Neoflex
.Hack 2.0

Задачи

Инфраструктура, необходимая для решения задачи:
https://github.com/goodmove/hack-template

Java

Описание проекта

Задача
Реализовать систему, которая поможет отслеживать курс криптовалют через Telegram-бота.

Команды бота
Бот должен поддерживать следующие команды:
1. Подписка на стрим (1 уровень)

Бот принимает от пользователя `figi` валюты, за обновлением которой хочет следить.
Бот каждые 3 секунды отправляет пользователю данные из соответствующей свечи

2. Отписка от стрима (2 уровень)

При получении сообщения бот прекращает отправку данных пользователю

3. Подписка на события вида "падение курса валюты С от текущего на X%" (3 уровень)

При получении сообщения бот начинает следить за курсом и при необходимости уведомляет об событии пользователя

Опционально:
Можно реализовать подписку на несколько `figi` одновременно и отписку от стрима конкретного `figi`

Нотификации
Ваши коллеги из scala-команды реализуют механизм регистрации webhook для обработки триггеров. Про триггеры и события можете почитать в документации к их задаче.
Ваша задача – реализовать http сервер, на который будут приходить запросы для вызова webhook.

Схема потока данных примерно следующая:

```
1. пользователь регистрирует триггер
user --> telegram --> java app --> scala app

2. триггер активируется по данным котировок
kafka stream --> scala app --> java app --> telegram --> user
```
Если у вас нет Scala-разработчика
Если нет скалиста, вас не смогут уведомлять о срабатывании регистрируемых триггеров.
В этом случае можно реализовать регистрацию триггера полностью на своей стороне (описание есть в задаче скалистов) и сделать HTTP API для активации триггеров (вызывать методы API можно руками).

Итого

Ваш сервис должен реализовывать:
1. интеграцию с телеграмом
2. регистрацию и удаление тригеров на события (webhook). где и как хранить триггеры можете обсудить с коллегами из scala-команды
3. интерфейс для webhook
4. экспорт метрик по вебхукам для prometheus через micrometer

Scala
Описание проекта
Candles
Здесь (https://github.com/goodmove/hack-template) лежит код, который вам предстоит модифицировать для решения задачи хакатона.

> В `build.sbt` можете добавить в этот проект любые нужные зависимости

Задача
Реализовать pipeline для обработки свечей котировок криптовалют

`object Main` - входная точка приложения. Функция `def start` будет вызвана когда приложение запустится – ее вам и нужно модифицировать. Структура требует, чтобы вы вернули не код-результат исполнения, а инстанс класса `akka.actor.ActorRef`, через который в ваш стрим будут попадать объекты класса `rht.common.domain.candles.Candle`

Вам нужно сделать akka streams Graph, который будет принимать `Candle` и обрабатывать его, поддерживая как минимум:

- min/avg/max значения за все время работы приложения
- скользящее min/avg/max за 1, 5, 10 минут
- анализ срабатывания триггеров (про это ниже)

Триггеры
Клиент вашего сервиса хочет получать уведомления о событиях с котировками (например, "среднее за 10 минут достигло значения X" или "текущее значение упало на N% от максимума за день").
Будем называть `триггером` – условие, при котором нужно создать событие, и действие, которое нужно выполнить. `Событием` будем называть информацию о пользователе, который зарегистрировал триггер, валюте, к которой привязан триггер, и контексте, по которому сработал триггер.

Таким образом, есть условие (триггер), действие и контекст для выполнения действия (событие).

> Например, пользователь зарегистрировал триггер с условием "падение цены биткоина на 10% от текущей" и действием "получить уведомление об этом в Telegram".
Тогда триггер имеет вид: `trigger = (id=trigger_1, figi=BTC, current_price=100, target_price=90, tg_notification)`. Событие, которое необходимо отправить при срабатывании триггера будет иметь вид:
`event = (user_1, BTC, price=90)`.

Про объект Candle

`interval` – интервал свечи

`figi` – единый глобальный идентификатор ценной бумаги (здесь эмулируется, но является уникальным для каждой криптовалюты).
Список доступных идентификаторов можно посмотреть в объекте `rht.common.domain.candles.Figis`

`details` – детали свечи

`details.low` – нижняя граница стоимости инструмента в этой свечи

`details.high` – верхняя граница стоимости инструмента в этой свечи

`details.open` – стоимость инструмента на момент открытия этой свечи

`details.close` – стоимость инструмента на момент закрытия этой свечи

`details.openTime` – время открытия этой свечи

Запуск приложения

1. Приложение можно запустить с помощью sbt. Добавлен плагин `sbt-revolver`, поэтому можно запускать с помощью команд `<module-name>/reStart`,
а останавливать с помощью `<module-name>/reStop`
2. Работа с Kafka (**Очень важно**)

Каждая команда должна отредактировать файл `application.conf` и заменить значение `group.id` на следующее: `hackathon-<team-name>`, где team-name – название вашей команды. Если указать другое имя, у вас не будет прав на чтение стрима.
Если все команды сделают одинаковое значение, только одна команда будет получать данные из стрима

Объединение данных
Так как бэкенд для бота будет написан на Java, нужно решить, как организовать код.
Это важно для передачи информации из стрима (реализуют ваши коллеги на scala) в бота.

Варианты:
1. Одно приложение (в одной программе запускается и стрим, и телеграм бот).
Один будет процесс ОС, что означает, что данные будут находиться в общей памяти, а значит, к ним легко получить доступ боту. Однако это же означает, что Java-разработчику придется работать в этом sbt проекте, разбираться с его устройством, деталями работы и нужно будет следить за зависимостями в Java<->Scala коде

2. Разные приложения (разные процессы ОС). Это проще с точки зрения организации работы (каждый независимо делает свой модуль), но
требует дополнительной координации информации между процессами, так как общей памяти у них уже нет

Рекомендации

- выбрать вариант 2, описанный выше (чтобы коллеге джависту было легче)
- использовать простое in-memory хранилище для обмена данными между приложениями (например, [redis](https://redis.io/), для которого есть куча библиотек на scala и java). можно считать, что redis - `java ConcurrentHashMap`, в который есть доступ из любого процесса
- Telegram-боту делать polling данных из redis и отдавать заинтересованным пользователям

Если выберете вариант 1, можно делать то же самое с `ConcurrentHashMap`

Заметки
ActorRef
Рекомендуется создать `Source` как `Source.actorRef` без backpressure,

потому что это не будет поддерживаться источником сообщений.
Можно установить размер буфера в 2000 сообщений – этого должно хватить для real-time обработки

Вы никак не ограничены в структуре графа, которым будет обрабатываться свеча

Итого
Ваш сервис должен уметь:
1. обрабатывать стрим с котировками (методы анализа можете выбрать самостоятельно)
2. регистрировать триггеры с webhook'ами в качестве действий. webhook – это механизм обратной связи при асинхронном взаимодействии систем. Вы можете реализовать это в виде хранения `(http method, url, payload)`, с которыми необходимо выполнить HTTP запрос. Сервис, обрабатывающий webhook, будут делать ваши коллеги из java команды
3. отдавать метрики приложения через micrometer для prometheus. Конкретные метрики можете выбрать самостоятельно (например, число активных триггеров, число ошибок, курс конкретной криптовалюты)

SQL
Задача

Ваша задача – обсудить с коллегами из команды QA & Support, какие метрики и как они будут анализировать.
Для всех таких метрик нужно реализовать SQL запросы/процедуры/триггеры, чтобы получить эквивалентные данные или способствовать их формированию (например, через триггер на вставку или обновление)

Support
Задача

Ваша задача – построить дашборд в grafana по данным из prometheus. В prometheus будут прилетать метрики приложений, которые реализуют ваши коллеги из java и scala команд. Обсудите с ними, какие данные вам будут полезны и попросите коллег настроить экспорт этих данных.
Кроме этого, необходимо интегрировать slack, zabbix и prometheus: по триггерам на данных из prometheus отправлять zabbix events/alarms в slack (в канал или конкретному пользователю)
> например, не сработал вебхук –> пользователю не отправилось уведомление по триггеру –> поддержка узнает об этом через слак и бежит проверять или дергать ответственных

Тестировщик

Составить план тестирования перед и после гипотетического релиза

Аналитик

1. Прочитать условия задач, составить вопросы и пойти с ними к экспертам – собрать требования и дополнить ими условия задач. Считать, что эксперты – бизнес-заказчики
2. Составить модели процессов всех уровней для разрабатываемой системы, используя модель C4 (https://c4model.com/) или BPMN (используя механизм https://app.diagrams.net/),
3. Сформировать ТЗ на базе всех поставленных команде задач.

Дополнительный функционал

После реализации основных заданий вам предлагается реализовать на свое усмотрение дополнительный функционал бота и обосновать его необходимость с точки зрения пользователя. За это задание вы получите дополнительные баллы в зависимости от сложности.

Как на GitHub корректно скопировать себе репозиторий

1. Завести приватный репозиторий на github (bitbucket/gitlab/whatever)
2. Сделать его приватным и раздать права нужным людям
3. Клонировать репозиторий с шаблоном приложения
   ```shell script
   git clone https://github.com/goodmove/hack-template.git
   ```
4. Перейти в репозиторий и поменять `origin`
   ```shell script
    cd hack-template
    git remote remove origin
    git remote add origin <private-repo-url>
    git push
   ``

## Общие критерии оценки

Командами необходимо в первую очередь реализовать обязательную часть задачи и получить таким образом допуск к дополнительной части.
В обязательной части необходимо реализовать задачи в рамках тех компетенций, которые присутствуют в команде. Например, если в команда java, scala - разработчики
   ```

и тестировщик, то им необходимо реализовать обязательные задания по java, scala и тестированию для получения допуска к реализации дополнительного функционала. Пока у них в команде нет SQL-разработчика, то делать SQL-часть для допуска не требуется.

Дополнительная часть оценивается баллами от 0 до 5.

Компетенция	**Критерий**	**Оценка**
Обязательная часть (допуск)		
Java/Scala	Работоспособность (1 уровень)	Да - допуск Нет - не допуск
	Работоспособность (2 уровень)	Да - допуск Нет - не допуск
	Работоспособность (3 уровень)	Да - допуск Нет - не допуск
SQL	Работоспособность	Да - допуск Нет - не допуск
Support	Работоспособность	Да - допуск Нет - не допуск
Аналитик	Корректность определения метрик	Есть метрики и их обоснование - допуск Нет метрик или они обоснованы некорректно - не допуск
Тестировщик	Корректность плана тестирования	План корректен - допуск План некорректен - не допуск
Дополнительная часть		
-	Дополнительный функционал	0 - дополнительный функционал отсутствует 1 - есть идеи доп.функционала, но без реализации 2 - есть реализация одной простой функции 3- есть реализация двух-трех простых функций 4-5 - есть реализация сложный функций

	Презентация решений	0 - решение непонятно, на вопросы ответов нет 1- решение понятно, но нет ответов на вопросы 2- решение понятно, но не на все вопросы есть ответы 3- решение понятно, есть ответы на вопросы