

EMIKG

Next Steps



Some software engineering vocabulary

Let's start with a few software engineering terms:

- Scripts / Monolith / Modular
- Interfaces / APIs
- Containers and Swarms
- Unit Tests & Integration tests
- Fuzzing
- Rust

Scripts - Scripts everywhere

— — —

Challenge: Over Reliance on scripts.

Drawbacks:

- Limited scalability
- Maintenance difficulties

Needs to evolve to structured software solution



Modular app

— — —

Modular applications are loosely integrated

Lots of moving parts and dependencies - a lot can go wrong and it inevitably does

Python is the duck-taping language of choice

X (the app formerly known as Twitter) is a modular mess



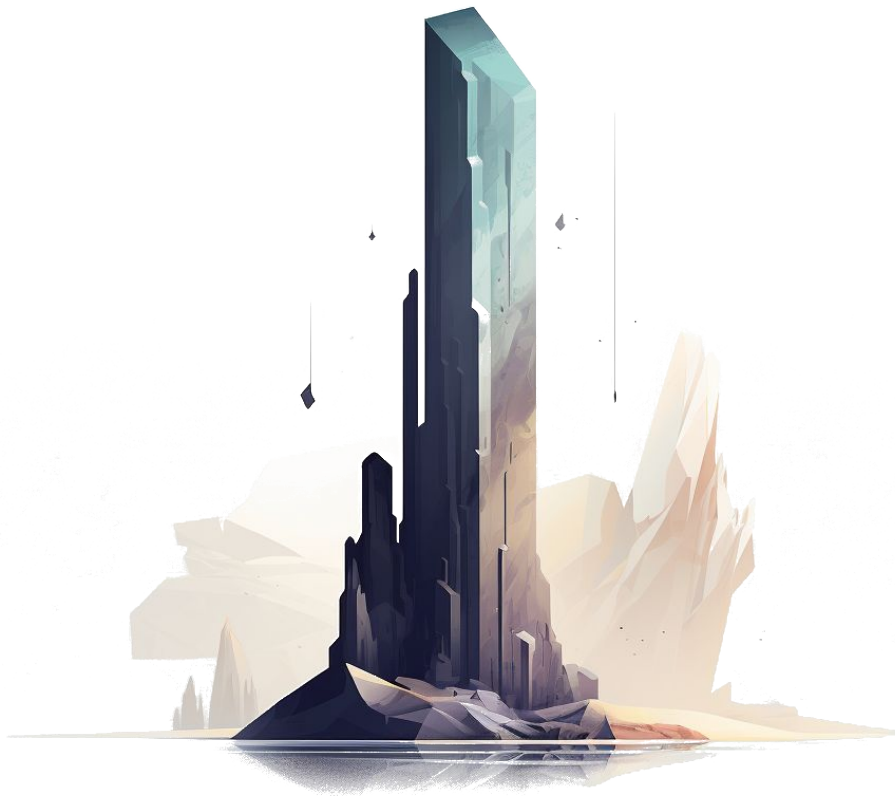
Monolith

A **Monolith** is a tightly integrated application

More efficient than a modular application - compiled languages like Rust can be used effectively

Strict control on all moving parts: dependencies kept to the bare minimum

***TikTok** is a monolith*



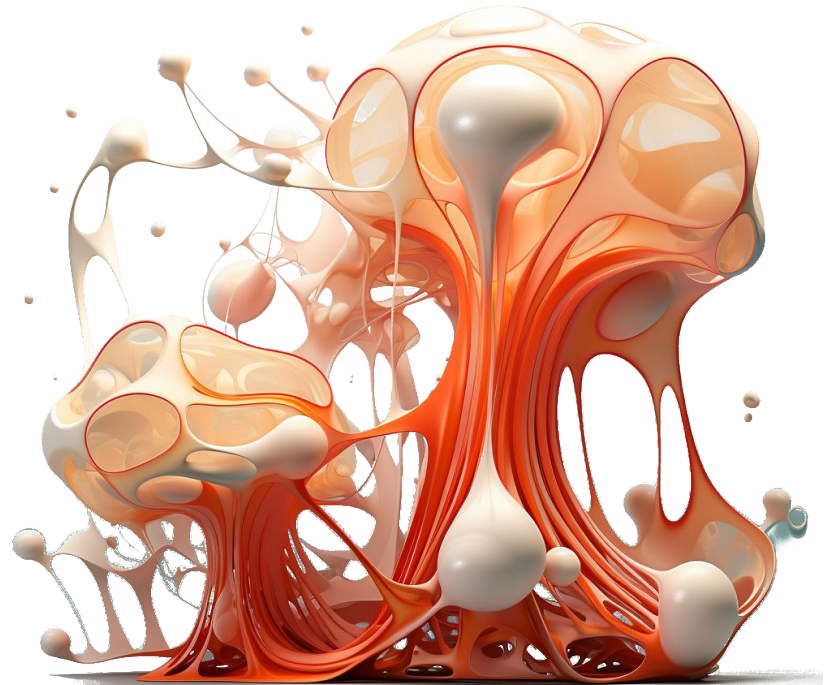
Interfaces

— — —

An abstract, contract-based programming construct

Specifies required methods and properties

Enforces a contract & promotes code reusability



APIs (Application Programming Interfaces)

— — —

- Interconnection Channels
- Allow software components to communicate
- Define methods for data exchange
- Enable integration of different systems



Containers & Swarms

Container:

- Self-contained software package
- Includes code, runtime, and libraries

Swarm of Containers:

- Managed group of containers
- Orchestrated for scalability and resilience



Unit & Integration Tests

— — —

Unit Test:

- **Testing at Micro Level**
- Evaluates individual code components
- Ensures correctness of isolated functions or methods

Integration Test:

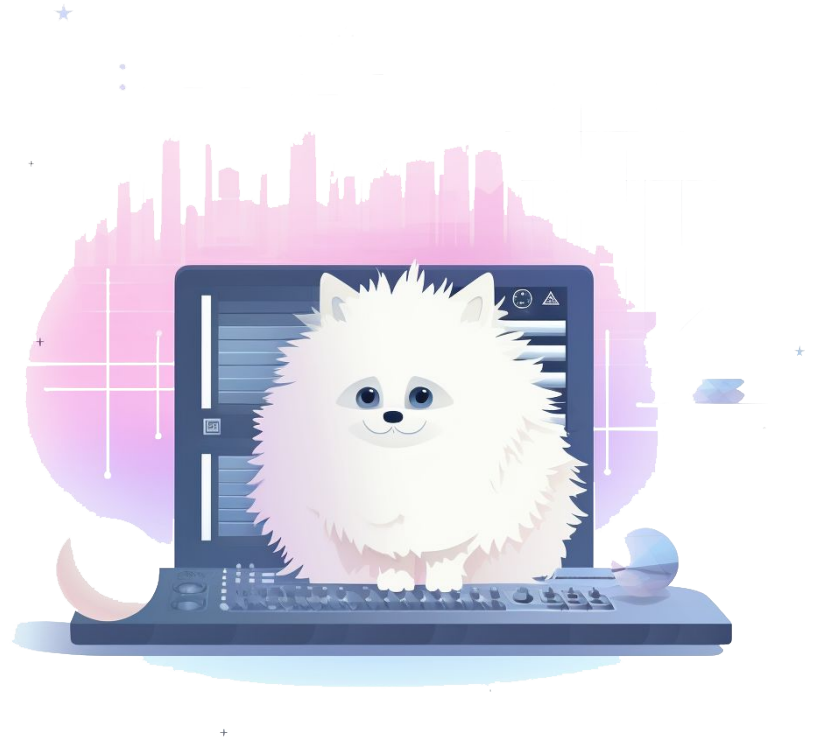
- **Testing at Macro Level**
- Assesses interactions between components
- Validates data flow and collaboration across multiple units or modules



Fuzzing

— — —

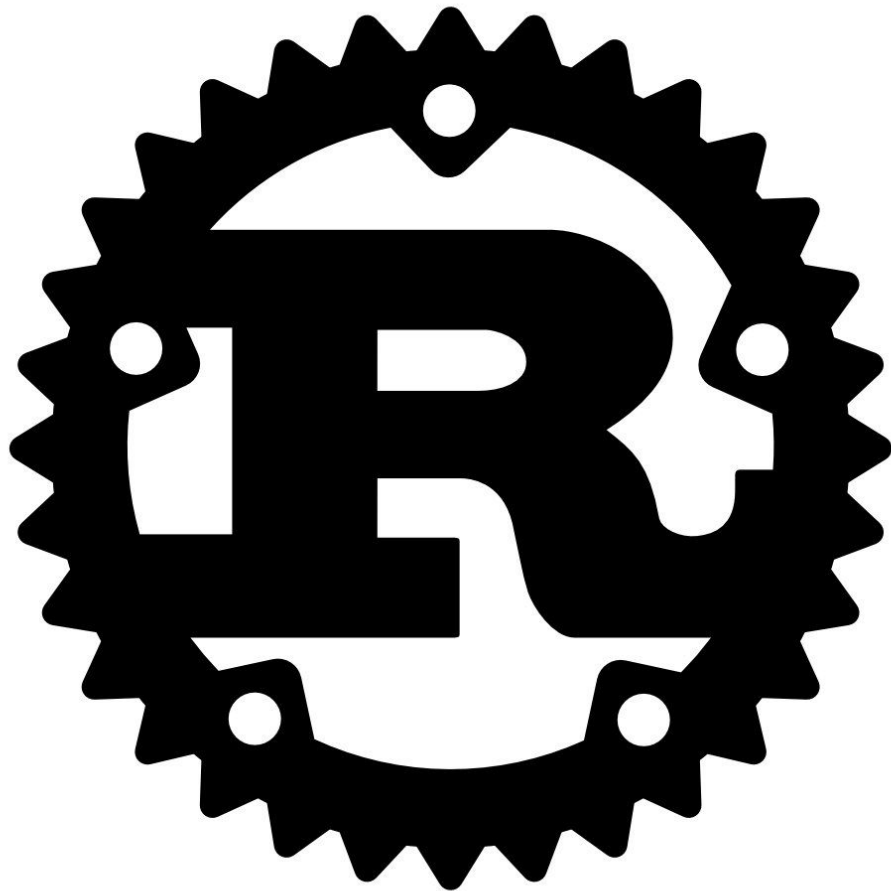
- Automated Testing Technique
- Feeds random or unexpected data inputs
- Identifies vulnerabilities and software weaknesses hopefully before others do



Rust language

— — —

- Compiled Programming Language
- Designed for safety and performance
- Emphasizes memory safety and prevents common programming errors
- Known for its strong type system and a focus on concurrency



Next steps

Updates necessary

- Update (create) cluster design
- Refactor all EMIKG-related app into cohesive modules
- Introduce unit and integration testing across applicatives
- Start to deliver regular KG versions
- Start to run Graph ML on the KG
- Start to deliver regular Graph ML reports

Proposed modules

Current modules

— — —

Website:

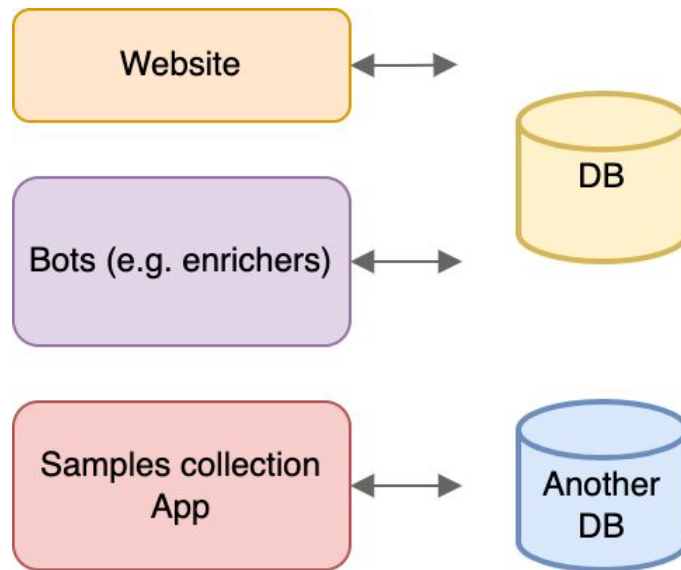
- UI for data upload
- UI for task process

Bots:

- Query and enrich data automatically

Samples app (Eduard):

- Allows easy samples metadata collection



Proposed modules

Website:

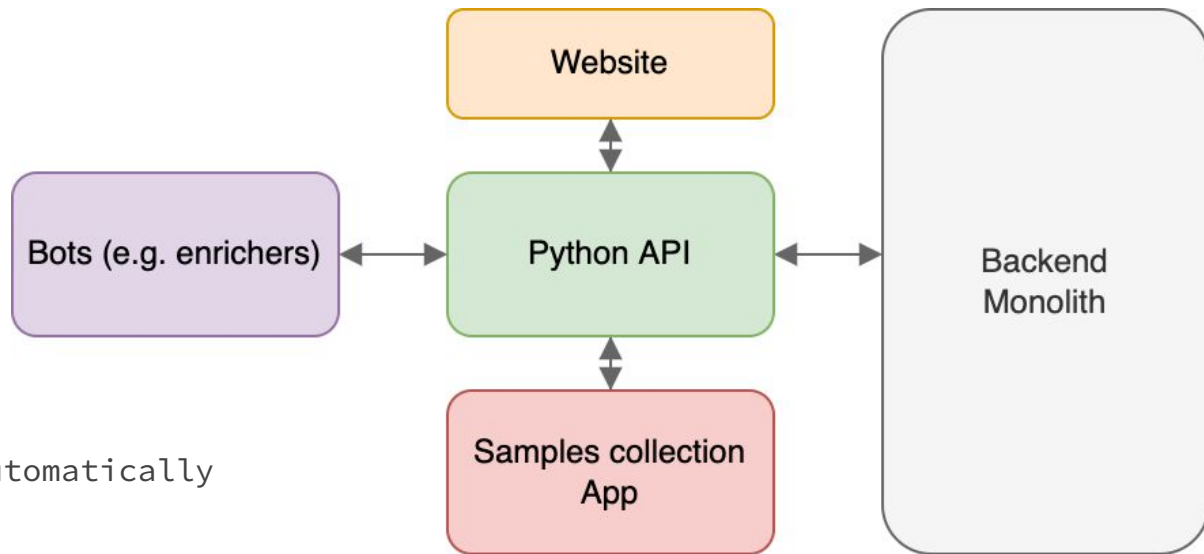
- UI for data upload
- UI for task process
- **UI for KG exploration**
- **Grants tokens for APIs**
- **Data downloads**

Bots:

- Query and enrich data automatically
- **Run ML tasks**

Samples app (Eduard):

- Allows easy samples metadata collection



Main changes

1. APIs are database agnostic
2. DBs be handled internally using best suited data structure for the different operations
3. Internal monolith for efficient data storage & processing
4. External modularity for elastic growth of features

ML Operations

— — —

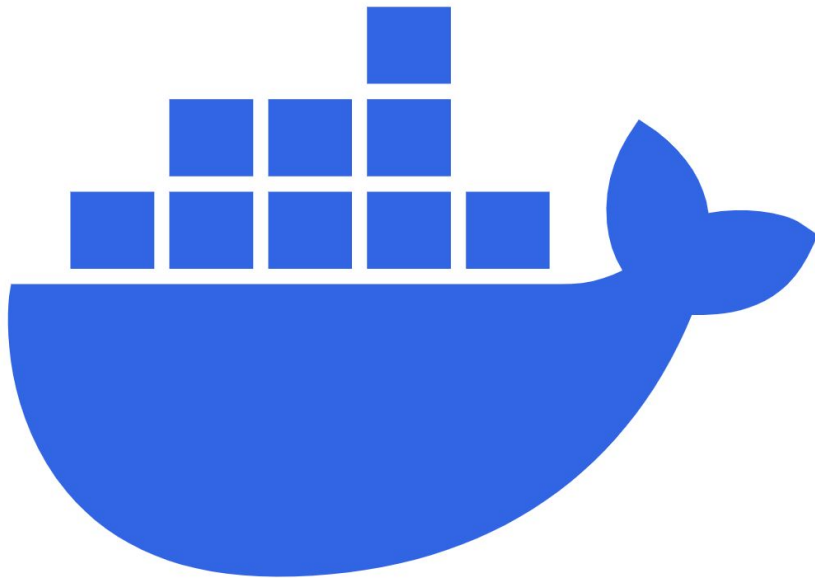
- Provide rich general-purpose KG embedding
 - Includes embedding of nodes, edges (on-demand) and their characteristics
- Spectra embedding
- Feature prediction for spectra
- Link predictions between subsets of interest

Tools

Docker

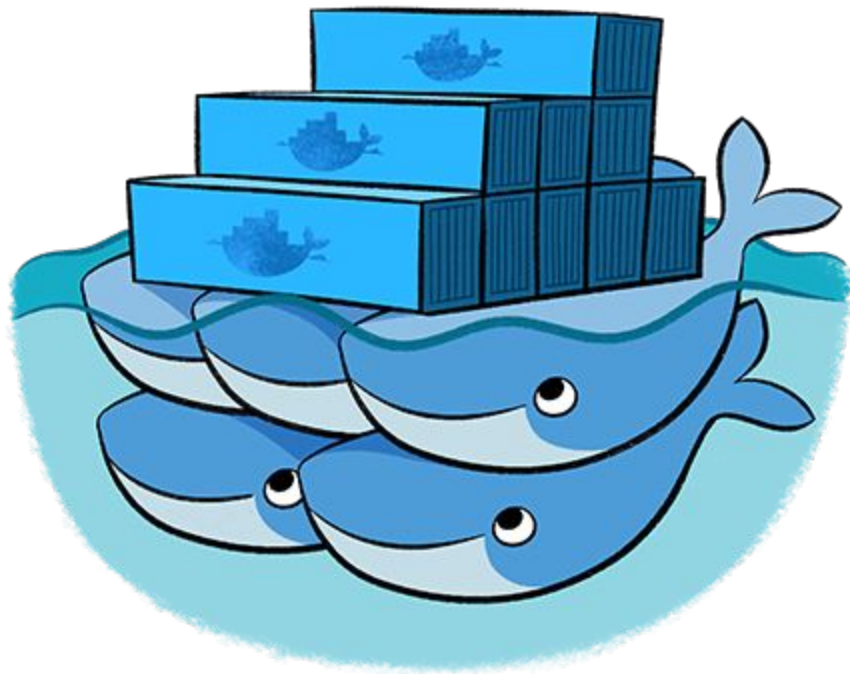
— — —

- Containerization Platform
- Isolates and runs applications
- Portable & relatively safe



Docker Swarm

- Container Orchestration Tool
- Manages clusters of Docker containers
- Enables scaling and load balancing



Jail shell

— — —

- Isolated Environment
- Restricts user/process access
- Enhances system security
- Limits unauthorized actions



Considered: Kubernetes

- Container Orchestration Platform
- Manages and automates the deployment, scaling, and operation of containerized applications
- Provides **container cluster management** and ensures application availability and scalability
- ***MUCH MORE COMPLEX THAN DOCKER!***



Pytests

— — —

- Python Testing Framework
- Open-source testing tool
- Simplifies test creation and automation
- Widely used for unit testing and functional testing in Python applications



pytest

LibFuzzer (cargo fuzz)

— — —

- Fuzz Testing Library
- Part of LLVM Compiler Infrastructure
- Focuses on coverage-guided fuzzing
- Used to discover software vulnerabilities by generating and testing random inputs
- Promotes automated and effective security testing



Cargo test

— — —

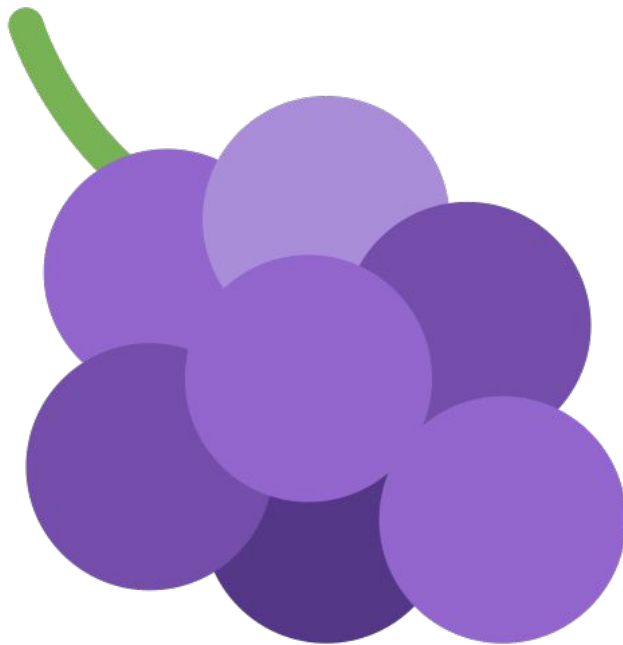
- Rust Testing Command
- Part of the Rust programming language's build tool, Cargo
- Executes unit tests and integration tests defined in a Rust project
- Aids in verifying the correctness and reliability of Rust code
- An essential tool for maintaining code quality and identifying issues during development

GRAPE

— — —

A graph processing and machine learning library

We will be using it for running some of the graph analysis and graph ML tasks



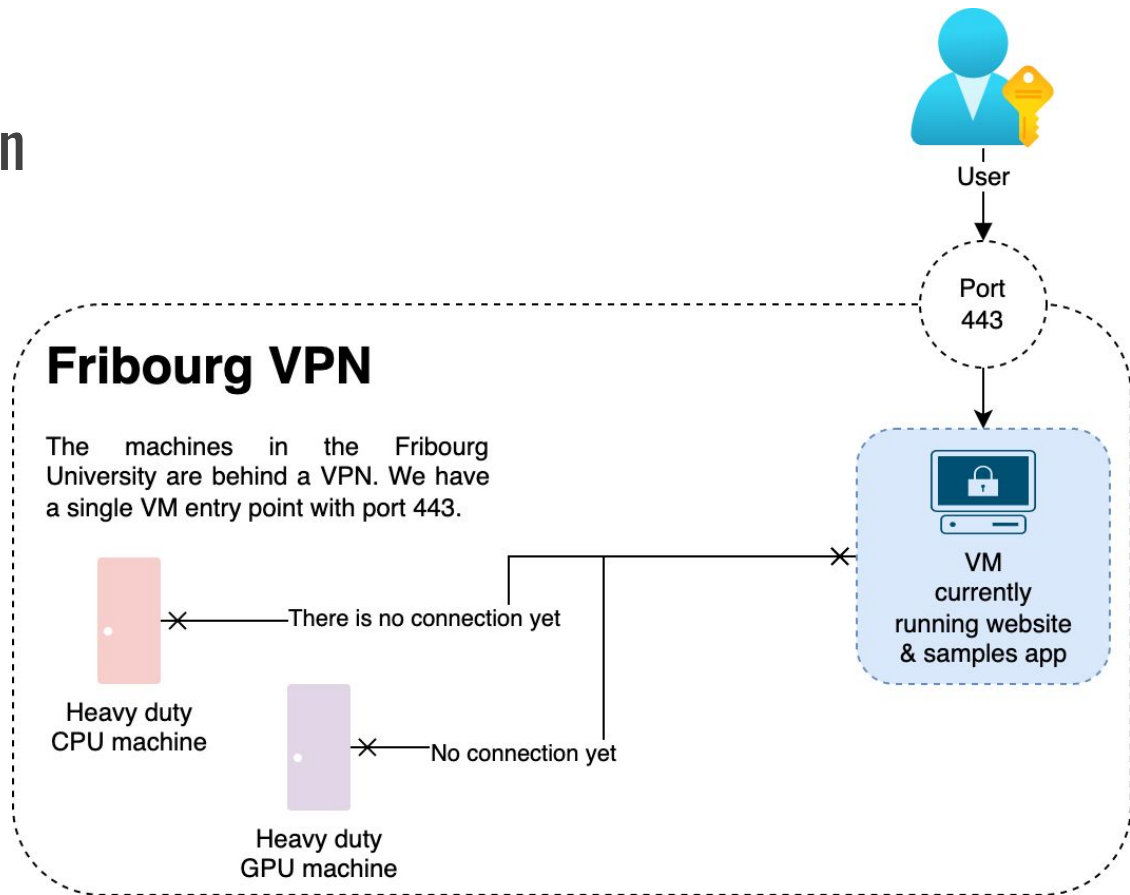
Cluster architecture

Current cluster configuration

In the current cluster, we have a single access point from the internet

Currently, all of our services run on this small VM

None of the other hardware is currently in use



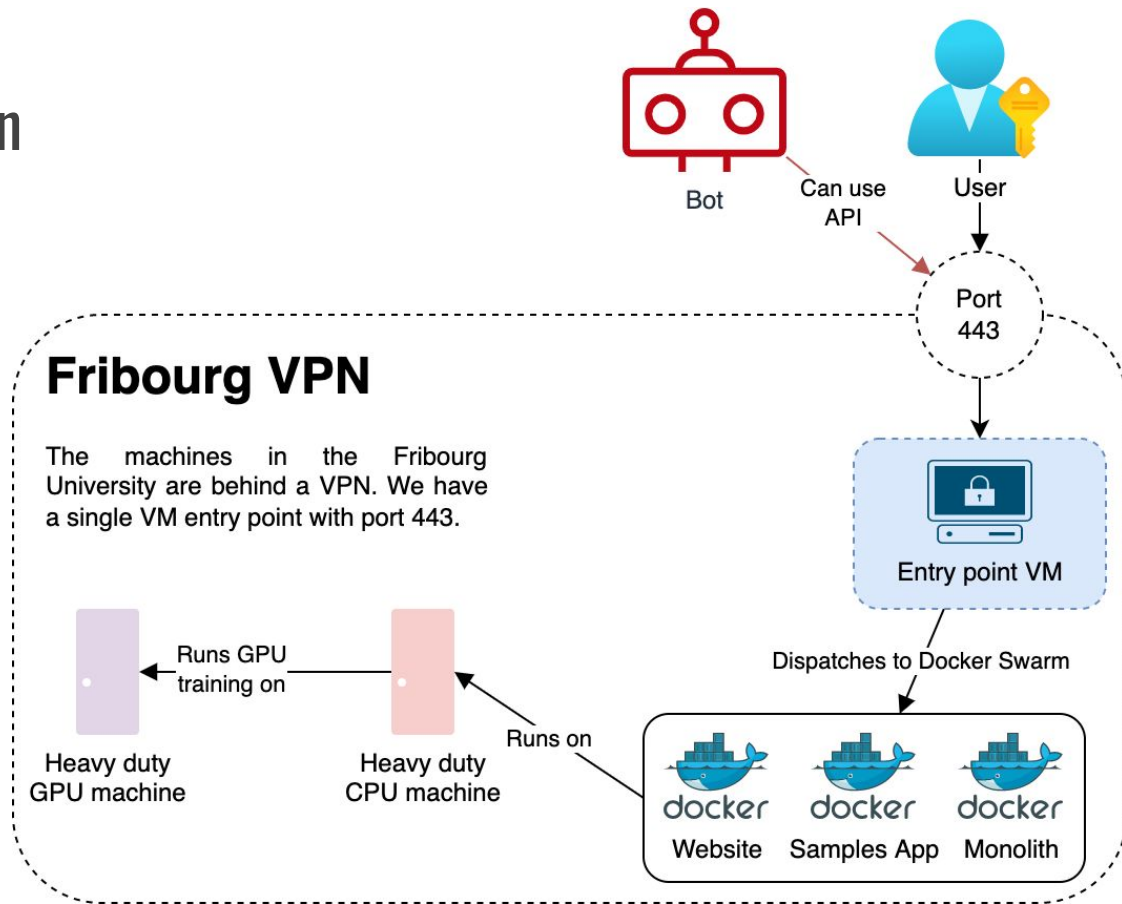
Future cluster configuration

In the new cluster, we want to move all services to a Docker Swarm, in **jail shells**

The VM will function as entry point for dispatching

The Docker Swarm will run on a heavy duty CPU machine

Training task will run on **Stefano Vani** GPU cluster



New Tools

Mascot

— — —

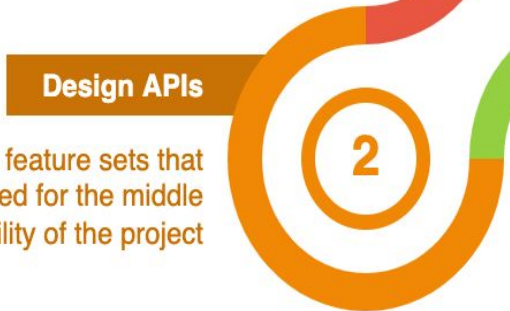
- Efficient validation for MGF files
- Efficient data format for MGF
- **Computes similarity between MGF files (to be finalized)**
- **Computes embedding of spectra (to be finalized)**

Timeline



Refactor compute cluster

Redesign the cluster to suite our project needs, setting up jail shells & docker swarm. Run at least a test on GPUs and order the dedicated machine.



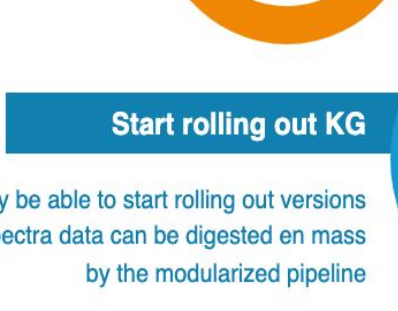
Design APIs

Design the objects and feature sets that we expect will be needed for the middle and long term viability of the project



Build backend for APIs

Concretely implement the APIs we agreed upon in a solid Monolith backend, setup continuous integration of third parties app



Start rolling out KG

At this point, we may be able to start rolling out versions of the KG, as the spectra data can be digested en mass by the modularized pipeline



Start running ML on KG

Start running ML tasks on the data we collected, including spectra-wise and KG-wise embedding