Events:
Design, Catalogue, Discover and Use

Tom Fairbairn
March 2nd, 2020
Agenda

• Events & Event Driven Architectures
• Events as Assets
• Tools. Or the lack of...
• Approach & Architecture
• Demo
Who:

• Has deployed an EDA?
• Is developing one?
• Is planning one?
• Has no idea about events?
• Wants to hear me drone on about why they’re a good idea?
What is an “event?”

- A piece of data
- About something that’s happened
  Usually a *change*

Characteristics

- Has a source
- Has one or more sinks... we hope
- Relevant only in a *context*
- Temporal (= real time?)
- Asynchronous
Event myths

• New fangled
• *Only* asynchronous
  – API design
  – Must have asynchronous programming model
• The schema is the API/contract
• Can be implemented with REST point to point
To be useful, deployable and manageable, most event driven architectures are:

- Distributed
- Micro-service orientated
- Asynchronous
- Use the Publish/Subscribe pattern
- Make use of a *centralised*, distributed event distribution layer
• How did you feel about event driven when you first started to use it?
• Why?
Event non-myths

• **Difficult/hard to**
  – Understand
  – Govern
  – Monitor/Manage
  – Lack of orchestration (choreography) makes tracing, tracking and debug hard

• “Pain in the behind’

• “Annoying”

• “Disruptive”
• What event driven technologies/tools are you using?
• Do you have your own tooling?
It’s Hard to Build Event Driven Applications

Where do you discover events for reuse?

Why does an event exist?

What topic do you use to subscribe to it?

How do you determine the data structure of its payload?

Who should have access to it?

Who can tell you more about it?

Is your change backward compatible?

Who is impacted by your change?

Does it comply with security policy?
Event Assets

Events form *part* of your event assets

- Context
- Documentation
- Skills
- Applications
- Data
- Systems
Do we have a precedent?

API Management platforms answer who, what, when, where, why, how for RESTful APIs
PubSub+ Event Portal

Single place to design, create, discover, share, secure and manage your event assets
The Elements of Event Portal

**Application Domain**
Team, LOB, process (i.e. HR, Inventory, Billing)

**Application**
A Publisher and/or Subscriber

**Event**
Topic address + metadata that references a payload schema

**Schema**
Payload object definition JSON, Text, Binary, XML, Avro

---

**Acme Rideshare**

- Passenger App
- RideRequested
- Driver Management
The Demo!
Design Principles

Independent of:

- Architecture
- Event Distribution mechanism/tools
- Language
- Protocol
- Schema/serialisation
Warning!

• Early Days
• We want your feedback
Specify and Generate Async Applications

**Management**

**Spec**

**Codegen**

**Generated Code**

**Runtime Protocol**

### Synchronous

- API Mgmt
- **OPENAPI Initiative**
- `{}`
- **Swagger**
- **HTTP**

Available Clients:
- `node-scala`, `android`, `async-scala`, `epsilon`, `spring`, `coherence`, `nutmeg`,
- `xml`, `java`, `javascript`, `javaScript`, `closure`, `angular`,
- `java-csf-client`, `jmeter`, `ibm`, `apache`, `python`,
- `gRPC`, `ruby`, `scala`, `swagger`, `swagger-spring`,
- `net`, `typescript`, `typescript-angular`, `typescript-angular2`,
- `typescript-fetch`, `typescript-node`

Available Servers:
- `asnet5`, `asnetcore`,
- `spring-boot`, `java`, `jitter`,
- `jpa`, `jpa-csf`, `jpa-csf-cdi`, `jpa-resteasy`,
- `jpa`, `spring5`, `spring-boot`, `spring-cloud`,
- `mysql5`, `mongodb`, `nodejs-server`,
- `python-flask`, `rails`, `scala`, `gilex-mpe`,
- `cassandra`, `mysql`, `spring`, `undertow`

### Asynchronous

- Event Portal
- **AsyncAPI**
- **AsyncAPI**
- `node`
- AMQP, AMQPS, MQTT, MQTTs, WebSocket, WSS

Open Standard for asynchronous API definition

https://www.asyncapi.com/

"OpenAPI or RAML for asynchronous design"
The Demo! Discovery
Complete Event Lifecycle Management

**Discover**
- **Event Streams**: Search and locate events which are of interest
- **Relationships**: Understand the relationship between events, their sources and who is consuming them

**Operate**
- **Deploy**: start and connect to the broker
- **Secure**: enforce authentication and authorization
- **Audit**: identify runtime vs design time violations
- **Monitor**: understand utilization of events, schemas and apps
- **Inform**: discern insights to enhance apps, events and schemas; deprecate events that aren’t being utilized

**Define**
- **Use Case**: identify business outcome that can be improved by being event driven
- **Schema**: create business objects that represent the event context
- **Event**: create metadata that includes addressing (topic) and headers
- **Specify Application**: link events as inputs/outputs to the business logic

**Develop**
- **Generate Code**: use code generators to create application scaffolding
- **Add Business Logic**: create apps that perform required business function

**Develop**
Unifying Design Time and Run Time

**Event Portal:** Design & Development

- Application Developers
- Code Generators
- Managers & Administrators
- Data Architects
- Employee/Architects
- Business Analysts

**Event Mesh:** Runtime

- Code Generators
- Event Portal: Design & Development
- PubSub+ Event Broker
- IoT Devices, Vehicles, and Sensors
- Mobile & Web
- Enterprise Applications
- Event Sources/Handlers

**PubSub+ Event Portal**

- AsyncAPI Definition
- Map Payload Schema
- Review, Export Applications
- Discover, Create, Update, Review Applications
- Discover Events, Create More Event Driven Use Cases

**PubSub+ Event Broker**

- AMQP
- MQTT
- Websocket
- Kafka Connector

**Event Sources/Handlers**

- Push Event Driven App
- Discover, Create, Update, Review Applications
- Map Payload Schema
- Review, Export Applications
- Discover Events, Create More Event Driven Use Cases

**Metrics and Apps Status**

- Apply Runtime Config and Security Policies
- Audit Runtime for Rule Violations

**AsyncAPI Definition**

- Discover Events, Create More Event Driven Use Cases
solace.

That’s Possible