Github kode [Schwencke/DSL-COMP (github.com)](https://github.com/Schwencke/DSL-COMP)

Fault Isolation ( See swimlane.pdf)

Services communicate with http and thru the message broker RabbitMQ.

Calc-Service receives requests from the web UI and distributes it to the different services, addition, and subtraction.

The Subtraction and addition service will post the results via RabbitMQ when they are done calculating, and Calc-Service will listen for these results and persist them accordingly.

**For fault isolation here are some of the strategies implemented.**

* Service Modularity
* Logging and Monitoring
* Distributed tracing
* Independent service through containerization
* Error handling
* Statelessness
* Idempotency
* Circuit braker and retry mechanisms

# Scaling evidence

I have attached a PDF called dockerlogs, it contains evidence that multiple instances are up and running and requests are equally distributed. Also in the performance report, performance-report.pdf, it is visualized that the application is capable of handling many requests.

# Adding additional endpoints.

**1. Create the Multiplication Service:**

1. **Initialize** a new service like the addition and subtraction services.
2. **Endpoint**:
   * Create an endpoint, e.g., **/Multiply**, which:
     + Accepts a request with two numbers.
     + Calculates the multiplication of those numbers.
     + Publishes an event after the calculation.
     + Returns the result.
3. **Testing**: Before integrating it with the main API, ensure this service is running and can handle multiplication requests correctly.

**2. Modify the Main API:**

1. **HttpClient Configuration**:
   * Add a new HttpClient configuration in the main API for the multiplication service
2. **Endpoint Creation**:
   * Add a new endpoint, e.g., **/multiplication**, which:
     + Accepts a multiplication request.
     + Uses the configured HttpClient to make a request to the multiplication service.
     + Handles any errors or issues with the request.
     + Returns the multiplication result to the caller.
   * Adding distributed tracing around the logic for consistency.
3. **Testing**:
   * Once integrated, test the main API's multiplication endpoint to ensure it correctly communicates with the multiplication service and returns the expected results.
4. Docker:
   * Add the new service to the docker-compose file and configure it.

# Using docker swarm.

Setup or join a swarm.

Update or configure docker-compose to have an update\_config stating the degree of parallelism and the delay. Also the default number of replicas should be set (if not already).

Then you can use the *Docker stack deploy* with the docker-compose file. You can also manually add single services to a swarm using the *docker service create –replicas [number of replicas] – name [name] [image]* command.

When scaling in a swarm you can use the scale command.