Dummy Workload Creation Guide

Due to lack of open source IoT applications following microservices architecture and inorder to evaluate application with a comprehensive workloads (covering a considerable range of MSA characteristics) we decided to evaluate the platform with a dummy workload.

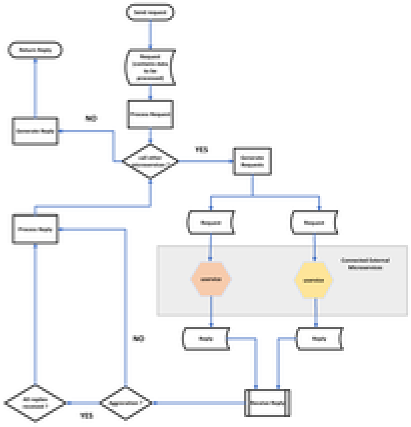
Requirements

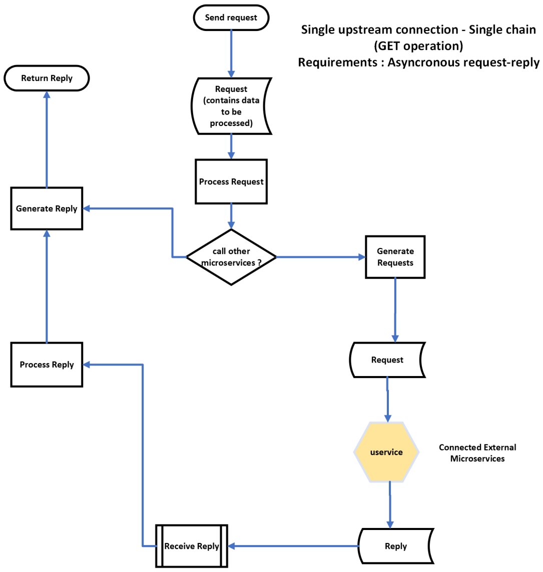
1. Model generic applications as DAGS. **IMPLEMENTED**
2. Ability to demonstrate different data flow patterns (a. chained b. aggregator c.hybrid). **IMPLEMENTED**
3. Mimic complex interactions such as shared microservices, candidate microservices. **IMPLEMENTED**
4. Asynchronous communication among microservices. **IMPLEMENTED**
5. Support for different programming languages. (Provide a template that can be adopted to other programming languages) **IMPLEMENTED** **TODO: SAMPLE PYTHON MICROSERVICE**
6. Easily configurable (configs handled through configmaps to enable easy integration to create different workloads) **IMPLEMENTED**

Background and Implementation details

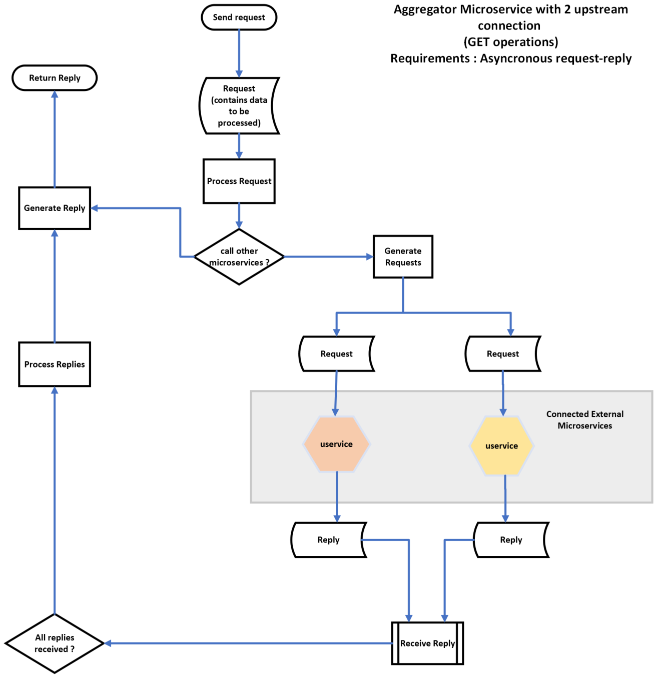
💡 Reactive Programming : This results in non-blocking / asynchronous API requests and is built around publish-subscribe model. This enables support for more concurrent users while minimising resource requirements as there’s no need to create more threads to maintain the required level of performance.

Dummy Microservice flow

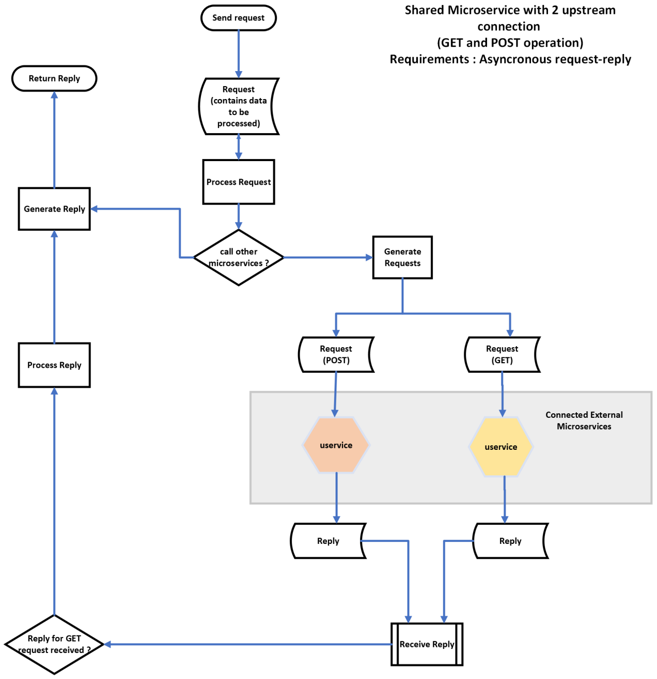




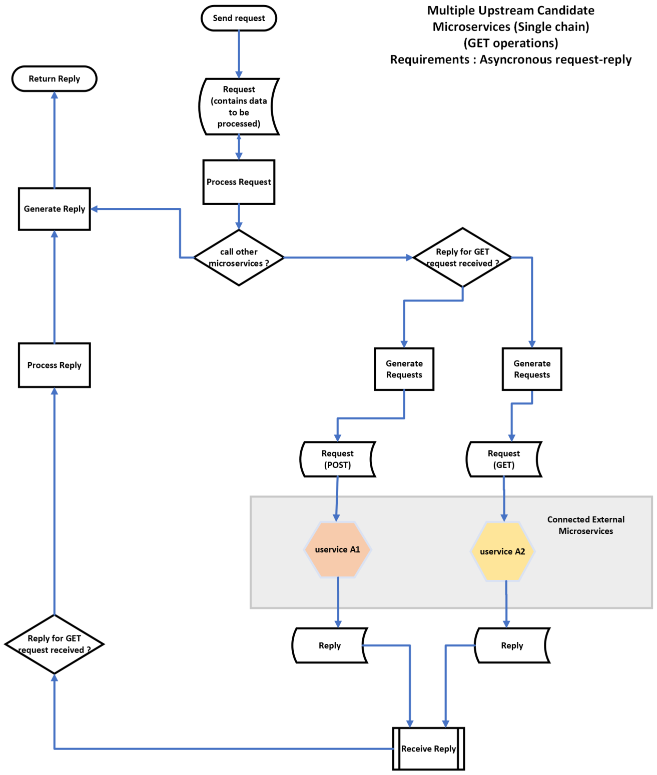
Single upstream connection - Single chain
(GET operation)
Requirements : Asyncronous request-reply



Aggregator Microservice with 2 upstream
connection
(GET operations)
Requirements : Asyncronous request-reply



Shared Microservice with 2 upstream
connection
(GET and POST operation)
Requirements : Asyncronous request-reply



Multiple Upstream Candidate
Microservices (Single chain)
(GET operations)
Requirements : Asyncronous request-reply

API

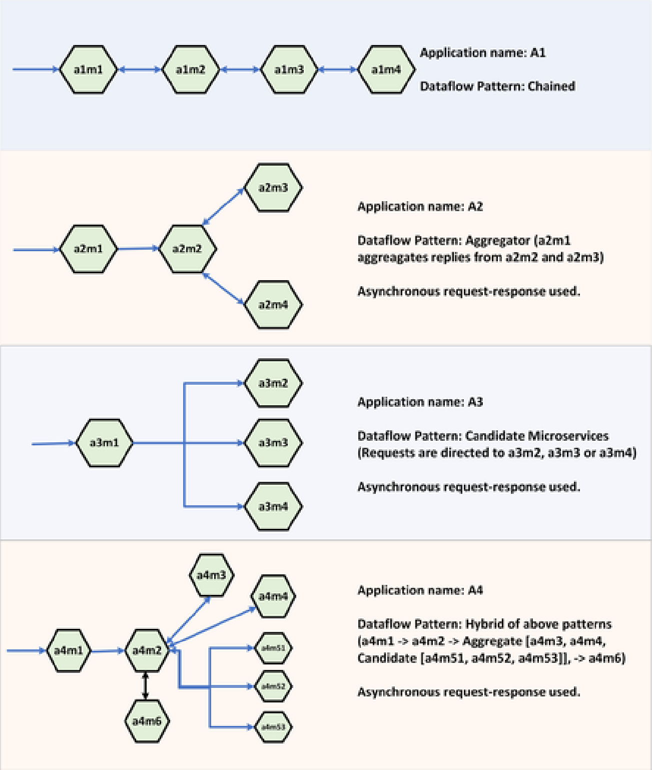
```
GET http://localhost:8080/DummyM/getResults
Content-Type: application/json

{  "message":[65,68,67],
   "dataProcessingTime":10
}
```

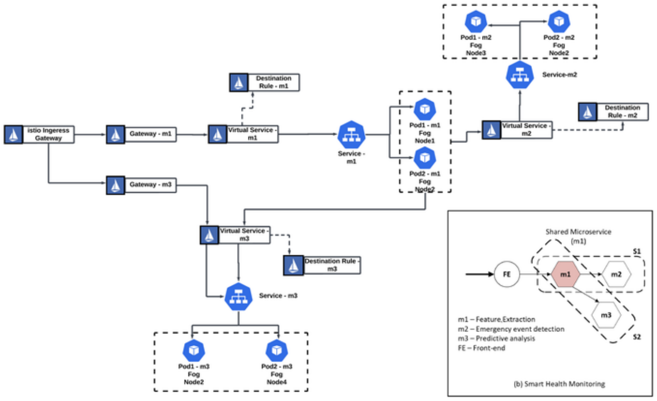
```
curl -HHost:a1m1.app1.svc.cluster.local -X GET "http://115.146.94.17:80
/DummyM/getResults" -H 'Content-Type: application/json' -d '{"message":
[65,68,67],"dataProcessingTime":30}'
```

Example Applications

Resources:

1. Public images for creating following composite services available at https://hub.docker.com/repository/docker/samodha/dummy-microservice
2. Dummy microservice code
    a. Chained pattern - Java - https://github.com/MSA-FogBus-Extension/WorkloadGenerator/tree/dummyMicroservices/dummy-micro
    b. Aggregator pattern - Java - https://github.com/MSA-FogBus-Extension/WorkloadGenerator/tree/dummyMicroservices/dummy-micro-aggregator
3. Deployment resources (Kubernetes and Istio)
    a. https://github.com/MSA-FogBus-Extension/WorkloadGenerator/tree/dummyMicroservices/sample-deployment-resources


a1m1
a1m2
a1m3
a1m4
Application name: A1
Dataflow Pattern: Chained
a2m1
a2m2
a2m3
a2m4
Application name: A2
Dataflow Pattern: Aggregator (a2m1 aggreagates replies from a2m2 and a2m3)
Asynchronous request-response used.
a3m1
a3m2
a3m3
a3m4
Application name: A3
Dataflow Pattern: Candidate Microservices (Requests are directed to a3m2, a3m3 or a3m4)
Asynchronous request-response used.
a4m1
a4m2
a4m3
a4m4
a4m51
a4m52
a4m53
a4m6
Application name: A4
Dataflow Pattern: Hybrid of above patterns (a4m1 -> a4m2 -> Aggregate [a4m3, a4m4, Candidate [a4m51, a4m52, a4m53]], -> a4m6)
Asynchronous request-response used.


istio Ingeress Gateway
Gateway - m1
Gateway - m3
Destination Rule - m1
Virtual Service - m1
Service - m1
Pod1 - m1 Fog Node1
Pod2 - m1 Fog Node2
Virtual Service - m2
Destination Rule - m2
Service-m2
Pod1 - m2 Fog Node3
Pod2 - m2 Fog Node2
Virtual Service - m3
Destination Rule - m3
Service - m3
Pod1 - m3 Fog Node2
Pod2 - m3 Fog Node4
Shared Microservice (m1)
S1
S2
FE
m1
m2
m3
m1 – Feature,Extraction
m2 – Emergency event detection
m3 – Predictive analysis
FE – Front-end
(b) Smart Health Monitoring