cloudevents

Why?

Events are everywhere. However, event publishers tend to describe events differently.

Consequences...

Consistency

The lack of a common way of describing events means developers must constantly re-learn how to receive events.

Consequences...

Accessibility

This also limits the potential for libraries, tooling and infrastructure to aide the delivery of event data across environments, like SDKs, event routers or tracing systems.

Consequences...

Portability

The portability and productivity we can achieve from event data is hindered overall.

CloudEvents

CloudEvents is a specification for describing event data in common formats to provide interoperability across services, platforms and systems.

24 October 2019

- CNCF approved graduation from "incubator" to "sandbox"
- CloudEvents specification v1.0 released
- Currently being adopted in Knative Eventing 🚧

Concepts

An *event* includes context and data about an *occurrence*. Each occurrence is uniquely identified by the data of the event.

Events represent facts and therefore do not include a destination, whereas messages convey intent, transporting data from a source to a given destination.

Events

Events can be delivered through:

- Industry standard protocols (e.g. HTTP, AMQP, MQTT, SMTP)
- Open-source protocols (e.g. Kafka, NATS)
- Platform/vendor specific protocols (AWS Kinesis, Azure Event Grid).

Layered architecture

1. The *base* specification defines an abstract model of attributes (key-value pairs) and rules defining a CloudEvent.
2. The *extensions* add use-case specific extension attributes and rules, e.g. to support different tracing standards.
3. The event format encodings, e.g. JSON, define how the base specification together with any extensions is encoded.
4. The protocol bindings, e.g. HTTP, defines how the CloudEvent is bound to a transport frame.

Message

Events are transported from a source to a destination via messages.

A "structured-mode message" is one where the event is fully encoded using a stand-alone event format and stored in the message body.

A "binary-mode message" is one where the event data is stored in the message body, and event attributes are stored as part of message meta-data.

Attributes

Every CloudEvent conforming to this specification MUST include context attributes designated as REQUIRED, MAY include one or more OPTIONAL context attributes and MAY include one or more extension attributes.

Required Attributes

`id`

Type: String

Description: Identifies the event.

Examples:
- An event counter maintained by the producer
- A UUID

`source`

Type: URI-reference

Description: Identifies the context in which an event happened. Often this will include information such as the type of the event source, the organization publishing the event or the process that produced the event. The exact syntax and semantics behind the data encoded in the URI is defined by the event producer.

Producers MUST ensure that source + id is unique for each distinct event.

Examples:

Internet-wide unique URI with a DNS authority.

- https://github.com/cloudevents
- mailto:cncf-wg-serverless@lists.cncf.io

Universally-unique URN with a UUID:

- urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66

Application-specific identifiers

- /cloudevents/spec/pull/123
- /sensors/tn-1234567/alerts
- 1-555-123-4567

`specversion`

Type: String

Description: The version of the CloudEvents specification which the event uses.

Example:

```
- 1.0
```

`type`

Type: String

Description: This attribute contains a value describing the type of event related to the originating occurrence. Often this attribute is used for routing, observability, policy enforcement, etc.

Examples:
- com.github.pull.create
- com.example.object.delete.v2

Optional attributes

`datacontenttype`

Type: String

Description: Content type of `data` value.

Example: `"application/xml"`

`subject`

Type: String

Description: This describes the subject of the event in the context of the event producer (identified by source). In publish-subscribe scenarios, a subscriber will typically subscribe to events emitted by a source, but the source identifier alone might not be sufficient as a qualifier for any specific event if the source context has internal sub-structure.

`time`

Type: Timestamp

Description: Timestamp of when the occurrence happened.

Extension Context Attributes

A CloudEvent MAY include any number of additional context attributes with distinct names, known as "extension attributes". Extension attributes MUST follow the same naming convention and use the same type system as standard attributes.

Extension attributes have no defined meaning in this specification, they allow external systems to attach metadata to an event, much like HTTP custom headers.

Event Data

CloudEvents MAY include domain-specific information about the occurrence. When present, this information will be encapsulated within `data`.

Description: The event payload. This specification does not place any restriction on the type of this information. It is encoded into a media format which is specified by the datacontenttype attribute (e.g. application/json)

Examples

Example JSON event with `String`-valued data:

```
{
    "specversion" : "1.0",
    "type" : "com.example.someevent",
    "source" : "/mycontext",
    "id" : "A234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "text/xml",
    "data" : "<much wow=\"xml\"/>"
}
```

Example JSON event with Binary-valued data:

```
{
    "specversion" : "1.0",
    "type" : "com.example.someevent",
    "source" : "/mycontext",
    "id" : "B234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "application/vnd.apache.thrift.binary",
    "data_base64" : "... base64 encoded string ..."
}
```

Example JSON event with JSON data for the "data" member, either derived from a `Map` or JSON data:

```
{
    "specversion" : "1.0",
    "type" : "com.example.someevent",
    "source" : "/mycontext",
    "id" : "C234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "application/json",
    "data" : {
        "appinfoA" : "abc",
        "appinfoB" : 123,
        "appinfoC" : true
    }
}
```

Format encodings

Available:
- JSON
- Avro

Unavailable in v1.0:
- Protocol Buffers
- Thrift
- CBOR
- FlatBuffers
- BSON

HTTP protocol binding

The HTTP Protocol Binding for CloudEvents defines how events are mapped to HTTP 1.1 request and response messages.

HTTP Message Mapping

The receiver of the event can distinguish between the modes by inspecting the `Content-Type` header value.

- `application/cloudevents` -> structured mode
- `application/cloudevents-batch` -> batched mode
- otherwise -> binary mode

Binary Content Mode

HTTP `Content-Type` header indicates the `datacontenttype` attribute.

Other CloudEvents attributes are transmitted as custom HTTP headers

The `data` is the HTTP message body.

Binary example

```
POST /someresource HTTP/1.1
Host: webhook.example.com
ce-specversion: 1.0
ce-type: com.example.someevent
ce-time: 2018-04-05T03:56:24Z
ce-id: 1234-1234-1234
ce-source: /mycontext/subcontext
    .... further attributes ...
Content-Type: application/json; charset=utf-8
Content-Length: nnnn

{
    ... application data ...
}
```

Structured Content Mode

The structured content mode keeps event metadata and data together in the payload.

```
Content-Type: application/cloudevents+json;
charset=UTF-8
```

Structured example

```
PUT /myresource HTTP/1.1
Host: webhook.example.com
Content-Type: application/cloudevents+json; charset=utf-8
Content-Length: nnnn

{
    "specversion" : "1.0",
    "type" : "com.example.someevent",

    ... further attributes omitted ...

    "data" : {
        ... application data ...
    }
}
```

Batched Content Mode

In the batched content mode several events are batched into a single HTTP request or response body.

```
Content-Type: application/cloudevents-batch+json;
charset=UTF-8
```

Batched example

```
PUT /myresource HTTP/1.1
Host: webhook.example.com
Content-Type: application/cloudevents-batch+json; charset=utf-8
Content-Length: nnnn

[
    {
        "specversion" : "1.0",
        "type" : "com.example.someevent",

        ... further attributes omitted ...

        "data" : {
            ... application data ...
        }
    },
    {
        "specversion" : "1.0",
        "type" : "com.example.someotherevent",

        ... further attributes omitted ...

        "data" : {
            ... application data ...
        }
    }
]
```

Other protocol bindings

- HTTP
- AMQP
- Kafka
- MQTT
- NATS

SDKs

A separate specification describes requirements for CloudEvents SDKs.

- Contribution requirements (support, spec currency)
- Technical requirements (CE model, must have HTTP, idiomatic)
- Object Model Structure guidelines
- Documentation requirements

Available SDKs

- Go
- C#
- Java
- Javascript
- Python
- Ruby

Go SDK

```
import "github.com/cloudevents/sdk-go"

event := cloudevents.NewEvent()
event.SetID("ABC-123")
event.SetType("com.cloudevents.readme.sent")
event.SetSource("http://localhost:8080/")
event.SetData(data)
```

Sending a CloudEvent in Go

```go
t, err := cloudevents.NewHTTPTransport(
    cloudevents.WithTarget("http://localhost:8080/"),
    cloudevents.WithEncoding(cloudevents.HTTPBinaryV02),
)
if err != nil {
    panic("failed to create transport, " + err.Error())
}

c, err := cloudevents.NewClient(t)
if err != nil {
    panic("unable to create cloudevent client: " + err.Error())
}
if err := c.Send(ctx, event); err != nil {
    panic("failed to send cloudevent: " + err.Error())
}
```

Receiving a CloudEvent over HTTP in Go

```go
func Receive(event cloudevents.Event) {
    // do something with event.Context and event.Data (via event.DataAs(foo)
}

func main() {
    c, err := cloudevents.NewDefaultClient()
    if err != nil {
        log.Fatalf("failed to create client, %v", err)
    }
    log.Fatal(c.StartReceiver(context.Background(), Receive));
}
```

Go SDK support

Transports:

- HTTP
- AMQP
- NATS
- GCP PubSub

Formats:

- JSON
- Text
- XML

Thanks!