# Coatl - OSLC wrapper for APIs

OSLC Fest 2021
Juan Quintanar
juan.quintanar@koneksys.com

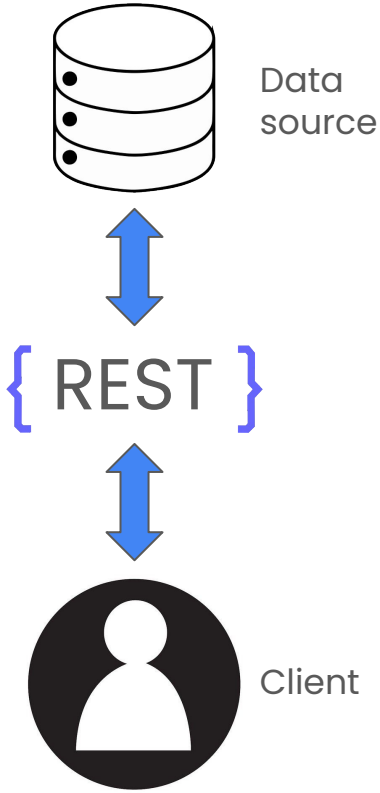
November 4, 2021

# Idea

Most web applications provide REST API architecture

REST APIs are good “data containers”, they provide a summary of application data.

REST API cannot connect with applications that work under OSLC specifications.



# Motivation

In order to create OSLC API developers need to employ a certain amount of time to accomplish it but...

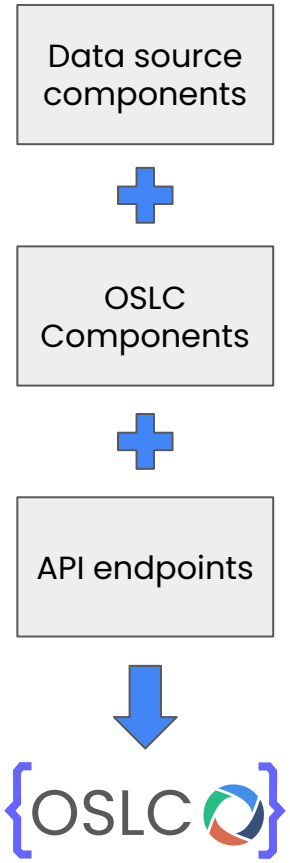
What if:

**Develop another OSLC API**?

N**eed the OSLC API now**?

**Partial solution:** we can reuse some OSLC API components such as:

- ❏ **Data source components** → Fetch data
- ❏ **OSLC Components** → Parse data into RDF
- ❏ **API endpoints** → HTTP requests



# Inspiration

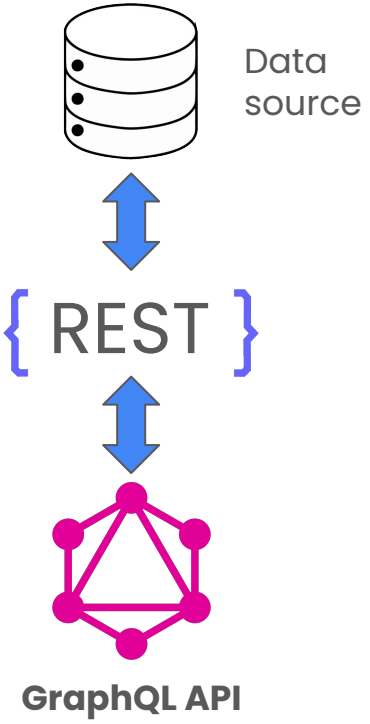
**GraphQL:**

- Query language for APIs
- Clients requests only data that they need

GraphQL server such as Apollo or Graphene can be employed to wrap a REST API and make requests to the REST API.

In order to wrap REST APIs the server uses modules:

- **Type definitions** → Data model
- **Resolvers** → Functions to perform mapping



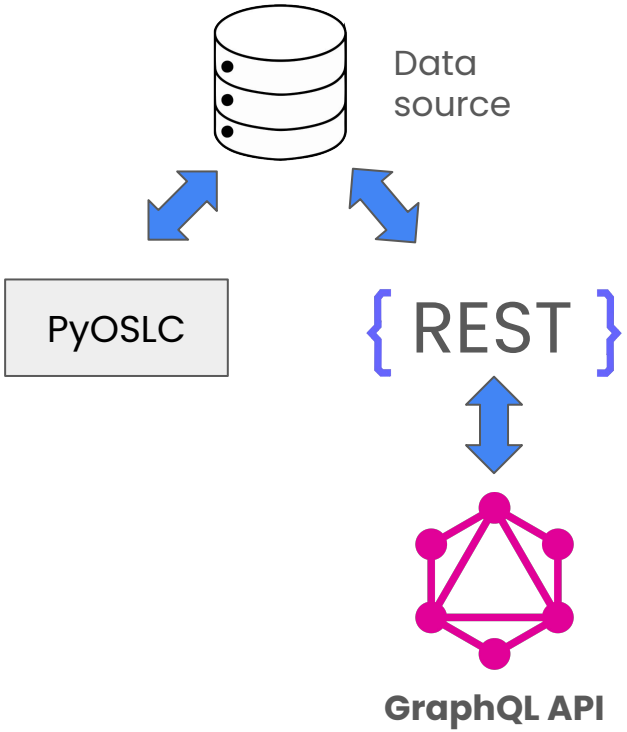
# Inspiration

**PyOSLC:**

- SDK based on Python
- Expose databases as OSLC SP
- Mapping of data sources into RDF

PyOSLC perform its operations by using modules:

- **RDF Mapping**
- **OSLC resources** described as Python objects

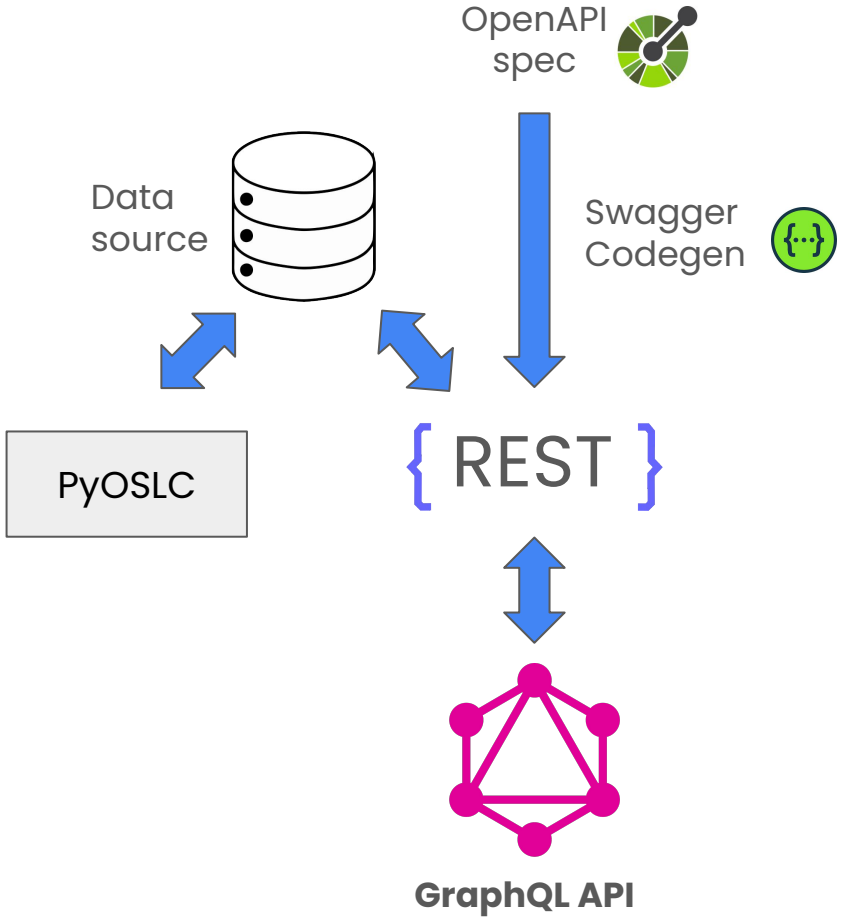


# Inspiration

**OpenAPI Specification (OA Spec)** → API description format for REST APIs.

**Swagger** → Set of open-source tools for designing, building and documenting REST APIs.

**Swagger Codegen** → Generates server stub, client libraries from an OpenAPI spec, create/use custom generators.



# What about creating a similar approach using OSLC?

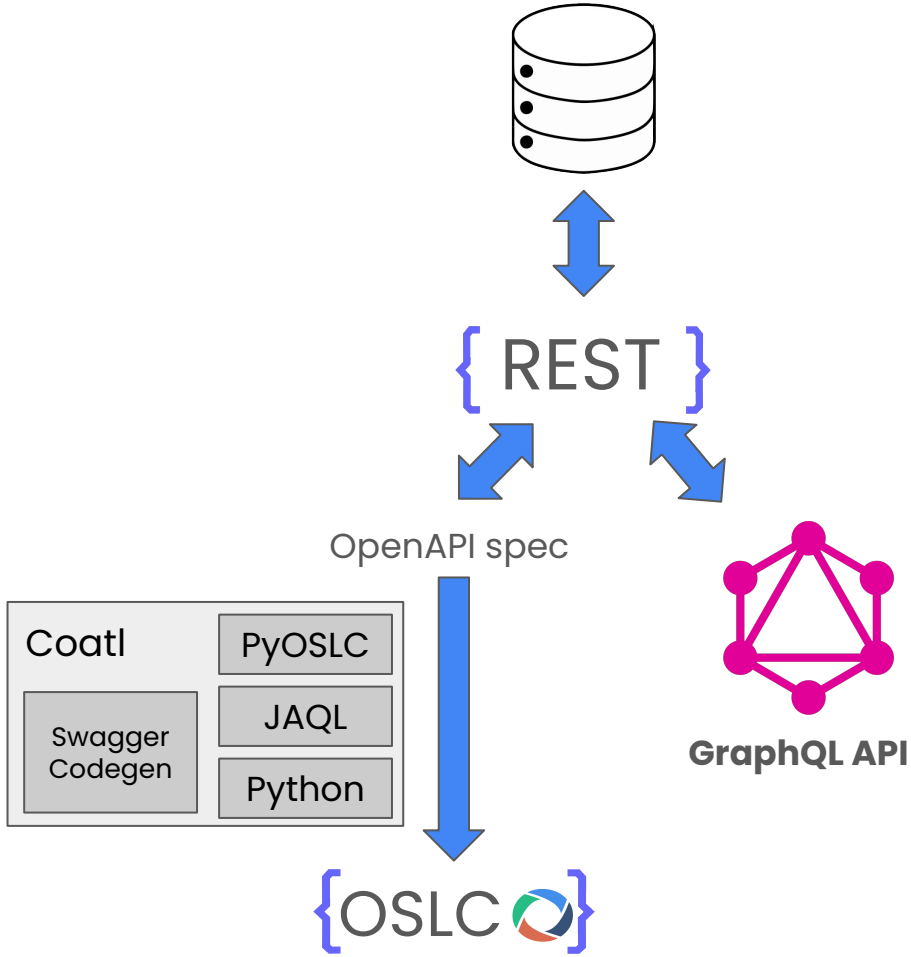
# Solution

**Coatl** → OSLC wrapper for REST APIs

Based on **Swagger Codegen / Python**

**Necessary:**

- REST API
- Swagger Codegen libraries
- Open API document

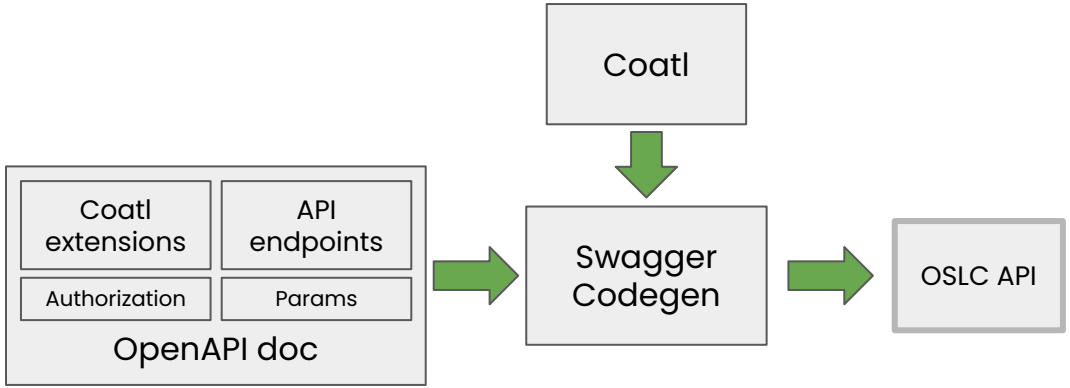


# How Coatl works?

Coatl works by documenting REST API data into OA spec with extensions.

**Input**: OpenAPI spec file

**Output**: OSLC API



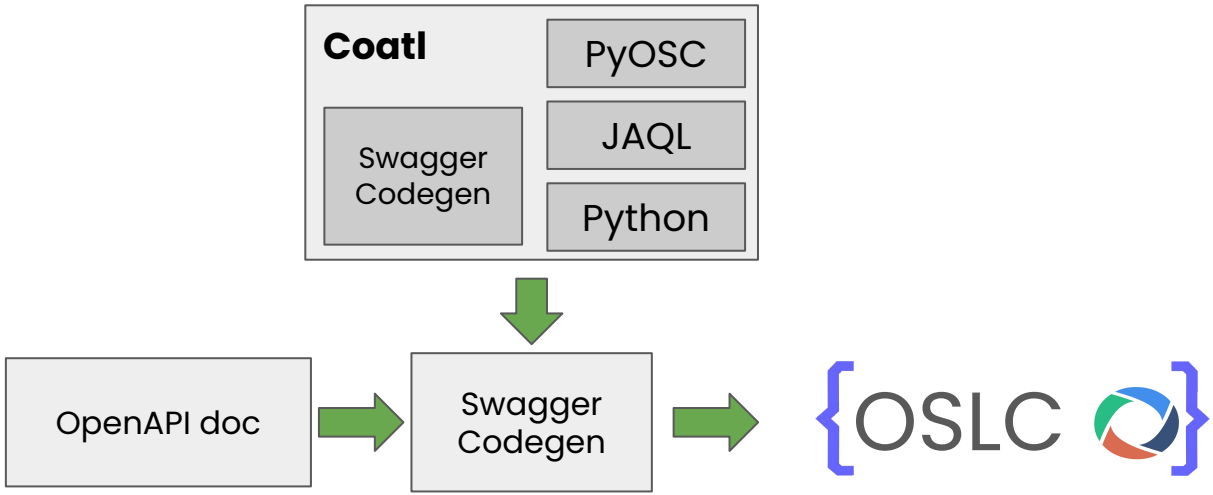

Specify:

- OSLC API endpoints
- Coatl extensions:
  - REST API → OSLC API data source
  - JSON keys of REST→ RDF properties

# How to use Coatl?

You need to perform two operations:

1. **Create OpenAPI spec**
2. **Generate OSLC API** by using Swagger Codegen



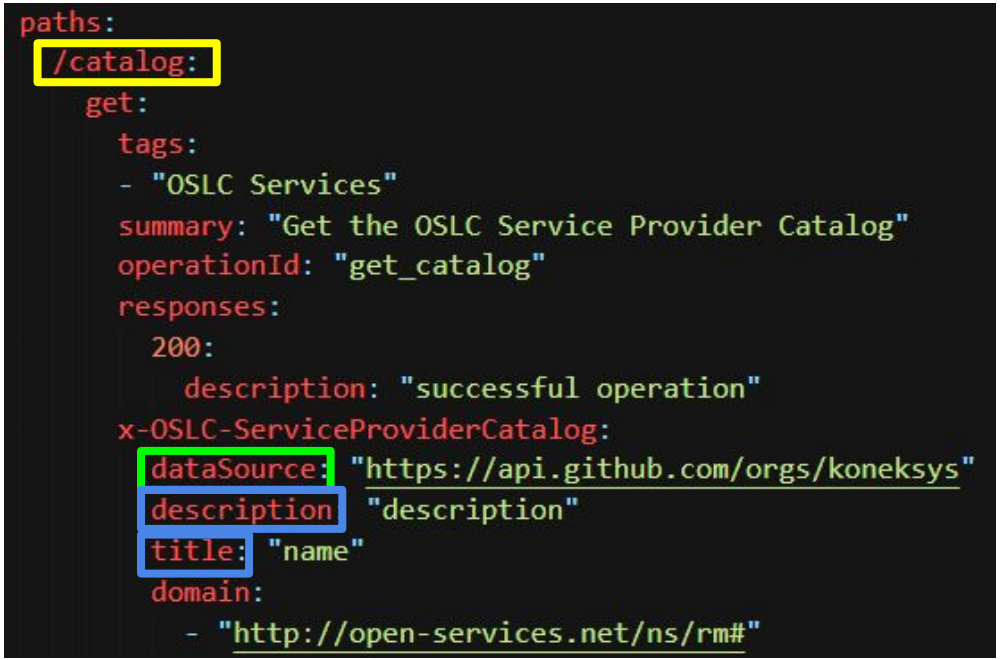
# 1.- OpenAPI document

Specify an OpenAPI document:

OSLC API endpoints

Via vendor extensions:

- **REST API URLs** → OSLC API data source
- **JSON keys** → RDF properties



# OpenAPI Coatl Extensions for OSLC

The following extensions are employed to map REST API URL/data into OSLC Structure:

**Operations**

- x-Jazz-Root Services
- x-OSLC-Service Provider Catalog
- x-OSLC-ServiceProvider
- x-OSLC-Services
  - Query Capability
  - UI dialog
- x-OSLC-UI-Preview

**Endpoints**

- x-OSLC-QueryCapability-endpoint
- x-OSLC-Publisher-endpoint
- x-OSLC-ResourceShape-endpoint
- x-OSLC-SelectDialog-endpoint
- x-Jazz-Configuration-Catalog

**Models**

- x-OSLC-ResourceProperties
- x-RDF-Vocabulary

**Configuration Management**

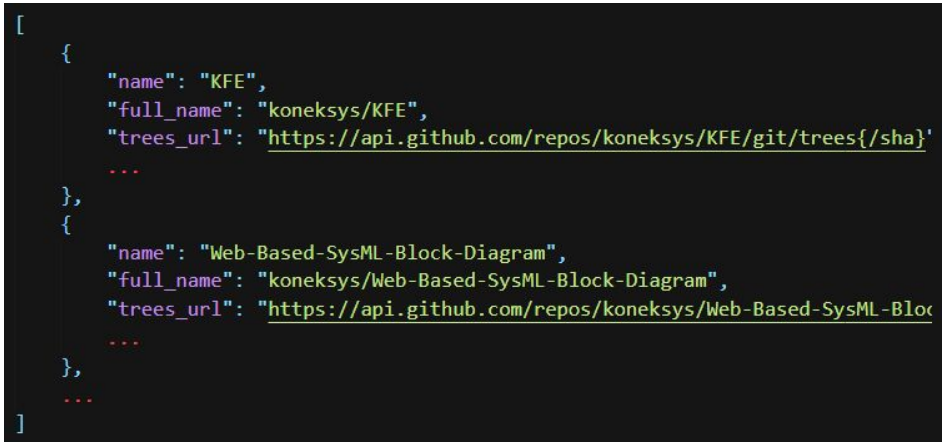
- x-OSLC-Components
- x-OSLC-Baseline
- x-OSLC-Stream
- x-OSLC-Configuration-Selection
- x-OSLC-Configuration-Versioned-Resource

# Example: REST API endpoints

The data provided by Github API endpoints will be employed to create a Github OSLC API resources.

GET: https://api.github.com/orgs/koneksys


GET: https://api.github.com/orgs/koneksys/repos



# OSLC API data source

REST resource describing github account.

Github API URL:

https://api.github.com/orgs/koneksys

```
{
    "login": "koneksys",
    "id": 16809825,
    ...
    "repos_url": "https://api.github.com/orgs/koneksys/repos",
    "description": "Software services based on state-of-the-ar
    "name": "Koneksys"
}
```

OpenAPI spec with Coatl extensions for OSLC Service Provider Catalog creation.

```
x-OSLC-ServiceProviderCatalog:
  dataSource: "https://api.github.com/orgs/koneksys"
  description: "description"
  title: "name"
  domain:
    - "http://open-services.net/ns/rm#"
```

OSLC Service Provider Catalog resource describing github account.

```
@prefix ns1: <http://open-services.net/ns/core#> .
@prefix ns2: <http://purl.org/dc/terms/> .

<https://github-oslc-api/oslc/catalog> a ns1:ServiceProviderCatalo
    ns2:description "Software services based on state-of-the-art n
    ns2:publisher <https://github-oslc-api/oslc/publisher> ;
    ns2:title "Koneksys" .
```

# JSON keys → RDF

**JSON Query Language** (JAQL) is employed to parse JSON data into RDF properties

**JAQL** → Handle stored data in JSON documents.

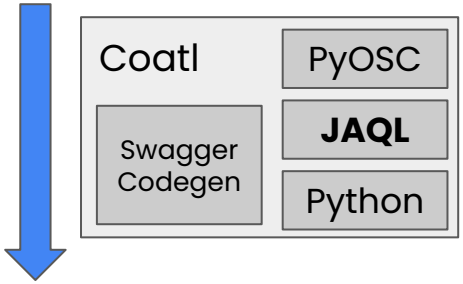
JAQL provides tools to perform other operations such as:

- ❏ Get keys from JSON nested data
- ❏ Multiple Selection
- ❏ Functions expressions

Coatl contains libraries for managing JAQL queries.

REST resource describing github repository.

```
[
    {
        "name": "KFE",
        "full_name": "koneksys/KFE",
        "trees_url": "https://api.github.com/repos/koneksys/
        ...
    }
]
```




Modified resource describing OSLC Service Provider

```
[
    {
        "SPtitle": "KFE",
        "SPdescription": "koneksys/KFE",
        "OslcServiceDataSource": "https://api.github.com/re
        ...
    }
]
```

# JSON keys → RDF

REST resource describing github account.

Github API URL:

https://api.github.com/orgs/koneksys

```
{
    "login": "koneksys",
    "id": 16809825,
    ...
    "repos_url": "https://api.github.com/orgs/koneksys/repos",
    "description": "Software services based on state-of-the-ar
    "name": "Koneksys"
}
```

OpenAPI spec with Coatl extensions for OSLC Service Provider Catalog creation.

```
x-OSLC-ServiceProviderCatalog:
  dataSource: "https://api.github.com/orgs/koneksys"
  description: "description"
  title: "name"
  domain:
    - "http://open-services.net/ns/rm#"
```

OSLC Service Provider Catalog resource describing github account.

```
@prefix ns1: <http://open-services.net/ns/core#> .
@prefix ns2: <http://purl.org/dc/terms/> .

<https://github-oslc-api/oslc/catalog> a ns1:ServiceProviderCatalo
    ns2:description "Software services based on state-of-the-art r
    ns2:publisher <https://github-oslc-api/oslc/publisher> ;
    ns2:title "Koneksys" .
```

# JSON keys → RDF

REST resource describing github repository.
Github API URL:
https://api.github.com/orgs/koneksys/repo

```
[
    {
        "name": "KFE",
        "full_name": "koneksys/KFE",
        "trees_url": "https://api.github.com/repos/koneksys/KFE/git/tr
        ...
    },
    {
```

OpenAPI spec with Coatl extensions for OSLC Service Provider creation.

```
x-OSLC-ServiceProvider:
  serviceProviderId: '[:].name'
  title: '[:].name'
  description: '[:].full_name'
```

OSLC Service Provider resource describing github repository.

```
<https://github-oslc-api/oslc/provider/KFE> a ns1:Servic
    ns2:description 'koneksys/KFE' ;
    ns2:identifier "KFE"^^xsd:string ;
    ns2:publisher <http://127.0.0.1:5000/oslc/publisher>
    ns2:title 'KFE'^^xsd:Literal .
```

# JSON keys → RDF

REST resource describing github files.

Github API URL:

https://api.github.com/repos/koneksys/KFE/git/trees/master?recursive=1

```
{
    "tree": [
        {
            "path": "FEA_model/FEA_Model_1.py",
            "size": 5028,
            ...
        },
```

OpenAPI spec with Coatl extensions for OSLC Query Capability service creation.

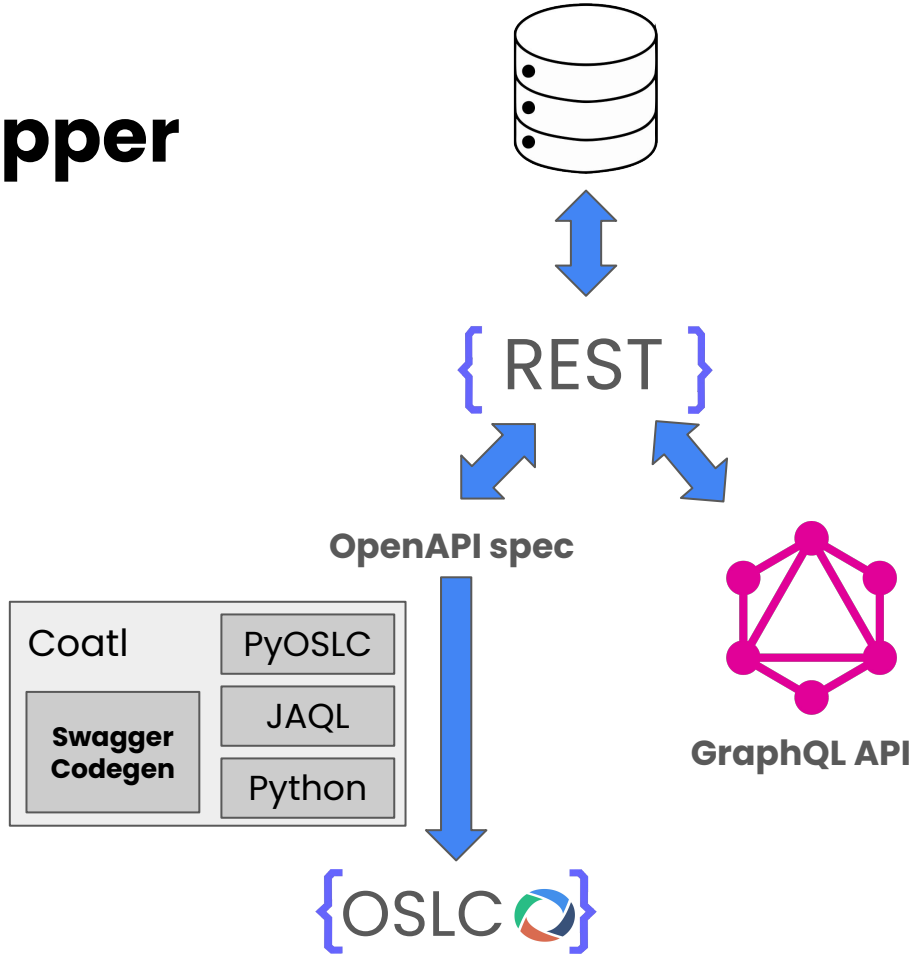
```
x-OSLC-Services:
  - queryCapability: "github_files"
    dataSource: '[:].join(``, [to_string(trees_url),`/master?recursive=1`])
    query: "tree[:].{id: replace_all_substrings(path,`/`,`-`), title: repla
```

OSLC artifact describing a single github file.

```
<https://github-oslc-api/oslc/provider/KFE/resources/githu
    ns3:instanceShape <https://github-oslc-api/oslc/provid
    ns3:serviceProvider <https://github-oslc-api/oslc/prov
    ns2:identifier "FEA_model-FEA_Model_1.py" ;
    ns2:title "FEA model-FEA Model 1.py" ;
    ns1:size 5028
```

# 2.- Create OSLC wrapper

OSLC API will be created through Swagger Codegen and Coatl library.



# Result → Live demo

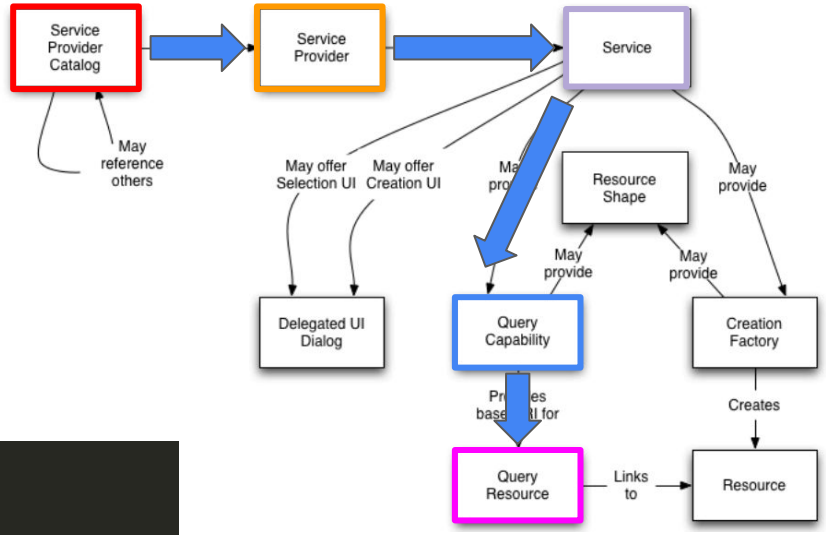
The information provided by REST API is exposed through OSLC Resources.

**Github API example:**

```
@prefix ns1: <http://open-services.net/ns/core#> .
@prefix ns2: <http://purl.org/dc/terms/> .

<http://127.0.0.1:5000/oslc/catalog> a ns1:ServiceProviderCatalog ;
    ns1:domain <http://open-services.net/ns/rm#> ;
    ns1:serviceProvider <http://127.0.0.1:5000/oslc/provider/Blockchain4LinkedData>,
        <http://127.0.0.1:5000/oslc/provider/Git4RDF>,
        <http://127.0.0.1:5000/oslc/provider/KFE>,
        <http://127.0.0.1:5000/oslc/provider/KLD>,
        <http://127.0.0.1:5000/oslc/provider/SPARQL_to_GraphFrames>,
        <http://127.0.0.1:5000/oslc/provider/Web-Based-SysML-Block-Diagram>,
        <http://127.0.0.1:5000/oslc/provider/aras-oslc>,
        <http://127.0.0.1:5000/oslc/provider/cameo2rdf>,
        <http://127.0.0.1:5000/oslc/provider/oslc_openapi_extensions> ;
    ns2:description "Software services based on state-of-the-art non-proprietary web
    ns2:publisher <http://127.0.0.1:5000/oslc/publisher> ;
    ns2:title "Koneksys" .
```

```
<http://127.0.0.1:5000/oslc/provider/KFE> a ns2:ServiceProvider ;
    ns3:globalConfigurationAware "yes"^^xsd:String ;
    ns3:supportContributionsToLinkIndexProvider true ;
    ns3:supportLinkDiscoveryViaLinkIndexProvider true ;
    ns3:supportOSLCSimpleQuery true ;
    ns2:details <http://127.0.0.1:5000/oslc/provider/KFE> ;
    ns2:service [ a ns2:Service ;
            ns2:domain <http://open-services.net/ns/rm#> ;
            ns2:queryCapability [ a ns2:QueryCapability
                    ns2:label "Query Capability for repository content (SP: KFE)"^^xsd:string ;
                    ns2:queryBase <http://127.0.0.1:5000/oslc/provider/KFE/resources/github_files>
                    ns2:resourceShape <http://127.0.0.1:5000/oslc/provider/KFE/resourceShape/github_files> ;
```



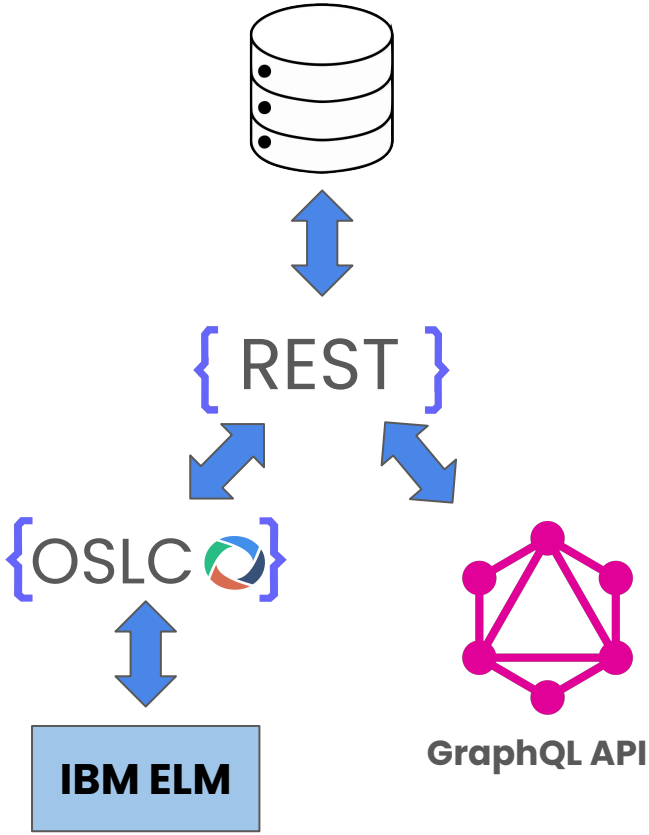
# Coatl extensions for integration with IBM ELM™

# IBM ELM™

- Provides integrated end-to-end solution across all engineering data.
- Optimizes collaboration and communication across all stakeholders.

**OSLC APIs integration**

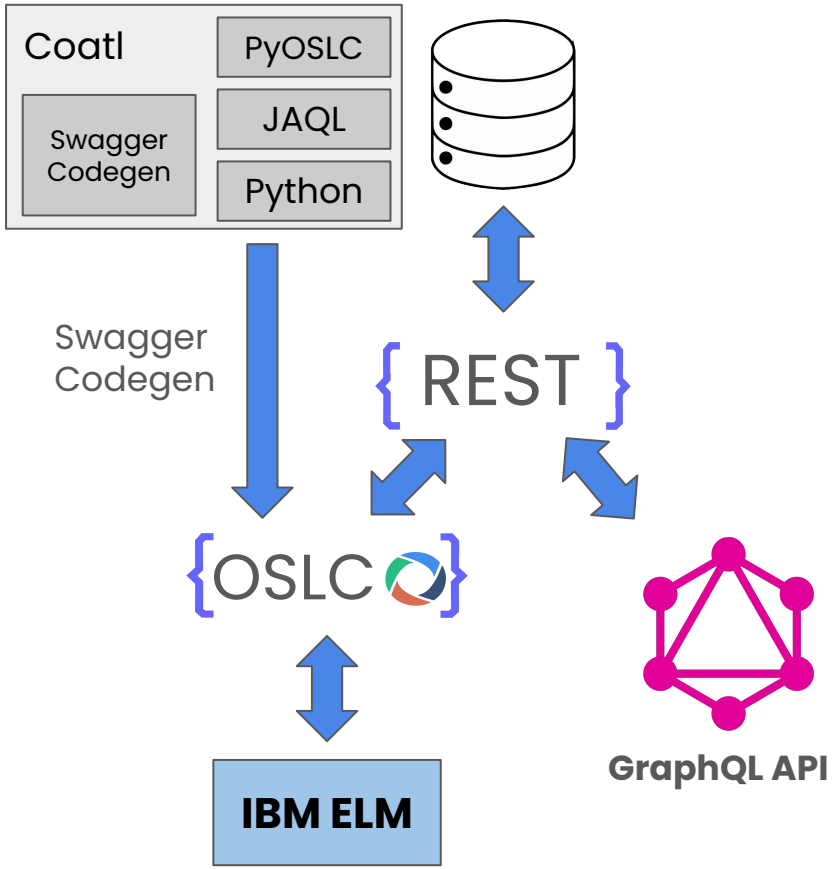
- Enables extensibility through open standards such as OSLC
- Optimize communication, collaboration and verification of artifacts from different vendors
- Improve decision-making by linking artifacts from different vendors to Jazz.



# Coatl & IBM/ELM™

**Support for Oauth v1.0a** for registering external OSLC API as Jazz friend application which enables:

- To associate providers (e.g. Github repo/OSLC SP) to a IBM ELM project
- To use delegated dialogs from external OSLC API within IBM ELM apps
- To define contributing components from external OSLC API to a global component in IBM GCM



# Next steps

# Next steps (OSLC)

- ❏ Improve RDF mapping definition in OpenAPI Spec
- ❏ OSLC Configuration Management compatible with IBM ELM
- ❏ Tracked Resource Set

# Next steps (Code)

- ❏ Deploy new OSLC APIs in cloud / container
- ❏ Include in Coatl library more automated tests
- ❏ Test Coatl library with more REST APIs
- ❏ Include Coatl library support for OpenAPI 3.0

# Conclusion

- **Coatl** → OSLC wrapper for REST APIs
- **Automatic code generation**
- Coatl significantly **reduce the implementation effort** for developers creating OSLC APIs
- Developers **only need to understand OSLC concepts**, they **don’t need to program** them

# Questions about Coatl?
# See a Coatl demo?

# Contact us

**axel.reichwein@koneksys.com**
juan.quintanar@koneksys.com

November 4, 2021