Проектування програмного забезпечення

Лабораторна робота №4

Тема: Горизонтальне масштабування програмних систем
Мета: Поглиблення розуміння задач та принципів реалізації балансувальників навантаження, закріплення навичок імплементації інтеграційних тестів

Завдання на дану роботу складається з 3-х частин:

1. Реалізація алгоритма балансування навантаження відповідно до вашого варіанта та покриття його юніт-тестами.

2. Реалізація інтеграційних тестів для балансувальника навантаження.

3. Налаштування процесу безперервної інтеграції із запуском інтеграційних тестів.

Перед виконанням завдання підготуйте оточення на своїй робочій машині.

- Встановіть Docker (посилання для Windows/MacOS та Linux)
- Встановіть docker-compose
- Склонуйте репозиторій-шаблон
  git clone git@github.com:roman-mazur/architecture-practice-4-template.git
  Репозиторій має наступну структуру:

```
$ tree
.
├── Dockerfile      # Опис образа контейнера для сервера та балансувальника
├── Dockerfile.test # Опис образа контейнера, який запускає інтеграційні тести
├── cmd
│   ├── client      # Клієнт для ручного тестування балансувальника
│   │   └── client.go
│   ├── lb          # Реалізація балансувальника (основна робота завдання 1)
│   │   ├── balancer.go
│   │   └── balancer_test.go
│   ├── server      # Реалізація бекенда, який знаходиться за балансувальником
│   │   ├── report.go
│   │   ├── report_test.go
│   │   └── server.go
│   └── stats       # Допоміжна команда, яка збирає статитистику серверів
│       └── main.go
├── docker-compose.test.yaml  # compose-файл для інтеграційного тестування
├── docker-compose.yaml       # compose-файл з основними сервісами
├── entry.sh                  # точка входу для контейнера
├── go.mod
├── httptools
├── integration               # Пакет з інтеграційними тестами
│   └── balancer_test.go
└── signal
```

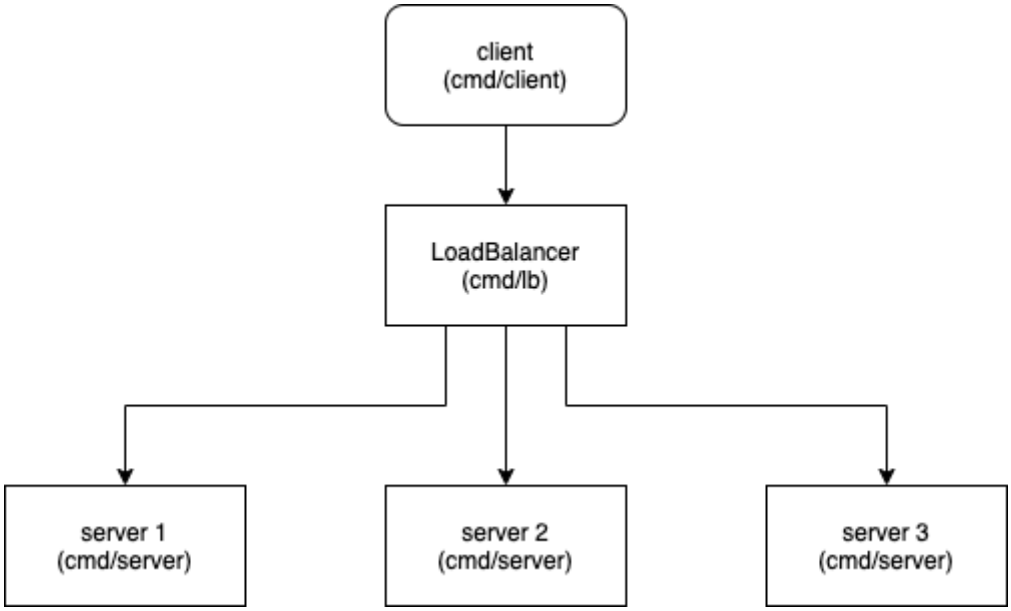
- Перевірте, що ви можете збирати та працювати з ресурсами в репозиторії:
  - Запустіть
    `docker build -t practice-4 .`
    У результаті, ви повинні побачити повідомлення про успішну збірку.
    Подальший запуск
    `docker run --rm practice-4`

```
має завершитися запуском HTTP сервера
2023/05/06 19:58:14 Staring the HTTP server…
```

- Запустіть
  `docker-compose up`
  У результаті, ви повинні побачити логи запущених серверів та балансвальника. Поки вони працюють, спробуйте запустити cmd/client: його логи мають повідомляють про успішну відповідь, а логи балансувальника та одного з серверів - про обробку запита від клієнта.

Завдання 1. Реалізація алгоритма балансування навантаження

Репозиторій, який ви склонували, має базові реалізації всіх компонентів системи з горизонтальним масштабуванням на основі балансувальника навантаження. На рисунку нижче зображено, як пакети репозиторія відповідають даним компонентам.



Реалізація балансувальника в репозиторії відправляє всі запити (за допомогою функції `forward`) на перший сервер зі списку доступних. На одному із занять ви реалізуєте алгоритм Round Robin, а в даній роботі вам доведеться імплементувати один зі складніших алгоритмів, які описано в таблиці нижче.

Алгоритм	Опис
Хеш адреси клієнта	Балансувальник визначає сервер шляхом обчислення хеш-суми адреси клієнта. Цей підхід добре працює, коли необхідно, щоб один і той самий клієнт обслуговувався тим самим сервером. `serverIndex = hash(req.RemoteAddr) % len(serversPool)`
Хеш шляху URL	Підхід аналогічний попередньому, але хеш обчислюється для шляху з URL, який ідентифікує контент. Цей алгоритм оптимізує кешування контента: усі клієнти, який роблять запит за одним URL обслуговуються одним сервером, що дозволяє мінімізувати дублювання кешованого контента між серверами. `serverIndex = hash(req.URL.Path) % len(serversPool)`
Найменша кількість з'єднань	Балансувальник підтримує лічильник поточних з'єднань для кожного з серверів та обирає сервер з мінімальною кількістю з'єднань (запитів, які той обробляє в даний момент). `serverIndex = min(serversPool, func(a, b) { a.connCnt < b.connCnt })`
Найменший об'єм трафіка	Підхід аналогічний попередньому, але замість кількості з'єднань, підтримується сума кількості байт, які сервер віддавав у своїх відповідях

Реалізація балансувальника повинна використувати один з алгоритмів наведених вище (відповідно до варіанта) для вибору сервера зі списку "здорових" серверів. Сервер називається "здоровим", якщо він успішно відповідає на HTTP запит /health (для визначення "здоров'я" сервера ви можете користуватися вже імплементованою функцією `health`). Балансувальник підтримує список "здорових" серверів за рахунок проактивного опитування усіх наявних серверів.

Верифікувати роботу свого алгоритма ви можете

- в ручному режимі, запустивши балансувальник за допомогою `docker-compose up` та вручну відправляючи запити на балансувальник або запускаючи `cmd/client;`
- автоматично, покривши алгоритм балансування юніт-тестами. Наявність таких тестів я важливим фактором при оцінці першого завдання. Для їхньої реалізації доцільно скористатися підходом заглушок.

Під час виконання даного завдання вам потрібно буде модифікувати код в пакеті cmd/lb.

Варіанти

№	Алгоритм	Пакет для перевірок в тестах
1	Хеш адреси клієнта	testify
2	Хеш шляху URL	testify
3	Найменша кількість з'єднань	testify
4	Найменший об'єм трафіка	testify
5	Хеш адреси клієнта	gocheck
6	Хеш шляху URL	gocheck

№	Алгоритм	Пакет для перевірок в тестах
7	Найменша кількість з’єднань	gocheck
8	Найменший об’єм трафіка	gocheck

Свій варіант, як завжди, ви можете визначити за допомогою команди `/team-variant 4` у Slack.

Завдання 2. Інтеграційні тести

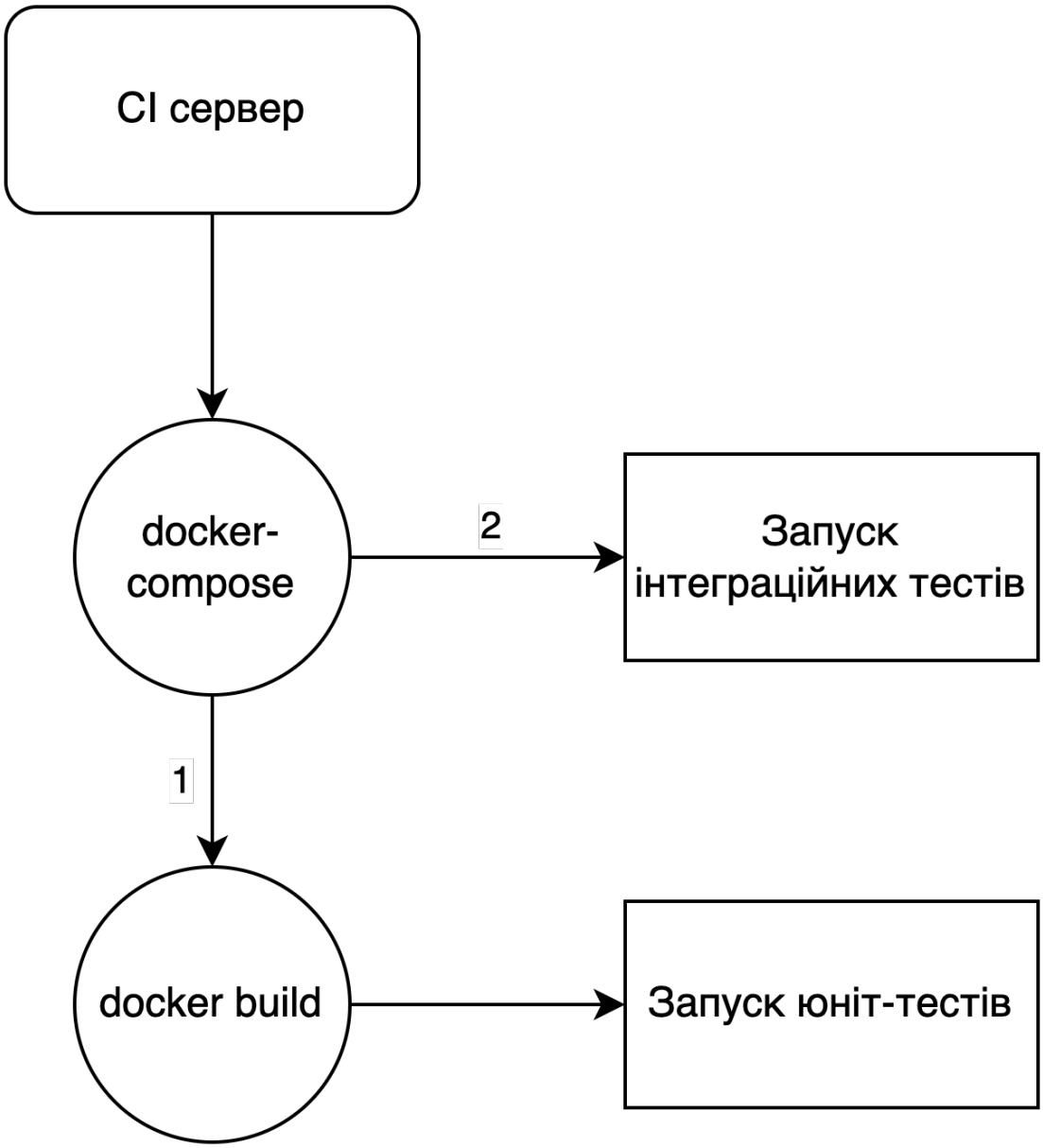
У юніт-тестах агоритма балансування ви маєте максимально повно проверифікувати аспекти його логіки. Завданням інтеграційного теста є перевірка того, що вже зібраний компонент (а саме балансувальник) може успішно бути запущеним та використовує потрібний алгоритм.
Головним завданням для вашого інтеграційного теста у даній роботі є відправка певної кількості запитів на балансувальник та перевірка, що вони були доставлені на різні сервера. З даною метою, балансувальник запускається у режимі, в якому він додає інформацію про те, від якого сервера прийшла відповідь у вигляді додаткового заголовка (“lb-from”). Ви можете перевірити, що множина значень, які приходять у даному заголовку відповідає вашим очікуванням.
Для даного завдання нам потрібно, на додачу до запуску серверів та балансувальника нам потрібно також запустити тести. Для цього docker-compose.test.yaml доповнює визначення сервісів тестовим контейнером та змінює параметри запуска балансувальника.
Запускати інтеграційний тест можливо за допомогою команди

```
docker-compose -f docker-compose.yaml -f docker-compose.test.yaml \
    up --exit-code-from test
```

Дана команда вказує, що визначення сервісів потрібно доповнити додатковим файлом (.test.yaml) та дочекатися завершення виконання сервіса *test*. Образ контейнера test будується за допомогою окремого docker-файла (*Dockerfile.test*).

Завдання 3. Безперервна інтеграція

Запуск збірки та інтеграційних тестів вам потрібно організувати через Github Actions. CI сервер має запускати docker-compose точно так, як ви це робили вручну у 2-гому завданні. docker-compose буде запускати збірку образів відповідних контейнерів, ті, в свою чергу запускатимуть go build та go test, який збиратиме бінарники компонентів та запускатиме юніт-тести (див рисунок нижче).



Оцінювання

Для оцінки роботи, відправте її через команду у Slack звичним чином:
`/submit 4 <github url>`

Максимальный бал за виконання роботи – 10 (5 балів за перше завдання, 3 бала за друге та 2 бала за останнє).