UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Runtimes for Concurrency and Distribution

Project: Exploration of orchestration challenges in a microservice-based application

Luca Marchiori

June 2024

Outline

1. Report corrections
2. Take-home message
3. Learning outcome
4. CFU evaluation
5. Feedback on the course
6. More

Missing notions

After reading the comments, I realized that I missed some notions in the report:

- Theoretical definitions (Microservices, REST, ...)
- Tool descriptions (Swagger, OpenAPI, RabbitMQ, ...)
- Components behavior (Nodeport, Ingress, ...)

In the earliest version of the report I included these sections, but I had to cut them to respect the 10 pages limit.

As a technical report, I focused more on explaining the project itself and the implementation details.

My responses to the "B" comments

The HPA problem

The Horizontal Pod Autoscaler (HPA) was not working as expected: Pods were not scaling up when multiple requests were sent to the service.

My mistake: I stopped looking for a solution after the first assumption that made sense to me.

New debugging session:

```
lucam@pop-os ~ kubectl get hpa
NAME                    REFERENCE                          TARGETS              MINPODS   MAXPODS   REPLICAS   AGE
users-ms-deployment     Deployment/users-ms-deployment     cpu: <unknown>/50%   1         10        1          10d
lucam@pop-os ~
```

Something seems to be wrong with the metrics-server. The HPA is not able to retrieve the metrics from the metrics-server.

The HPA problem

Checking with the HPA logs:

```
horizontal-pod-autoscaler: failed to get cpu utilization: unable to get
    metrics for resource cpu: no metrics returned from resource metrics API
horizontal-pod-autoscaler: invalid metrics (1 invalid out of 1), first error
    is: failed to get cpu resource metric value: failed to get cpu utilization:
     unable to get metrics for resource cpu: no metrics returned from resource
    metrics API
```

Problem identified: the metrics-server is not able to retrieve the metrics from the pods.

The HPA problem

After some debugging sessions, I found the problems:

- Minikube needs to be started with "–extra-config=kubelet.housekeeping-interval=10s" to avoid caching issues with the metrics-server.
- Missing configuration in the users-ms deployment for specifying the resource requests and limits.

Furthermore, instead of deploying the metrics-server from command line, I moved its deployment to a separate file to better manage the configuration.

The HPA problem


```
NAME           REFERENCE                          TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
users-ms-hpa   Deployment/users-ms-deployment     cpu: 5%/50%   1         10        1          22m
→  k8s git:(main)
```

After starting the load test again, the HPA was able to scale up the pods as expected.

```
→  k8s git:(main) ✗ kubectl get hpa
NAME           REFERENCE                          TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
users-ms-hpa   Deployment/users-ms-deployment     cpu: 98%/50%  1         10        10         34m
→  k8s git:(main) ✗
```

Minikube limitation: no matter the number of pods, the host CPU is the same, so the performance is limited.

Updates

Microservices updates are easy or not? In the report i begin by saying that microservices updates are hard but then I show that they are pretty easy.

- Without orchestration, updates are very challenging.
- With Kubernetes are more manageable but still require proper planning.

Updates

The "latest" tag leads to:

- Unpredictability
- Inconsistent updates
- Difficult in debugging and replication

Instead, use tags consistent with a version control strategy to ensure stability and traceability.

This approach allows for precise control over which image version is deployed, facilitating easier rollbacks and consistent environments across development, testing, and production stages.

My response to the "B" comments

The focus of the presentation is on:

- Why virtualization, abstraction and orchestration
- How abstraction works in containerization
- The notion of business domain separation
- The separation between connection state and application state
- The problems encountered with the Horizontal Pod Autoscaler

Take-home message

Microservices are much more complex than traditional monolithic applications. They require an experienced architectural view because even if tools are available to simplify the orchestration, the complexity of the system requires a deep understanding of the underlying concepts of distributed systems.

Learning outcome

These are the key concepts I learned both from the course and the project:

- Tools and technologies are worthless without a clear understanding of the underlying concepts.
- Abstraction and virtualization are much more important than I thought.
- Microservices are interesting but the choice of implementing them should be driven by actual needs, not just because they are trendy.

CFU evaluation

6 CFU equals to 150 hours of work.
How I spent my time:

- Lessons: $\sim$ 48 hours
- Study: $\sim$ 50 hours
- Project: 10 weeks, $\sim$ 6/8 hours per week (60 hours)

Good balance between lessons and project, consistent with the CFU assigned.

Feedback on the course

Good points

- Very interesting topics that connect basic C.S. concepts with more advanced and real-world applications.
- I enjoyed the interactions between the students and the professor, especially for flipped classroom lessons.
- It would have been interesting to have more discussion sessions.

Improvements

- I felt a little lost during the project development. It is hard to start from scratch without a clear idea of what are the expectations.

Missing definitions

Extra slides for missing definitions:

- Microservices definitions
- REST architectural style
- What are Swagger and OpenAPI
- Why choosing GO as programming language
- The choice of implementing common and basic microservices

Sources of Information

- **Official Docker documentation website**
  - `https://docs.docker.com/`
- **Official Kubernetes documentation website**
  - `https://kubernetes.io/docs/`
- **Introduction to Microservices**
  - `https://web.archive.org/web/20240202154248/https://www.nginx.com/blog/introduction-to-microservices/`
- **Building Microservices with Go** - Nic Jackson
- **Building Microservices: Designing Fine-Grained Systems** - Sam Newman
- **Microservices** - Martin Fowler
  - `https://web.archive.org/web/20180214171522/https://martinfowler.com/articles/microservices.html`

Thank you for your attention!

Microservices definitions

"Microservices are **independently** releasable services that are modeled around a **business domain**. A service encapsulates functionality and makes it accessible to other services via **networks**" [1]

"A microservice is an architectural pattern that arranges an application as a collection of **loosely coupled**, **fine-grained services**, communicating through lightweight protocols" [2]

Back to index

[1] Sam Newman, Building Microservices

[2] Martin Fowler, Microservices

REST architectural style

REST (Representational State Transfer) is a web architecture style using standard HTTP methods (GET, POST, PUT, DELETE) to interact with resources via URLs.

- POST: /users (create a new user)
- GET: /users/id (get a user by id)
- GET: /users (get all users)
- PUT: /users/id (update a user by id)
- DELETE: /users/id (delete a user by id)

Back to index

Why GO as programming language

Microservices are language-agnostic, so they can be written in any language as long as they support necessary communication protocols.

It is also possible to have different microservices written in different languages as long as they can communicate with each other.

GO was chosen for the project because it has native support for web server making development easier and faster. [3]

Back to index

[3] Thanks to the concurrency features of the language, and the native support for web standards, both Docker and Kubernetes are written in GO.

Common and basic microservices

Why implementing common and basic microservices such as users, auth, and notifications?

- The goal was to learn about orchestration, not to develop impressive features.
- I didn't want to lose time by thinking of exceptional use cases.
- Those are component that I could find in any future project.

Back to index

What are Swagger and OpenAPI

Swagger

Swagger is a set of open-source tools built around the OpenAPI Specification that can help in design, build, document and consume REST APIs.

OpenAPI

OpenAPI Specification is a specification for building APIs. It is a language-agnostic specification that describes the structure of REST APIs.

In the project i used OpenAPI to define the API of the application and Swagger to have a GUI for reading the API documentation and test the endpoints.

Back to index