**Описание проекта**

**Caster**

**Caster** — это компонент, который считывает сообщения из Kafka и пересылает их через UDP на другой конец, в **Catcher**. Caster подписывается на определенные топики в Kafka и обрабатывает полученные сообщения. Для надежности передачи сообщений используется механизм вычисления хэша (SHA-1), что позволяет **Catcher** проверить целостность данных.

**Catcher**

**Catcher** — это компонент, который получает сообщения через UDP, проверяет их целостность с помощью хэширования и отправляет обработанные данные обратно в Kafka. Catcher слушает входящие сообщения через определенный порт и топики, восстанавливая сообщения в Kafka для дальнейшей обработки.

**Используемые библиотеки и технологии**

**Go (Golang)**

* Оба проекта **Caster** и **Catcher** написаны на языке программирования **Go**

**Kafka (segmentio/kafka-go)**

* Для взаимодействия с **Kafka** используется библиотека **segmentio/kafka-go**.
  + **Kafka Writer**: Используется для записи сообщений в Kafka (в **Catcher**).
  + **Kafka Reader**: Используется для чтения сообщений из Kafka (в **Caster**).
  + С помощью этой библиотеки, оба компонента взаимодействуют с Kafka для передачи данных между распределенными системами.

**UDP (net)**

* Для передачи данных между **Caster** и **Catcher** используется **UDP**.
  + **net** пакет в Go обеспечивает отправку и прием сообщений через UDP, что обеспечивает высокую скорость передачи данных.

**SHA-1 (crypto/sha1)**

* Для проверки целостности данных используется алгоритм хэширования **SHA-1**, реализованный в пакете **crypto/sha1**.
  + **Caster** генерирует хэш сообщений перед их отправкой.
  + **Catcher** проверяет полученные данные по хэшу.

**Prometheus (prometheus/client\_golang)**

* Используется для сбора метрик работы системы.
  + **Prometheus** собирает метрики, такие как количество запросов или ошибок, и экспонирует их для мониторинга.
  + **prometheus/client\_golang** предоставляет возможность сбора и экспонирования метрик.

**Docker и Docker Compose**

* **Docker** используется для контейнеризации проектов **Caster** и **Catcher**, что обеспечивает независимость от окружения и простоту развертывания.
* **Docker Compose** позволяет автоматически развернуть всю систему, включая Kafka, Zookeeper и сервисы **Caster** и **Catcher**, с помощью одного конфигурационного файла.

**Основные компоненты**

**Caster**

* **Kafka Reader**: Подписывается на один или несколько топиков Kafka, читает сообщения и отправляет их через UDP.
* **UDP Sender**: Отправляет сообщения на указанный IP и порт через UDP.
* **SHA-1 Hash Generator**: Генерирует хэш для каждого сообщения, отправленного по UDP.

**Catcher**

* **UDP Receiver**: Получает сообщения через указанный порт по UDP.
* **SHA-1 Hash Verifier**: Проверяет целостность сообщений с помощью хэша SHA-1.
* **Kafka Writer**: Записывает проверенные сообщения в указанный топик Kafka.

**Переменные окружения**

**Caster**

| **Переменная** | **Описание** |
| --- | --- |
| UDP\_ADDRESS | Адрес для отправки сообщений через UDP. |
| TOPICS | Название топика для чтения сообщений. |
| KAFKA\_LOGGER\_TOPIC | Название топика для служебныз сообщений. |
| DUPLICATES\_COUNT | Количество дублирование сообщениий. |
| ENABLE\_HASH | Включить хэширование сообщений (значения: true или false). |

**Catcher**

| **Переменная** | **Описание** |
| --- | --- |
| KAFKA\_LOGGER\_TOPIC | Название топика для служебных сообщений. |
| UDP\_IP | IP-адрес для приема сообщений через UDP. |
| UDP\_PORT | Порт для приема сообщений через UDP. |
| ENABLE\_HASH | Включить проверку хэш-сообщений (значения: true или false). |

**Системные требования**

* **Go 1.20 или выше**: Для сборки и выполнения проекта.
* **Kafka и Zookeeper**: Для работы с сообщениями и топиками.
* **Docker и Docker Compose**: Для контейнеризации и упрощенного развертывания.

**Как запустить проекты**

**Шаг 1: Сборка проектов**

1. Склонируйте репозиторий с кодом **Caster** и **Catcher**.
2. Соберите проекты с помощью Go:

go build -o caster

go build -o catcher

**Шаг 2: Настройка Docker Compose**

Используйте следующий файл **docker-compose.yml**, чтобы развернуть **Kafka**, **Zookeeper**, **Caster**:

version: '3'  
services:  
 zookeeper\_caster:  
 image: confluentinc/cp-zookeeper:latest  
 environment:  
 ZOOKEEPER\_CLIENT\_PORT: 2181  
 ZOOKEEPER\_TICK\_TIME: 2000  
 ports:  
 - "2181:2181"  
 networks:  
 - kafka\_net  
  
 kafka\_caster:  
 image: confluentinc/cp-kafka:latest  
 depends\_on:  
 - zookeeper\_caster  
 ports:  
 - "9092:9092"  
 environment:  
 KAFKA\_BROKER\_ID: 1  
 KAFKA\_ZOOKEEPER\_CONNECT: zookeeper\_caster:2181  
 KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka\_caster:9092  
 KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT  
 KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1  
 networks:  
 - kafka\_net  
  
 topic\_creator:  
 image: confluentinc/cp-kafka:latest  
 depends\_on:  
 - kafka\_caster  
 networks:  
 - kafka\_net  
 entrypoint: /bin/sh  
 command: -c "  
 echo 'Waiting for Kafka to be ready...';  
 while ! nc -z kafka\_caster 9092; do  
 sleep 1;  
 done;  
 echo 'Kafka is ready. Creating topics...';  
 kafka-topics --create --topic LogsAndMetrics --bootstrap-server kafka\_caster:9092 --replication-factor 1 --partitions 1;  
 echo 'Topics created.';  
 sleep 10;"  
  
 kafka-ui:  
 image: provectuslabs/kafka-ui  
 container\_name: kafka-ui  
 ports:  
 - 8090:8080  
 restart: always  
 environment:  
 - KAFKA\_CLUSTERS\_0\_NAME=local  
 - KAFKA\_CLUSTERS\_0\_BOOTSTRAPSERVERS=kafka\_caster:9092  
 - KAFKA\_CLUSTERS\_0\_ZOOKEEPER=zookeeper\_caster:2181  
 networks:  
 - kafka\_net  
  
 caster:  
 build:  
 context: https://github.com/isatay012or02/kafka-diode-caster.git  
 dockerfile: Dockerfile  
 environment:  
 UDP\_ADDRESS: "188.188.0.1:9999"  
 TOPICS: "SyslogProducer,MQTTProducer,FileTransferProducer"  
 ENABLE\_HASH: "true"  
 DUPLICATES\_COUNT: 1  
 KAFKA\_LOGGER\_TOPIC: "LogsAndMetrics"  
 networks:  
 - kafka\_net  
 restart: always  
  
networks:  
 kafka\_net:  
 driver: bridge

Используйте следующий файл **docker-compose.yml**, чтобы развернуть **Kafka**, **Zookeeper**, **Catcher**:

version: '3'  
services:  
 zookeeper\_catcher:  
 image: confluentinc/cp-zookeeper:latest  
 environment:  
 ZOOKEEPER\_CLIENT\_PORT: 2182  
 ZOOKEEPER\_TICK\_TIME: 2000  
 ports:  
 - "2182:2182"  
 networks:  
 - kafka\_net  
  
 kafka\_catcher:  
 image: confluentinc/cp-kafka:latest  
 depends\_on:  
 - zookeeper\_catcher  
 ports:  
 - "9093:9093"  
 environment:  
 KAFKA\_BROKER\_ID: 1  
 KAFKA\_ZOOKEEPER\_CONNECT: zookeeper\_catcher:2182  
 KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka\_catcher:9093  
 KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT  
 KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1  
 KAFKA\_AUTO\_CREATE\_TOPICS\_ENABLE: 'true'  
 networks:  
 - kafka\_net  
  
 topic\_creator:  
 image: confluentinc/cp-kafka:latest  
 depends\_on:  
 - kafka\_catcher  
 networks:  
 - kafka\_net  
 entrypoint: /bin/sh  
 command: -c "  
 echo 'Waiting for Kafka to be ready...';  
 while ! nc -z kafka\_catcher 9093; do  
 sleep 1;  
 done;  
 echo 'Kafka is ready. Creating topics...';  
 kafka-topics --create --topic LogsAndMetrics --bootstrap-server kafka\_catcher:9093 --replication-factor 1 --partitions 1;  
 echo 'Topics created.';  
 sleep 10;"  
  
 kafka-ui-catcher:  
 image: provectuslabs/kafka-ui  
 container\_name: kafka-ui-catcher  
 ports:  
 - 8092:8080  
 restart: always  
 environment:  
 - KAFKA\_CLUSTERS\_0\_NAME=local-catcher  
 - KAFKA\_CLUSTERS\_0\_BOOTSTRAPSERVERS=kafka\_catcher:9093  
 - KAFKA\_CLUSTERS\_0\_ZOOKEEPER=zookeeper\_catcher:2182  
 networks:  
 - kafka\_net  
  
 catcher:  
 build:  
 context: https://github.com/isatay012or02/kafka-diode-catcher.git  
 dockerfile: Dockerfile  
 environment:  
 UDP\_IP: "127.0.0.1"  
 UDP\_PORT: 9999  
 ENABLE\_HASH: "true"  
 KAFKA\_LOGGER\_TOPIC: "LogsAndMetrics"  
 networks:  
 - kafka\_net  
 restart: always  
  
networks:  
 kafka\_net:  
 driver: bridge

**Шаг 3: Запуск контейнеров**

Запустите сервисы через **Docker Compose**:

docker-compose up –build

**Мониторинг и логирование**

* **Prometheus**: Метрики обоих сервисов отправляются в Kafka через специально настроенные топики.
* **Kafka Logger**: Логи обоих сервисов отправляются в Kafka через специально настроенные топики.

**Дополнительные инструкции**

* **Изменение подписываемых топиков в caster:** достаточно изменить переменную среды TOPICS в docker-compose
* **Примечение по Catcher:** по умолчаниюпорт для контейнера Catcher должен быть "9999:9999/udp"
* **Дупликатор для реализации избыточности:** отправляются последовательно из Caster в Catcher копии пакета (количество копий указывается в переменной environment DUPLICATES\_COUNT контейнера catcher в файле docker-compose.yml)