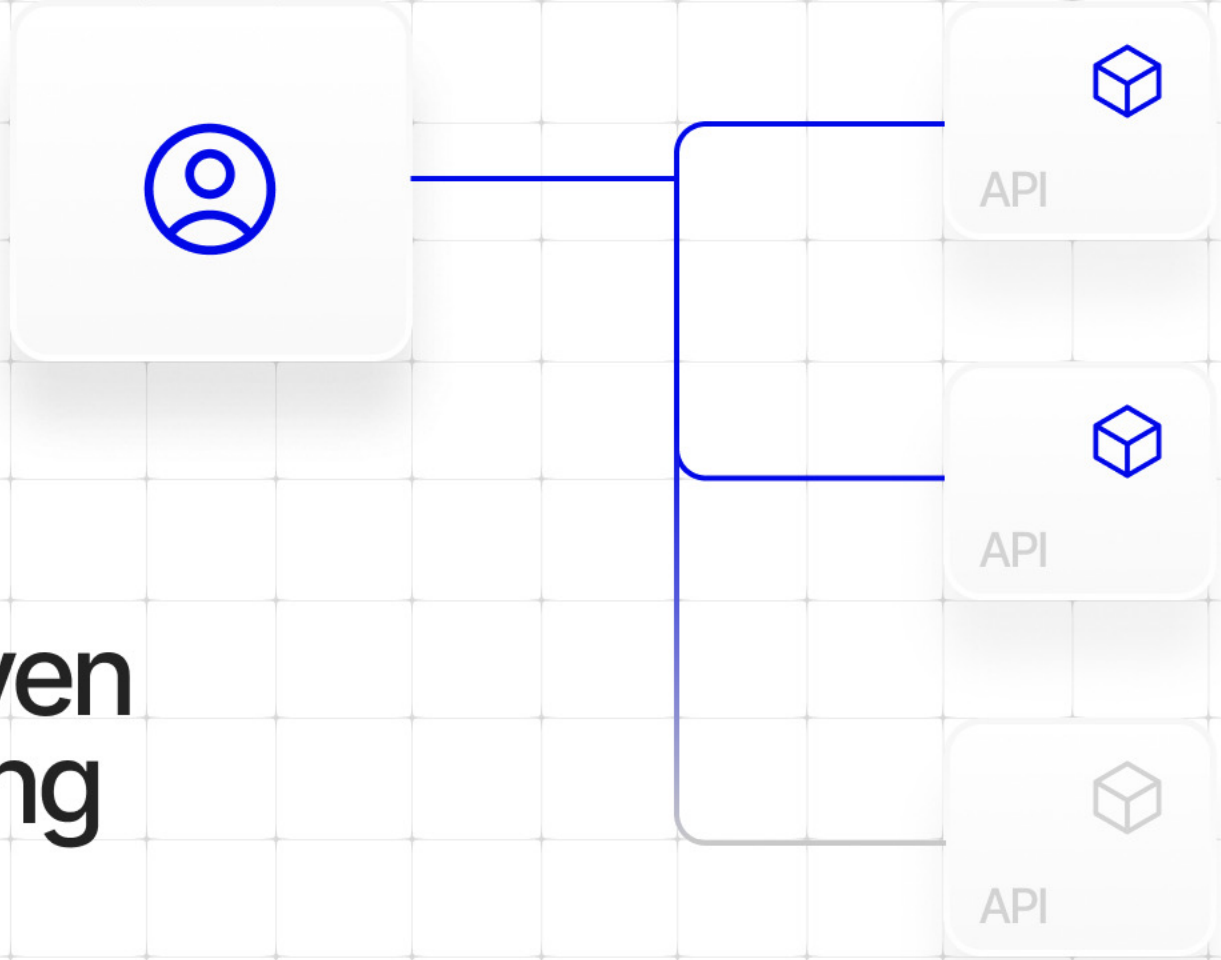
EURO PYTHON PRAGUE &REMOTE
8-14 July 2024



Building Event-Driven Python service using FastStream and AsyncAPI

abhinand-c

EURO PYTHON PRAGUE &REMOTE 8-14 July 2024

@abhinand-c Strollby.com UST

Abhinand C

Product Engineer
UST Strollby


EVENTS,
EVENTS EVERYWHERE!

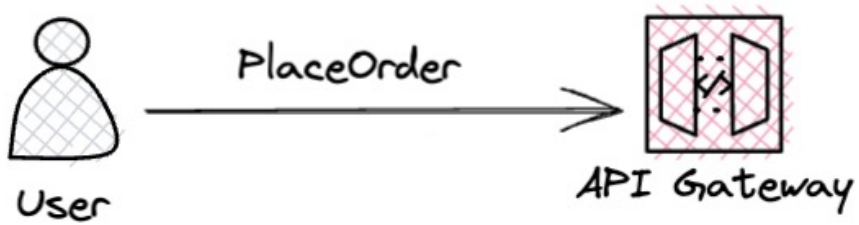
What is an **Event**?

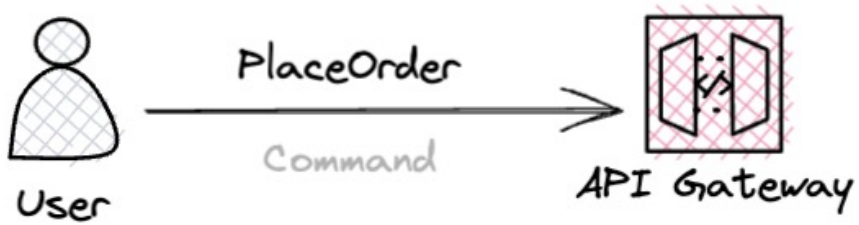
“An **event** is a ***change in state***, or **an update**, like an item being placed in a shopping cart on an e-commerce website.

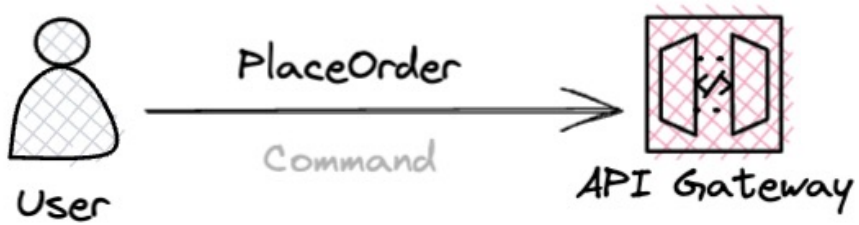
Events can either carry the state or events can be identifiers.

It represents facts that have happened in the system”

Events vs Commands

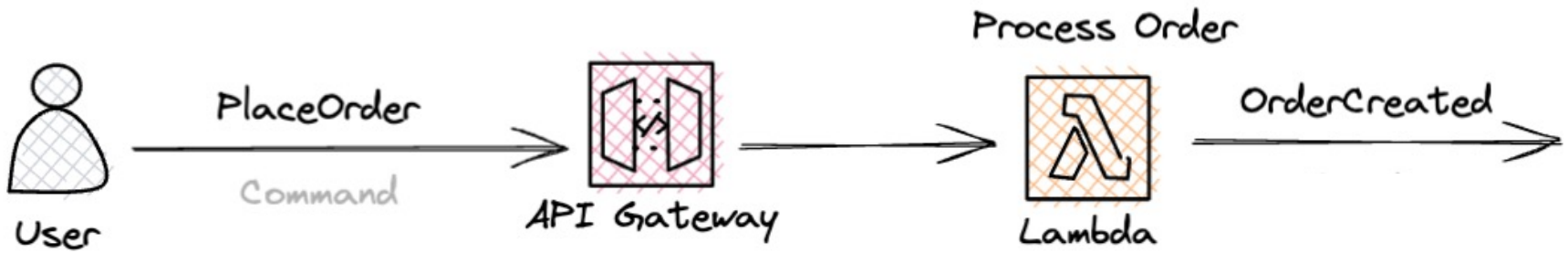






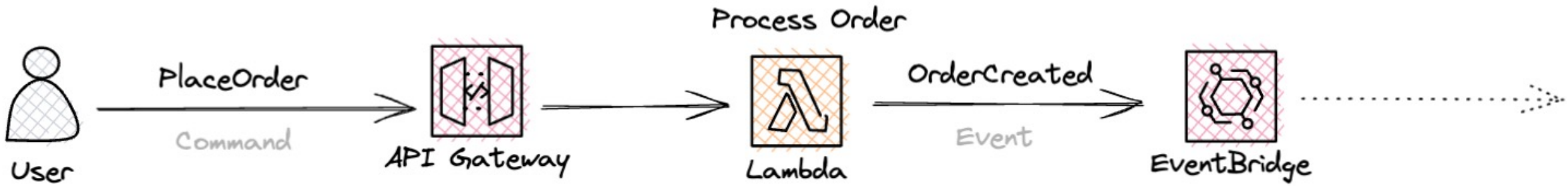
Commands

- Represents the intent
- Object that wants to perform an action
- Ask to do something, failures can happen.
- Can be rejected
- One consumer
- Verb
- Example "PlaceOrder" into SQS



Commands

- Represents the intent
- Object that wants to perform an action
- Ask to do something, failures can happen.
- Can be rejected
- One consumer
- Verb
- Example "PlaceOrder" into SQS



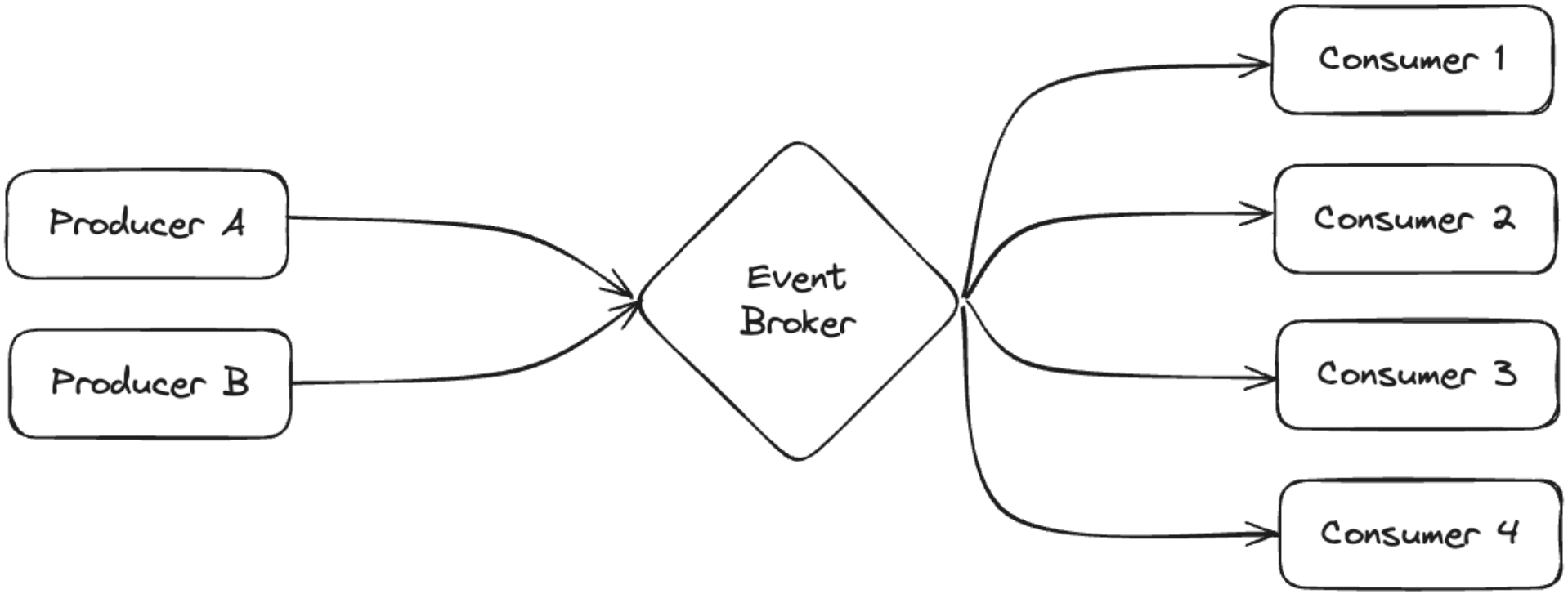
Commands

- Represents the intent
- Object that wants to perform an action
- Ask to do something, failures can happen.
- Can be rejected
- One consumer
- Verb
- Example "PlaceOrder" into SQS

Events

- Facts that have happened
- Immutable information
- Cannot be changed
- Consumers can ignore events
- Many consumers
- Past tense
- Can use events to determine history of transaction
- Example "OrderCreated" into EventBridge

Event Broker



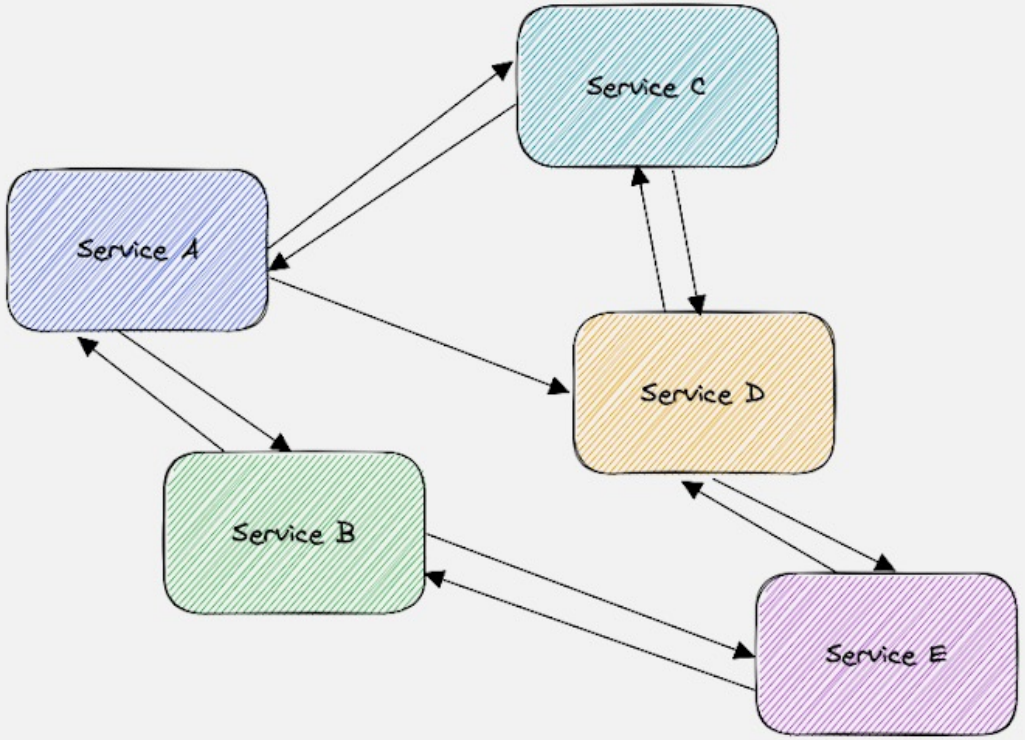
What is **Event Driven Architecture**?

“An Event Driven Architecture is an of software architecture pattern built upon the production, detection, consumption & reaction to events. It uses events to trigger and communicate between decoupled services.”

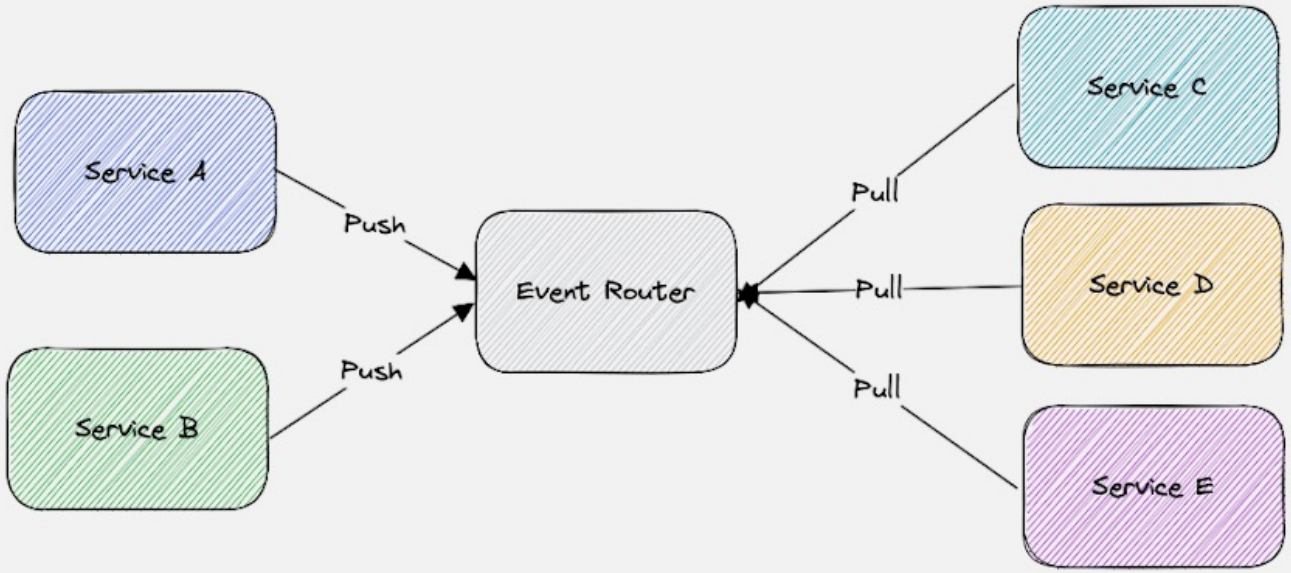
The three key components are:

Event Producers, Event Brokers, and Event Consumers

"Traditional" Microservices



Event-Driven Architecture



Some of the **tools** we can use:

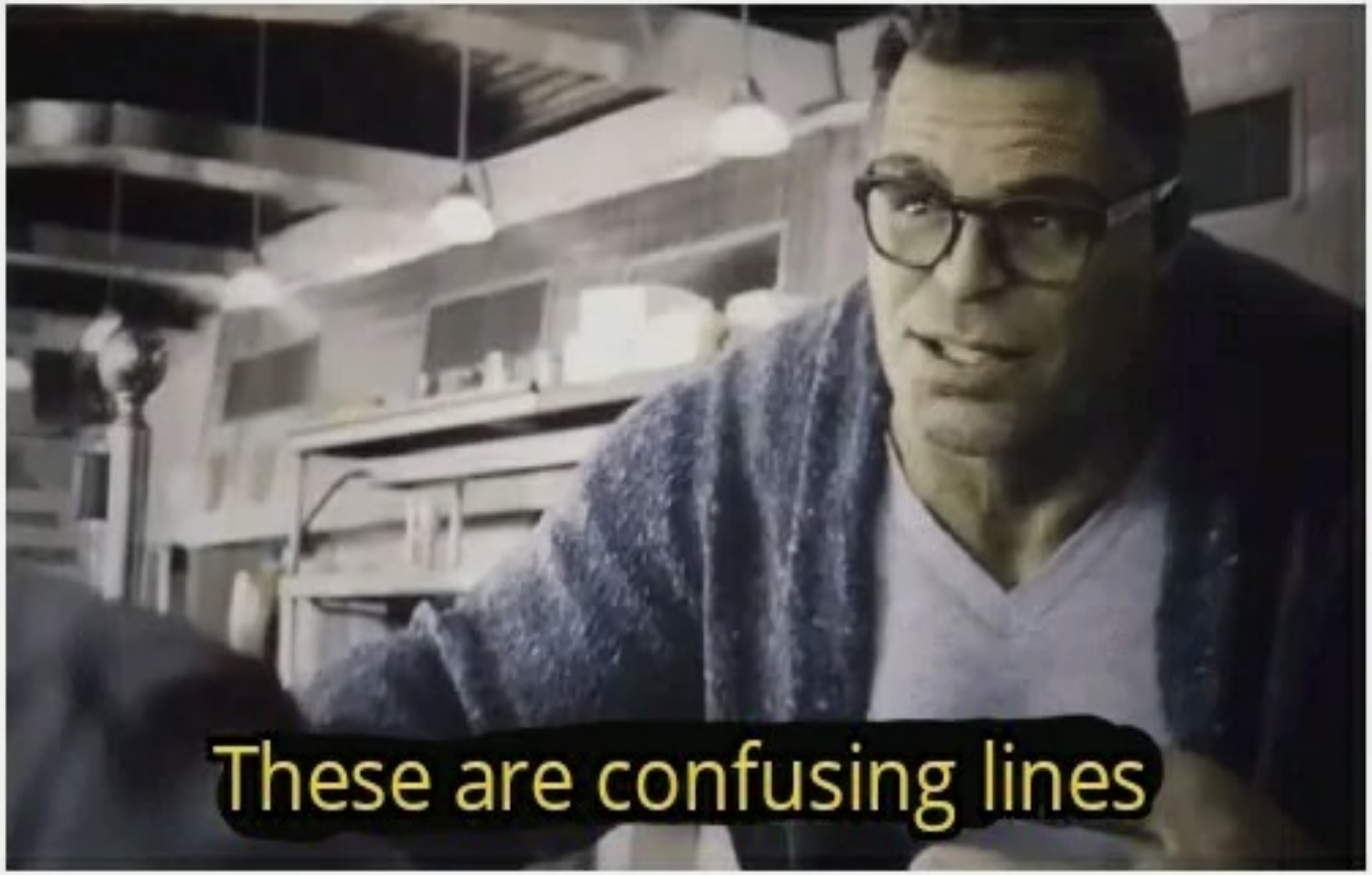
AWS EventBridge

Cloud Pub/Sub

The Challenge in large teams.

Me: writes code with no documentation

Me: *one month later*

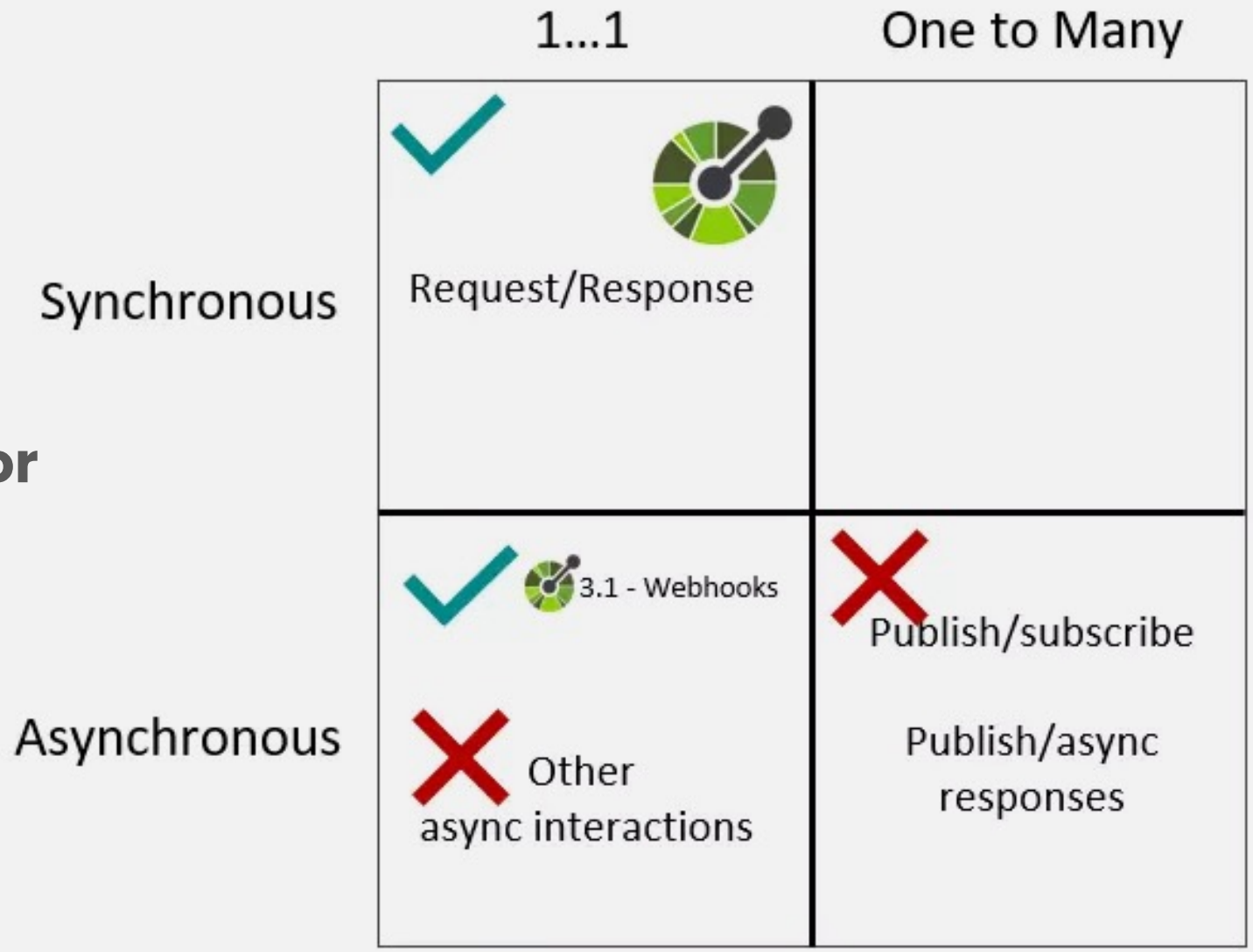


We usually use
OpenAPI for docs.

Open API
Specification

Swagger

But there's no support for Asynchronous APIs in OpenAPI



We create
OpenAPI standard for
Asynchronous APIs?


AsyncAPI

Building the future of Event-Driven Architectures (EDA)

Open-Source tools to easily build and maintain your event-driven architecture. All powered by the AsyncAPI specification, the **industry standard** for defining asynchronous APIs.

Read the docs ›

Quick search... ⌘ K

Proud to be part of the Linux Foundation

asyncapi.yaml

```yaml
asyncapi: 3.0.0
info:
  title: Account Service
  version: 1.0.0
  description: This service is in charge of processing user
signups :rocket:
channels:
  userSignedup:
    address:'user/signedup'
    messages:
      userSignedupMessage:
        $ref:'#/components/messages/UserSignedUp'
operations:
  processUserSignups:
    action:'receive'
    channel:
      $ref: '#/channels/userSignedup'
```

Account Service Documentation

Account Service 1.0.0

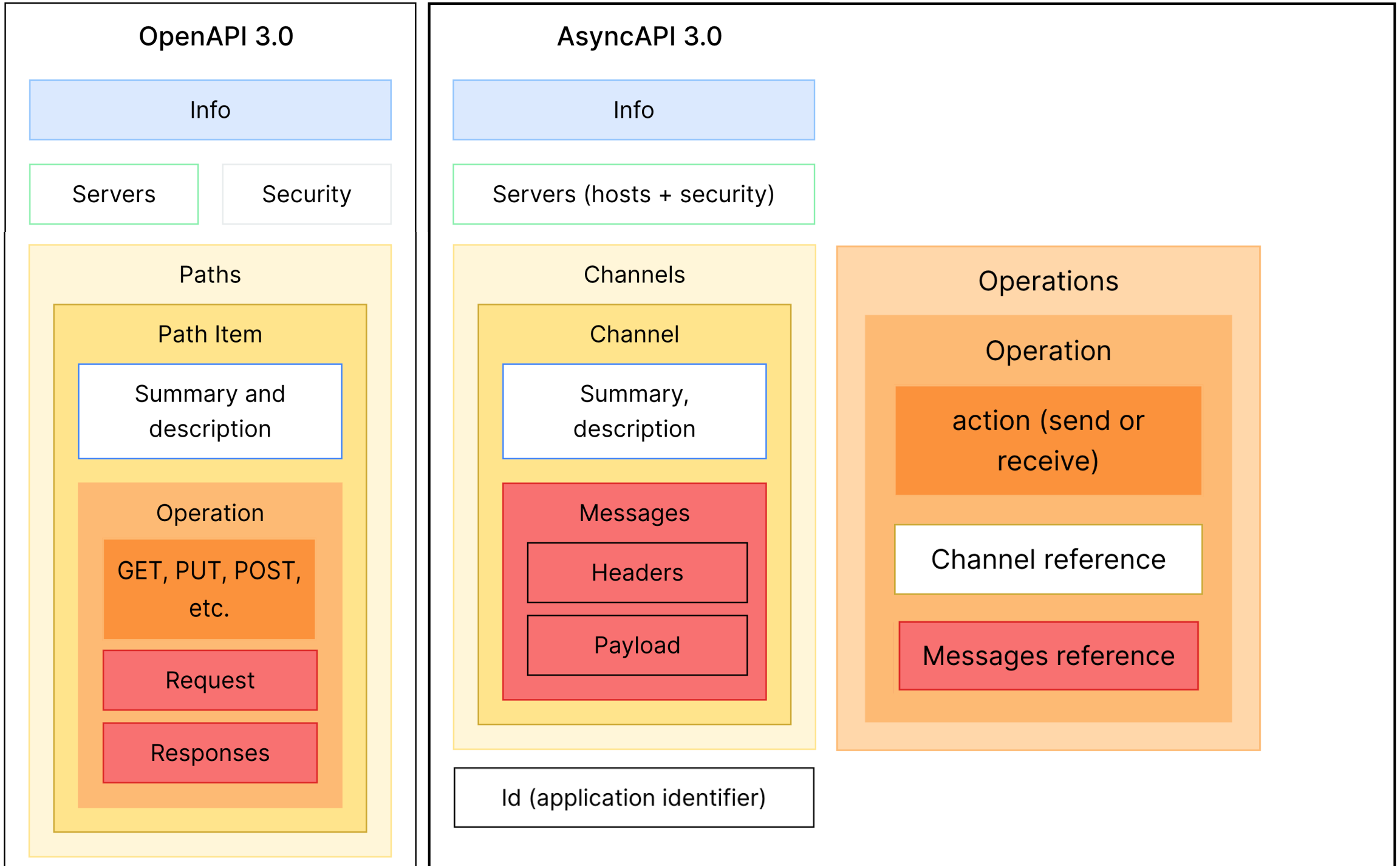
This service is in charge of processing user signups 🚀

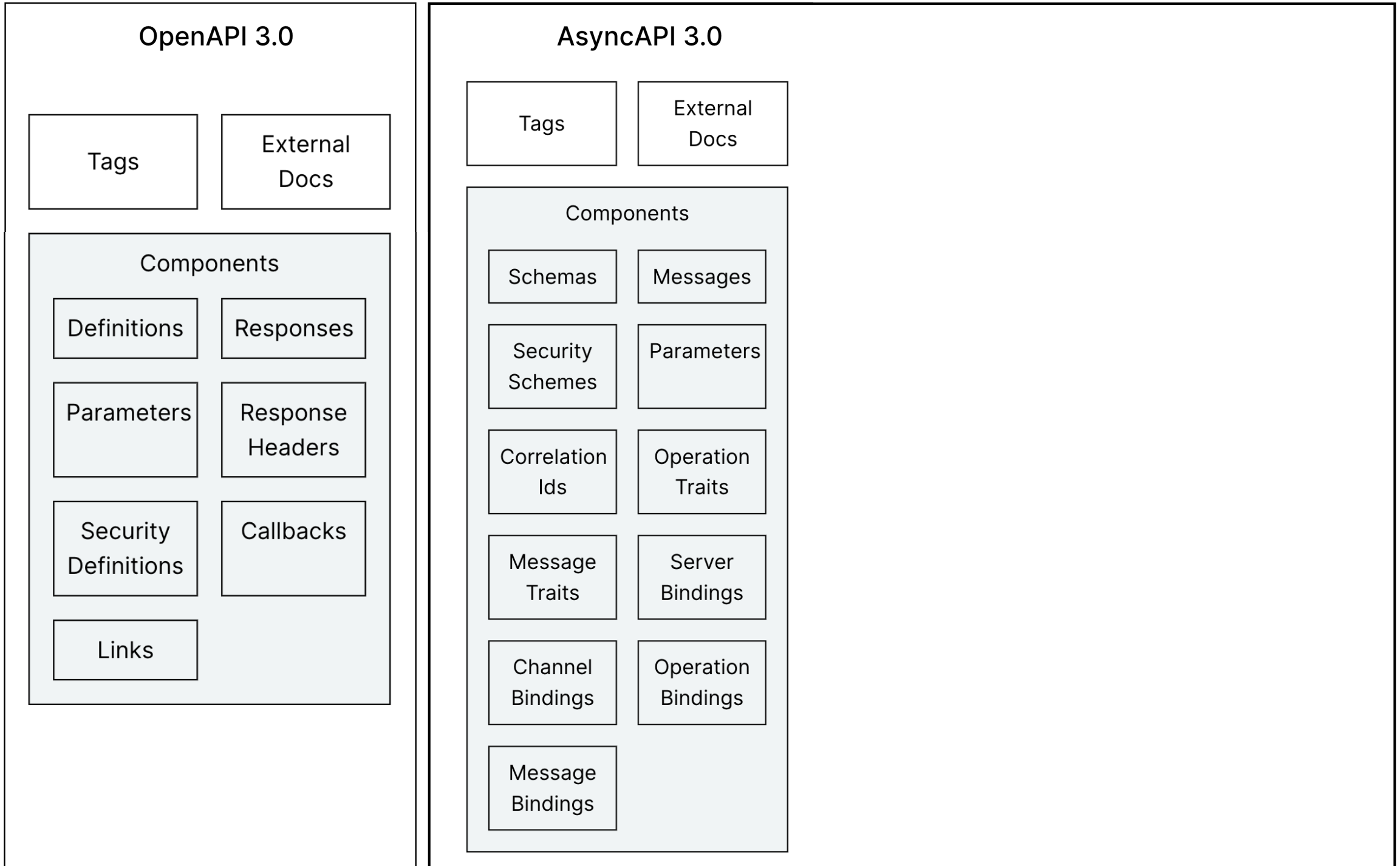
RECEIVES user/signedup

Accepts the following message:

Payload Object

displayName	String Name of the user
email	String email Email of the user


OpenAPI 3.0
Info
Servers
Security
Paths
Path Item
Summary and description
Operation
GET, PUT, POST, etc.
Request
Responses
AsyncAPI 3.0
Info
Servers (hosts + security)
Channels
Channel
Summary, description
Messages
Headers
Payload
Id (application identifier)
Operations
Operation
action (send or receive)
Channel reference
Messages reference


OpenAPI 3.0
Tags
External Docs
Components
Definitions
Responses
Parameters
Response Headers
Security Definitions
Callbacks
Links
AsyncAPI 3.0
Tags
External Docs
Components
Schemas
Messages
Security Schemes
Parameters
Correlation Ids
Operation Traits
Message Traits
Server Bindings
Channel Bindings
Operation Bindings
Message Bindings

```yaml
asyncapi: 3.0.0
info:
  title: Example application
  version: '1.1.0'
channels:
  userSignup:
    address: 'user/signedup'
    messages:
      userSignedupMessage:
        description: A message describing that a user just signed up.
        payload:
          type: object
          additionalProperties: false
          properties:
            fullName:
              type: string
            email:
              type: string
              format: email
            age:
              type: integer
              minimum: 18
operations:
  publishUserSignedup:
    action: 'send'
    channel:
      $ref: '#/channels/userSignup'
```

Example application 1.1.0

Operations

SEND `user/signedup`

Operation ID `publishUserSignedup`

Accepts the following message:

`userSignedupMessage`

Message ID `userSignedupMessage`

A message describing that a user just signed up.

Payload ^ Expand all **Object**

fullName	**String**	
email	**String**	**format: email**
age	**Integer**	**>= 18**

Additional properties are **NOT** allowed.

Examples

Payload >


UP TO DATE DOCUMENTATION
NOW THERE IS SOMETHING I HAVEN'T SEEN IN A LONG TIME.
imgflip.com


AsyncAPI
X
python™

FastStream

A Unified API catering to Multiple Brokers

Major **Features!**

- **Multiple Brokers**: FastStream provides a unified API to work across multiple message brokers (Kafka, RabbitMQ, NATS, Redis support)
- **Pydantic Validation**: Leverage Pydantic's validation capabilities to serialize and validates incoming messages
- **Automatic Docs**: Stay ahead with automatic AsyncAPI documentation
- **Powerful Dependency Injection System**: Manage your service dependencies efficiently with FastStream's built-in DI system
- **Testable:** Supports in-memory tests, making your CI/CD pipeline faster and reliable
- **Extensible:** Use extensions for lifespans, custom serialization and middleware
- **Integrations:** Fully compatible with any HTTP framework (Especially FastAPI)

Building our first **FastStream** App

app.py

```python
from faststream.nats import NatsBroker

broker = NatsBroker("nats://localhost:4222")
```

You could use
KafkaBroker or
RabbitBroker or
RedisBroker

app.py

```python
from faststream.nats import NatsBroker

broker = NatsBroker("nats://localhost:4222")



@broker.subscriber("test")
async def base_handler(body):
    print(f"Recieved {body}")
```

app.py

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker("nats://localhost:4222")

app = FastStream(broker)


@broker.subscriber("test")
async def base_handler(body):
    print(f"Recieved {body}")
```

Running the **EventBroker!** (NATS.IO)

docker-compose.yml

```yaml
version: '3.9'
services:
    nats:
        image: 'nats:latest'
        container_name: nats-server
        ports:
            - '6222:6222'
            - '8222:8222'
            - '4222:4222'
        command: '-js -m 8222'
```

Dashboard
FAQ
About

External Links
GitHub
NATS.io
NATS Docs

NCUC3SDRULV46KLJJQM3T4OKCQIK6KMKGLEX4HLU6BLSABT2J2IY4IJK

v2.10.17

Uptime 21m 7s
Server Time 12/07/2024, 10:25:13
Start Time 12/07/2024, 10:04:06
Config Load Time 12/07/2024, 10:04:06

CPU Load	Memory Usage	Connections	Total Connections	Subscriptions	Slow Consumers
0 %	12.83 MiB	0	0	59	0

Total Messages Received	Total Messages Sent	Total Data Received	Total Data Sent
0	0	0 B	0 B

Messages Received Rate	Messages Sent Rate	Data Received Rate	Data Sent Rate
0 /s	0 /s	0 B/s	0 B/s

Leaf Nodes	Routes	Remotes	Max Connections	Max Payload	Max Pending
0	0	0	65.54 K	1 MiB	64 MiB

Max Control Line	Ping Interval	Max Pings Out	Write Deadline	Auth. Timeout	TLS Timeout
4 KiB	2 m	2	10 s	2 s	2 s

Server

JetStream Stats

Slow Consumer Stats

Monitoring Server

app.py

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker("nats://localhost:4222")

app = FastStream(broker)


@broker.subscriber("test")
async def base_handler(body):
    print(f"Recieved {body}")
```

app.py

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker("nats://localhost:4222")

app = FastStream(broker)


@broker.subscriber("test")
async def base_handler(body):
    print(f"Recieved {body}")


@app.after_startup
async def test():
    await broker.publish("Hello EuroPython", subject="test")
```

```
$ faststream run app:app
2024-07-12 16:27:31,960 INFO     - FastStream app starting...
2024-07-12 16:27:31,974 INFO     - test |            - `BaseHandler` waiting for messages
2024-07-12 16:27:31,975 INFO     - FastStream app started successfully! To exit, press CTRL+C
2024-07-12 16:27:31,977 INFO     - test | 78f6122f-7 - Received
Recieved: Hello EuroPython
2024-07-12 16:27:31,977 INFO     - test | 78f6122f-7 - Processed
```

Let’s integrate with

FastStream

Pydantic Model

models.py

```python
from pydantic import BaseModel, Field


class UserModel(BaseModel):
    user_id: int = Field(
        ...,
        description="Unique Identifier",
        examples=[1],
    )
    name: str = Field(
        ..., description="User's Full Name",
        examples=["John Doe", "My Name"]
    )
```

Creating an **Event Producer**

app.py

```python
from faststream import FastStream
from faststream.nats import NatsBroker
from models import UserModel

broker = NatsBroker("nats://localhost:4222/")

app = FastStream(broker)


@broker.subscriber("user-register")
async def handle_msg(user: UserModel) -> None:  # Return type is also validated
    print(f"Welcome to EuroPython {user.name}, your ID is {user.user_id}")


@app.after_startup
async def test():
    user = UserModel(name="Abhinand", user_id=1)
    await broker.publish(user, subject="user-register")
```

Creating an **Event Producer**

producer.py

```python
import asyncio
from faststream.nats import NatsBroker

from models import UserModel

broker = NatsBroker("nats://localhost:4222/")

async def main():
    async with NatsBroker() as br:
        user = UserModel(name=f"Tony Stark", user_id=10)
        await br.publish(user, subject="user-register")

asyncio.run(main())
```

Using **FastAPI** Integration

app.py

```python
from faststream.nats.fastapi import NatsRouter
from fastapi import FastAPI
from models import UserModel

router = NatsRouter("nats://localhost:4222/")


@router.subscriber("user-register")
async def handle_msg(user: UserModel) -> None:  # Return type is also validated
    print(f"Welcome to EuroPython {user.name}, your ID is {user.user_id}")


app = FastAPI(lifespan=router.lifespan_context)
app.include_router(router)
```

FastAPI 0.1.0

APPLICATION/JSON

Servers

nats://localhost:4222/ NATS CUSTOM DEVELOPMENT

Operations

SUB user-register:HandleMsg

Available only on servers:

development

Channel specific information NATS › Expand all

Accepts the following message:

user-register:HandleMsg:Message user-register:HandleMsg:Message

Correlation ID $message.header#/correlation_id

Payload › Expand all Object uid: UserModel

Examples

Payload

```
{
  "user_id": 1,
  "name": "John Doe"
}
```

This example has been generated automatically.

Messages

#1 user-register:HandleMsg:Message `user-register:HandleMsg:Message`

Correlation ID `$message.header#/correlation_id`

Payload ^ Collapse all **Object** `uid: UserModel`

user_id
required

Integer
Unique Identifier
Examples values: `1`

name
required

String
User's Full Name
Examples values: `"John Doe"` `"My Name"`

Additional properties are allowed.

Schemas

UserModel ^ Collapse all **Object** `uid: UserModel`

user_id
required

Integer
Unique Identifier
Examples values: `1`

name
required

String
User's Full Name
Examples values: `"John Doe"` `"My Name"`

Additional properties are allowed.

Chaining Events

app.py

```python
from faststream.nats.fastapi import NatsRouter
from fastapi import FastAPI
from models import UserModel

router = NatsRouter("nats://localhost:4222/")

@router.subscriber("user-register")
@router.publisher("chain-event")
async def handle_msg(user: UserModel) -> UserModel:  # Return type is also validated
    print(f"Welcome to EuroPython {user.name}, your ID is {user.user_id}")
    return user


@router.subscriber("chain-event")
async def handle_msg(user: UserModel) -> None:
    print(f"Recieved: {user.user_id}:{user.name}")

app = FastAPI(lifespan=router.lifespan_context)
app.include_router(router)
```

EURO PYTHON PRAGUE &REMOTE 8-14 July 2024

@abhinand-c Strollby.com UST

Thank you

Abhinand C

Product Engineer
UST Strollby

Let's connect
@abhinand-c

Resources to take things forward!

1. EDA Visuals by Dave Boyney - eda-visuals.boyney.io
2. FastStream Docs - faststream.airt.ai/latest/getting-started
3. “The Many Meanings of Event-Driven Architecture” - Martin Fowler | GOTO 2017 - youtube.com/watch?v=STKCRSUsyP0
4. “What is Event Driven Architecture?” - Alex Hyett - www.alexhyett.com/event-driven-architecture
5. NATS.io Blogs - nats.io/blog
6. Rethink Connectivity Series | Synadia - youtube.com/watch?v=srARy0m9SdI&list=PLgqCaaYodvKY22TpvwlsalIArTmc56W9h&index=4
7. “Introducing FastStream: Code Generation Meets Stream Processing” - Davor Runje - youtube.com/watch?v=2AdJnCP970w
8. “Event-Driven Microservices, the Sense, the Non-sense and a Way Forward” - Allard Buijze" | GOTO 2019 - youtube.com/watch?v=jrbWIS7BH70