**Technologie internetu przyszłości**

**Migracja obliczeń między klastrami OpenFaas**

Bartosz Kucharz, Ivan Smaliakou, Wojtek Szmelich

AGH, Kraków

2024

1. **WSTĘP**

Ten projekt ma na celu wykonać migrację obliczeń między klastrami Kubernetes ze środowiskiem OpenFaas. Dla realizacji tego projektu zaimplementowaliśmy prosty lokalny serwer, mający dwa punkta końcowych: /1 oraz /2. Wysyłając żądanie na */1* najpierw wdrażamy program na klaster pierwszy klaster, ewentualnie usuwamy funkcje na drugim klastrze oraz wywołamy funkcje. Kiedy te klastry zmieniają się, robimy to samo. Każde kolejne żądanie na ten sam klaster tylko wywołuje tę funkcję (czyli jeśli ciąg żądań ma kształt {/1, /1, /1, /1 ...} albo {/1, /2, /2, /2 ...}, to wtedy tylko pierwsze żądanie na ten klaster będzie zawierało wdrożenie oraz ewentualne usunięcie z innego klastra).

Sama funkcja, umieszczona na obu klastrach, robi nic innego jak zwraca IP adres tego węzła, na którym została uruchomiona.

**2. URUCHOMIENIE**

Dla uruchomienia serwera prozę wykonać następne kroki:

1. Proszę stworzyć dwa klastera kubernetes o nazwach cluster1 oraz cluster2 (tylko tak) w środowisku AWS EKS oraz dodać do nich tzw. Node grupy;
2. Proszę ustalić środowisko tak, aby do tych dwóch klastrów można było mieć dostęp poprzez kubectl;

2. Proszę ustalić środowisko tak, aby do tych dwóch klastrów można było mieć dostęp poprzez kubectl;

3. Proszę pobrać ten repozytorium: <https://github.com/iivansmaliakou/tip-openfaas-migration>

4. Uruchomić po koleje: cd tip-openfaas-migration/, chmod 777 ./setup\_cluster.sh;

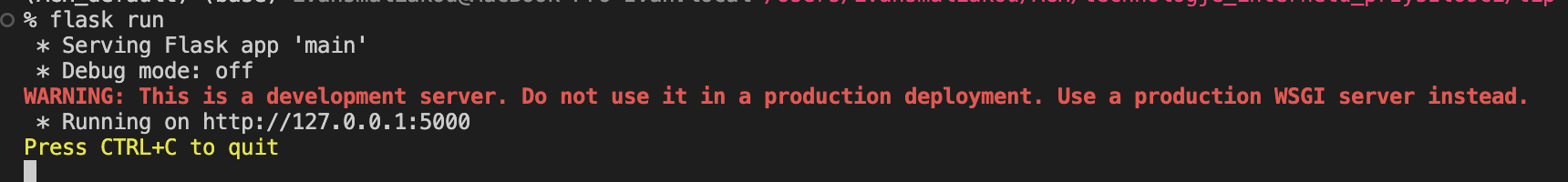
6. Mając ustalony kubectl context na cluster1, proszę uruchomić source ./setup\_cluster.sh 1

7. Zmienić kubectl context na cluster2, a po tym proszę uruchomić source ./setup\_cluster.sh 2;

8. Proszę zainstalować (globalnie albo po utworzeniu venv) Flask: pip install Flask;

9. Proszę ustalić zmienne: export FLASK\_APP=main oraz export FLASK\_ENV=development;

10. I teraz proszę uruchomić: flask run

Teraz można wysyłać żądania. W przypadku serwera, ustalonego domyślne jak na Rys. 1, będzie to <http://127.0.0.1:5000/1> albo <http://127.0.0.1:5000/2>.

Rysunek 1. Logi serwera Flask

**2. TESTOWANIE**

Aby przetestować program, proszę uruchomić następujące kroki:

1. Proszę zainstalować wrk2: <https://github.com/giltene/wrk2>.
2. Aby przetestować prędkość działania w trybie, gdy zawsze wysyłamy żądanie do tego samego klastra, proszę uruchomić z terminalu najpierw curl -XGET 127.0.0.1:5000/1 , a po tym wrk -t30 -c60 -d30s http://localhost:5000/1.
3. Aby przetestować prędkość działania w trybie, gdy każdy kolejny raz zmieniamy klaster, na którym funkcja jest umieszczona, proszę uruchomić prosty skrypt:

import time

import requests

ITER\_COUNT = 100

start\_time = time.time()

for i in range(ITER\_COUNT):

requests.get(f'http://127.0.0.1:5000/{i%2+1}')

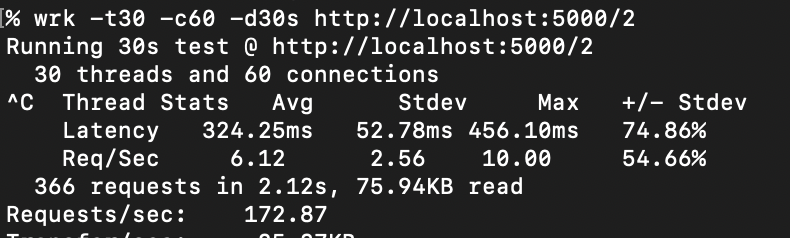
end\_time = time.time()

print(f'Average execution time is: {(end\_time-start\_time) / ITER\_COUNT} seconds')

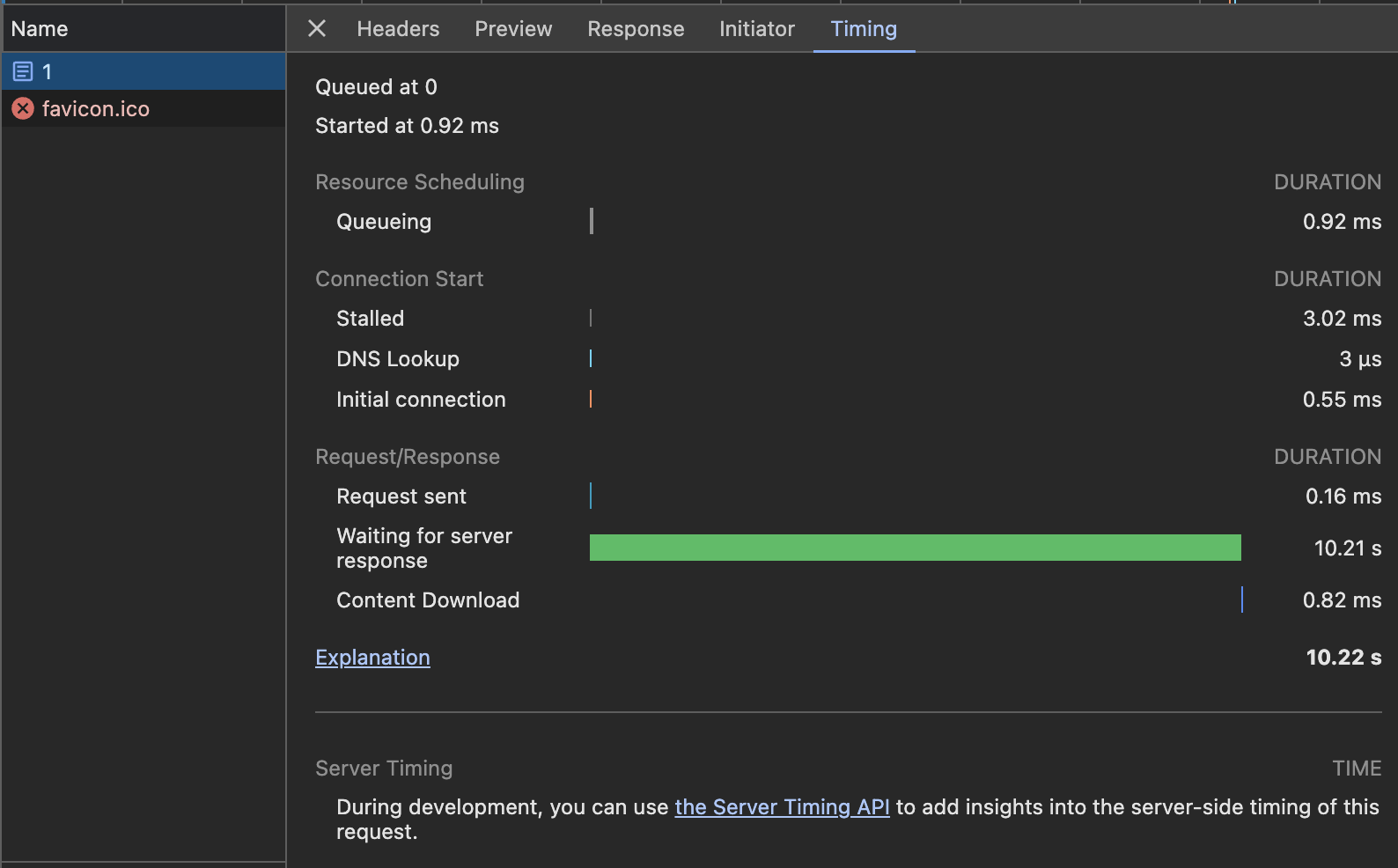
**3. WYNIKI**

Poniżej zostały przedstawione wyniki pomiarów wydajności. Na Rys. 2 został przedstawiony wynik z kolejnymi wywołaniami tęgo samego punktu końcowego (czyli tego samego klastra).

Na Rys. 3 natomiast został przedstawiony inny wykres ze średnim wynikiem wywołania poprzez panel dewelopera Chrome.

****

Rysunek 2. Wyniki benchmarku wrk2 dla tego samego punktu końcowego

Rysunek 3. Średni wynik benchmarku Chrome dla różnych punktów końcowych