коллект!ми

Система контроля и автоматизации задач сбора данных Коллект!Ми

Collect!Me

Пояснительная записка

Версия 1.8

2023

СОДЕРЖАНИЕ

Аннотация

Данный документ содержит в себе описание Системы контроля и автоматизации задач сбора данных Коллект!Ми. Архитектура и принцип работы Системы являются универсальными и позволяют функционировать совместно с различными источниками данных без привязки к конкретному производителю и версии.

Сокращения

В настоящем документе использованы следующие сокращения:

Сокращение	Полное наименование
БД	База данных
ИБ	Информационная безопасность
ИС	Информационная система
ОЗУ	Оперативное запоминающее устройство
ОС	Операционная система
ПО	Программное обеспечение
Система, Коллект!Ми	Система контроля и автоматизации задач сбора данных Коллект!Ми (англ. Collect!Me)
СУБД	Система управления базами данных
ТЗ	Техническое задание
ТУЗ	Технологическая учетная запись
УЗ	Учетная запись
SIEM	System Information and Event Management
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol

1 ОБЩИЕ ПОЛОЖЕНИЯ

1.1 Наименование системы

Полное наименование: Система контроля и автоматизации задач сбора данных Коллект!Ми (англ. Collect!Me).

Условное обозначение и краткое наименование: Система, Коллект!Ми.

1.2 Организация-разработчик

Организация-разработчик Системы: Общество с ограниченной ответственностью «Отечественные интеллектуальные продукты», 127282, г. Москва, вн.тер.г. муниципальный округ Северное Медведково, проезд Студёный, д. 4 к. 1, помещ. 5А/1. ИНН 9715457987.

1.3 Цели и задачи Системы

Использование Системы Коллект!Ми позволяет достичь следующие цели:

- автоматизация и ведение задач по сбору данных от источников данных, обработка данных для представления в унифицированном виде;
- автоматизация действия при выполнении условий на основе обработки данных;
- снижение трудозатрат на сбор метрик и показателей от различных источников данных.

Система Коллект!Ми предназначен для решения следующих задач:

- сбор данных ИБ и ИТ от различных источников;
- хранение собранных данных с учетом временных атрибутов, а также обогащение данных;
- автоматизация сценариев и последовательностей выполнения задач, реагирования в случае выполнения некоторых условий;
- унификация задач сбора данных и приведение к единому формату;
- контроль и управление задачами сбора данных;
- интеграция с внешними системами визуализации и отчетности.

1.4 Сведения об использованных при разработке нормативно-технических документах

При разработке Системы использовались следующие нормативно-технические документы:

- ГОСТ 34.601-90 «Автоматизированные системы. Стадии создания»;

- ГОСТ 34.201-89 «Виды, комплектность и обозначение документов при создании автоматизированных систем»;
- ГОСТ 34.603-92 «Виды испытаний автоматизированных систем»;
- ГОСТ 34.003-90 «Комплекс стандартов на автоматизированные системы. Термины и определения»;
- РД 50-34.698-90 «Автоматизированные системы. Требования к содержанию документов».

2 ОПИСАНИЕ ОБЪЕКТА АВТОМАТИЗАЦИИ

2.1 Описание объекта автоматизации

Объектом автоматизации является процесс сбора разнородных данных (метрик), поступающих в виде данных СУБД, файлов, электронных писем и т.д.

2.2 Пользователи Системы

Реализация комплекса мероприятий, связанных с эксплуатацией Системы Коллект!Ми, возлагается на персонал Системы. Описание решений по численности, квалификации и функциям персонала Системы приведено в разделе 3.3 настоящего документа.

3 ОСНОВНЫЕ ТЕХНИЧЕСКИЕ РЕШЕНИЯ

3.1 Решения по структуре Системы

Общая схема Системы Коллект!Ми представлена на рисунке (Рисунок 1).

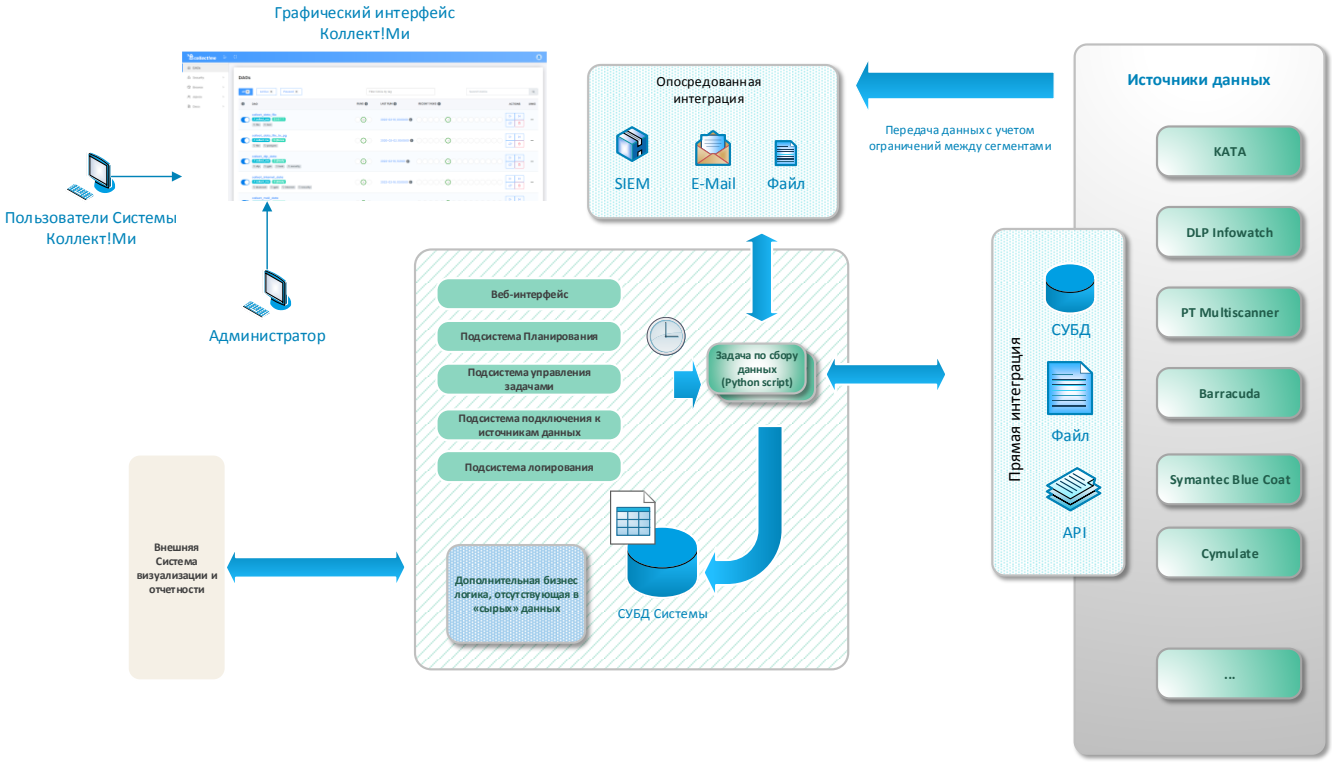

Рисунок 1

Система Коллект!Ми обеспечивает ведение задач по сбору данных от источников данных ресурсов (в качестве примера, система защиты от утечек информации, системы фильтрации данных, бухгалтерские системы, ИТ-системы и т.д.), обработку данных для представления в унифицированном виде с последующей передачей данных для хранения. По умолчанию для хранения используется СУБД Postgres.

Функции графических пользовательских интерфейсов Системы Коллект!Ми реализованы при помощи web–интерфейсов (интерфейсов взаимодействия пользователя с приложением через web–браузер). Доступ осуществляется с использованием протокола HTTP или HTTPS.

Технические решения Коллект!Ми предусматривают основной вариант развертывания, в котором все компоненты Системы находятся внутри корпоративного периметра. Передача данных за пределы корпоративного периметра не предусматривается.

Система Коллект!Ми включает в свой состав следующие логические компоненты:

- Подсистему управления заданиями;
- Подсистему планирования;
- Подсистему подключения к источникам данных;
- Подсистему логирования;
- Внутреннюю СУБД для хранения данных;
- Веб-интерфейс.

3.1.1 Подсистема управления заданиями

Подсистема управления заданиями обеспечивает возможность создания, редактирования и удаления последовательностей задач сбора данных. Задачи могут быть объединены в ациклические графы - объединение задач, которые необходимо выполнить в строго определенной последовательности согласно указанному расписанию. Такой граф выглядит как граф, не имеющий циклических зависимостей.

В качестве узлов графа выступают задачи. Это непосредственно операции, применяемые к данным, например, загрузка данных из различных источников, их сбор, очистка от дубликатов и т.д. На уровне кода задачи могут представлять собой интерпретируемый код, например: Python-функции, Bash-скрипты.

За реализацию задач отвечают операторы. Если задачи описывают, какие действия выполнять с данными, то операторы — как эти действия выполнять.

В составе Коллект!Ми присутствует встроенный набор операторов, имеется возможность добавления пользовательских операторов — за счет расширения базового класса BaseOperator.

3.1.2 Подсистема планирования

Подсистема планирования предназначена для планирования задач Коллект!Ми. Подсистема отслеживает все созданные задачи и последовательности задач в виде графов, по мере выполнения необходимых для их запуска условий. По умолчанию раз в минуту подсистема планирования анализирует пул задач и проверяет, нет ли задач, готовых к запуску.

В качестве описания параметров периодичности запуска могут быть использоваться как стандартные cron описания, так и более удобные синонимы вида @daily

3.1.3 Подсистема подключения к источникам данных

Подсистема подключения к источникам данных обеспечивает интеграцию с источниками данных ресурсов (Бухгалтерия, Почта, DLP, NGFW, SIEM, AV, OpenSource Monitoring и т.д.).

Интеграция обеспечивается за счет набора предварительно настроенных коннекторов, имеется возможность самостоятельного изменения коннекторов пользователем для поддержки специфичного источника данных.

По умолчанию обеспечивается возможность прямой интеграции (в случае открытых сетевых проходов) и косвенной интеграции.

Прямая интеграция:

- Коннекторы к типовым СУБД;
- Чтение данных из файла(ов);
- REST API.

Косвенная интеграция:

- Чтение данных с технологического файла (CSV);
- Чтение данных с технологического Email;
- Query к SIEM (по API).

3.1.4 Подсистема логирования

Подсистема логирования обеспечивает логирование информации о каждом запуске задания и задач в соответствии с указанным расписанием.

Например, если указать, что задание должно запускаться начиная с 01.03.2022 00:00:00 раз в сутки — Система Коллект!Ми будет хранить информацию о запуске экземпляров задания для следующих временных отметок: 01.03.2022 00:00:00, 02.03.2022 00:00:00, 03.03.2022 00:00:00 и так далее.

3.1.5 СУБД для хранения данных

В качестве хранилище данных может использоваться СУБД с поддержкой языка SQL. По умолчанию используется СУБД PostgreSQL.

На базе «сырых» данных в хранилище данных формируется интеллектуальный слой бизнес логики.

3.2 Решения по режимам функционирования Системы

В процессе функционирования Системы предусмотрено три режима работы:

- нормальный режим работы;
- аварийный режим работы;
- режим обслуживания и обновления.

Нормальный режим работы является основным режимом, при котором функционирование Системы Коллект!Ми осуществляется в полном объеме с заданными параметрами без нарушения работы Системы. Система Коллект!Ми функционирует по схеме 24x7 (24 часа в сутки, 7 дней в неделю) с перерывами на обслуживание. Для обеспечения нормального режима функционирования Системы Коллект!Ми должны выполняться требования и выдерживаться условия эксплуатации программного обеспечения и комплекса технических средств Коллект!Ми, указанные в эксплуатационной документации.

Аварийный режим функционирования Системы Коллект!Ми характеризуется отказом или сбоем одного, или нескольких компонент Системы. Аварийный режим функционирования, при наличии технической возможности, должен обеспечивать оповещение персонала о сбое компонентов Системы Коллект!Ми. Восстановление компонент Системы Коллект!Ми из аварийного режима в нормальный осуществляется в соответствии с порядком, установленным конечным потребителем системы, с учетом возможностей восстановления Системы Коллект!Ми.

Аварийный режим функционирования Системы Коллект!Ми влечет за собой частичное или полное нарушение возможностей сбора метрик от источников данных, и не влияет на АС, являющиеся источником данных.

Режим обслуживания и обновления предназначен для проведения профилактических работ с компонентами Системы Коллект!Ми, установки обновлений с временной остановкой Системы Коллект!Ми. В данном режиме допускается недоступность Системы Коллект!Ми в связи с проведением профилактических работ, затрагивающих программную часть Системы Коллект!Ми, а также обновлением программной или аппаратной части.

3.3 Решения по численности, квалификации и функциям персонала Системы

3.3.1 Описание ролевой модели Системы

В Системе Коллект!Ми предусмотрены следующие роли:

- Администратор;
- Администратор ИБ;
- Аналитик.

Администратор должен выполнять следующие основные обязанности:

- обеспечение эксплуатации и работоспособности общесистемного и специализированного ПО и технических средств Системы Коллект!Ми;
- настройка параметров функционирования и администрирование общесистемного и специализированного ПО и технических средств Системы Коллект!Ми (настройки даты и времени, сетевых настроек, хранилища сертификатов, интеграции с почтовым сервером, источниками данных, настроек обновления);
- взаимодействие с подразделениями, ответственными за проведение резервного копирования, контроль выполнения резервного копирования;
- установка, удаление и обновление графов, подготовленных Аналитиком;
- управление ролями;
- назначение прав пользователям.

Администратор ИБ должен выполнять следующие основные обязанности:

- контроль за защитой Системы Коллект!Ми от несанкционированного доступа к информации;
- аудит и анализ событий безопасности, регистрируемых компонентами Системы Коллект!Ми;
- просмотр перечня пользователей, их ролей и привилегий.

Аналитик должен выполнять следующие основные обязанности:

- создание заданий;
- управление подключениями;
- управление переменными;
- управление заданиями и его составными элементами (запуск, останов, просмотр статуса).

3.3.2 Описание решений по квалификации персонала

Администратор Системы Коллект!Ми должен:

- иметь высшее техническое образование и стаж работы в должности специалиста по администрированию не менее 3 лет;
- обладать практическим опытом выполнения работ по установке, настройке и администрированию программных и технических средств, применяемых в Системе.

Администратор Системы Коллект!Ми должен:

- иметь высшее техническое образование и стаж работы в должности специалиста по защите информации не менее 3 лет;
- обладать практическим опытом выполнения работ по настройке и реализации функций ИБ программных и технических средств, применяемых в Системе.

Аналитик Системы Коллект!Ми должен:

- иметь высшее техническое образование и стаж работы в должности специалиста по защите информации не менее 1 года;
- обладать практическим опытом программирования на языке Python;
- обладать практическим опытом формирования запросов к СУБД на языке SQL.

Работа персонала с Системой Коллект!Ми осуществляется на персональных компьютерах, поэтому требования к организации труда и режима отдыха при работе с ней должны устанавливаться исходя из требований к организации труда и режима отдыха при работе с этим типом средств вычислительной техники в соответствии с ТК РФ, и СанПиН.

3.4 Сведения об потребительских характеристиках Системы, определяющих ее качество

Основные потребительские характеристики – надежность и производительность.

Надежность обеспечивается за счет регулярного резервного копирования конфигурации ПО и данных на внешние носители. Резервное копирование позволяет восстановить работоспособность Системы Коллект!Ми на резервном или восстановленном оборудовании. Периодичность, методы, процедуры и средства резервного копирования и восстановления данных и конфигурации программного обеспечения Системы Коллект!Ми должны выбираться с учётом минимизации времени восстановления Системы Коллект!Ми после сбоя.

Производительность обеспечивается за счет используемых программно-технических средств. При увеличении нагрузки, производительность Системы Коллект!Ми обеспечивается путем модернизации технических средств за счет увеличения их мощности.

4 СОСТАВ ФУНКЦИЙ, РЕАЛИЗУЕМЫХ СИСТЕМОЙ

4.1 Автоматизированный сбор и обработка данных

Система Коллект!Ми является ETL решением, обеспечивает следующие возможности:

- извлечение данных из внешних источников;
- трансформацию данных с целью, чтобы они соответствовали потребностям бизнес-модели;
- загрузка данных в хранилище данных.

Системы Коллект!Ми предоставляет простой и удобный инструмент для построения, планирования и мониторинга цепочек обработки данных. Ключевой особенностью является то, что используя Python-код и встроенные функциональные блоки, можно соединить множество различных современных технологий.

4.1.1 Направленный ациклический граф

Для выполнения ETL операций в Системе Коллект!Ми используются направленные ацикличные графы (Directed acyclic graph, DAG).

DAG — это ориентированный ациклический граф, концептуальное представление серии действий или, другими словами, математическая абстракция конвейера данных (data pipeline).

Хотя оба термина, DAG и data pipeline, используются в разных сферах, они представляют собой почти идентичный механизм. DAG (или конвейер) определяет последовательность этапов выполнения в любом неповторяющемся алгоритме.

Аббревиатура DAG расшифровывается так:

- DIRECTED — направленный. В общем, если существует несколько задач (тасков), каждая из них должна иметь по крайней мере одну материнскую (предыдущую) или дочернюю (последующую) задачу. Возможно, предыдущих или последующих задач будет больше одной. (Однако важно отметить, что существуют также DAG, которые имеют несколько параллельных задач, что означает отсутствие зависимостей между параллельным задачами.)
- ACYCLIC — ациклический. Ни одна задача не может создавать данные, которые будут ссылаться на самих себя. Это может создать проблему бесконечного цикла. В DAG нет циклов.
- GRAPH — граф. В математике граф представляет собой конечный набор узлов с вершинами, соединяющими узлы. В контексте разработки данных каждый узел в графе представляет собой

задачу. Все задачи изложены в четкой структуре, с дискретными процессами, происходящими в заданных точках, и прозрачными взаимосвязями с другими задачами.

Преимущества DAG:

Конвейеры, основанные на коде, чрезвычайно динамичны. Если вы можете написать это в коде, то вы можете сделать это в своем конвейере данных.

Конвейеры на основе кода обладают большими возможностями **расширения**. Вы можете интегрироваться практически с любой существующей системой, если у нее есть API.

Конвейеры на основе кода более **управляемы**: поскольку все находится в коде, он может легко интегрироваться существующие процессы. Язык, используемый в DAG-ах - это Python.

Пример DAG (Рисунок 2, Рисунок 3):

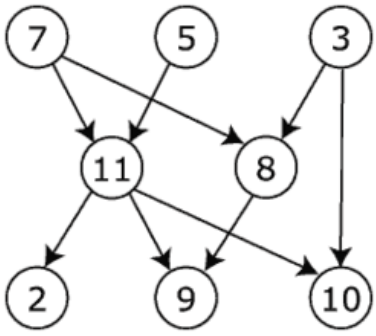

Рисунок 2

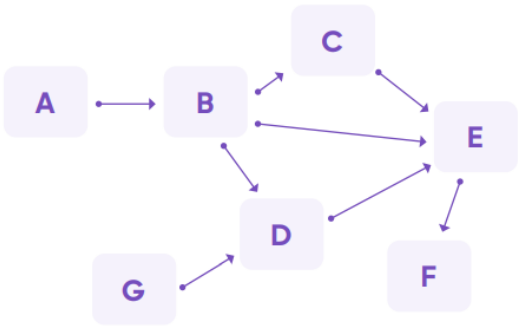

Рисунок 3

Рассмотрим приведенный (Рисунок 3) ориентированный ациклический граф. Каждая линия, исходящая из вершины, имеет определенное направление (обозначенное стрелкой), соединяя данную вершину с другим узлом.

Это ключевое качество ориентированного графа: данные **могут следовать только в одном направлении**. Например, данные могут передаваться из A в B, но никогда из B в A. Точно так же, как вода течет по трубам в одном направлении, данные должны следовать в направлении, определенном графиком. Узлы, от которых исходят данные, считаются материнскими (верхними), в то время как узлы принимающие данные считаются дочерними (нижними).

Вдобавок к тому, что данные перемещаются в одном направлении, есть еще одно свойство: узлы никогда не становятся **самореферентными**. То есть они никогда не смогут проинформировать самих себя (передать данные сами себе), так как это может создать бесконечный цикл. Таким образом, данные могут переходить от A к B, далее к C / D / E, но, оказавшись там, ни один последующий процесс никогда не сможет привести обратно к A / B / C / D / E, поскольку данные перемещаются вниз по графику. Данные, поступающие из нового источника, такого как узел G, все равно могут привести к узлам, которые уже подключены, но никакие последующие данные не могут быть переданы обратно в G. Это определяющее качество ациклического графа.

Почему эти условия должны соблюдаться для конвейеров данных? Если бы у F был обратный переток в узел D, мы бы увидели график, где D информирует E, который информирует F, который информирует D и так далее. Это создает сценарий, в котором конвейер может работать бесконечно, никогда не останавливаясь. Подобно воде, которая никогда не попадет в кран, такой цикл стал бы пустой перегонкой потока данных.

Чтобы представить этот пример в реальности, представьте, что приведенный выше DAG представляет собой такой процесс обработки данных:

- Узел A может быть кодом для извлечения данных из API;
- Узел B может быть кодом для анонимизации данных и удаления любого IP-адреса;
- Узел D может быть кодом для проверки отсутствия повторяющейся по ID записи;
- Узел E может помещать эти данные в базу данных;
- Узел F может выполнять SQL-запрос к новым таблицам для обновления информационной панели (дашборда).

В Системе Коллект!Ми DAG - это ваш конвейер данных. Он представляет собой набор инструкций, которые должны быть выполнены в определенном порядке. Это выгодно для управления данными (data orchestration) по нескольким причинам:

- Зависимости в DAG гарантируют, что ваши задачи обработки данных каждый раз выполняются в одном и том же порядке, это создает надежный процесс для вашей повседневной инфраструктуры обработки данных.
- Графический компонент DAG позволяет визуализировать зависимости в пользовательском интерфейсе Системы Коллект!Ми.
- Поскольку каждый путь в DAG является линейным, легко разрабатывать и тестировать конвейеры данных на соответствие ожидаемым результатами.

DAG начинается с задачи (task), написанной на Python. Вы можете думать о задачах как об узлах вашего DAG: каждый из них представляет собой одно действие, и оно может зависеть как от вышестоящих, так и от нижестоящих задач.

Задачи упаковываются в операторы, которые являются строительными блоками, определяющими поведение входящих в них задач. Например, задача оператора **Python Operator** выполнит функцию Python.

На следующей диаграмме (Рисунок 4) показано, как эти концепции работают на практике. Как видите, написав один DAG-файл на Python, вы можете начать определять сложные взаимосвязи между данными и действиями.

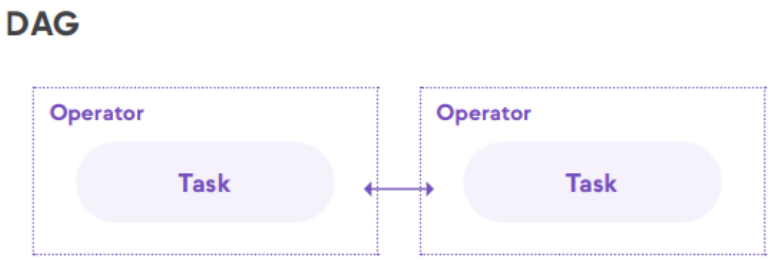

Рисунок 4

Внутри оператора содержится задача. Например, задача оператора Python Operator выполнит функцию Python.

С помощью DAG можно например:

- Извлечь данные из хранилища данных и загрузить их в промежуточное хранилище.
- Выполнить преобразование данных, в зависимости от используемых вами данных.
- Сохранить результаты предыдущего действия в базе данных.
- Отправить информацию обо всем процессе в различные системы сбора метрик и отчетности.

Оркестрация данных связывает DAG вместе, организуя поток данных и оповещая в случае сбоев. Контроль за сквозным жизненным циклом данных позволяет поддерживать взаимозависимость во всех системах, что жизненно важно для эффективного управления данными.

4.1.2 Компоненты задания (DAG)

Система Коллект!Ми включает в себя ряд структур, которые позволяют определять DAG в коде.

- Операторы являются строительными блоками. Операторы содержат логику того, как данные обрабатываются в конвейере. Существуют разные Операторы для разных типов работы: некоторые Операторы выполняют общие типы кода, в то время как другие предназначены для выполнения очень специфических типов работ. Мы рассмотрим различные типы операторов в главе Операторы.
- Задача (таск, task) — это экземпляр Оператора. Для того чтобы Оператор мог совершить работу в контексте DAG, он должен быть создан с помощью Задачи. Другими словами, вы используете Задачи для конфигурации важного контекста для вашей работы, в том числе при ее выполнении в вашем DAG.

  Задачи — это узлы в DAG. DAG — это группа Задач, которые были скомпонованы для последующего выполнения направленным, ациклическим образом.
- Выполнение Задачи в реальном времени называется task instance - Экземпляром Задачи (это также принято называть task run - Выполнением Задачи).
- DAG run — это однократное конкретное выполнение DAG.

  Если Экземпляр Задачи - это выполнение Задачи, то DAG run, Запуск DAG — это просто экземпляр полного DAG, который был запущен или выполняется в данный момент. На уровне кода DAG становится Запуском DAG, как только у него появляется Дата исполнения, execution_date. Как и в случае с экземплярами задач, информация о каждом Запуске DAG регистрируется в базе метаданных Системы.

4.1.3 Планирование и расписания

Одной из фундаментальных особенностей Системы Коллект!Ми является возможность планировать задания. Можно планировать расписание, задав schedule_interval с выражением cron, timedelta.

Расписания позволяют пользователям создавать свои собственные пользовательские расписания с помощью Python, эффективно устраняя ограничения cron. С помощью расписаний теперь вы можете запланировать запуск DAG в любое время для любого условия выбора.

Ключевой особенностью Системы Коллект!Ми является то, что запуски DAG являются атомарными, **идемпотентными** (то есть не влияющими друг на друга) элементами, и планировщик по умолчанию проверяет время жизни DAG (от начала до конца / текущего момента, по одному интервалу за раз) и запускает DAG для любого интервала, который не был запущен (или был очищен). Эта концепция называется Catch up («наверстать упущенное»).

Следующие параметры являются производными от концепций, описанных выше, и важны для обеспечения выполнения вашего DAG в нужное время:

- data_interval_start: объект datetime, определяющий начальную дату и время интервала данных. Расписание DAG будет возвращать этот параметр для каждого запуска DAG. Этот параметр либо создается автоматически, либо может быть задан пользователем при реализации пользовательского расписания.
- data_interval_end: объект datetime, определяющий дату и время окончания интервала данных. Расписание DAG вернет этот параметр для каждого запуска DAG. Этот параметр либо создается автоматически, либо может быть задан при реализации пользовательского расписания.
- schedule_interval: параметр, который можно задать на уровне DAG, чтобы определить, когда этот DAG будет запущен. Этот аргумент принимает выражения cron или объекты timedelta (подробнее об этом см. следующий раздел).
- timetable: параметр, который может быть установлен на уровне DAG для определения его расписания (пользовательское или встроенное). Для каждого DAG должно быть определено либо timetable, либо schedule_interval, но не оба вместе.
- start_date: первая дата, когда будет выполнен ваш DAG. Этот параметр обязателен для того, чтобы ваш DAG был поставлен в расписание.
- end_date: последняя дата, когда будет выполнен ваш DAG. Этот параметр является необязательным.

Допускается установка интервала расписания в следующем виде:

- Выражение Cron. Допускает передать любое выражение cron в виде строки параметру schedule_interval в вашем DAG. Например, если вы хотите запланировать свой DAG в 4:05 утра каждый день вы бы использовали schedule_interval='5 4 * * * '.
- Представление Cron. Можно использовать представления cron для общих базовых расписаний. Например, schedule_interval='@hourly' запланирует запуск DAG в начале каждого часа. Если ваш DAG не должен выполняться по расписанию и будет запускаться только вручную или внешне другим процессом, вы можете установить schedule_interval=None.

- Timedelta. Если вы хотите запланировать свой DAG на определенную частоту (ежечасно, каждые 5 минут и т.д.), а не в определенное время, вы можете передать объект timedelta в расписание. Например, schedule_interval=timedelta(minutes=30) будет запускать DAG каждые тридцать минут, а schedule_interval=timedelta(days=1) будет запускать DAG каждый день.

4.1.4 Элементы задания

Задание представляет собой Python скрипт, в котором присутствуют обязательные элементы:

- Переменная DAG;
- Операторы.

Перечень доступных операторов:

- airflow.operators.bash
- airflow.operators.bash_operator
- airflow.operators.branch
- airflow.operators.branch_operator
- airflow.operators.check_operator
- airflow.operators.dagrun_operator
- airflow.operators.datetime
- airflow.operators.docker_operator
- airflow.operators.druid_check_operator
- airflow.operators.dummy
- airflow.operators.dummy_operator
- airflow.operators.email
- airflow.operators.email_operator
- airflow.operators.empty
- airflow.operators.gcs_to_s3
- airflow.operators.generic_transfer
- airflow.operators.google_api_to_s3_transfer
- airflow.operators.hive_operator
- airflow.operators.hive_stats_operator
- airflow.operators.hive_to_druid
- airflow.operators.hive_to_mysql
- airflow.operators.hive_to_samba_operator
- airflow.operators.http_operator

- airflow.operators.jdbc_operator
- airflow.operators.latest_only
- airflow.operators.latest_only_operator
- airflow.operators.mssql_operator
- airflow.operators.mssql_to_hive
- airflow.operators.mysql_operator
- airflow.operators.mysql_to_hive
- airflow.operators.oracle_operator
- airflow.operators.papermill_operator
- airflow.operators.pig_operator
- airflow.operators.postgres_operator
- airflow.operators.presto_check_operator
- airflow.operators.presto_to_mysql
- airflow.operators.python
- airflow.operators.python_operator
- airflow.operators.redshift_to_s3_operator
- airflow.operators.s3_file_transform_operator
- airflow.operators.s3_to_hive_operator
- airflow.operators.s3_to_redshift_operator
- airflow.operators.slack_operator
- airflow.operators.smooth
- airflow.operators.sql
- airflow.operators.sql_branch_operator
- airflow.operators.sqlite_operator
- airflow.operators.subdag
- airflow.operators.subdag_operator
- airflow.operators.trigger_dagrun
- airflow.operators.weekday

Подробную информацию по операторам можно получить по ссылке

https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html

4.1.5 Пример задания

Пример задания представлен в приложении (**Приложение А Пример**).

4.1.6 Особенности использования операторов

Для логического разделения элементов DAG целесообразно использовать «пустые» (dummy) операторы:

```
start = DummyOperator(task_id='Start')

end   = DummyOperator(task_id='End')
```

Такое логическое деление позволяет улучшить представление задания, например, (Рисунок 5):

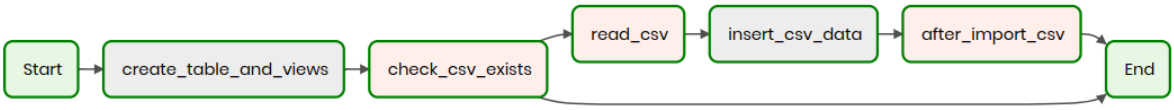

Рисунок 5

При использовании оператора взаимодействия с СУБД (например, PostgresOperator), удобно использовать ссылку на запускаемый SQL файл, а не строковое представление SQL, например:

```
create_table_and_views = PostgresOperator(

    task_id="create_table_and_views",

    postgres_conn_id=db_conn_id,

    sql=sql_create,

)
```

Для оператора ветвления BranchOperator необходимо передать ссылку на Python функцию, возвращающую в зависимости от выполнения условия текст последующей ветки задачи:

```
def py_check_csv_exists():

    import os

    if os.path.isfile(dir_csv + '/' + csv_file):
```

```
            return 'read_csv'
        else:
            return 'End'
    branch_task = BranchPythonOperator(
        task_id='check_csv_exists',
        python_callable=py_check_csv_exists)
```

Возможно передавать результаты выполнения одной задачи для другой задачи, с помощью механизма, называемого X-Com. Для этого необходимо сохранить результаты в виде переменной.

В последующем задании можно использовать эту переменную в следующем формате:

```
def py_read_csv(ti):
    sql_ins = 'SQL CODE HERE ….'
    return sql_ins
read_csv = PythonOperator(
                    task_id='read_csv',
                    python_callable=py_read_csv)
insert_csv_data = PostgresOperator(
                task_id="insert_csv_data",
                postgres_conn_id=db_conn_id,
                sql= "{{task_instance.xcom_pull('read_csv')}}")
```

4.1.7 Переменные

В Системе Коллект!Ми используется понятие переменных. Переменные задаются с использование веб-интерфейса, а в дальнейшем используются в заданиях (DAG).

За счет этого обеспечивается гибкость и лаконичность: единожды создав DAG, в дальнейшем не потребуется доработка Python скрипта, а только редактирование переменной с использованием веб интерфейса.

Управление переменными выполняется в разделе Переменные (Рисунок 6).

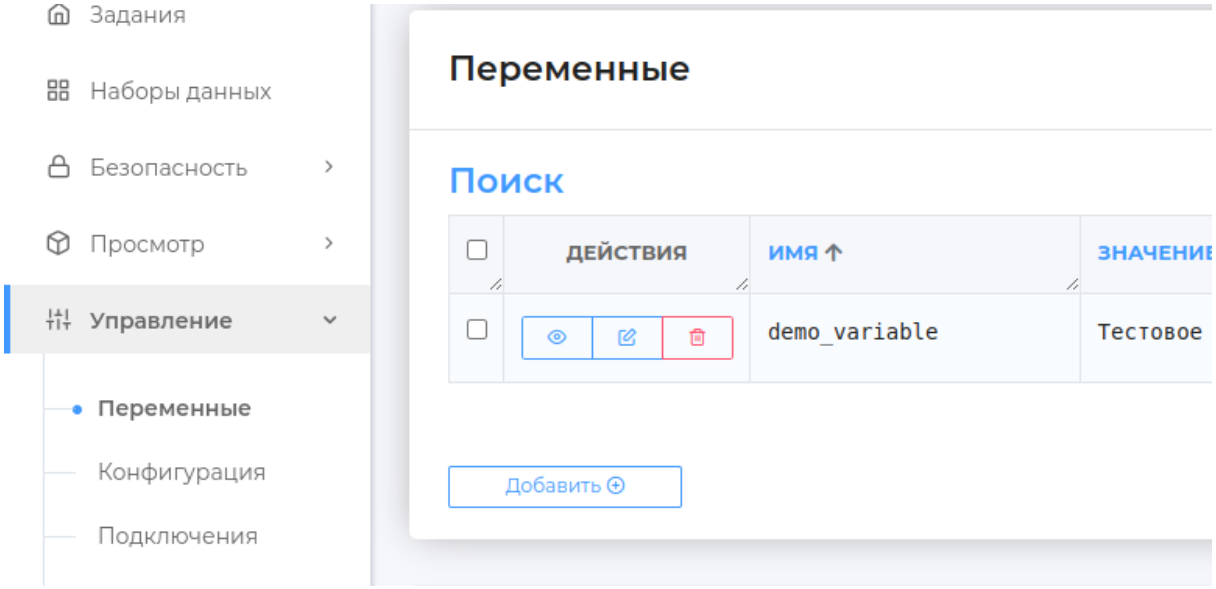

Рисунок 6

Для обращения к переменной необходимо в DAG воспользоваться конструкцией вида:

```
from airflow.models import Variable

# получение значения переменной

foo = Variable.get("foo")
```

Также информацию по переменным можно получить на сайте https://airflow.apache.org/docs/apache-airflow/stable/concepts/variables.html

4.1.8 Подключения

В Системе Коллект!Ми используется понятие подключений. Подключения являются ссылками на набор данных для подключения к СУБД, SSH, почтовым ящикам и т.д. Параметры подключения создаются с использование веб-интерфейса, а в дальнейшем используются в заданиях (DAG).

Например, для подключения к СУБД PostgreSQL необходимо создать подключение с некоторым id (например, db_conn_id).

При обращении к СУБД достаточно будет указать идентификатор подключения:

```
insert_csv_data = PostgresOperator(
                  task_id="insert_csv_data",
                  postgres_conn_id=db_conn_id,
                  sql= "{{task_instance.xcom_pull('read_csv')}}")
```

4.1.9 Плагины

В Системе Коллект!Ми используется понятие плагинов. Плагины используются для расширения функциональности взаимодействия с источниками данных. Примером плагина является сбор данных из файлов CSV по протоколу IMAP.

Плагины располагаются в директории /opt/collect-me/plugins

5 РЕШЕНИЯ ПО ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

5.1 Аутентификация и авторизация пользователей

Все пользователи Системы Коллект!Ми для доступа к функциональным возможностям Системы Коллект!Ми проходят процедуры идентификации, аутентификации и авторизации. Пользователю, не прошедшему аутентификацию, доступ к Системе запрещается.

Доступ к веб-консоли Системы Коллект!Ми может быть осуществлен под доменными УЗ. Для этого в Системе реализована интеграция со Службой каталогов Active Directory. При попытке доступа пользователей к веб-консолям Системы Коллект!Ми происходит отправка запроса на аутентификацию в службу каталогов Active Directory по протоколу LDAP (также поддерживается LDAP over SSL). Пароли пользователей домена в Системе Коллект!Ми не хранятся.

Предусмотрено время действия сессии к веб-консолям Системе Коллект!Ми, при повторном входе в Систему Коллект!Ми выполняется повторная аутентификация и авторизация пользователя.

В Системе Коллект!Ми предусмотрены следующие механизмы управления учетными записями: создание, активация, блокирование, назначение и изменение прав, удаление учетных записей.

Доступ к функциональным возможностям Системе Коллект!Ми предоставляется на основании ролевой модели, описание приведено в п. 3.3.1.

В Системе Коллект!Ми предусмотрено блокирование сеанса доступа после установленного времени бездействия (timeout), а также по команде пользователя.

При использовании доменной аутентификации, смена паролей в Системе Коллект!Ми не осуществляется.

Чувствительная информация (пароли подключений и данных переменных) в открытом виде не хранится. Для хранения указанной информации в кодированном виде используется Python библиотека Fernet.

5.2 ТУЗ для сбора данных

Для проведения сбора данных от источников могут использованы технологические учетные записи (СУБД, почтового сервера и т.д.).

Данные для ТУЗ вводятся однократно в Системе Коллект!Ми, пароли в открытом виде не хранятся.

5.3 Аутентификация при взаимодействии компонентов Системы

Взаимодействие компонентов Системы Коллект!Ми осуществляется путем обращений к СУБД PostgreSQL (порт по умолчанию 5432 TCP). Поддерживается режим SSL.

5.4 Ограничение программной среды

Развертывание компонент Системы Коллект!Ми выполняется на базе ОС семейства Linux (по умолчанию – ОС Debian). Функционирование компонент выполняется в ограниченном виртуальном окружении (virtual environment), реализованном средствами языка Python. На все файлы Системы Коллект!Ми в ОС устанавливаются минимально необходимые права доступа и владельцы (пользователь-группа collect-me:collect-me).

5.5 Регистрация событий

В Системе Коллект!Ми осуществляется регистрация и аудит системных событий путем записи syslog сообщений.

Связанная с каждой записью информация включает: дату; объект; внесенные изменения (если применимо); сообщения, связанные с действием, пользователя, который инициировал действие; IP-адрес, с которого было выполнено подключение.

Дополнительно в Системе Коллект!Ми присутствует логирование пользовательских событий, события сохраняются в СУБД PostgreSQL.

Доступ к функциям управления механизмами регистрации событий (аудита) предоставляется только администраторам Системы Коллект!Ми.

5.6 Антивирусная защита

Компоненты Системы Коллект!Ми размещаются на ОС, предоставляемых конечным пользователем, без использования доступа к сети Интернет. В качестве мер по противодействию вредоносному ПО на сервер Системы Коллект!Ми может устанавливается антивирусное ПО (например, с антивирусом Касперского).

ПРИЛОЖЕНИЕ А ПРИМЕР ЗАДАНИЯ

```
from datetime import datetime, timedelta

# импорт DAG
from airflow import DAG

# импорт оператора
from airflow.operators.bash import BashOperator
with DAG(
    'tutorial',
    # установка параметров
    default_args={
        'depends_on_past': False, # не зависеть от прошлого
        'email': ['test@example.com'],
        'email_on_failure': False,
        'email_on_retry': False,
        'retries': 1,
        'retry_delay': timedelta(minutes=5),
        # 'queue': 'bash_queue',
        # 'pool': 'backfill',
        # 'priority_weight': 10,
        # 'end_date': datetime(2016, 1, 1),
        # 'wait_for_downstream': False,
        # 'sla': timedelta(hours=2),
        # 'execution_timeout': timedelta(seconds=300),
        # 'on_failure_callback': some_function,
        # 'on_success_callback': some_other_function,
        # 'on_retry_callback': another_function,
        # 'sla_miss_callback': yet_another_function,
        # 'trigger_rule': 'all_success'
    },
    description='Пример DAG',
    schedule_interval=timedelta(days=1),
    start_date=datetime(2021, 1, 1),
    catchup=False,
    tags=['example'],
) as dag:

    # создаем три простых Bash оператора t1, t2 и t3.

    # t1 – запуск команды date
    t1 = BashOperator(
        task_id='print_date',
        bash_command='date',
    )

    # t2 – запуск команды sleep 5
```

```
t2 = BashOperator(
    task_id='sleep',
    depends_on_past=False,
    bash_command='sleep 5',
    retries=3,
)

# t3 – запуск команды hostname

t3 = BashOperator(
    task_id='templated',
    depends_on_past=False,
    bash_command='hostname',
)
# задаем последовательность выполнения
t1 >> [t2, t3]
```

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ докум.	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					