Migrating from Microservices to Serverless: An IoT Platform Case Study - M. Chadha, V. Pacyna, A. Jindal, J. Gu. M. Gerndt

TCSS562 Cloud Computing Team 1:
Jeffrey Stockman, Rick Morrow, Mahmoud Ali Elkamhawy

BE BOUNDLESS

1

Overview

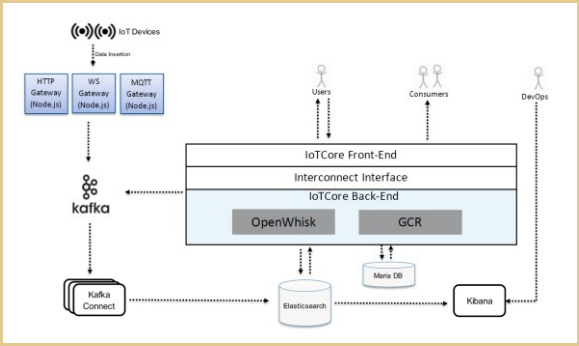
Migration of a complex microservice IoT platform onto Google Kubernetes Engine [GKE], OpenWhisk [OW] and Google Cloud Run [GCR] FaaS applications

- **Evaluation of performance and cost across varying load-testing scenarios**
- **Microservice architecture is the standard of cloud-based application deployment**
  - **Isolation, replaceability, and scalability of code base**
  - **Outsourced infrastructure, elasticity, fault tolerance**
  - **Well-defined interfaces**
  - **Challenges include: Complexity of deployment, provisioning of services, DevOps overhead**
- **Serverless computing enhances microservices and cloud deployments to include:**
  - **No infrastructure to manage**
  - **Pay-per-use billing**
  - **On-demand scaling (scale-to-zero)**

UNIVERSITY of WASHINGTON

2

IoT System Design



Components of IoT Model:

IoT Core: React Front-End; Node.Js backend

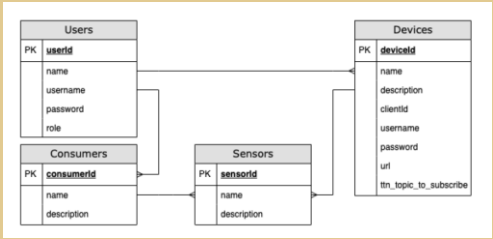
MariaDB: relational database

Kafka: publish-subscribe topic management

Kafka-Connect: streaming data service into ES, DBs

Elasticsearch: search & analytics engine via inverted index

Kibana: data visualization



Data Model:

Users: name, credentials; grants access to consumers

Devices: Users have devices with multiple sensors

Consumers: Retrieve sensor data; users can have multiple consumers

Sensors: single, multiple parameters; credentials

3

Investigation

- **Compare competing software architectures for an IoT platform:**
  - **OpenWhisk**
  - **Google Cloud Run**
- **Performance & cost evaluation against deployment strategies:**
  - **Google Kubernetes Engine [GKE] standard**
  - **OpenWhisk**
  - **Google Cloud Run**
- **Ease of migration from microservices to FaaS**

UNIVERSITY of WASHINGTON

4

Related Work

- Focus on performance comparison between microservices and serverless deployment strategies
  - AWS Lambda vs. Elastic Container Service. Fan et. al.
  - GKE, OW, GCP functions. Jindal et. al.
- Migration of stateful microservices onto OpenWhisk. Jin et. al.
  - Focus on minimizing code changes with equal performance

"Using application knowledge to Reduce Cold Starts in FaaS Service." D. Bermbach et. al.

UNIVERSITY of WASHINGTON

5

Considerations

- IoT is inherently challenged by:
  - Performance on the edge
  - Latency
    - HTTP has excess overhead in comparison to more efficient MQTT or CoAP
  - Security issues & associated overhead
- "Currently… no open FaaS platforms built specifically for IoT data processing at the edge."
- "Lean OpenWhisk… on a Raspberry Pi… is unable to successfully respond to 2/3 of all requests."
  - "TinyFaaS: A Lightweight FaaS platform for Edge Environments." T. Pfandzelter et al.

6

Migration Approach

- Migrating the IoT platform application
  - off-the-shelf software components, i.e.,Kafka, Kafka-connect, MariaDB, Elastic Search, and Kibana.
  - OpenWhisk (OW) and Google Cloud Run (GCR).
- Migrating IoTCore backend.
  - Focus on migrating the API endpoints.
  - Decompose the application logic for each API endpoint into a separate function.

UNIVERSITY of WASHINGTON

7

Migration of IoTCore Backend

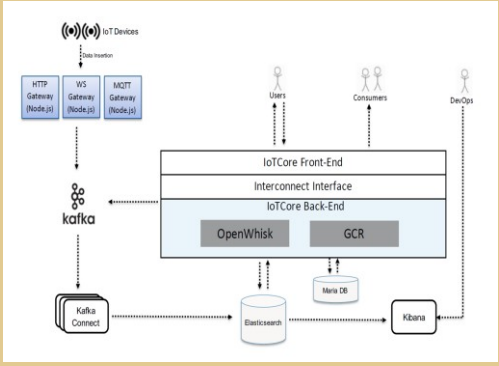
- Sensors-Get API study case
  - Authentication handler.
- IoTCore vs. individual functions
  - The runtimes contain only required packages in individual functions.
  - Docker images for the individual functions are smaller in size compared to IoTCore.
- OpenWhisk sequencing functionality
  - Utilizing sequencing functionality to chain the authentication function with middleware functions.
- Google Cloud Run migration
  - Sequencing function.
  - Adding application logic.

UNIVERSITY of WASHINGTON

8

Migration of IoTCore Backend (Cont.)

- Implementing an Interconnect interface
  - Intercepting the incoming request to API endpoint and forward it to OW or GCR function.
  - Responsible for forwarding the function response to the user.
- Migration of stateless microservices onto OpenWhisk
  - Migrating the HTTP gateway into separate OW and GCR functions



9

Key Contributions

- Migrating a microservices-based IoT platform application to OW and GCR.
- Evaluating the performance of the IoT platform across different deployment strategies i.e., Google Kubernetes Engine (GKE), OW, and GCR with different load testing scenarios.

UNIVERSITY of WASHINGTON

10

Experiment Design

- Off the shelf components of the tech stack couldn't be broken down into functions.
  - MariaDB (data storage)
  - ElasticSearch (metrics storage and search)
  - Kibana (data visualization)
  - Kafka (pub/sub)
- Tested three deployment strategies after migrating microservices to two serverless delivery methods.
  - Original microservices deployed on Google Kubernetes Engine (GKE).
  - OpenWhisk (OW) functions on GKE.
  - Functions deployed on Google Cloud Run (GCR) FaaS platform.

UNIVERSITY of WASHINGTON

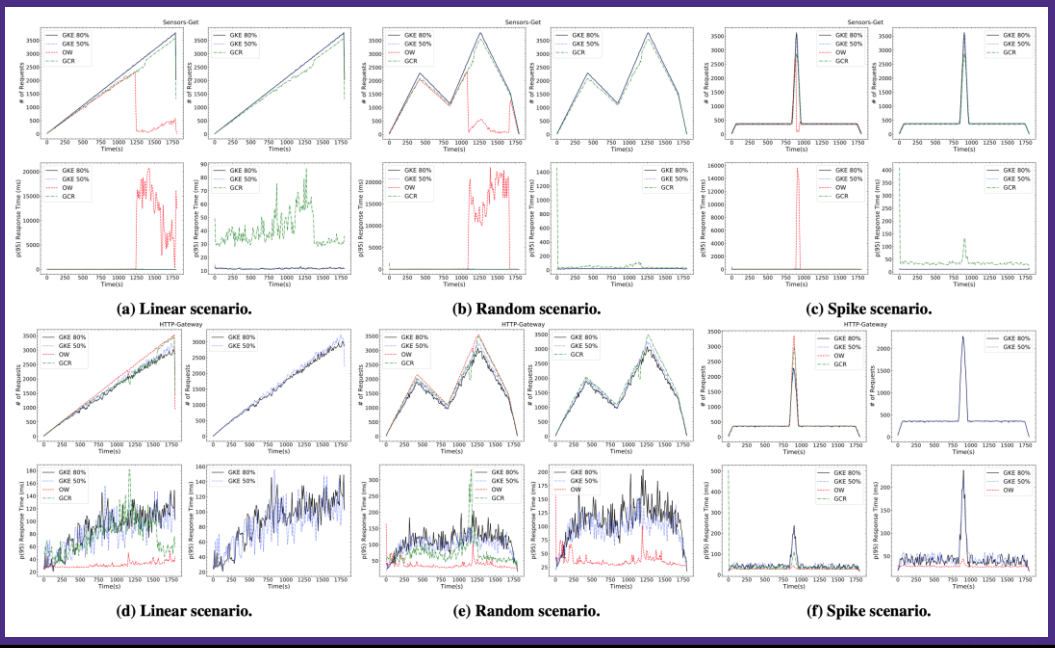
11

Experiment Methodology

- Used the "k6" performance and regression testing tool
  - Open source load test script
  - Virtual Users (VUs)
  - Quickly repeated HTTP(s) requests
- Deployment architectures tested with three workload scenarios
  - Linear: request volume increases over time as experiment runs
  - Random: random request volume random lengths of time
  - Spike: low volume request plateau for most of the experiment, with a spike to the highest volume of request in the middle.

12

Results



(a) Linear scenario. (b) Random scenario. (c) Spike scenario.

(d) Linear scenario. (e) Random scenario. (f) Spike scenario.

13

Author's Conclusions

- **Cost**
  - **Google Cloud Run is cheapest cost per 1000 requests in all scenarios**
  - **High loads would make fixed-cost structures cheaper**
- **Performance**
  - **FaaS and CaaS have high response times on bursts of requests**
  - **Microservices architecture is faster when there is action sequencing or function chaining**
  - **Migrating microservices to FaaS and CaaS is expensive**

API Endpoint	Load Testing Scenario	Deployment Strategies Cost (in USD).			
		GKE-50	GKE-80	OW	GCR
Sensors-Get	Linear	0.1054	0.1053	0.5195	**0.0542**
	Random	0.1043	1.0433	0.4752	**0.0544**
	Spike	0.4151	0.4153	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**
Users-Get	Linear	0.1103	0.1113	0.5401	**0.0493**
	Random	0.1098	0.1101	0.3221	**0.0478**
	Spike	0.4425	0.4368	1.3610	**0.0475**
Devices-Add	Linear	0.1210	0.1291	0.4734	**0.0617**
	Random	0.1178	0.1295	0.5020	**0.0569**
	Spike	0.4506	0.4630	1.2797	**0.0586**
Devices-Get	Linear	0.1081	0.1087	0.3677	**0.0517**
	Random	0.1073	0.1079	0.5151	**0.0521**
	Spike	0.4496	0.4346	1.3464	**0.0586**
Consumer-Consume-Get	Linear	0.1157	0.1162	0.2660	**0.0636**
	Random	0.1135	0.1141	0.2636	**0.0624**
	Spike	0.4432	0.4439	1.0527	**0.0657**

API Endpoint	Deployment Strategy	Load Testing Scenarios								
		Linear			Random			Spike		
		#Requests	Average (ms)	p(95) (ms)	#Requests	Average (ms)	p(95) (ms)	#Requests	Average (ms)	p(95) (ms)
Sensors-Get	GKE-50	343,158	11.47	19.72	346,864	**11.18**	**19.54**	87,132	**11.24**	**19.59**
	GKE-80	343,343	**11.35**	**19.63**	346,620	11.39	19.65	87,085	11.38	19.76
	OW	161,053	310.08	94.15	176,065	267.47	86.75	68,409	88.80	65.11
	GCR	319,046	31.29	53.74	321,963	31.45	52.86	79,628	36.05	69.73
HTTP-Gateway	GKE-50	295,154	54.16	113.41	297,088	55.11	109.61	75,969	49.87	148.34
	GKE-80	283,433	66.82	123.64	283,071	70.29	130.61	76,256	48.76	152.55
	OW	322,721	**28.02**	**39.33**	326,900	**27.15**	**37.63**	81,971	**27.74**	**33.61**
	GCR	299,420	49.73	108.39	311,514	40.88	88.38	81,029	30.98	72.45
Users-Get	GKE-50	327,921	**23.70**	**36.50**	329,417	25.09	42.25	81,725	28.69	62.50
	GKE-80	324,931	26.20	47.90	328,549	25.82	44.07	82,791	25.09	43.56
	OW	154,893	332.29	145.88	259,701	590.08	2290	61,472	127.90	52.44
	GCR	325,239	25.94	38.36	330,382	**24.28**	**35.58**	83,120	**23.94**	**32.75**
Devices-Add	GKE-50	298,950	50.29	93.35	307,056	45.20	76.89	80,252	**33.83**	89.10
	GKE-80	280,226	70.41	125.90	279,296	74.61	135.03	78,108	41.68	124.49
	OW	176,729	261.33	**66.40**	166,670	296.90	78.76	65,377	105.43	**52.15**
	GCR	310,090	**39.45**	74.11	318,868	**34.20**	**62.10**	79,616	36.05	72.29
Devices-Get	GKE-50	334,605	**18.18**	**26.35**	337,044	**18.81**	**24.49**	80,429	33.17	131.56
	GKE-80	332,665	19.75	29.13	339,255	20.26	28.40	83,207	**23.64**	62.57
	OW	227,558	146.35	76.99	162,407	310.86	101.04	62,137	125.25	**52.14**
	GCR	322,234	28.48	39.29	324,856	28.93	37.55	79,616	36.05	72.29
Consumer-Consume-Get	GKE-50	312,530	37.18	83.18	318,553	**34.48**	72.17	81,593	**29.07**	61.77
	GKE-80	311,265	38.37	86.08	316,838	36.02	77.74	81,478	29.45	63.27
	OW	314,545	**35.31**	**52.23**	317,326	35.57	**51.52**	79,473	36.53	**45.68**
	GCR	307,828	41.56	86.52	312,235	40.23	81.38	77,493	43.85	115.31

14

Strengths

- One of the few papers that assesses cost vs performance, as cloud pricing is
  1. variable across providers, and
  2. should be part of any cloud-optimization case study
- HTTP data transfer has more overhead than MQTT or CoAP. It would be interesting to see how these protocols fared using this model.

15

Weaknesses

- Aside from cold starts, the model didn't factor in 1) latency or 2) edge computing
- The model incorporated some VMs with abnormally high memory and processor speed, whereas most IoT devices are relatively much smaller and less powerful
- Not using function chaining in the Google Cloud Run deployment misses an opportunity to compare managed FaaS infrastructure against self-deployed infrastructure

UNIVERSITY of WASHINGTON

16

Evalutation

header

- Slide 18 Critique: Evaluation
- How good is the paper's evaluation? Is something missing? Are the results believable? Is enough information available to repeat/reproduce tests? Are there problems with the graphs or the discussion? Is the analysis complete, or are some points left for the Page 5 of 7 reader to try and understand on their own?

UNIVERSITY of WASHINGTON

17

Gaps

- **Future work potential:**
  - **Compare other microservice application migration to the cloud using the same model to see if there is a correlation with this study's results**
- Slide 19 Identify Gaps ONE SLIDE ONLY
- Are there gaps and open problems remaining in the research? Did the authors fail to solve some aspect of the problem? What constraints and limitations exist for the solution? What future work remains?

18

Questions?

Thank you

- **Probably don't need anything here.**

UNIVERSITY of WASHINGTON

19