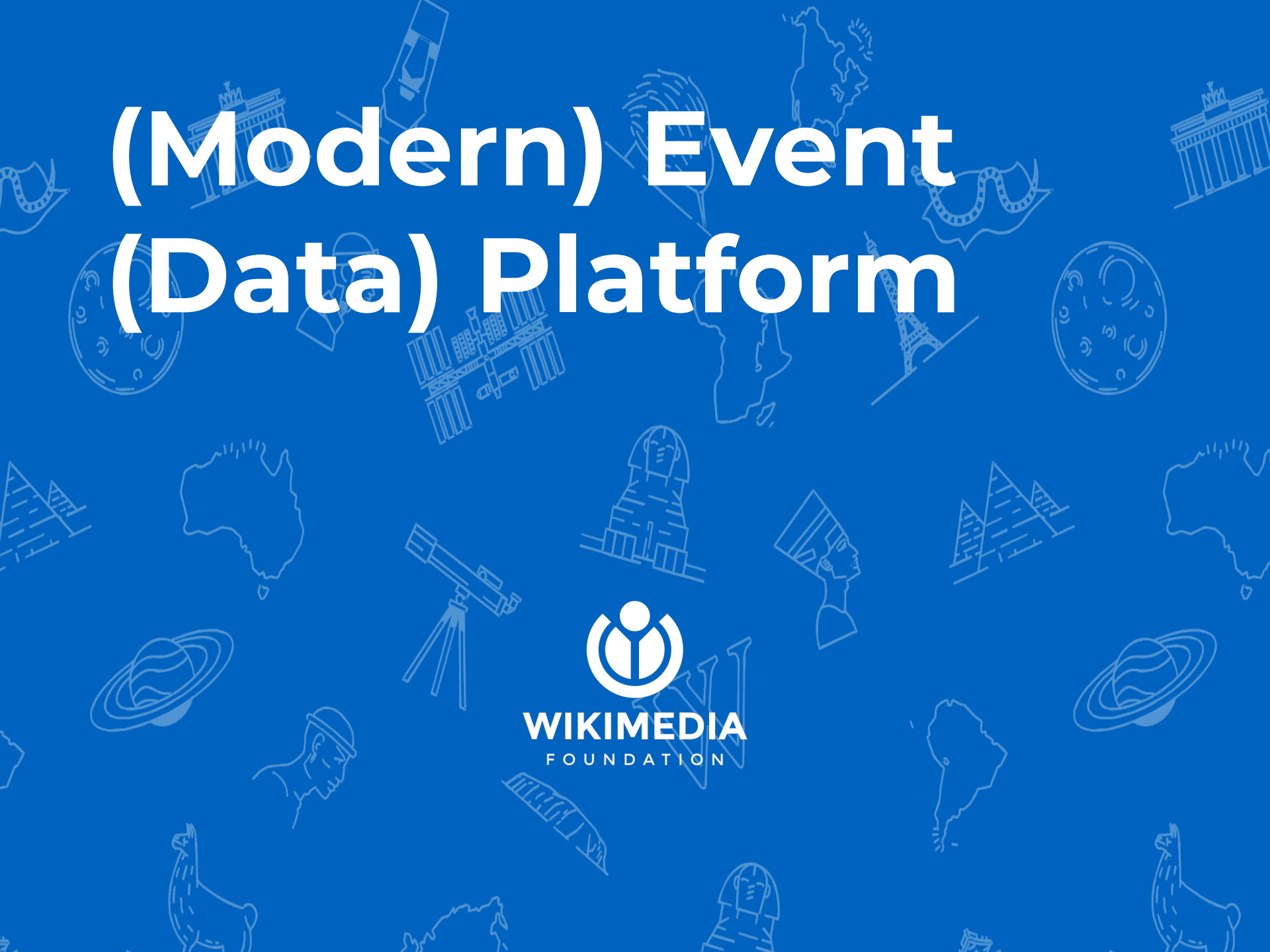
(Modern) Event (Data) Platform



Starting off with an old slide...

In 2018 I gave a tech talk titled

Event Stream Infrastructure

That talk contained the following slide.

EventLogging + EventBus

Quote from EventBus wikitech doc:

Ideally, these services would not be as different as they are. In the (probably distant) future, we'd like to modify ***EventLogging Analytics*** *so that it looks a little more like* ***EventBus***

Annnnnd today we have...

Event Platform

Event Platform

Motivation

WMF's **Event Platform**

enables building

event driven software.

Wait, first, what is an **event**?

What is an **event**?

Events are just way of modeling data:

something happens at a specific time.

- *edit saved at 9am*
- *user clicked button at 3pm*
- *luca made coffee at 2pm*, etc.

Events are facts

Modeling data as **events** is closer to reality* than modeling data as **state**.

Things happen, **then** state changes.

Events are history

One of **MediaWiki's** strengths is that the revision table is essentially an **event** history store. However:

Not all of **MediaWiki** data has history

(link changes, user renames, etc, user preferences, etc.)

Revision **events** are locked inside of MediaWiki MySQL

Events decouple

If **events** are **consumable** by anyone, they allow for building **decoupled** services.

Want to update your Elasticsearch index with the current state of a page? Just **consume events** as they happen and update; don't reach out to a centralized database.

Events+Kafka liberate data

If we emit **events** to a pub/sub message bus (like **Kafka**), unforseen use cases and services can access source data without altering the source data's code or datastore.

This empowers teams to make incremental architectural changes. Data is no longer siloed in a datastore behind an app or a service. It is exported by default.

Events liberate data

However, doing this requires a data **contract**.
Producers of data should not change its format in a way that might break **consumers**.

This **contract** is enforced by event **schemas**.

Events are complex

Events are simpler data, but the systems needed to process those events into useful data can be complicated.

Event Platform

Schemas

Why do we need **schemas?**

If you own both the **producer** and **consumer** of event data, then perhaps you don't.

But if you want data to be shared between many uses, you must ensure that data format changes don't break **consumers**!

Schemas are useful for

Ensuring event data satisfies a contract

Solving data integration problems (AKA **ETL**)

JSONSchema, ok!

WMF uses **JSONSchema**. Great!

But **JSONSchema** on its own is missing features we need:

Distributed schema **lookup**
(for validation and/or data integration)

Schema **evolution**
AKA versioning

Schema lookup

We should **always** be able to know the schema of an event.

With so many **producers** and **consumers**, we need to be able to do this from anywhere.

How does **Event Platform** solve this?

Schema lookup

JSONSchema already has a convention for locating ‘meta’ schemas (these are schemas of schemas, like a JSONSchema spec schema).

$schema

$schema is a **URI** pointing at the **JSONSchema** of the current JSON document. We can use this!

Schema lookup

But we want to be decentralized!

Decentralized **Schemas**

We set $schema to
a versioned path URI.

e.g.

```
/mediawiki/revision/create/1.0.0
```

Decentralized **Schemas**

Software then **prefixes** this with a base URI, either as a path in the **local** filesystem, or a **remote** HTTP location.

e.g.

```
https://schema.wikimedia.org/repositories/primary/jsonschema/mediawiki/revision/create/1.0.0
```

Event data is anywhere and everywhere!

All versions of all **schemas** must be look-up-able for-ev-uh.

Each new version of a **schema** must be 100% backwards compatible with the old one.

How to enforce?

jsonschema-tools

jsonschema-tools

is a schema repository manager.

Edit a single file and '**materialize**' static versioned schema files.

```
/mediawiki/revision/create/current.yaml ->
/mediawiki/revision/create/1.0.0
```

jsonschema-tools

Static version files give us consistent file path based **URI**s, from which we can lookup the schema.

Certain rules and conventions, including backwards compatibility are enforced via **tests**.

Versioning + **rules** enforcement satisfies our 2nd requirement:

Schema Evolution!

jsonschema-tools

demo

We've got **schemas**!

Now that we've got a good system for **versioned schemas**, how do we produce **events**?

Event Platform

EventGate

EventGate

is a HTTP event intake service.

By default, it knows how to use $schema **URIs** to lookup **event schemas**, validate incoming event data, and produce it to **Kafka**.

EventGate

EventGate is non-WMF specific. WMF provides custom functions that do what we need:

- validate using our **schema** repositories
- produce events to **kafka**

These WMF specific functions are in the eventgate-wikimedia repository.

The implementations of validate and produce are **pluggable**.

Event Platform

Event Stream Config

Event Stream Config

Original motivation:

modifying analytics **event** producer sampling rates.

Event Stream Config

Usage today:

- By **EventGate** to ensure only events of a single **schema** are allowed in a **stream**
- Determining which **EventGate** instance is allowed to produce which **streams**.
- Identifying which **streams** to produce canary **events** into for monitoring purposes
- Mapping from a **stream** (topic) name to a **schema** for structured stream processing.

EventStreamConfig

is a MediaWiki extension.

PHP and HTTP API to get arbitrary settings for a specific **stream**

Stream configs are stored in MW global `$wgEventStreams`.

```
[
    'stream' => 'mediawiki.revision-score',
    'schema_title' => 'mediawiki/revision/score',
    'destination_event_service' => 'eventgate-main',
],
```

Event Platform

Future work

What's next?

Thus far we've been focusing on the **production** of valid and consistent **event** data.

But what about actually **consuming** and using that data?

Two components still to do:

Event Stream Connectors

Event Stream Processing

Event Stream **Connectors**

abstract getting data out of (and into) **streams**.

We want connectors to get data into other datastores e.g. MySQL, ElasticSearch, Hadoop, Cassandra, etc.

We'd like to use Kafka Connect, but our first use case is a connector implementation from Confluent (Kafka Connect HDFS) which does not have a **FOSS** license.

Not yet sure where to go from here...

Event Stream **Processing**

is an abstraction for working with **streams**.

Allows you to think of **streams** as continuous datasets, and query them as such, possibly with **SQL**.

Stateful stream processing lets you build applications that keep and **redundant distributed state** updated by **streams**, a great way to do reliable **event** sourcing.

Check out upcoming **tech talk** from Ben Stopford for more about this! Wed Oct. 7 @ 15:00 UTC

We are likely to use Flink for this at WMF.

Event Platform

Questions?