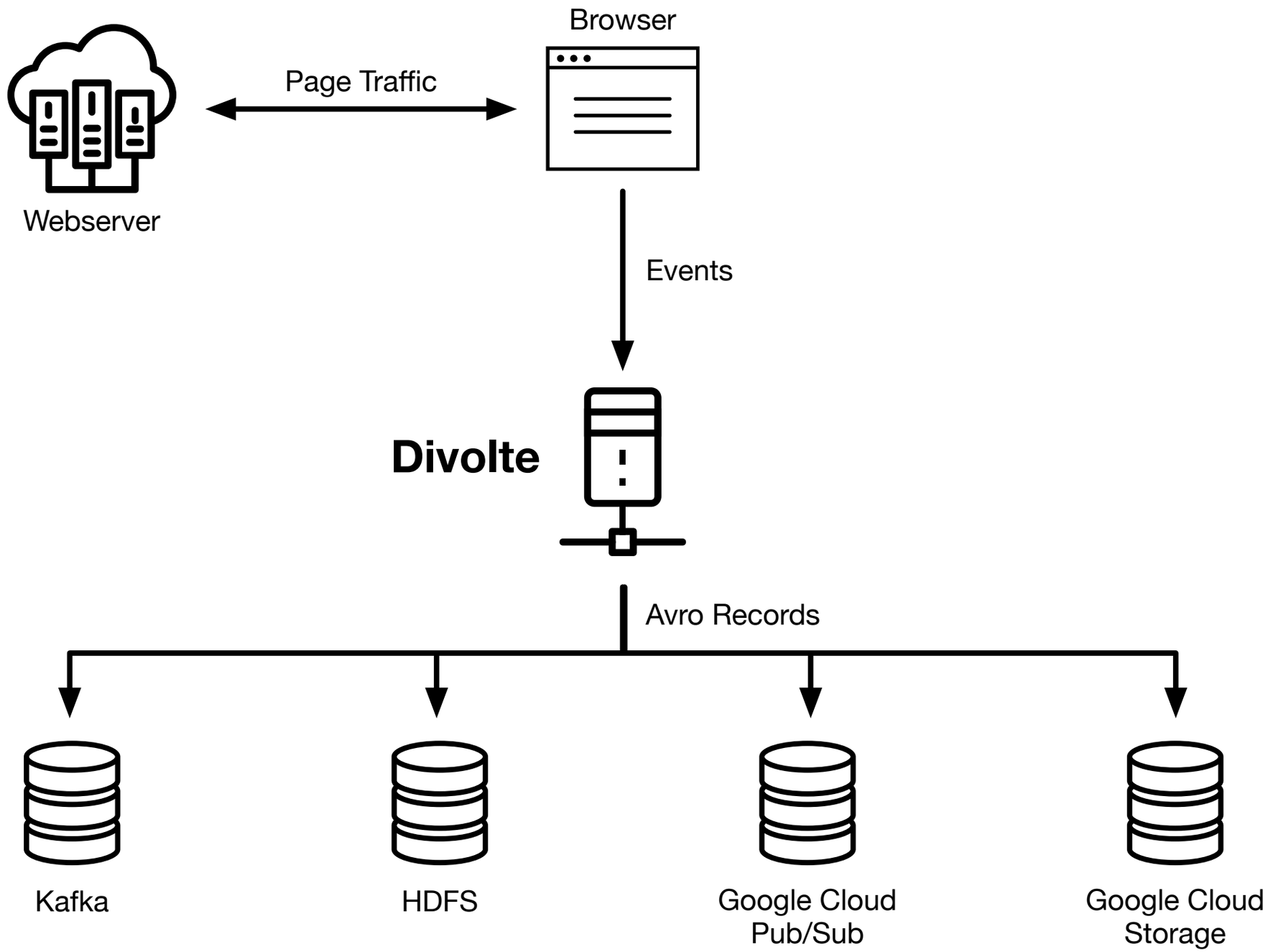
Andrew Snare
Big Data Engineer

@asnare
andrewsnare@godatadriven.com

GO DATA DRIVEN

What is it?

It Looks A Bit Like This...



What is it for?

Why?

- Gives you access to the raw clickstreams for your sites.
- Intended for data-science use-cases.
- No sampling or data aggregation.
- **Supports your business domain.**

GO DATA DRIVEN

What is it not?

- An analytics tag.
- A dashboarding tool.
- Generic event-routing infrastructure.

How do I use it?

- Deploy the server somewhere.
- Site integration starts with a single tag:

```
<script src="//example.com/divolte.js"
        defer async></script>
```

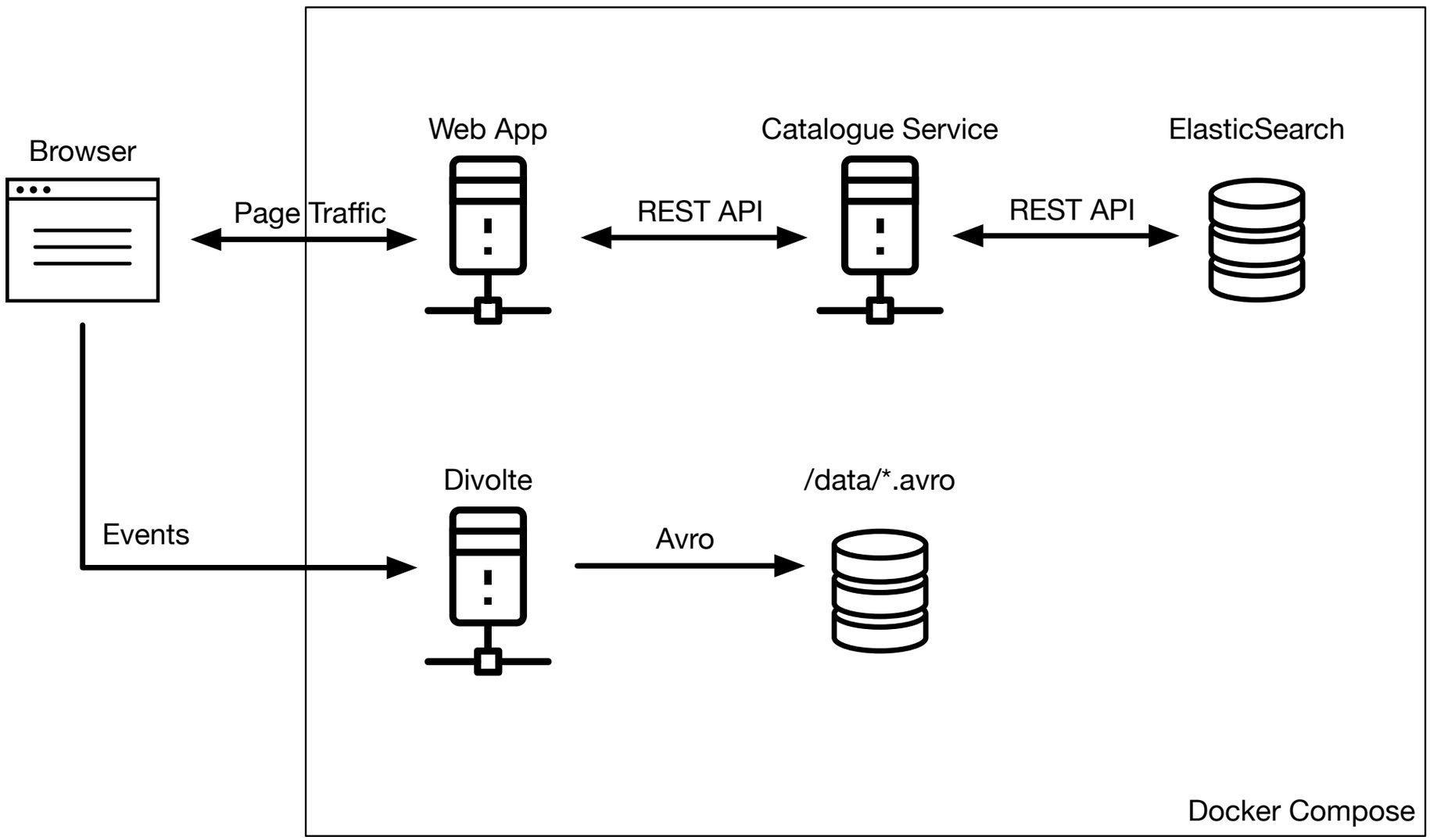
- This is enough to give you *page-view* events.

GO DATA DRIVEN

Shall we play?

The Playground

- Based on *Docker Compose*, so everyone can run it locally.
- We have a web shop.



GO DATA DRIVEN

First Things First

- Clone the repository:

  ```
  git clone git@github.com:godatadriven/divolte-workshop.git
  ```

- Have a quick look around:

  ```
  cd divolte-workshop
  ls -l
  ```

- Follow the steps for the first exercise:

  ```
  exercise-1.md
  ```

GO DATA DRIVEN

What was all that about?

What's a Page View?

- A page view starts when the user navigates to a specific page.
  - This triggers an automatic *Page View* event.
  - All events within the page view have the same *Page View Identifier.*

Example Event

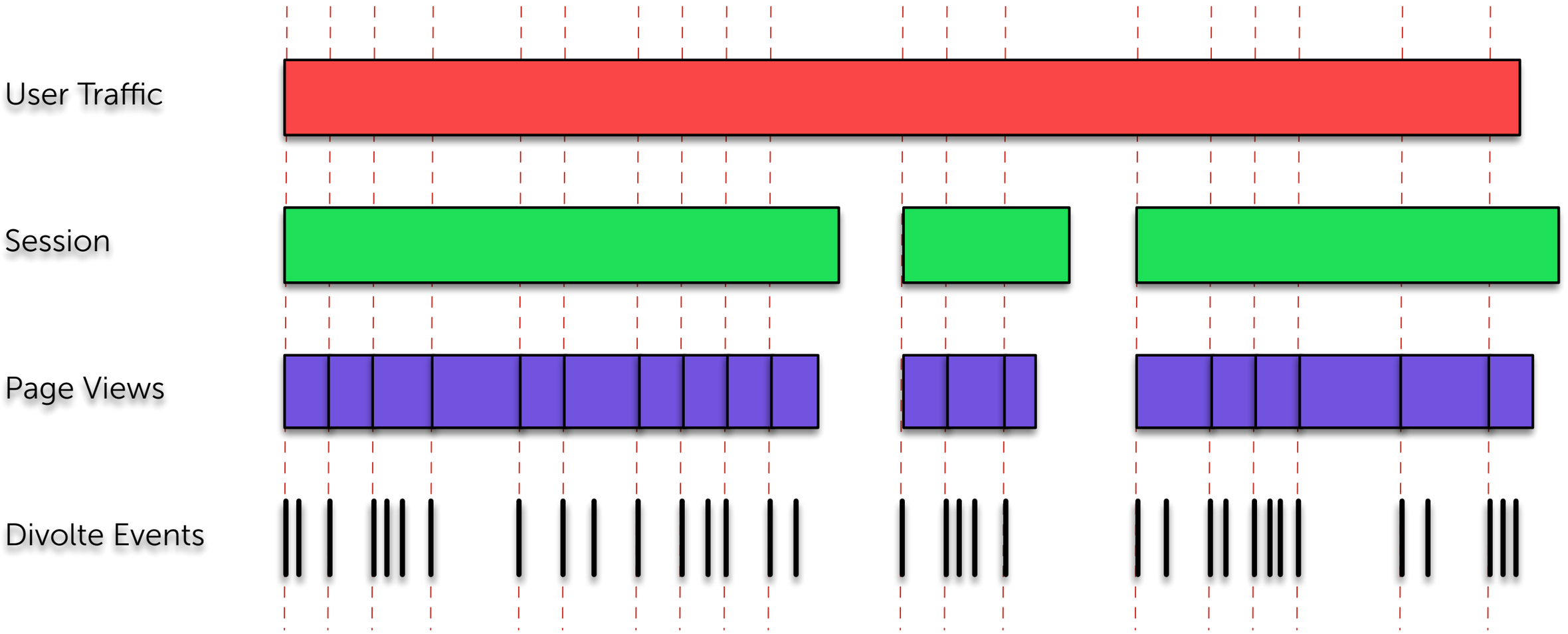

```
{ firstInSession: true,
  timestamp: 1539715158154,
  clientTimestamp: 1539715158139,
  remoteHost: '172.20.0.1',
  referer: { string: 'http://localhost:9011/category/animals/' },
  location: { string: 'http://localhost:9011/' },
  partyId: { string: '0:jg7unah9:jsMXeaWyxoSo5y5YFYuH6AR5uB8y7J3s' },
  sessionId: { string: '0:jnc2nfjt:nLSgoluDiSaSidlg1Oyhj~myoqgzc2OD' },
  pageViewId: { string: '0:m~kV05hM1Js3Ec2OzlCQY8VZcj9SYxRT' },
  eventId: { string: '0:m~kV05hM1Js3Ec2OzlCQY8VZcj9SYxRT0' },
  eventType: 'pageView' }
```


GO DATA DRIVEN

A few words about events...

- Events are normally triggered by interactions on a website.
- Every event has a unique *event identifier*.
- Events are associated with a *party*.
- Events belong to a session.
  - Each session has a unique *session identifier*.
  - A session ends when there are no events for half an hour.

...less words about events



GO DATA DRIVEN

What happens next?

- Events are sent by the browser to the collector.
- The collector *maps* these to Avro records.
- Mapping can be customised, and supports:
  - Extracting information from URLs, query parameters, cookies.
  - GeoIP reverse lookups.
  - User-agent parsing.

And then?

- Real-time:
  - Kafka
  - Google Pub/Sub
- Batch-based:
  - HDFS
  - Google Cloud Storage
  - Azure Blob Storage*
  - S3*

Avro

Um, Why Avro?

- Avro gives us some important things:
  - A technical schema that data must conform to.
  - A path forward when the schema needs to evolve.
    - Binary serialisation, including a file format.
- Divolte lets you *completely define your own schema*.
- **This is key to making the clickstream you collect relevant for your situation.**

Example Schema

```
{
    "namespace": "io.divolte.shop.record",
    "type": "record",
    "name": "ShopEventRecord",
    "fields": [
        { "name": "firstInSession",  "type": "boolean" },
        { "name": "timestamp",       "type": “long"    },
        { "name": "clientTimestamp", "type": "long"    },
        { "name": "remoteHost",      "type": "string"  },
        { "name": "referer",         "type": ["null", "string"], "default": null },
        { "name": "location",        "type": ["null", "string"], "default": null },
        { "name": "partyId",         "type": ["null", "string"], "default": null },
        { "name": "sessionId",       "type": ["null", "string"], "default": null },
        { "name": "pageViewId",      "type": ["null", "string"], "default": null },
        { "name": "eventId",         "type": ["null", "string"], "default": null },
        { "name": "eventType",       "type": "string",           "default": "unknown" }
    ]
}
```

GO DATA DRIVEN

Avro's Different Schemas

- Avro talks about the *writing* schema and the *reading* schema.
  - The *writing* schema is the one used to write the data.
  - The *reading* schema is the one a reader uses.
- They can be (and often are) the same.
- When reading, we need to know the writing schema.
- Avro files include the writing schema in a header.
- Understanding this is the key to evolution.

Schema Evolution

- Changes allowed by evolution:
  - Fields can be added.
  - Fields can be renamed.
    - The old names are mentioned in an alias.
  - Fields can be deleted.
    - "Old" readers can still read so long as the field had a default.
  - *Some* type changes: where conversion is trivial.
- These changes are similar to compatibility constraints on a database schema.

Let's add some fields...

```
{
    "namespace": "io.divolte.shop.record",
    "type": "record",
    "name": "ShopEventRecord",
    "fields": [
        { "name": "firstInSession",  "type": "boolean" },
        { "name": "timestamp",       "type": "long"    },
        { "name": "clientTimestamp", "type": "long"    },
        { "name": "remoteHost",      "type": "string"  },
        { "name": "referer",         "type": ["null", "string"], "default": null },
        { "name": "location",        "type": ["null", "string"], "default": null },
        { "name": "partyId",         "type": ["null", "string"], "default": null },
        { "name": "sessionId",       "type": ["null", "string"], "default": null },
        { "name": "pageViewId",      "type": ["null", "string"], "default": null },
        { "name": "eventId",         "type": ["null", "string"], "default": null },
        { "name": "eventType",       "type": "string",           "default": "unknown" }
    ]
}
```

GO DATA DRIVEN

Let's add some fields...

```
{
    "namespace": "io.divolte.shop.record",
    "type": "record",
    "name": "ShopEventRecord",
    "fields": [
        { "name": "firstInSession",  "type": "boolean" },
        { "name": "timestamp",       "type": "long"    },
        { "name": "clientTimestamp", "type": "long"    },
        { "name": "remoteHost",      "type": "string"  },
        { "name": "referer",         "type": ["null", "string"], "default": null },
        { "name": "location",        "type": ["null", "string"], "default": null },
        { "name": "partyId",         "type": ["null", "string"], "default": null },
        { "name": "sessionId",       "type": ["null", "string"], "default": null },
        { "name": "pageViewId",      "type": ["null", "string"], "default": null },
        { "name": "eventId",         "type": ["null", "string"], "default": null },
        { "name": "eventType",       "type": "string",           "default": "unknown" },

        { "name": "pageType",        "type": "string",           "default": "unknown" },
        { "name": "page",            "type": ["null", "int"],    "default": null },
        { "name": "productId",       "type": ["null", "string"], "default": null },
        { "name": "downloadId",      "type": ["null", "string"], "default": null },
        { "name": "category",        "type": ["null", "string"], "default": null }
    ]
}
```

GO DATA DRIVEN

Mapping

Basic Mapping

- Divolte uses a groovy-based DSL to map event information onto the Avro fields.

```
mapping {
    map firstInSession()  onto 'firstInSession'
    map timestamp()       onto 'timestamp'
    map clientTimestamp() onto 'clientTimestamp'
    map remoteHost()      onto 'remoteHost'
    map referer()         onto 'referer'
    map location()        onto 'location'
    map partyId()         onto 'partyId'
    map sessionId()       onto 'sessionId'
    map pageViewId()      onto 'pageViewId'
    map eventId()         onto 'eventId'
    map eventType()       onto 'eventType'
}
```

Basic Mapping

- Divolte uses a groovy-based DSL to map event information onto the Avro fields.

```
mapping {
    map firstInSession()  onto 'firstInSession'
    map timestamp()       onto 'timestamp'
    map clientTimestamp() onto 'clientTimestamp'
    map remoteHost()      onto 'remoteHost'
    map referer()         onto 'referer'
    map location()        onto 'location'
    map partyId()         onto 'partyId'
    map sessionId()       onto 'sessionId'
    map pageViewId()      onto 'pageViewId'
    map eventId()         onto 'eventId'
    map eventType()       onto 'eventType'
}
```

Event Information

Avro Fields

More Mapping

- Event information can be parsed:

```
def locationUri = parse location() to uri
map locationUri.path() to 'someField'
```

- Conditional mapping:

```
when locationUri.path().equalTo('/') apply {
  map 'home' onto 'pageType'
}
```

- Regular expression testing (and extraction) is possible.

GO DATA DRIVEN

Even More Mapping

- Control flows:

```
section {
    when something_something_something() apply {
        …
        // Skip rest of section.
        exit()
    }
    when something_something_else() apply {
        …
        // Skip rest of section.
        exit()
    }
    …
}
```

Additional Event Information

- Lots of event information is available:
  - Headers
  - Cookies
  - User-agent attributes
  - GeoIP data

GO DATA DRIVEN

Let’s play some more...

Better Events

- Follow the steps for the second exercise:

  `exercise-2.md`

GO DATA DRIVEN

Site Integration

More Events

- Out of the box, we get **`pageView`** events.
- There is a JavaScript API for triggering more events:

```
divolte.signal(EventType[, Event Parameters]);
```

- For example:

```
divolte.signal("videoStarted", {
                    videoId: 'whms1989',
                    audio: 'en',
                    subtitle: 'nl'
                });
```

How do we map these?

- Parameters for custom events can be mapped:

```
map eventParameters().value('videoId')
  onto 'videoId'
```

- You can map an entire JSON object onto a set of fields.

- JMES expressions can also be used to slice information out of the parameters:

```
map eventParameters().path('$.videoId')
  onto 'videoId'
```

Putting It All Together

Let's Do It!

- We're going to add some addition events into our application.
- Follow the steps for the third exercise:

  `exercise-3.md`

Odds and Sods

Is that all?

- Today we've explored the most simple setup. Divolte also supports:
  - Working with multiple sites.
  - Routing events to different destinations, each of which can have its own mapping.
- It's not supposed to do everything: it's supposed to be a solid start to collecting data which you process further.

Um, and mobile?

- Divolte supports a JSON endpoint.
- Mobile applications can write to this.
  - So can servers.
- SDKs?
  - Yes, please.

Recap

- Today, hopefully you've learnt:
  - How Divolte can be deployed to collect clickstream data.
  - How to use Avro schemas to specify a data schema for your events.
  - How mapping is the link between events and the schema.
  - How to integrate Divolte with your site.

Exploring Further

- Divolte has a lot more information on mapping and configuration: https://divolte.io/
- The Avro specification is concise and excellent: http://avro.apache.org/docs/1.8.2/spec.html
- Spark has excellent support for processing Avro files: https://github.com/databricks/spark-avro/
- JMES isn't as powerful as `jq`, but is still useful: http://jmespath.org/

GO DATA DRIVEN

Questions?

Thank-You!

We're hiring!

Andrew Snare
Big Data Engineer

@asnare
andrewsnare@godatadriven.com