Proposed Architecture

The architecture that is suitable for X5GON going forward is a system that can work with a collection of Microservices that can be orchestrated on-demand in different configurations. On the surface, this architecture would look like a set of tiny independent services enriching the document metadata record in a pre-defined order gradually adding more enrichments regarding a learning material to the datastore. The services and the workflow that is triggered can be represented using a graph as shown in figure 1.

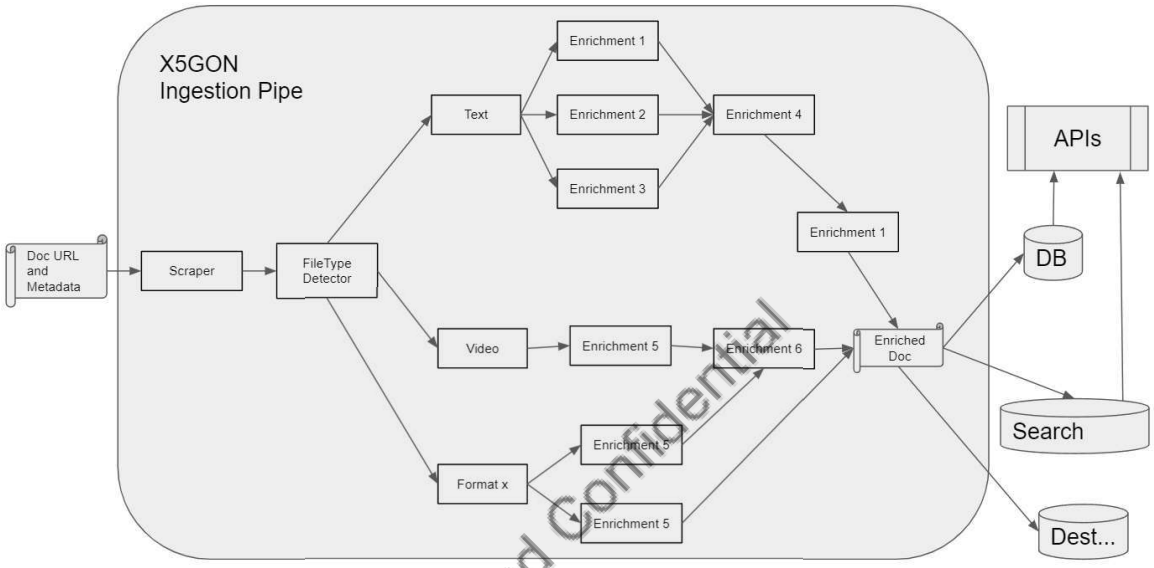

Figure 1: An example realisation of the X5GON pipeline with multiple file types going through various enrichment steps in a pre-defined order and finally ending up in different data sources as destinations (DB, Search etc…)

The main feature needed going forward for X5GON is to be able to dynamically insert and remove blocks (rectangles in Figure 1) and reroute how documents will run through enrichments with minimal developer intervention (Low code or No Code). This can be achieved by:

- Keeping the code pattern of all the different blocks the same (less steep learning curve for contributors when authoring enrichment blocks)
- To have a standardised way for the blocks to talk to each other.
- To have a overarching process that tracks the routing and progress of individual documents through the enrichment graph (Business Process/Workflow Management)

This can be achieved by having a system that has a central runner and worker blocks that builds the “Orchestrator Framework” (https://medium.com/trueengineering/a-review-of-microservice-orchestration-frameworks-d22797b34ea5). The setup of the Orchestrator framework is shown in figure 2.

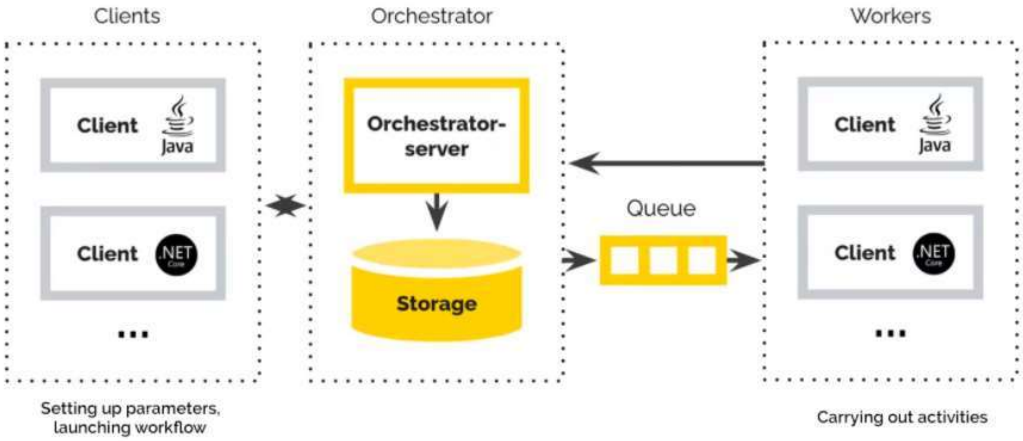

Figure 2: The setup of "Orchestrator Framework" for microservices. This framework uses a client to define how the microservices should be orchestrated. The orchestration plan is passed to the "orchestrator" which is the central server that makes sure the relevant data record is passed through the relevant service (client in the figure) correctly based on the plan defined through the client. The workers are the microservices that do the enrichment process that are independent of each other and act upon the calls made by the orchestrator.

This framework consists of:

- **Worker**: An army of workers that contain individual enrichment blocks (microservices) that takes a set of data and parameters as input and outputs the enriched version. They are independent of each other and only get triggered by the instructions from the orchestrator.
- **Orchestrator**: The main runner that is aware of the Directed Acyclic Graph (DAG) that will pass the record (learning material records in X5GON) from one enrichment service to another based on the current state of that record (As per Figure 1). The runner prescribes instructions to workers.
- **A client**: An interface that allow the administrators(human) to communicate to the orchestrator on how to change its behavior with the DAG
    - Change the overall structure of the DAG
    - Set Input, output parameters of each worker block
    - Set parameters such as number of retries, delays etc.

In the new system, wrapping all enrichment/scraping blocks as http-based microservices (APIs in service oriented architecture) that will create the workers. This will allow us to:

- Have a standard interface to communicate with them.
- The actual enrichment logic can depend on various libraries, programming languages etc. and has no impact on interoperability with X5GON
- Developers will have a well known standard to stick to
- Can exploit other toolings such as automatic API documentation to make the blocks more accessible to the contributors
- Fairly straightforward integration testing

Deployment Decisions

When determining what technologies to use in implementing this orchestrator system, it is important to be:

- Less reliant on cloud vendor specific technologies
- Use the least number of unique technologies within the core system to help maintainability and reduce tech debt
    - This allows contributors to adapt a smaller number of stable standards in development practice
    - Testing, integration and release becomes simpler

Proposed Architecture

The proposed architecture should use containers to run different workers and also the orchestrator and clients.

Advantages

- Can be deployed in any cloud vender or on-premise (Amazon ECS/EKS, Google CloudRun, Azure Container Service or on-premise Docker)
- The environment can be code rather than human actions or OS specific shell scripts
- Can be tested easily
- Any contributor can use the tools they need as long as they are accessible via Docker orchestration (any programming language)
- Can be deployed and managed separately in different infrastructure and scaled independently based on demands to different applications

Disadvantages:

- Extra effort having to maintain docker images and have them stored in a registry
- Using the network to communicate with the services adds some latency.
- Computational Overhead in running a container for a extremely simple process
    - Which is rare in X5GON