DOI: 10.15514/ISPRAS-2022-34(2)-9

Система управления заданиями автоматизированного сбора данных из сети Интернет

[1,2] *В.А. Лазарев, ORCID: 0000-0002-4074-5424 <vlazarew@ispras.ru>*
[1] *М.И. Варламов, ORCID: 0000-0002-1083-6210 <varlamov@ispras.ru>*
[1] *А.К. Яцков, ORCID: 0000-0002-1312-1675 <yatskov@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет им. М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*

Аннотация. Работа посвящена исследованию и разработке системы управления заданиями автоматизированного сбора данных из сети Интернет. Статья содержит описание реализованных методологий и повествует о созданных приемах взаимодействия с контейнерами, содержащими в себе приложения для сбора данных. В ходе работы были изучены и представлены существующие различные сервисы автоматизированного сбора данных из сети Интернет: готовые решения с открытым исходным кодом, облачные сервисы с обширным функционалом, а также собственное решение под управлением Kubernetes. В результате работы реализована и внедрена в платформу для анализа данных Talisman система управления заданиями, которая обеспечивает горизонтальную масштабируемость, изолированность окружения сборщиков и независимость от технологии их разработки.

Ключевые слова: сбор данных; система управления заданиями; виртуализация; Kubernetes

Для цитирования: Лазарев В.А., Варламов М.И., Яцков А.К. Система управления заданиями автоматизированного сбора данных из сети Интернет. Труды ИСП РАН, том 34, вып. 2, 2022 г., стр. 111-122. DOI: 10.15514/ISPRAS–2022–34(2)–9

Job management system for automated data collection from the Internet

[1,2] *V.A. Lazarev, ORCID: 0000-0002-4074-5424 <vlazarew@ispras.ru>*
[1] *M.I. Varlamov, ORCID: 0000-0002-1083-6210 <varlamov@ispras.ru>*
[1] *A.K. Yatskov, ORCID: 0000-0002-1312-1675 <yatskov@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*

Abstract. This work is devoted to the research and development of a task management system for automated data collection from the Internet. This article contains a description of the implemented methodologies and tells about the techniques created by interacting with containers containing data collection applications. In the course of the work, various existing services for automated data collection from the Internet were studied and presented: ready-made open source solutions, cloud services with extensive functionality, as well as our own solution running Kubernetes. As a result of the work, a task management system was implemented for Talisman data analysis platform, which provides horizontal scalability, isolation of the crawler environment and independence from the technology of their development.

Keywords: data collection; task management system; virtualization; Kubernetes

For citation: Lazarev V.A., Varlamov M.I., Yatskov A.K Job management system for automated data collection from the Internet. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 2, 2022, pp. 111-122 (in Russian). DOI: 10.15514/ISPRAS–2022–34(2)–9

1. Введение

В настоящее время активно развиваются приложения и сервисы, позволяющие собирать, накапливать и анализировать информацию для уменьшения затрат на временные ресурсы пользователей и предоставления актуальной информации и контента.

Возможных направлений, в которых бы пригодился сбор и анализ данных, достаточно много. Базовым инструментом в каждом из них является сборщик данных – приложение, которое принимает на вход некоторые аргументы и настройки, отправляет HTTP-запросы на указанные web-страницы и на основе своей спецификации формирует и возвращает сериализуемый ответ. Для получения обширной и полной картины по какому-либо объекту необходимо обрабатывать несколько источников одновременно. Из этого вытекает, что для системы, которая хочет агрегировать в себе всевозможные источники данных для предоставления качественного анализа, предстоит использование большого количества приложений для сбора информации.

Система управления заданиями автоматизированного сбора данных из сети Интернет должна позволять загружать новые сборщики, производить запуск и отмену сбора данных для конкретной версии сборщика, получать собираемые сборщиками документы, а также отслеживать метрики сбора, отправляемые запросы и логи заданий. При этом такая система должна удовлетворять ряду требований.

- Горизонтальная масштабируемость. Количество одновременных задач сбора данных должно быть ограничено аппаратными возможностями сервера, но никак не на уровне приложения.
- Изолированность зависимостей сборщиков. Каждый сборщик должен содержать только необходимые ему зависимости от библиотек. Необходимо для разрешения конфликтов между зависимостями различных приложений для сбора данных
- Независимость от технологии разработки сборщиков. Система должна позволять загружать и запускать сборщики, написанные с помощью различных языков программирования и фреймворков.

В работе описывается методология разработки системы управления заданиями сбора данных, отвечающей поставленным требованиям, и ее реализация в рамках платформы анализа данных Talisman [1].

2. Существующие решения

Рассматриваемые инструменты можно разбить на две категории: локальные системы, которые можно установить, использовать и дорабатывать, с открытым исходным кодом; облачные платформы, предоставляющие доступ к информации, анализу за определенную плату.

2.1 Локальные системы управления заданиями

Авторы статьи [2] предложили свой подход, но он основывается на реализации одного параллельного сборщика. В нашем же случае необходимо осуществлять параллельную работу различных сборщиков данных.

Scrapy Cluster [3] – проект Scrapy для создания распределенного кластера для сбора данных по запросу. Поддерживает динамические сборщики, работающие по запросу и позволяющие произвольно собирать любую web-страницу. Обеспечиваются масштабирование

экземпляров; параллельное выполнение и механизм приоритетов. Итогом сбора является HTML-код страницы, из которого далее необходимо структурировать информацию.

Scrapyd [4] – инструмент, позволяющий управлять несколькими проектами сбора данных. Сборщики загружаются в систему как составные части проекта, причем сам файл проекта имеет удобное для Python расширение .egg. Обладает всеми преимуществми Scrapy Cluster, а также поддерживает сборщики, написанные при помощи фреймворка scrapy, что позволяет получать сразу структурированные данные. Однако инструмент обеспечивает только вертикальную масштабируемость и не поддерживает изолированность зависимостей между сборщиками.

Scrapyrt (Scrapy realtime) [5] – это HTTP-сервер, который предоставляет API для настройки расписаний запусков Scrapy сборщиков и получения результатов сбора данных. Обладает довольно скудным функционалом, невозможно останавливать сбор или ставить его на паузу, работает только со scrapy-сборщиками и не рекомендуется для использования с длительным сбором данных.

Ferrit [6] – HTTP-сервер, предоставляющий API для создания сборщиков, их запусков и мониторинга задач сбора данных, написан на языке программирования Scala. Однако созданные сборщики получаются весьма примитивными, без возможности детально настроить поведение и извлекаемую информацию, равно как и указать формат выхода сообщений. Загрузку каких-либо сторонних сборщиков не поддерживает.

2.2 Облачные сервисы сбора данных

Zyte Scrapy Cloud [7] – это сервис для сбора данных. Платформа предоставляет интерфейс для мониторинга собранных данных. Есть возможность фокусированного сбора данных, основываясь на целевом бренде или товаре. Также поддерживается загрузка сборщиков на любом языке программирования, поскольку есть возможность загрузить сборщики как Dockerfile, из которого соберется необходимый образ. Сервис обладает средствами мониторинга и тестирования, высокой масштабируемостью, поддерживает сборщики на основе любых технологий, если Dockerfile соответствует описанному контракту. Платформа доступна только по подписке.

Web Scraper Cloud [8] – сервис, использующий плагин Web Scraper и предоставляющий над надстройки для обеспечения автоматизации, масштабируемости и управления задачами сбора. Сервис обладает встроенным прокси, планировщиком задач, средствами мониторинга и тестирования, возможностью планирования задач и их параллельного запуска. Однако используется собственный JSON-формат описания сборщиков, загрузка других сборщиков не поддерживается, и платформа доступна только по подписке без возможности доработок под собственные нужды.

Octoparse [9] – сервис для извлечения данных. Не требует знаний в программировании, поскольку перед сбором предоставляется возможность выделения областей с требуемой информацией для извлечения. Предоставляет возможность производить каждый последующий запрос с нового IP-адреса. Отсутствует мониторинг задач, требуется платная подписка на сервис, и нет возможности загружать и использовать сборщики на любом фреймворке.

DataOx [10] – сервис для получения данных из сети Интернет в большом количестве. Отличительной чертой является то, что имеется возможность рассылки обновленной информации по указанным ресурсам. Сервис предлагает сбор для сайтов любой сложности, планировщик задач, параллельный запуск задач сбора данных, средства мониторинга и тестирования. Отсутствует возможности загружать и использовать сборщики на любом фреймворке, требуется платная подписка на сервис.

В результате исследования существующих систем управления заданиями автоматизированного сбора данных из сети Интернет можно сформулировать следующие выводы:

- бесплатные сервисы не полностью удовлетворяют наши требования, зачастую поддерживают только один фреймворк и сильно ограничены в масштабируемости;
- облачные сервисы удобны в использовании, но все они не предоставляют исходный код для доработок, и для использования платформ необходимо платить.

Поскольку рассмотренные системы не полностью удовлетворяют требованиям, сформулированным в разд. 1, и, кроме того, необходима изолированность приложений и высокая горизонтальная масштабируемость, следует рассмотреть системы оркестрации контейнеризированных приложений.

3. Исследование и выбор средства контейнерной оркестрации

Основная функциональность инструментов контейнерной оркестрации – развертывание приложений, упакованных в контейнеры, отслеживание статуса жизнеспособности запущенных контейнеров, поддержание необходимого числа реплик контейнеров, мониторинг состояния контейнеров, обновление их версий, и уничтожение развернутых приложений.

Типичный сборщик – это запуск некоторого приложения внутри отдельного контейнера. Используя контейнеры, можно достичь изолированности зависимостей приложений для сбора, горизонтально масштабироваться за счёт существующих средств контейнерной оркестрации, а также поддерживать сборщики на любых технологиях их разработки.

Kubernetes [11] – это открытый проект для автоматизации развертывания, масштабирования и управления контейнерными приложениями. Поддерживает работу с Docker и предоставляет средства для управления контейнерами. Модель делится на два типа узлов: управляющие и рабочие. Рабочие узлы объединены в группы, каждую из групп контролирует свой управляющий узел, который распределяет задачи создания и управления контейнерами, а также отслеживания их состояния. Kubernetes поддерживает работу с Containerd-контейнерами и предоставляет механизмы для развертывания, управления, масштабируемости и мониторинга работы контейнеров.

Удобным инструментом для создания модулей в Kubernetes является Job (задание). При помощи задания можно создать один или несколько модулей, причем если какой-либо из модулей не смог развернуться, то задание повторит попытку указанное в конфигурационном файле количество раз, либо до тех пор, пока все связанные модули успешно не завершат работу. Задание отслеживает состояние всех модулей, которые оно породило. При удалении задания все порожденные модули также будут удалены. Зачастую задания в Kubernetes используются для параллельного запуска нескольких модулей.

Docker Compose [12] – инструмент для настройки и запуска многоконтейнерных приложений Docker. Для конфигурации приложения используется файл docker-compose.yml, в котором указываются необходимые для развертывания приложения контейнеры, взаимосвязь и взаимозависимость между контейнерами, настройки и параметры окружающей среды для каждого контейнера. Это очень удобное средство для небольших или локальных приложений, поскольку обладает простой настройкой и конфигурацией. Но присутствуют функциональные ограничения и недостатки по сравнению с другими платформами.

Docker Swarm [13] – специальный режим управления в Docker Engine. Однако в актуальных версиях Docker режим swarm поддерживается изначально без дополнительных настроек. Этот инструмент отлично подходит для небольших кластеров с малыми вычислениями, где требуется по большей части простота в настройке и конфигурации. Главный недостаток инструмента – поддержка контейнеров Docker и только.

Apache Mesos [14] – создан как менеджер кластера нового поколения. Это программное обеспечение с открытым исходным кодом, которое обеспечивает эффективную изоляцию ресурсов и их совместное использование в распределенных приложениях или платформах. Mesos – это ядро распределенной системы с полным API для программирования непосредственно в центре обработки данных. Архитектура Mesos состоит из основного процесса, который управляет подчиненными демонами, работающими на каждом узле кластера, и фреймворков, выполняющих задачи на этих подчиненных устройствах. Mesos лучше всего подходит для больших систем и обеспечивает максимальную избыточность. Mesos рекомендуется, если у вас есть существующие рабочие нагрузки, такие как Hadoop, Kafka и т.д. Он дает платформу, которая позволяет чередовать эти рабочие нагрузки друг с другом. Это самая стабильная платформа, но слишком сложная для небольших систем до 10-20 узлов. Основная сила Mesos заключается в больших данных и аналитике. Оркестровка контейнеров не совсем его дело. Mesos обладает чрезмерно большим функционалом и слишком универсален.

Приведенный анализ систем контейерной оркестрации показывает, что наиболее взвешенным решением, на базе которого стоит реализовать требуемую систему, является Kubernetes, поскольку система достаточно универсальна, соответствует необходимым функциональным требованиям и не перегружена излишними возможностями.

Кстати, в статье [15] авторы исследовали архитектуру и производительность различных систем контейнерной оркестрации. Их выводы совпали с нашими: они считают Kubernetes лучшим решением на текущий момент.

4. Жизненный цикл абстрактного сборщика данных

Чтобы составить спецификацию для работы со сборщиками как приложениями внутри контейнеров, необходимо рассмотреть этапы жизненного цикла произвольного сборщика в системе.

Жизненный цикл любого сборщика подразумевает под собой все этапы от его загрузки в систему до выполнения запусков на основе сборщика. При желании сборщик можно удалить из системы, после чего доступ к нему теряется навсегда.

I. Загрузка сборщика в систему управления заданиями автоматизированного сбора данных.

II. После загрузки сборщик может быть запущен. На его основе создается задача сбора, которая может находиться в нескольких состояниях:

   a. В ожидании – задача создана только в рамках базы данных, ожидает готовности окружения для начала работы.

   b. В работе – задача непосредственно находится в процессе выполнения. Есть возможность остановить процесс по необходимости. Если задача выполняется длительное время, по полученным сообщениям видно, что коэффициент полезного действия стремится к нулю, но при этом место в пуле запусков занято.

   c. Завершена – задача сбора данных завершена, место в пуле запусков освобождено, все собранные сообщения, логи, запросы сохранены в соответствующих хранилищах. Задача может быть завершена с несколькими статусами:

      i. успешно – ошибок в ходе сбора не обнаружено;

      ii. с ошибками – в ходе сбора были обнаружены и обработаны ошибки сбора, некоторые сообщения утеряны;

      iii. критическая ошибка – в ходе сбора была выявлена критическая ошибка, вследствие которой процесс сбора данных был экстренно завершен;

      iv. отменена – при отмене задачи запуска пользователем. При этом та информация, которая успела собраться, будет представлена для хранения и анализа.

III. Когда сборщик становится неактуальным, необходимо иметь возможность удалить его полностью или только его версию. При удалении версии она помечается как удаленная. Создавать задачи сбора по сборщику с удаленной версией невозможно.

Абстрактный сборщик имеет возможность принимать на вход аргументы и настройки. Эти параметры необходимы для более гибкого сбора данных и возможности переиспользовать один и тот же сборщик для различных целей. Настройки влияют на конфигурационные параметры сборщика, аргументы – определяют поведение сборщика в целом.

Результат работы сборщика данных – множество сообщений, логов, метрик и запросов, которые являются, не ограничивая общности, Json-объектами. И поскольку методов сбора данных из сети Интернет огромное количество, а обрабатывающая система одна, имеется необходимость в стандартизации выходного формата сообщений сборщиков данных.

Представленная модель абстрактного сборщика частично совпадает с моделью жизненного цикла сборщика, реализованного на Scrapy. Разница состоит в том, что Scrapy подразумевает загрузку в систему проектов сбора данных – набор сборщиков, объединенных общим конфигурационным файлом. Как следствие и версионирование осуществляется в рамках проекта сбора данных.

5. Спецификация устройства контейнера со сборщиками

Спецификация устройства контейнеров, содержащих приложения для сбора данных, включает спецификации для загрузки сборщиков в систему и получения результатов их работы.

5.1 Спецификация загрузки сборщиков

Для корректной работы внутри системы поставщик сборщиков должен обеспечить следующее:

- Dockerfile для корректного создания контейнера (может использоваться как простое копирование jar-файла, так и установка специального окружения с разархивацией исходных файлов);
- скрипт-entrypoint – входная точка для Dockerfile, в котором должны быть реализованы команды:
  - list – извлечение списка сборщиков в командную строку;
  - schedule – запуск конкретного сборщика с переданными аргументами командной строки.
- архив с исходным кодом.

5.2 Спецификация формата сообщений сборщиков данных

Для обработки сообщений сборщиков была разработана спецификация, которой следует придерживаться для оптимальной работы и корректного разбора сообщений по финальным хранилищам.

Сборщики должны выводить в поток стандартного вывода сообщения в виде сериализованного Json-объекта в формате {"MessageType" : value}.

Поддерживаемые типы сообщений:

- Item – элемент собранных данных;
- Log – лог с уровнем (DEBUG, INFO, WARNING, ERROR, CRITICAL);

- Request – элемент с описанием выполненного запроса;
- Stats – текущие метрики (статистика) сборщика;
- Finish – элемент с описанием статуса завершения сбора.

Описание сообщения типа Item:

- Item – {"Item" : itemContent}, где itemContent – json-объект с допустимыми полями:
  - _url – url-адрес страницы, откуда извлечены данные;
  - _timestamp – время сбора;
  - _attachements – вложения:
    - path – путь в файловом хранилище;
    - filename – имя файла;
    - checksum – хеш файла;
  - value – собранный элемент.

Остальные типы сообщений схожи по структуре с незначительными смысловыми изменениями.

6. Архитектура системы

Общая архитектура системы представлена на рис. 1.

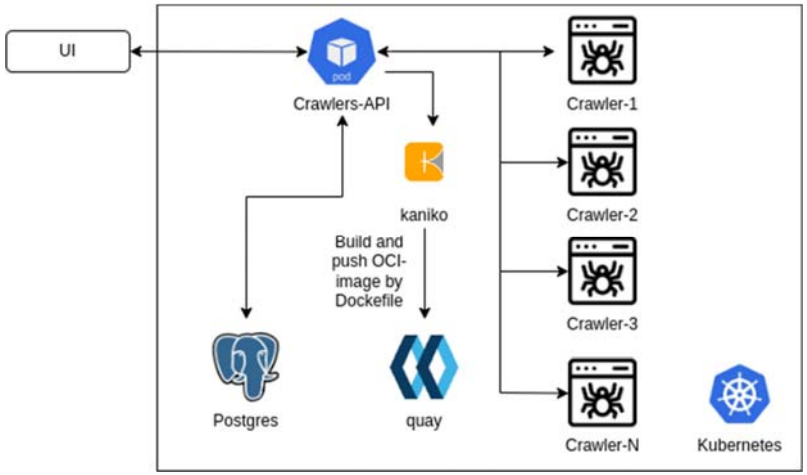

Рис. 1. Диаграмма основных компонентов системы
Fig. 1. Diagram of the main components of the system

Для реализации системы использовался язык программирования Scala.

skuber предоставляет собой полнофункциональный, высокоуровневый и строго типизированный Scala API для управления ресурсами кластера Kubernetes (такими как Pods, Services, Deployments, ReplicaSets, Ingresses и т. д.) через сервер Kubernetes REST API. Был выбран именно skuber, поскольку, по сравнению с его аналогами, является наиболее легковесным и распространённым, с хорошей документацией.

Quay – это реестр образов контейнеров, который позволяет создавать, организовывать, распространять и развертывать контейнеры. В текущем случае необходим такой репозиторий, поскольку есть потребность в хранении собранных OCI-образов проектов, с которыми далее будет вестись работа. Таким образом, каждый проект будет сохраняться отдельно от любого другого, а значит, не будет библиотек, требуемых для других проектов.

Рассматривая работу Kubernetes и API как основную часть, можно выделить общие закономерности:

- в начале каждого из реализованных методов собирается собственный шаблон задания из Kubernetes с указанием контейнера, его переменных, параметров запуска и непосредственно команды;
- каждый шаблон является основой для задания Kubernetes, которое при помощи фреймворка skuber удается собрать;
- каждое задание генерирует модуль, внутри которого разворачивается docker-контейнер и выполняются необходимые процессы;
- цикл жизни каждого пода отслеживается, и в случае его успешного завершения из Kubernetes удаляются выполненные задания, модули, а также конфигурационные файлы модулей.

Ключевой список методов, необходимый для реализации и полноценного взаимодействия с системой управления заданиями:

- createProjectVersion – создание новой версии проекта;
- getCrawlers – извлечение списка сборщиков проекта;
- cancelJob – отмена задачи сбора данных;
- scheduleJob – постановка в очередь задачи сбора данных;
- deleteProject – удаление проекта;
- deleteProjectVersion – удаление версии проекта;
- listJobs – список всех задач по проекту в разрезе статуса задачи (ожидание, в работе, выполнено).

6.1 Особенности реализации контракта загрузки проекта

Загрузка проекта происходит в несколько этапов. Сначала с пользовательского интерфейса на сервер поступает файл с проектом, который сохраняется в базе данных. Далее выполняется разархивирование проекта, извлечение необходимых файлов во временную папку. На основе путей к извлеченным данным и шаблонных Docker-файлов составляется конфигурационный файл модуля и сохраняется в Kubernetes. Далее создается контейнер со всеми необходимыми переменными, образом которого является kaniko – инструмент для сборки OCI-образов внутри Kubernetes на основе Dockerfile.

С использованием вышеперечисленных компонентов формируется задача сбора образа и размещения его в quay – репозитории собранных образов. Запуск задачи генерирует под, в котором можно отслеживать ход работы, его статус отслеживается для передачи информации о завершении работы пода. Поскольку мы не можем знать заранее, успешно ли соберется образ проекта, сможет ли последний попасть в хранилище, то данный метод реализован на Promise, которые возвращают результат только в том случае, если под получил статус «Завершен» или «Ошибка».

6.2 Загрузка проекта через постоянные тома

Второй реализованный метод работает несколько иначе, дольше, но при этом универсальнее и не имеет лимита на объем загружаемых сборщиков. Изначально создается временная директория, в которую складываются шаблонные параметры сборки OCI-образа. Затем в нее же распаковывается загружаемый архив с флагом замены в случае, если какой-либо файл уже существует. После подготовительных действий создается постоянный том (Persistance Volume Claims) с правами чтения и записи единожды и заданным размером. Создается специальный временный модуль, к которому как том добавляется ранее созданный

постоянный том. После создания постоянного тома и модуля выполняется команда, которая копирует содержимое из временной папки на сервере, который монтируется по пути /mnt.

Также был разработан специфичный для фреймворка Scrapy sh-скрипт, который используется по завершению сборки образа для проверки корректности и содержит команды:

- activate – опциональное поле, в случае если надо выполнить какие-то дополнительные действия при загрузке проекта. Вызов этой команды должен быть выполнен в рамках Dockerfile;
- list – обязательное поле, должен возвращать список загруженных сборщиков;
- schedule – обязательное поле, должен выполнять запуск сборщика, а также передавать в качестве аргументов имя сборщика, его аргументы и настройки.

6.3 Извлечение списка сборщиков

Для данного метода уже необходимо знать, по какому пути в хранилище лежит собранный несколько шагов назад образ проекта. На основе сохраненного образа строится контейнер, которому подается аргумент *list* для скрипта-входной точки, получающего список сборщиков. В случае, если загруженный проект не имеет sh-скрипта, будет использован шаблонный, нацеленный на scrapy проект. Поэтому scrapy-проекты можно загружать без sh-скриптов, но для остальных технологий сбора данных необходимо указывать, как именно выполнять команды в созданном контейнере. Составленное задание Kubernetes автоматически скачает указанную версию проекта и применит к последнему команду. Результатом считается модуль в статусе «Выполнено» или «Ошибка». Список сборщиков извлекается из потока стандартного вывода модуля.

6.4 Создание запуска

Поскольку процесс сбора сообщений может занимать продолжительное время, для пользователя успешным созданием задачи сбора данных является успешно созданное задание Kubernetes, а сама работа и обработка сообщений выполняются в фоновом режиме, незаметно для пользователя. Для удобства мониторинга за запусками все созданные задачи имеют следующий шаблон:

crawl-pid-«id проекта»-«модифицированное имя сборщика»-«номер версии»-jobid-«id задачи запуска»

При этом переданные настройки и аргументы преобразуются из пары «ключ-значение» в строку, которую может принять командная строка. Создается контейнер, которому передается аргумент *schedule* для скрипта-входной точки с именем сборщика, настройками и аргументами. На основе этого контейнера создается задание Kubernetes, которое впоследствии выполняется.

Во время работы запуска выполняется команда, которая складывает все записи из потока стандартного вывода модуля во временный файл, с которым будет вестись работа далее. Записанные в файл логи считываются и проверяются по их типу. Если записано сообщение в формате, отличному от указанного в контракте, то оно игнорируется. Если формат корректный, то сообщение дополнительно насыщается необходимой информацией и распределяется:

- Item – отправляется в Postgres для хранения;
- Log – отправляется в ElasticSearch с индексом LogsIndex;
- Request – отправляется в ElasticSearch с индексом RequestIndex;
- Stats – отправляется в ElasticSearch с индексом MetricsIndex;
- Finish – отправляется в ElasticSearch с индексом LogsIndex и пометкой как сообщение

окончания работы сборщика.

6.5 Отмена запуска

Функция отмены запуска реализована как установка значения «Отменено» в поле «Статус сбора» в базе данных. Все запуски, которым проставлено данное значение, перестают отслеживаться системой.

6.6 Список запусков

Список запусков получается путем запроса в Kubernetes тех заданий, у которых имеется пометка с ключом *projectId* и значением из списка запрашиваемых.

Правило установки статуса запускам:

- в ожидании – если у задания Kubernetes отсутствует статус, или сформированные задачей модули Kubernetes не имеют статусов «Завершено» или «Ошибка»;
- выполняется – если у задания Kubernetes есть статус и сформированные модули имеют статус «Активно»;
- завершено – если у задания Kubernetes есть статус и сформированные модули имеют статус «Завершено» или «Ошибка».

7. Экспериментальное тестирование

Прежде всего проверялась корректность работы системы манипуляциями со scrapy-сборщиками, поскольку они являются приоритетными и требование по обратной совместимости одно из важнейших. Но поскольку реализация подразумевает поддержку работоспособности сборщиков на прочих фреймворках и языках программирования, то был реализован тестовый сборщик на языке программирования Scala.

При помощи тестового сборщика[1] можно полностью промоделировать поведение любого другого сборщика. Достигается этот эффект за счет возможности детальной настройки ожидаемого результата при помощи аргументов.

Как результат, предложенное решение удовлетворяет каждому поставленному требованию.

За счет выбора архитектурного решения и системы оркестрации контейнерных приложений обеспечивается горизонтальная масштабируемость и изолированность зависимостей сборщиков.

Также были реализованы все необходимые взаимодействия в рамках жизненного цикла сборщика, продемонстрирована реализация и тесты, подтверждающие возможность работы с приложения сбора на основе других фреймворков.

8. Заключение

В работе создана методология разработки системы управления заданиями сбора данных из сети Интернет.

В процессе выполнения проекта было проведено исследование, которое включало анализ литературы и существующих решений для управления заданиями сбора данных из сети Интернет. Обзор показал, что текущие решения с открытым исходным кодом зачастую завязаны на один язык программирования и фреймворк разработки сборщиков, имеют проблемы с обеспечением изолированности и ограниченную степень масштабирования, так что они применимы только для небольших проектов по сбору данных. Платные решения лишены части проблем, но не обладают исходным кодом для дальнейших модификаций. Мы обосновали необходимость использования инструментов контейнерной оркестрации и

[1] https://github.com/vlazarew/scala_test_crawler

провели анализ существующих систем, по итогам которого для разработки выбран Kubernetes.

Для создания системы управления заданиями автоматизированного сбора данных мы определили сущности, с которыми такая система должна работать, и выделили основные элементы жизненного цикла абстрактного сборщика. Затем была описана методология для работы со сборщиками, реализованными над контейнерами, включающая спецификации для загрузки и сборки образов проектов сбора, запуска сбора и получения результатов запуска.

На основе описанной методологии реализована и внедрена в Talisman система управления заданиями автоматизированного сбора данных из сети Интернет, которая удовлетворяет требованиям горизонтальной масштабируемости, полной изолированности зависимостей сборщиков и независимости от технологий их разработки.

Список литературы / References

[1] ИСП РАН. Talisman: платформа для обработки данных. Доступно по ссылке: https://www.ispras.ru/technologies/talisman/ ISP RAS. Talisman: a data processing framework. Available at: https://www.ispras.ru/en/technologies/talisman/

[2] Anand V. Saurkar, Kedar G. Pathare, Shweta A. Gode. An Overview on Web Scraping Techniques and Tools. International Journal on Future Revolution in Computer Science & Communication Engineering, vol. 4, no. 4, 2018, pp. 363 - 367

[3] IST Research. Scrapy Cluster 1.3 Documentation. Available at: https://scrapy-cluster.readthedocs.io/en/dev/.

[4] Scrapy group. Scrapyd. Available at: https://scrapyd.readthedocs.io/en/stable/.

[5] ScrapyRT (Scrapy Realtime). Available at: https://github.com/scrapinghub/scrapyrt.

[6] Ferrit. Available at: https://github.com/reggoodwin/ferrit.

[7] Zyte. Web Scraping Cloud Hosting Data Extraction - Zyte. Available: https://www.zyte.com/scrapy-cloud/.

[8] Web Scraper Cloud. Web Scraper Cloud | Web Scraper documentation. Available: https://webscraper.io/documentation/web-scraper-cloud.

[9] Octopus Data Inc. Web Scraping Tool & Free Web Crawlers | Octoparse. Available at: https://www.octoparse.com/, 2022

[10] data-ox.com. Web Data Scraping Company | DataOx. Available at: https://data-ox.com/.

[11] The Kubernetes Authors. What is Kubernetes? Available at: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes.

[12] Docker Inc. Overview of Docker Compose. Available at: https://docs.docker.com/compose/.

[13] Docker Inc. Swarm mode overview. Available at: https://docs.docker.com/engine/swarm/.

[14] The Apache Mesos Software Foundation. Mesos Architecture. Available at: https://mesos.apache.org/documentation/latest/architecture/.

[15] Isam Mashhour Al Jawarneh, Paolo Bellavista et al. Container Orchestration Engines: A Thorough Functional and Performance Comparison. In Proc. of the IEEE International Conference on Communications (ICC), 2019, pp. 1-6

Информация об авторах / Information about authors

Владимир Александрович ЛАЗАРЕВ является студентом магистратуры кафедры системного программирования МГУ и работает в ИСП РАН. Его научные интересы включают сбор данных из открытых источников, исследование и разработку систем управления заданиями сбора данных.

Vladimir Alexandrovich LAZAREV is a master's student of the System Programming Department of MSU and also is an employee of ISP RAS. His research interests include open source data collection, research and development of data collection task management systems.

Максим Игоревич ВАРЛАМОВ – научный сотрудник ИСП РАН. Сфера научных интересов: автоматизация сбора данных из веб-ресурсов, машинное обучение для извлечения информации из Веба.

Maksim Igorevich VARLAMOV – researcher at ISP RAS. Research interests: automation of data collection from web resources, machine learning for web data extraction.

Александр Константинович ЯЦКОВ – аспирант ИСП РАН. Сфера научных интересов: сбор данных из Веба, автоматизация процесса сбора данных, фокусированный сбор данных, извлечение информации, машинное обучение.

Alexander Konstantinovich YATSKOV – PhD Student at ISP RAS. Research interests: web crawling, web data extraction, focused crawling, information extraction, machine learning.