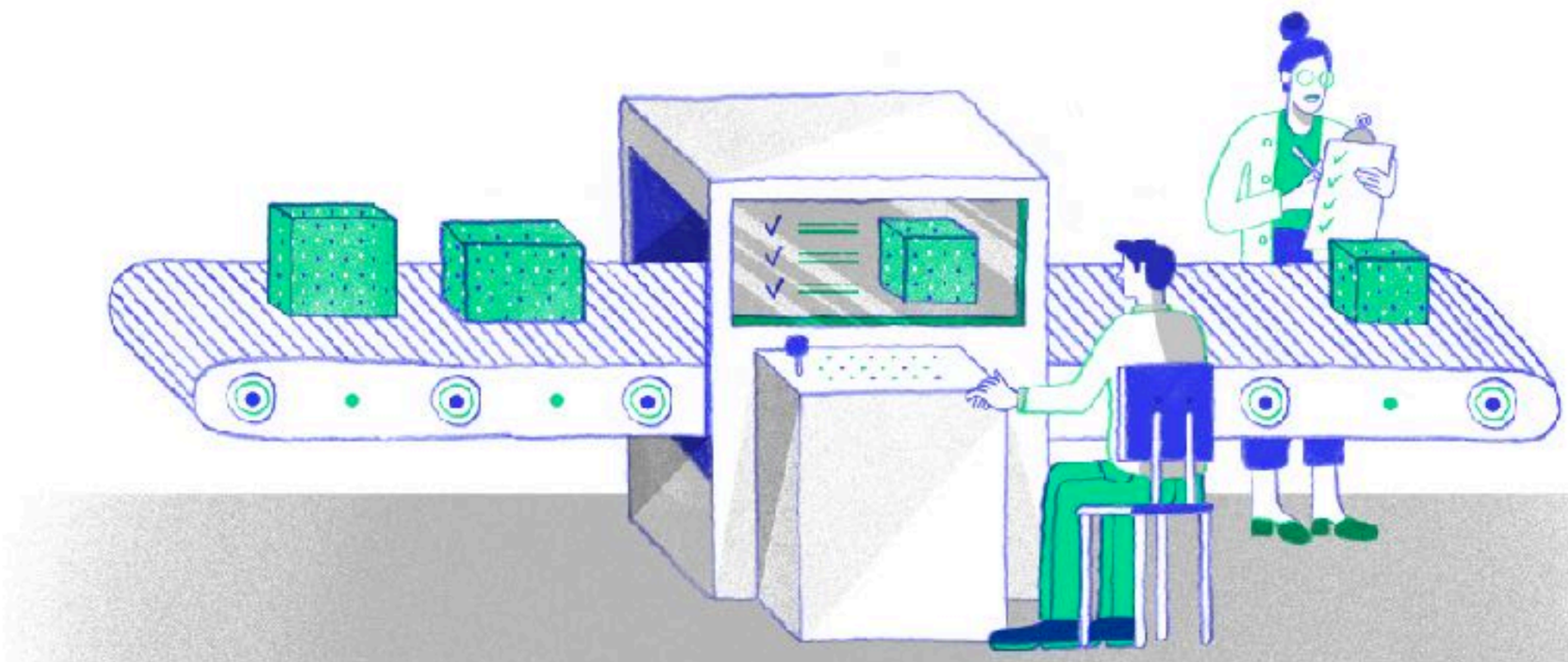
Olist Architecture

From monotith to microservices



Osvaldo

- Programador amador desde 1986, profissional desde 1989 e pythonista desde 2000
- Criador e ex-moderador da da python-brasil@yahoo (atual python-brasil@googlegroups)
- Criador da primeira versão do www.python.org.br
- Sócio fundador e ex-presidente da Associação Python Brasil
- Autor do (antigo) livro Python e Django
- Atualmente na Olist
- osvaldo@olist.com e @osantana (ou @osantanabr)

olist

O que nós fazemos?

Merchants

Online | Offline

olist

Submarino

AMERICANAS.COM

mercado livre

extra

CASAS BAHIA

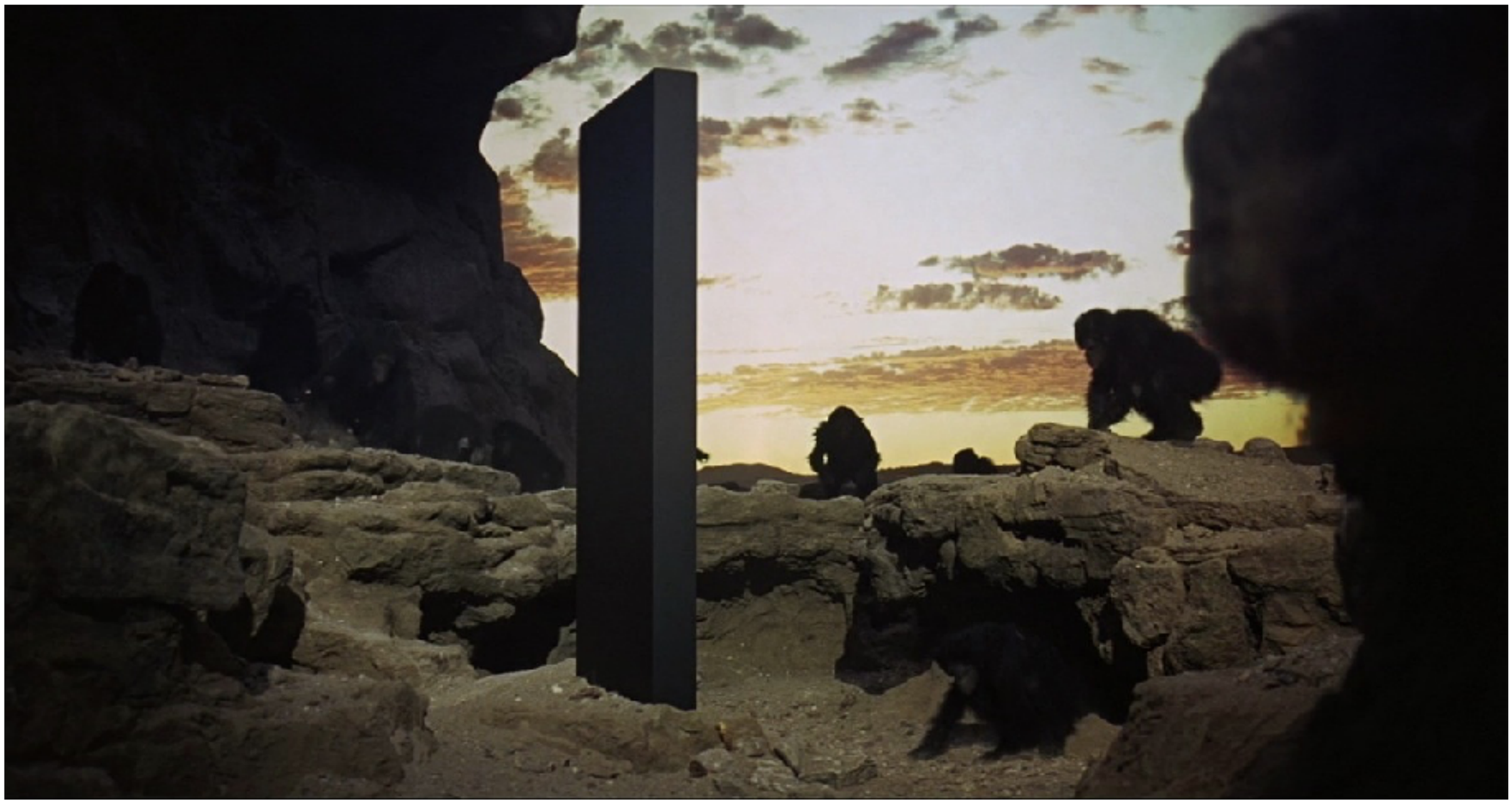
Walmart

olist

Previous System

Monolithic

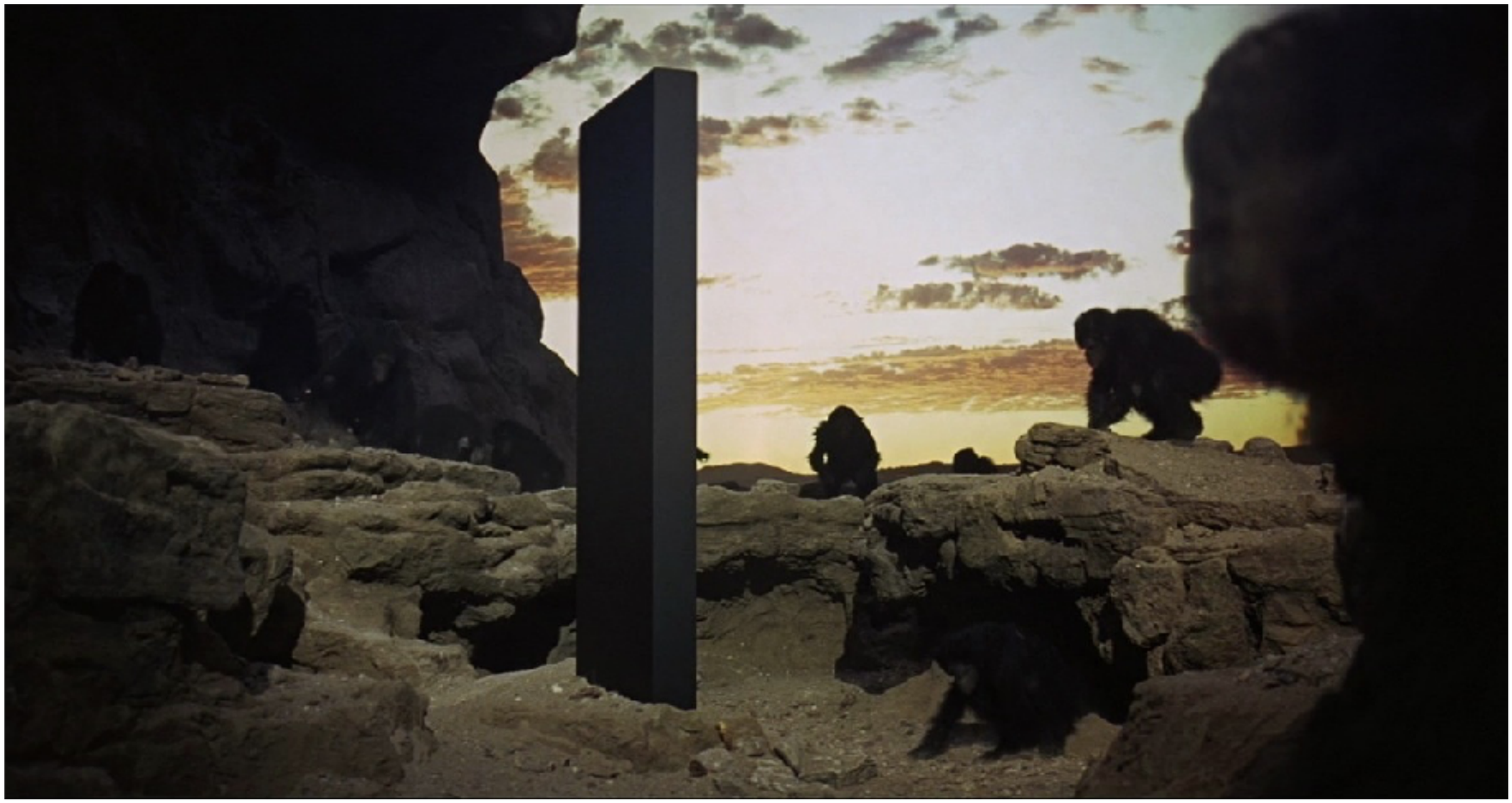
olist



Monolithic

No Scalability

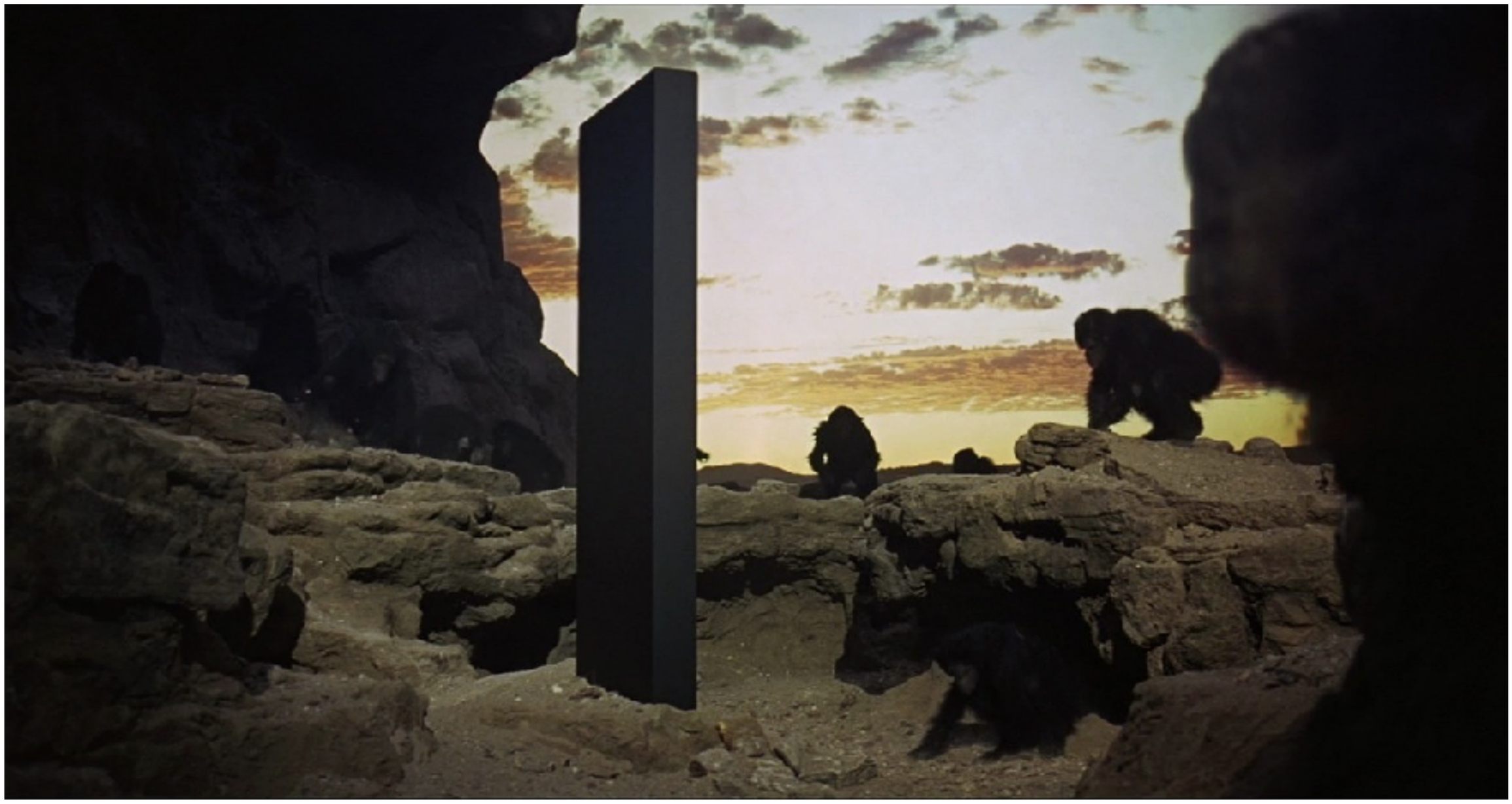
olist



Monolithic

No Reliability

olist



Monolithic

No Safety

olist

Monolithic

Complex

olist

New Platform

Let's write a new version...

olist

Requirements

olist

Requirements

Simplicity

olist

Requirements

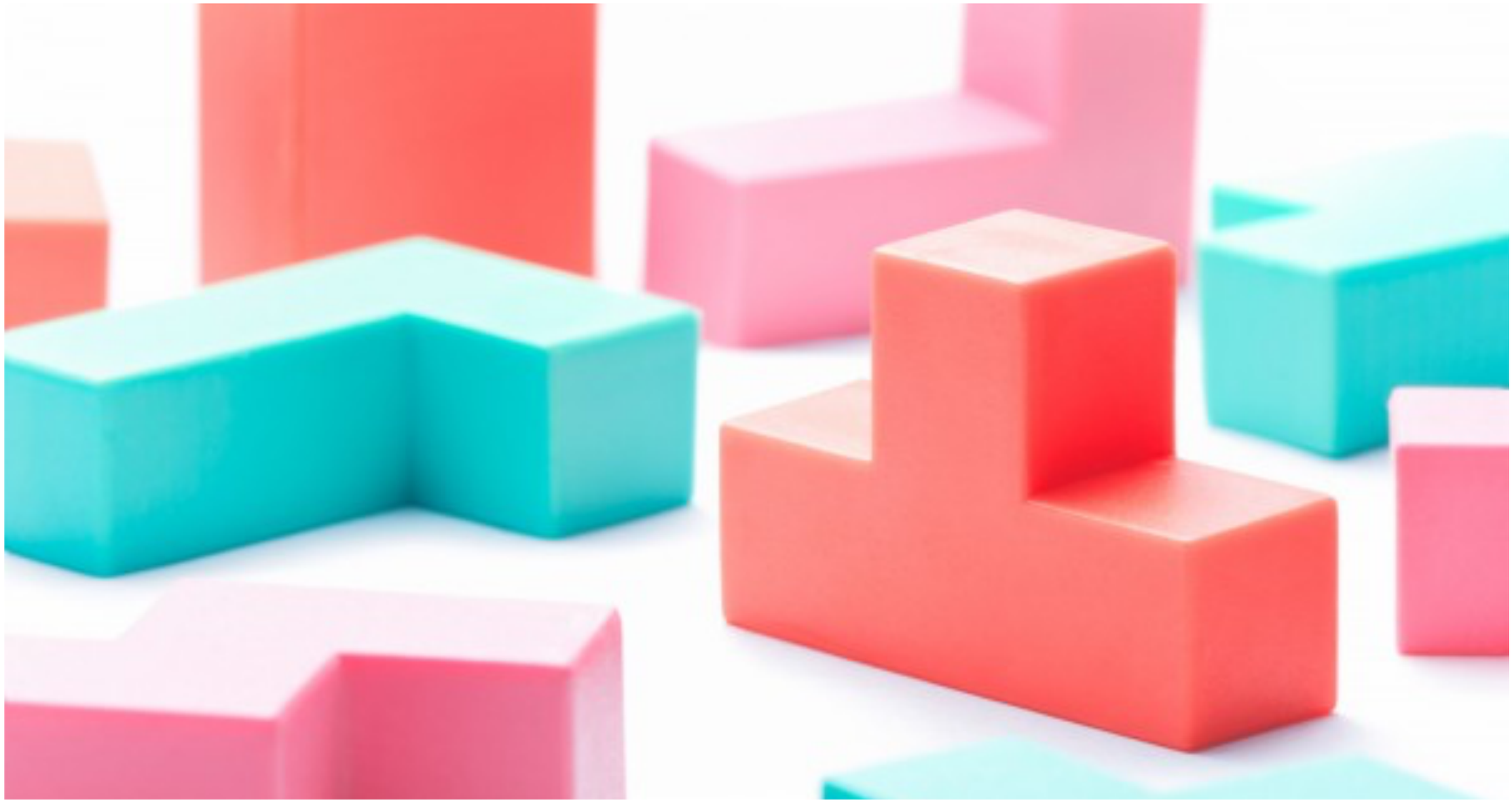
Scalability

olist

Requirements

Resilience

olist



Requirements

Modularity

olist

Requirements

Safety

olist

Premisse

- No matter if a **system** is internal or external, it eventually...
- ... goes **offline**...
- ... **crashes**...
- ... or **change their behaviour without notice**.

olist

New Architecture

olist

Microsservices

... or SOA

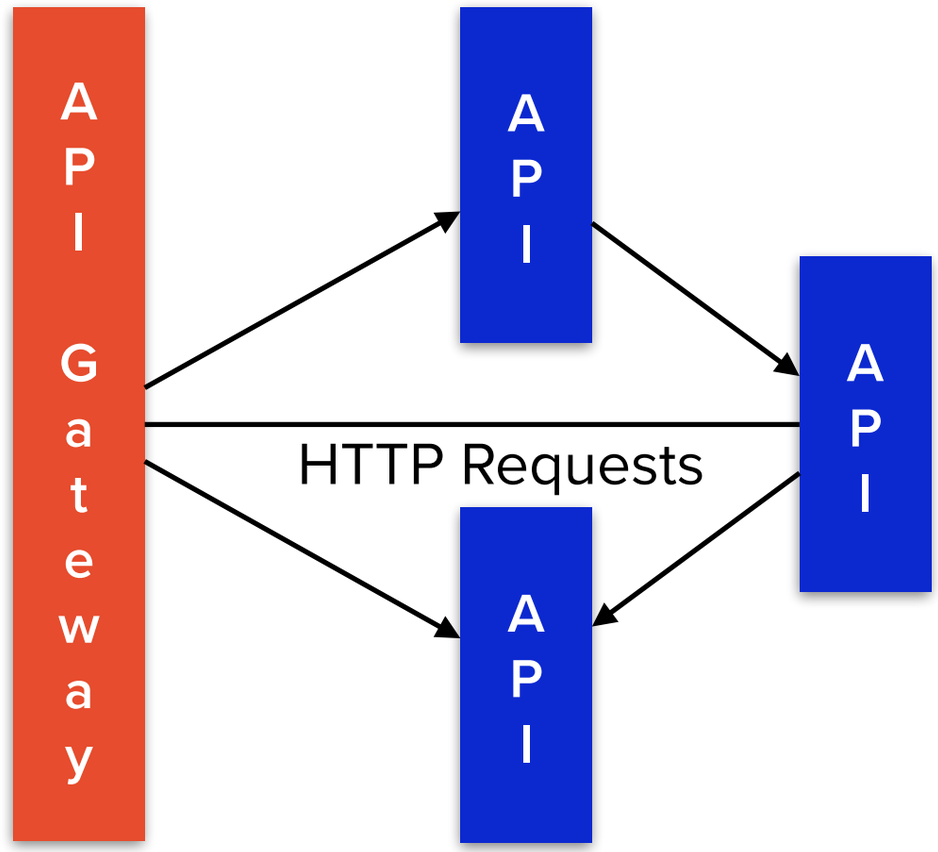
olist

Microservices

Implementation Models

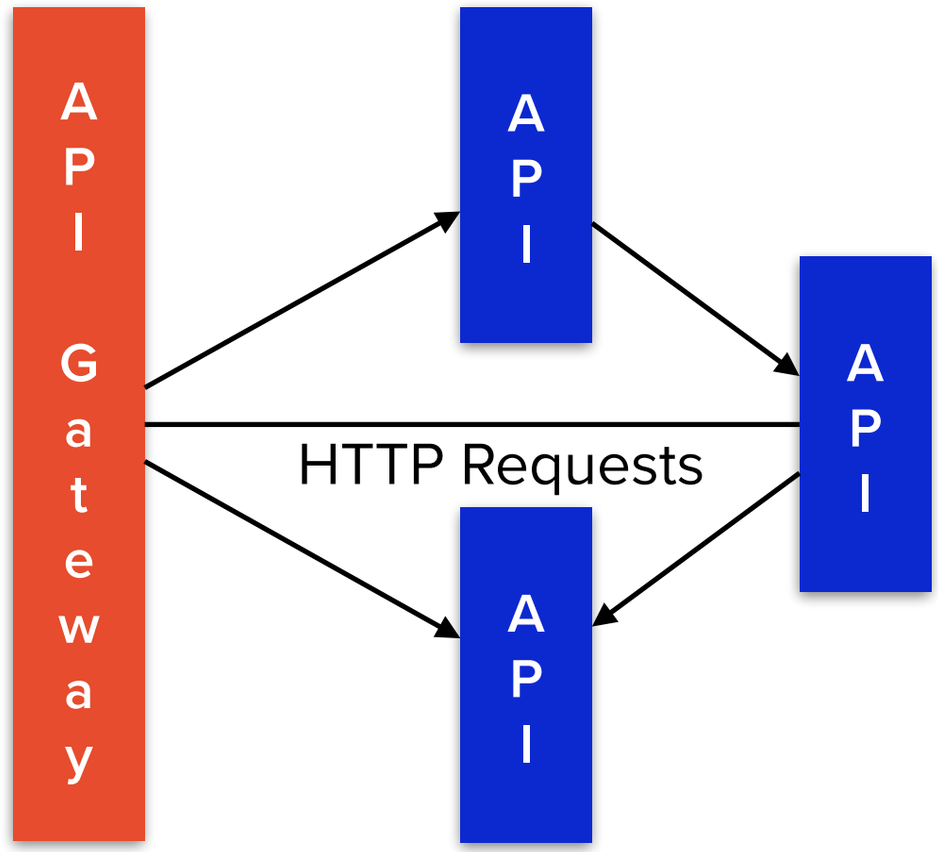
olist

Communication via API



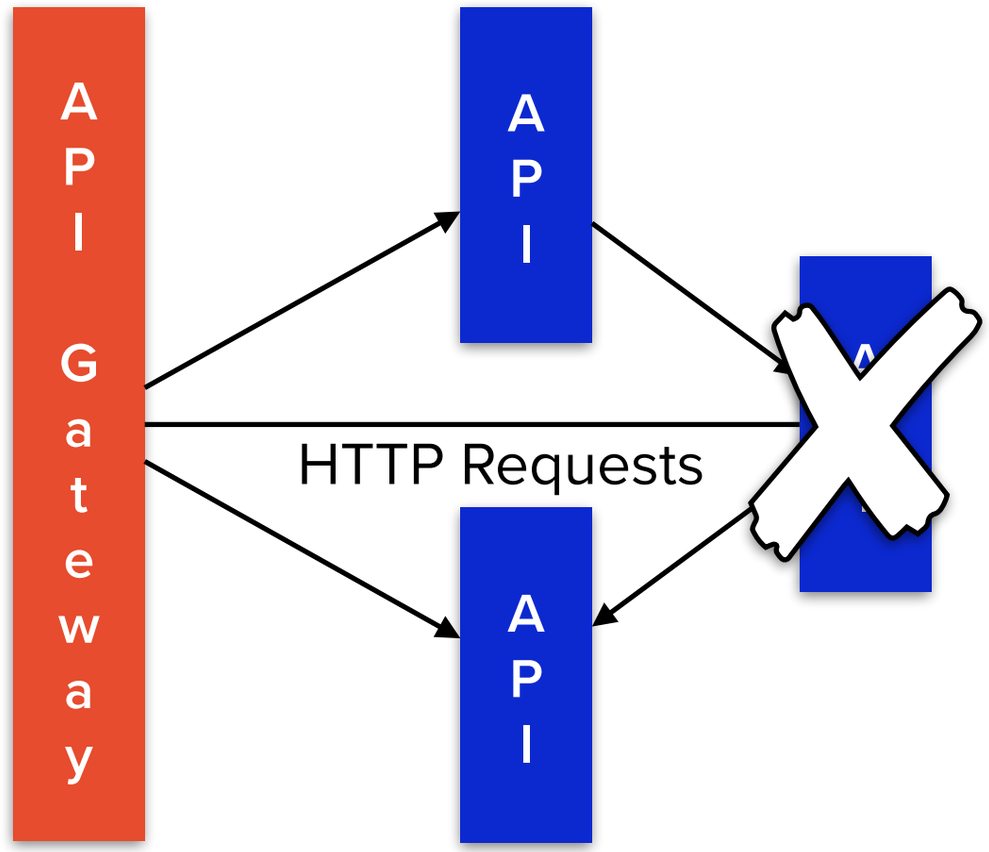
Communication via API

- RESTful communication between services
- Synchronous Service Calls
- Strict dependency between services



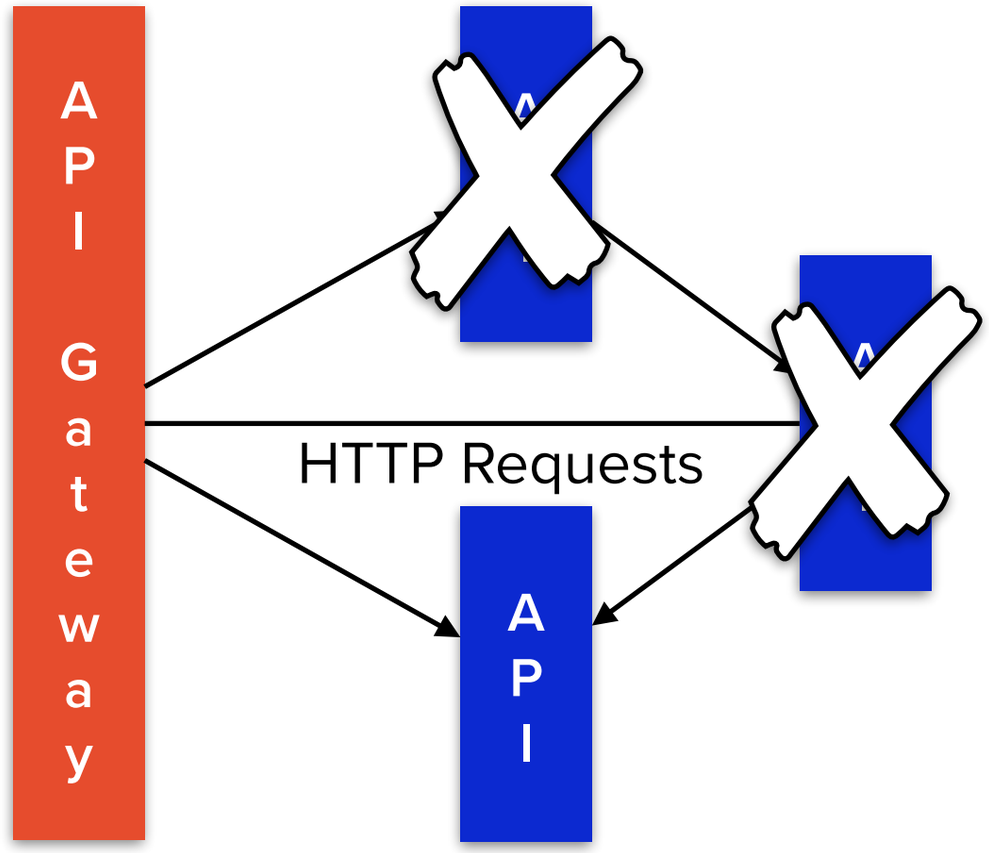
Communication via API

- RESTful communication between services
- Synchronous Service Calls
- Strict dependency between services



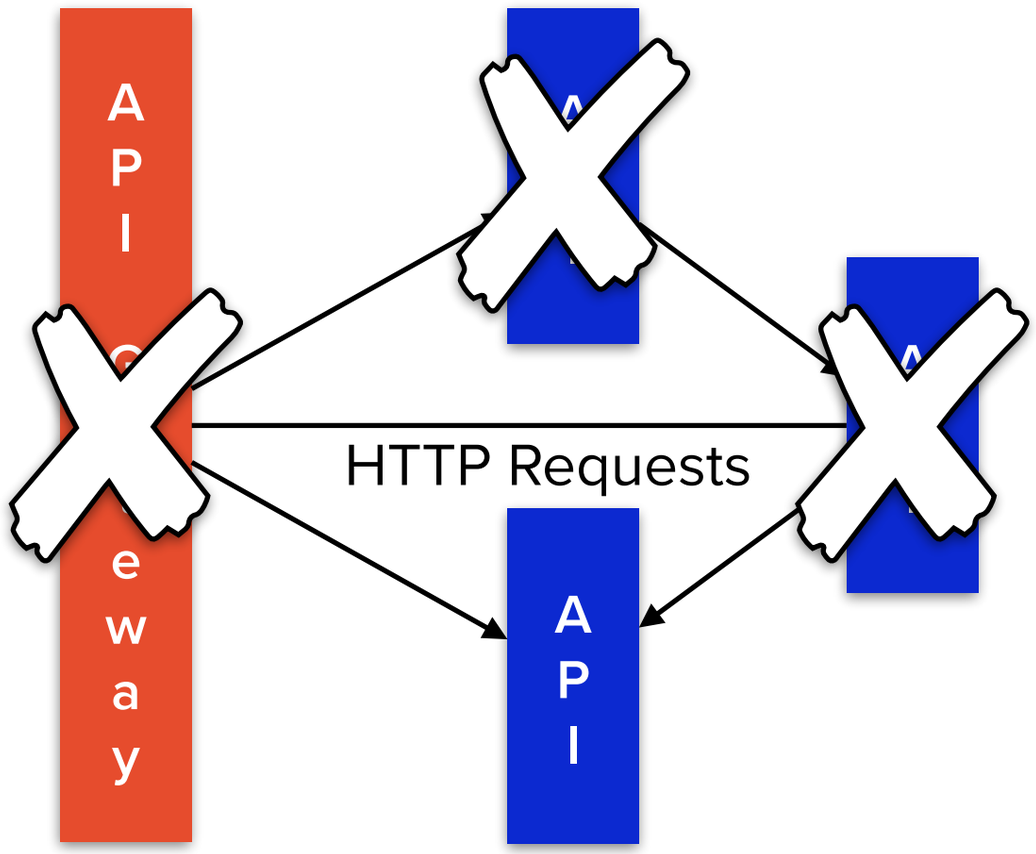
Communication via API

- RESTful communication between services
- Synchronous Service Calls
- Strict dependency between services



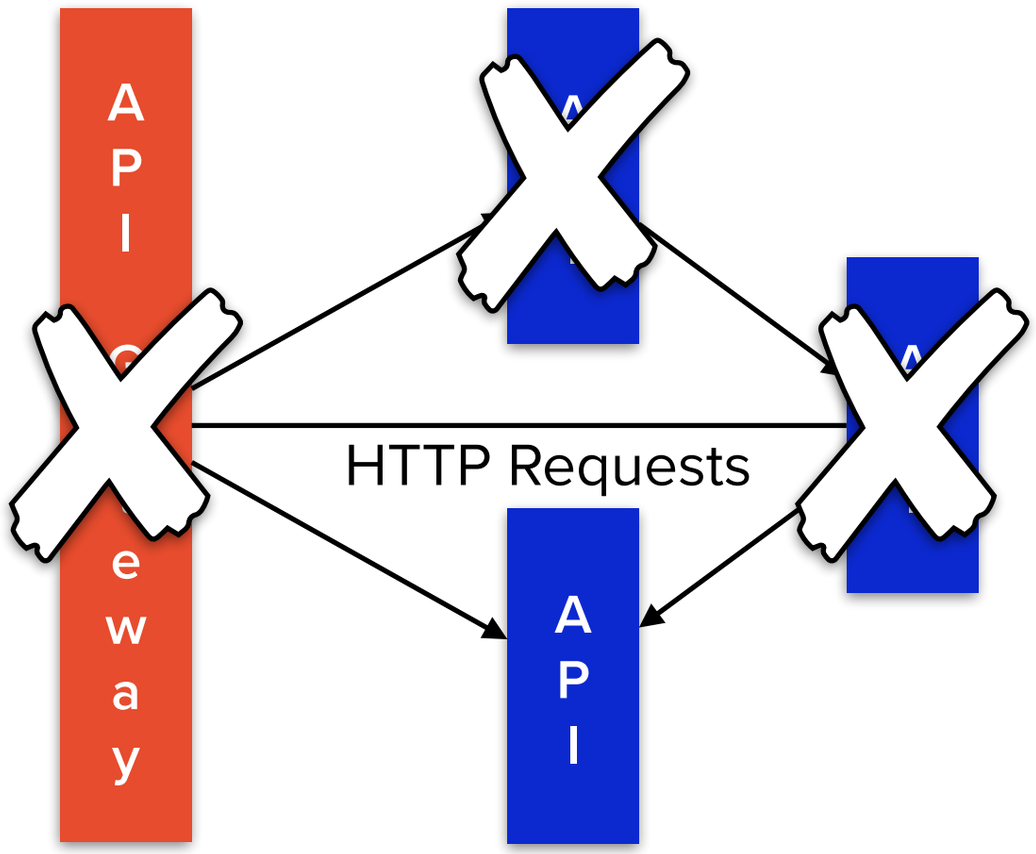
Communication via API

- RESTful communication between services
- Synchronous Service Calls
- Strict dependency between services

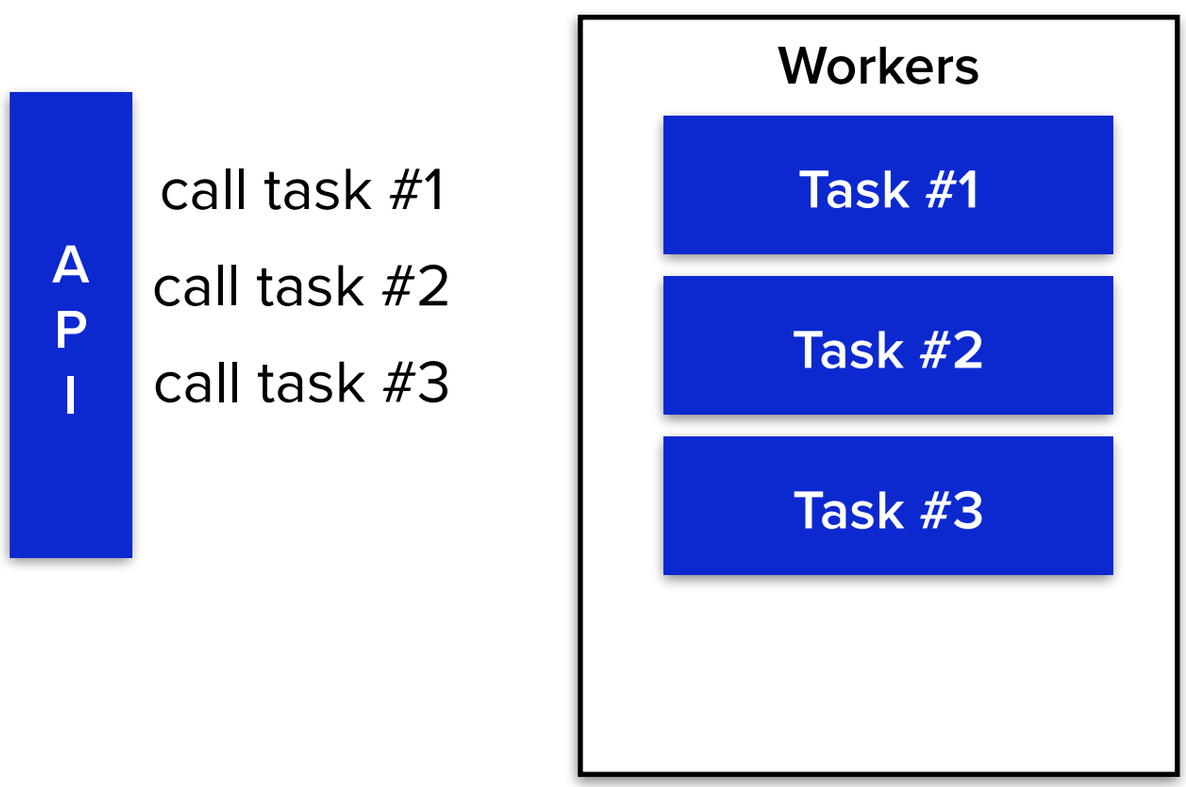


Communication via API

- RESTful communication between services
- Synchronous Service Calls
- Strict dependency between services
- No resilience
- No safety* (eg. data loss on request failures)

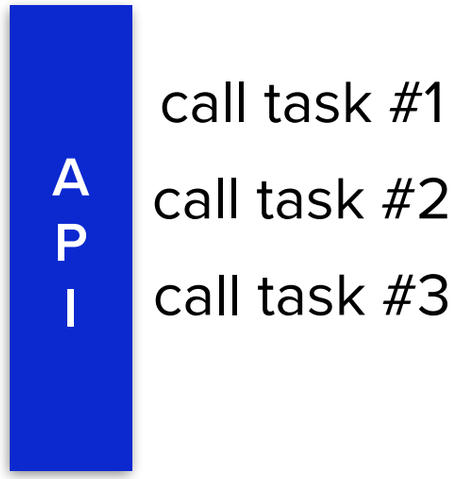


Async Task Execution



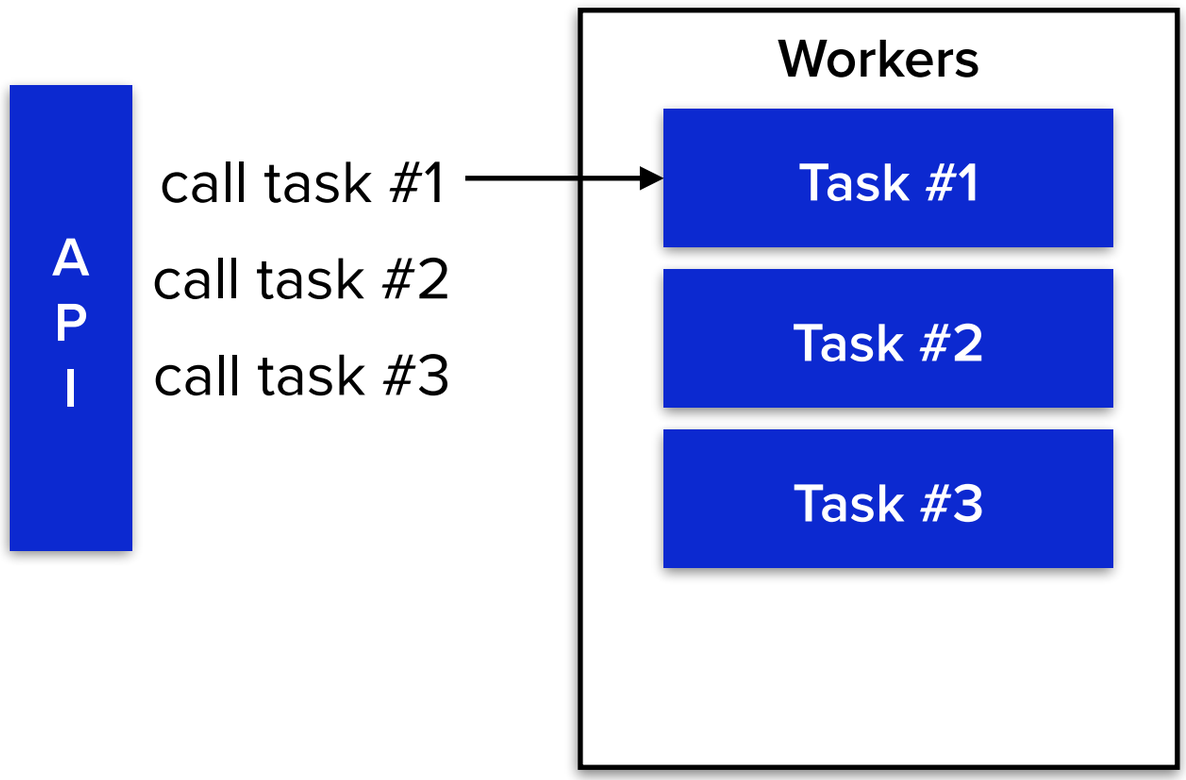
Async Task Execution

- Asynchronous Procedure Calls
- Queue based task execution
- Client must knows about their dependents (I've to change API every time I need to add a new task)



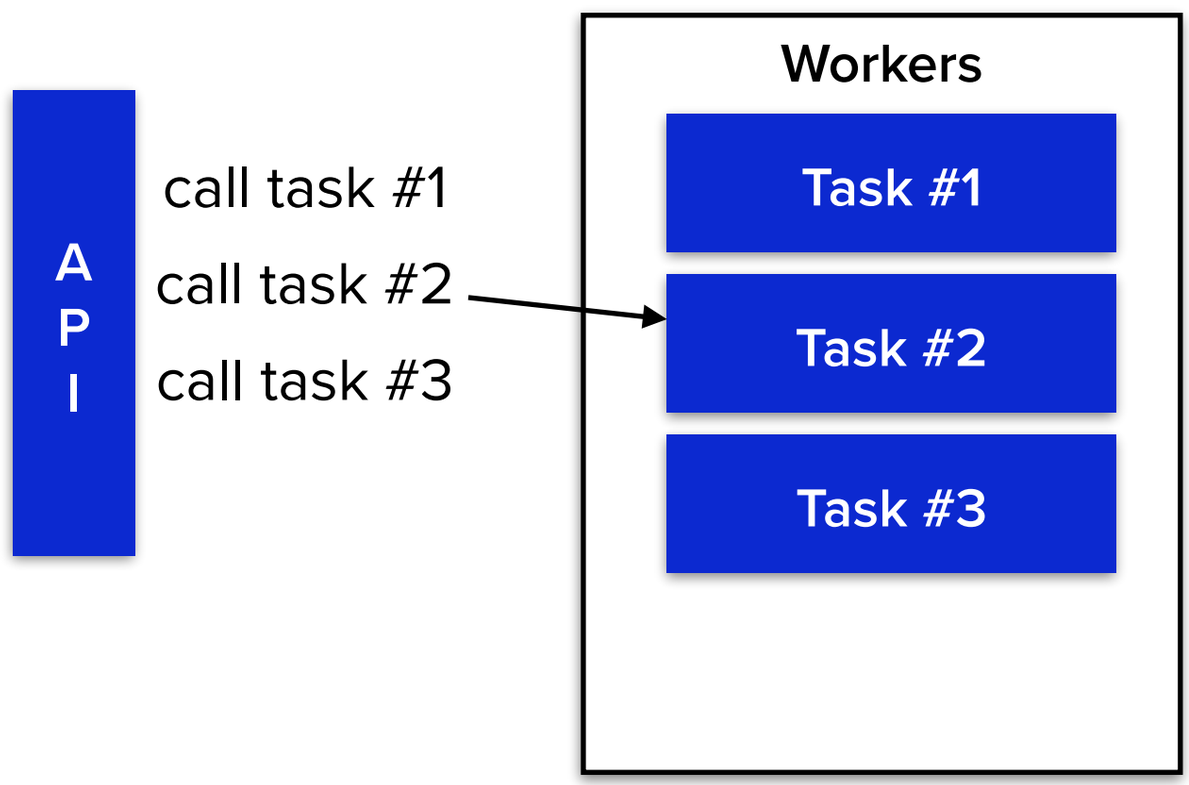
Async Task Execution

- Asynchronous Procedure Calls
- Queue based task execution
- Client must knows about their dependents (I've to change API every time I need to add a new task)



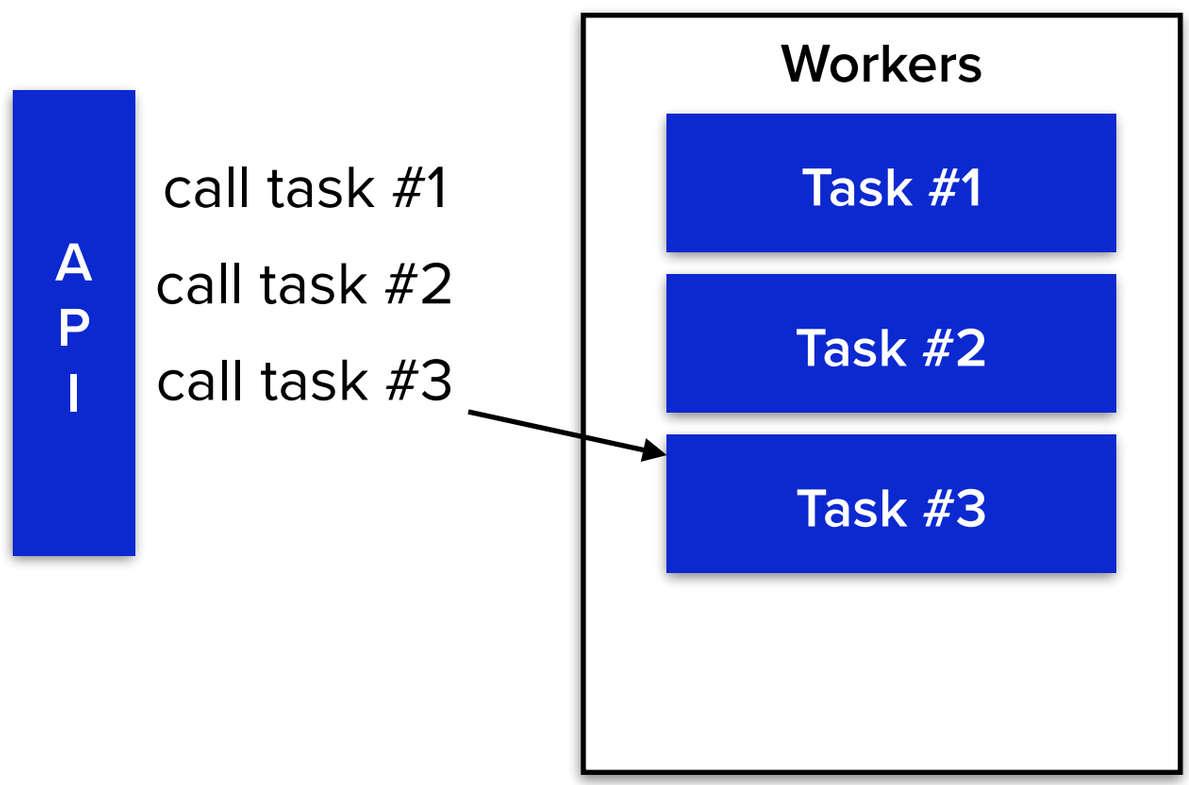
Async Task Execution

- Asynchronous Procedure Calls
- Queue based task execution
- Client must knows about their dependents (I've to change API every time I need to add a new task)



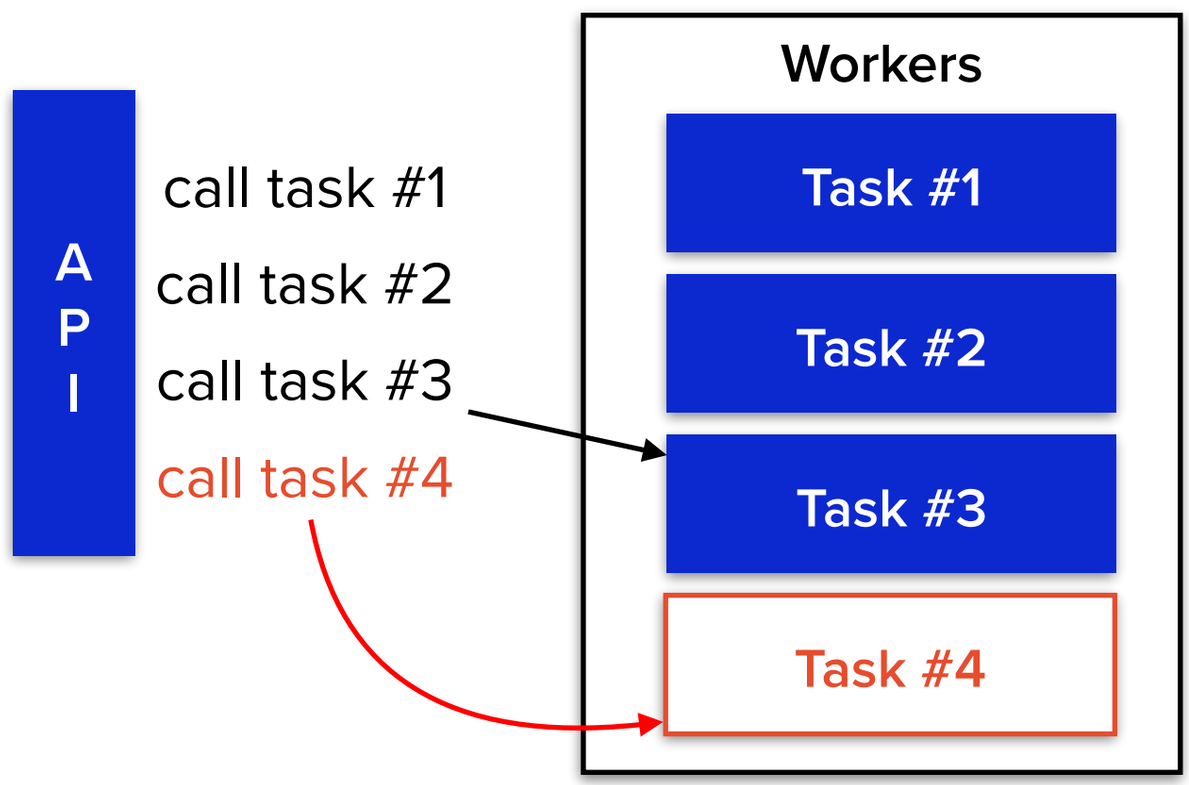
Async Task Execution

- Asynchronous Procedure Calls
- Queue based task execution
- Client must knows about their dependents (I've to change API every time I need to add a new task)



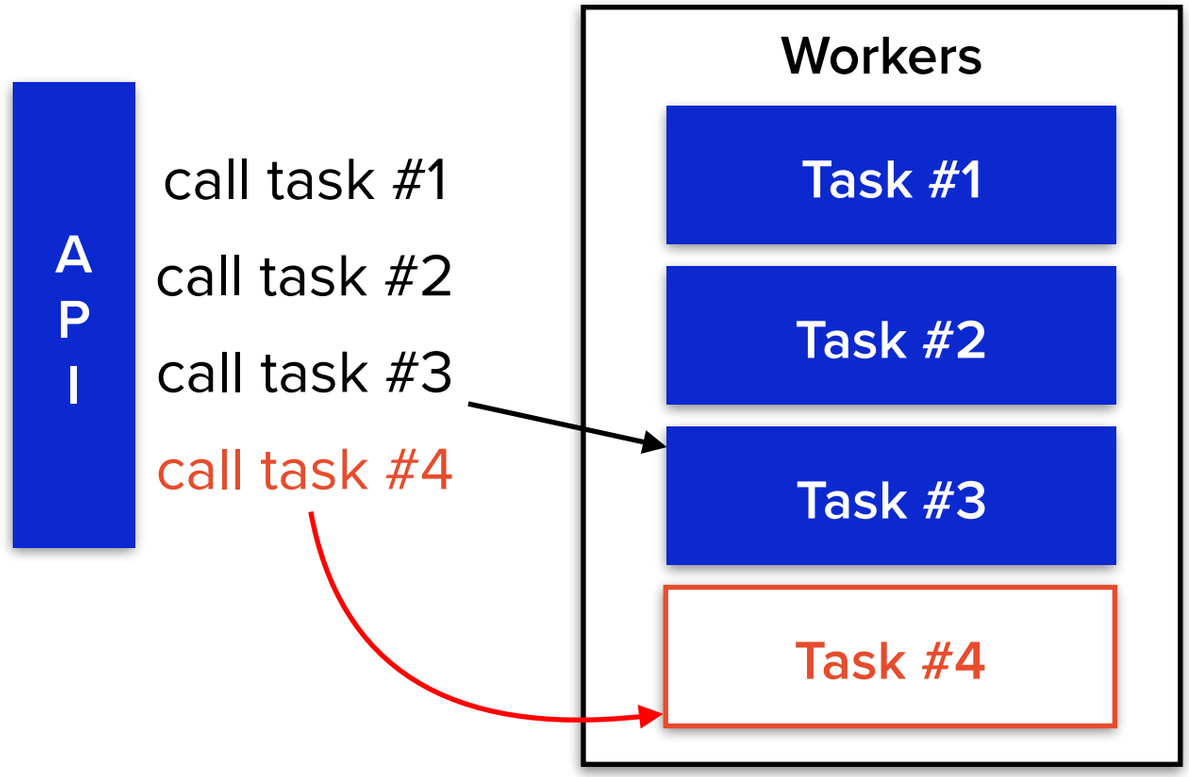
Async Task Execution

- Asynchronous Procedure Calls
- Queue based task execution
- Client must knows about their dependents (I've to change API every time I need to add a new task)

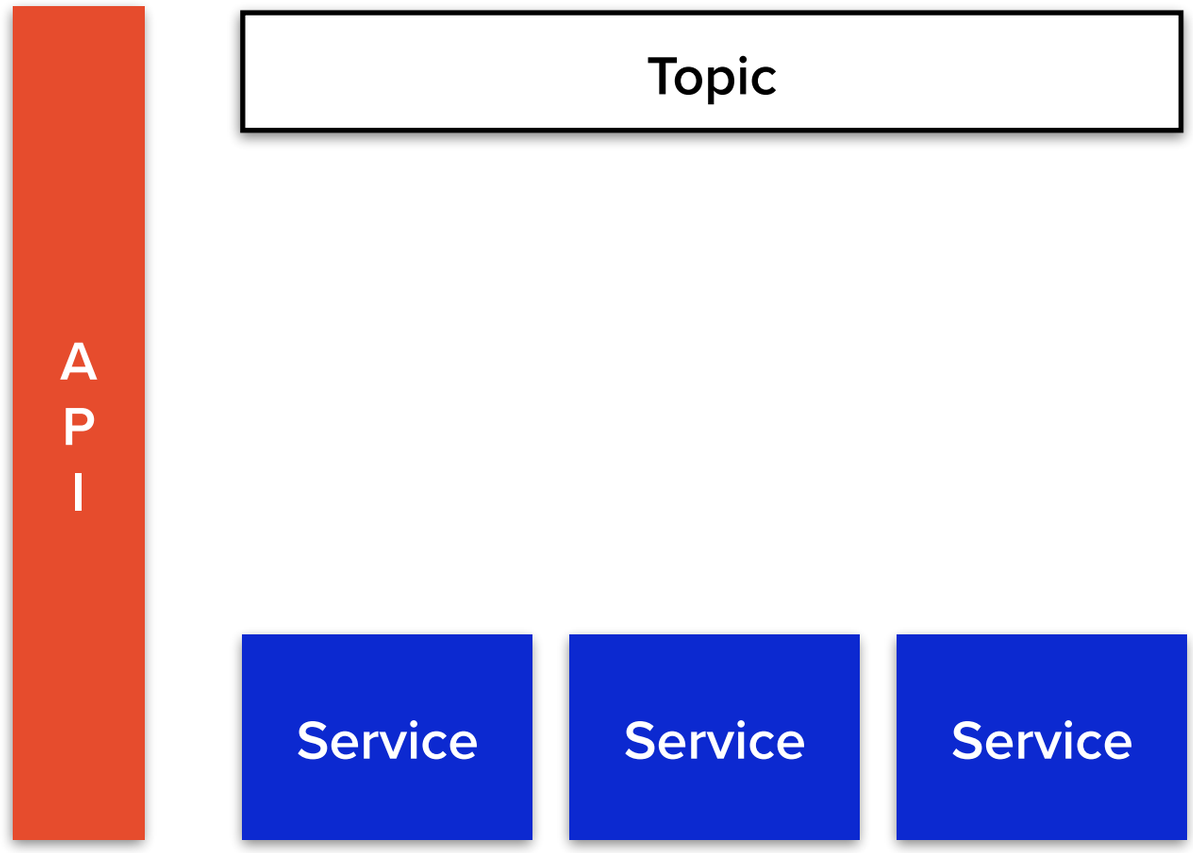


Async Task Execution

- Asynchronous Procedure Calls
- Queue based task execution
- Client must knows about their dependents (I've to change API every time I need to add a new task)
- No decoupling ➡ No modularity
- No safety* (eg. data loss on deployment failures)

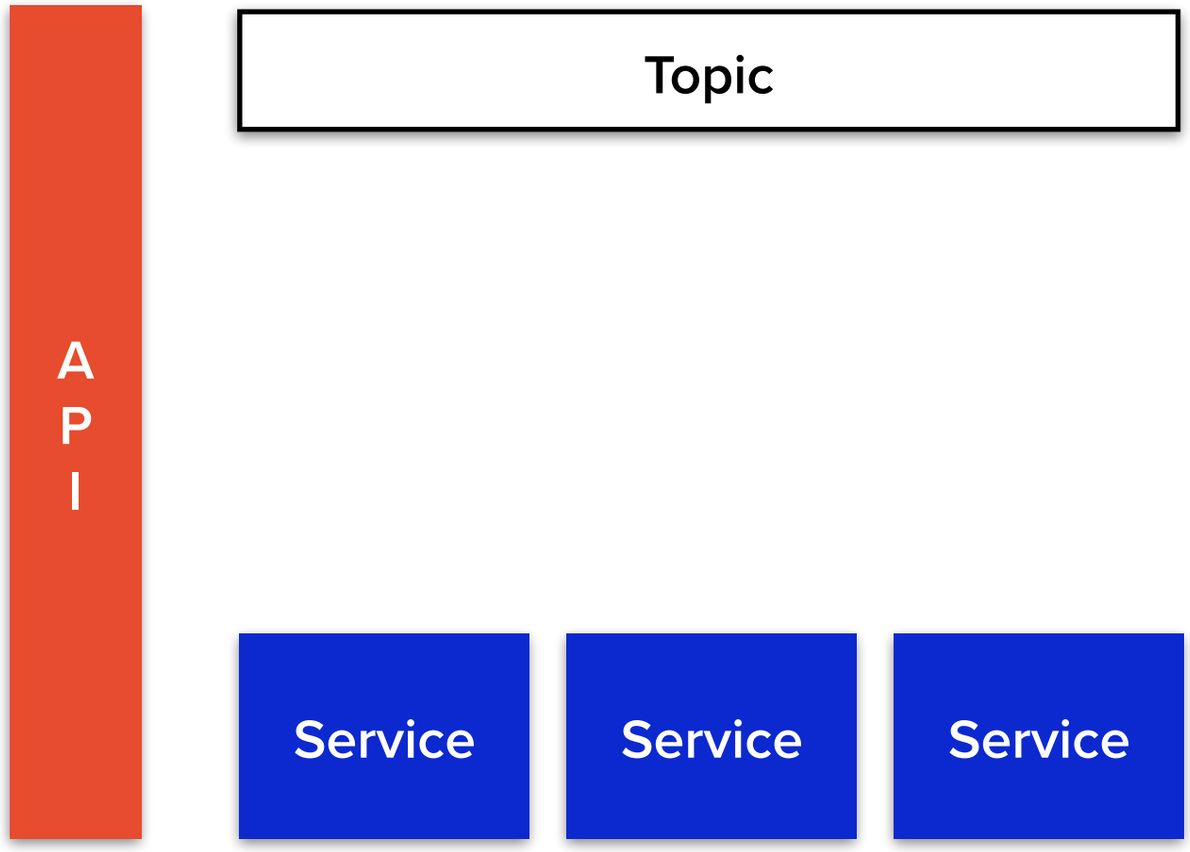


Message Event Triggering



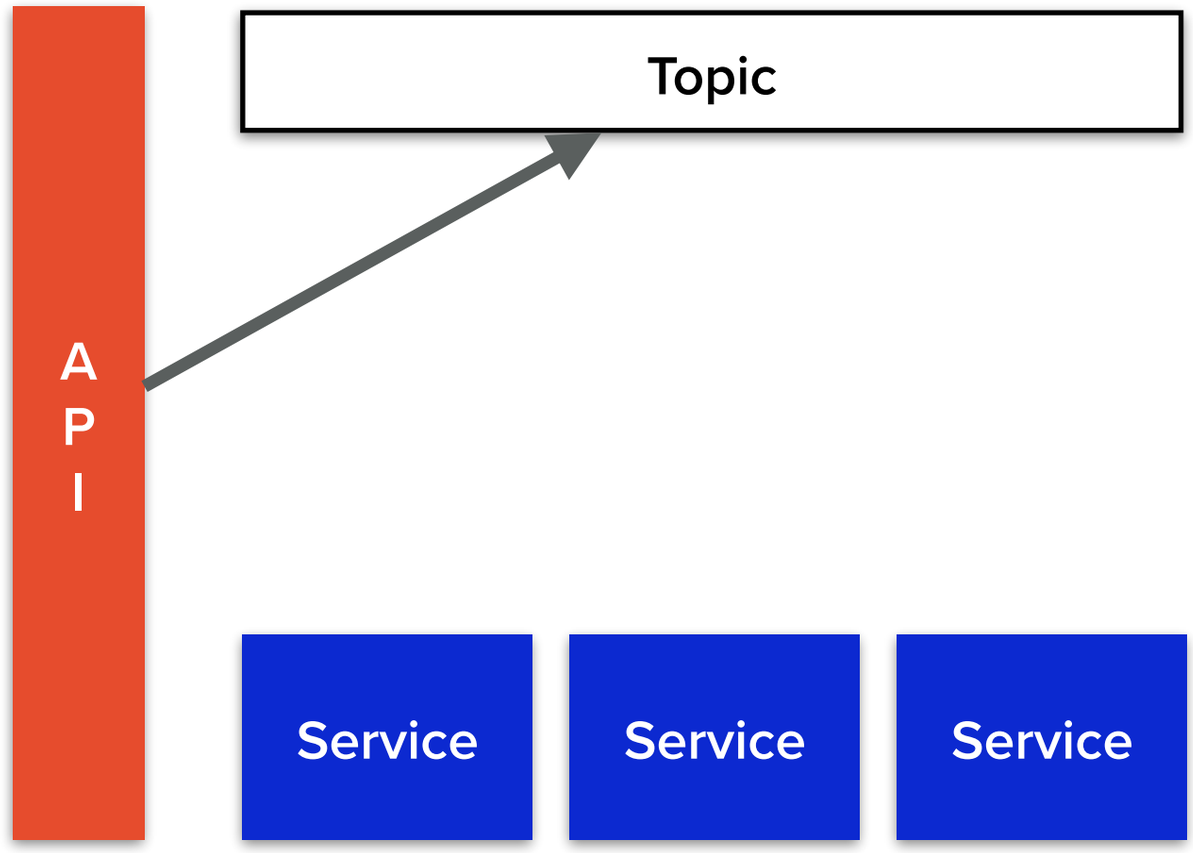
Message Event Triggering

- Action Event triggering
- Queue based message event handling
- Event as messages



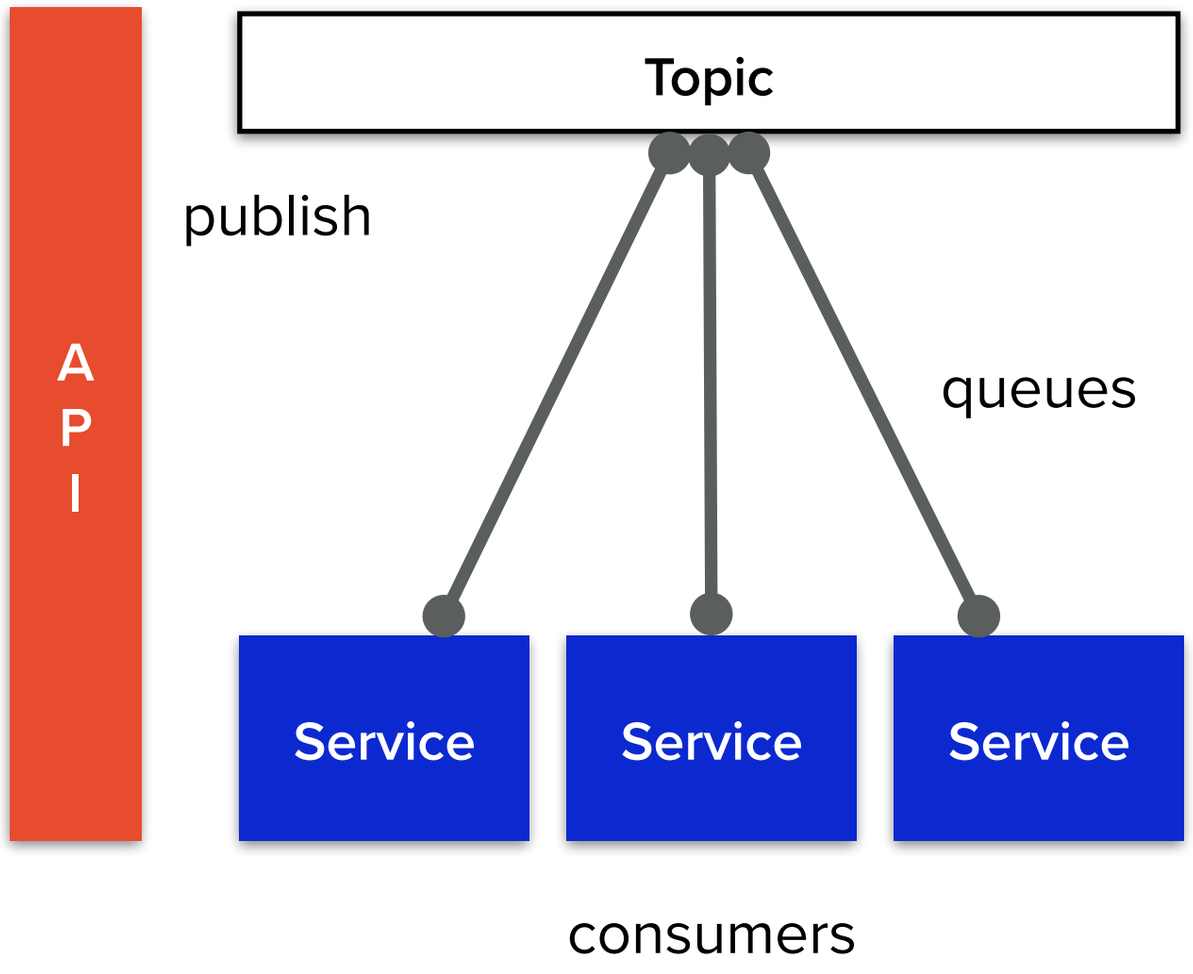
Message Event Triggering

- Action Event triggering
- Queue based message event handling
- Event as messages



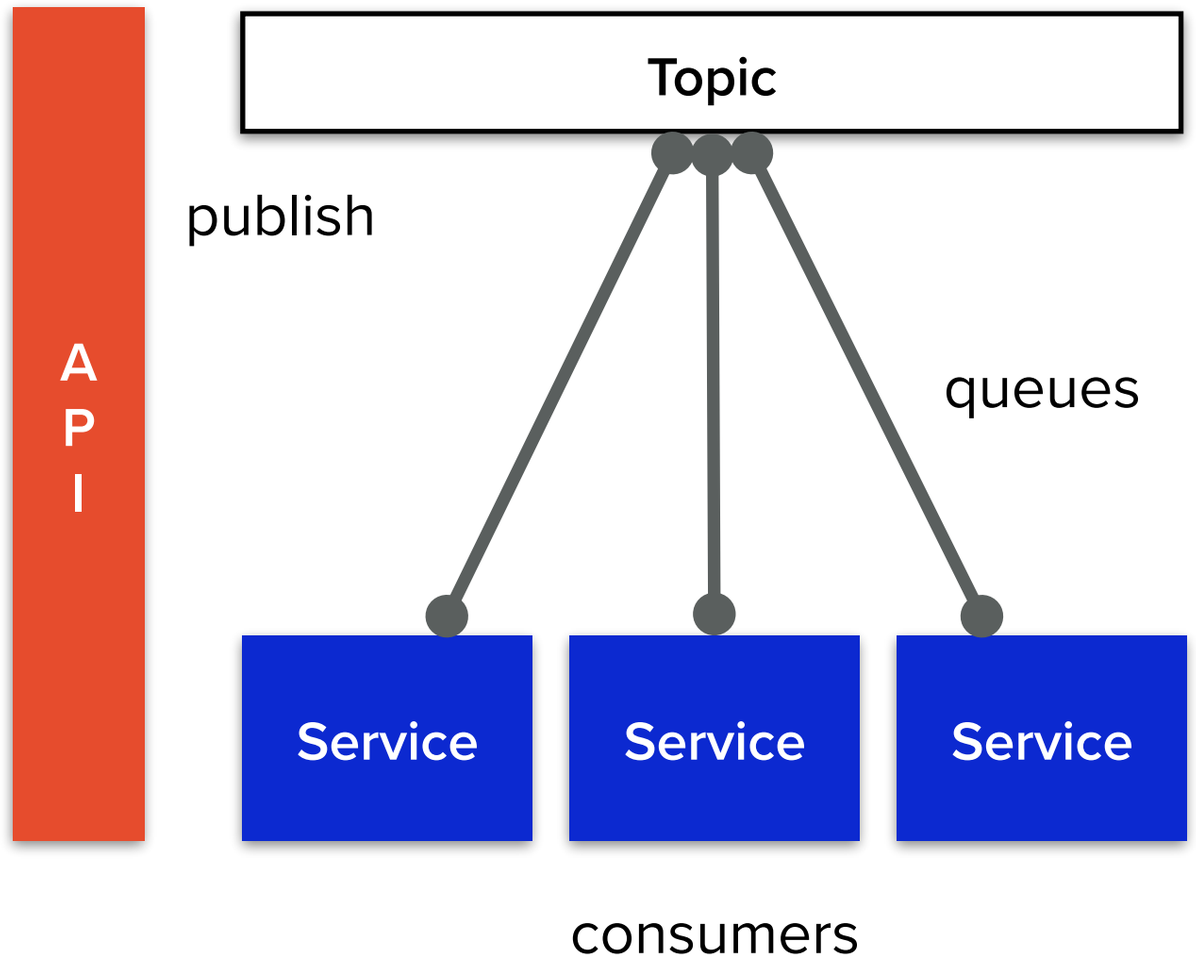
Message Event Triggering

- Action Event triggering
- Queue based message event handling
- Event as messages



Message Event Triggering

- Action Event triggering
- Queue based message event handling
- Event as messages
- That's it!



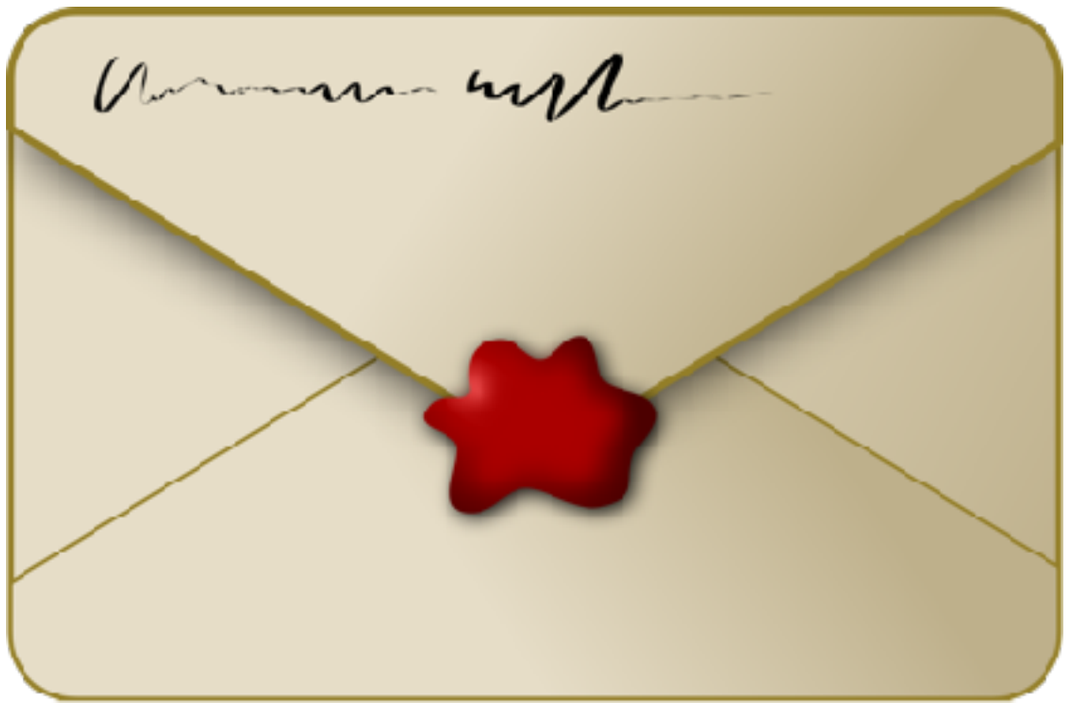
Microservices

Building Blocks

olist

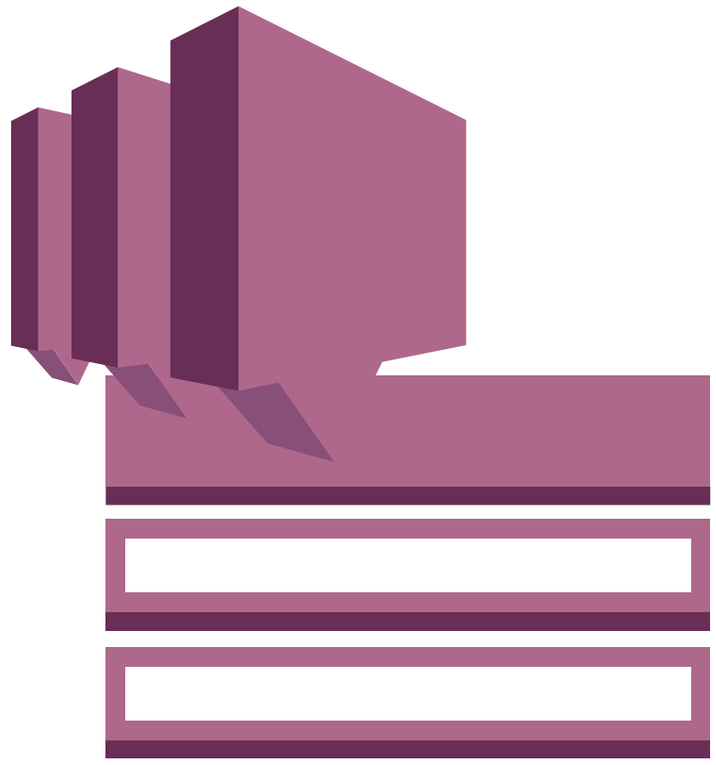
Messages

- Also known as Resource in REST context
- Follow a contract (schema)
- Can be wrapped with metadata (eg. SNS/SQS metadata)



Topics (global)

- Publisher in PubSub Pattern
- Global topics for message publication
- Topics belong to the system (or architecture) and not to a (micro)service
- AWS SNS



Service Queues

- Subscribers in PubSub Pattern
- Queues subscribe topics
- One queue belong exclusively to one (micro)service
- AWS SQS
  - SQS can be used as a SNS subscriber

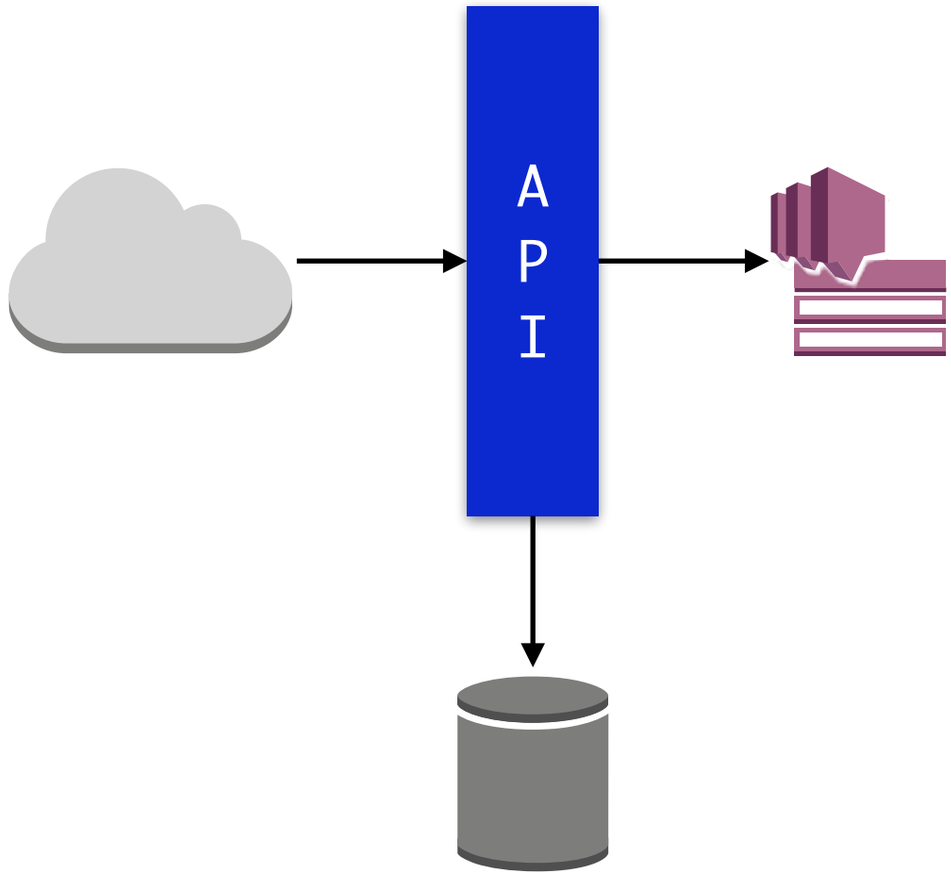
Microservices

Patterns

olist

API

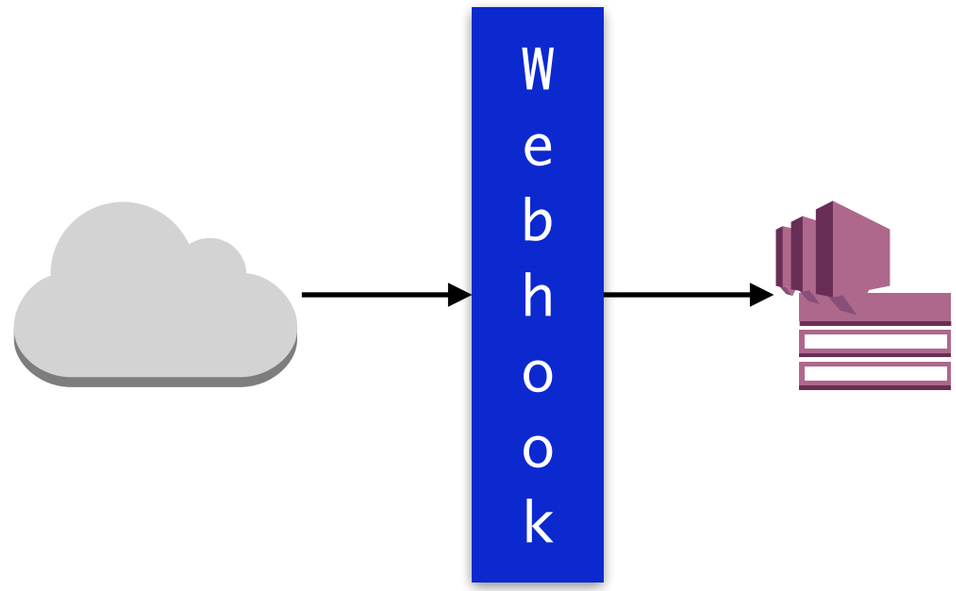
- Data Entry Point
- Data Validation
  - Workflow Management (eg. status/state machine)
- Data Persistence
- Event Triggering
- Idempotency handling (eg. discard duplicated requests returning a HTTP 304 Not Modified)
- Python, Django, DRF



olist

API (webhook)

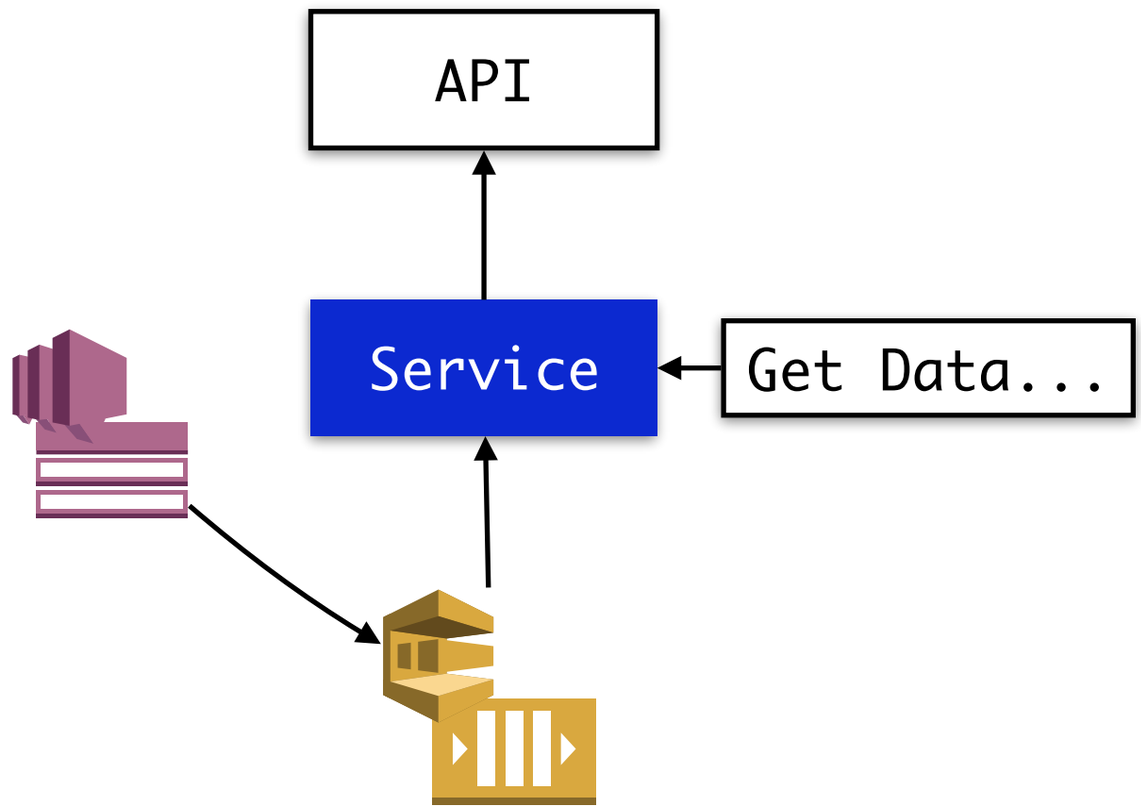
- Data Entry Point
- No Data Persistence
- Proxy HTTP ➡ SNS
- Event Triggering
- Python, Django, DRF



olist

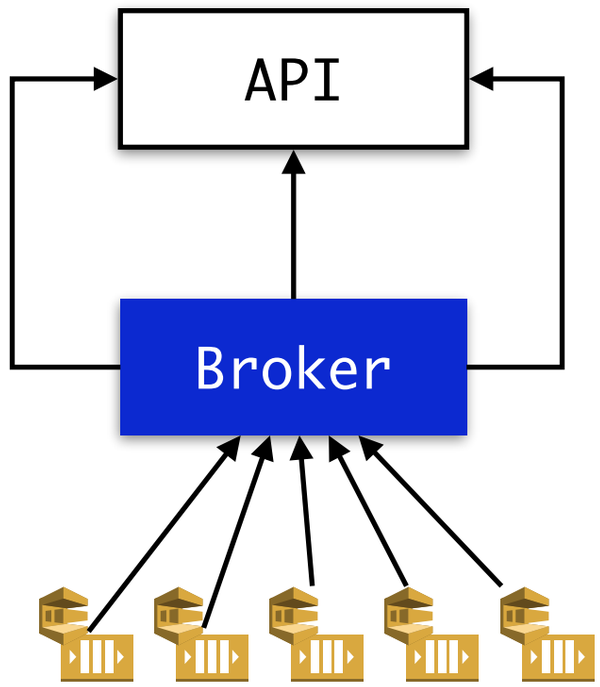
Service (consumer)

- Event Handling / Message Processing
- Business Logic
- Python, Loafer



Service (broker)

- Event Handling / Message Processing
- Business Logic
- Python, Loafer



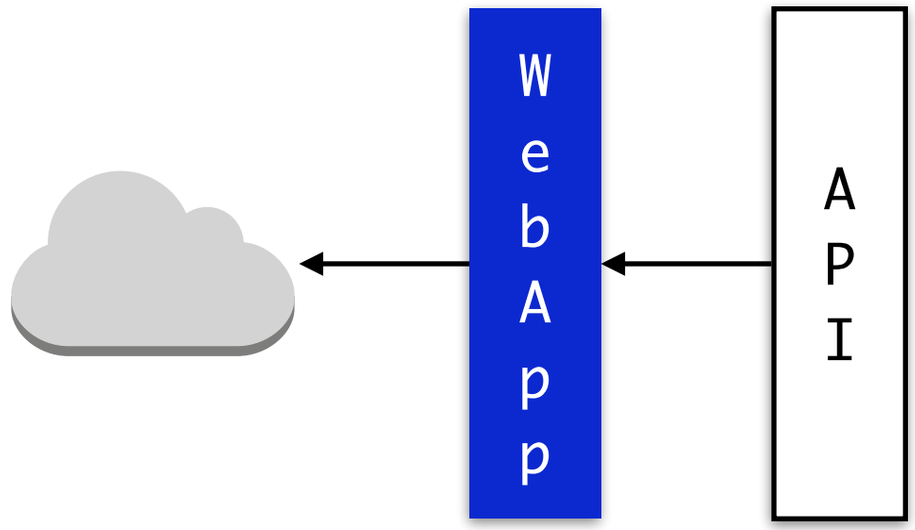
olist

Service (job)

- Scheduled Job
- No Persistence
- Event Triggering
- Python 3

Client

- Web or Mobile Applications
- No Persistence (or basic persistence)
- Web presentation of APIs' resources
- Python 3, Django



Libraries

- **Common Libraries** — common utilities for APIs and Services (eg. event triggering/topic publishing)

- **Client Libraries** — libraries to connect our APIs

- **Open Source Libraries** — useful libraries for community (eg. correios)

Tools

- **Data Migrator** — tool that connects in all databases and provides a small framework for data migration. It uses Kenneth Reiz's records library. It was initially used to migrate data from the old version of our application.

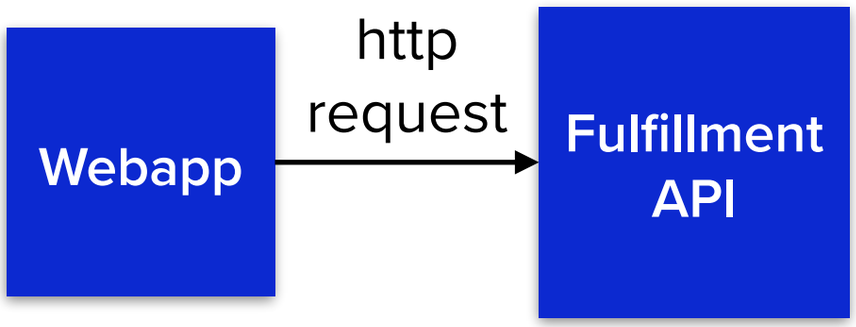
- **Toolbelt** — tool that provide basic management commands to interact with our APIs, partner APIs and to make ease to manage SQS queues or trigger some events in SNS Topics.

Working Sample

Webapp

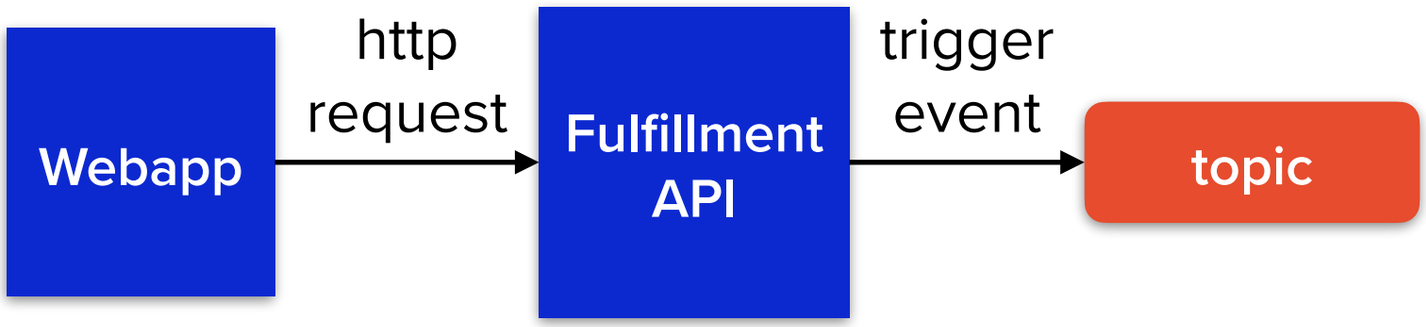
olist

Working Sample



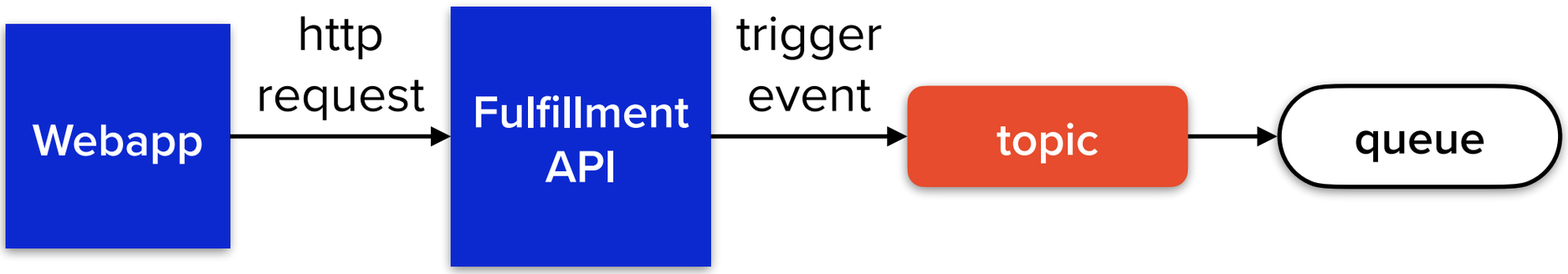
olist

Working Sample



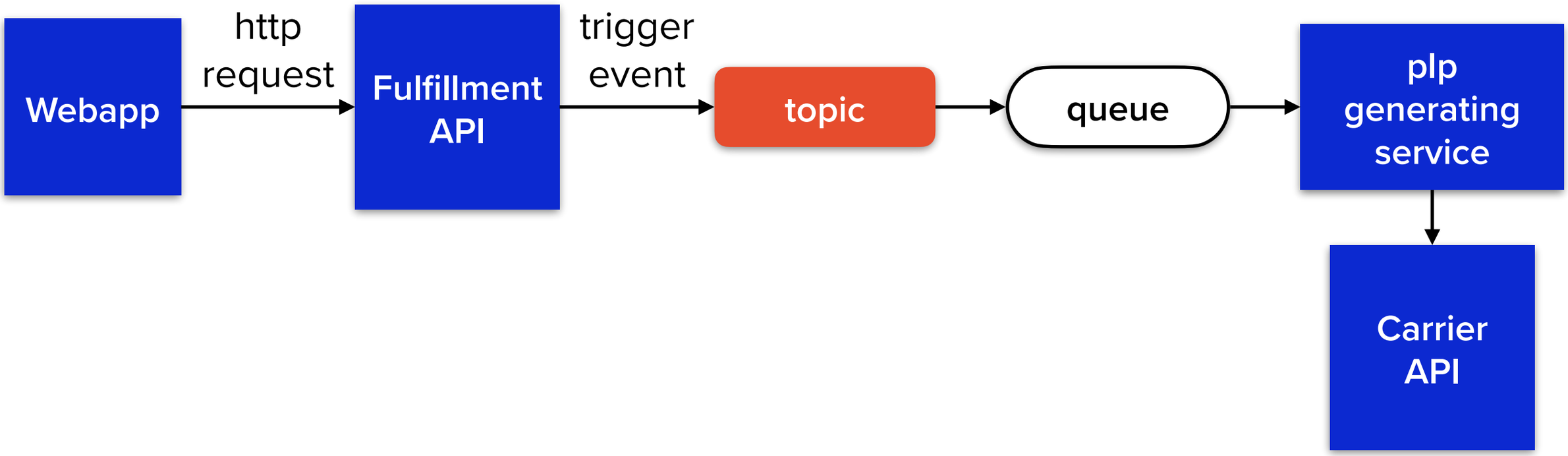
olist

Working Sample



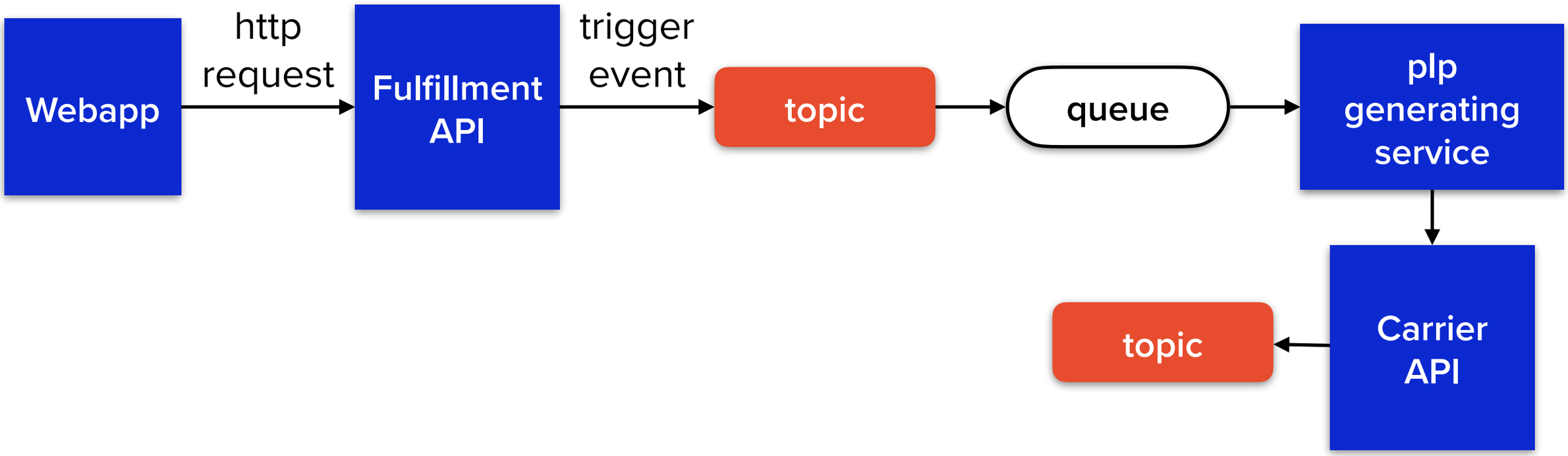
olist

Working Sample

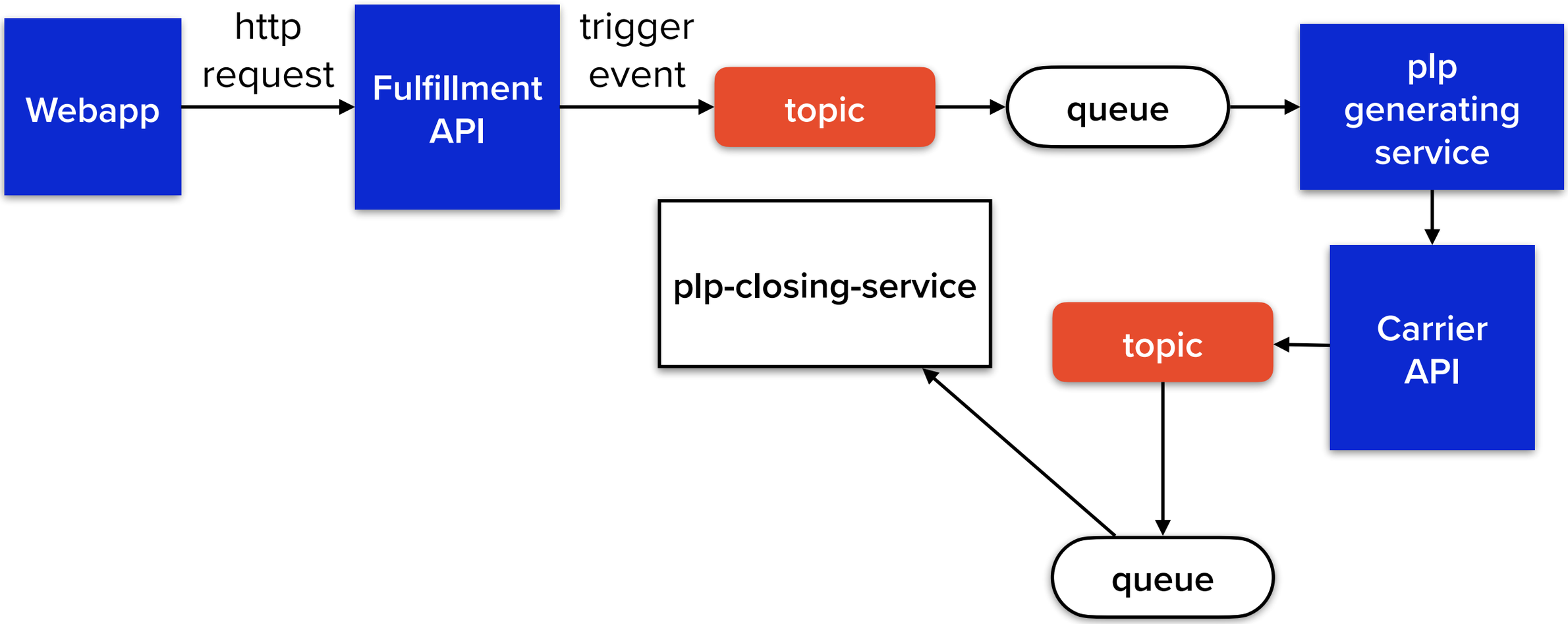


olist

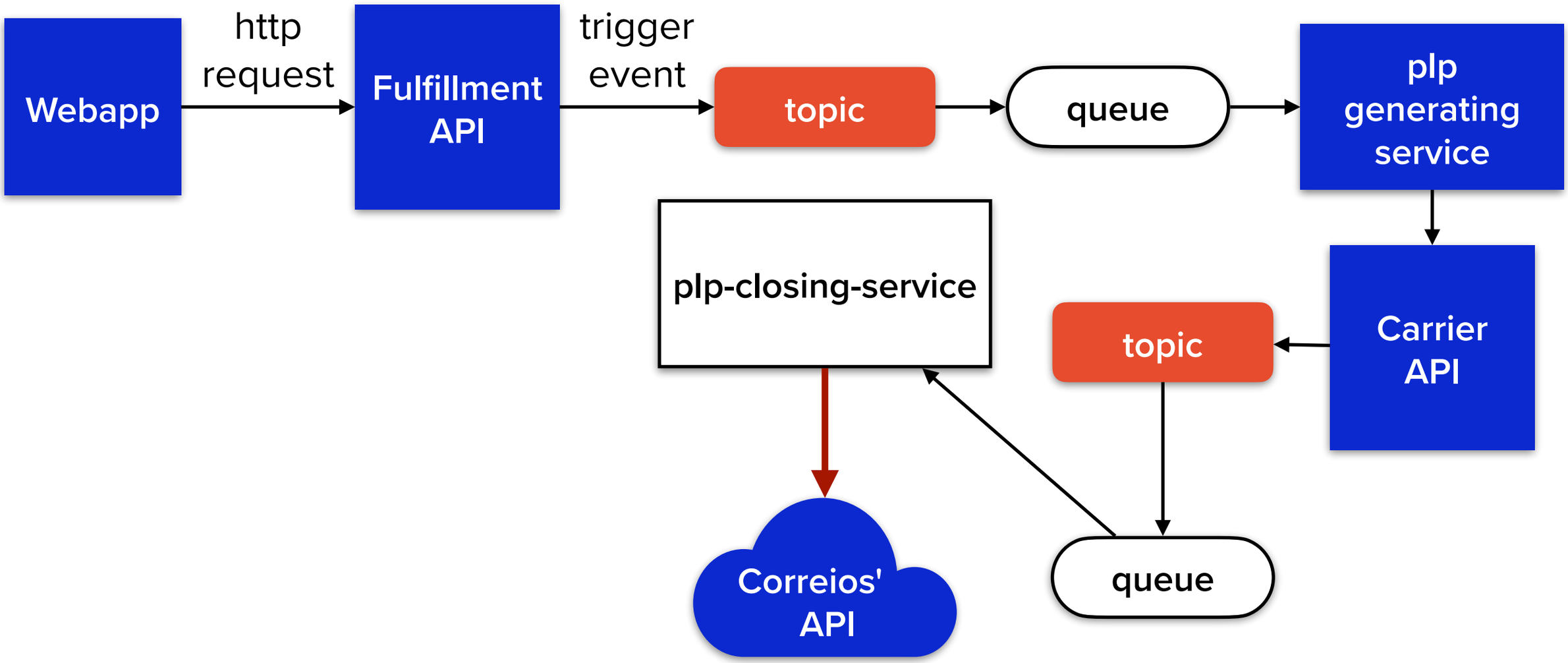
Working Sample



Working Sample

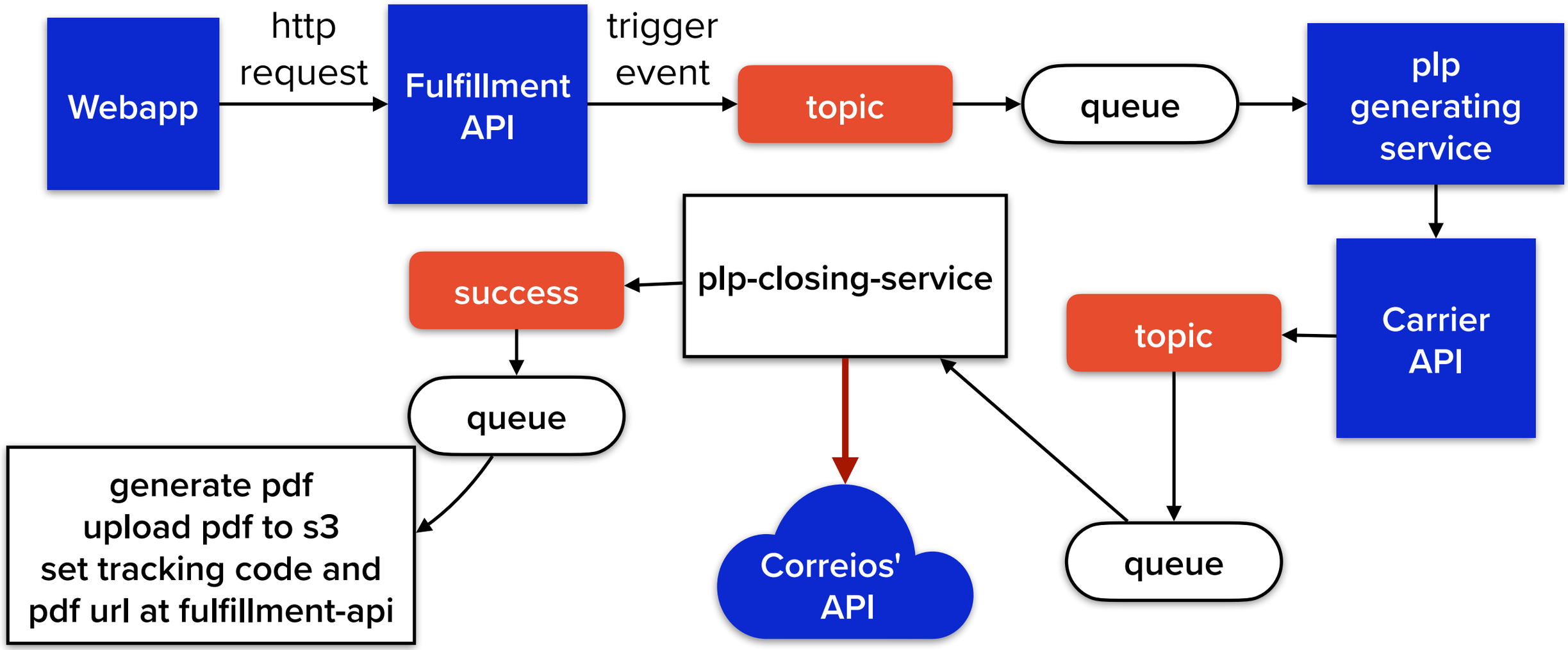


Working Sample

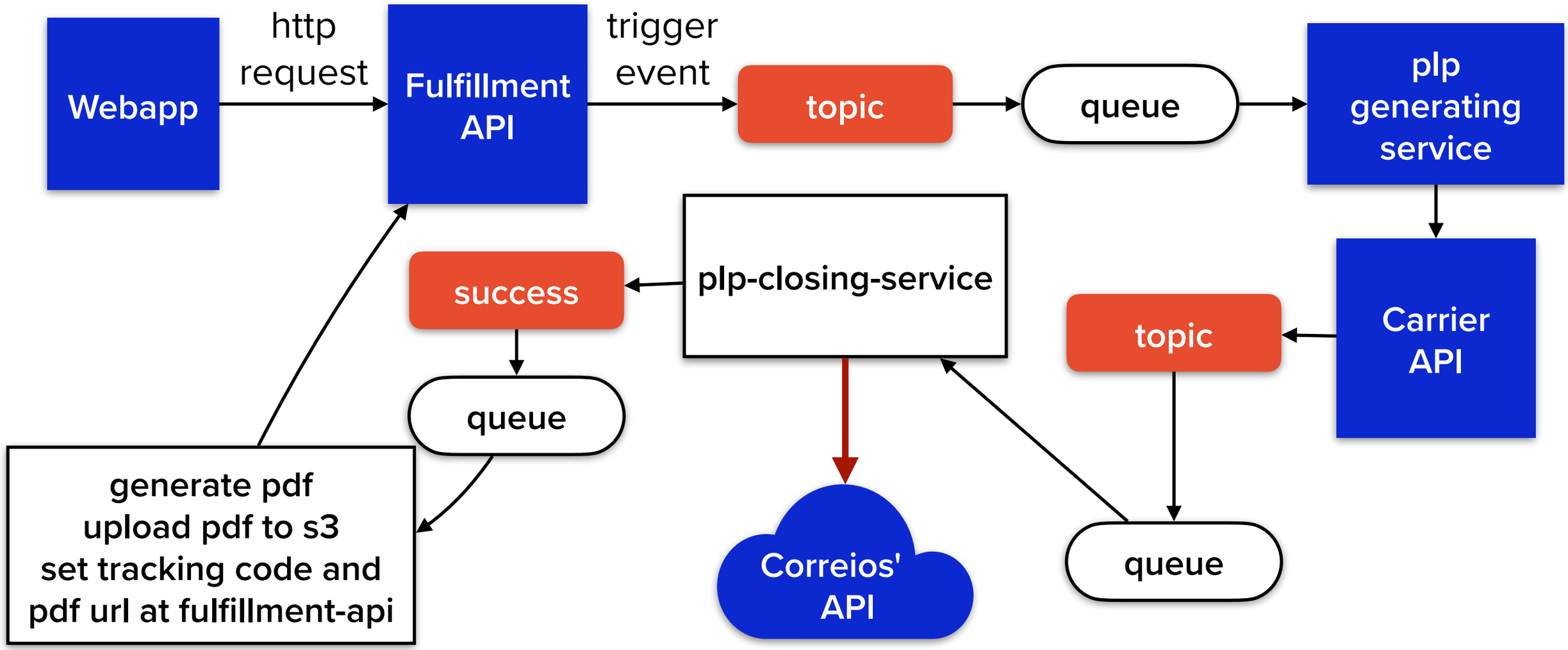


olist

Working Sample



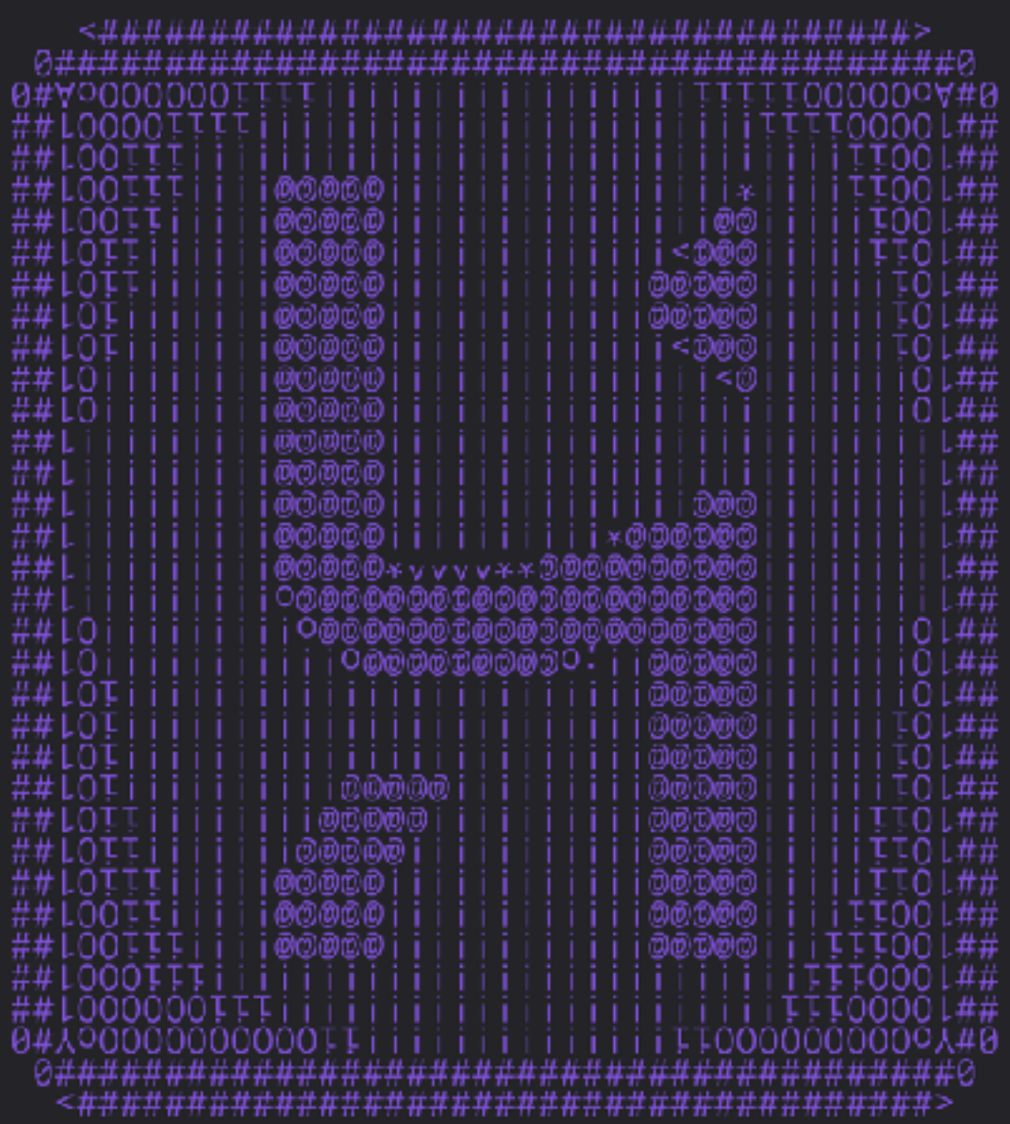
Working Sample



Microservices

Deployment

olist



amazon
web services

- Hosted on Heroku PaaS (our secret weapon!)
    - Easy deployment, configuration management, log handling, etc
- Heroku PostgreSQL Database
    - Easy deployment, easy configuration and setup, easy backup, replica and foreign data wrappers
- Other services and tools
    - Logentries, Sentry and New Relic

olist

Microservices

Challenges

olist

- It's hard to make evolution of message/resource contracts between services

- All sequential process must be splitted over multiple (small) services

- Denormalization of data can easily lead to problems of data consistency if we do not take certain precautions

- Information needed for one API must be replicated through services and stored locally

- Data migration or refactoring in several services requires the development of an specific application

olist

Development

olist

Development

Remote Team

olist

Development Tools

- Github - code management
- CircleCI - continuous integration
- vim / PyCharm / SublimeText / etc - development

Communication Tools

- JIRA - project management
- Confluence - documentation
- Google Apps - Mail, Calendar, Docs
- Slack - chat and monitoring integrations
- Mumble - voice conference
- tmux, ngrok, vim (and mumble) - pair programming

Perguntas?

Estamos contratando!

https://olist.com/trabalhe-conosco/