Wandering and getting lost:

the architecture of an app activating local communities on dementia issues

Nicklas S. Andersen, PhD Student in CS
Marco Chiarandini, Asc. Prof. in CS
Jacopo Mauro, Asc. Prof. in CS

University of Southern Denmark
Department of Mathematics & Computer Science

SEH 2021, June, 2021

Outline

1 Sammen Om Demens

2 Implementation

3 Experimental Setup

4 Results

5 Conclusion & Future Work

Sammen Om Demens (SOD)

- Introduction

- What is SOD?
  - An app implemented for a Danish municipality

Sammen Om Demens (SOD)

- Introduction

- What is SOD?
  - An app implemented for a Danish municipality
- Motivation:
  - Improve the handling of cases where people with dementia get lost
  - Use new technological innovations in doing so

Sammen Om Demens (SOD)

- Introduction

- What is SOD?
  - An app implemented for a Danish municipality
- Motivation:
  - Improve the handling of cases where people with dementia get lost
  - Use new technological innovations in doing so
- Goals:
  - Create awareness about dementia among ordinary citizens
  - Involve ordinary citizens in helping persons with dementia
  - Alleviate the anxiety of persons with dementia and caregivers

Sammen Om Demens (SOD)

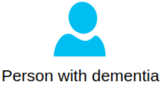
- Overview

- The goals are accomplished through features of the SOD application

Sammen Om Demens (SOD)

- Overview

- The goals are accomplished through features of the SOD application



Sammen Om Demens (SOD)

- Overview

- The goals are accomplished through features of the SOD application

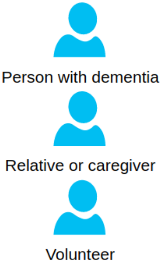
Person with dementia

Relative or caregiver

Sammen Om Demens (SOD)

- Overview

- The goals are accomplished through features of the SOD application



Person with dementia

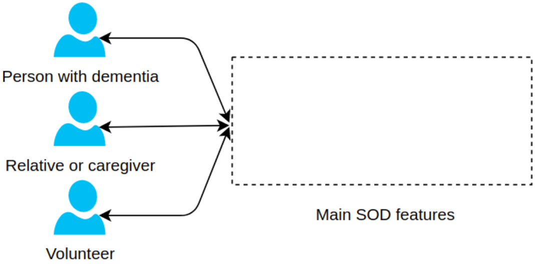
Relative or caregiver

Volunteer

Sammen Om Demens (SOD)

- Overview

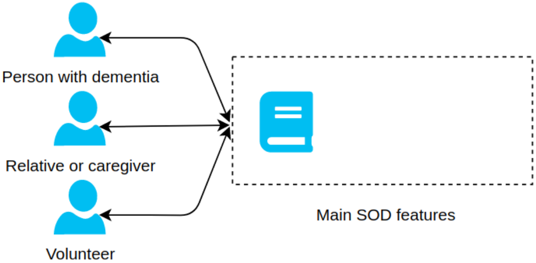
- The goals are accomplished through features of the SOD application



Sammen Om Demens (SOD)

- Overview

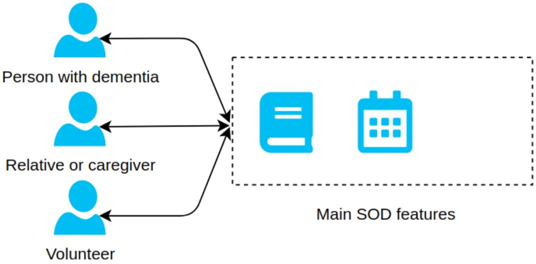
- The goals are accomplished through features of the SOD application:
  - A knowledge bank



Sammen Om Demens (SOD)

- Overview

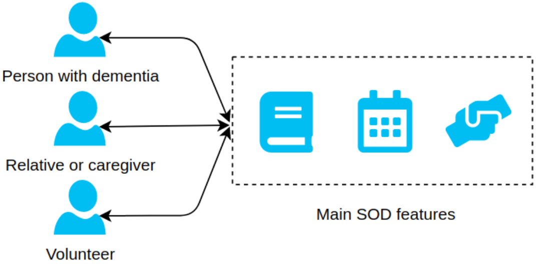
- The goals are accomplished through features of the SOD application:
  - A knowledge bank
  - A recreational activity calendar



Sammen Om Demens (SOD)

- Overview

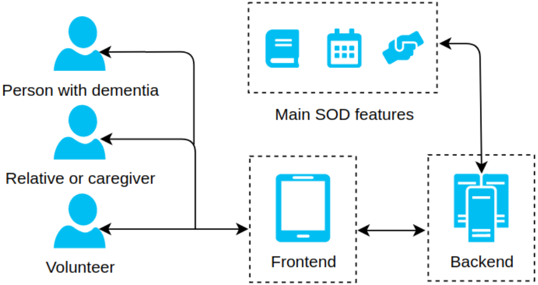
- The goals are accomplished through features of the SOD application:
  - A knowledge bank
  - A recreational activity calendar
  - A help component



Sammen Om Demens (SOD)

- Overview

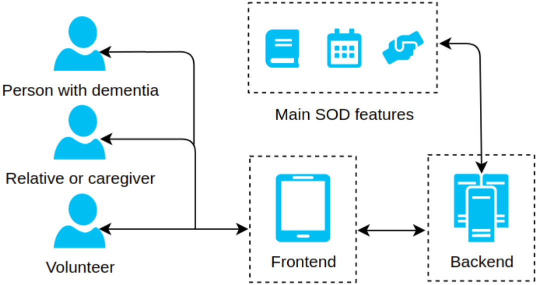
- The goals are accomplished through features of the SOD application:
  - A knowledge bank
  - A recreational activity calendar
  - A help component



Sammen Om Demens (SOD)

- Overview

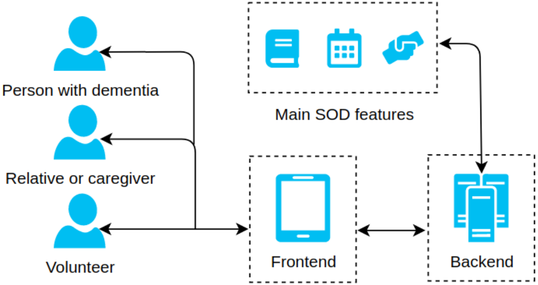
- The goals are accomplished through features of the SOD application:
  - A knowledge bank
  - A recreational activity calendar
  - A help component
- Backend system requirements:



Sammen Om Demens (SOD)

- Overview

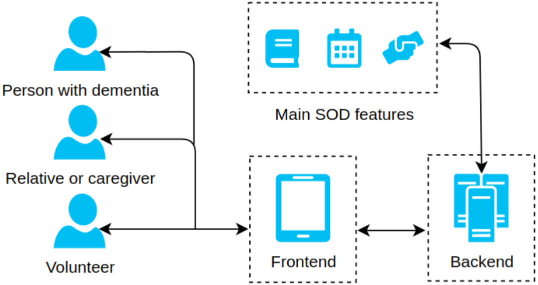
- The goals are accomplished through features of the SOD application:
  - A knowledge bank
  - A recreational activity calendar
  - A help component
- Backend system requirements:
  - It should be scalable and able to process data effeciently and reliably



Sammen Om Demens (SOD)

- Overview

- The goals are accomplished through features of the SOD application:
  - A knowledge bank
  - A recreational activity calendar
  - A help component
- Backend system requirements:
  - It should be scalable and able to process data effeciently and reliably
  - It should be maintainable and structurally flexible



Outline

1 Sammen Om Demens

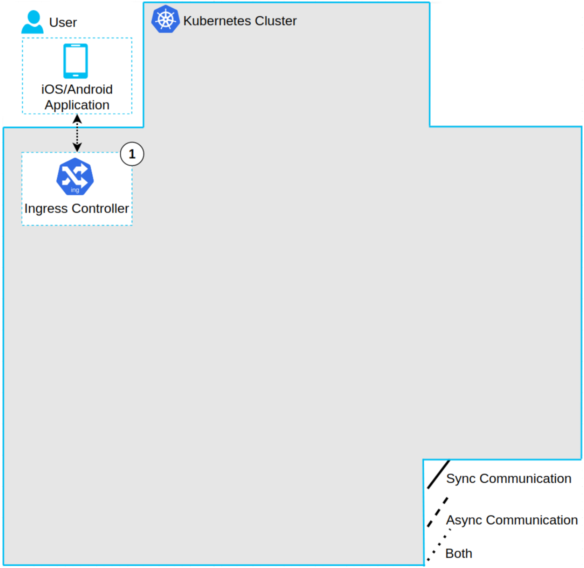
2 Implementation

3 Experimental Setup

4 Results

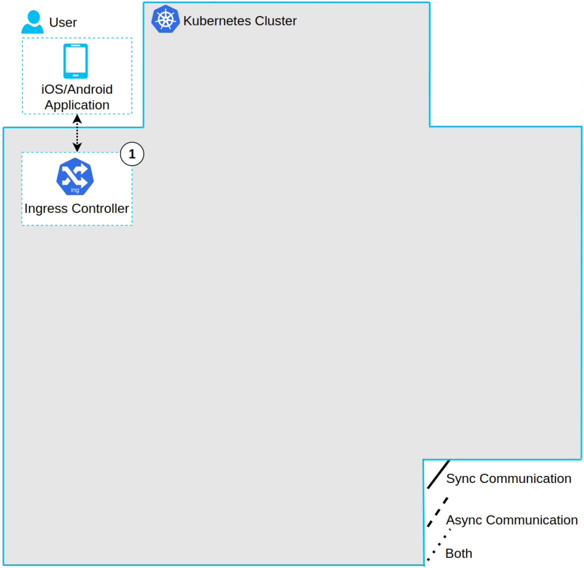
5 Conclusion & Future Work

Implementation



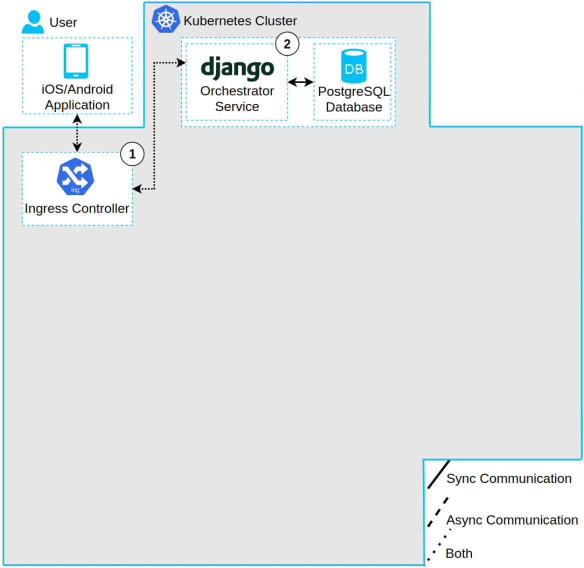
Implementation

- The entry point of the system is an Ingress Controller ①



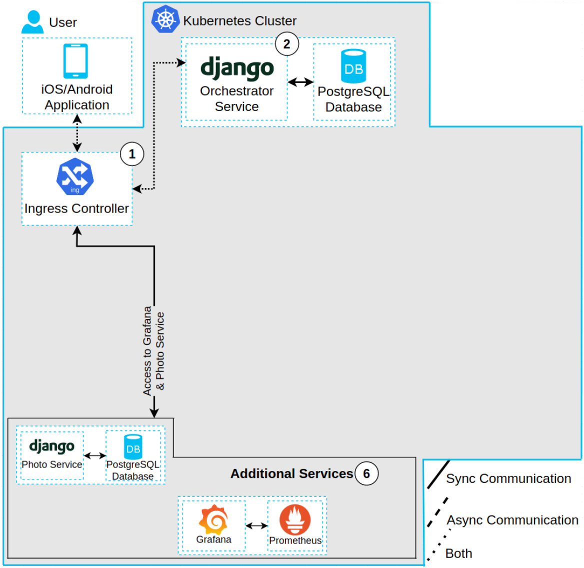
Implementation

- The entry point of the system is an Ingress Controller ①
- It redirects traffic to the Orchestrator ②



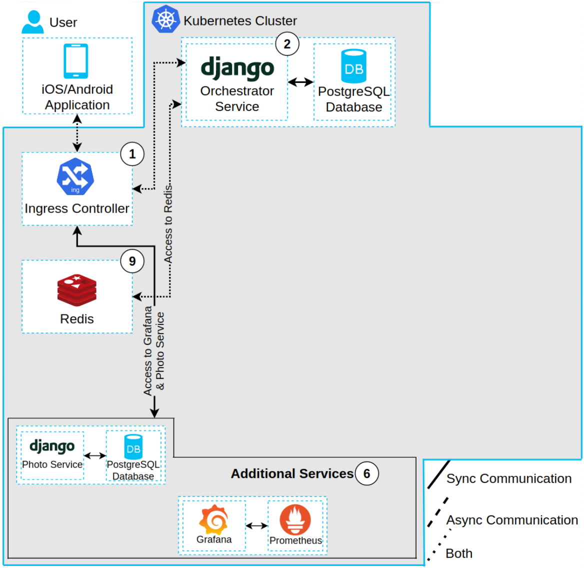
Implementation

- The entry point of the system is an Ingress Controller ①
- It redirects traffic to the Orchestrator ②
- It also reserves a direct route to Additional Services ⑥



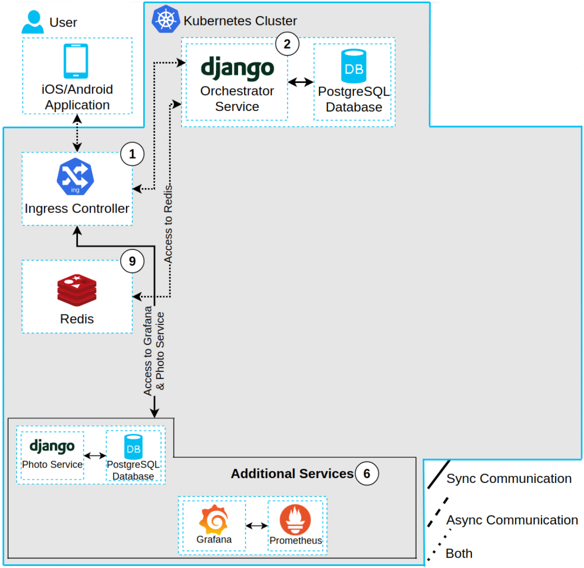
Implementation

- The Orchestrator ② routes requests to other microservices through Redis ⑨



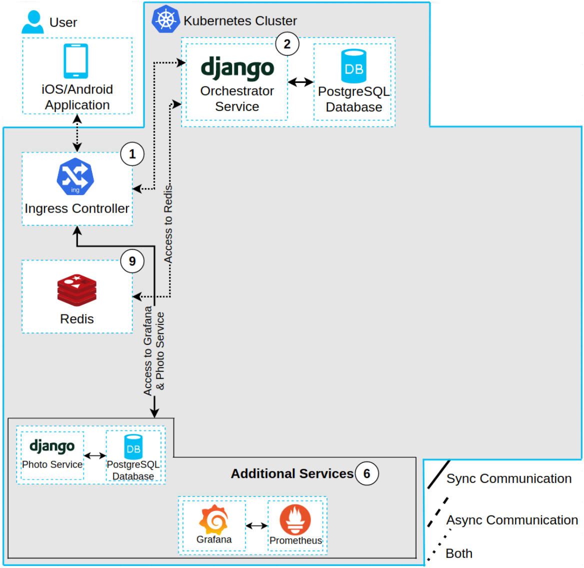
Implementation

- The Orchestrator ② routes requests to other microservices through Redis ⑨
- It handles the creation, activation, deletion, update, and retrieval of users



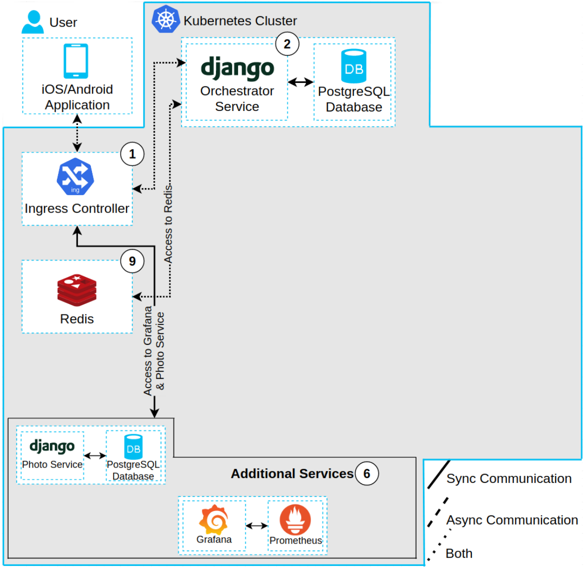
Implementation

- The Orchestrator ② routes requests to other microservices through Redis ⑨
- It handles the creation, activation, deletion, update, and retrieval of users
- It makes it easy to authorize and authenticate a user in a single place



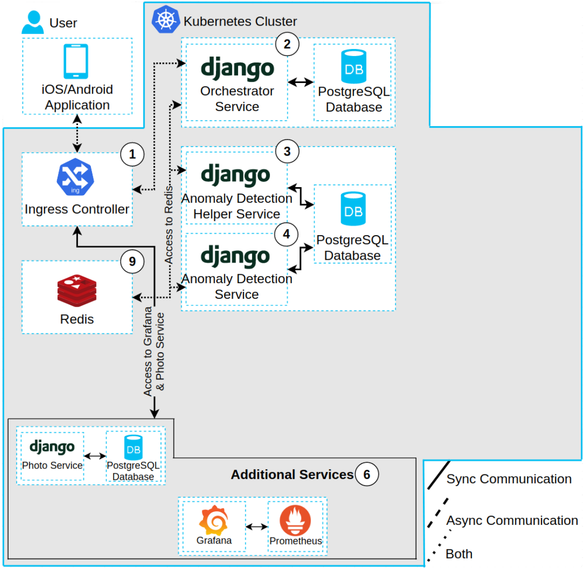
Implementation

- The Orchestrator ② routes requests to other microservices through Redis ⑨
- It handles the creation, activation, deletion, update, and retrieval of users
- It makes it easy to authorize and authenticate a user in a single place
- It handles WebSocket connections



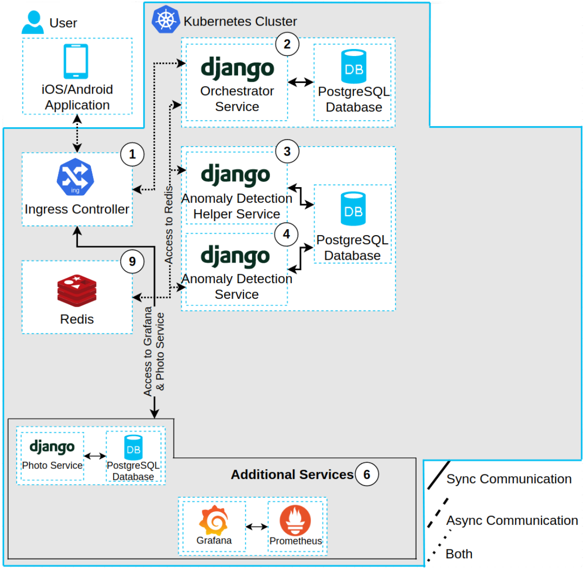
Implementation

- The help component is implemented by ③ and ④



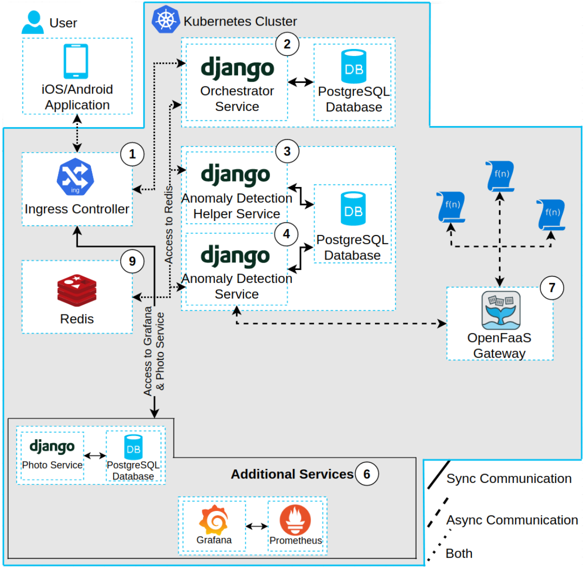
Implementation

- The help component is implemented by ③ and ④
- Microservice ③ acts as a database buffer and handles bulk operations on raw data



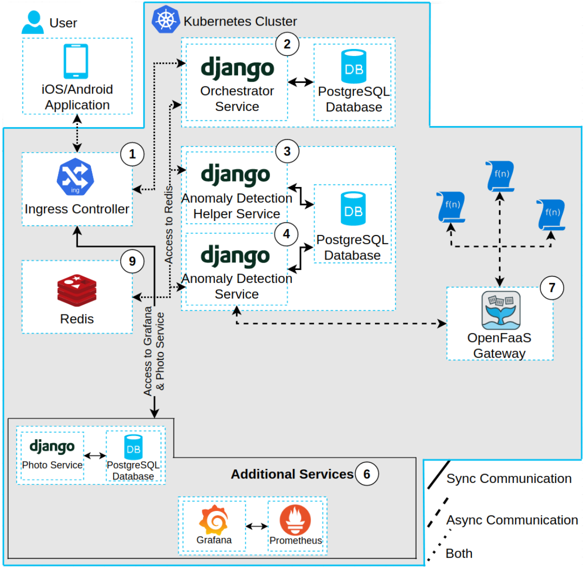
Implementation

- The help component is implemented by ③ and ④
- Microservice ③ acts as a database buffer and handles bulk operations on raw data
- Microservice ④ handles coordination of function execution triggered through the OpenFaaS Gateway ⑦



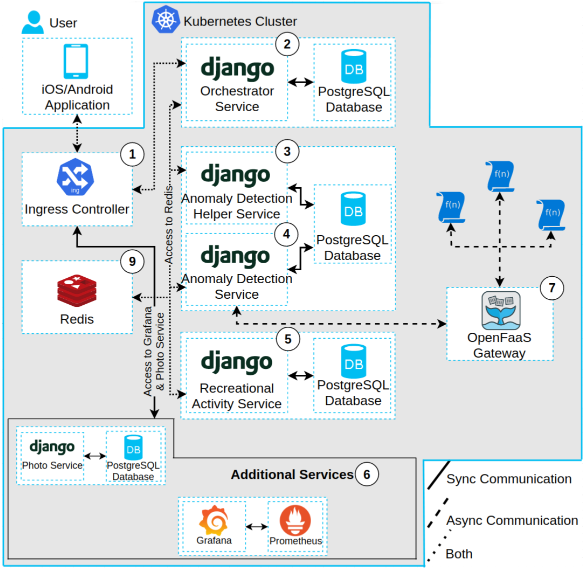
Implementation

- The help component is implemented by ③ and ④
- Microservice ③ acts as a database buffer and handles bulk operations on raw data
- Microservice ④ handles coordination of function execution triggered through the OpenFaaS Gateway ⑦
- OpenFaaS provides the infrastructure for implementing the detection algorithms



Implementation

- The recreational activity calendar is implemented by ⑤



Outline

1 Sammen Om Demens

2 Implementation

3 Experimental Setup

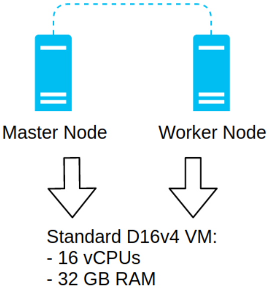
4 Results

5 Conclusion & Future Work

Experimental Setup

- System Configuration

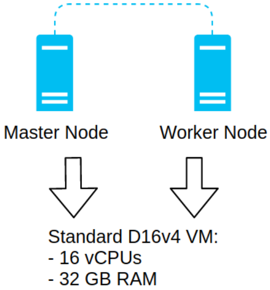
- Microsoft Azure Setup:



Standard D16v4 VM:
- 16 vCPUs
- 32 GB RAM

Experimental Setup
- System Configuration

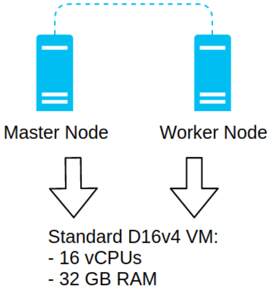
- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node



Experimental Setup

- System Configuration

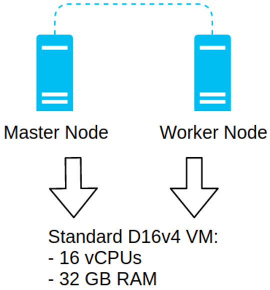
- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage



Experimental Setup

- System Configuration

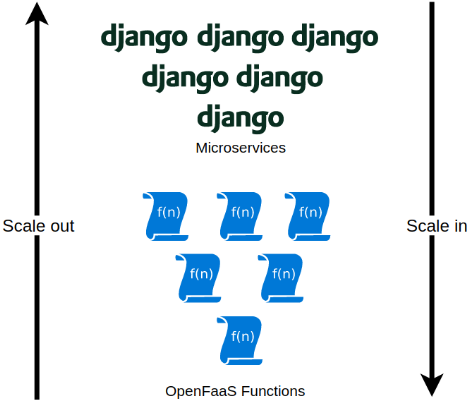
- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
- Kubernetes autoscaling configuration:



Experimental Setup

- System Configuration

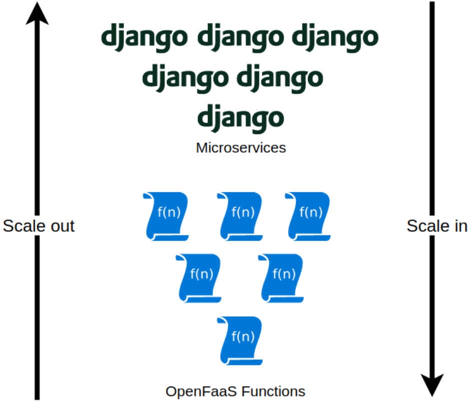
- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
- Kubernetes autoscaling configuration:
  - Min & max number of replicas



Experimental Setup

- System Configuration

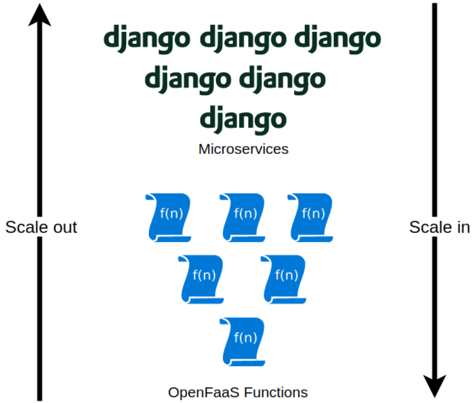
- **Microsoft Azure Setup:**
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
- **Kubernetes autoscaling configuration:**
  - Min & max number of replicas
  - CPU and memory resource requests



Experimental Setup

- System Configuration

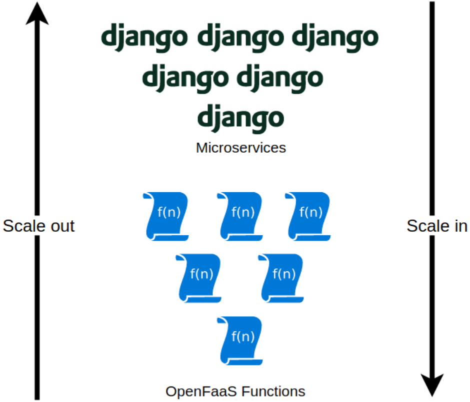
- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
- Kubernetes autoscaling configuration:
  - Min & max number of replicas
  - CPU and memory resource requests
  - Autoscaling triggered based on:



Experimental Setup

- System Configuration

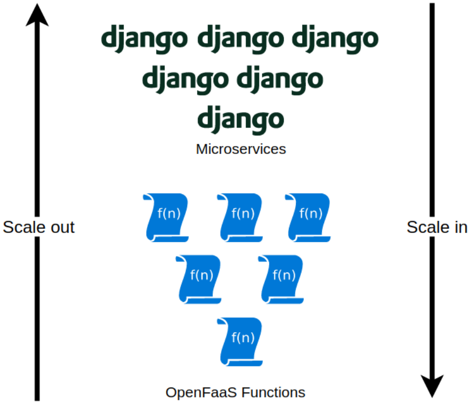
- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
- Kubernetes autoscaling configuration:
  - Min & max number of replicas
  - CPU and memory resource requests
  - Autoscaling triggered based on:
    - CPU utilization for microservices



Experimental Setup

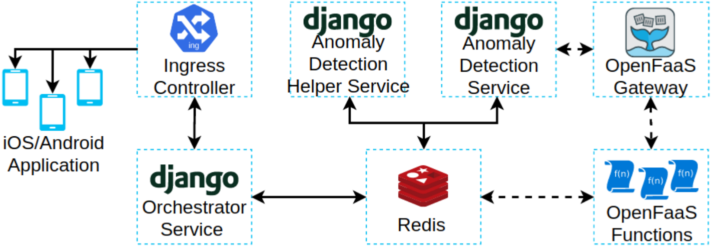
- System Configuration

- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
- Kubernetes autoscaling configuration:
  - Min & max number of replicas
  - CPU and memory resource requests
  - Autoscaling triggered based on:
    - CPU utilization for microservices
    - Requests per second for OpenFaaS functions



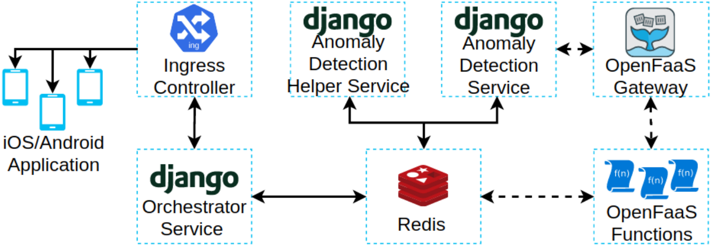
Experimental Setup

- Load Test Description


Ingress Controller
django Anomaly Detection Helper Service
django Anomaly Detection Service
OpenFaaS Gateway
iOS/Android Application
django Orchestrator Service
Redis
OpenFaaS Functions

Experimental Setup

- Load Test Description

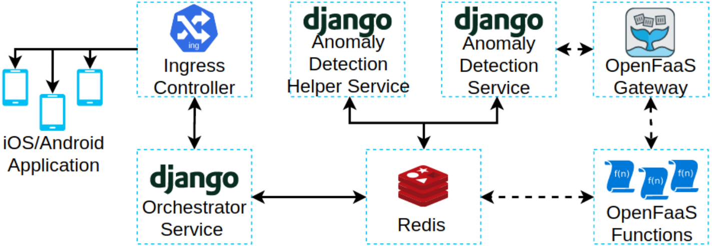


Ingress Controller
django Anomaly Detection Helper Service
django Anomaly Detection Service
OpenFaaS Gateway
iOS/Android Application
django Orchestrator Service
Redis
f(n)
f(n)
f(n)
OpenFaaS Functions

- Load tests target the infrastructure used by the help component

Experimental Setup

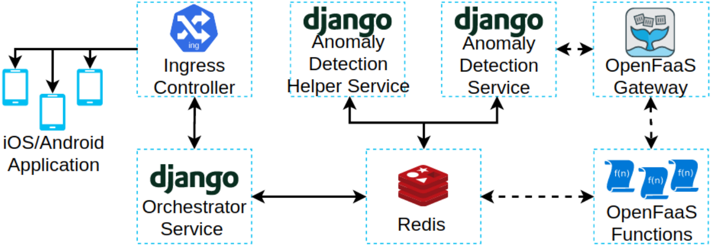
- Load Test Description



- Load tests target the infrastructure used by the help component
- They measure the performance of the system under the following conditions:

Experimental Setup

- Load Test Description

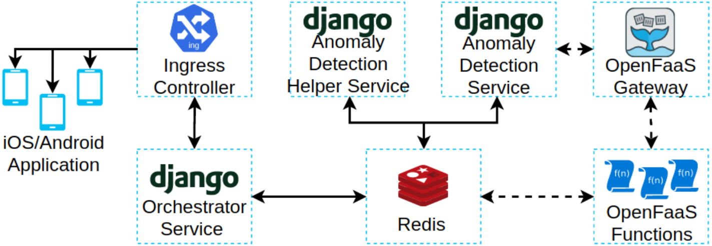


Ingress Controller
django Anomaly Detection Helper Service
django Anomaly Detection Service
OpenFaaS Gateway
iOS/Android Application
django Orchestrator Service
Redis
f(n)
f(n)
f(n)
OpenFaaS Functions

- Load tests target the infrastructure used by the help component
- They measure the performance of the system under the following conditions:
  - HTTP POST requests are sent to the backend system every $\tau \in U(1, 5)$ seconds

Experimental Setup

- Load Test Description


Ingress Controller
django Anomaly Detection Helper Service
django Anomaly Detection Service
OpenFaaS Gateway
iOS/Android Application
django Orchestrator Service
Redis
f(n)
OpenFaaS Functions

- Load tests target the infrastructure used by the help component
- They measure the performance of the system under the following conditions:
  - HTTP POST requests are sent to the backend system every $\tau \in U(1, 5)$ seconds
  - Invoked functions compute a moving average of the incoming location data

Outline

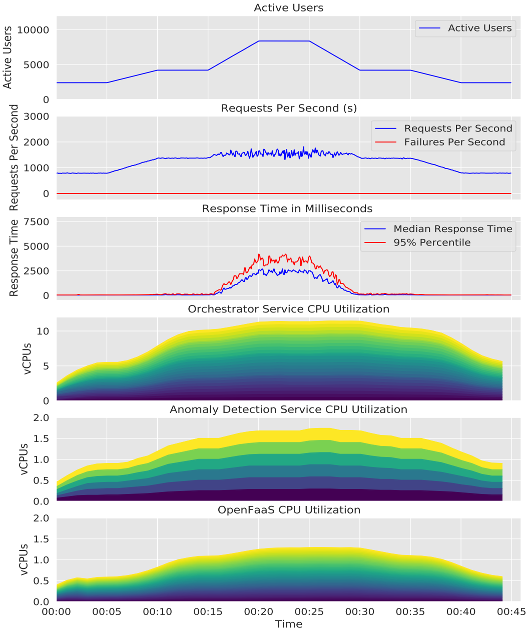
1 Sammen Om Demens

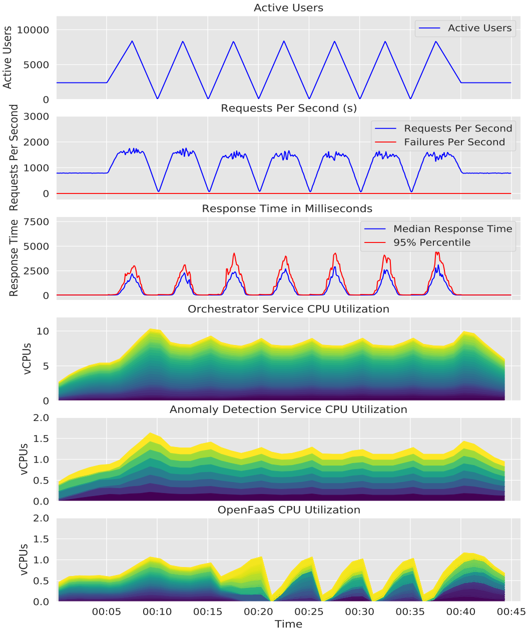
2 Implementation

3 Experimental Setup

4 Results

5 Conclusion & Future Work


Active Users
Active Users
10000
5000
0
Active Users
Requests Per Second (s)
Requests Per Second
3000
2000
1000
0
Requests Per Second
Failures Per Second
Response Time in Milliseconds
Response Time
7500
5000
2500
0
Median Response Time
95% Percentile
Orchestrator Service CPU Utilization
vCPUs
10
5
0
Anomaly Detection Service CPU Utilization
vCPUs
2.0
1.5
1.0
0.5
0.0
OpenFaaS CPU Utilization
vCPUs
2.0
1.5
1.0
0.5
0.0
00:00
00:05
00:10
00:15
00:20
00:25
00:30
00:35
00:40
00:45
Time


Active Users
Active Users
10000
5000
0
Active Users
Requests Per Second (s)
Requests Per Second
3000
2000
1000
0
Requests Per Second
Failures Per Second
Response Time in Milliseconds
Response Time
7500
5000
2500
0
Median Response Time
95% Percentile
Orchestrator Service CPU Utilization
vCPUs
10
5
0
Anomaly Detection Service CPU Utilization
vCPUs
2.0
1.5
1.0
0.5
0.0
OpenFaaS CPU Utilization
vCPUs
2.0
1.5
1.0
0.5
0.0
00:05
00:10
00:15
00:20
00:25
00:30
00:35
00:40
00:45
Time

Outline

1 Sammen Om Demens

2 Implementation

3 Experimental Setup

4 Results

5 Conclusion & Future Work

Conclusion & Future Work

- We designed the backend system of SOD relying on:

Conclusion & Future Work

- We designed the backend system of SOD relying on:
    - Microservices and serverless computing

Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - $\rightsquigarrow$ Structurally flexible and maintainable system

Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - $\rightsquigarrow$ Structurally flexible and maintainable system
- We confirmed with load tests that:

Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - ⇝ Structurally flexible and maintainable system
- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately

Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - ⇝ Structurally flexible and maintainable system
- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably

Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - ⇝ Structurally flexible and maintainable system
- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably
- We need to extended and improve the functionalities:

Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - ⇝ Structurally flexible and maintainable system
- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably
- We need to extended and improve the functionalities:
  - Implement artificial intelligence techniques

Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - ⇝ Structurally flexible and maintainable system
- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably
- We need to extended and improve the functionalities:
  - Implement artificial intelligence techniques
  - Build out the OpenFaaS function execution pipeline

Thank you for your attention!