

DigiFoF Training Module:

AI-Based Domain-Specific Assessment Service

Introduction to the use of cloud-based services for assessment

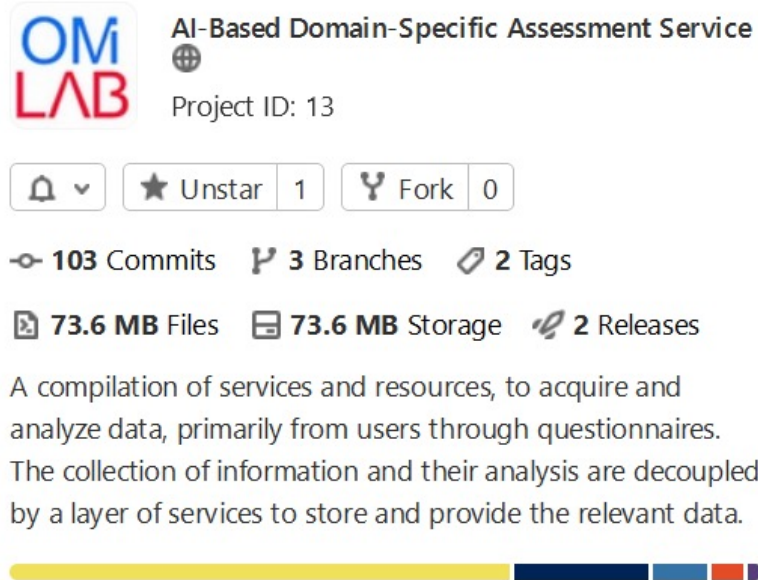


Agenda

- Introduction
- [Approach](#)
- [Technology](#)
- [Demonstration](#)

The Cast

- AI-Based Domain-Specific Assessment Service



OMLAB AI-Based Domain-Specific Assessment Service
Project ID: 13

🔔 Unstar 1 🍴 Fork 0

🔗 103 Commits 🌿 3 Branches 🏷️ 2 Tags

📁 73.6 MB Files 📦 73.6 MB Storage 📄 2 Releases

A compilation of services and resources, to acquire and analyze data, primarily from users through questionnaires. The collection of information and their analysis are decoupled by a layer of services to store and provide the relevant data.

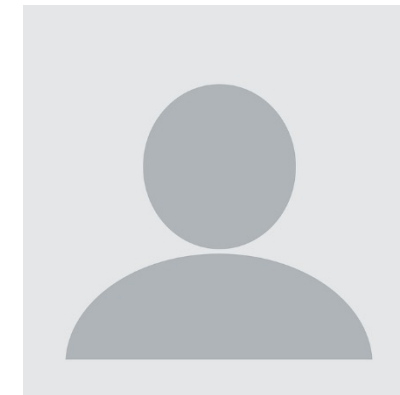
- Wilfrid Utz



- Patrik Burzynski



- You



A Case

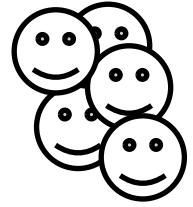
- Domain: Enterprise Architecture
- Your IT architecture is composed of **a lot** of applications, infrastructure elements, services etc.
- A change in your IT architecture is necessary, e.g. due to
 - Changes in licenses
 - A component's end-of-life
 - A new project
 - ...
- To support the decision making you decide to assess parts of the current IT architecture.

Goals

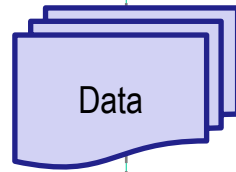
- HOW to realize, but neither what nor why.
- How to use cloud-based services to perform an assessment.
- NOT:
 - Enterprise Architecture Management.
 - Why to use cloud-based services.
 - Who should be asked which questions.
 - What formulas / functions to use for processing data.
 - What is the correct interpretation / decision for a result.

APPROACH

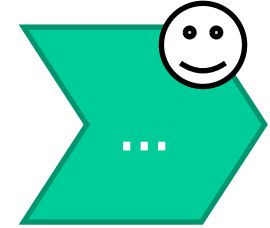
Overview



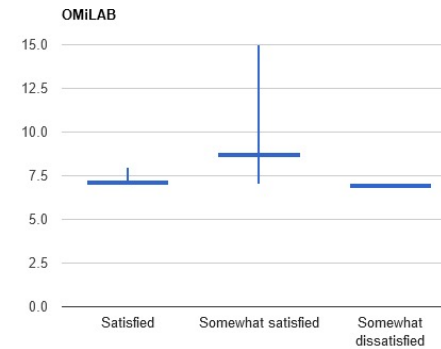
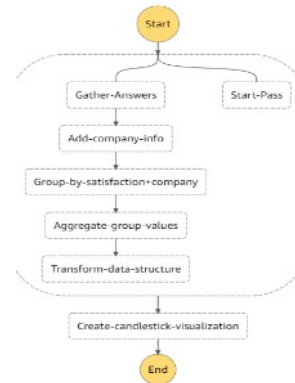
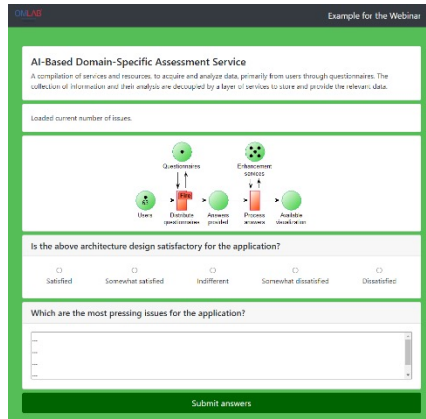
Data Collection



Data Enhancement



Example



Data Collection – Example mockup

Fixed Approach

Dynamic Approach

OMLAB Example for the Webinar

AI-Based Domain-Specific Assessment Service
A compilation of services and resources, to acquire and analyze data, primarily from users through questionnaires. The collection of information and their analysis are decoupled by a layer of services to store and provide the relevant data.

Loaded current number of issues.

```
graph LR; Users((63)) --> Distribute[Distribute questionnaires]; Distribute --> Fire[Fire]; Fire --> Answers[Answers provided]; Answers --> Process[Process answers]; Process --> Visualization[Available visualization]; Enhancements[Enhancement services] --> Process;
```

Is the above architecture design satisfactory for the application?

Satisfied Somewhat satisfied Indifferent Somewhat dissatisfied Dissatisfied

Which are the most pressing issues for the application?

...
...
...
...

Submit answers

Write everything manually in the questionnaire and update the questionnaire when parts change.

Loads the name and description of the project from GitLab at time of showing the questionnaire.

Loads number of issues for the project from GitLab at time of showing the questionnaire.

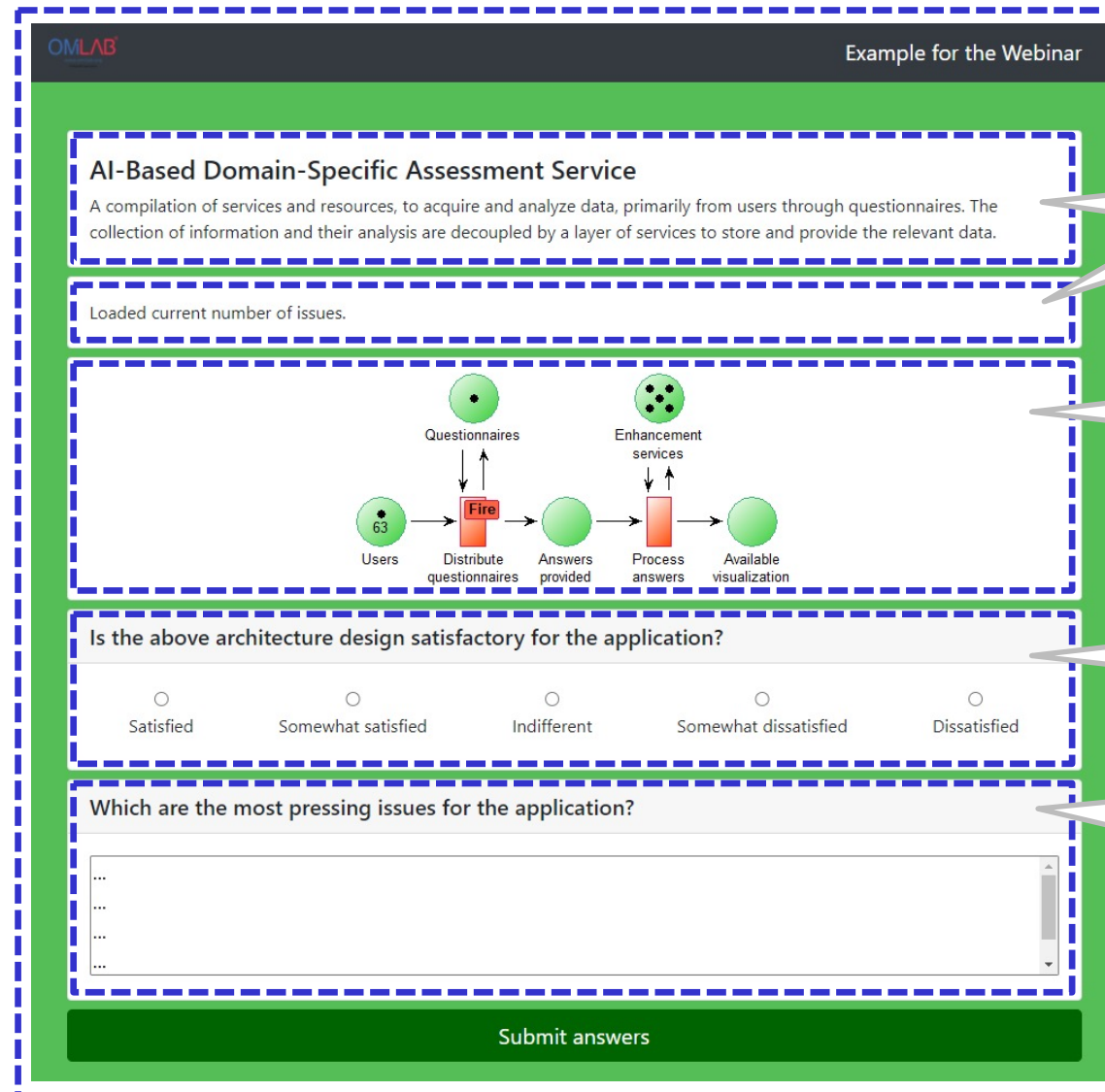
The survey system needs to interact with other systems.

Loads the names of the issues for the project from GitLab at time of showing the questionnaire and populate the selection list.

Data Collection – Dynamic Approach

We can not cover all possible current and future interactions and integrations out of the box.

How can we keep the dynamic approach easily extensible?



Component that assembles the questionnaire.

Component to "load data from GitLab".

Component to "show a specific image".

Component to "ask a Likert-scale type question".

Component to "load data as a list from GitLab and provide a selection".

Not just components, but **cloud-based services** with a simple interface.

Data Enhancement – Example

- **Gather answers**

- Collect answers in an easier to process format

- **Enrich data**

- Use semantic queries to enrich the data with information about their company

- **Group data**

- Create groups of data based on the selected satisfaction and company

- **Aggregate values**

- Aggregate the values according to their groups

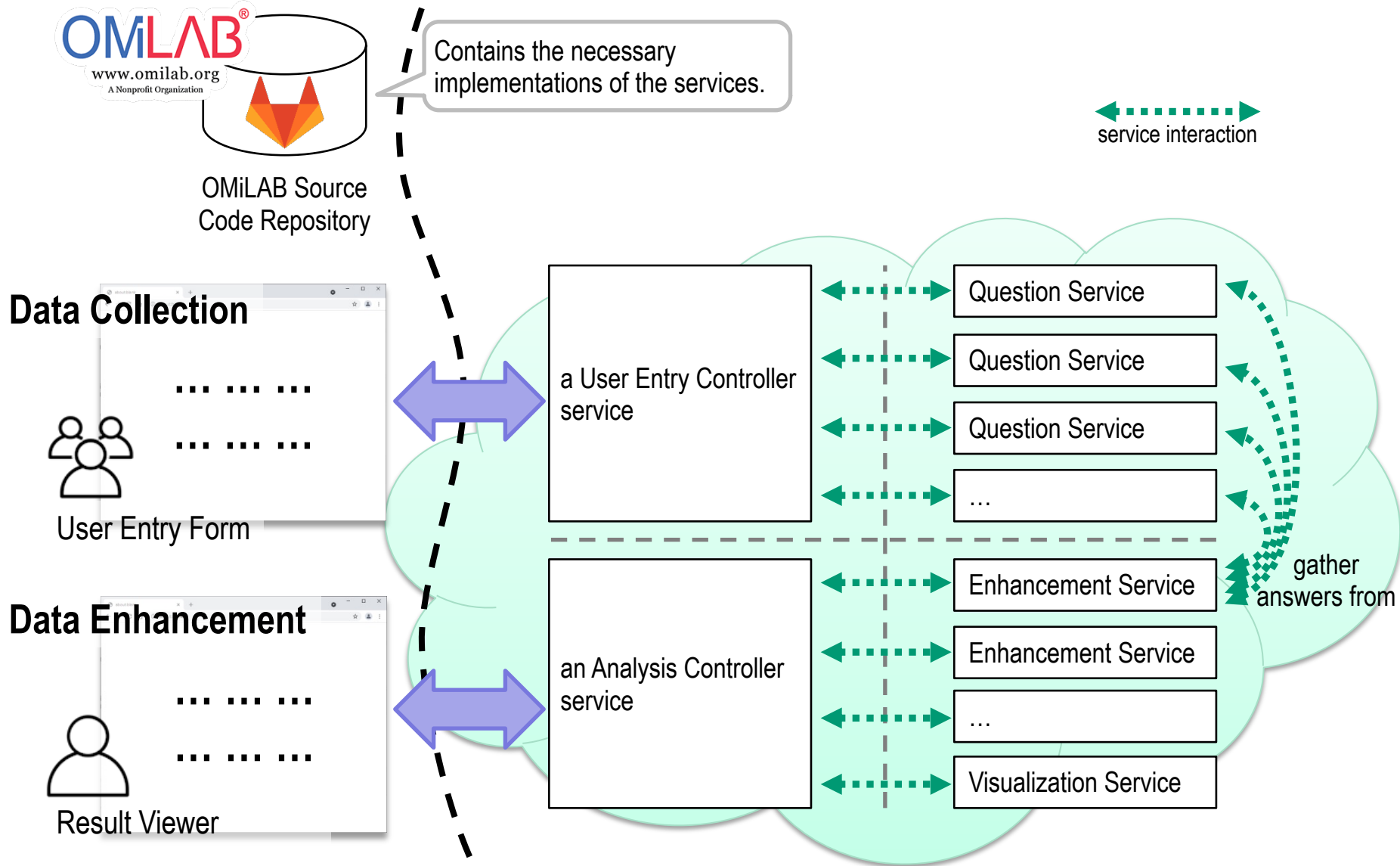
- **Visualize results**

- Create a visualization of the result using a candle-stick diagram

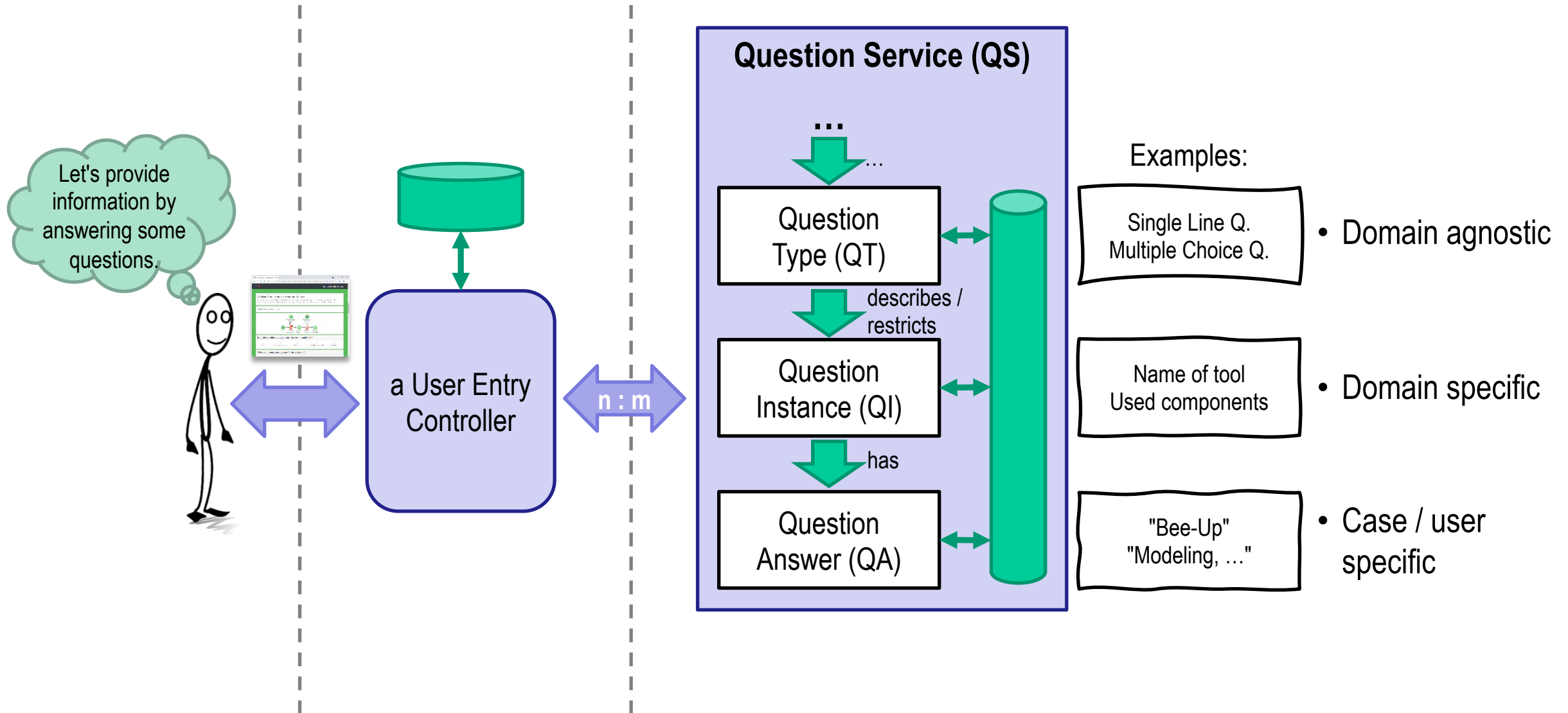
Similar to the Data Collection: how can we keep this **easily extensible**?

Partitioning the process into individual steps that are implemented using **cloud-based services**.

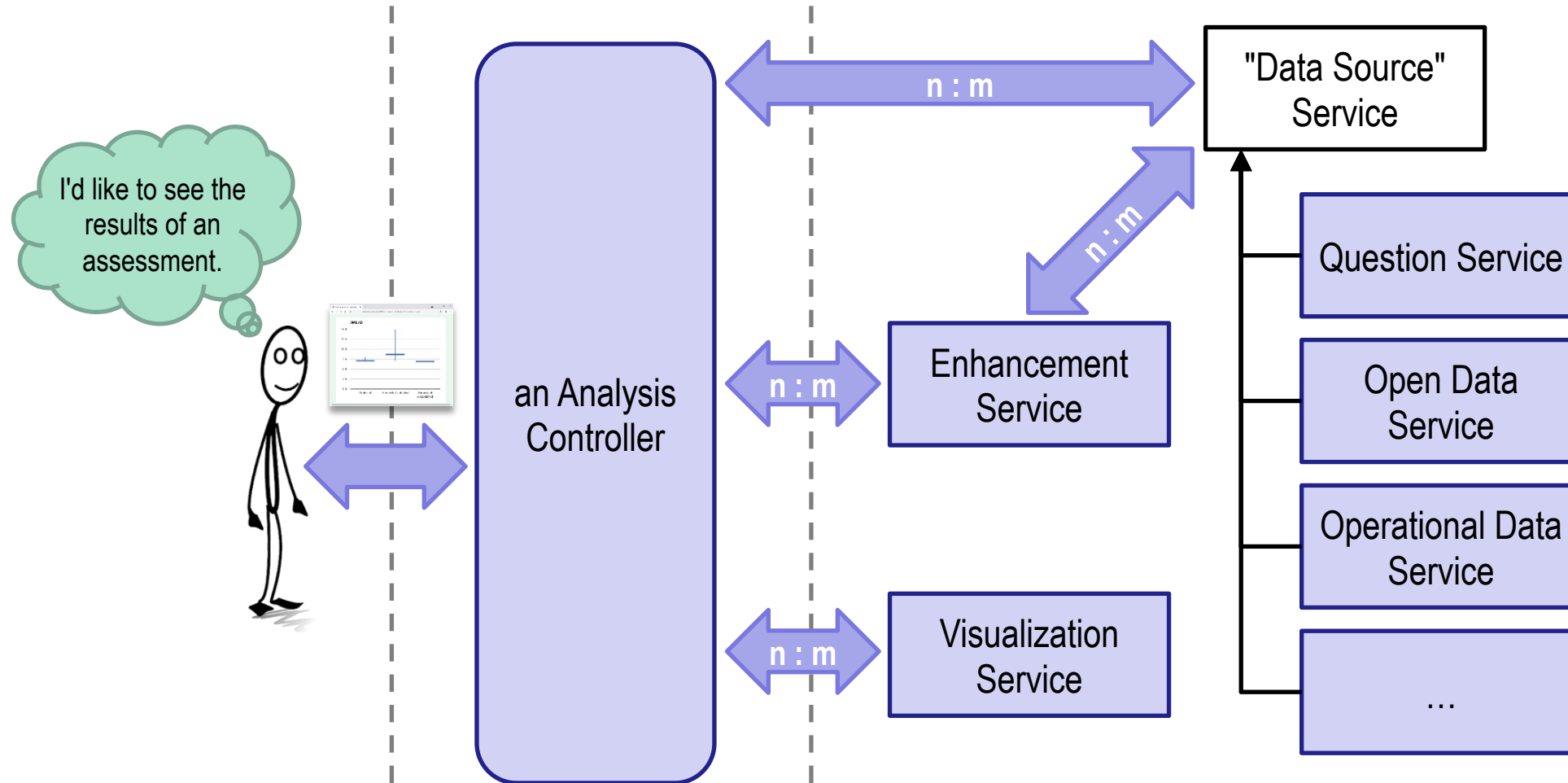
Conceptual architecture



Data Collection – Concepts



Data Enhancement – Concepts

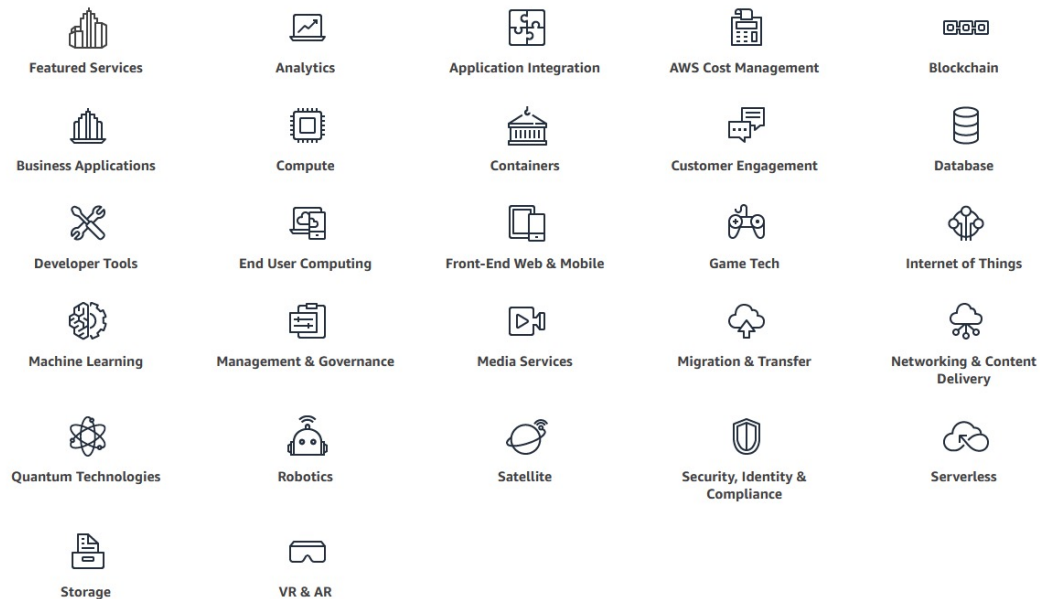


TECHNOLOGY

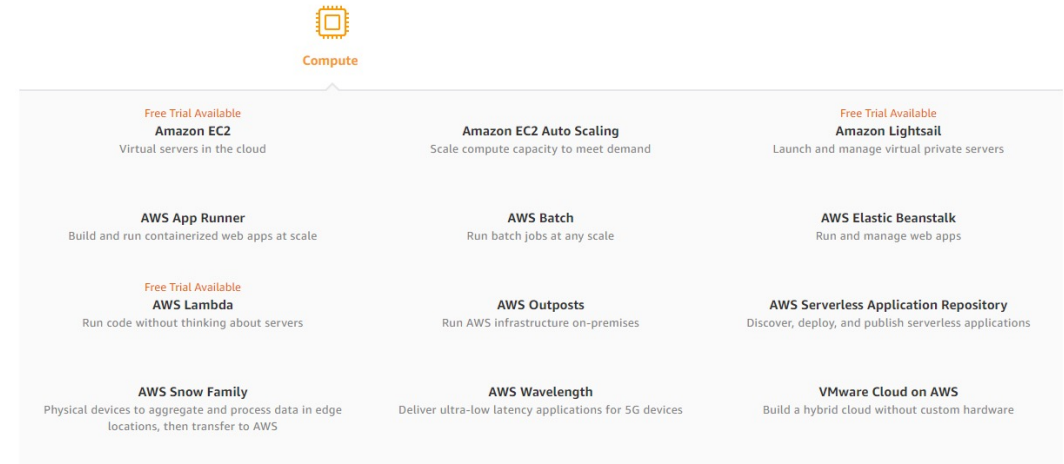
Filling the gaps – Amazon AWS cloud computing

- "Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 200 fully featured services from data centers globally."
[Amazon]

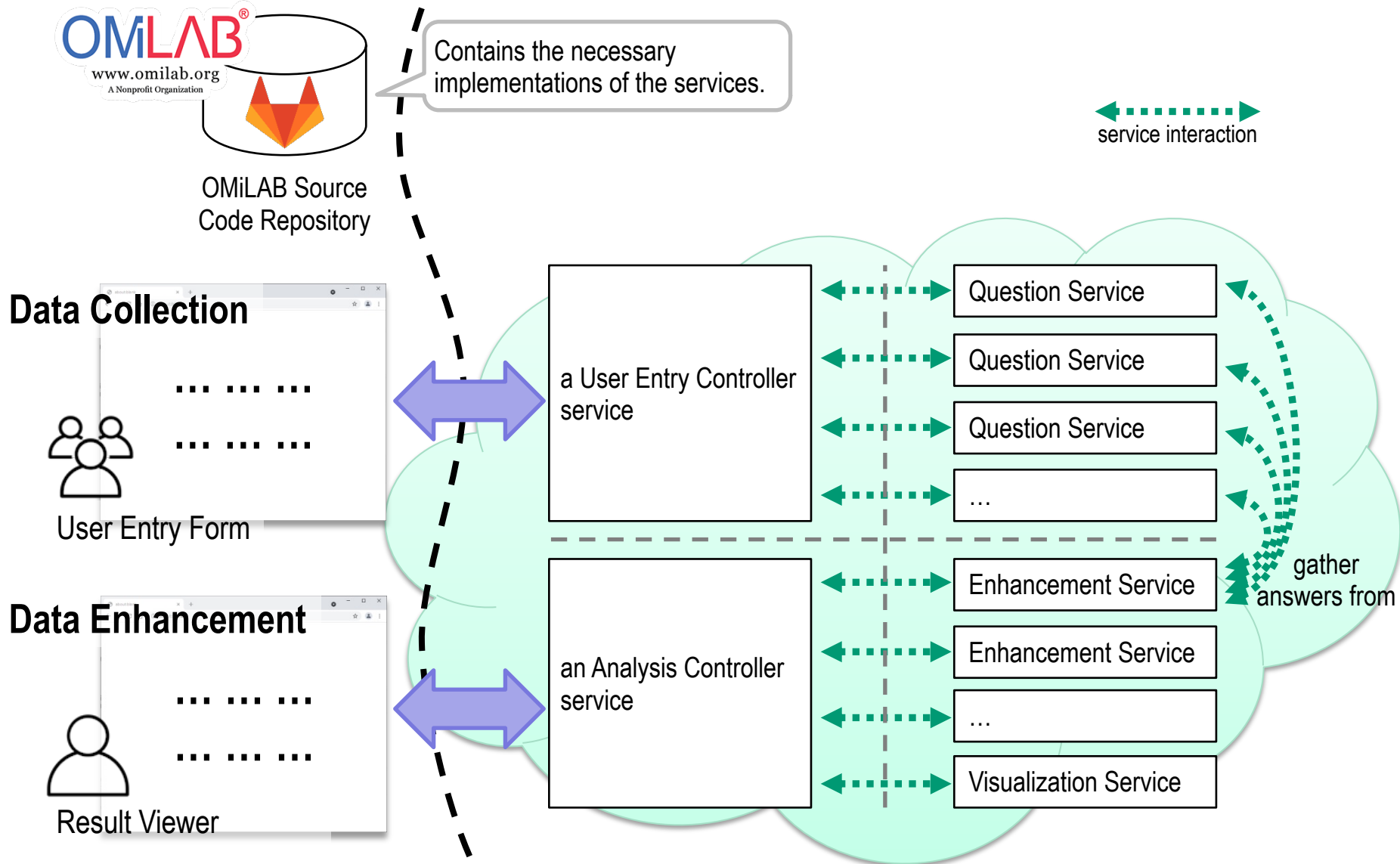
AWS categories



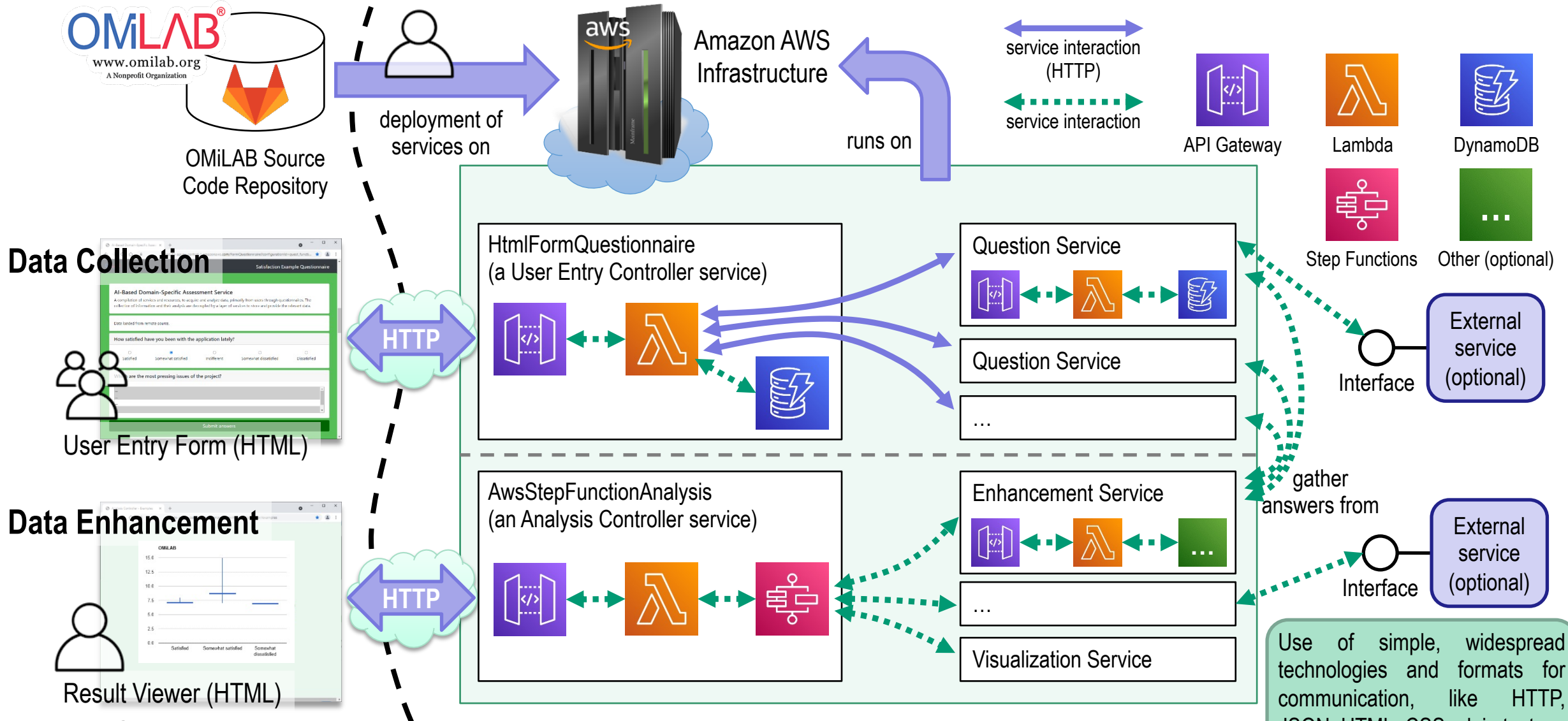
Entries in the "Compute" category



Conceptual architecture

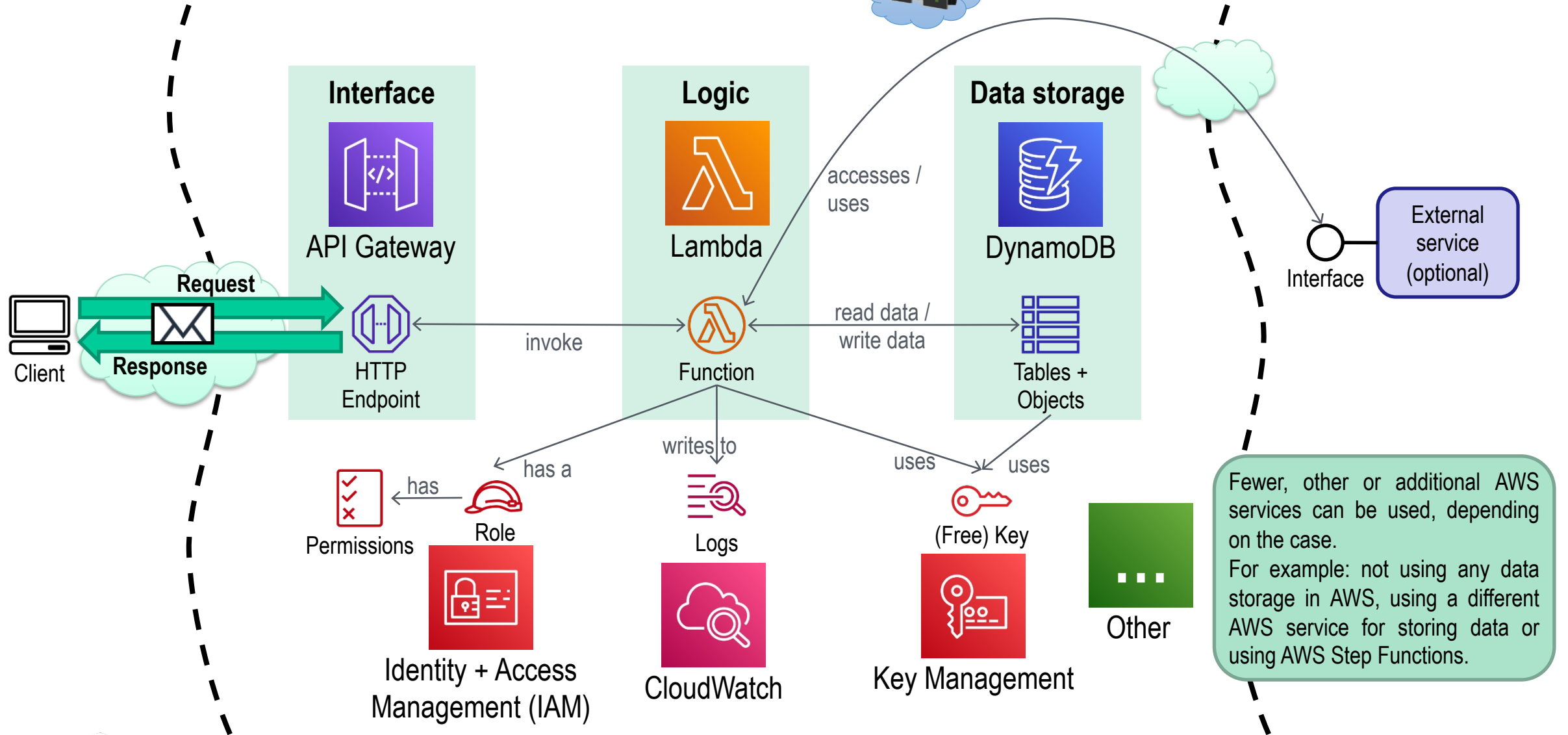


Deployment architecture – Detailed



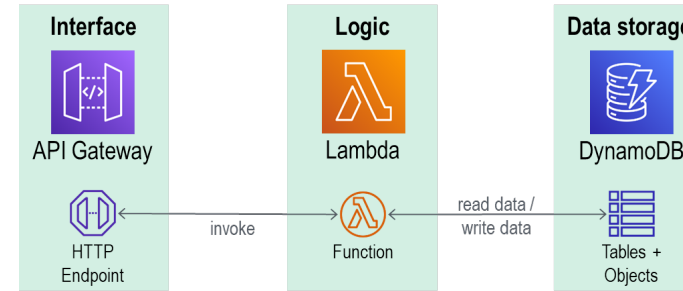
Use of simple, widespread technologies and formats for communication, like HTTP, JSON, HTML, CSS, plain text ...

General service deployment architecture using



Main technologies, tools and concepts used

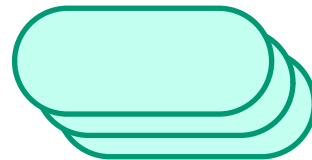
- For execution:
 - AWS specific (API Gateway) – for interface
 - JavaScript (Node.js) / Python – for program logic
 - AWS specific (DynamoDB) – for data storage
 - RESTful – avoid state-based communication



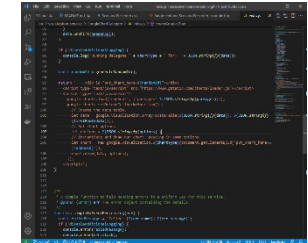
- For interaction with users
 - HTTP
 - HTML + CSS + JavaScript



- For interaction with services
 - HTTP
 - JSON / plain text



- For development
 - Git
 - Visual Studio Code



- For deployment
 - PowerShell
 - AWS CLI



DEMONSTRATION

Demonstration – Assumptions

- ✓ The needed accounts are created.
 - ✓ The necessary tools are installed and configured.
 - ✓ The relevant services are deployed and available.
 - ✓ The people are ready.
-
- We have to create the questionnaire.
 - We have to fill out the questionnaire a few times to have data to process.
 - We have to process the data.
 - We have to admire the result.

Example – Data Collection

The questionnaire captures two parts of data:

- The current amount of issues
- The user's satisfaction

User Entry
Controller Service:
HtmlFormQuestionnaire

=

This example can be deployed using the files found in the "examples/Questionnaire - Create" folder. The questionnaire structure is described in the file "Issues+Satisfaction2.json"

Questionnaire

The screenshot shows a questionnaire interface with the following sections:

- Header:** OMLAB logo and "Example for the Webinar".
- Service Description:** "AI-Based Domain-Specific Assessment Service" with a sub-description: "A compilation of services and resources, to acquire and analyze data, primarily from users through questionnaires. The collection of information and their analysis are decoupled by a layer of services to store and provide the relevant data."
- Status:** "Loaded current number of issues."
- Process Diagram:** A flowchart showing "Users" (63) leading to "Distribute questionnaires" (Fire icon), then "Answers provided", then "Process answers" (Enhancement services icon), and finally "Available visualization".
- Satisfaction Question:** "Is the above architecture design satisfactory for the application?" with radio buttons for "Satisfied", "Somewhat satisfied", "Indifferent", "Somewhat dissatisfied", and "Dissatisfied".
- Open-Ended Question:** "Which are the most pressing issues for the application?" with a text input area.
- Submit Button:** "Submit answers" at the bottom.

Question Services

Question Service:
FetchHttpJsonData

Question Service:
StaticImage

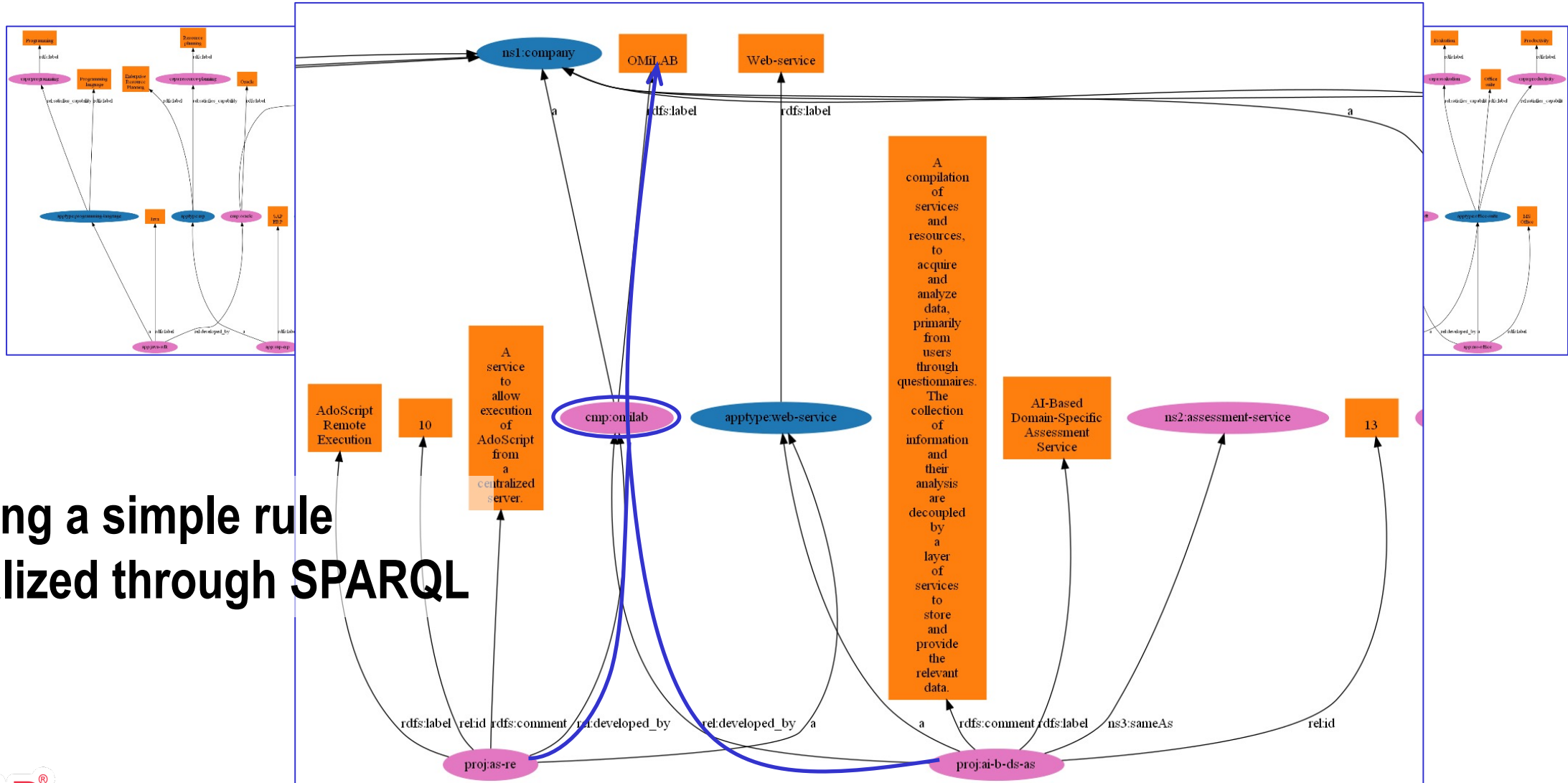
Question Service:
LikertScale

This service does not yet exist, BUT could be implemented by the community.

Example – Data Enhancement steps

1. Gather the data from the filled-out questionnaires in an object-like structure:
 - ID of the application / project
 - Issue count
 - User's satisfaction
2. Add information about the company developing the project.
3. Split data into groups based on:
 - User's satisfaction
 - Company
4. Calculate the min, max and average for each group.
5. Transform data structure to fit visualization:
 - [Satisfaction, min, avg, avg, max]
6. Create the candle-stick visualization.

Adding company information from an RDF graph



using a simple rule realized through SPARQL

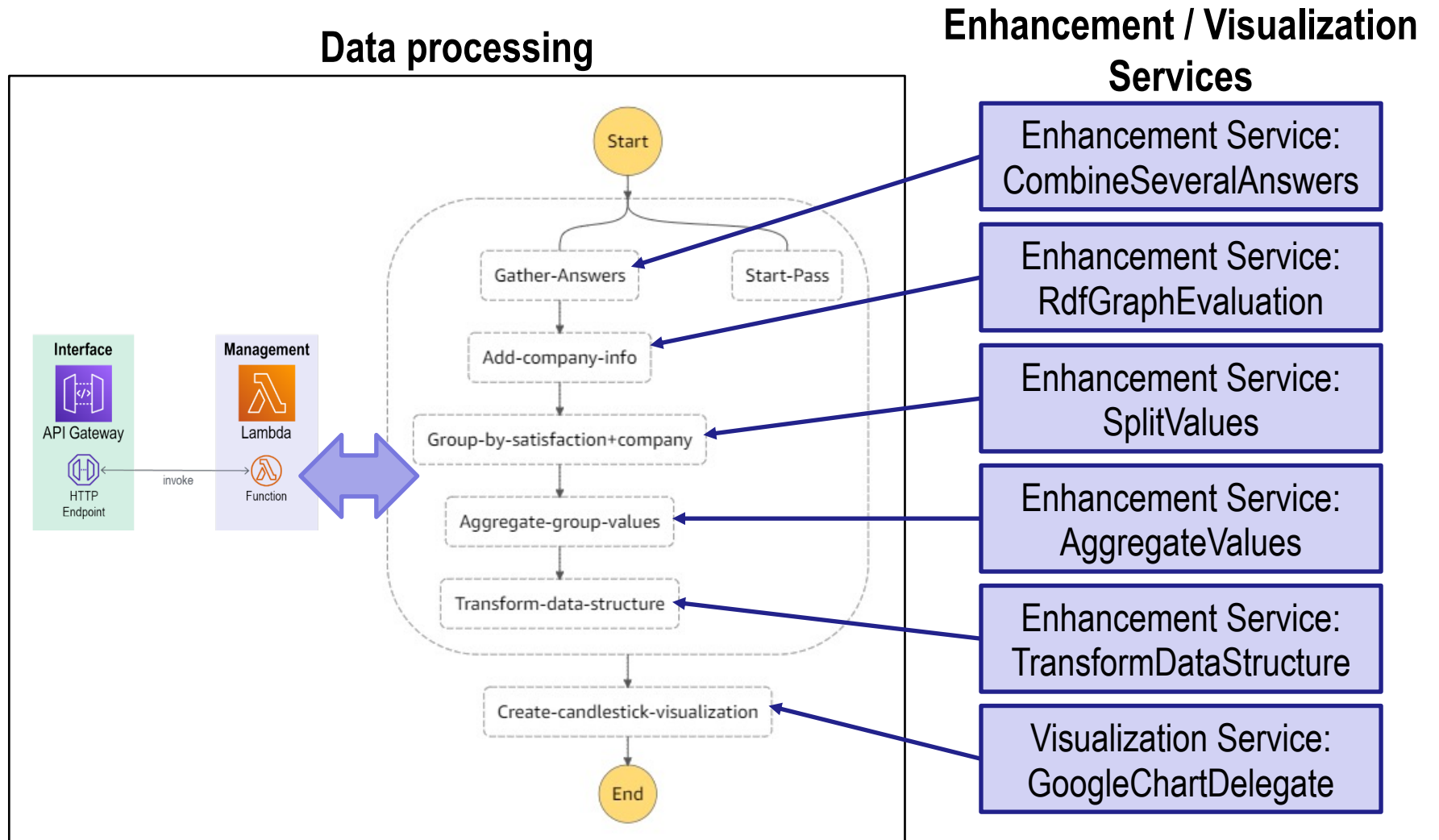
Example – Data Enhancement

Allows HTTP access and management of AWS State Machine executions.

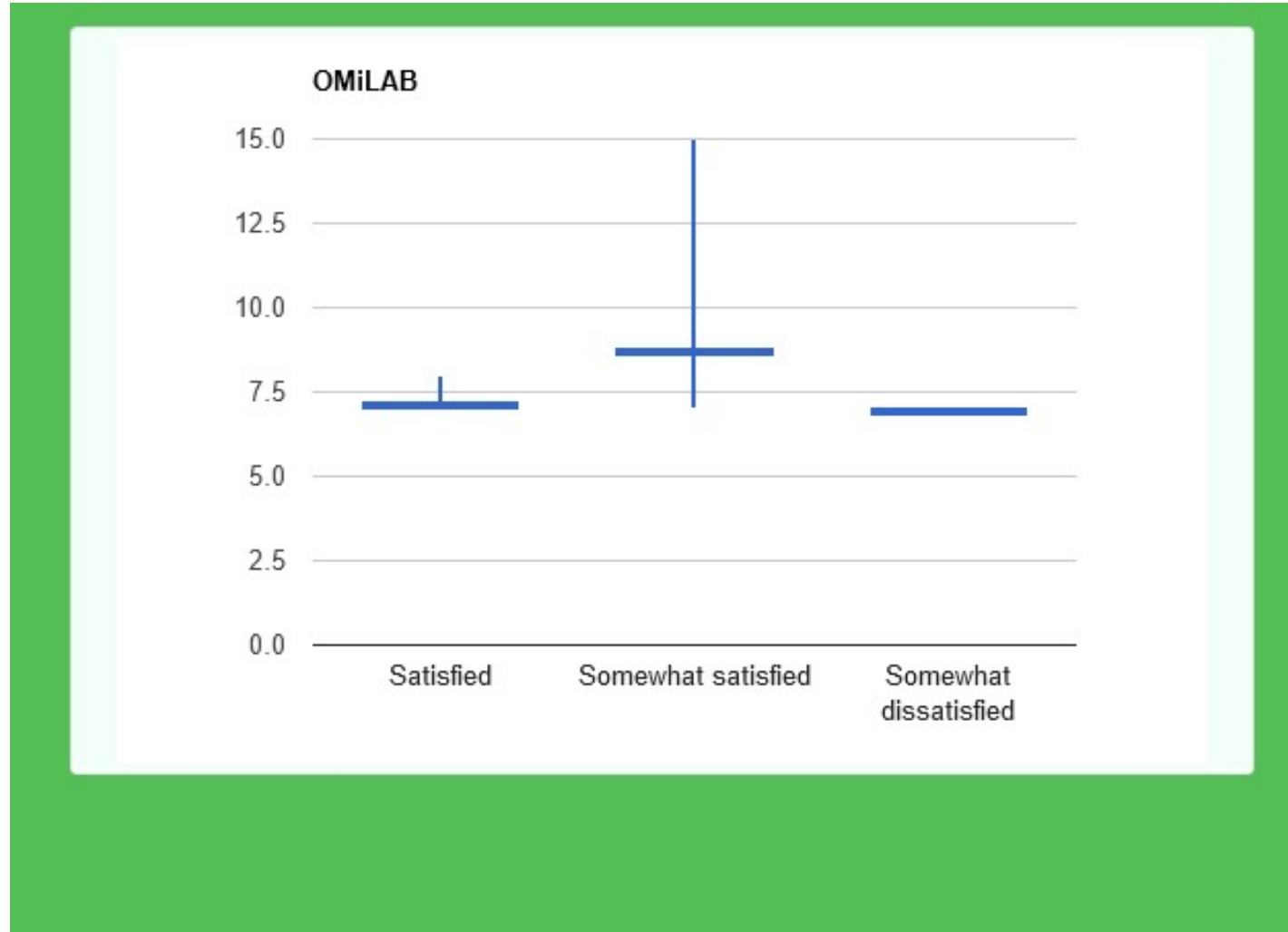
Analysis Controller Service:
AwsStepFunctionAnalysis

The AWS State Machine code for this example can be found in the "examples/AwsStepFunctionAnalysis" folder in the file "Issues+Satisfaction-Company-StateMachine.json". The values for "ApiEndpoint" must be added before it can be used.

=



Example – Result



AI-Based Domain-Specific Assessment Service

- Project page at <https://www.omilab.org/assessmentservice/>



- Code repository at <https://code.omilab.org/services/assessment-service>

