Вычислятор by Владислав Шкляров

Краткое описание:

Проект реализован на языке Go, состоит из 4-х микросервисов: http-service, business-service, logger-service, dashboard-service. Основной ролью является быстрая обработка запросов с операциями типа:

```
{
  "operations": [
 { "type": "calc", "op": "+", "var": "x", "left": 10, "right": 2 },
 { "type": "print", "var": "x" },
 { "type": "calc", "op": "-", "var": "y", "left": "x", "right": 3 },
 { "type": "calc", "op": "*", "var": "z", "left": "x", "right": "y" },
 { "type": "print", "var": "w" },
 { "type": "calc", "op": "*", "var": "w", "left": "z", "right": 9 }]
}
```

Логика и основные инструменты:

http-service

Центральным элементом проекта является http-service. Он слушает входящие запросы на порту :8080. Всего может быть три типа запросов:

1. /**process** – для обработки операций (POST)

Как только на сервер приходит сообщение, оно преобразуется в формат protobuf. Далее, информация о запросе отправляется по gRPC на два других микросервиса: log-service и business-service.

2. /**getLog** – для получения лога (GET)

Получает id лога и имя файла, в котором хранятся логи. Файлов может быть 3: business_logs.json, http_logs.json и undefined_logs.json. Эта информация переслывается в log-service, который ищет указанный лог в указанном файле, и, если находит, возвращает информацию о нем.

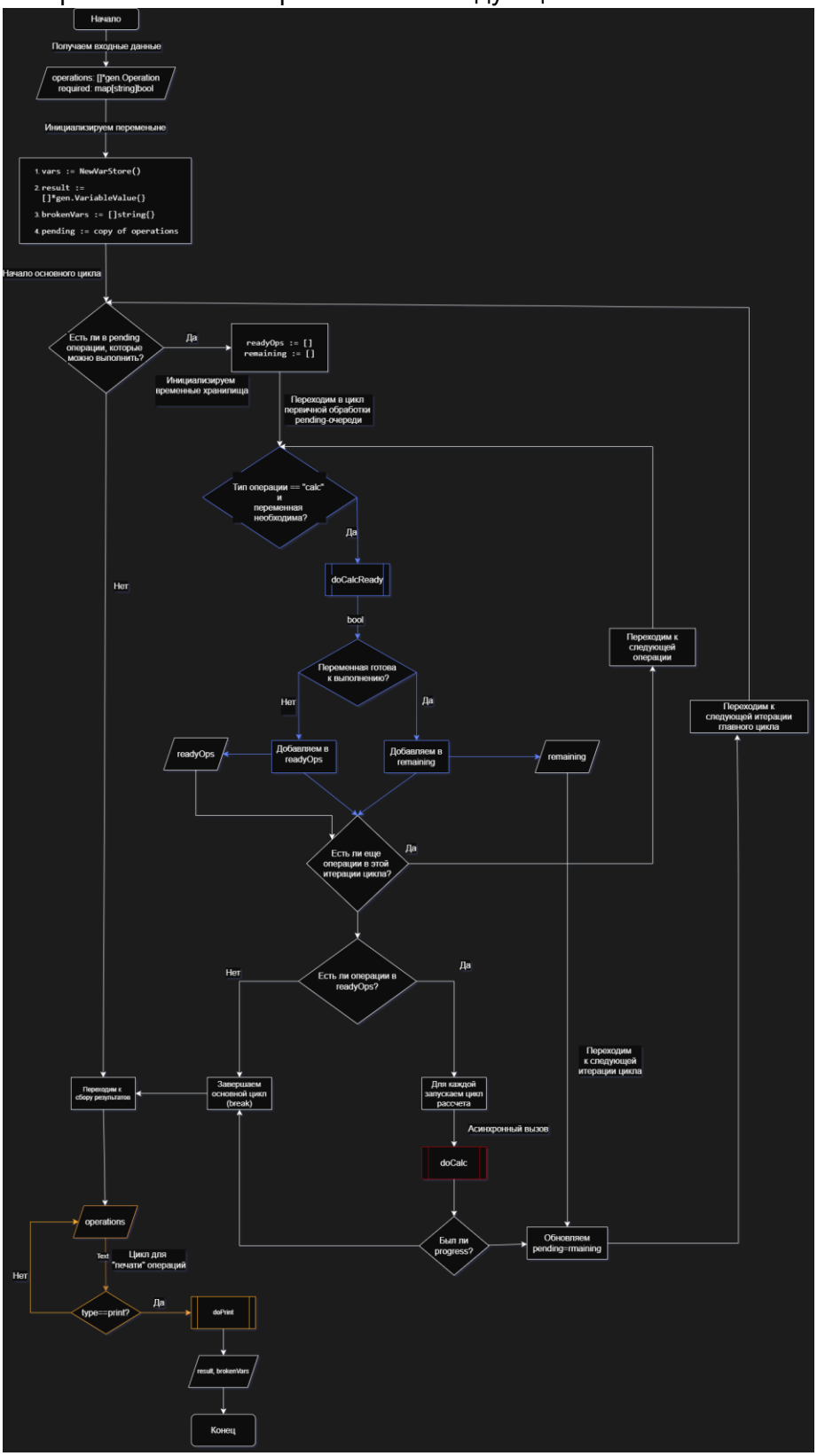
3. /**deleteLog** – для удаления лога (Delete)

Входные данные аналогичны getLog и также пересылаются лог-сервису. Он удаляет указанный лог из файла, возвращает ответ об успешности операции.

business-service

Этот сервис занимается вычислениями. Получает преобразованные операция из http-сервиса, считает результат и возвращает обратно. Кроме того отчет о проделанной работе он пересылает в лог-сервис, где информация записывается в business_logs.json.

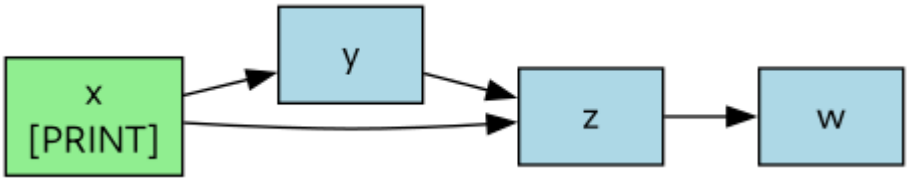
Алгоритм вычисления работает по следующей блок-схеме:



По сути дела, строится граф зависимостей переменных. Если переменная готова к рассчету – мы ее считаем, при этом если таких переменных несколько, то распараллеливаем через горутины. Если переменная не готова (например чтобы посчитать x надо знать y) она ждет своей очереди.

По окончании вычислений замеряется время, и результат возвращается в http-service.

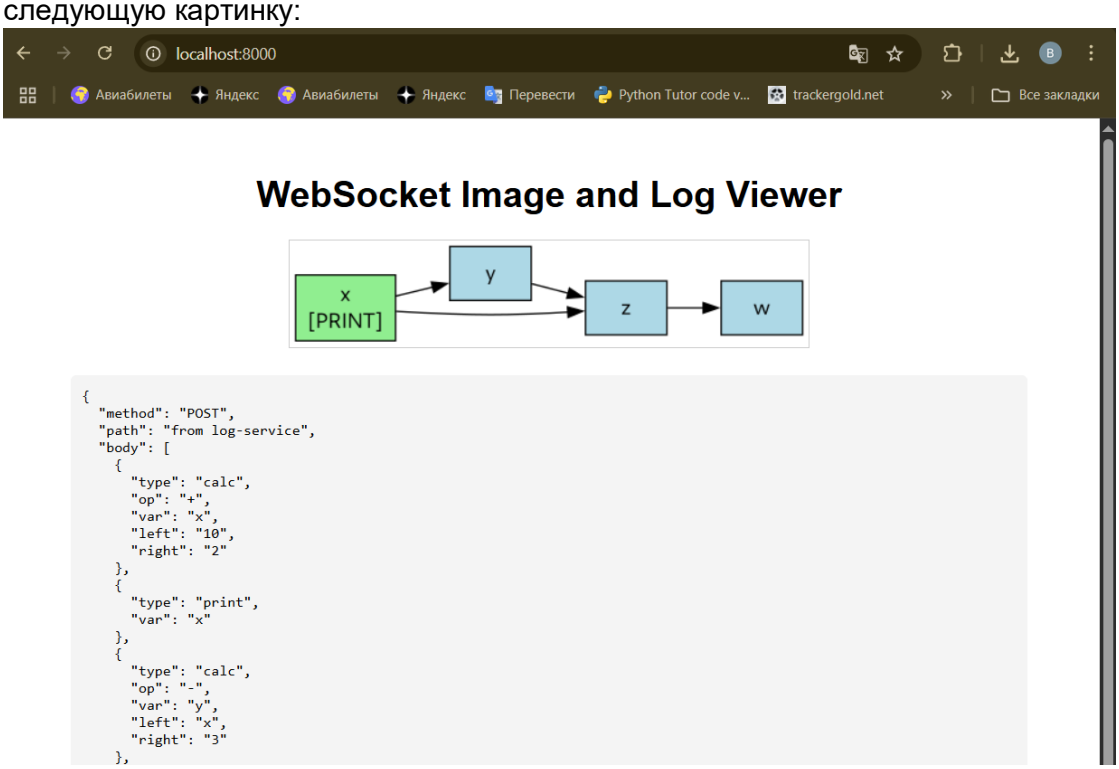
Кроме того, на основании построенного графа формируется картинка, например:



Когда картинка сформирована, бизнес-сервис отправляет сообщение в соответствующий топик кафке и саму картинку. Консьюмер кафки, расположенный в микросервисе dashboard в свою очередь, получив картинку передает ее веб-сокету.

log-service

Приняв запросы от http- или бизнес-сервиса он формирует логи при помощи логера Zap, и раскладывает их по соответствующим файлам в директории log_files. Также он может их либо читать и возвращать http-сервису, либо удалять. После того как новый запрос об операциях сохранился, лог запроса помещается в канал. Далее специальная функция ищет информацию о результатах, советующих этим операциям, и, как только соответствие найдено, пушит сообщение с операциями и результатом в кафку в советующий топик в формате protobuf. Конмьюмер на все том же dashboard-сервисе, получив это сообщение, выводит его рядом с картинкой. В результате, если все прошло без ошибок, клиент увидит следующую картинку:



dashboard-service

В основном про него уже все описано: он подписывается на топики в кафке, и держит открытым веб-сокет. Как только приходят сообщения, отсылает их клиентам. При этом он никак не связан с другими микросервисами, в отличии от остальных трех, связанных по gRPC.

Дополнительная информация:

Тестовое покрытие составляет примерно 40%. Тесты написаны для особо критичных частей, таких как чтение и запись логов, расчёт операций. Также реализовано несколько мок-тестов, например в http-сервисе.

http-сервис принимает запросы на порту **:8080**
Документация swagger доступпна по :**8080/swagger/index.html**
Веб-сокет: **:8000**