Nexthink

Technical Assessment

François Ferrari

15 december 2020

Table of Contents

1 Service deployment

The goal of this technical assessment is to handle the deployment of microservices, handling the dependencies to each other, and allowing to check that the hierarchy of microservices is healthy.

Two endpoints should be exposed :

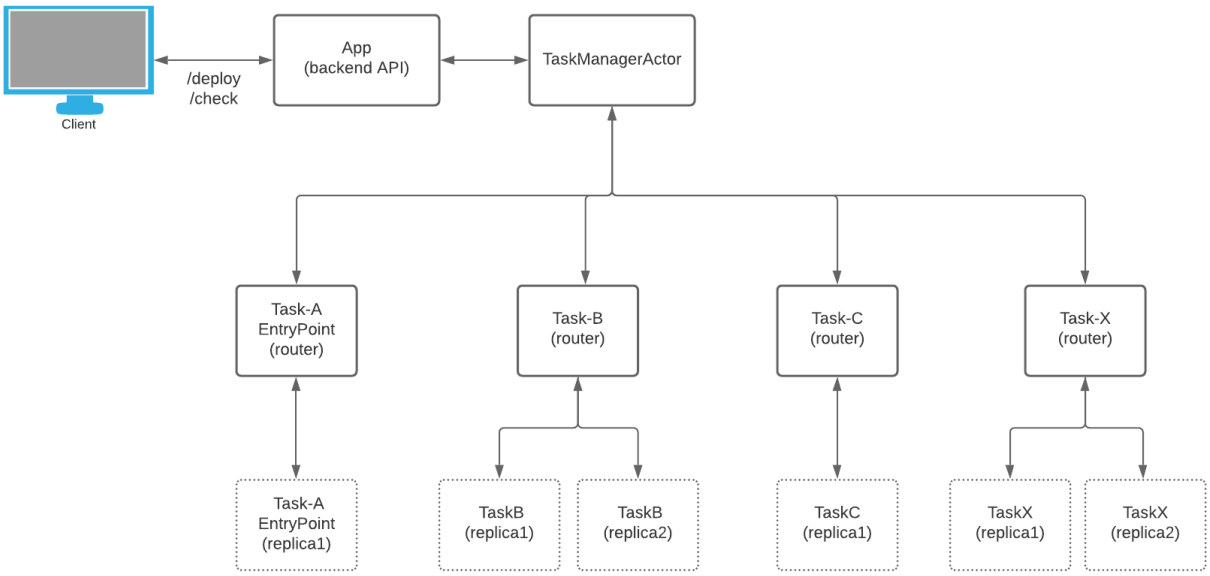
- /deploy that will allow to deploy the microservices based on a service deployment specification provided as a json document
- /check that will allow to check that the hierarchy of microservices is healthy

The technical stack we will use is the following :

- Scala as it is a requirement from the technical assessment description
- Akka Typed to handle the process of creation of the Microservices, that we name Tasks. This is a good choice because this framework allows to handle the creation of replicas for each Task and this is a requirement
- Akka Http to handle the Http requests, it fits perfectly with Akka Typed of course, and it is easily testable
- scalax.collection.Graph will be used to handle the hierarchy of Tasks and will allow us to :
    - ensure that the Graph is acyclic
    - handle the health check requests for which we have to check for each Task that their dependencies (Tasks) are healthy

1.1 Architecture

The general architecture of the solution is presented below, the components are the following :



The components are the following :

- Client ; it sends /deploy and /check requests to the App Backend
- App Backend ; it receives the HTTP requests from the Client, and handles it using Akka HTTP
- Task Manager Actor, it has the following responsibilities :
  - creation of the Tasks when the /deploy requests is sent by the Client
  - checking if the Tasks are healthy and returning the status when the /check requests is sent by the Client
- Tasks routers :
  - they create the Task actors for each Task and with as many replicas as needed
  - they also handle the routing mechanism of the messages they receive to the Task replicas they have created

1.2 API

1.2.1 /deploy

Topic	**Description**
Role	To deploy a service based on a description provided as a JSON payload
Method	POST
Payload	A JSON payload is mandatory for the endpoint to be accepted
StatusCodes	OK
	InternalServerError

Here is an example of a JSON payload, that can be used with Postman for example :

```
[
  {
    "serviceName": "A",
    "entryPoint": true,
    "replicas": 1,
    "dependencies": ["B", "C", "E"]
  },
  {
    "serviceName": "B",
    "entryPoint": false,
    "replicas": 1,
    "dependencies": ["C", "E"]
  },
  {
    "serviceName": "C",
    "entryPoint": false,
    "replicas": 2,
    "dependencies": ["D"]
  },
  {
    "serviceName": "D",
    "entryPoint": false,
    "replicas": 2,
    "dependencies": []
  },
  {
    "serviceName": "E",
    "entryPoint": false,
    "replicas": 2,
    "dependencies": []
  }
]
```

1.2.2 /check

Topic	**Description**		
Role	To check if the Tasks are healthy, from the Root Task down to the dependencies		
Method	GET		
Payload	No payload is required here		
StatusCodes	OK	{ "status": "HEALTHY" }	All Tasks are healthy
	InternalServerError	{ "status": "UNHEALTHY" }	At least one Task is unhealthy
	InternalServerError	{ "status": "UNKNOWN" }	Unexpected server error
	BadRequest	{ "status": "NOT_DEPLOYED" }	The services are not deployed

2 Improvements

- The /deploy endpoint should be improved, at the moment when a /deploy is issued after a previous deploy, the previous Tasks are not stopped. There is a branch in the repo in which a cleanUpAndWatch method exists for this, but it is not finished/integrated yet (not part of the requirements)
- The actors representing the Tasks are the children of a unique actor, the TaskManagerActor. A better hierarchy of actors with a parent/children/dependencies may be useful.