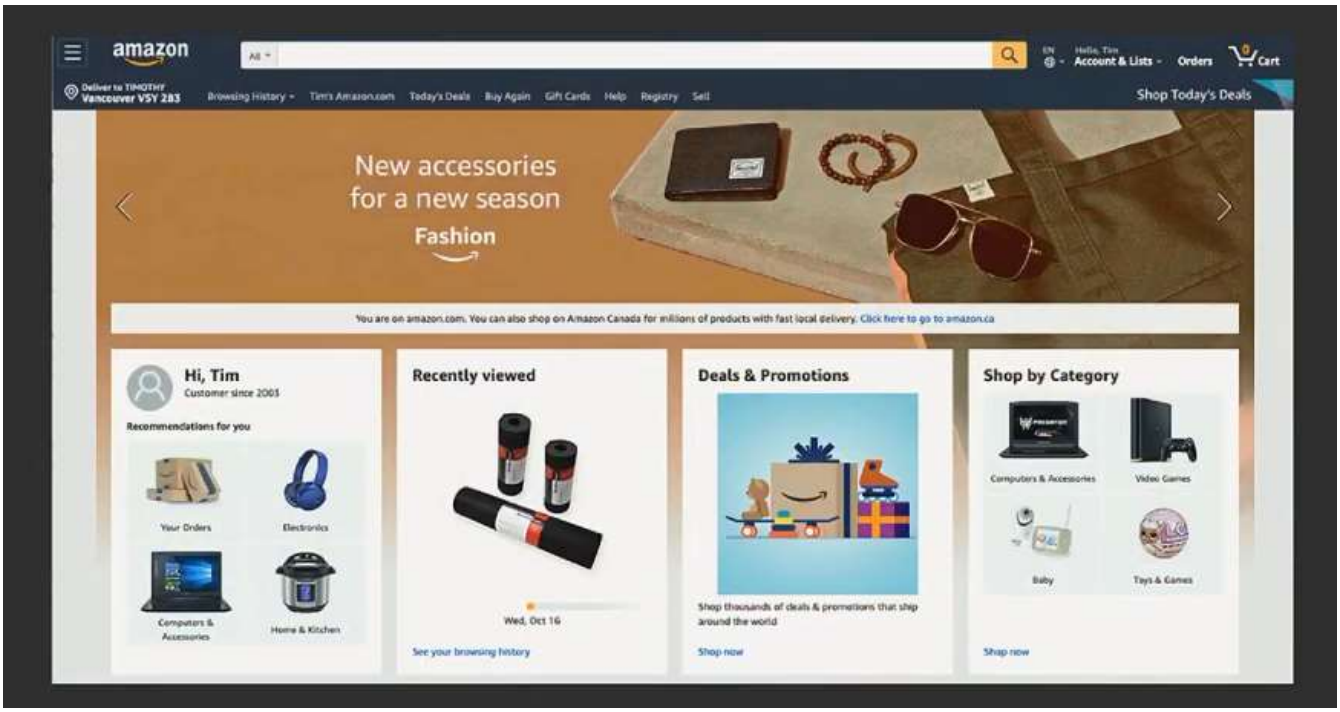
Event-driven architectures are getting a lot of attention. We have recently invested in tools and infrastructure to make event-driven architectures easier to implement and operate. In this session, we discuss what events are, why the community is increasingly interested in event-centric applications, and what's new in the domain and with AWS in particular. In addition, we discuss the challenges that still face us and our customers. By the end of this session, you understand the key principles and benefits of being event-driven.



This website is built as an event-driven architecture. A lot of things happen when the user clicks the buy button to generate the 'purchase' event.

A lot of async things need to get done after the customer clicks the ‘buy’ button, a ‘purchase’ event is generated and dropped in the queue with details saying that the customer bought something. Another system then picks up the event, opens the payload and then does its own stuff like creating an ‘order’ or ‘packing’ or ‘shipping’ event.

Amazon.com does not need strong ordering guarantees, it knows how to sort and filter out duplicates, does not do push into other backend systems and instead lets them ingest from queues at their own rates. These are the types of questions you need to ask when building an event-driven architecture.

Is event-driven for you?

1. Are you passing around self-contained transactions?

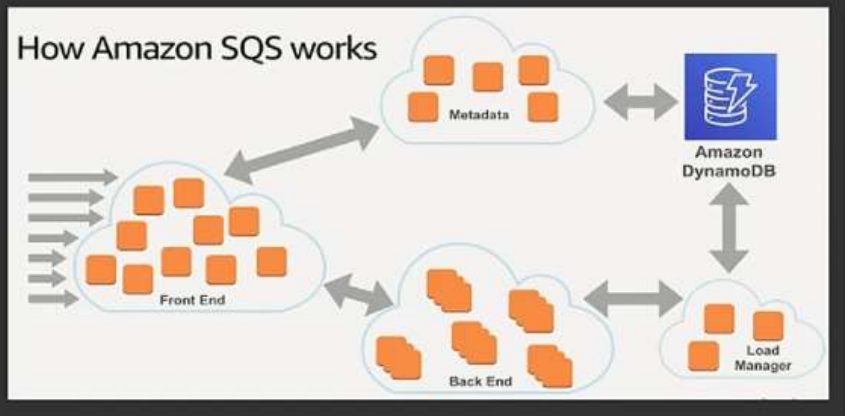
"Account 0973482 bought an Instapot ID 238479r8732 for $131.32, Visa confirmation 341513, in Prime, sending to 510 W Georgia St, Vancouver."

Is event-driven for you?

1. Are you passing around self-contained transactions?
2. Are useful events available for free?

Is event-driven for you?

1. Are you passing around self-contained transactions?
2. Are useful events available for free?
3. Do you need to strongly decouple your microservices?



Is event-driven for you?

1. Are you passing around self-contained transactions?
2. Are useful events available for free?
3. Do you need to decouple your microservices?
4. Do you need publish-and-subscribe?

Photo: Terje Skjerdal

A small AWS event

```
{
  "version": "0",
  "id": "f7a39f75-eff9-a823-5534-1075b196edd3",
  "detail-type": "EC2 Instance State-change Notification",
  "source": "aws.ec2",
  "account": "************",
  "time": "2018-08-21T20:55:26Z",
  "region": "us-east-1",
  "resources": [ ],
  "detail": {
    "instance-id": "i-00b414b880501ae45",
    "state": "running"
  }
}
```

This is an event that tells you about a state change in EC2 from our EventBridge service. The ***detail*** field tells you what happened. The top-level part that don’t change is called the event envelope

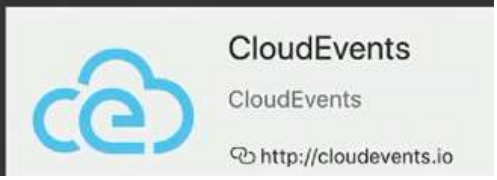
A larger AWS event


```
{
  "version": "0",
  "id": "5af0d99b-0841-2766-e5d5-06a865895fdf",
  "detail-type": "Support Ticket: Status Changed",
  "source": "aws.partner/zendesk.com/9242270/default",
  "time": "2019-05-25T01:23:45Z",
  "region": "us-east-1",
  "resources": [ ],
  "detail": {
    "ticket_event": {
      "type": "Status Changed",
      "previous": "open",
      "current": "solved",
      "ticket": {
        "id": 35436,
        "created_at": "2019-05-20T22:55:29Z",
        "updated_at": "2019-05-25T01:23:45Z",
        . . .
```


This is another event generated from Zendesk to AWS for injecting events over a special API.

Example CloudEvent


```
{
    "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
    }
}
```

This needs to have some data in it as below

AWS event → CloudEvent

```
func toCloudEvent(ae *awsevents.AWSEvent) (*cloudevents.Event, error) {
    ce := cloudevents.NewEvent();

    ce.SetType(ae.GetDetailType())
    awsSource := ae.GetSource()
    if (strings.HasPrefix(awsSource, "aws.")) {
        awsSource = awsSource[4:]
    }
    ce.SetSource("com.amazonaws/"  + string(ae.GetRegion()) + "/" + awsSource)
    ce.SetID(ae.GetId())
    ce.SetTime(ae.GetTime());
    ce.SetExtension("awsregion", ae.GetRegion())
    ce.SetExtension("awsaccount", ae.GetAccount())
    ce.SetDataContentType("application/json")
    err := ce.SetData(ae.GetDetail())
    if err != nil {
        return nil, err
    }
    return &ce, nil
}
```

AWS event → CloudEvent

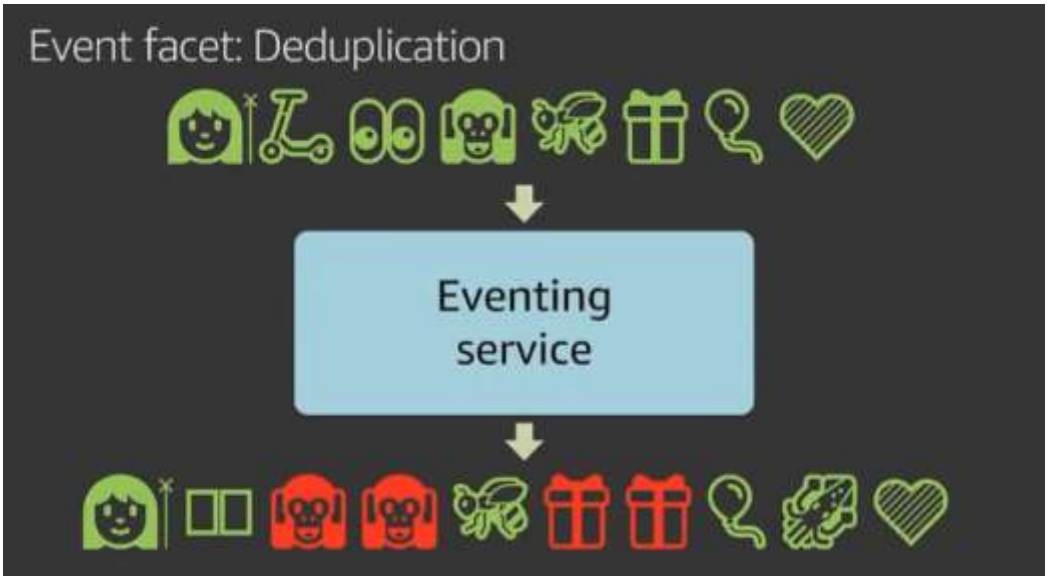

```
{
 "awsaccount": "123456789012",
 "awsregion": "us-east-2",
 "contenttype": "application/json",
 "data": {
  "instance-id": "i-00b414b880501ae45",
  "state": "running"
 },
 "id": "B977EE77-75E0-7A64-DD73-73A159BB4FF2",
 "source": "com.amazonaws/us-east-2/ec2",
 "specversion": "0.2",
 "time": "2019-10-28T19:56:14.195503Z",
 "type": "EC2 Instance State-change Notification"
}
```


This is the equivalent EC2 state change event wrapped up as a CloudEvent

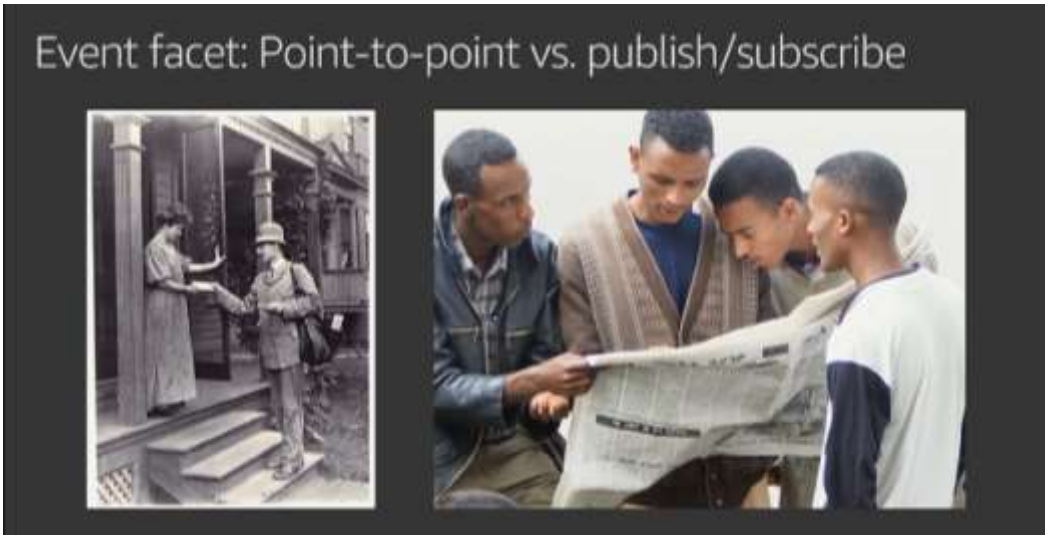
An event can be a command, a fact that something happened, a record of what somebody/something did, etc. What we generally care about is what we can do regarding processing, reformatting etc of events.

Facets allow us to structure our events

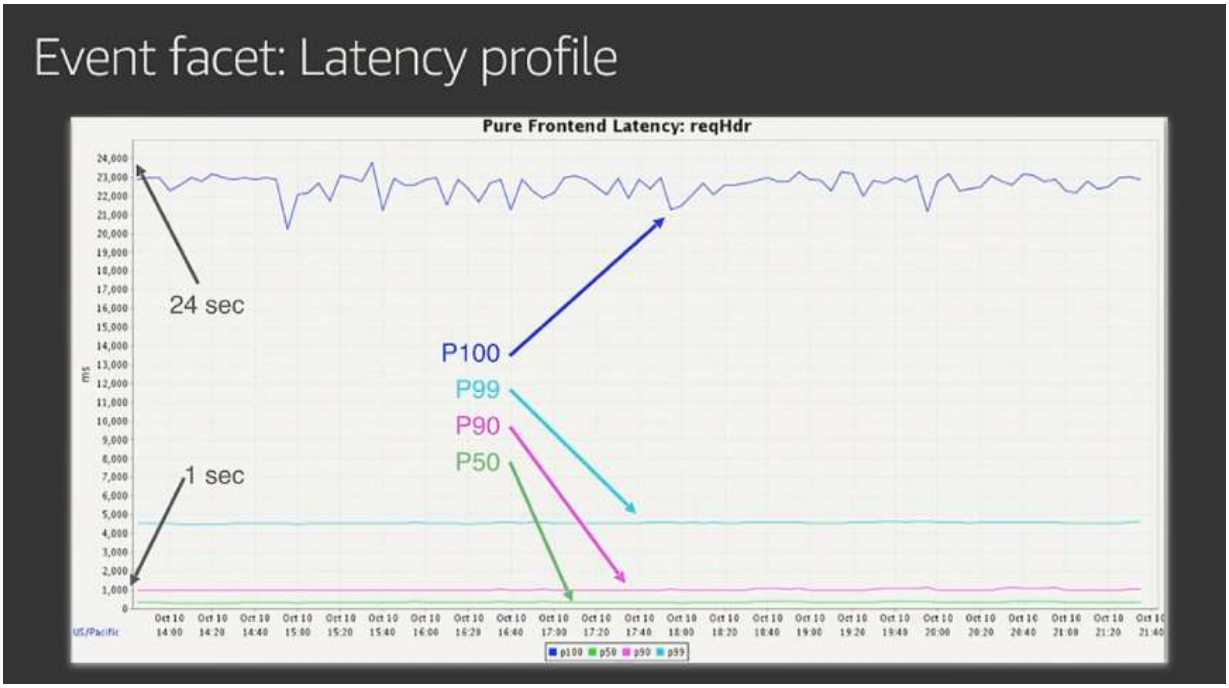
Strict FIFO is only meaningful if you have a single sender and banking transactions will generally be FIFO but more expensive and less scalable

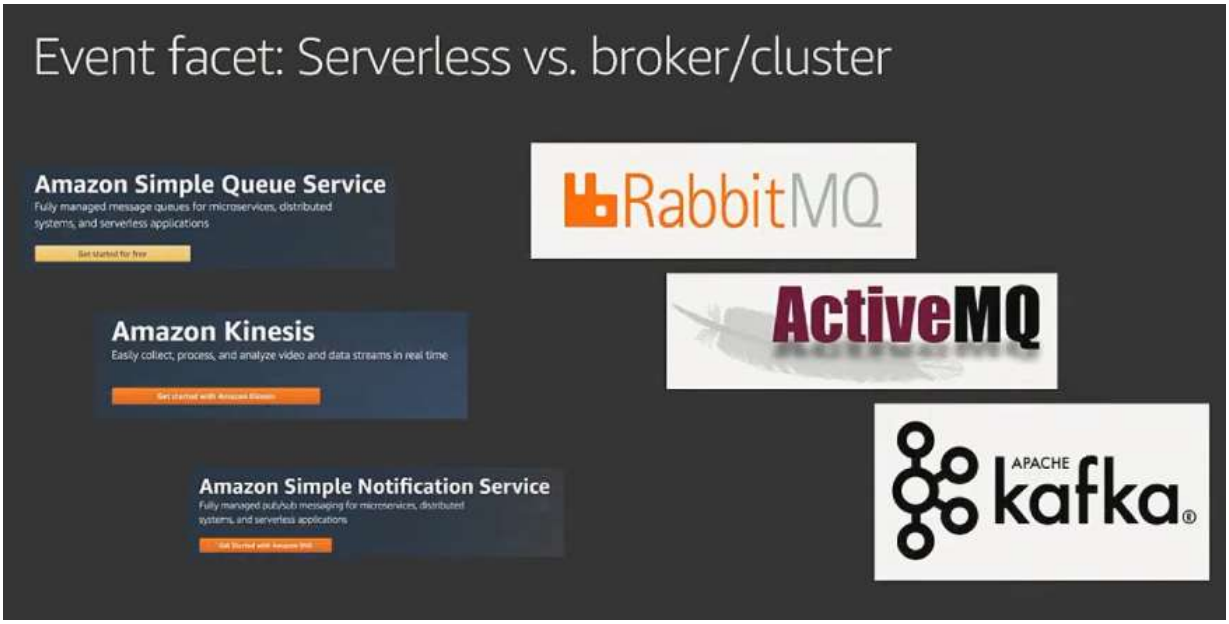


Does an event created get available and consumed exactly once or are you implementing at-least-once? Even when things go down. You can make sure APIs idempotent like the SQS createQueue API, or you can make your database backends transactional that can detect the state of a record and ignore other prior states.



You can get the events by polling the source periodically or pushing on to topics or lambda endpoints. Polling allows you to consume data when you have the capacity to do that.





There are 2 kinds of ***eventing and messaging infrastructure***, *Serverless* and *Broker/Cluster*, which are actual servers that needs maintaining and will require scaling. Brokers mostly don't use ***HTTP*** (that is slower since it sets up a connection and tears it down) and instead use ***TCP*** or some framing protocols like ***AMQP*** that guarantees lower latency. There is now ***HTTP long polling*** and ***HTTP2*** for multiple transactions. ***Quic or HTTP3*** is UDP that is coming with very different latency profile and very fast.

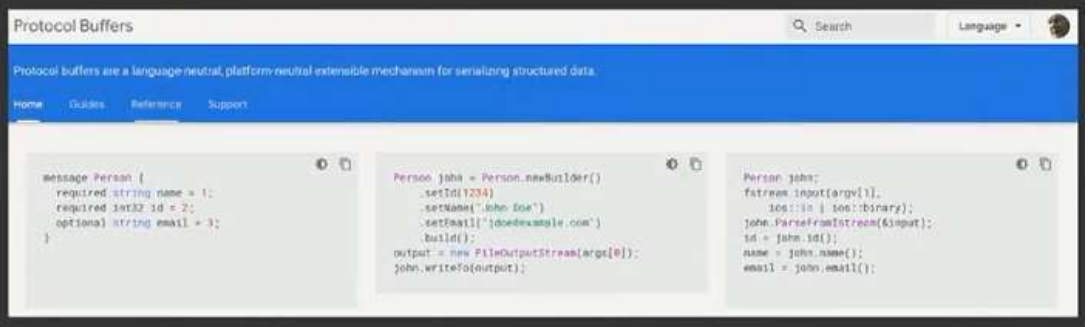
Event facet: Filtering vs. firehose

```
{
  "vendor": [ "vitamix", "instapot" ],
  "product_detail": {
    "price_usd": [
      { "numeric": ["<=", 150] }
    ]
  }
}
```

Most Pub/Sub and eventing APIs now come with filtering capabilities to reduce the streams data coming through to the event subscriptions

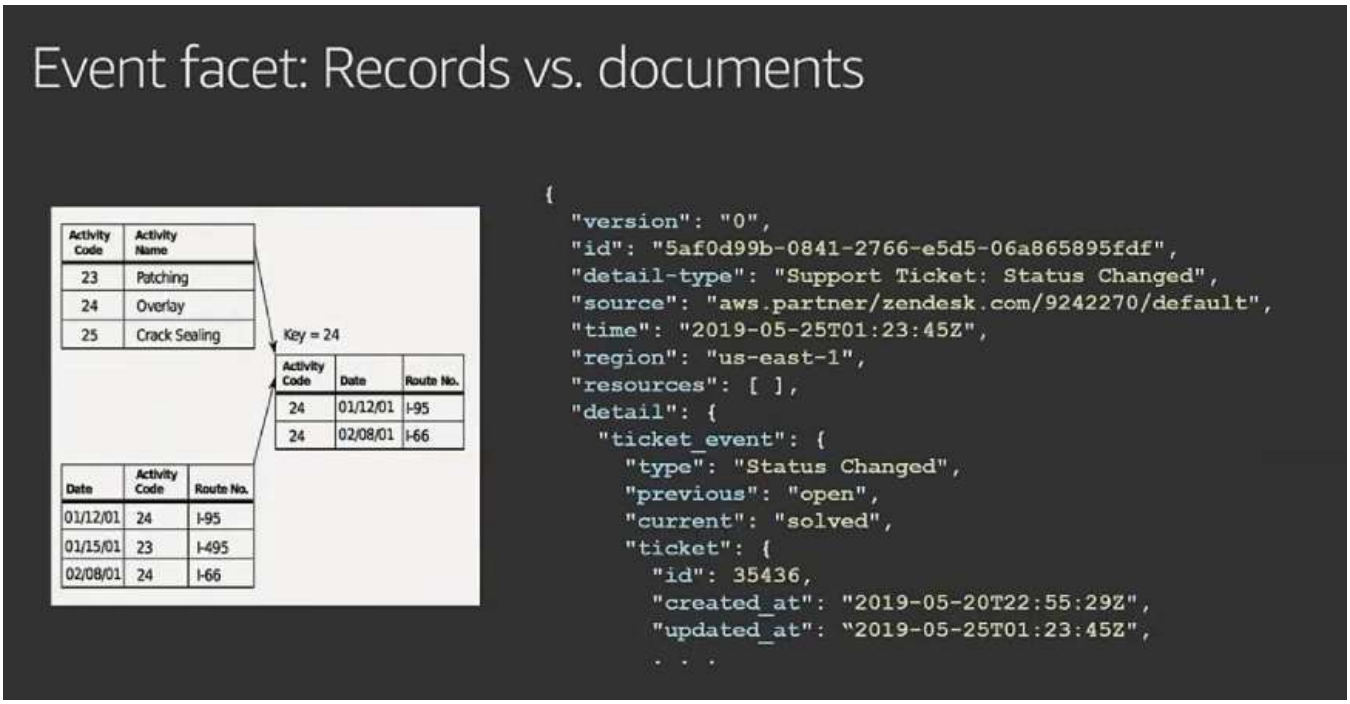
Event facet: Data blobs vs. structured objects

```
{
  "name":  "John Doe",
  "id":    1234,
  "email": "johndoe@example.com"
}
```

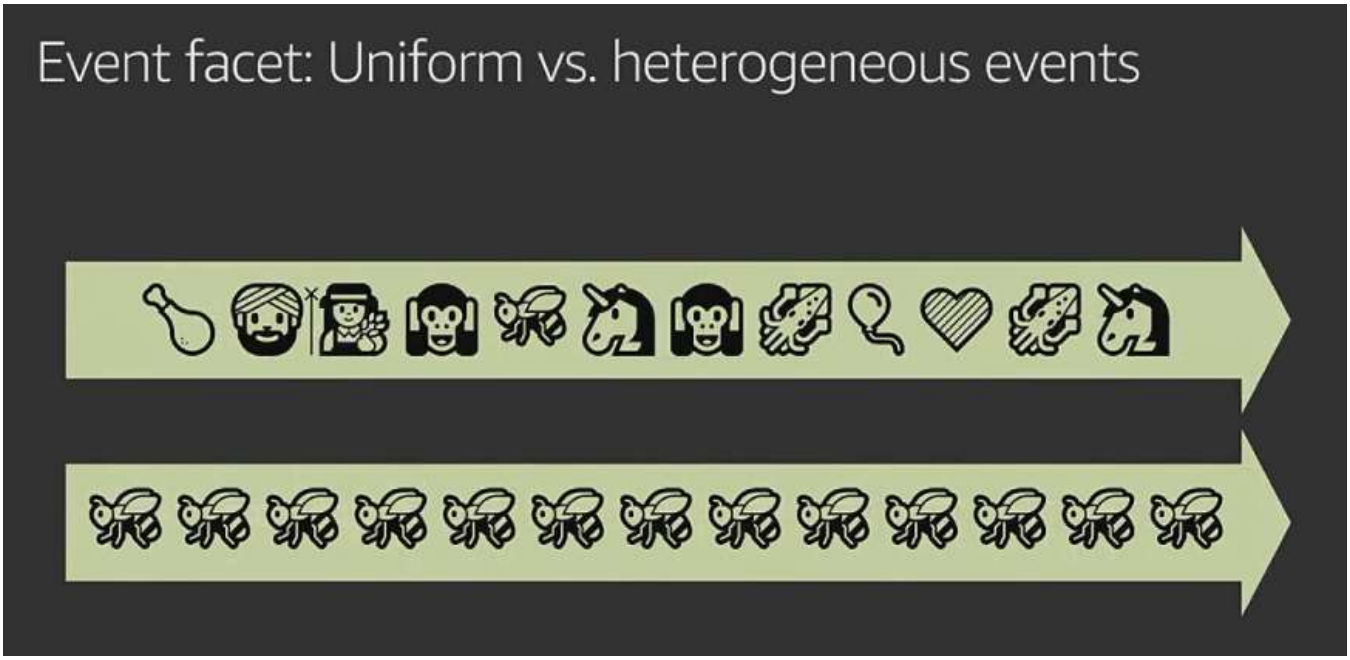


(Also Avro, Thrift, Parquet, Cap'n Proto, Ion, etc.)

You can wrap/unwrap your data in JSON or blobs, binary formats like Avro, Protocol Buffers are beginning to be used even though they are smaller to transport but can't be unpacked easily.



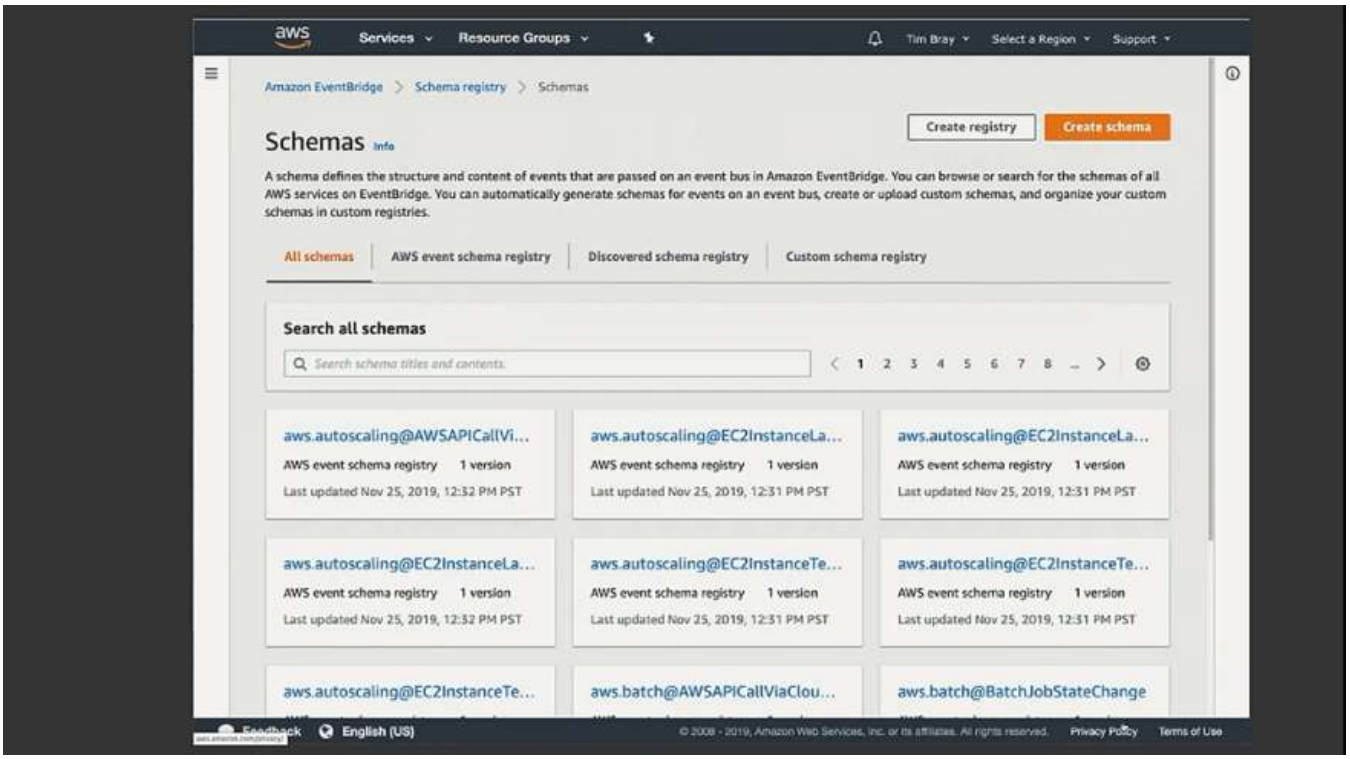
A lot of the event content is business data that can be complex and spread over relational data. These are different from having JSON-like data that are deeply nested data.



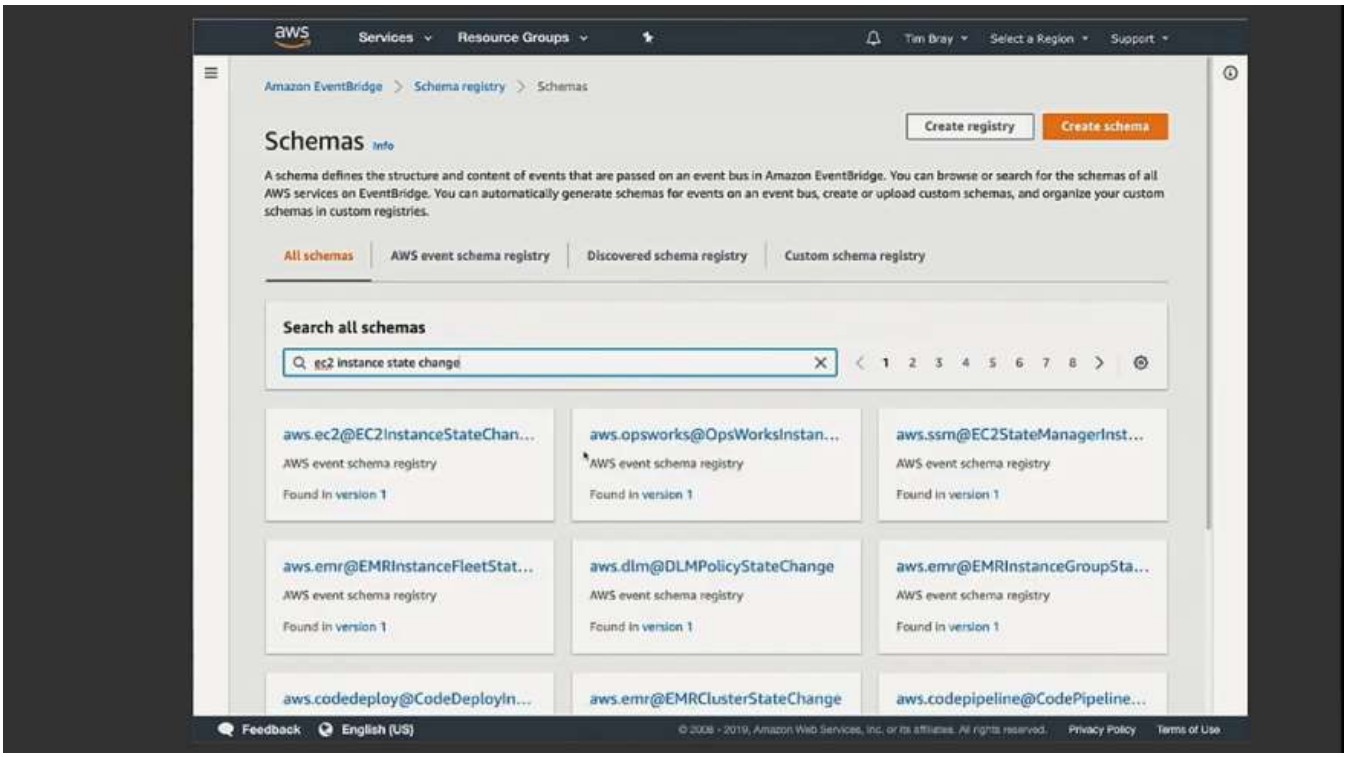
Are your events more the same or very different events and formats. This will affect the design of your event bus.

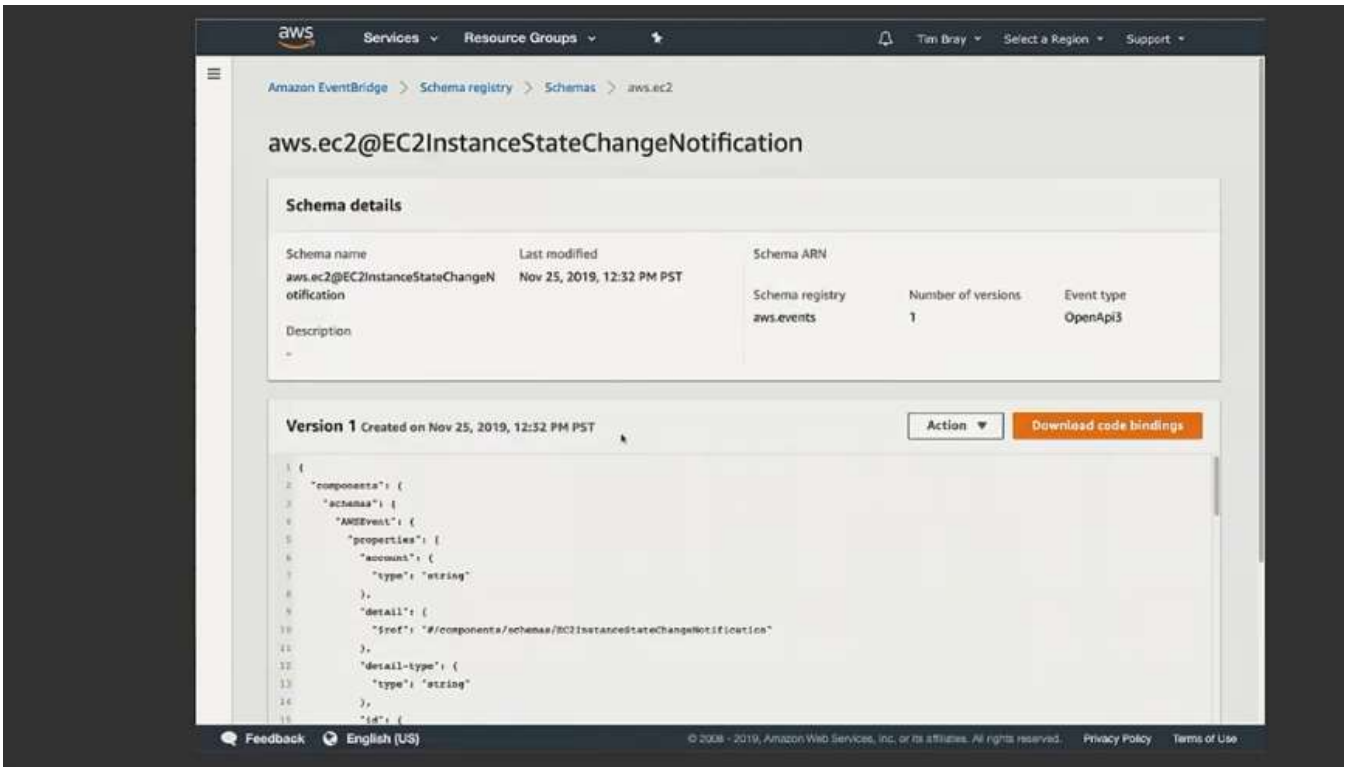
```
boolean isPassNode(final JsonNode node) {
  if (node.isObject()) {
    final JsonNode child = node.get(Constants.TYPE_FIELD);
    if (child != null) {
      if (child.isTextual()) {
        return Constants.PASS_TYPE.equals(child.asText());
      }
    }
  }
  return false;
}
```

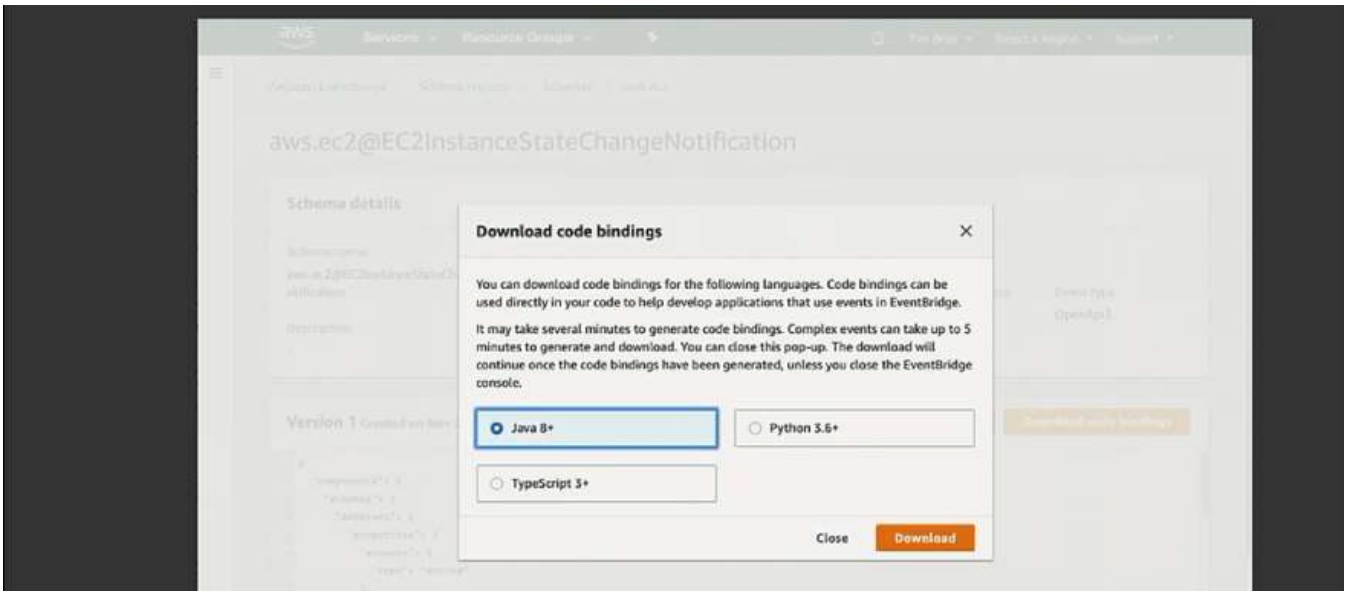
Complexity means you might have to do things like above, finding what type your event payload is made of and then checking for fields in it. This is an anti-pattern.

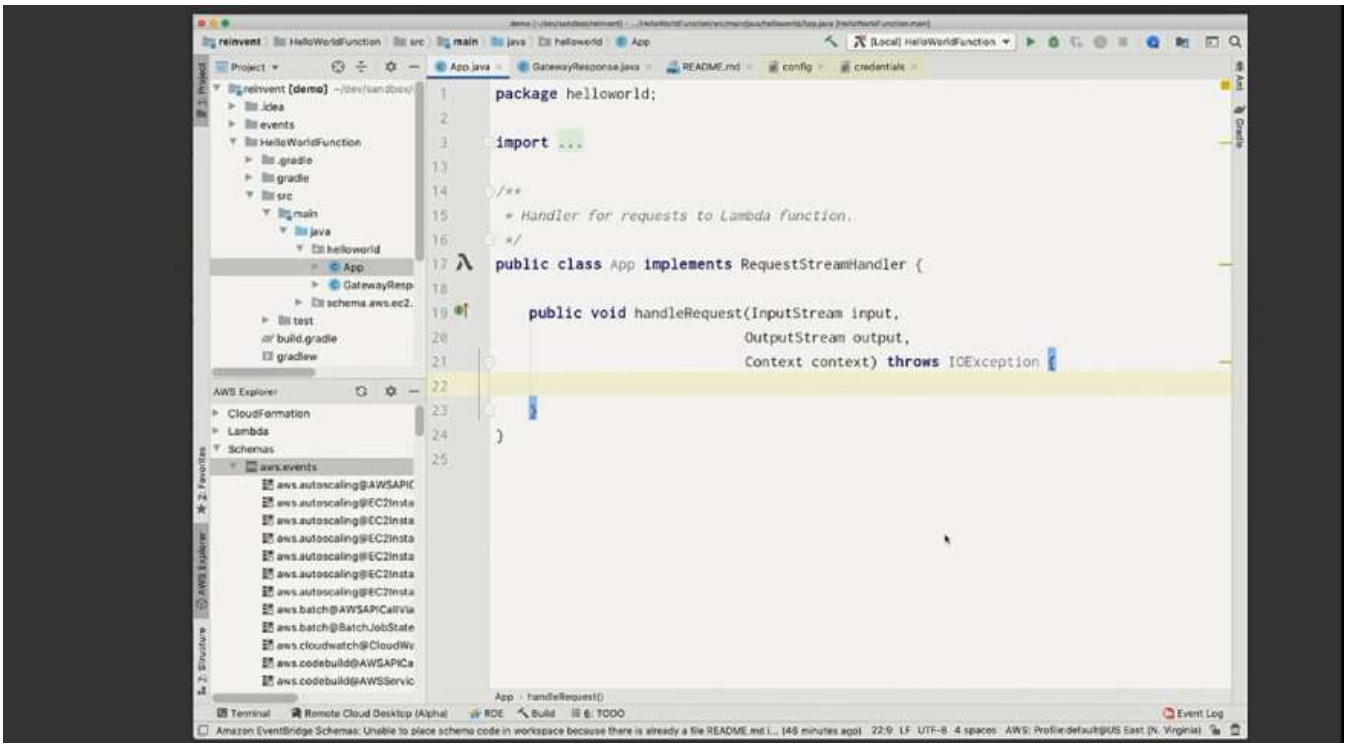


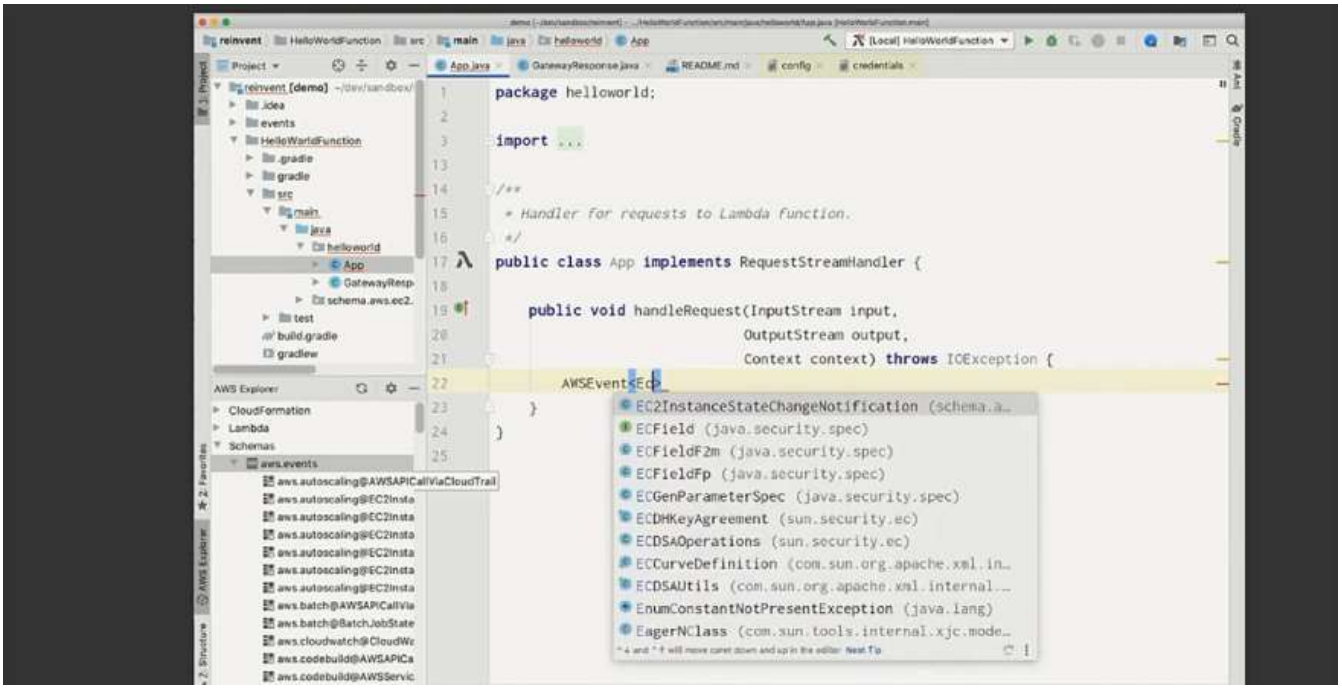
We can use a schema registry to store schemas you use and make things easier. You can also use Athena to automatically generate your schemas for you automatically and upload them into the service registry.

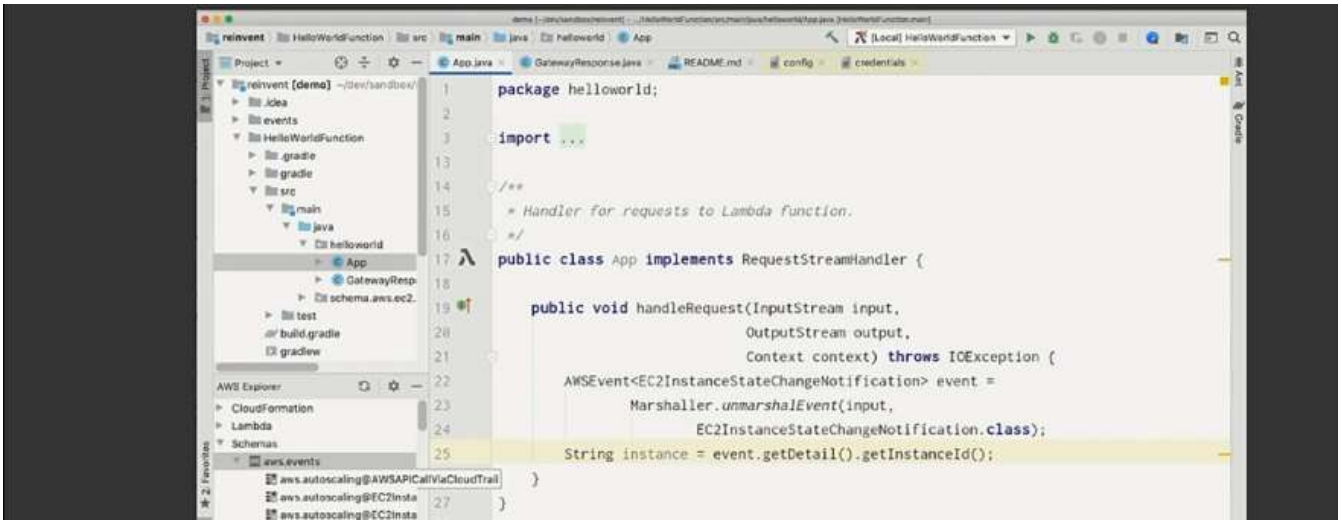

Services
Resource Groups
Tim Bray
Select a Region
Support
Amazon EventBridge > Schema registry > Schemas
Schemas Info
Create registry
Create schema
A schema defines the structure and content of events that are passed on an event bus in Amazon EventBridge. You can browse or search for the schemas of all AWS services on EventBridge. You can automatically generate schemas for events on an event bus, create or upload custom schemas, and organize your custom schemas in custom registries.
All schemas
AWS event schema registry
Discovered schema registry
Custom schema registry
Search all schemas
ec2 instance state change
aws.ec2@EC2InstanceStateChan...
aws.opsworks@OpsWorksInstan...
aws.ssm@EC2StateManagerInst...
aws.emr@EMRInstanceFleetStat...
aws.dlm@DLMPolicyStateChange
aws.emr@EMRInstanceGroupSta...
aws.codedeploy@CodeDeployIn...
aws.emr@EMRClusterStateChange
aws.codepipeline@CodePipeline...
Found in version 1
Feedback
English (US)
Privacy Policy
Terms of Use


Services
Resource Groups
Tim Bray
Select a Region
Support
Amazon EventBridge > Schema registry > Schemas > aws.ec2
aws.ec2@EC2InstanceStateChangeNotification
Schema details
Schema name
aws.ec2@EC2InstanceStateChangeNotification
Last modified
Nov 25, 2019, 12:32 PM PST
Schema ARN
Schema registry
aws.events
Number of versions
1
Event type
OpenApi3
Description
Version 1 Created on Nov 25, 2019, 12:32 PM PST
Action
Download code bindings
Feedback
English (US)
Privacy Policy
Terms of Use


aws.ec2@EC2InstanceStateChangeNotification
Schema details
Download code bindings
You can download code bindings for the following languages. Code bindings can be used directly in your code to help develop applications that use events in EventBridge.
It may take several minutes to generate code bindings. Complex events can take up to 5 minutes to generate and download. You can close this pop-up. The download will continue once the code bindings have been generated, unless you close the EventBridge console.
Java 8+
Python 3.6+
TypeScript 3+
Close
Download


reinvent
HelloWorldFunction
src
main
java
helloworld
App
App.java
GatewayResponse.java
README.md
config
credentials
reinvent [demo]
.idea
events
HelloWorldFunction
.gradle
gradle
src
main
java
helloworld
App
GatewayResp
schema.aws.ec2.
test
build.gradle
gradlew
AWS Explorer
CloudFormation
Lambda
Schemas
aws.events
aws.autoscaling@AWSAPIC
aws.autoscaling@EC2Insta
aws.batch@AWSAPICallVia
aws.batch@BatchJobState
aws.cloudwatch@CloudWa
aws.codebuild@AWSAPICa
aws.codebuild@AWSServic
package helloworld;
import ...
/**
* Handler for requests to Lambda function.
*/
public class App implements RequestStreamHandler {
public void handleRequest(InputStream input,
OutputStream output,
Context context) throws IOException {
}
}
App › handleRequest()
Terminal
Remote Cloud Desktop (Alpha)
Build
6: TODO
Event Log
Amazon EventBridge Schemas: Unable to place schema code in workspace because there is already a file README.md i... (48 minutes ago)
22:9
LF
UTF-8
4 spaces
AWS: Profile:default@US East (N. Virginia)





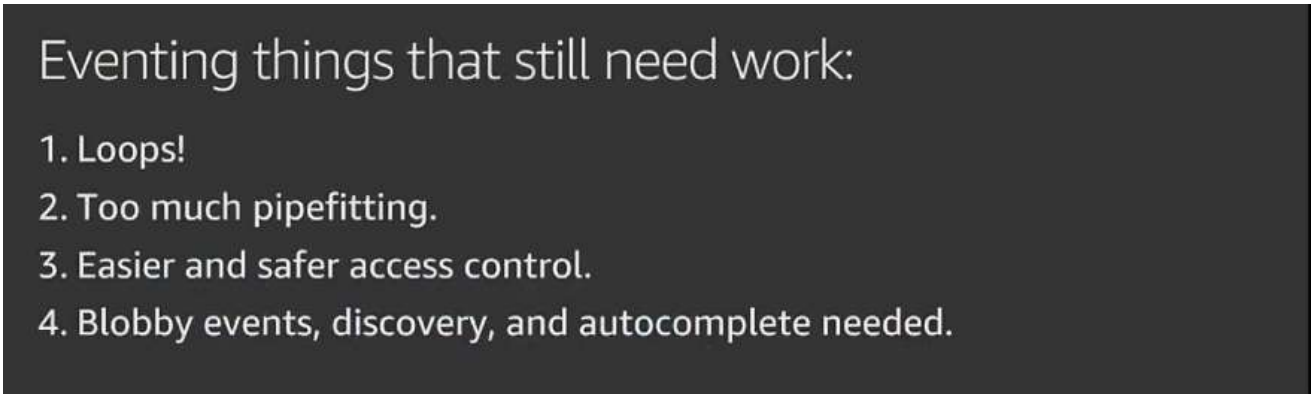
Event facets

	ActiveMQ	Artemis	Event Bridge	Kafka	Amazon Kinesis	RabbitMQ	Amazon SNS	Amazon SQS	Amazon SQS FIFO
Ordering	Yes	Yes	No	Yes	Yes	Yes	No	No	Yes
Dedupe	*	Yes	No	Yes	No	No	No	No	Yes
P2p vs. Pub/sub	P/S	P/S	P/S	P/S	P/S	P/S	P/S	P2p	P2p
Push vs. Pull	Both	Both	Push	Pull	Pull	Both	Push	Pull	Pull
Serverless	No	No	Yes	No	Yes	No	Yes	Yes	Yes
Filtering	Yes	Yes	Yes	*	No	No	Yes	No	No

*: "It's complicated"

API thinking is associated with synchronous thinking, but we can use asynchronous features to it like making async call or using futures or doing request/response while doing things. You can also now fire off an API call and get the response back from a different queue.

You can also use a Service Mesh to help your work with APIs. These are smart proxies that allows you to skip calling APIs with DNS or IP addresses and instead call the generic service name like ‘Customer Info Service’.



Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development

Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures

Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

aws training and certification

Thank you!

Tim Bray

timbray@amazon.com
@timbray
https://tbray.org

re:Invent

aws