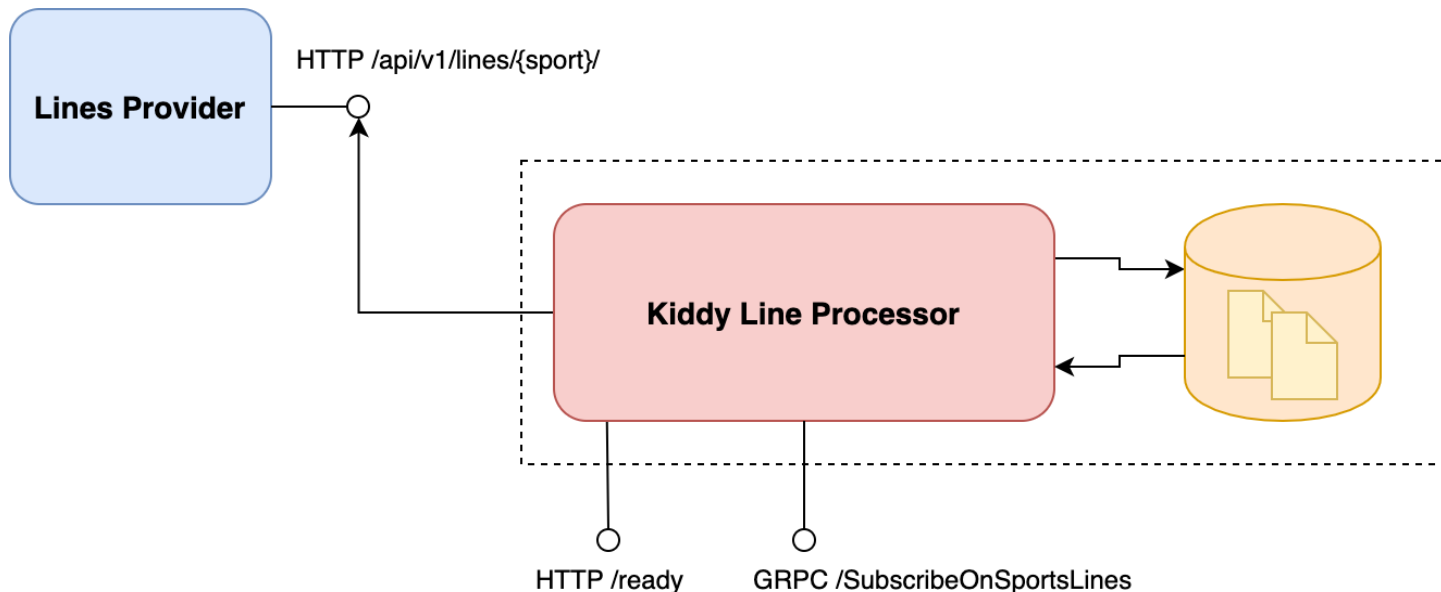


Тестовое задание

Описание

Общая схема



Компоненты

1) Lines Provider

Представлен в виде публичного образа **antonboom/lines-provider**

```
$ docker run -p8000:8000 antonboom/lines-provider
2020/07/24 08:44:55 listen and serve at :8000
```

```
$ curl http://localhost:8000/api/v1/lines/baseball
{"lines":{"BASEBALL":"0.774"}}
```

```
$ curl http://localhost:8000/api/v1/lines/football
{"lines":{"FOOTBALL":"0.721"}}
```

```
$ curl http://localhost:8000/api/v1/lines/soccer
{"lines":{"SOCCER":"1.667"}}
```

Как видно, позволяет получать некие коэффициенты (линии) для трёх видов спорта: бейсбола, американского футбола и футбола.

2) Kiddy Line Processor

Сервис, который необходимо реализовать на любом из трёх языков (в порядке убывания приоритета): Golang, C++, Python.

Алгоритм работы сервиса:

- На старте приложение запускает воркеры для каждого спорта, работающие в "фоновом" режиме и выполняющие следующие действия:
 - Раз в N секунд сходить в **Lines Provider** за линией для данного спорта (спуллить её).
 - Сохранить значение линии в хранилище.
 - Интервал N может быть у каждого спорта свой.
- Приложение должно предоставлять два вида API:
 - **HTTP**, с единственной ручкой **/ready**. Ручка проверяет соединение к хранилищу, а также говорит о том, что прошла первая синхронизация линий. Отвечает 200 OK или не-200 статусом. При желании может возвращать JSON с более детальной информацией о состоянии приложения.
 - **gRPC**, с единственным методом **/SubscribeOnSportsLines**. Метод отвечает за подписку на линии для указанных видов спорта. Является полнодуплексным (*bidirectional streaming RPC*).
В запросе - список видов спорта и с какой периодичностью получать линии, в первом ответе - линии по ним, в последующих - дельты от того, что уже было отправлено.
Если последующие запросы в стриме содержат изменение списка видов спорта, то снова присылаем линии полностью.
Линии берутся из хранилища, обращений к **Lines Provider** быть не должно.
Пример на псевдоязыке:

```
Full-duplex stream:
10:00:00 > /SubscribeOnSportsLines [soccer, football] 3s
10:00:00 < {soccer: 1.13, football: 2.19}
10:00:03 < {soccer: 0.03, football: -0.01}

10:00:07 > /SubscribeOnSportsLines [soccer, football] 1s
10:00:07 < {soccer: 0.01, football: 0.02}
10:00:08 < {soccer: 0.02, football: -0.20}
10:00:09 < {soccer: -0.13, football: 0.03}

10:00:09 > /SubscribeOnSportsLines [baseball, football] 5s
10:00:09 < {baseball: 2.40, football: 1.98}
10:00:14 < {baseball: 0.12, football: 0.01}
10:00:19 < {baseball: -0.11, football: 0.03}
```

Protobuf-спецификации запроса и ответов, а также их формат необходимо разработать самостоятельно.

- HTTP API доступно сразу, но gRPC с того момента, как прошла первая синхронизация сторэжей (/ready стала возвращать 200 OK).

- Минимально необходимый список параметров конфигурации приложения:
 - Адрес HTTP сервера.
 - Адрес GRPC сервера.
 - Интервал N синхронизации хранилища с **Lines Provider** для каждого из трёх спортивных.
 - Уровень логирования.

3) Хранилище для Kiddy Line Processor

Хранилище может быть как нереляционного типа (например, Redis), так и реляционного (например, PostgreSQL), на выбор разработчика.

Каждый спорт должен быть представлен собственным ключом / таблицей (в зависимости от типа хранилища).

Инфраструктура

Необходимо:

- Реализовать Dockerfile для приложения и docker-compose файл с приложением и необходимым ему окружением (Lines Provider, хранилище, etc.).
- Реализовать Makefile с базовым набором команд:
 - **make lint** - запускает статические анализаторы кода;
 - **make tests** - запускает юнит-тесты;
 - **make run** - запускает docker compose, приложение начинает работать без дополнительных действий, все необходимые порты доступны с хостовой машины.
 - **make stop** - останавливает работу сервисов.

Формат сдачи задания

Задание принимается в виде ссылки на публичный SCV-совместимый репозиторий (Bitbucket, GitLab, GitHub, etc.), содержащий исходный код проекта.

Необходимо настроить простой CI-пайплайн (в любой системе на выбор, например, Travis CI, GitLab CI, Circle CI, Bitbucket Pipelines, GitHub Actions, etc.) со следующими шагами:

- запуск статических анализаторов кода (в зависимости от выбранного языка реализации - разные);
- запуск юнит-тестов приложения;
- для компилируемого языка - запуск сборки приложения.

Данное ТЗ не нужно копировать в README файл.