FaaS. Function as a Service

Goal

The goal of the lab project is to design and implement a FaaS (Function as a Service)

Operational definition of FaaS

At a basic level, a FaaS allows users to register functions and run them on-demand. The following additional considerations should be taken into account

1. The service exposes a REST API (https-based)
2. All accesses to the system are carried through the API
3. Users must be registered
    1. Registration is carried out through the API
4. Users must authenticate to operate
    1. The authentication method is left up to the implementer
5. Users can register functions
    1. Each user registers its own functions
    2. Users cannot share function registrations
6. Functions can be deregistered by the user that created them
7. A function consists of a docker image reference
8. A function admits just one string parameter.
    1. The string can encode other data types, but from the point of view of the FaaS, it is just a string
9. A function returns a string result.
    1. As before, the string can encode an arbitrary type, but the FaaS sees only strings.
        1. (think of JSON)
10. Functions are activated via an API call
    1. The call must carry with it the string parameter
    2. The call must result in the execution of a container based on the registered function (a docker image), configured to obtain the parameter.
    3. Completion of the call must result, on the part of the container, on the emission of the string result. stdout MUST be devoted to the emission of the result.
    4. Any logging must be emitted via stderr.
    5. How execution of the container proceeds is up to the implementer
    6. After an activation completes and the result is retrieved, the container must be eliminated.

11. A function activation call is synchronous: the caller waits for the result, or, alternatively, for an error.
12. The system should enable parallel activation of multiple functions.
    1. It is admissible to setup/configure a maximum number of concurrent activations
    2. If a maximum number of concurrent activations is configured, any activation requests received while the maximum is reached should be rejected with a corresponding error.

The above specifies a generic FaaS mechanism, capable of running almost anything.

The general architecture should contemplate AT LEAST these elements:

1. A reverse proxy performing at least authentication, but potentially load balancing
2. An API server implementing the API
3. A message queue, holding requests for activations, and responses of carried out activations.
4. A database holding specifications of registered users and functions.
5. A set of workers, capable of carrying out the activations.

Technology selection

Reverse proxy

1. The reverse proxy will be implemented with APISIX (https://apisix.apache.org/)
2. The message QUEUE will be implemented with NATS (https://nats.io/)
3. The database will be implemented as a key-value store. The NATS key-value store will be used
4. The API server and workers can be implemented either with the GO language or with typescript.
5.

What is expected

1. A complete description of the microservices architecture
    1. Microservices
    2. Their degree of replication
    3. Their elasticity (variation with load)
2. An implementation of the workers and the API server
3. A configuration of the reverse proxy
4. A description of the system as a docker-compose file
5. ~~A description of the system as a Kumori service~~
    1. ~~Including a description of each microservice as a Kumori component~~
6. A write-up describing the system. It should include the following:

    1. The microservices architecture,
    2. How the workers do work
    3. How workers are scaled
    4. How the DB scales
    5. How the whole system should be configured to adapt to changes in load
    6. How access to external services on the part of functions should be handled in your implementations
        1. What handicaps/virtues presents.
        2. What additional data ought to be included in the encoded strings
        3. Any other important matter
    7. Optionally () a comparison with other proposals for open source FaaS systems.
7. Code with the whole implementation and configurability points
    1. Dockerfile
    2. Packages/modules
    3. All in one git repo
    4. Handed in as a tgz file.