Migrating from Microservices to Serverless: An IoT Platform Case Study

8th International Workshop on Serverless Computing (WoSC8)
In conjunction with, ACM/IFIP Middleware 2022

Mohak Chadha, Victor Pacyna, Anshul Jindal, Jianfeng Gu, Michael Gerndt

mohak.chadha@tum.de, victor.pacyna@tum.de, anshul.jindal@tum.de ,jianfeng.gu@tum.de, gerndt@in.tum.de

Chair of Computer Architecture and Parallel Systems,

Technical University of Munich (TUM)

Garching (near Munich), Germany

https://www.serverlesscomputing.org/wosc8/

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

Microservices Architecture

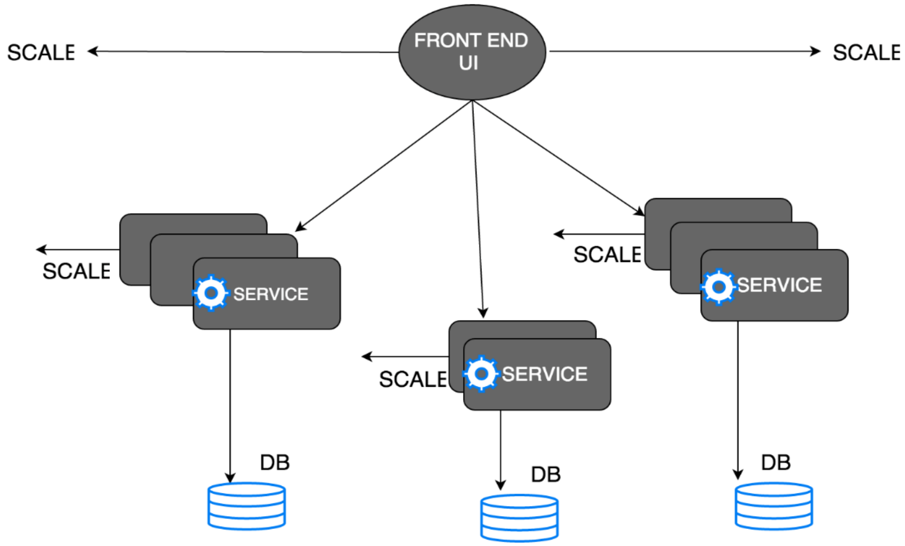
- The idea here is to split the application into a set of smaller and interconnected services.

Advantages:

- Easier to understand and maintain.
- Independence of Service.
- No Barrier on Adopting New Technologies.
- Independent Service Deployment.
- Each Service Scaling.

Disadvantages:

- Complexity of creating a distributed system.
- Deployment complexity.

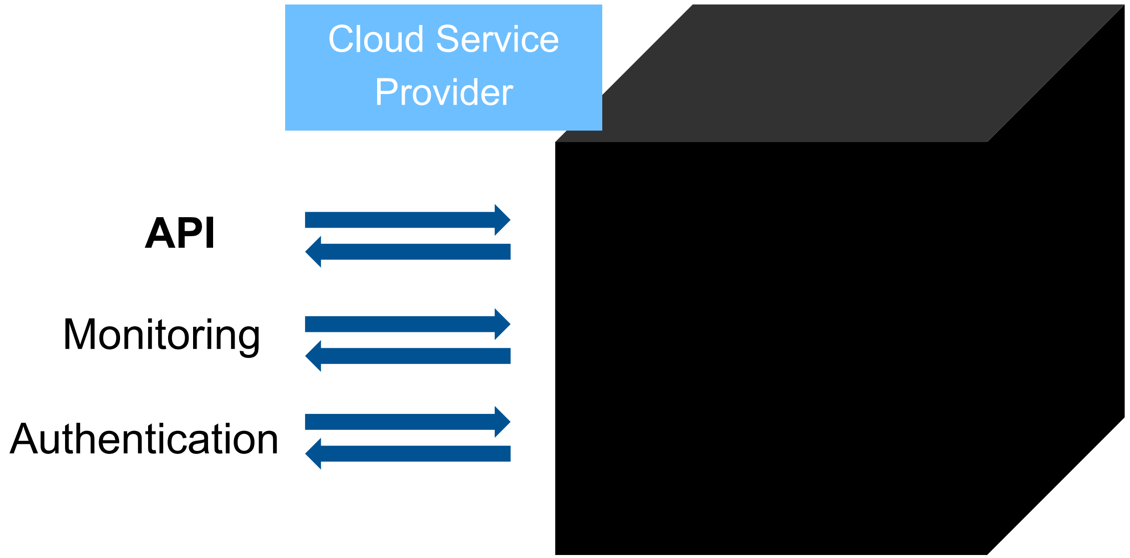


Introducing: Serverless

Serverless Computing
- Provide a platform that developers can use.
- No infrastructure management.

Function-as-a-Service (FaaS):
- Provides API endpoints that users can call.
- Smaller granularity than microservices.
- Billed per-use.
- Scale-to-zero.

Disadvantages:
- Focused on stateless functions.
- Performance variations due to restart latencies.



Microservices or Serverless?

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

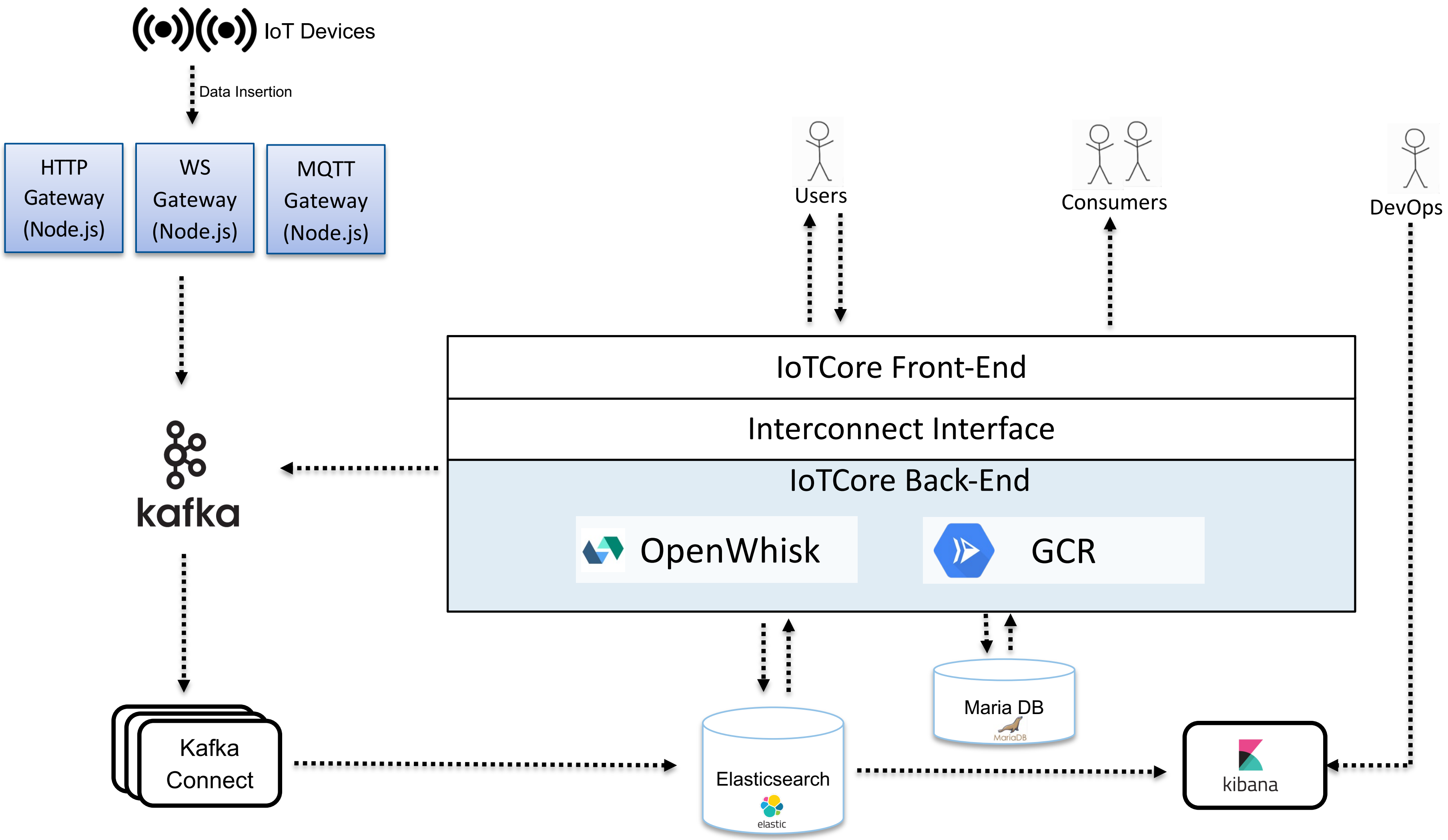
Goals

1. Migrate a IoT platform application onto OpenWhisk and GCR.
2. Performance evaluation of different deployment strategies.
3. Lessons Learned.

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

IoT Platform



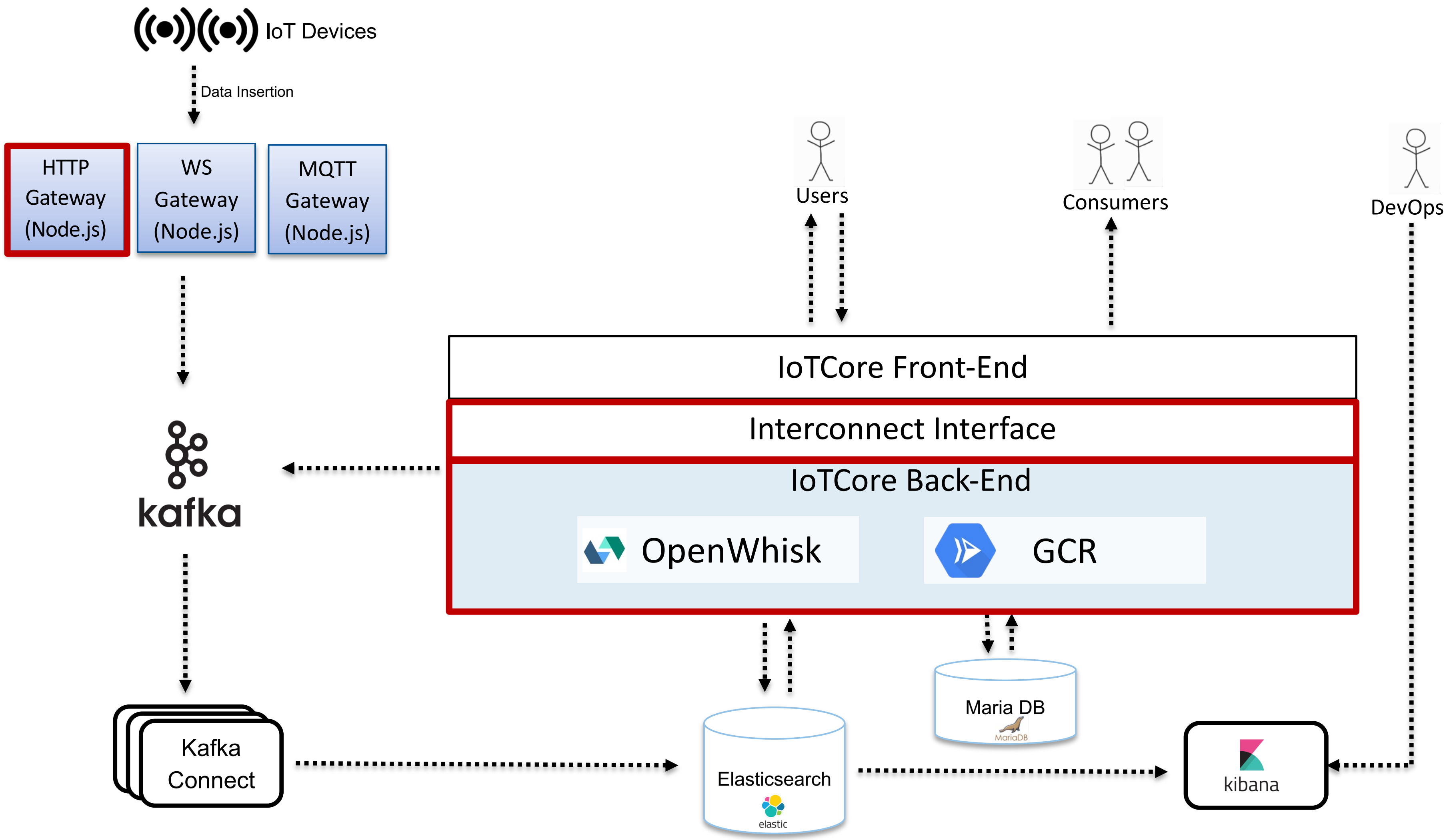
Endpoints under investigation:

- `Users-Get`
- `Devices-Add`
- `Devices-Get`
- `Sensors-Get`
- `HTTP-Gateway`
- `Consumers-Consume-Get`

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

Migration Methodology



Endpoints under investigation:

- `Users-Get`
- `Devices-Add`
- `Devices-Get`
- `Sensors-Get`
- `HTTP-Gateway`
- `Consumers-Consume-Get`

Migration Methodology

```
const Devicecontroller = new class extends BaseController {

    getAll(req, res) {
        return res.status(200).json({...});
    }

    add(req, res) {
        return res.status(200).json({...});
    }

    update(req, res) {
        return res.status(200).json({...});
    }

    delete(req, res) {
        return res.status(200).json({...});
    }

};
```

Flatten the services

```
const Devicecontroller = new class extends BaseController {

    getAll(req, res) {
        return res.status(200).json({...});
    }

};
```

```
const Devicecontroller = new class extends BaseController {

    add(req, res) {
        return res.status(200).json({...});
    }

};
```

```
const Devicecontroller = new class extends BaseController {

    update(req, res) {
        return res.status(200).json({...});
    }

};
```

```
const Devicecontroller = new class extends BaseController {

    delete(req, res) {
        return res.status(200).json({...});
    }

};
```

Migration Methodology

- Externalize state
- Adjust communication protocols
- Add abstraction layer for invocations

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

Experiment Setup

- Shared services on a separate Google Kubernetes Engine (GKE) cluster

- Four different deployments for IoT core and gateway functionalities:
  - GKE-50 (horizontal pod autoscaling set to 50% CPU utilization)
  - GKE-80 (80% CPU utilization)
  - OW (on top of GKE)
  - GCR

Experiment Procedure

- Performance analysis was conducted using K6 for load testing
- K6's principle: Virtual users try to send as many requests as possible
- Tests were scheduled on a separate VM on the Institution's Compute Cloud
- Three workload patterns (linear, random, spike) for 30 min each
- In total:
  - 3 workloads x 4 deployments x 6 endpoints = 72 experiment runs of 30 min
  - Total number of requests in eight figures:

$$\bar{x}_{Linear} = 281{,}527;\ \bar{x}_{Random} = 262{,}495;\ \bar{x}_{Spike} = 77{,}310$$

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

Performance Analysis

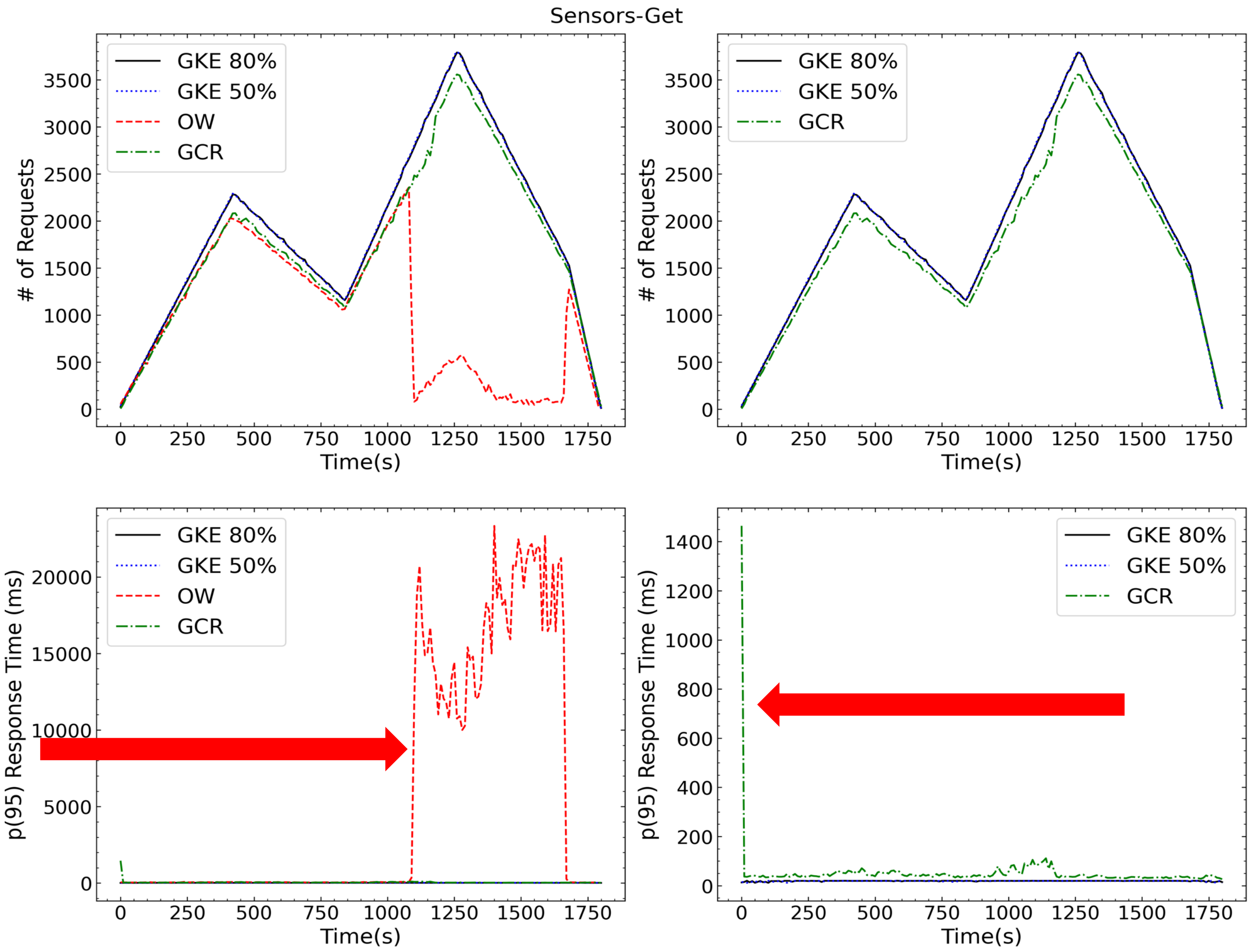


Figure: Sensors-Get APIEndpoint for the Random Workload.

Performance Analysis

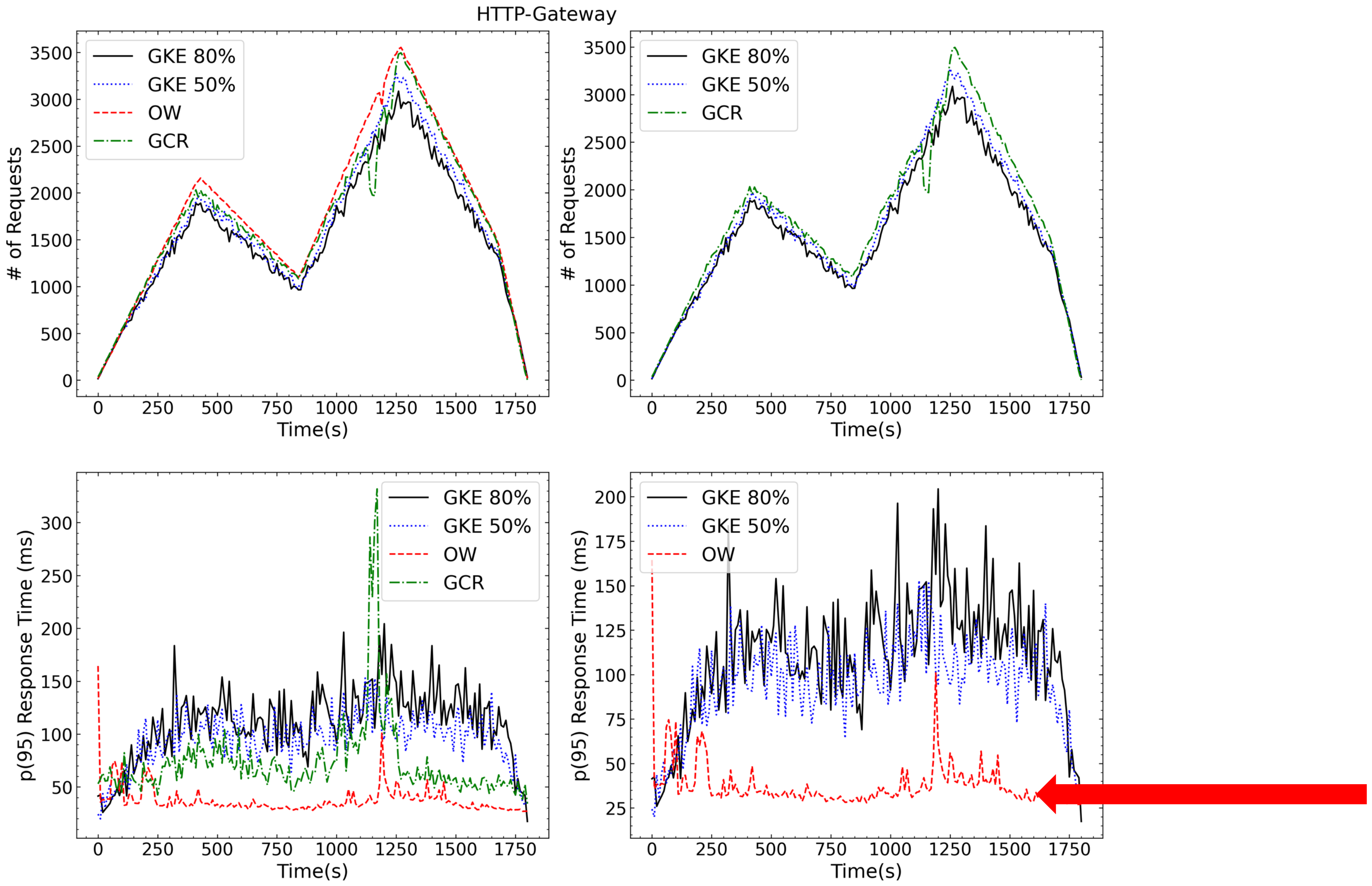


Figure: HTTP-Gateway APIEndpoint for the Random Workload.

Takeaways: Performance Analysis

- Generally, GKE setups with highest performance, followed by GCR. OW fell behind for higher workloads.
- Marginal differences for three workload patterns. OW dysfunctional when ~2,000 requests per 10 sec exceeded.
- Autoscaling without large effects, GKE-50 performed slightly better than GKE-80.
- GCR in general outperformed OW and was more robust.

Cost Analysis

- GKE and OW with reservation-based pricing, GCR pay-per-use model

API Endpoint	Workload	GKE-50	GKE-80	OW	GCR
Sensors-Get	Linear	0.1054	0.1053	0.5195	**0.0542**
	Random	0.1054	1.0433	0.4752	**0.0544**
	Spike	0.1054	1.0433	1.2230	**0.0586**
HTTP-Gateway	Linear	0.1225	0.1276	0.2592	**0.0710**
	Random	0.1217	0.1278	0.4752	**0.0630**
	Spike	0.4760	0.4742	1.0206	**0.0539**

Costs per 1000 requests in USD cents.

Outline

- ❑ Motivation
- ❑ Goals
- ❑ Background
- ❑ Migration Methodology
- ❑ Experimental Setup
- ❑ Results
- ❑ Conclusion and Future Work

Conclusion and Future Work

- Migration process is ad-hoc and time-consuming.
- GCR is relatively stable with cold start problems on traffc spikes.
- GKE in standard mode is best in terms of pure performance.
- How much should you decompose?

Future Work:

- Experiments with GKE-Autopilot.
- Fine-tuning serverless deployment for better comparisons.
- Migration of larger scale microservices applications.

Contact

Thank you for your attention!

Questions ?

mohak.chadha@tum.de

Code