МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

УТВЕРЖДАЮ
Зав.кафедрой,
к. ф.-м. н.
_____ А. С. Иванов

ОТЧЕТ О ПРАКТИКЕ

студента 4 курса 411 группы факультета КНиИТ
Вязкова Андрея Андреевича

вид практики: преддипломная
кафедра: математической кибернетики и компьютерных наук
курс: 4
семестр: 8
продолжительность: 4 нед., с 30.04.2020 г. по 27.05.2020 г.

Руководитель практики от университета,

доцент, к. т. н. _____ В. М. Соловьев

Руководитель практики от организации (учреждения, предприятия),

к. ф.-м. н _____ С. В. Миронов

Тема практики: «Построение архитектуры ETL процесса и ее программная реализация»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Большие данные являются популярным трендом в современном IT. С помощью них можно прогнозировать события, угадывать поведение пользователя, обнаруживать мошенническую активность. Большие данные используются в таких отраслях как: финансовые услуги, здравоохранение, физика.

Однако не всегда данные могут быть однородными и готовыми для дальнейшего анализа. Чтобы они были готовы их необходимо предварительно обработать. Подготовка данных для дальнейшего анализа называется ETL процессом от английского Extract — извлечение, Transform — трансформация, Load — загрузка. В данной работе показан пример построения архитектуры этого процесса с помощью современных инструментов в облаке Amazon Web Services.

1 Большие данные

1.1 Основные понятия

Большие данные — обозначение структурированных и неструктурированных данных огромных объёмов и значительного многообразия, эффективно обрабатываемых горизонтально масштабируемыми программными инструментами.

Термин «Большие Данные» **??** вызывает множество споров, многие полагают, что он означает лишь объем накопленной информации, но не стоит забывать и о технической стороне. К данной сфере именно относится обработка и хранение большого объема информации, для которого традиционные способы являются неэффективными, а также сервисные услуги. В таблице 1 представлена сравнительная характеристика традиционной базы данных и базы больших данных.

Таблица 1 – Сравнительный анализ традиционной базы данных и базы Больших данных

Характеристика	Традиционная БД	База больших данных
Объем информации	От гигабайт до терабайт	От петабайт до эксабайт
Способ хранения	Централизованный	Децентрализованный
Структурированность данных	Структурирована	Неструктурирована или полуструктурирована
Модель хранения и обработки данных	Вертикальная модель	Горизонтальная модель
Взаимосвязь данных	Сильная	Слабая

Большие данные характеризуются следующими признаками:

— **Volume** — объем, накопленная база данных представляет собой большой объем информации, который трудно хранить и обрабатывать традиционными способами;

— **Velocity** — скорость, данный признак указывает как на увеличивающуюся скорость накопления данных, так и на скорость обработки данных, в последнее время стали более востребованы технологии обработки данных в реальном времени.

— **Variety** — многообразие, возможность одновременной обработки струк-

турированной и неструктурированной разноформатной информации. Главное отличие структурированной информации – это то, что она может быть классифицирована. Неструктурированная информация включает в себя видео, аудио файлы, свободный текст, информацию, поступающую из социальных сетей. Данная информация нуждается в комплексной обработке для дальнейшего анализа.

— **Veracity** — достоверность, насколько точны полученные данные? **??**

1.2 Отрасли применения

Большие Данные получили широкое распространение во многих отраслях. Их используют в здравоохранении, телекоммуникациях, торговле, логистике, в финансовых компаниях, а также в государственном управлении. Ниже представлены примеры их применения:

— Финансовые услуги — большие данные дают возможность проанализировать кредитоспособность заемщика. Внедрение технологий Больших Данных позволит сократить время рассмотрения кредитных заявок. С помощью Больших Данных можно проанализировать операции конкретного клиента и предложить подходящие именно ему банковские услуги;

— Телекоммуникации — в этой отрасли большие данные получили популярность у сотовых операторов. Операторы сотовой связи наравне с финансовыми организациями имеют одни из самых объемных баз данных, что позволяет им проводить наиболее глубокий анализ накопленной информации. Главной целью анализа данных является удержание существующих клиентов и привлечение новых;

— Горнодобывающая и нефтяная промышленности — большие данные используются как при добыче полезных ископаемых, так и при их переработке и сбыте. Предприятия могут на основании поступившей информации делать выводы об эффективности разработки месторождения, отслеживать график капитального ремонта и состояния оборудования, прогнозировать спрос на продукцию и цены;

1.3 Технологии Больших данных

Для сбора и обработки больших данных используются как известные технологии и концепции, так и достаточно новые. Вот некоторые из них:

— **SQL** (Structured Query Language) — структурированный язык запросов для работы с базами данных. С его помощью можно манипулировать данными, а за управление данными отвечает движок базы данных и система управления базами данных;
— **NoSQL** (Not Only SQL) — совокупность подходов, направленных на создание базы данных, отличной от реляционной. Такой подход удобно использовать при меняющейся структуре данных;
— **MapReduce** — модель программирования распределенных вычислений над большими объемами данных, представленная компанией Google в 2004 году. Весь процесс делится на две фазы: map и reduce. Все вычисления производятся на кластере из главного узла (master node) и рабочих узлов (worker nodes) Каждая из фаз принимает на вход и отдает список пар ключ/значение. Фаза map принимает список, главный узел разбивает этот список на части отправляет на каждый рабочий узел. Каждый рабочий узел применяет заранее написанную функцию к каждому элементу такого списка. Фаза reduce группирует значений по ключам и отправку результатов на главный узел. Также между этими фазами есть промежуточная фаза, называемая shuffle. На этой фазе данные перераспределяются между рабочими узлами на основе ключей, таким образом, чтобы все данные по одному ключу лежали на одном рабочем узле.
— **Apache Spark** — фреймворк для быстрых кластерных вычислений, разработанный в Университете Беркли и переданный в фонд Apache. Распространяется по открытой лицензии. Spark использует парадигму резидентных вычислений. Все данные хранятся и обрабатываются в оперативной памяти. Использует ленивые вычисления. Основной концепцией в Spark является RDD (Resilient Distributed Dataset)**??** — неизменяемая коллекция объектов, которую можно обрабатывать параллельно. RDD создается путем загрузки внешнего набора данных или распределения коллекции из основной программы. RDD поддерживает 2 типа операций**??**: трансформации — операции над RDD, результатом которых является новый RDD и действия — операции, которые возвращают значения вычислений над RDD. Ядро Spark состоит из высокоуровневых библиотек, таких как: Spark SQL — обертка над SQL, позволяю-

щаяя делать запросы в БД из кода, Spark Streaming — библиотека для потоковой обработки данных в реальном времени, MLlib — библиотека машинного обучения, GraphX — это библиотека для манипуляций над графами и выполнения с ними параллельных операций. Spark имеет программные интерфейсы на языках Scala, Java, Python, R.

Также существуют сервисы которые предоставляют готовые услуги по предоставлению баз данных и вычислительных ресурсов. Такие компании как Amazon, Google и Microsoft предлагают подобные решения. Подробнее об этих решениях будет описано в разделе 2.

2 ETL процесс. Его задачи и технологии

ETL (Extract, Transform, Load) — одна из главных процедур копирования данных из одного или нескольких источников в конечную систему. Данные в конечной системе имеют общий структурированный вид. Термин набрал популярность в 1970-х годах и преимущественно используется при построении хранилищ данных.**??**.

ETL процесс состоит из трех фаз. Ниже представлено описание каждой из них**??**:

— Extract — извлечение данных из различных источников. Источниками могут выступать: результаты работы программ, логи этих программ, копии таблиц базы данных, любой внешний набор данных;

— Transform — выполнение преобразований над данными, их фильтрация, группировка и агрегация. На этом этапе сырые данные превращаются в готовый для анализа датасет;

— Load — загрузка обработанных данных в место конечного использования, например в хранилище данных. Эти данные могут быть использованы конечными пользователями или их можно подать на вход другому ETL процессу.

ETL придает данным значительную ценность. Это не просто копирование данных из исходного источника в хранилище данных. ETL решает такие проблемы как:

— Удаляет ошибки и исправляет недостающие данные;

— Настраивает данные из нескольких источников для совместного использования:

— Структурирует данные для использования конечными инструментами:

2.1 Технологии для решения задач ETL

Так как ETL — весьма комплексная структура, то к ней выдвигаются определенные требования.

Хранилища данных должны иметь копию исходных данных, подтверждение транзакций, которые изменили данные, а также подтверждение безопасности копий данных с течением времени. Исходные данные должны быть качественными. Над некачественными данными придется проводить слишком много манипуляций, чтобы они были пригодны для дальнейшего исполь-

зования в ETL. Поэтому необходимо производить профилирование данных. Профилирование данных - это систематическое исследование качества, объема и контекста источника данных, позволяющее построить ETL процесс. Если источник данных содержит некачественные данные,то для его пригодности может потребоваться:

— Удаление некоторых полей полностью;
— Автоматическая замена поврежденных значений;
— Разработка нормализованного представления данных;

Управление и мониторинг процесса ETL — очень важно понимать каким образом будут строиться ETL пайплайны (от англ. pipeline — трубопровод). Будет ли разработчик конструировать их через пользовательский интерфейс или же писать код? Как хорошо хорошо инструмент реализует мониторинг и оповещение об ошибках? Реализован ли процесс перезапуска в случае ошибки? Существует ли возможность заново обработать данные, которые уже были обработаны?

Обработка данных — будет ли использован JVM-ориентированный язык (например Java или Scala) или будет использован SQL (например Hive). Плюсы первого подхода заключаются в использовании парадигмы MapReduce, а также в том, что с помощью этого подхода удобно писать пользовательские функции, минусы подхода в том, что придется изучать один из языков. Во втором подходе весь процесс происходит вокруг SQL таблиц, что весьма удобно. Минусы подхода заключаются в том, что для того чтобы написать пользовательские функции придется задействовать сторонний язык программирования.

В данном разделе будут описаны технологии и инструменты для решения и реализации той или иной задачи.

2.1.1 Хранилища данных

Amazon Redshift?? — сервис управления хранилищами данных в облаке, представленный компанией Amazon в 2012 году. Представляет собой кластер из узлов, который состоит из узла@=лидера и одного или нескольких вычислительных узлов. Тип и количество вычислительных узлов, которые нужны, зависят от размера данных, количества запросов, которые будут выполняться, и требуемой производительности выполнения запроса. Позволяет добавляет и удалять узлы без остановки кластера. Может хранить объ-

емы данных от сотен гигабайт до нескольких сотен петабайт. Работает на измененном движке PostgreSQL. Amazon Redshift позволяет делать резервные копии, как в ручном так и в автоматическом режиме. Также имеется шифрование данных и защищенный доступ по SSL. Данные хранятся на каждом узле в блоках, которые называются срезами, используется колоночное хранение. Redshift **??** использует архитектуру MPP (Massively Parallel Processing), разбивая большие наборы данных на куски, которые назначаются срезами в каждом узле. Быстрая производительность достигается параллельной обработкой каждого среза на узле. Главный узел объединяет результаты и возвращает их клиентскому приложению.

Apache Hive?? — хранилище данных создан компанией Facebook в 2010 году, затем передан фонду Apahe под открытой лицензией. Входит в экосистему Hadoop. Имеет собственный SQL диалект, называемый HiveQL. HiveQL имеет SQL запросы, выполняемые в Hive конвертируются в MapReduce программы. Есть возможность создания пользовательских функций. Данные таблиц хранятся в HDFS. Поддерживаются следующие форматы:

1. Текстовый файл — данные располагаются в строках, каждая строка является записью. Строки заканчиваются символом новой строки в UNIX формате.
2. Последовательные файлы — кодируются как ключ и значение для каждой записи. Записи хранятся в двоичном формате и, следовательно, занимают меньше места, чем текстовый формат:
3. Колоночный формат (RCFile, Apache Parquet) — позволяет хранить значения столбцов рядом друг с другом и все метаданные об этой таблице:
4. Avro файлы — формат кодирует схему своего содержимого непосредственно в файле, что позволяет пользователю самостоятельно сохранять сложные объекты. Avro не только формат файла, но и структура сериализации и десериализации;
5. Файлы ORC - это формат файла столбцов Optimized Row, который обеспечивает высокоэффективный способ хранения данных Hive и преодолевает ограничения других форматов файлов Hive.
6. Пользовательские форматы файлов;

Google BigQuery?? — хранилище данных, разработанное компанией Google в 2011 году. Архитектура BigQuery не требует настройки серверов.

В качестве движка для выполнения запросов используется Dremel — разработанный в Google и позволяющий сканировать миллиарды строк в секунду. Dremel использует архитектуру массивных параллельных запросов для сканирования данных в базовой системе управления файлами Colossus**??**. Colossus распределяет файлы на блоки по 64Мб среди множества вычислительных ресурсов, называемых узлами, которые сгруппированы в кластеры. Dremel использует колоночную структуру данных, аналогичную Redshift. Древовидная архитектура отправляет запросы тысячам машин за считанные секунды.

2.1.2 Планировщики ETL процессов

AWS Glue?? — бессерверный сервис для построения задач ETL, представленный компанией AWS в 2017 году, состоит из:

- централизованное хранилище метаданных хранилищ данных (AWS Data Catalog);
- подсистему ETL, автоматически генерирующую код на Python или Scala, кото- рый описывает трансформацию данных;
- планировщик заданий;
- подсистему мониторинга и перезапуска.

Основное назначение сервиса — построение системы доставки данных из различных источников в централизованное хранилище данных.

Архитектура AWS Glue представлена на рисунке 1 и состоит из следующих компонентов:

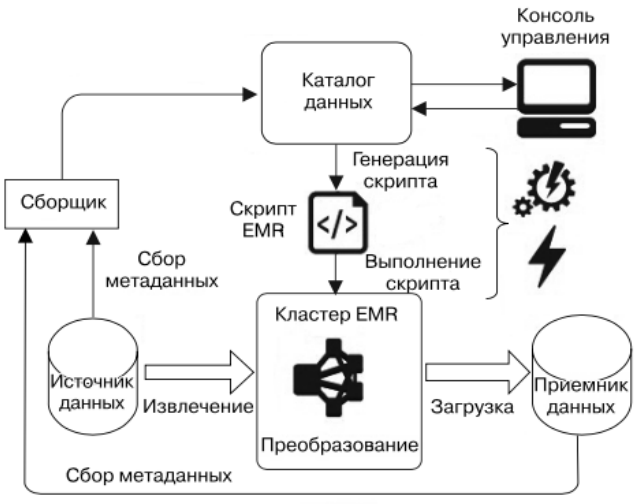

Рисунок 1 – Архитектура AWS Glue

— Выполняемые задания (jobs) — непосредственно рабочие процессы, производящие требуемое извлечение, трансформацию и загрузку данных из источников в назначения;
— Сборщик, или краулер (crawler) — программа, подключаемая к хранилищам данных, которая используется для наполнения каталога данных метаданными источников. В каталоге сборщику указывается хранилище данных, и он, определив схему источника, создает в каталоге соответствующий объект, называемый таблицей, поскольку все источники данных так или иначе могут быть описаны как таблицы. Вдобавок к описанию таблицы сборщик извлекает дополнительные метаданные, необходимые для построения сценария выполняемого задания ETL;
— Классификатор (classifier) — часть сборщика, определяющая схему данных источника. Поддерживает источники типа JSON, CSV, AVRO и XML. Кроме того, имеет возможность работать с реляционными базами данных, поддерживающими протокол JDBC;
— Сценарий трансформации данных — сценарий, выполняющий задачу трансформации и копирования, генерируемый AWS Glue или предоставляемый пользователем. Для описания задач поддерживаются языки Python и Scala.
— Триггер (trigger) — встроенный механизм запуска задания, работающий по расписанию или на основе перехваченного события. При срабатывании триггера происходит основной процесс ETL: на внутренних вычислительных ресурсах, содержащих фреймворк Apache Spark, выполняется сценарий трансформации данных, сгенерированный AWS Glue или предоставляемый пользователем.
— Сервер Notebook — интерактивный редактор кода, позволяющий писать программы на PySpark;
— Каталог данных (data catalog) — централизованное хранилище данных, которое содержит описания таблиц, сгруппированные в базы данных, выполняемых заданий и пр.

Apache Airflow?? — система управления рабочими процессами, созданная в компании Airbnb в 2014 году. В 2016 году стал проектом инкубатора Apache, а в 2019 проектом верхнего уровня фонда Apache. В Airflow весь процесс представлен в виде ориентированного ациклического графа (англ.

DAG — Directed Acyclic Graph), где узлы отображают задачи, а дуги связывают данные задачи. Airflow состоит из двух основных компонентов: веб-сервера и планировщика. Веб-сервер отображает основную информацию о задачах, их статус, логи выполнения и другие полезные данные. Планировщик отвечает за запуск задач и очередность их выполнения. В качестве узлов могут выступать следующие структуры:

— Операторы (operators) — сущность, на основе которой создаются экземпляры задач, в которой описываются необходимые действия
— Сенсоры (sensors) — операторы, наступления события (наступление нужного времени, появление данных в базе и т. д.)
— Хуки (hooks) — интерфейсы для интеграции со сторонними сервисами (базы данных, файловые хранилища и т. д.)

Еще одна важная сущность Airflow это исполнитель (англ. Executor). Он представляет собой механизм выполнения задач, то есть каким образом будут выполняться задачи. Airflow имеет следующие виды исполнителей:

— Sequential Executor — все задачи выполняются на той же машине, где и запущен Airflow, отсутствует поддержка параллелизма;
— Local Executor — схож с SequentialExecutor, однако присутствует поддержка параллелизма;
— Celery Executor — исполнитель построенный на Celery. Celery — это Python-библиотека, позволяющая организовать очередь, асинхронное и распределенное исполнение задач. Для использования необходима дополнительные конфигурации;
— Kubernetes Executor — исполнитель, позволяющий запускать задачи в Kubernetes кластере. Требует запущенный Kubernetes кластер и дополнительную конфигурацию;

2.1.3 Хранилища общего назначения

Hadoop Distributed File System (HDFS)?? — распределенная файловая система, входящая в экосистему Hadoop и выпущенная в 2011 году. Предназначена для хранения больших файлов. Файлы разделены на блоки и распределены между всеми узлами кластера. Размеры блоков одинаковы, кроме последнего блока. Каждый блок может быть размещён на нескольких узлах, размер блока и коэффициент репликации (количество узлов, на которых должен быть размещён каждый блок) определяются в настройках

на уровне файла. Благодаря репликации обеспечивается устойчивость распределённой системы к отказам отдельных узлов. Модификация файлов в HDFS не поддерживается. Организация файлов — иерархическая. Развертывание файловой системы требует наличия главного узла (name node) в котором хранится все метаинформация файловой система, информация о распределении блоков и узлы данных (data node) на которых хранятся сами данные. Главный узел отвечает за обработку операций уровня файлов и каталогов — открытие и закрытие файлов, манипуляция с каталогами, узлы данных непосредственно отрабатывают операции по записи и чтению данных.

Amazon Simple Storage Service (Amazon S3) — объектное хранилище компании Amazon, представленное в 2006 году. Позволяет хранить бесконечные объемы данных. Хранение производится в корзинах (англ. bucket — ведро, корзина), логически разделенных участках хранилища. Настройки каждой корзины определяют видимость объектов в нем, шифрование, версионирование и жизненный цикл. S3 предлагает несколько классов хранения данных:

— Стандартный (Standard) — класс, задаваемый по умолчанию. Предназначен для случая, когда частота доступа к файлам высокая, в связи с чем требуется высокая производительность операций чтения и записи;

— Стандартный с нечастым доступом (Standard_IA, Standard Infrequent Access) — предназначен для долговременного хранения файлов, доступ к которым будет производиться нечасто (например, бэкапы), но тем не менее высокая производительность операций чтения-записи все еще важна. Оба класса — STANDARD и STANDARD_IA — обеспечивают доступ к файлу в режиме, близком к реальному времени, то есть с минимальной задержкой между запросом и получением данных;

— Замороженный (Glacier) — предназначен для файлов большого размера, доступ к которым будет запрашиваться очень редко (архивы, бэкапы и др.). Время доступа к данным варьируется от минут до нескольких часов;

— Замороженный, глубокая архивация (Glacier_DA, Glacier Deep Archive) — класс схожий с Glacier, однако доступ к данным осуществляется за 12-48 часов.

2.1.4 Среды преобразования данных

Amazon Elastic MapReduce (EMR)?? — сервис компании Amazon, позволяющий обрабатывать большие объемы данных с помощью фреймворков и инструентов экосистемы Hadoop. Имеется интеграция с другими сервисами AWS, такими как Amazon S3 и Amazon Redshift. Основная структурная единица в EMR это кластер. Кластер это набор виртуальных серверов, каждый сервер называется узлом. При создании кластера, на каждый сервер устанавливается различные программные компоненты зависящие от типа узла, всего типов узлов 3:

- Главный узел (Master Node) — выполняет управление кластером, распределением данных и вычислительных задач между другими узлам, отслеживание статуса задач и состояние других узлов в кластере. Имеется возможность запуска кластера, состоящего только из главного узла;
- Основной узел (Core node) — выполняет задачи, назначенные главным узлом, а также является узлом для хранения данных в HDFS;
- Узел исполнитель (Task Node) — выполняет задачи, но не хранит данные HDFS;

Запуск задач в кластере осуществляется с помощью пользовательского интерфейса, интерфейса командной строки или с помощью специального API

Архитектура AWS EMR состоит из нескольких уровней, каждый из которых покрывает определенный аспект работы кластера. **??**

Самым базовым является уровень хранения данных, включающий различные файловые системы, используемые в кластере. В качестве таковых могут выступать HDFS-распределенная файловая система Hadoop, файловая система EMR (EMRFS) — расширение HDFS, которое задействует хранилище Amazon S3, или локальная файловая система каждого конкретного узла.

Следующий уровень — это система управления ресурсами кластера (cluster resource management). По умолчанию EMR использует YARN (Yet Another Resource Negotiator), централизованно управляющий ресурсами кластера и совместимый с большинством фреймворков обработки данных. Кроме того, каждый узел EMR содержит специальный агент, который отвечает за мониторинг состояния кластера и коммуникацию виртуального сервера с AWS EMR.

Уровень фреймворка обработки данных (data processing frameworks) включает непосредственно тот фреймворк (движок), который используется для выполнения заданий в кластере. Главные фреймворки EMR — Hadoop MapReduce и Apache Spark.

И последний уровень — это уровень дополнительных приложений и программ (applications and programs) — надстройка поверх фреймворка обработки данных. На этом уровне доступны такие компоненты, как Hive, Spark Streaming, MLib, GraphX, Spark SQL (поверх Apache Spark).

Cloudera Distribution including Apache Hadoop (CDH)?? — продукт компании Cloudera, Hadoop дистрибутив включающий в себя все основные проекты экосистемы Hadoop, такие как MapReduce, Spark, Hive, HDFS, Solr, Yarn, HBase, Flume, Kafka. Имеет бесплатную и коммерческую лицензию с предоставлением поддержки.

Clouder Manager — компонент CDH, позволяющий упростить и автоматизировать развертывание Hadoop кластера и производить его мониторинг и обслуживание.

Компоненты Cloudera Hadoop распространяются в виде бинарных пакетов, называемых парселами. По сравнению со стандартными пакетами и пакетными менеджерами парселы имеют следующие преимущества:

- каждый парсел представлен в виде одного файла, в котором объеденены все нужные компоненты;
- все компоненты внутри парсела тщательно протестированы, отлажены и согласованы между собой, поэтому вероятность возникновения проблем с несовместимостью компонентов очень мала;
- при обновлении минорной версии все новые процессы (задачи) будут автоматически запускаться под этой версией, уже запущенные задачи продолжат исполняться в старом окружении до своего завершения. Однако обновление до более новой мажорной версии возможно только посредством полного перезапуска всех сервисов кластера, и соответственно всех текущих задач;
- при возникновении каких-либо проблем в работе с новой версией CDH ее можно легко откатить до предыдущей.

3 Подробное описание архитектуры

В данном разделе будет описана архитектура построенного ETL процесса в облаке Amazon Web Services, а также будут описаны компоненты и характеристики этой архитектуры.

3.1 Amazon Web Services

Amazon Web Services?? — сервис, предоставляющий услуги облачных вычислений, созданный компанией Amazon в 2006 году. Сервис предоставляет услуги как по инфраструктурной модели (виртуальные серверы, ресурсы хранения), так и платформенного уровня (облачные базы данных, облачное связующее программное обеспечение, облачные бессерверные вычисления, средства разработки), оплата производится только за используемые ресурсы. Является одним из самых крупных поставщиков услуг облачных вычислений.

Структура датацентров AWS состоит из регионов, текущее количество которых насчитывает 24, в каждом из регионов имеются зоны доступности в количестве от 2 до 3. Модель регионов AWS позволяет запускать приложения, которые требуют высокой доступности

Amazon Elastic Compute Cloud (EC2)?? — это веб-сервис, предоставляющий масштабируемые вычислительные ресурсы в облаке. Он упрощает процесс крупномасштабных вычислений в облаке для разработчиков. Простой веб-интерфейс сервиса Amazon EC2 позволяет получить доступ к вычислительным ресурсам и настроить их с минимальными трудозатратами. Пользователи имеют полный контроль над ресурсами, которые они используют. Виртуальный сервер, запущенный в EC2 называется инстанс. Amazon предлагает большой спектр инстансов различной вычислительной мощности, на различных процессорах, а также инстансы для вычислений на GPU.

Amazon Virtual Private Cloud (VPC)?? — то логически изолированный раздел облака AWS, в котором можно запускать ресурсы AWS в самостоятельно заданной виртуальной сети. Таким образом можно полностью контролировать среду виртуальной сети, в том числе выбирать собственный диапазон IP-адресов, создавать подсети, а также настраивать таблицы маршрутизации и сетевые шлюзы. Для обеспечения удобного и безопасного доступа к ресурсам и приложениям в VPC можно использовать как IPv4, так и

IPv6. Сервис использует многоуровневую систему безопасности, которая состоит из групп безопасности (Security Groups) и сетевых списков контроля доступа (NACL). Такая система позволяет контролировать доступ к ресурсам в каждой подсети.

3.2 Инфраструктура как код

Инфраструктура как код (англ. Infrastructure as Code) — практика управления и описания инфраструктуры датацентров через конфигурационные файлы и программный код, а не через прямое взаимодействие с серверами. Преимущества инфраструктуры как кода заключаются в увеличении скорости развертывания и уменьшении рисков.

Terraform?? — инструмент компании HashiCorp, позволяющий декларативно управлять инфраструктурой с помощью высокоуровневого языка конфигураций HashCorp Configuration Language (HCL), основанного на языке программирования Go. Поддерживает создание модулей. Основная структурная единица в HCL это ресурс. Ресурс — это любая возможная инфраструктурная единица облачного провайдера. Terraform обеспечивает сопоставление ресурсов, описанных в конфигурационном файле, с соответствующими ресурсами облачного провайдера. Такое сопоставление именуется состоянием. Каждый ресурс имеет присущий ему список атрибутов для конфигурации. Все ресурсы хранятся в файлах с расширением `.tf`. Процесс развертывания инфраструктуры проходит в несколько этапов:

1. Планирование?? — написанный скрипт (или группа скриптов) запускается с помощью команды `terraform plan`. Будет проведена проверка синтаксических ошибок и в командную строку или текстовый файл будет выведен план выполнения ресурсов.
2. Применение?? — с помощью команды `terraform apply` Terraform обновляет состояние, направляя соответствующий API запрос облачному провайдеру. Затем сравнивает возвращенные ресурсы с той информацией, что записана в вашей конфигурации Terraform. Если обнаружится какая-либо разница, то создается план, — перечень изменений, которые нужно внести в ресурсы облачного провайдера, чтобы фактическая конфигурация соответствовала той, что указана в файле. Затем, Terraform применяет эти изменения, направляя соответствующие API запросы облачному провайдеру.

Ansible — система управления конфигурациями серверов, написанная на языке Python. Использует YAML формат для описания конфигураций. Используется для автоматизации равзертывания программного обеспечения на сервера. Обычно используется для UNIX-систем, однако присутствует поддержка Windows. Для обеспечения конфигурации сервер на нем должен быть установлен интерпретатор Python версии 2.4 или выше, а также возможность подключения по SSH.

3.3 Описание архитектуры в общем виде

В данной главе будет описана архитектура построенного ETL процесса в общем виде. Откуда поступают данные, где хранятся сырые данные, как они обрабатываются и куда записывается результат. Схема архитектуры представлена на рисунке 2

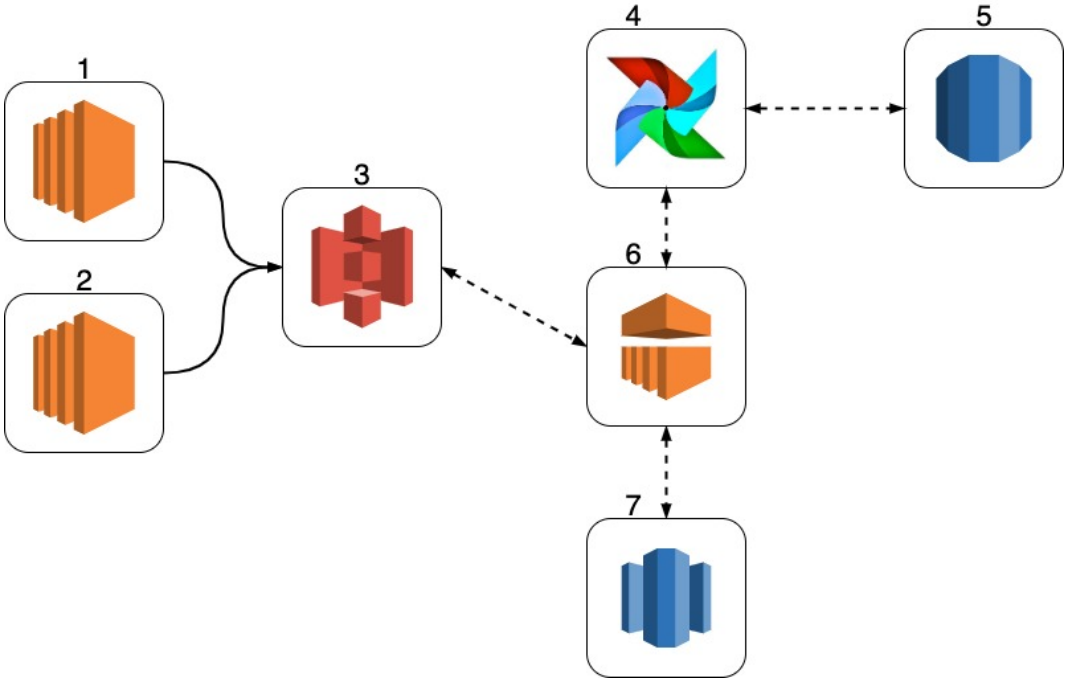

Рисунок 2 – Архитектура ETL процесса

Ключевым элементом архитектуры является Apache Airflow, который запускает задачи обработки данных. Вся архитектура расположена в виртуальной частной сети, предоставленной AWS. Необработанные данные хранятся на отдельных инстансах (1, 2), которые моделируют источники данных. При запуске графа на Airflow инстансе (4) происходит проверка наличия данных на инстансах. Если данные присутствуют, начинается загрузка в S3 корзину (3) для их дальнейшей обработки. Затем запускается задача транс-

формации данных. Она выполняется в кластере EMR (7). Код программы для обработки написан с помощью фреймворка Spark на языке Python. Данная программа производит расчет метрик и конвертацию исходных файлов в формат Apache Parquet. Метрики сохраняются в хранилище данных Amazon Redshift (6), а файлы Parquet сохраняются в отдельную папку в S3 корзине (3). Задачи Airflow запускаются на том же инстансе, где и сам Airflow, файл с описанием задач и логи выполнения хранятся в локальной файловой системе.

3.4 Серверы с необработанными данными

Как было сказано ранее, серверы с необработанными данными являются EC2 инстансами типа t2.micro с одним виртуальным ядром и 1Гб оперативной памяти. Инстансы используют дистрибутив Amazon Linux 2, основанный на дистрибутиве Red Hat Enterprise Linux компании Red Hat. Кроме стандартного пакета программ, на инстансы установлен Python интерпретатор версии 3.6, а также интерфейс коммандной строки AWS CLI, позволяющий управлять сервисами AWS прямо из командной строки. Это небходимо для копирования данных с сервера в S3 корзину. Необработанные данные хранятся в папке `data/csv` и `data/json` в домашней директории стандартного пользователя на каждом сервере соответственно.

3.5 Среда запуска Spark приложений

3.5.1 Кластер

Кластер для выполнения трансформации данных работает в сервисе AWS EMR. На кластере установлен фреймворк Apache Spark версии 2.4.4, а также в конфигурации добавлена поддержка Python третьей версии. Кластер состоит из двух узлов: одного главного узла и одного основного. Узлы являются инстансами типа m4.large с 4 виртуальными ядрами и 8Гб оперативной памяти, в качестве файловой системы кластера используется S3 корзина. Также при старте кластера на каждый узел будет установлен Python модуль boto3 с помощью, которого можно осуществить доступ к другим сервисам AWS из кода.

3.5.2 Объектное хранилище

В качестве файловой системы кластера и объектного хранилища для необработанный выступает S3 корзина. Все объекты в ней имеют класс хра-

нения Standard. В корзине расположены следующие папки:

— `application` — содержит код Spark приложений;
— `data` — соедржит необработанные данные, в ней есть 2 подпапки, csv и json, отвечающие за данные с разных серверов;
— `dependencies` — папка с зависимостями для Spark приложения. В ней расположены `.jar` файлы необходимых библиотек:
— `emr-logs` — папка для логов. Все логи, производимые кластером записываются в эту папку;
— `parquet` — папка, в которую сохраняются конвертированные файлы в формате Parquet;
— `tmp`— временная папка для кэширования результатов записи в хранилище данных;

3.5.3 Хранилище данных

В качестве хранилища данных выступает кластер AWS Redshift. Кластер состоит из одного узла типа dc2.large с двумя виртуальными ядрами и 15Гб оперативной памяти, размер дискового пространства в кластере — 160Гб. В кластере была создана база данных `metrics`, в которую записываются результаты работы Spark приложения.

3.6 Apache Airflow

3.6.1 Мастер сервер

В качестве главного сервера на котором работает Airflow, выступает инстанс типа t3.medium с двумя виртуальными ядрами и 4Гб оперативной памяти. На сервер с помощью Ansible устанавливаются интерпретатор Python версии 3.6, интерфейс коммандной строки AWS CLI. Затем производится установка Airflow и создание рабочей директории для него, туда копируется файл конфигурации. Также для возможности запуска задач в Kubernetes кластере устанавливается утилиты `aws-iam-authenticator` и `kubectl` и производится их настройка. С их помощью и при правильной конфигурации Airflow сможет запускать задачи в Kubernetes.

3.6.2 Хранилище метаданных

В качестве хранилища метаданных для Airflow использует инстанс базы данных в сервисе AWS RDS типа db.t2.micro с 1 виртуальным ядром, 1Гб

оперативной памяти и 10Гб дискового пространства.

3.6.3 Kubernetes кластер

Docker — программное обеспечение для автоматизации развертывания приложений в средах с поддержкой виртуализации. Позволяет переносить приложение со всеми зависимостями на любую Linux систему с поддержкой cgroups в ядре.

Kubernetes?? — платформа для автоматизации развертывания, управления и масштабирования контейнеров в кластере из серверов. Основные концепции Kubernetes:

- Кластер — набор серверов, хранилищ данных и сетевых ресурсов, с помощью которых Kubernetes запускает контейнеры в системе:
- Узел — это отдельный сервер, который запускает контейнеры. Каждый узел содержит в себе Kubernetes агент kubelet и прокси сервис;
- Ведущий узел — панель управления Kubernetes. Состоит из API-сервера, планировщика и диспетчера контроллеров;
- Под — единица работы в Kubernetes. Каждый под содержит один или несколько контейнеров. Поды всегда работают совместно, то есть на одном сервере Все контейнеры внутри пода имеют одни и те же IP-адрес и пространство портов, они могут общаться между собой через локальный сервер или посредством межпроцессного взаимодействия. Кроме того, все контейнеры имеют доступ к обще- му локальному хранилищу данных узла, на котором находится под;
- Том — харнилище, которое может быть подключено к любому количеству контейнеров и не удаляется после удаления контейнеров. Существует различные типы томов для различных облачных платформ, сетевых файловых систем и Git репозиториев;
- Пространства имен — это виртуальный кластер. Один физический кластер может содержать несколько виртуальных, разделенных пространствами имен. Все виртуальные кластеры изолированы друг от друга, а общение происходит через публичные интерфейсы.
- Запрос тома — запрос на доступ к тому для использования в поде;

4 Программная реализация

4.1 Скрипты создания инфраструктуры

Для достижения большей гибкости при создании инфраструктуры и возможности переиспользования кода, Terraform позволяет параметризовать атрибуты ресурсов. С помощью конструкции `variable??` Terraform позволяет задать переменную определенного типа и задать ей стандартное значение. Пример конструкции `variable`

```
1 variable "example" {
2   type = string
3   description = "..."
4   default = "Hello Terraform"
5 }
```

В строке 2 указывается тип переменной, в данном случае это переменная строкового типа. В строке 3 указывается краткое описание этой переменной, а в строке 4 указывается стандартное значение.

Также для объединения и создания группы ресурсов Terraform поддерживает модульность. Модули, созданные другими разработчиками загружаются с помощью команды `terraform get??`, также у Terraform есть конструкции `data??` и `output??`, которые позволяет хранить данные из других конфигураций или из внешних файлов и выводить необходимые данные в консоль после применения конфигурации. Создание собственных модулей обычно предусматривает наличие трех файлов:

- `main.tf` — главный файл, в нем содержится вся конфигурация инфраструктуры;
- `variables.tf` — файл, содержащий описание переменных;
- `outputs.tf` — файл, содержащий конфигурацию выводов в консоль;

4.1.1 Airflow сервер

Для создания сервера был использован ресурс `aws_instance`, который создает EC2 инстанс с необходимыми параметрами. Помимо создания инстанса необходимо создать правила для группы безопасности, чтобы разрешить входящий трафик на определенные сетевые порты.

Листинг файла с переменными для Airflow сервера:

```
1 variable "region" {
2   type        = string
```

```
3   description = "Default region for Airflow instance"
4   default     = "us-east-2"
5 }
6
7 variable "ami_id" {
8   type        = string
9   description = "ID of Amazon Machine Image"
10  default     = "ami-0f7919c33c90f5b58"
11 }
12
13 variable "instance_type" {
14  type        = string
15  description = "Instance type"
16  default     = "t3.medium"
17 }
18
19 variable "availability_zone" {
20  type        = string
21  description = "Availability zone for instance"
22  default     = "us-east-2c"
23 }
24
25 variable "security_groups" {
26  type        = list(string)
27  description = "List of security groups for instance"
28  default = [
29    "sg-06ddfa8684cb6197b",
30    "sg-0db432ff9749de98b",
31    "sg-5d707a33",
32  "sg-0db432ff9749de98b"]
33 }
```

В переменной `region` указан регион в котором будет создана инфраструктура, регион по умолчанию us-east-2. В переменной `ami_id` указан номер образа операционной системы, которая будет запущена на инстансе. В перемнной `instance_type` содержится информация о типе инстанса, который будет запущен, а в переменной `availability_zone` указана зона доступности в которой будет запущен инстанс. Переменная `security_groups` содержит список групп безопасности, которые будут прикреплены к инстансу.

Листинг файла с описанием конфигурации Airflow сервера:

```
provider "aws" {
  region = var.region
}

resource "aws_security_group_rule" "db" {
  cidr_blocks       = ["0.0.0.0/0"]
  from_port         = 5432
  protocol          = "tcp"
  security_group_id = var.security_groups[2]
  to_port           = 5432
  type              = "ingress"
}

resource "aws_security_group_rule" "webserver" {
  cidr_blocks       = ["0.0.0.0/0"]
  from_port         = 8080
  protocol          = "tcp"
  security_group_id = var.security_groups[2]
  to_port           = 8080
  type              = "ingress"
}

resource "aws_security_group_rule" "nfs" {
  cidr_blocks       = ["0.0.0.0/0"]
  from_port         = 2049
  protocol          = "tcp"
  security_group_id = var.security_groups[2]
  to_port           = 2049
  type              = "ingress"
}

resource "aws_instance" "airflow_master" {
  ami                         = var.ami_id
  instance_type               = var.instance_type
  availability_zone           = var.availability_zone
  key_name                    = "my-key.pem"
  vpc_security_group_ids      = var.security_groups
  associate_public_ip_address = true
  root_block_device {
    volume_type = "gp2"
    volume_size = 30
```

```
42    }
43
44    provisioner "local-exec" {
45      command = <<EOT
46      bash wait.sh ${aws_instance.airflow_master.public_ip}
47      EOT
48    }
49
50    provisioner "local-exec" {
51      command = <<EOT
52      ansible-playbook \
53      -i ${aws_instance.airflow_master.public_ip}, \
54      install_packages.yaml \
55      --key-file ~/Documents/my-key.pem
56      EOT
57    }
58  }
```

В строке 1 указан облачный провайдер, в котором будет создана инфраструктура, в аттрибуте `region` указан регион.

В строках 5, 14 и 23 создаются ресурсы для правил группы безопасности. Они прикрепляются в необходимой группе безопасности, которая указана в аттрибуте `security_group_id`. Аттрибуты `from_port` и `to_port` обозначают диапазон портов для которых разрешен входящий/исходящий трафик. Если порт один, эти аттрибуты принимают одинаковое значение Аттрибут `protocol` определяет протокол для которого действует данное правило применяется, аттрибут `security_group_id` определяет группу безопасности, к которой будет прикреплено данное правило. Порты 5432, 8080 и 2049 необходимы для того, чтобы сервер мог получать траффик от хранилища метаданных, пользователей Airflow веб-сервера и сетевой файловой системы соответственно.

Ресурс `aws_instance` создает Airflow сервер. Описание аттрибутов:

- `ami` — ID образа операционной системы;
- `instance_type` — тип инстанса;
- `availability_zone` — зона доступности;
- `key_name` — название зарегистрированного SSH ключа для доступа к серверу посредством одноименного протокола;
- `vpc_security_group_ids` — группы безопасности, прикрепленные к ин-

стансу;

- `associate_public_ip_address` — флаг, который указывает назначать ли инстансу публичный IP адрес;
- `root_block_device` — составной аттрибут отвечающий за дисковое пространство инстанса. В аттрибуте `volume_type` указывается тип диска, а в `volume_size` его размер;

Также хотелось бы выделить составной аттрибут `provisioner` (рус. поставщик), который позволяет запускать различные команды в зависимости от его типа. В данном случае после создания ресурса будет запущенна команда в локальном терминале. В строке 44 будет поставщик запустит скрипт, который будет в цикле подключаться к инстансу по SSH, пока тот не станет доступен. Как только это произойдет, поставщик на строке 50 запустит Ansible конфигурацию Airflow сервера.

4.1.2 Кластер EMR и S3 корзина

Для создания кластера для обработки данных и S3 корзины были использованы модули `cloudposse/terraform-aws-emr-cluster??` и `cloudposse/terr`
Рассмотрим файл с переменными:

```
1  variable "vpc_id" {
2    type        = string
3    description = "VPC id for security group"
4    default     = "vpc-a71a06cf"
5  }
6
7  variable "subnet_id" {
8    type        = string
9    description = "Subnet id where to run cluster"
10   default     = "subnet-08363260"
11 }
12
13 variable "core_instance_type" {
14   type        = string
15   description = "Instance type for Core instance group"
16   default     = "m4.large"
17 }
18
19 variable "core_instance_count" {
20   type        = number
```

```
21   description = "Number of Core instances launched in cluster"
22   default     = 1
23 }
24
25 variable "core_instance_ebs_size" {
26   type        = number
27   description = "Size of Elastic Block Storage for each core
       instance"
28   default     = 32
29 }
30
31 variable "core_instance_ebs_type" {
32   type        = string
33   description = "EBS volume type for core instances"
34   default     = "gp2"
35 }
36
37 variable "master_instance_type" {
38   type        = string
39   description = "Instance type for Master instance group"
40   default     = "m4.large"
41 }
42
43 variable "master_instance_count" {
44   type        = number
45   description = "Number of Master instances launched in cluster"
46   default     = 1
47 }
48
49 variable "master_instance_ebs_size" {
50   type        = number
51   description = "Size of Elastic Block Storage for each master
       instance"
52   default     = 32
53 }
54
55 variable "master_instance_ebs_type" {
56   type        = string
57   description = "EBS volume type for master instance"
58   default     = "gp2"
59 }
```

```
60
61 variable "key_name" {
62   type        = string
63   description = "Name of registered SSH key"
64   default     = "my-key.pem"
65 }
```

Переменная `vpc_id` хранит ID виртуальной частной сети, где будет запущен кластер, а `subnet_id` — ID подсети. Переменные с префиксами `core` и `master` имеют одинаковое назначение для основного и главного типов узлов соответственно. Рассмотрим переменные для основного узла:

- `core_instance_type` — тип инстанса, который будет запущен;
- `core_instance_count` — их количество, по умолчанию 1;
- `core_instance_ebs_size` — размер диска для кадого узла;
- `core_instance_ebs_type` — тип диска;

Листинг файла с описанием инфраструктуры:

```
1  provider "aws" {
2    region = "us-east-2"
3  }
4
5  module "data_and_other" {
6    source  = "cloudposse/s3-log-storage/aws"
7    version = "0.9.0"
8    name    = "data-and-other"
9    stage   = "prod"
10 }
11
12 data "local_file" "configuration_json" {
13   filename = file("./emr_bootstrap.json")
14 }
15
16 module "prod_etl_cluster" {
17   source                                =
       "cloudposse/emr-cluster/aws"
18   version                               = "0.5.0"
19   name                                  = "ETL prod"
20   region                                = "us-east-2"
21   vpc_id                                = var.vpc_id
22   subnet_id                             = var.subnet_id
23   subnet_type                           = "private"
```

```
24    applications                                = ["Spark"]
25    configurations_json                         =
        data.local_file.configuration_json.content
26    core_instance_group_instance_type    = var.core_instance_type
27    core_instance_group_ebs_size         = var.core_instance_ebs_size
28    core_instance_group_ebs_type         = var.core_instance_ebs_type
29    core_instance_group_instance_count   = var.core_instance_count
30    master_instance_group_instance_type  = var.master_instance_type
31    master_instance_group_ebs_size       =
        var.master_instance_ebs_size
32    master_instance_group_ebs_type       =
        var.master_instance_ebs_type
33    master_instance_group_instance_count = var.master_instance_count
34    log_uri                                     = format("s3n://%s",
        module.data_and_other.bucket_id)
35    key_name                                    = var.key_name
36  }
```

4.2 Пример Spark приложения

4.2.1 Подсчет метрик

4.2.2 Конвертация в Parquet

4.3 Скрипты транспортировки данных

4.4 Airflow DAG

4.5 Конфигурации

4.5.1 Общие тома для логов и DAGов

4.5.2 Kubeconfig

4.5.3 Конфигурацяи Airflow для Kubernetes кластера

4.5.4 Конфигурация Airflow

ЗАКЛЮЧЕНИЕ

Заключение

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ