Diving into Event-Driven Architectures with Python

Solving complexity at scale

EuroPython 2023, Prague, Czech Republic – 21.07.2023

Marc-André Lemburg :: eGenix.com GmbH

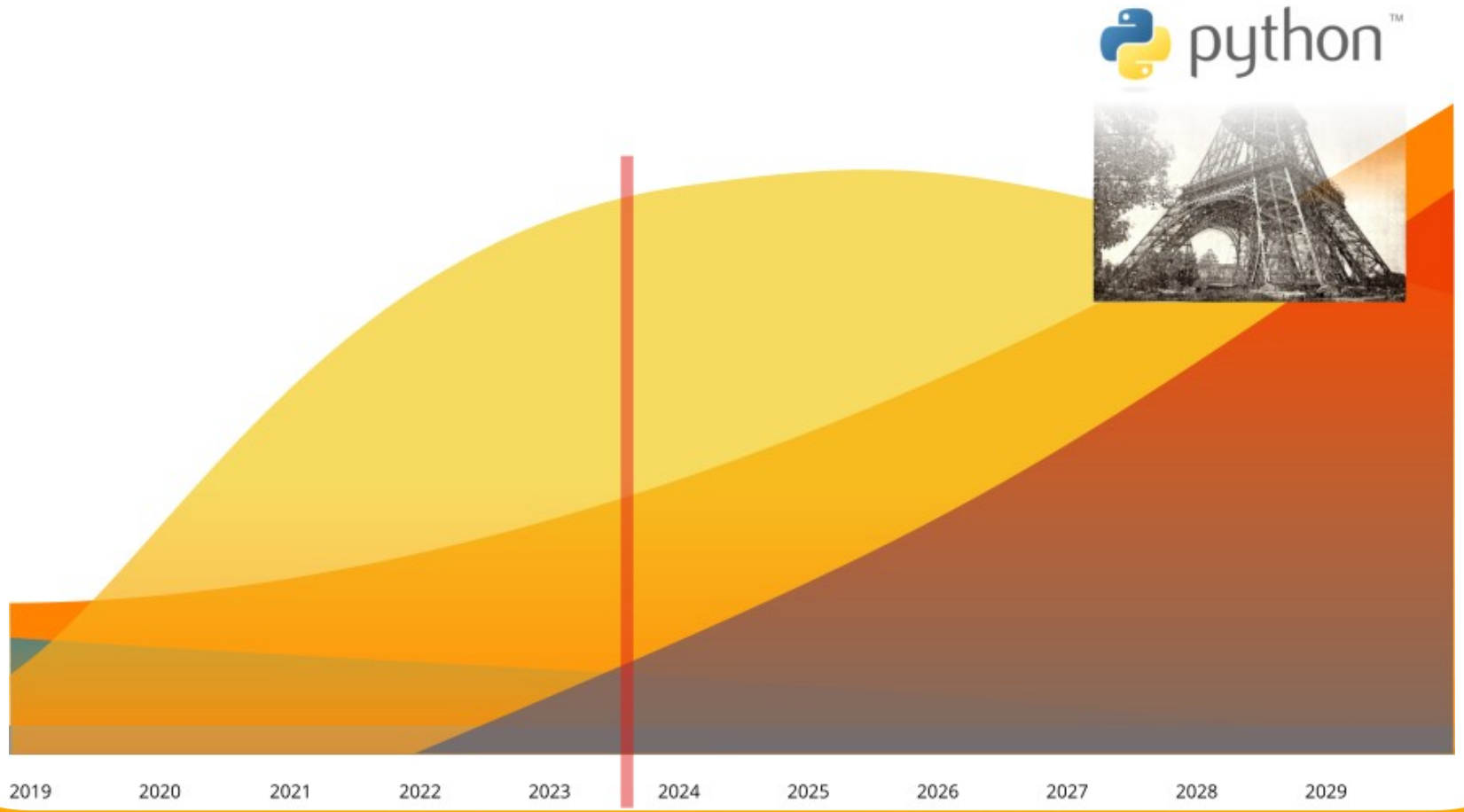
Speaker Introduction

- Marc-André Lemburg
  - Python since 1994
  - Studied Mathematics
  - CEO eGenix.com GmbH
  - Senior Solution Architect, Consulting CTO and Coach
  - Python Core Developer (PEP 100, DB-API)
  - EuroPython Society Fellow and former chair
  - Python Software Foundation Fellow and former director
  - Co-founder Python Meeting Düsseldorf
  - Based in Düsseldorf, Germany

  - More details: http://malemburg.com

Motivation: You're thinking of building the next Big Thing ...

Motivation: … but you have no idea where it will take you



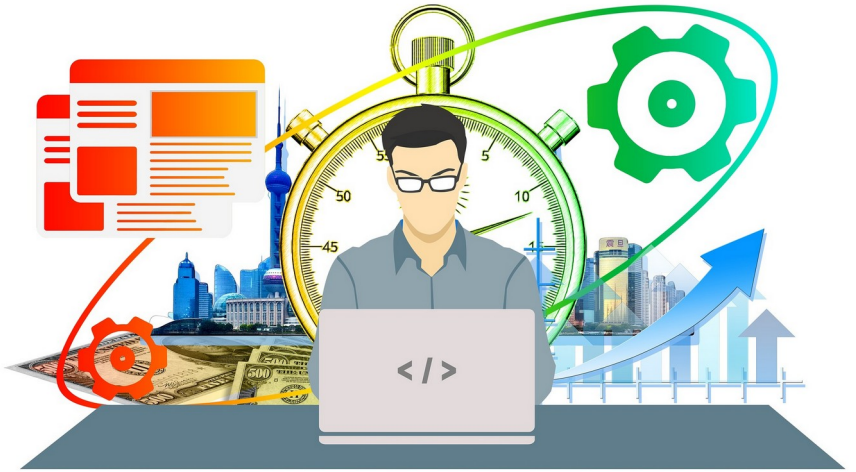
Motivation: Prepare for growth and flexibility

- Your architecture needs to do be:
  - scalable (horizontally and vertically)
  - easily adapt to new challenges
  - easy to maintain for devops
  - prepared for the enterprise
- It should also:
  - have good failure modes
  - integrate observability
  - automate governance

… just to name a few goals 😉

Well-known synchronous architectures

- REST
  - Good frontend support
  - Many available backend systems
  - Complex when it comes to adapting to new data
  - Difficult to scale



- GraphQL
  - Simplifies querying new data
  - New backend systems being developed, frontend support progressing
  - Difficult to scale

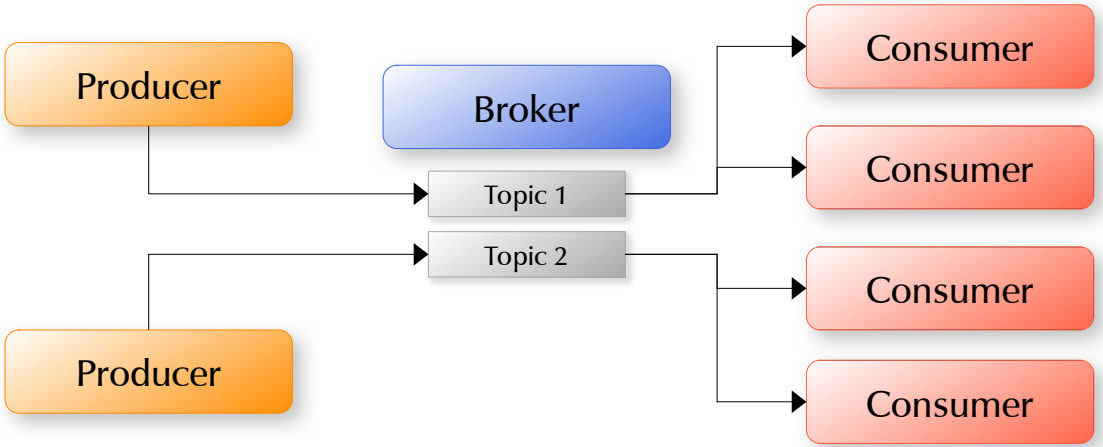
Event-Driven Architectures – *Rediscovered*

- EDA
  - Asynchronous
  - Can be combined with REST and GraphQL
  - Data agnostic
  - Scales well and promotes loose coupling
  - (Potentially) Solves many of the problems with integrating complex systems
  - Originally from the early 2000s

- *Let's have a closer look...*

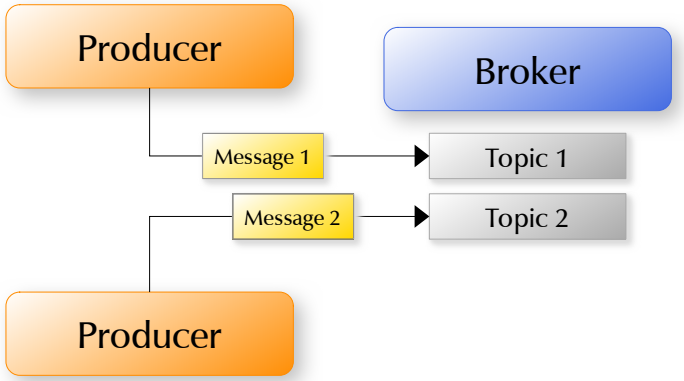
Main concept: Event-driven Communication

- Events are produced and consumed without direct connection
- Events are organized in topics
- A broker manages event distribution
- Fully asynchronous
- *Note: Consumers can be Producers as well*



Event Messages

- Events are captured using messages
  - “something happened”
  - “state changed”
  - “initiate command”
- Messages
  - Headers for meta data
  - Encoded payload
  - Best practice: 100-1000 bytes per message



- Bulk data is better stored in an object store or database

Event Message Payloads

- Common formats:
  - Apache Avro (default for Kafka)
  - Protobuf
  - Apache Thrift
  - MessagePack
  - JSON (typed using JSON Schema)

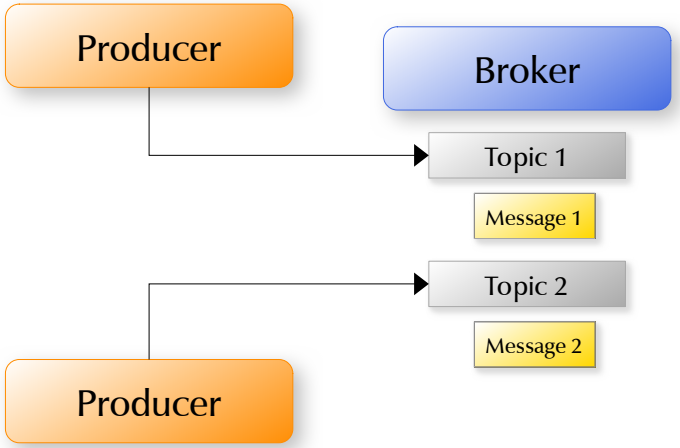
MessagePack

- Should be typed and signed for better security
- Compression helps keep traffic reasonable
  - zlib
  - snappy

Event Message Distribution (1/2)

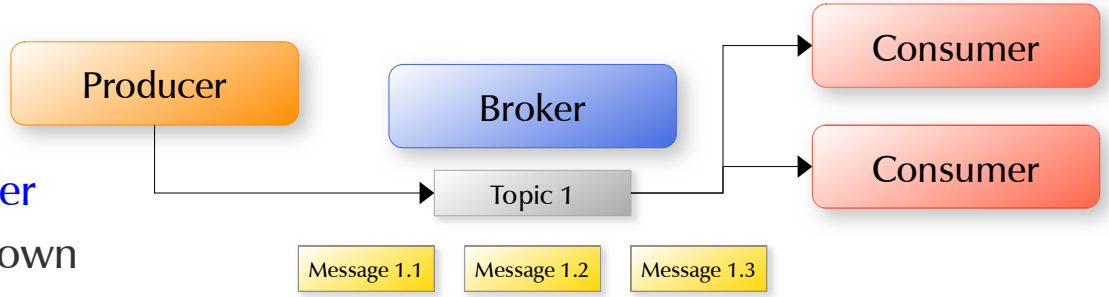
- Topics
  - Used to organize messages
  - Best practice: one message type per topic
  - Other terms: *channels, queues*

- Publish/Subscribe (PubSub)
  - Producers publish messages to a topic, consumers subscribe to messages on a topic
  - Brokers only queue messages until delivered
  - Messages cannot be resent to new subscribers



Event Message Distribution (2/2)

- Streaming
  - Broker stores messages in a stream buffer
  - Consumers manage their own reading from the stream
  - Enables replay and late joining

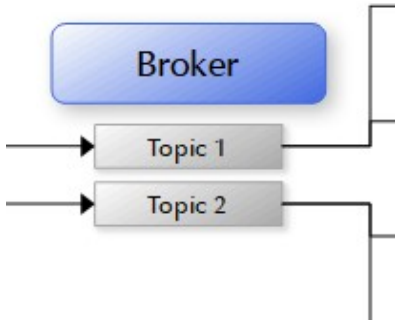


- Decoupled sending / receiving
  - Producer doesn't know who will be receiving the messages
  - Consumer doesn't need to know who produced the messages
  - Separation of concerns
  - Encapsulation

Message Brokers

- Custom protocols
  - Apache Kafka, Apache Pulsar, PostgresSQL, Redis
- Focus: AMQP
  - RabbitMQ
- Focus: MQTT
  - Apache ActiveMQ, HiveMQ, Mosquitto
- Cloud
  - AWS SNS, Google PubSub, Azure PubSub

- Older variants
  - IBM MQSeries

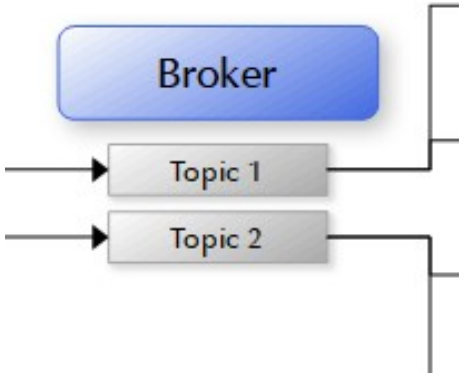


- Often come with connectors to simplify integrating sources (producers) and sinks (consumers)
  - Kafka Connect
  - Pulsar Connectors

kafka PULSAR RabbitMQ APACHE ACTIVEMQ

Broker Challenges

- Guaranteed delivery of messages
  - Even in the face of network issues, failures, etc.
- Processing messages exactly once
  - Make processing idempotent to soften this requirement
- Failure modes
  - Automatic retry/replay in case of failures
  - Backfilling data
- Trade-off message size vs. performance
  - Small message often result in additional source data queries
  - Larger messages are slower to handle and store



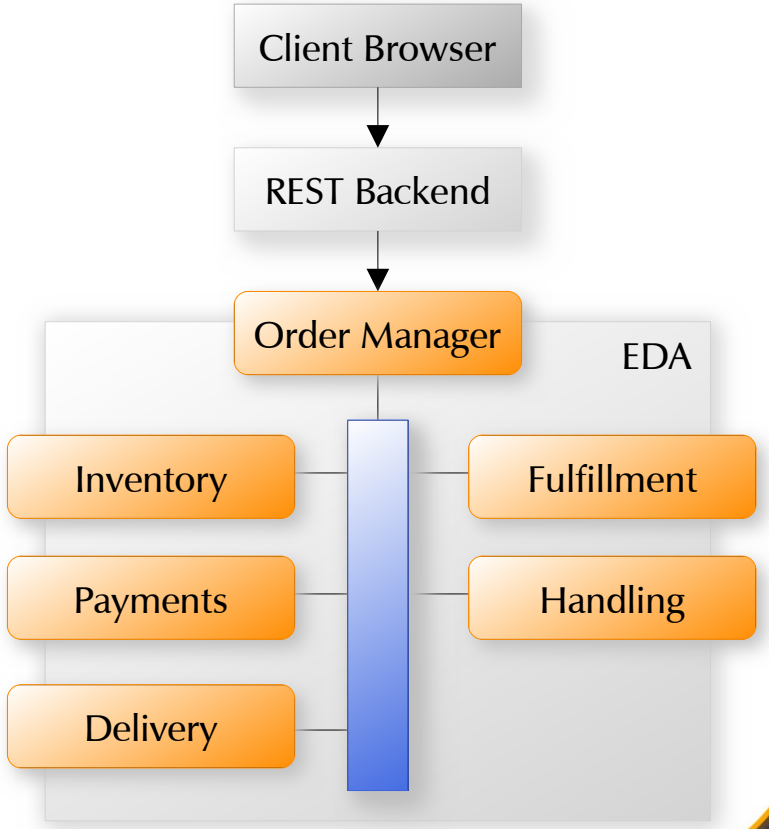
EDA API Specification and Documentation

- Requirements
  - Needed in machine readable form
  - Needed for discovery and ease of use
  - Needed for secure event API interaction (type checking)

- AsyncAPI – https://asyncapi.com/
  - Machine readable
  - Similar to OpenAPI/Swagger (REST)
  - Adapted for async processing, pub/sub
  - Relatively new (started in 2017)
  - Python is not really a key player (yet)
  - Node.js and Java dominate

Event-Driven Architectures (EDA) – *A Recap*

- Split applications into loosely coupled components
  - Have components communicate via events
  - Operate in asynchronous mode
- Use a broker to manage communication
- Scale up/down individual parts of the stack as needed
  - even on-demand...



Comparison: REST/GraphQL vs. EDA

- *Let's have a look at an example*

  Handle an order in web shop

Example: Handle order in web shop

- REST/GraphQL (Sync):
  - Interface to lots of subsystems to initiate order
    - Payment system
    - Fulfillment system
    - Inventory system
    - Queue order in handling and delivery system
  - Challenges
    - Shop backend is single point of failure (for the order)
    - Keeping track of dependencies between subsystems
    - Changes to a subsystems will often require changes to shop backend
    - Handling interfacing issues: network problems, temporary failures
    - Rollback order in case of problems

Example: Handle order in web shop

- EDA (Async):
  - Shop backend sends order event
  - Subsystems can react by processing their respective parts
  - An order management system manages the order
    - Assures successful completion of all subevents
    - Rolls back order in case of problems
  - System load can easily be distributed and scaled
    - Subsystem scaling can be applied independently
  - Code management is distributed as well
    - Fewer dependency issues
    - Easier to handle upgrades without changes to shop system

EDA sounds too good to be true: Challenges

- Debugging Challenges
  - Gathering logs from all nodes
  - Associating log entries with incoming requests
  - Introspection for processed requests
- Organizational Challenges
  - Common understanding of the architecture
  - Communicate the design to all participants
  - Document events, event types
    - AsyncAPI
  - Document expectations
    - When to send events
    - What to expect as a result

EDA and Python

- Python async support is great for pub/sub style APIs
  - Scales well
  - Easy to use
  - Can be combined with multi-threading access to interface libraries
    - External C libraries don't need the Python GIL

- *Let's have a closer look:*
  - *AsyncAPI support*
  - *Roll-your-own*
  - *Conclusion*

Python AsyncAPI Support

- Not much available for Python :-(
  - Would be great to get some more attention and support from the Python community
- AsyncAPI community
  - Much focus on Node.js and Java
- AsyncAPI Studio
  - Online tool for working with AsyncAPI specs
  - Can generate Paho (MQTT) code from spec
  - Just basic support for Python
  - Code generation not optimal

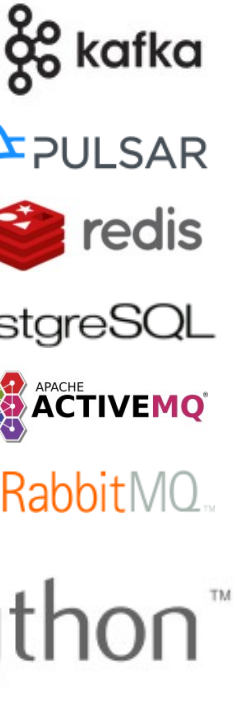
Python AsyncAPI Support: PyPI packages

- asyncapi package
  - Dynamically reads AsyncAPI spec and provides pub/sub APIs
  - Uses broadcaster package for the lower level pub/sub interfaces
  - Supports Redis, Kafka, PostgreSQL, Google pub/sub
  - Development stalled (last active in 2020)

- fastapi-asyncapi package
  - Almost no documentation
  - Helps expose AsyncAPI specs on a web service
  - Development stalled (last active in 2021)

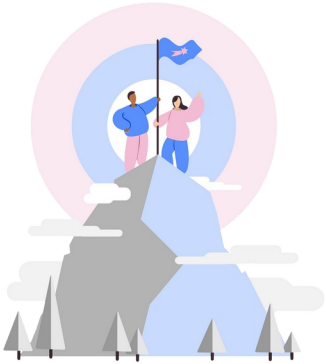
Python Roll-your-own EDA: Low level PyPI packages

- Use available package to interface to brokers directly
  - Kafka: aiokafka
  - Pulsar: pulsar
  - Redis: asyncio-redis
  - Postgres: asyncpg
  - MQTT:
    - paho-mqtt (sync)
    - asyncio-paho (async)
  - RabbitMQ:
    - amqp or rabbitmq-client (sync)
    - aioamqp (async)
  - *(many more, e.g. for cloud services, etc.)*



EDA and Python: Conclusion

- Roll-your-own is definitely possible now
  - And used a lot in major companies
- We need better support for EDA in Python
  - More abstractions
  - Better integration into existing web backends
- More integration of AsyncAPI standards would also help
  - Dynamic spec generation
  - Dynamic API generation
  - Code generation tools



Main takeaway: Never stop to learn and try out new things...

Thank you for your attention !

Time for discussion

Contact

eGenix.com Software, Skills and Services GmbH
Marc-André Lemburg
Pastor-Löh-Str. 48
D-40764 Langenfeld
Germany

Email: mal@egenix.com
Phone: +49 211 9304112
Fax: +49 211 3005250
Web: https://www.egenix.com/

LinkedIn:

References

- Several photos taken from Pixabay and Unsplash
- Some screenshots taken from the mentioned websites
- All other graphics and photos are (c) eGenix.com or used with permission
- Details are available on request
- Logos are trademarks of their respective trademark holders