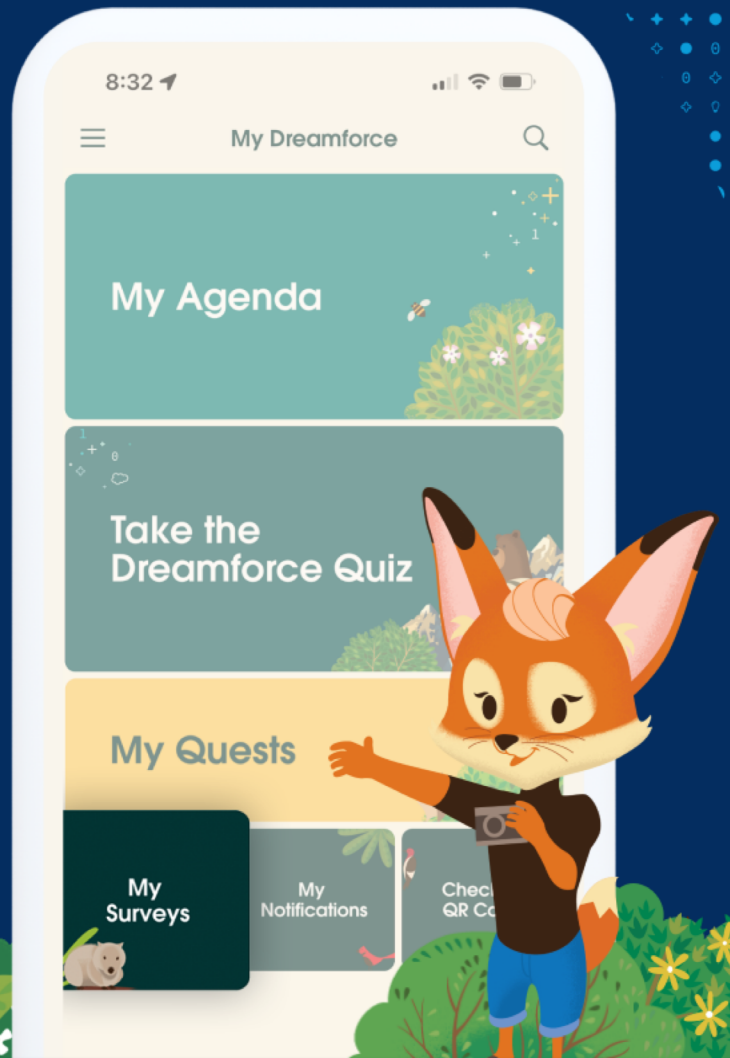
Provide Your Feedback for a Chance to Win.

For every session survey submitted, you will be entered to win one of 15 passes to Dreamforce 2024.*

1. Open the Salesforce Events mobile app.
2. Navigate to My Dreamforce.
3. Select My Surveys.
4. For every quick session survey you take, you'll be entered to win.*

* Restrictions apply. See rules at sforce.co/session-survey-terms



dreamforce

Fundamentals for Designing Event Driven Integrations

Ramanathan Pachaiyappan

Sr.Salesforce Solution Architect

Fujitsu North America Inc
He/Him

https://www.linkedin.com/in/ramanathansj

Agenda

- Introduction
- Why Event Driven Integration?
- Fundamental patterns for reliable integration
- Salesforce Platform Queues (custom) & Multi Cloud Demo
- Salesforce Platform Publish Subscribe (PubSub API) Integration & Multi Cloud Demo
- Q&A

dreamforce

Introduction

Why Event Driven Integration?

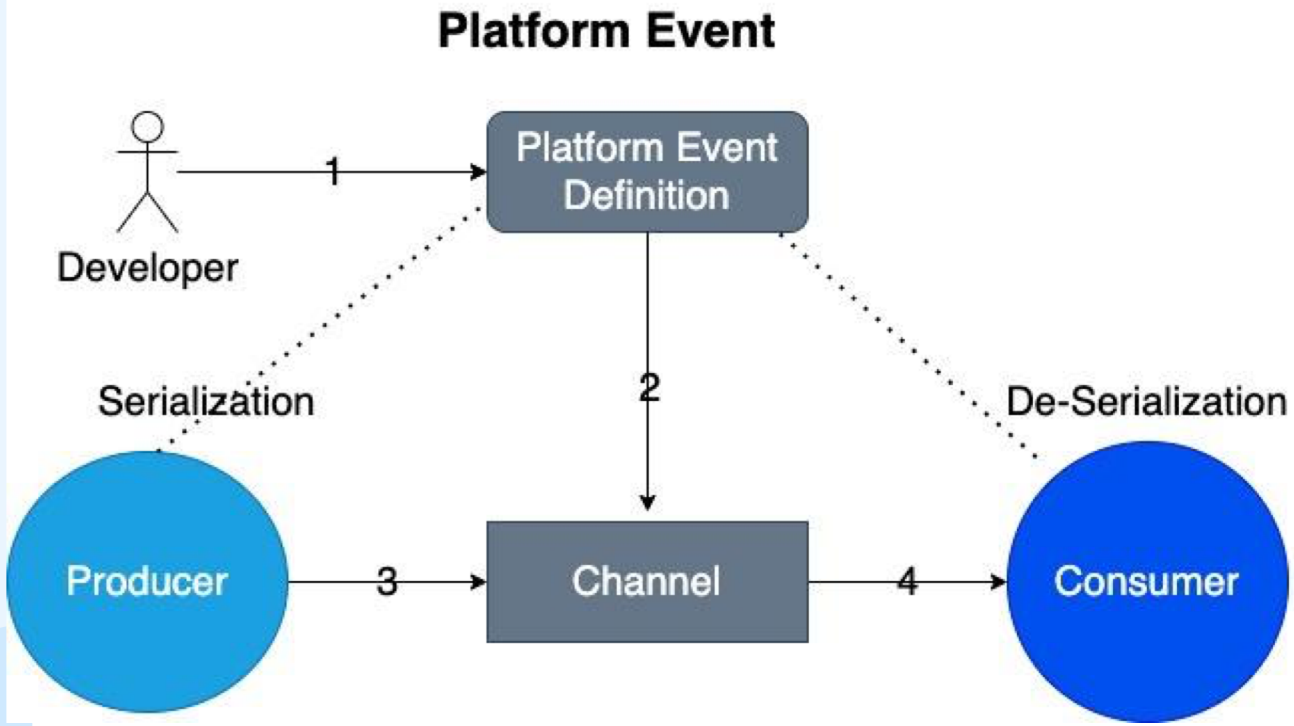
- Designing applications reactive to business events vs passive message applications
- High volume, Performance and Scalability
- Avoid duplicate integrations/APIs for new clients
- Change management, tightly coupled integration, technical debt and not able to modernize current system
- Pub/Sub API based on gRPC, HTTP/2 binary protocol is faster than HTTP/1.1 text protocol

Platform Events

A meaningful state change notification for interested subscribers

- Message - Sender knows receiver & responsible for delivery
- Event - Sender publishes to channel, not responsible for delivery
- Platform Event - Custom, Standard, Data events
- Benefits - Scale, loosely coupled, reactive, etc
- Challenges - complexity, state management, rollback, big picture

Platform Event- Behind the Scene



SETUP
Platform Events

Standard Fields

Action	Field Label	Field Name	Data Type
	Created By	CreatedBy	Lookup(User)
	Created Date	CreatedDate	Date/Time
	Event UUID	EventUuid	Text(36)
	Replay ID	ReplayId	External Lookup

Custom Fields & Relationships

New

Action	Field Label	API Name	Data Type	Indexed	C
Edit | Del	Payload	Payload__c	Long Text Area(92768)		

Triggers

New

Action	Name	Api Version	Status	Size Without Cor
Edit | Del	OrderInboundTrg	59.0	Active	274

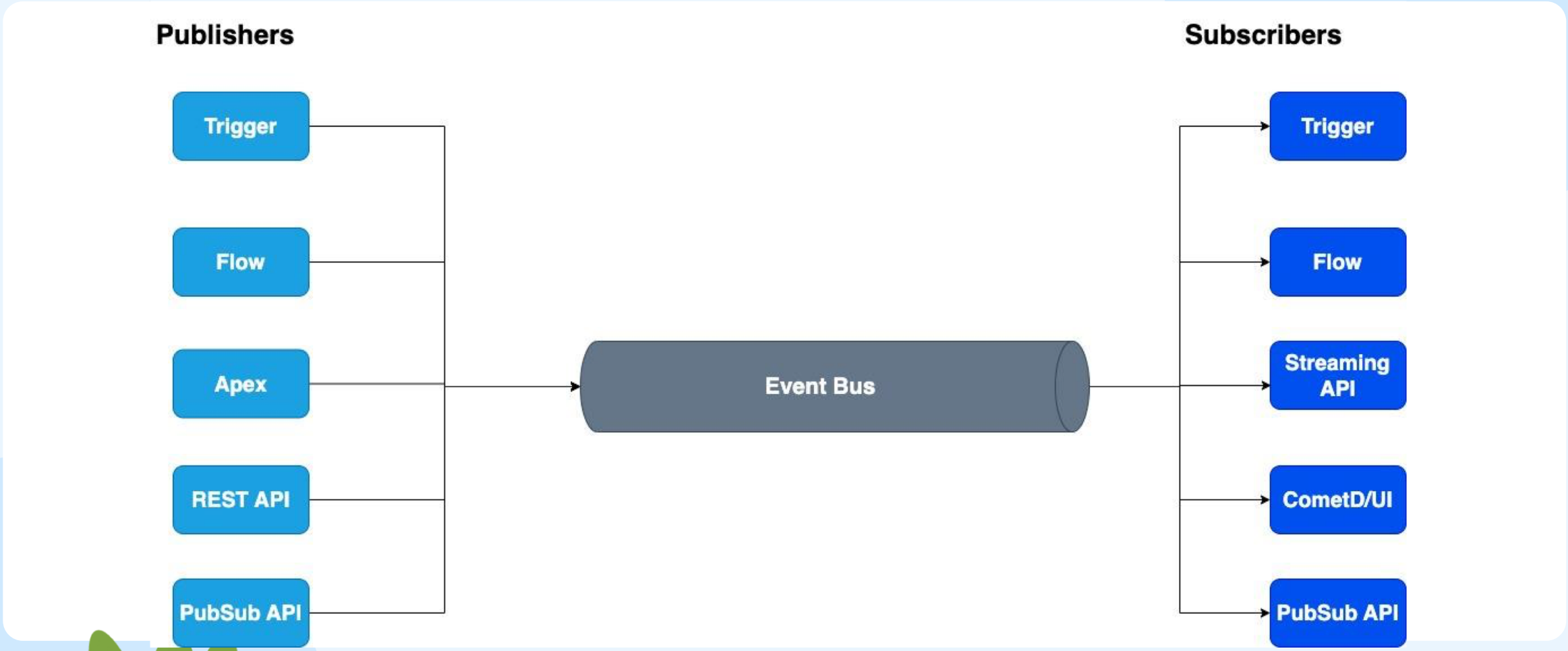
Subscriptions

Action	Subscriber	Last Processed Id
Manage	OrderInboundTrg	1027858

Publish Subscribe Options

- Change Data Capture
- Streaming API
- **Pub/Sub API (we will cover in depth)**
- EMP Connector (CometD)
- lightning:empApi (CometD)

Enterprise Messaging Bus



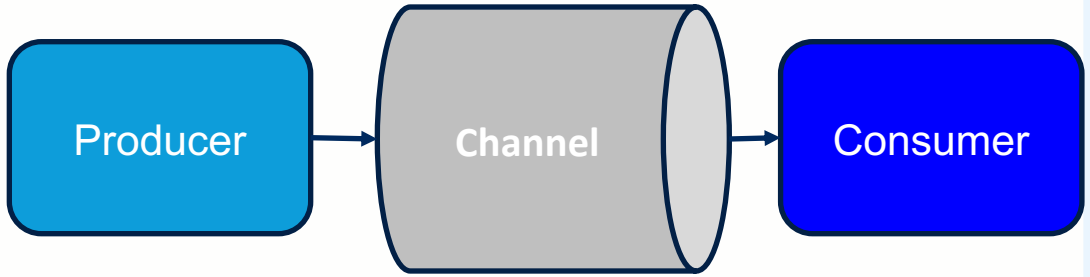
dreamforce

Fundamentals

Message Channel

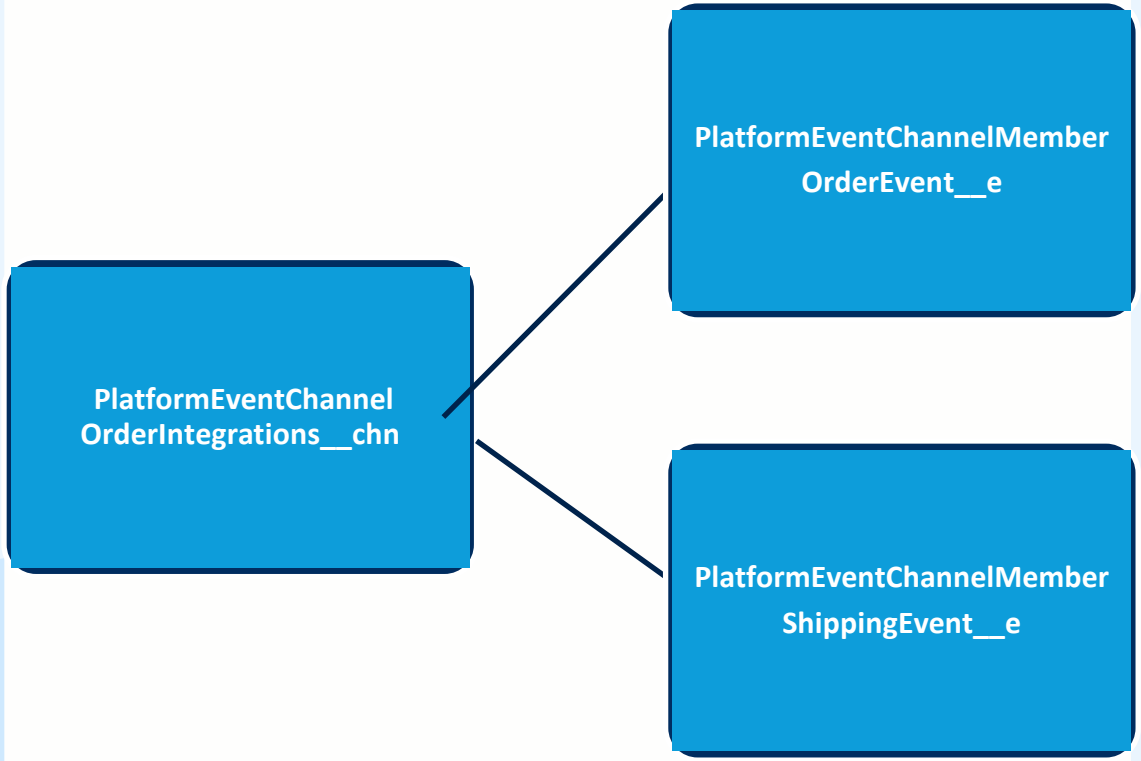
A communication channel between producer and consumer

- Channels Types – Point to Point (Queue) or Topic (Pub/Sub)
- Platform Channels (Pub/Sub) - Standard Channel, Custom Channel & Streaming API Topics (deprecated)
- Point to Point (Queue) Channels - Outbound Message Queues & Custom Solution (Objects, Apex, Flow,etc)



Custom Channel - Topics

Single channel to consume many platform events unlike Standard channels



POST - /services/data/v58.0/tooling/sobjects/PlatformEventChannel

```
{
"FullName": "
OrderIntegrations__chn",
"Metadata": {
"channelType": "event",
"label": "Common Channel for
Order Integrations"
}}
```

```
{
"id":
"0v8RM0000000N6uTAE",
"success": true,
"errors": [],
"warnings": [],
"infos": []
}
```

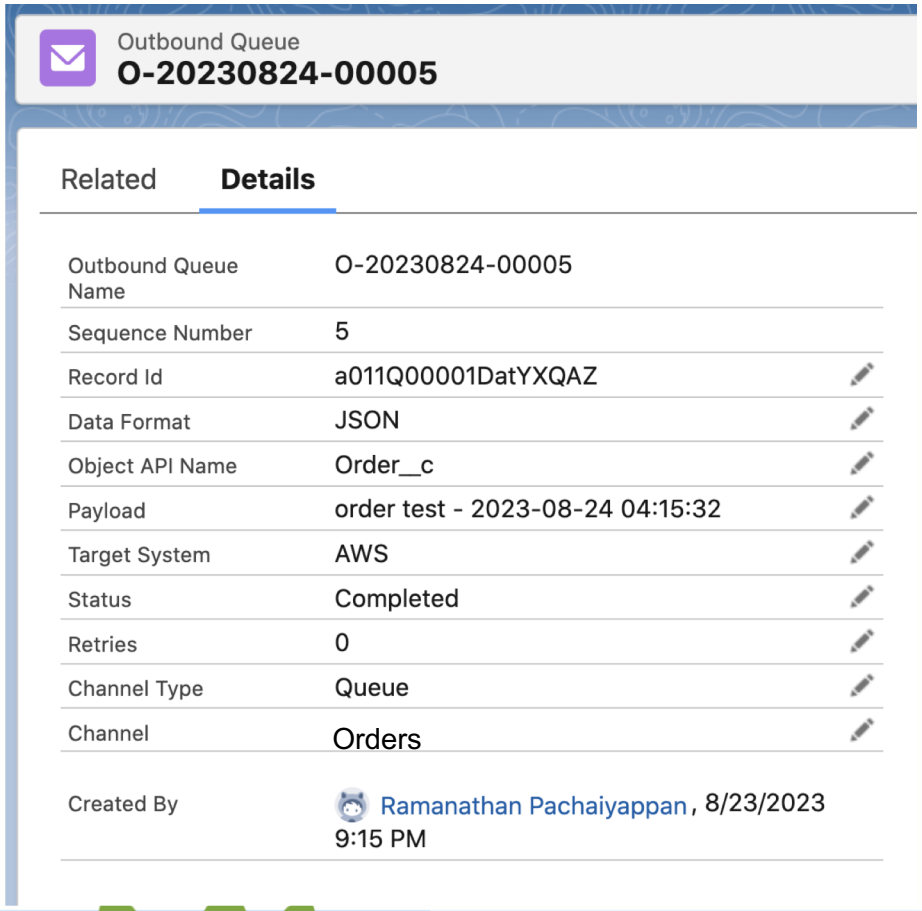
POST - /services/data/v58.0/tooling/sobjects/PlatformEventChannelMember

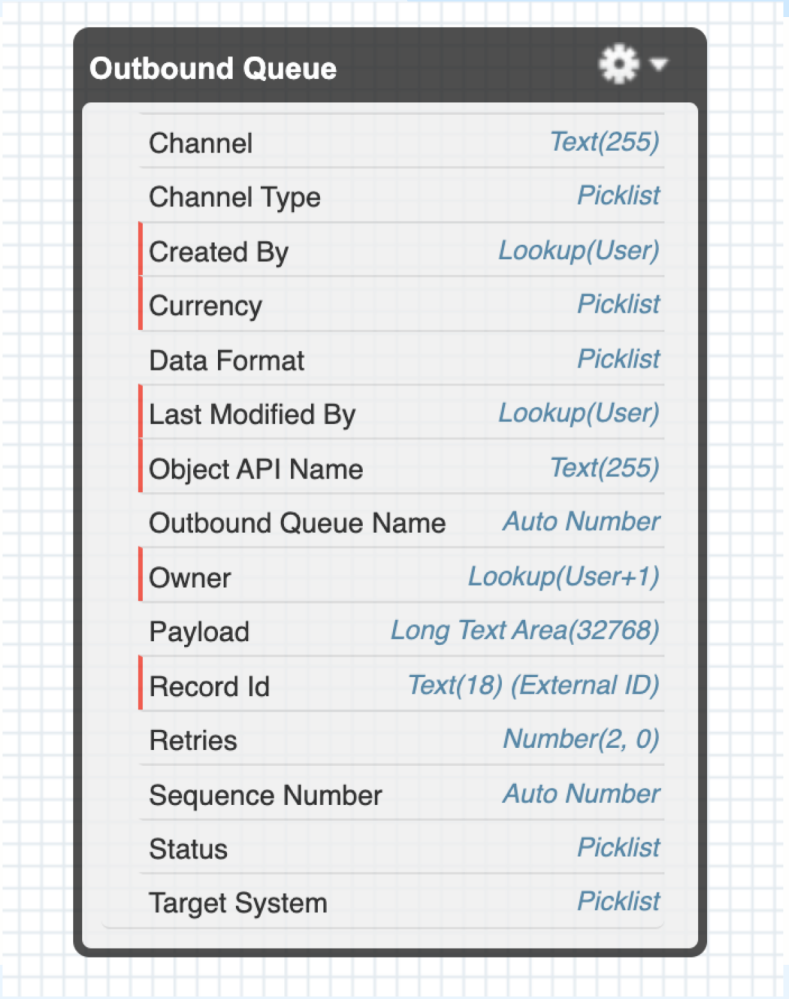
```
{
"FullName": " OrderIntegrations _chn_
OrderEvent_e",
"Metadata": {
"eventChannel": " OrderIntegrations
__chn",
"selectedEntity": " OrderEvent__e"
}}
```

```
{
"id": "0v8RM0000004VPJWA2",
"success": true,
"errors": [],
"warnings": [],
"infos": []
}
```

Custom Channel - Queues

Utilizing Custom Object as common message channel for different integration





Message Schema

A canonical model representing message/events

- Different services can talk to each other by converting request/response be to common canonical format
- Salesforce provides rich Metadata Describe API which can be used as schema
- JSON message schemas are other options.

GET
/services/data/v58.0/sobjects/OrderInboundEvent__e/eventSchema?payloadFormat=compact


```
{
  "name": "OrderInboundEvent__e",
  "namespace": "com.sforce.eventbus",
  "type": "record",
  "fields": [
    {
      "name": "Payload__c",
      "type": [
        "null",
        "string"
      ],
      "doc": "Data:StringPlusClob:00NB000000Giy0J",
      "default": null
    }
  ],
  "uuid": "pyTbOLAPGYIvlLAqGKQUow"
}
```

dreamforce

Queues

Asynchronous Request Reply

Queue based pattern to decouple producer and consumer

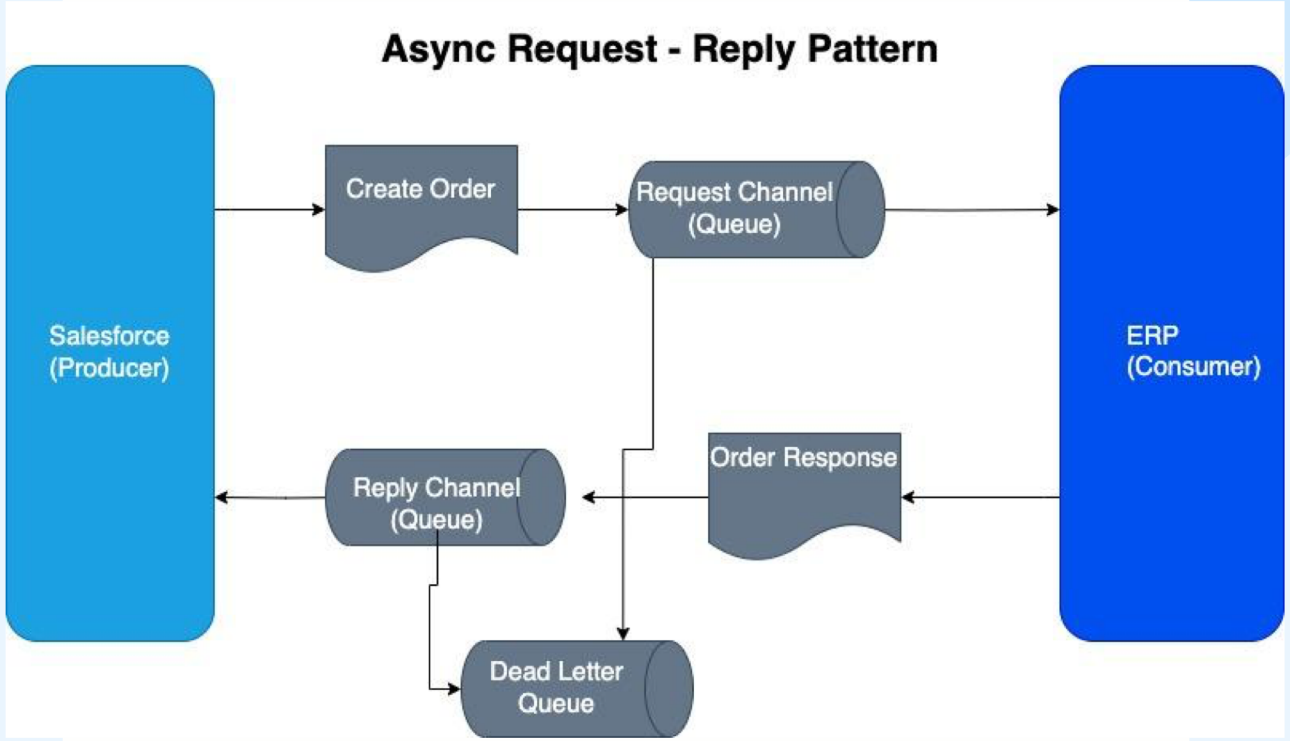
Pros

- Loosely coupled, Asynchronous & Scalable
- Failover and retry mechanism
- Flow control & batching

Cons

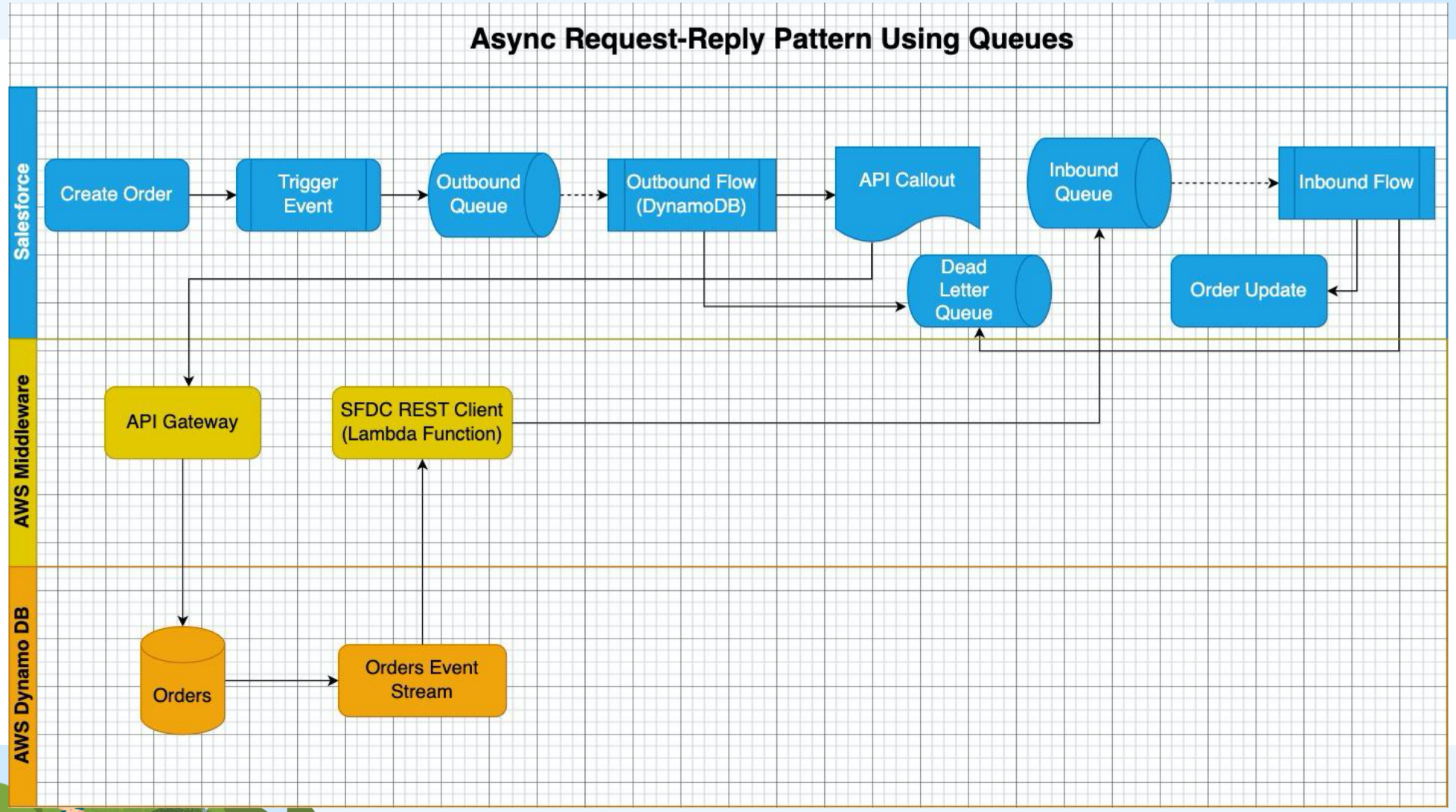
- Not a real time & No rollback support
- Requires end user education on SLAs
- Significantly complex design

Note : This pattern can also be implemented using events instead of messages



Demo #1

Salesforce Order integration with AWS DynamoDB

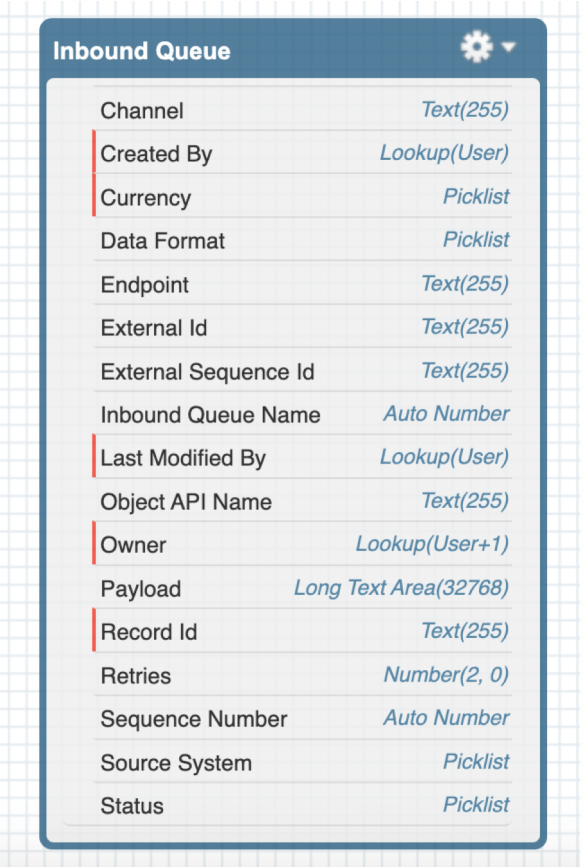


Inbound Channels – Demo Walkthrough

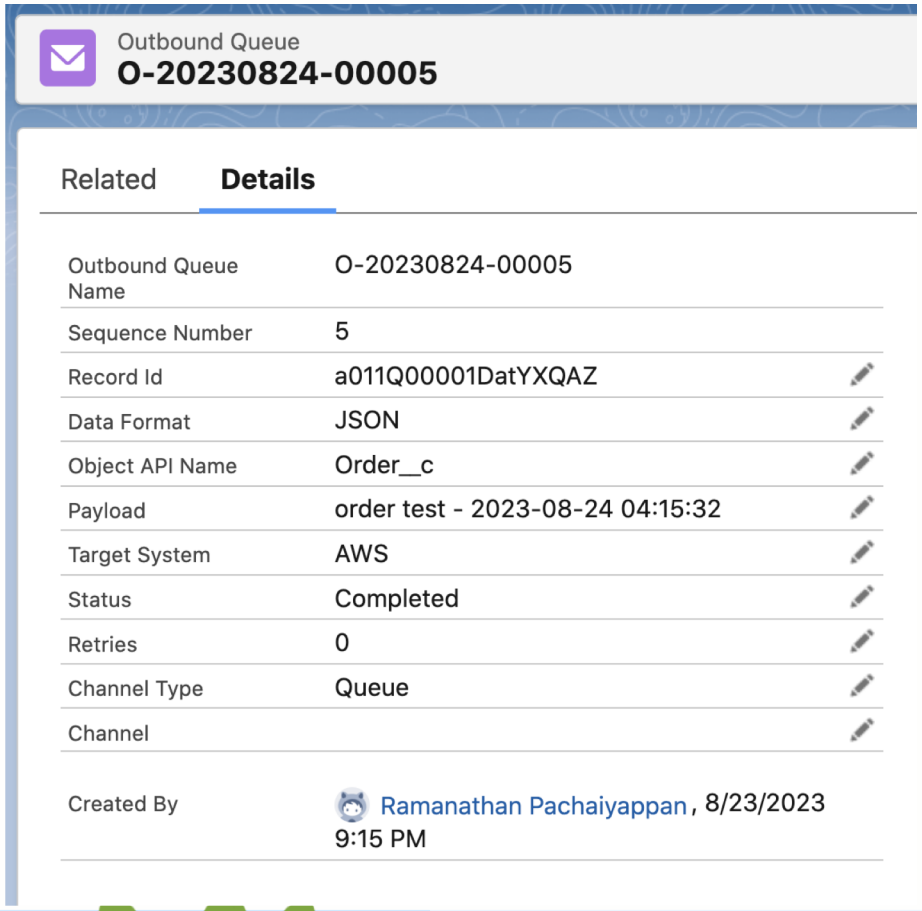
Inbound Queue
I-20230824-00011

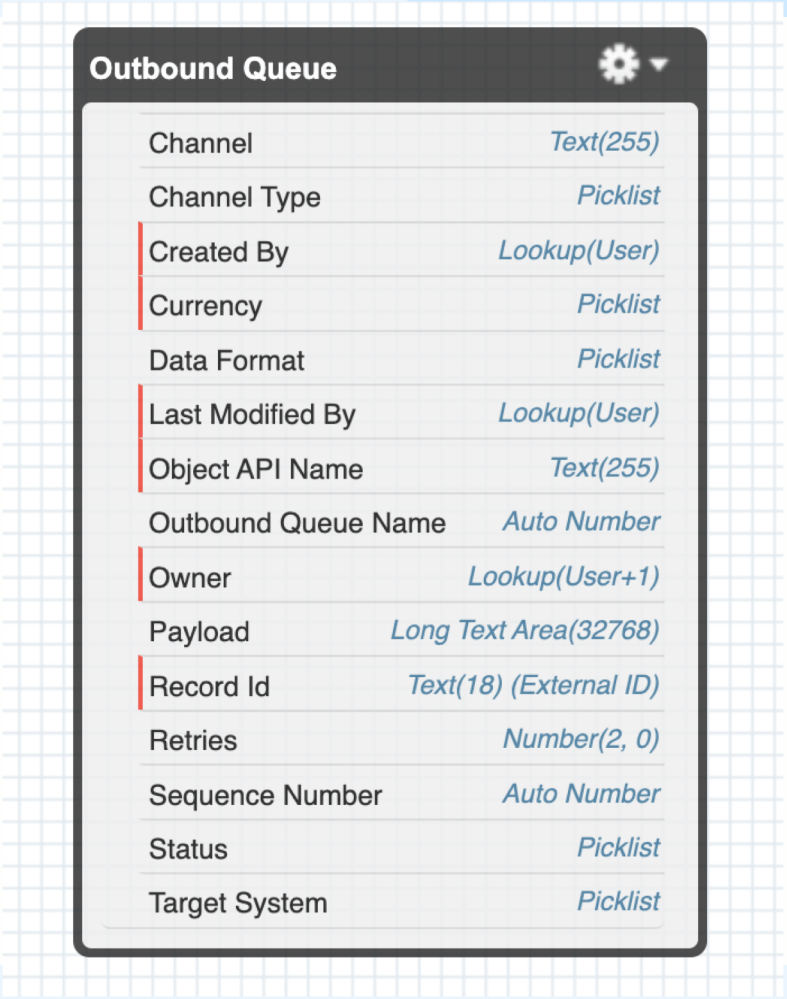
Related | **Details**

Inbound Queue Name	I-20230824-00011
Object API Name	Order__c
Payload	order test - 2023-08-24 04:15:32
Data Format	JSON
Sequence Number	11
Record Id	a011Q00001DatYXQAZ
Endpoint	https://hjewgvy6f73p774zsrqtklygxu0lfpno.lambda-url.us-east-1.on.aws/
Source System	AWS
Status	Completed
Retries	
External Id	5
External Sequence Id	867643000000000041187940965
Channel	
Created By	Ram P, 8/23/2023 9:15 PM

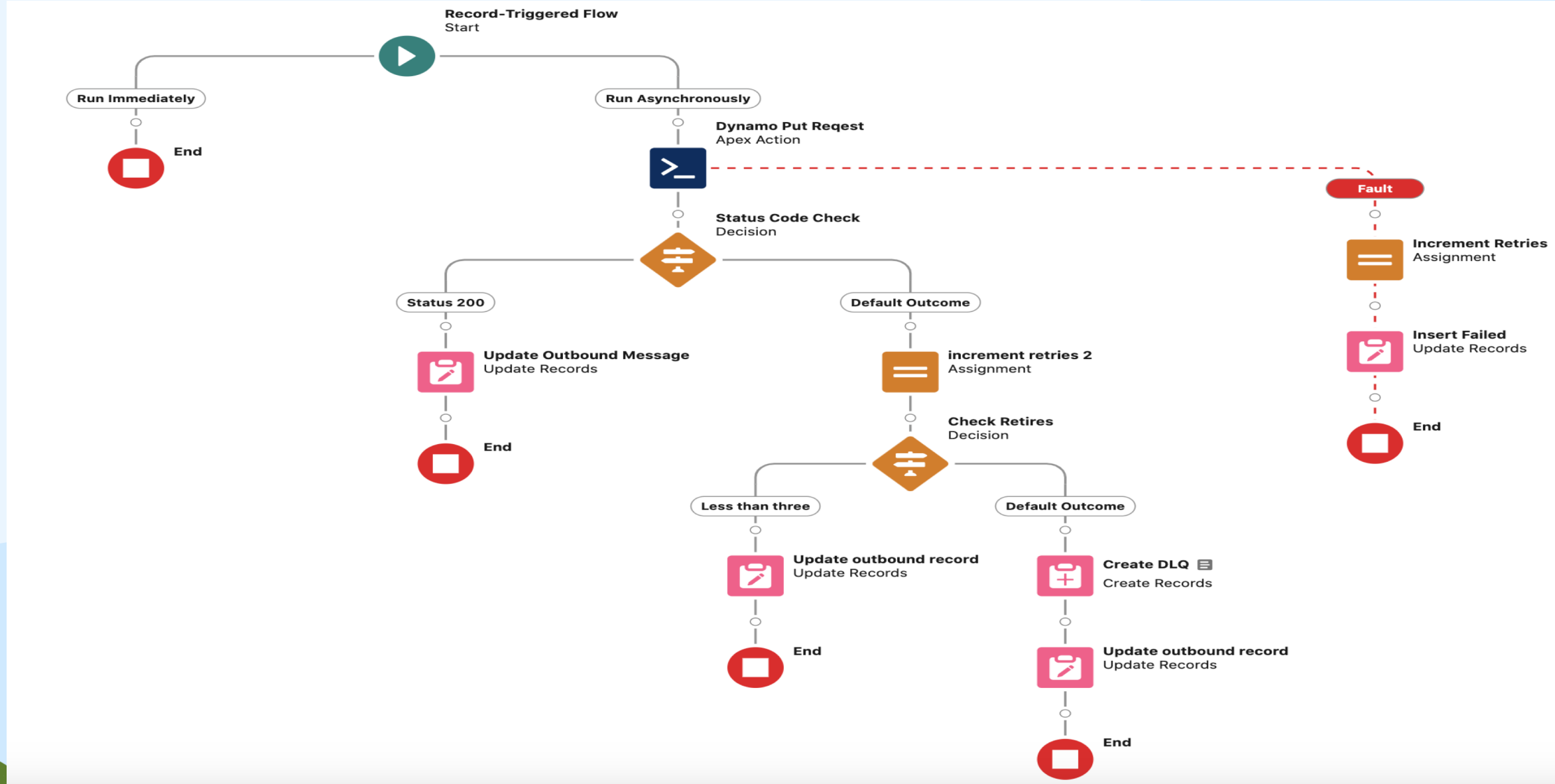


Outbound Channels - Demo Walkthrough





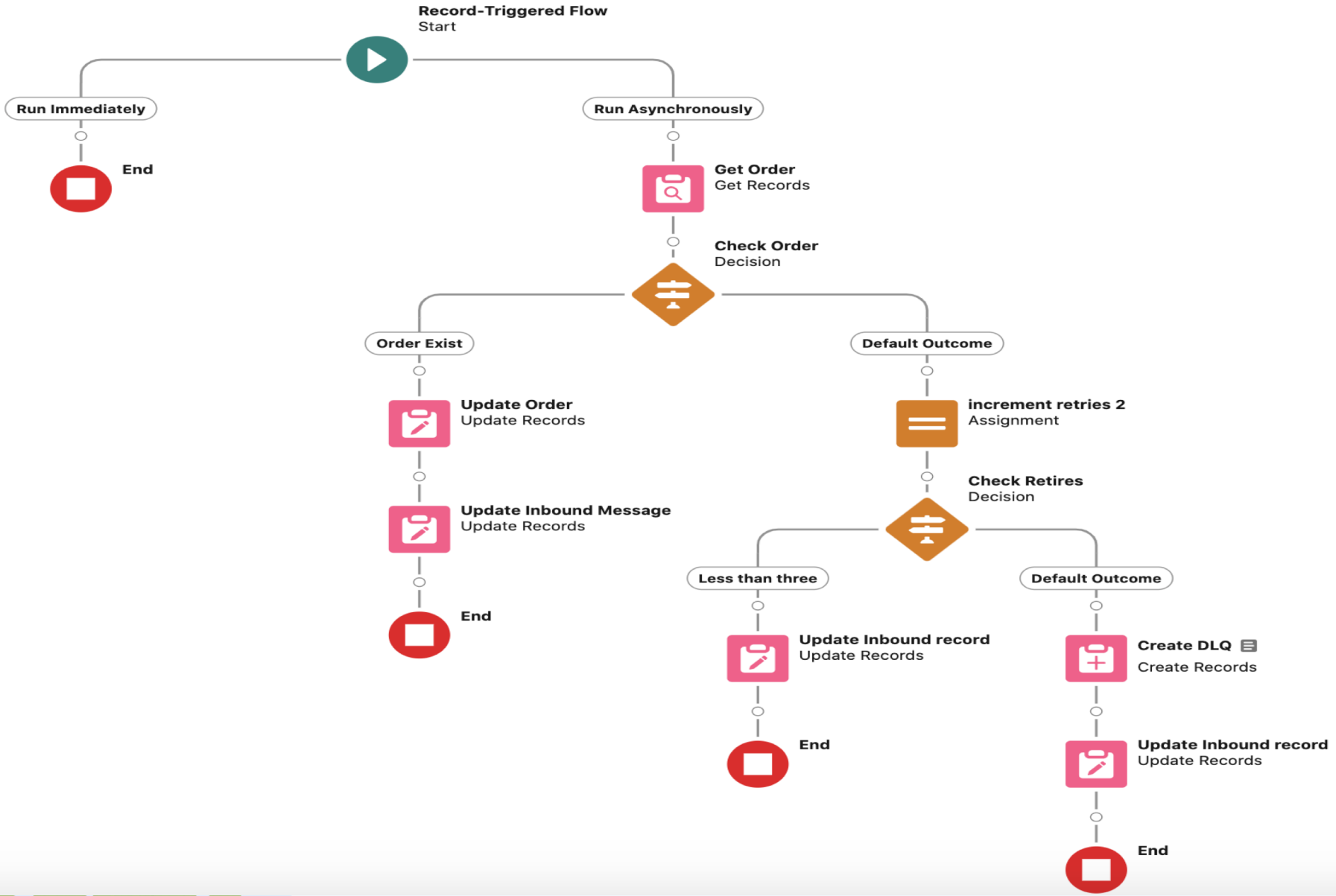
Outbound Service Flow - DynamoDB



Outbound Service Flow - DynamoDB

```
@InvocableMethod(label='Dynamo Put Request' description='Invoke DynamoDB put request' category='DynamoDb')
global static List<DynamoPutResp> putItemRequest (DynamoPutRequest[] dynamoReq) {
    HttpRequest req = new HttpRequest();
    HttpResponse res = new HttpResponse();
    Http http = new Http();
    req.setEndpoint('callout:AWS_Gateway_V1');
    req.setMethod('POST');
    req.setHeader('Content-Type', 'application/json');
    String body = JSON.serialize(dynamoReq[0]);
    req.setBody(body);
    req.setTimeout(20000);
    res = http.send(req);
    DynamoPutResp dynamoRsp = new DynamoPutResp();
    dynamoRsp.response = res.getBody();
    dynamoRsp.statusCode = res.getStatusCode();
    dynamoRsp.status = res.getBody().length() < 5 ? res.getStatus() : 'FAILED';
    return new List<DynamoPutResp> {dynamoRsp};
```

Inbound Service Flow – DynamoDB



Lambda Function

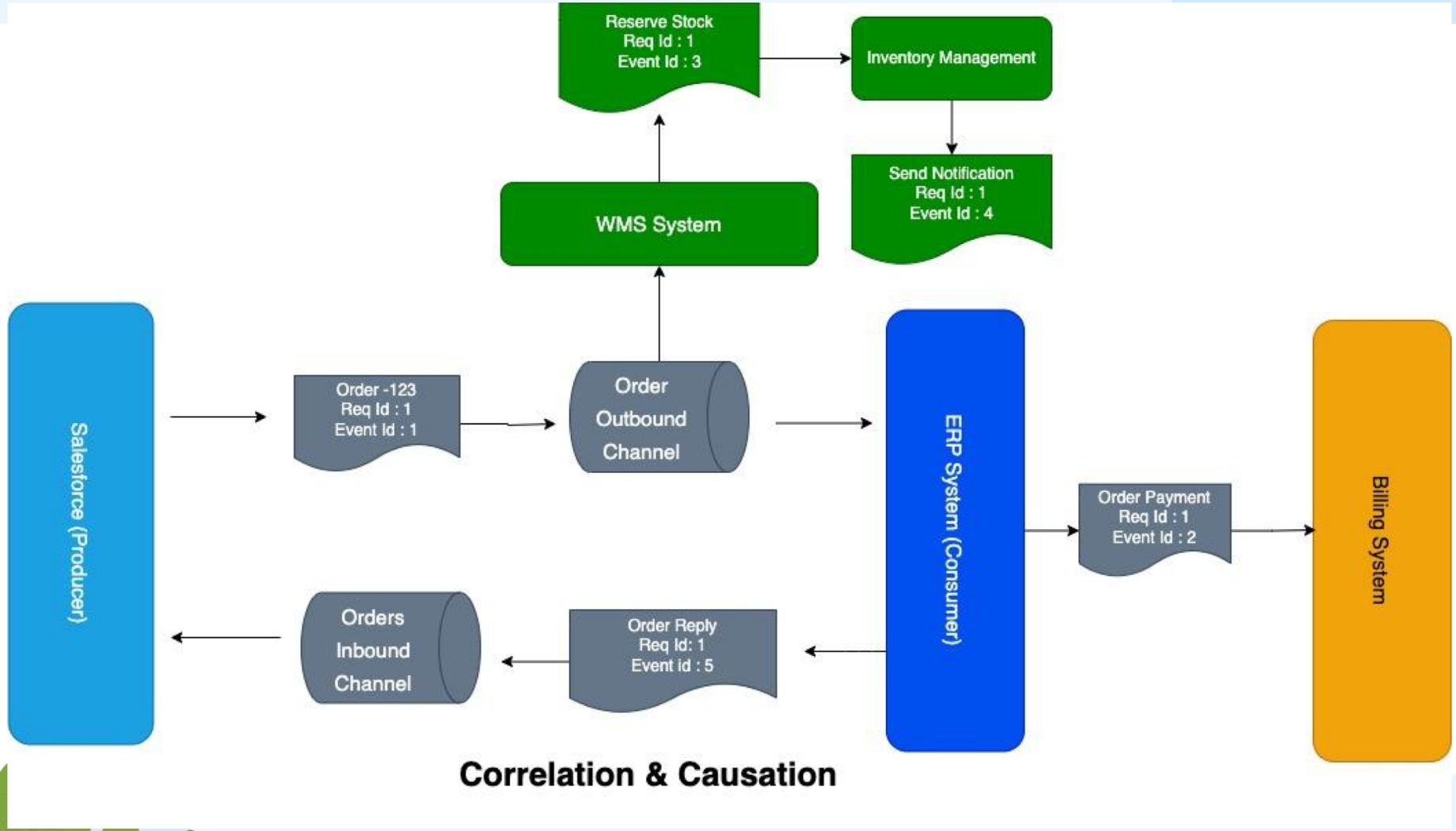
```
function createinboundObj(evtdata) {
  let inboundObj = {};
  if (evtdata["NewImage"]) {
    let record  = AWS.DynamoDB.Converter.unmarshall(evtdata.NewImage);
    inboundObj = {
      ExternalId__c: record.id,
      ObjectAPIName__c: "Order__c",
      Payload__c: JSON.stringify(record.orderdata),
      RecordId__c: record.externalId,
      SourceSystem__c: "AWS",
      Status__c: "Pending",
      ownerId: "005B0000005GMN7IAO",
      ExternalSequenceId__c: record.SequenceNumber,
      RequestLookup__c:(record.requestId.length <=18 ? record.requestId : null),
      CorrelationId__c : record.requestId,
      Channel__c : "Orders"
    };
  }
  return inboundObj;
}
```

dreamforce

More Fundamentals

Correlation Id

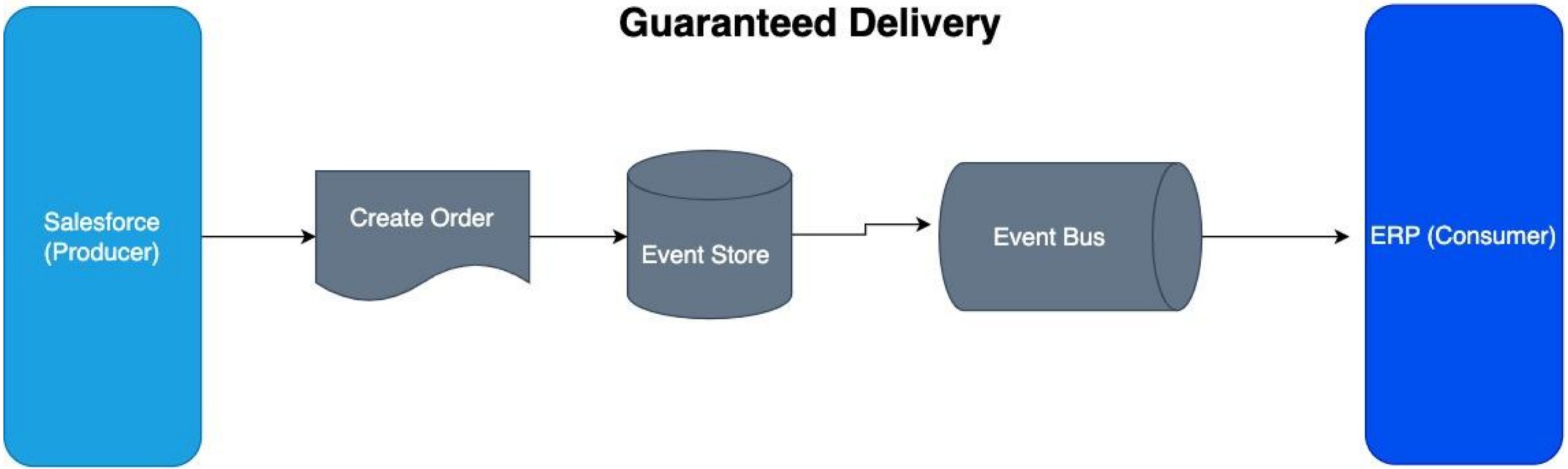
Message level unique Identifiers to track events across different services



Guaranteed Delivery

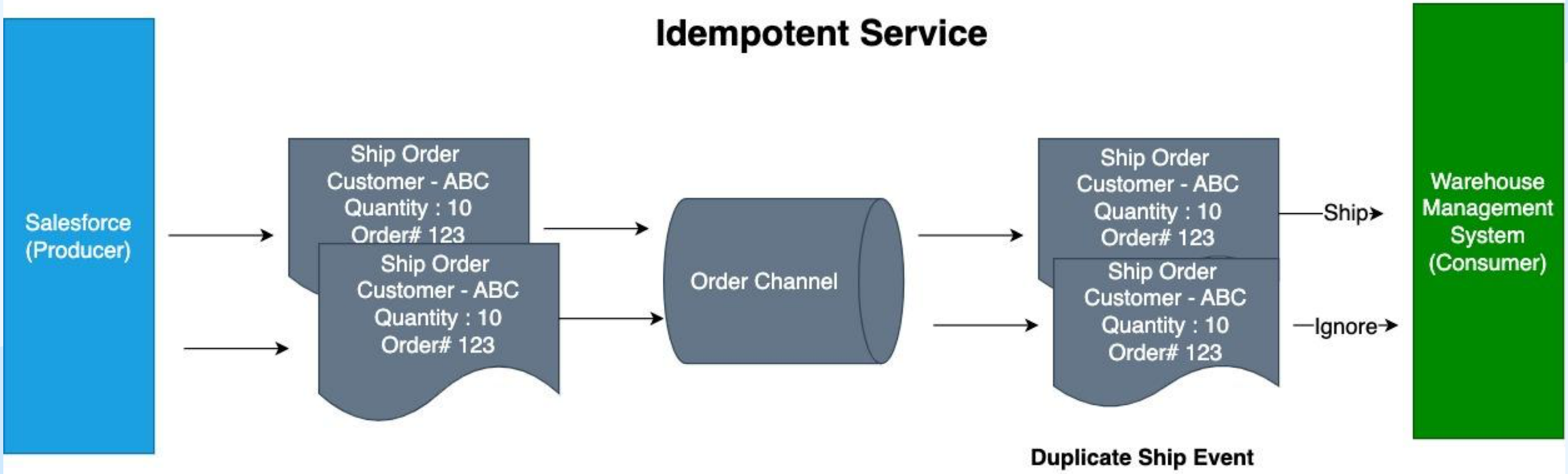
Success of event driven integration relies on delivery

Message delivery Modes - At most once, At least once and Exactly once



Idempotency and Duplicate Messaging

If a function is called with same input multiple times without impacting results is an Idempotent function



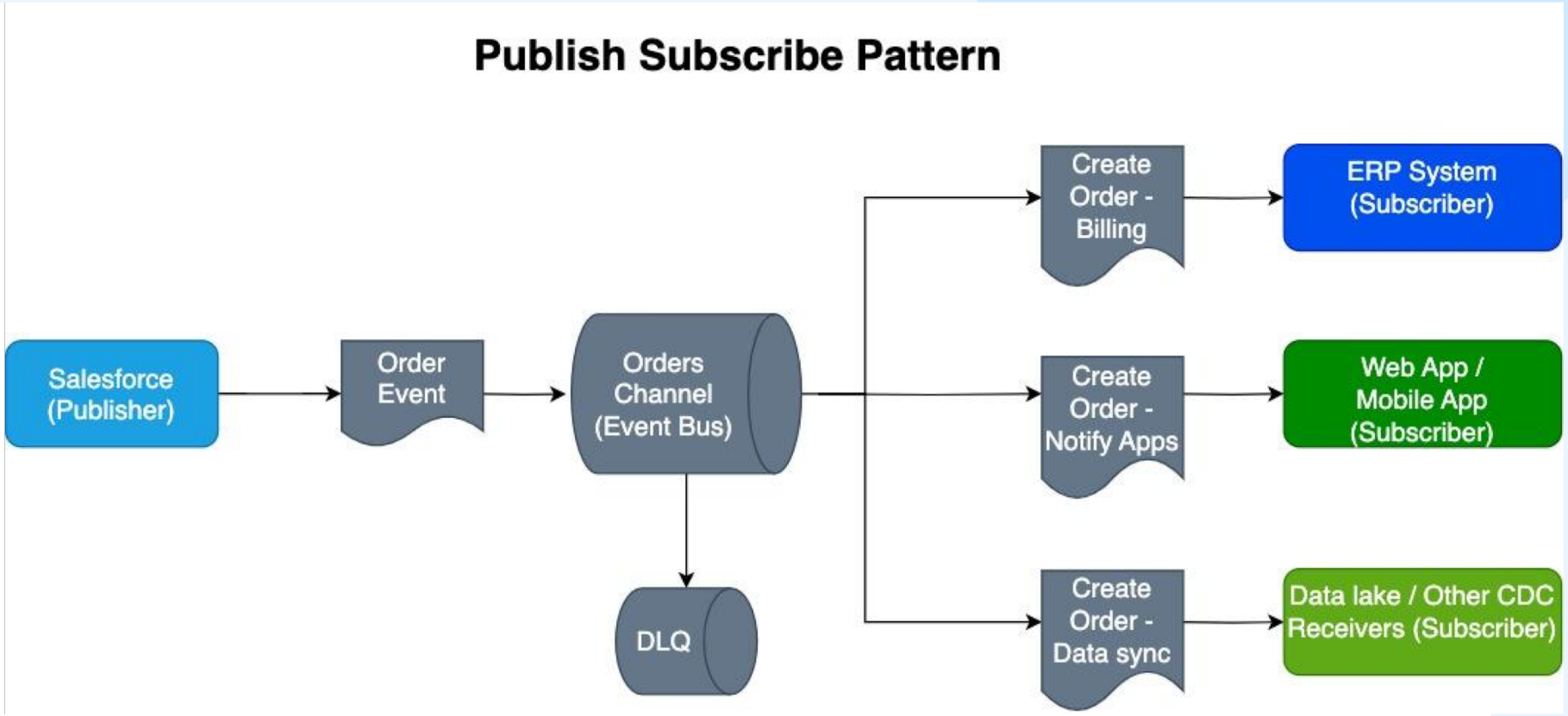
dreamforce

Topics

Publish-Subscribe Pattern

A broadcast messaging to notify all interested parties about state change

- Observer Pattern
- Pull or Push Subscription
- Asynchronous, Loosely Coupled, Change management
- Onboard new client painlessly



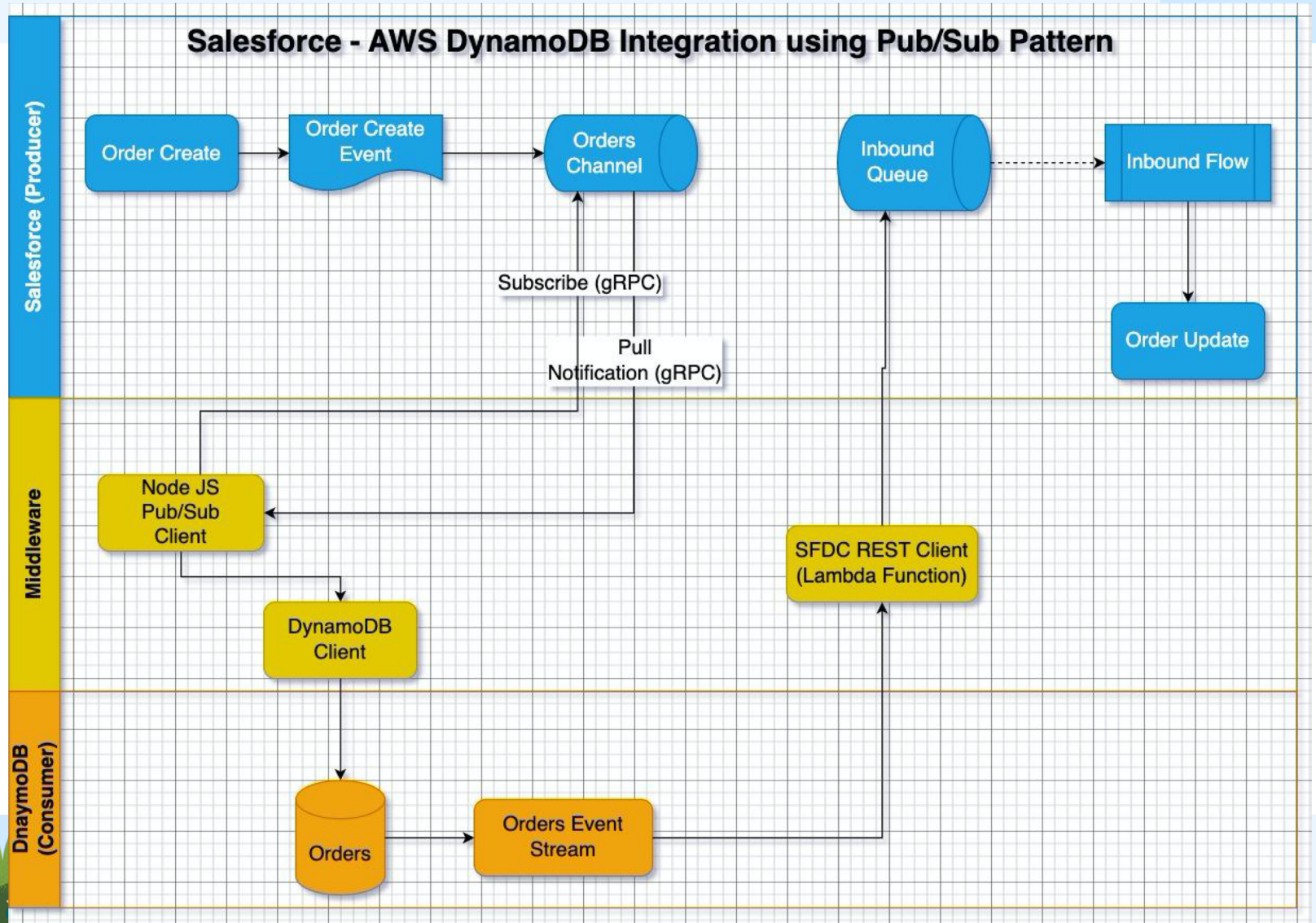
Salesforce Pub/Sub API Overview

Unified API for publish and subscribing to events

- Built using gRPC, HTTP/2 & Protocol Buffer for fast and efficient delivery of binary data
- Protocol buffer interface definition language (IDL) for describing services and messages
- IDL compiled into Stub/Skeleton classes
- Out of the box clients available
- CDC (Change Data Capture), Platform Event supported
- High performance integrations

Demo #2

Salesforce Order integration with DynamoDB using Pubsub API/Pattern



Sample IDL Services and Messages

```
message PublishRequest {
  string topic_name = 1;
  repeated ProducerEvent events = 2;
  string auth_refresh = 3;
}

message PublishResponse {
  repeated PublishResult results = 1;
  string schema_id = 2;
  string rpc_id = 3;
}

service PubSub {

  rpc Subscribe (stream FetchRequest) returns (stream FetchResponse);

  rpc Publish (PublishRequest) returns (PublishResponse);

  rpc PublishStream (stream PublishRequest) returns (stream PublishResponse);
}
```

Pub/Sub – Node JS Code

```
async function subscribe(channel, evtcount) {
    let evtdata = {};
    try {
        const client = new PubSubApiClient();
        let chnmeta = await getEventMetadata()
        client.setChannelMeta(chnmeta)
        await client.connect();
        const eventEmitter = await client.subscribe(channel,evtcount);
        await eventEmitter.on('data', async (event) => {
        evtdata = event;
        if (evtdata.payload.MetadataType__c &&
evtdata.payload.MetadataType__c.string == "Order__c") {
            await upsertOrders(evtdata);
        } else {
            console.log('no match found')
        }});
    } catch (error) {
        console.error(error);
    }
    return evtdata;
}
```

Pub/Sub – Describe Channel Members

```
async describeSchema(channel) {
  let authinfo;
      try {
          authinfo = await this.authWithUsernamePassword();
      } catch (error) {
         console.error(error);
      }
  let metacache = {};
  if (authinfo) {
      for await (const member of channel.members) {
          let schema = await this.fetchMetadata(authinfo, member);
          let avroSchm = {
              id : schema.uuid,
              type : avro.parse(schema)
          }
          metacache[schema.uuid] = avroSchm;
      }
  }
  return metacache;
}
```

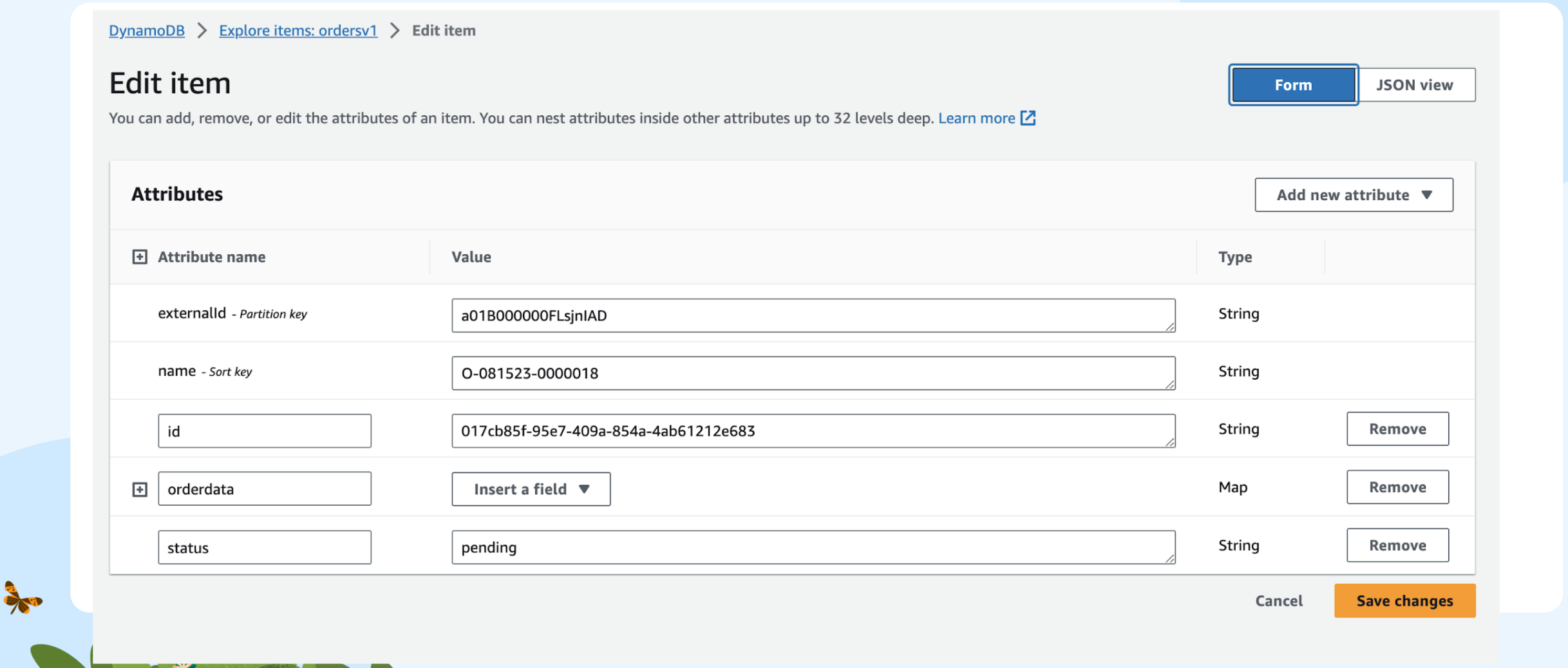
DynamoDB Upsert Orders

```
async upsertOrders(items) {
       let ordbody =
JSON.parse(items.payload.Payload__c.string);
       let itemdata = {
           externalId : ordbody.Id,
           name : ordbody.Name,
           status : "pending",
           id : uuidv4(),
           orderdata :ordbody
       };
       let dbresult =  await
this.executeDbCommand(itemdata, "ordersv1");
       let reqrps = { request : itemdata,
response : dbresult};
       return reqrps;
   }
```

```
async executeDbCommand(items, tblname) {
    console.log("items == "+ JSON.stringify(items));
    let tblcmd = new PutCommand({
        TableName : tblname,
        Item : items
    });
    let response = {}
    try {
        docClient.send(tblcmd).then(
            (data) => {
             return data;
            },
            (error) => {
                console.error("error == "+JSON.stringify(ex))
            }
          );
    } catch (ex) {
        console.error("error == "+JSON.stringify(ex))
    }
    return response;
```

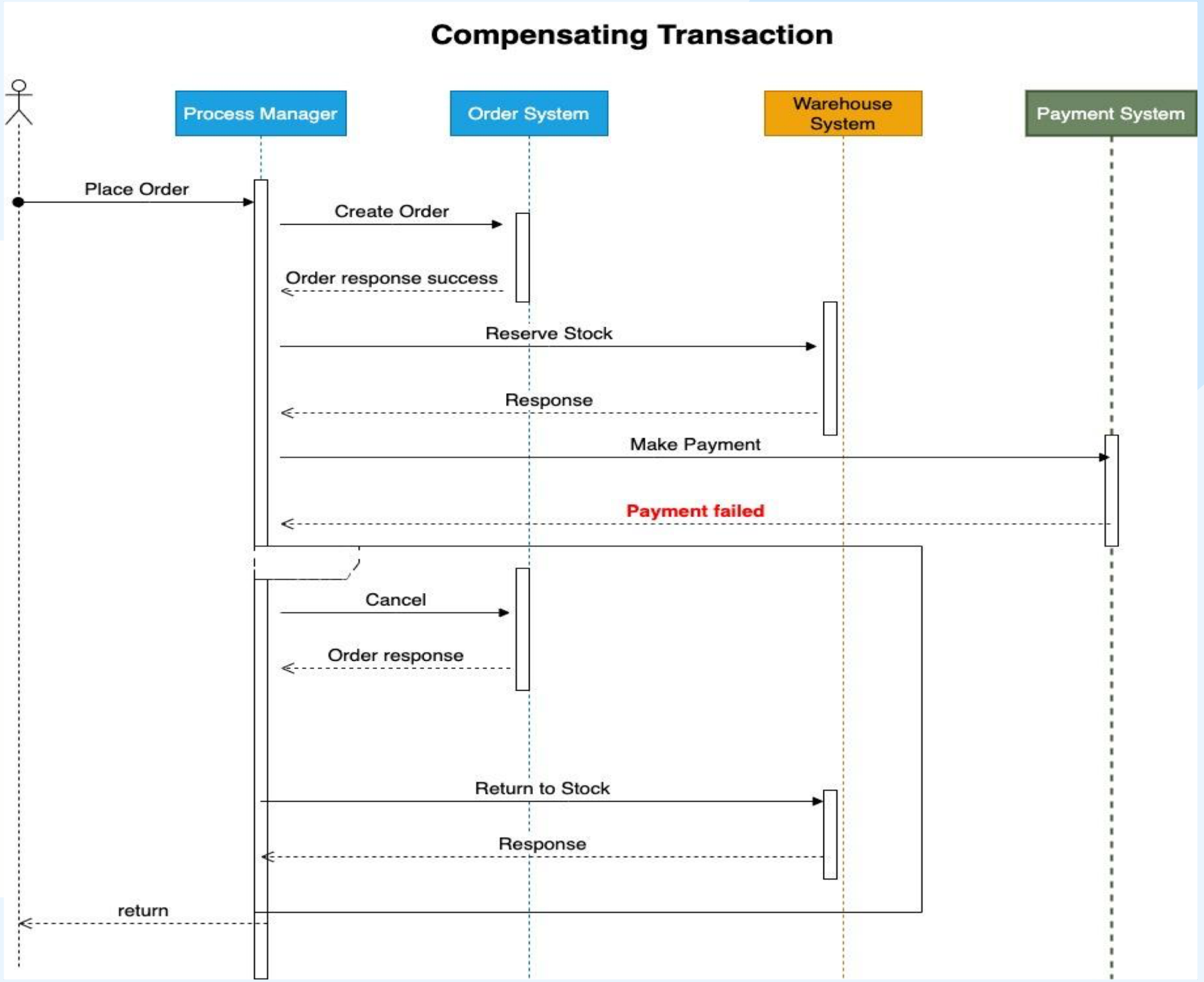
DynamoDB Table

Orders Table



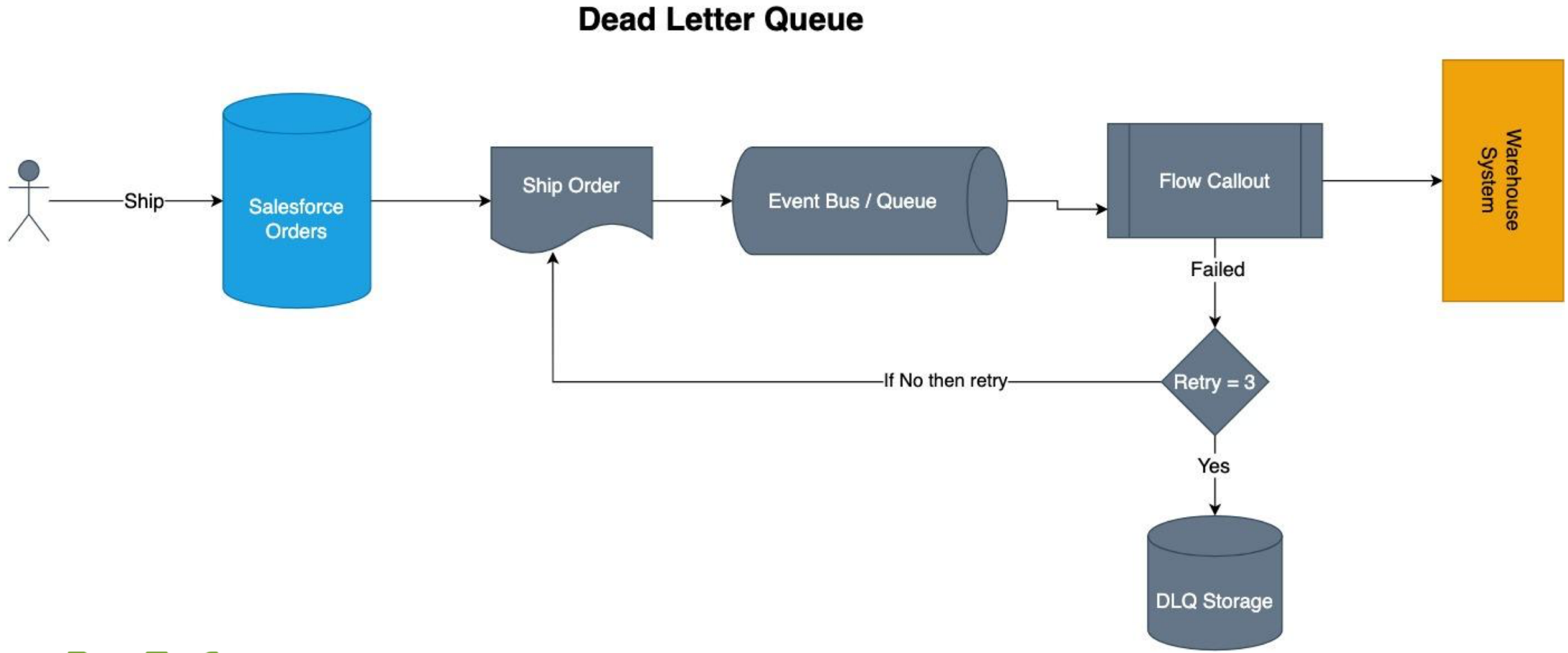
Compensating Transactions

Rollbacks are not possible in distributed applications, what can be done ?



Dead Letter Queue

A temporary queue for storing invalid & non deliverable messages



Wrap Up

Message or Event Driven Integration ?

References

- https://resources.docs.salesforce.com/latest/latest/en-us/sfdc/pdf/integration_patterns_and_practices.pdf
- https://architect.salesforce.com/decision-guides/event-driven
- https://www.enterpriseintegrationpatterns.com/patterns/messaging/index.html
- https://solace.com/blog/event-driven-architecture-difference-event-driven-integration/
- https://docs.confluent.io/platform/current/schema-registry/index.html
- https://developer.salesforce.com/blogs/2020/04/event-driven-app-architecture-on-the-customer-360-platform
- https://cloud.google.com/pubsub/docs/subscriber
- https://www.confluent.io/blog/put-several-event-types-kafka-topic/

Slides & Source Code Repo

DynamoDB -> SFDC REST Client (Common for both Async Request-Reply and Pub/Sub pattern) - https://github.com/ramanathansj/dynamodb-sfdc-client

Salesforce Request Reply pattern using Queues - https://github.com/ramanathansj/salesforce-async-request-reply-queues

Salesforce Pub/Sub API Implementation & Slides - https://github.com/ramanathansj/salesforce-pub-sub

Release Date : 9/17/23 EOD

Thank you

Provide Your Feedback for a Chance to Win.

For every session survey submitted, you will be entered to win one of 15 passes to Dreamforce 2024.*

1. Open the Salesforce Events mobile app.
2. Navigate to My Dreamforce.
3. Select My Surveys.
4. For every quick session survey you take, you'll be entered to win.*

* Restrictions apply. See rules at sforce.co/session-survey-terms