Краткое оглавление

Оглавление

Предисловие

Книга «Apache Pulsar в действии» – это недостающее руководство, которое проведет вас через все этапы работы с Apache Pulsar. Эту книгу я порекомендовал бы прочесть всем: от разработчиков, начинающих изучать механизм обмена сообщениями по схеме «публикация–подписка» (pub-sub), до тех, у кого есть опыт обмена сообщениями, и до опытных пользователей Pulsar.

Работа над проектом Apache Pulsar началась в компании Yahoo! приблизительно с 2012 г. во время экспериментов с новой архитектурой, которая должна была решить эксплуатационные задачи на существующих платформах обмена сообщениями. К тому же это было время, когда некоторые значительные сдвиги в мире инфраструктуры данных становились все более заметными. Разработчики приложений начали обращать больше внимания на масштабируемый и надежный механизм обмена сообщениями как на основной компонент для создания следующего поколения программных продуктов. В то же время компании определили крупномасштабные системы анализа потоковых данных в реальном времени как важнейший компонент и преимущество для бизнеса.

Pulsar был спроектирован «с нуля» с целью установления прочной связи между этими двумя сферами – механизма обмена по схеме «публикация–подписка» (pub-sub) и анализа потоковых данных, которые весьма часто были изолированы друг от друга. Мы работали над созданием инфраструктуры, представляющей следующее поколение платформ обработки данных в реальном времени, где единственная система могла бы поддерживать все варианты использования на протяжении полного жизненного цикла событий, связанных с данными.

Со временем это представление расширялось, как можно ясно видеть по широкому диапазону компонентов, описанных в этой книге. В проект была добавлена поддержка упрощенной обработки с помощью Pulsar Functions, фреймворка коннекторов Pulsar IO, поддержка схем данных и многие другие функциональные возможности. Не изменилась только конечная цель – создание в высшей степени масштабируемой, гибкой и надежной платформы для обработки данных в реальном времени, позволяющей любому пользователю работать с данными, хранящимися в Pulsar, в наиболее удобной форме.

Я знаком с автором этой книги Дэвидом Хьеррумгором и работал вместе с ним в течение нескольких лет. За это время я обратил вни-

мание на его глубокую увлеченность работой в сообществе Pulsar. Он всегда готов помочь пользователям решить технические проблемы и затруднения, а также продемонстрировать им все возможности Pulsar для решения конкретной задачи обработки данных.

Особенно важным я считаю тот факт, что в книге органично сочетается теория и абстрактные концепции с ясностью практических пошаговых примеров и что эти примеры основаны на широко известных вариантах использования и паттернах проектирования обмена сообщениями, которые, несомненно, заинтересуют многих читателей. Здесь действительно каждый найдет что-то полезное для себя, и каждый сможет ознакомиться со всеми аспектами и возможностями, которые предлагает Pulsar.

— *Маттео Мерли (Matteo Merli),*
технический директор (CTO) Stream Native,
один из создателей и председатель комитета управления
проектом (PMC Chair) Apache Pulsar

Предисловие автора

В 2012 г. команда Yahoo! искала глобальную платформу с георепликацией, которая могла бы обеспечить потоковую обработку всех данных Yahoo! при обмене сообщениями между различными приложениями, например Yahoo Mail и Yahoo Finance. В то время существовали два основных типа систем обработки динамических данных: очереди сообщений, обрабатывающие особо важные бизнес-события в реальном времени, и потоковые системы, которые обрабатывали динамически масштабируемые конвейеры данных. Но при этом не было платформы, предоставляющей одновременно обе функциональные возможности, требуемые для компании Yahoo.

После тщательного исследования положения дел в области обмена сообщениями и потоковой обработки данных стало ясно, что существующие технологии не способны удовлетворить эти потребности, поэтому команда Yahoo! начала работу по созданию объединенной платформы обмена сообщениями и потоковой обработки динамических данных под названием Pulsar. После четырех лет работы в 10 центрах данных, обрабатывающих миллиарды сообщений в день, компания Yahoo! решила открыть исходный код своей платформы обмена сообщениями под защитой лицензии Apache в 2016 г.

Впервые я встретился с Pulsar осенью 2017 г. Тогда я возглавлял группу поддержки профессиональных сервисов в Hortonworks, сосредоточенной на работе с платформой потоковой обработки данных под названием Hortonworks Data Flow (HDF), в которую входили компоненты Apache NiFi, Kafka и Storm. В мои обязанности входил контроль за развертыванием этих технологий в инфраструктуре пользователя и помощь на начальной стадии разработки приложений потоковой обработки данных.

Самым большим затруднением, с которым мы столкнулись при работе с Kafka, была помощь нашим клиентам в правильном администрировании этого средства и специфическом определении необходимого количества разделов для конкретной темы, чтобы достичь надлежащего баланса скорости и эффективности при допущении дальнейшего роста объема данных. Тем из вас, кто знаком с Kafka, печально известен тот факт, что это кажущееся простым решение оказывает серьезное воздействие на масштабируемость тем, а для выполнения процедуры изменения этого значения (даже с 3 на 4) необходим весьма медленный процесс ребалансировки, на

протяжении которого все балансируемые темы становятся недоступными для чтения или записи.

Это требование ребалансировки неизменно вызывало негативную реакцию всех клиентов, использовавших HDF, и вполне справедливо, потому что они воспринимали его как препятствие для масштабирования кластера Kafka при непрерывном росте объемов данных. Только из собственного опыта клиенты узнавали, как трудно изменять в обе стороны масштаб платформы обмена сообщениями. Еще более худшим являлся тот факт, что невозможно было просто «прикрутить» еще несколько узлов для наращивания вычислительной мощности существующего клиентского кластера без соответствующего изменения конфигурации тем, чтобы использовать их для выделения большего количества разделов для существующих тем, чтобы перераспределить данные по только что добавленным узлам. Такая невозможность горизонтального масштабирования мощности потоковой обработки без ручного вмешательства (или необходимости написания огромного объема скриптов) приводила к прямому конфликту с желанием большинства наших клиентов переместить свои платформы обмена сообщениями в облако и воспользоваться всеми преимуществами гибких вычислительных возможностей, предоставляемых облачной средой.

Именно тогда я открыл для себя платформу Apache Pulsar и обнаружил, что ее заявление о том, что она «естественна для облака», особенно привлекательно, потому что она устраняет обе болевые точки масштабируемости. Хотя HDF позволял клиентам быстро приступить к работе, возникали трудности в управлении, да и сама платформа не была предназначена для функционирования в облаке. Я понял, что Apache Pulsar был намного лучшим решением, чем предлагаемое в тот момент нашим клиентам, и попытался убедить свою группу разработчиков рассмотреть возможность замены Kafka на Pulsar в нашем продукте HDF. Я даже зашел так далеко, что написал коннекторы, которые позволили Pulsar работать с компонентом Apache NiFi нашего стека, чтобы облегчить внедрение, но безрезультатно.

Когда в январе 2018 г. ко мне обратились первоначальные разработчики Apache Pulsar и предложили присоединиться к небольшому стартапу под названием Streamlio, я сразу же воспользовался возможностью поработать с ними. В то время Pulsar был пока еще молодым проектом, его только что включили в программу инкубации Apache, и следующие 15 месяцев мы провели, работая над тем, чтобы наш неоперившийся «птенчик» прошел через процесс инкубации и получил статус проекта высшего уровня.

Это было в самый разгар ажиотажа вокруг потоковой передачи данных, и Кафка был доминирующим игроком на этом поле, поэтому естественно, что все считали эти термины взаимозаменяемыми.

По общему мнению, Kafka был единственной доступной платформой для потоковой передачи данных. На основе своего предыдущего опыта я взял на себя смелость неустанно проповедовать то, что знал как технологически превосходное решение, – одинокий голос вопиющего в пресловутой пустыне.

К весне 2019 г. количество участников и пользователей в сообществе Apache Pulsar существенно увеличилось, но надежной документации по этой технологии катастрофически не хватало. Поэтому, когда мне впервые предложили написать «Apache Pulsar в действии», я сразу же воспринял это как возможность удовлетворить острую потребность для сообщества Pulsar. Хотя мне так и не удалось убедить своих коллег присоединиться ко мне в этом начинании, они были бесценным источником рекомендаций и информации на протяжении всего процесса и использовали эту книгу как средство передачи вам части своих знаний.

Эта книга предназначена для абсолютных новичков в Pulsar и является сочетанием информации, собранной мною во время непосредственного сотрудничества с основателями Pulsar в процессе активной разработки этой платформы, и опыта, накопленного во время работы с организациями, включившими Apache Pulsar в производственный процесс.

Книга предназначена для того, чтобы помочь вам преодолеть препятствия и ловушки, с которыми другие столкнулись во время своих исследований Pulsar. Прежде всего эта книга придаст вам уверенности при разработке приложений потоковой обработки и микросервисов с использованием Pulsar и языка программирования Java. Несмотря на то что я решил использовать Java для большинства примеров кода в книге из-за личного знакомства с этим языком, я также создал аналогичный комплект исходного кода с использованием Python и загрузил его в свою учетную запись GitHub для тех, кто предпочитает писать код на этом языке.

Об этой книге

Книга «Apache Pulsar в действии» была написана как введение в технологию потоковой обработки данных, чтобы помочь читателям познакомиться с терминологией, семантикой и особенностями, которые необходимо знать для восприятия парадигмы потоковой обработки при переходе от технологии пакетной обработки данных. Она начинается с исторического обзора развития систем передачи сообщений за последние 40 лет и показывает, как Pulsar оказался на вершине этого эволюционного цикла.

После краткого описания основной терминологии в сфере передачи сообщений и обсуждения двух основных паттернов потребления сообщений рассматривается архитектура Pulsar с физической точки зрения, где основное внимание уделяется его проектному решению, наиболее подходящему для облачной среды, а также с точки зрения его логического структурирования данных и поддержки мультиарендности (множественной аренды, многоарендной архитектуры).

В остальной части книги все внимание сосредоточено на том, как можно использовать встроенную вычислительную платформу Pulsar под названием Pulsar Functions для разработки приложений с применением простого API. Этот подход демонстрируется реализацией варианта использования в обработке заказов: на воображаемом предприятии по доставке продуктов питания создается приложение с применением микросервисов исключительно на основе Pulsar Functions с дополнительным развертыванием модели машинного обучения для оценки времени доставки.

Для кого предназначена эта книга

Книга «Apache Pulsar в действии» предназначена главным образом для разработчиков на языке Java, интересующихся технологией обработки потоковых данных, или для разработчиков микросервисов, которые ищут альтернативный фреймворк для порождения (генерации) событий. Группы DevOps, заинтересованные в развертывании и функционировании Pulsar в своих организациях, также найдут эту книгу полезной. Одним из основных критических замечаний по Apache Pulsar является общее отсутствие документации и сообщений в блогах, доступных в интернете, и, хотя без сомнений ожидается, что это изменится в ближайшем будущем, я наде-

юсь, что книга поможет заполнить этот пробел в промежуточный период и принесет пользу всем, кто хочет узнать больше о потоковой обработке в целом и об Apache Pulsar в частности.

Как организована эта книга: дорожная карта

Книга состоит из 12 глав, разделенных на три части. Часть I начинается с введения в Apache Pulsar и определения его места в 40-летней истории развития систем передачи сообщений, а также сравнения Pulsar с разнообразными платформами обработки сообщений, существующими до него:

- в главе 1 представлена история систем передачи сообщений и определено место Pulsar в 40-летней истории развития технологии передачи сообщений. Также приведен краткий обзор некоторых архитектурных преимуществ Pulsar над другими системами и обоснование выбора Pulsar как единственной платформы обработки сообщений;
- в главе 2 подробно рассматривается многозвенная архитектура Pulsar, обеспечивающая независимое динамическое масштабирование хранилища данных или сервисных уровней. Здесь также описаны некоторые широко применяемые паттерны потребления сообщений, их отличия друг от друга и методы их поддержки в Pulsar;
- в главе 3 демонстрируется взаимодействие с Apache Pulsar из командной строки, а также с использованием его программного интерфейса (API). После изучения этой главы вы будете вполне уверенно запускать локальный экземпляр Apache Pulsar и взаимодействовать с ним.

В части II более глубоко рассматриваются основные варианты использования и функциональные возможности Pulsar, в том числе способы выполнения основных операций обработки сообщений и методы защиты кластера Pulsar, а также более продвинутые функциональные средства, например реестр схем. Кроме того, здесь представлен фреймворк Pulsar Functions с описанием того, как создавать, развертывать и тестировать функции:

- глава 4 представляет собственный потоковый вычислительный фреймворк Pulsar под названием Pulsar Functions с описанием некоторых основных принципов его проектирования и конфигурации, а также показывает, как разрабатывать, тестировать и развертывать функции;
- в главе 5 представлен фреймворк коннекторов Pulsar, предназначенный для перемещения (данных) между Apache Pulsar и внешними системами хранения, такими как реляционные базы данных, хранилища ключ–значение и хранилища blob-

объектов, например S3. Здесь вы узнаете, как разрабатывается коннектор – приводится пошаговое описание процесса разработки;

- глава 6 содержит подробные пошаговые инструкции по обеспечению безопасности кластера Pulsar для гарантии того, что ваши данные защищены как в режиме передачи, так и при хранении;
- в главе 7 рассматривается встроенный реестр схем Pulsar, обосновывается его необходимость, объясняется, как он может помочь в упрощении разработки микросервиса. Также описан процесс эволюции схемы и способ обновления схем, используемых внутри конкретного экземпляра Pulsar Functions.

В части III все внимание сосредоточено на использовании Pulsar Functions для реализации микросервисов и показано, как реализуются разнообразные паттерны проектирования широко применяемых микросервисов с помощью Pulsar Functions. В этой части демонстрируется процесс разработки приложения для предприятия по доставке пищи, чтобы сделать все примеры более реалистичными и применить более сложные варианты использования, включая обеспечение отказоустойчивости, доступ к данным и методику использования Pulsar Functions для развертывания моделей машинного обучения, которые могут работать с данными в реальном времени:

- в главе 8 показано, как реализовать часто применяемые паттерны маршрутизации сообщений, такие как разделение сообщений, маршрутизация на основе содержимого и фильтрация. Также демонстрируется реализация разнообразных паттернов преобразования сообщений, например извлечение значений и перевод сообщения;
- глава 9 подчеркивает важность механизма отказоустойчивости, встроенного в микросервисы, и демонстрирует его реализацию внутри экземпляра Pulsar Functions на основе Java с помощью библиотеки resiliency4j. Здесь рассматриваются различные события, которые могут произойти в программе, управляемой событиями, а также разнообразные паттерны, которые можно использовать для защиты сервисов от аварийных сценариев такого рода, чтобы максимизировать время работы приложения;
- в главе 10 рассматривается, как можно обеспечить доступ к данным из различных внешних систем при внутреннем использовании специально созданных функций Pulsar. Показаны разнообразные способы получения информации внутри микросервисов и условия, которые вы должны учитывать с точки зрения вероятных задержек;

- глава 11 представляет полный процесс развертывания разнообразных типов моделей машинного обучения внутри функции Pulsar с использованием различных фреймворков машинного обучения. Также рассматривается весьма важная тема: как передать необходимую информацию в модель, чтобы получить точный прогноз;
- в главе 12 описано использование Pulsar Functions в периферийной вычислительной среде для выполнения анализа в реальном времени данных интернета вещей (IoT). Глава начинается с подробного описания периферийной вычислительной среды и различных уровней ее архитектуры, и только после этого рассматривается, как применить Pulsar Functions для обработки информации на периферии, чтобы отправлять только краткие сводки, а не полный набор данных.

Завершают книгу два приложения, демонстрирующие более продвинутые рабочие сценарии, включающие развертывание в среде Kubernetes и георепликацию:

- в приложении A содержатся пошаговые инструкции, необходимые для развертывания Pulsar в среде Kubernetes с использованием схем Helm, предоставляемых как часть проекта с открытым исходным кодом. Также рассматривается изменение этих схем для соответствия среде, которую вы используете;
- в приложении B описан встроенный в Pulsar механизм георепликации и некоторые часто используемые паттерны репликации, применяемые в современном производстве. Далее подробно описан процесс реализации одного из таких паттернов георепликации в Pulsar.

Примеры исходного кода

Книга содержит множество примеров исходного кода как в форме пронумерованных листингов, так и в виде отдельной строки или нескольких строк в обычном тексте. В обоих случаях исходный код отформатирован с использованием `такого шрифта постоянной ширины` для отделения его от обычного текста. Иногда код также дополнительно выделяется **`полужирным шрифтом`**, чтобы специально выделить фрагмент кода, который изменился по сравнению с предыдущими этапами (примерами) в текущей главе, например при добавлении нового функционального средства (свойства) в существующую строку кода.

Во многих случаях изначальный исходный код был переформатирован: добавлены символы перехода на новую строку и изменено выравнивание для адаптации к доступному пространству на странице этой книги. В редких случаях, когда даже такие меры

оказались недостаточными, в листинги включены маркеры продолжения строки (➥). Кроме того, комментарии часто удалялись из исходного кода, если этот код подробно описывается в тексте. Многие листинги предваряются аннотациями к коду, которые особо выделяют наиболее важные концепции.

«Apache Pulsar в действии» – прежде всего книга по программированию, предназначенная для использования в качестве практического руководства для изучения того, как разрабатывать микросервисы с использованием Pulsar Functions. Поэтому я предоставил несколько репозиториев исходного кода, на которые я часто ссылаюсь на протяжении всей книги. Рекомендую загрузить код с соответствующей страницы веб-сайта издательства (*https://www.manning.com/books/apache-pulsar-in-action*) или из моей личной учетной записи на GitHub:

- этот репозиторий GitHub содержит код примеров из глав 3–6 и 8: *https://github.com/david-streamlio/pulsar-in-action*;
- код приложения микросервисов для предприятия по доставке пищи можно найти в GitHub-репозитории: *https://github.com/david-streamlio/GottaEat*;
- код приложения для анализа интернета вещей (IoT), рассматриваемого в главе 12, размещен здесь: *https://github.com/david-streamlio/Pulsar-Edge-Analytics*;
- для тех, кому необходимы примеры на языке Python, предназначен следующий репозиторий: *https://github.com/david-streamlio/pulsar-in-action-python*.

Прочие онлайн-ресурсы

Нужна дополнительная помощь?

- Веб-сайт проекта Apache Pulsar *https://pulsar.apache.org* – неплохой источник информации о параметрах настройки конфигурации различных компонентов программного обеспечения Apache Pulsar, а также различных практических руководств по реализации конкретных функциональных возможностей этого программного обеспечения. На этом сайте вы всегда найдете самую свежую информацию.
- Канал Apache Pulsar Slack: *apache-pulsar.slack.com* – действующий форум, где встречаются члены сообщества Apache Pulsar со всего мира для обмена опытом, самыми эффективными практическими методиками и предоставления рекомендаций по устранению проблем для людей, которые впервые столкнулись с неприятными ситуациями в Pulsar. Это отличное место, где можно получить полезный совет, если вы попали в ловушку.

- В моем нынешнем качестве девелопер-адвоката я продолжаю разрабатывать дополнительный образовательный контент, в том числе публикации в блоге и примеры кода, которые вскоре будут доступны в режиме онлайн на веб-сайте моей компании *streamnative.io*.

Форум обсуждения liveBook

Приобретение книги «Apache Pulsar в действии» подразумевает свободный (бесплатный) доступ к веб-форуму издательства Manning Publications, где вы можете добавлять комментарии к этой книге, задавать технические вопросы и отвечать на них, а также получать помощь от автора и других пользователей. Чтобы получить доступ к форуму, перейдите по адресу *https://livebook.manning.com/#!/book/apache-pulsar-in-action/discussion*. О форумах издательства Manning и правилах их модерации можно узнать более подробно здесь: *https://livebook.manning.com/#!/discussion*.

Обязательство издательства Manning перед читателями состоит в том, чтобы предоставить место, где может состояться содержательный диалог между отдельными читателями, а также между читателями и автором. Это не является обязательством какой-либо конкретной степени участия со стороны автора, чей вклад в форум остается добровольным (и неоплачиваемым). Мы предлагаем попробовать задать автору несколько сложных вопросов, чтобы он не потерял интерес к форуму. Форум и архивы предыдущих обсуждений будут доступны на веб-сайте издателя, пока книга находится в печати.

Об авторе

Дэвид Хьеррумгор (David Kjerrumgaard) – участник проекта Apache Pulsar и сотрудник в должности Developer Advocate компании StreamNative, главной обязанностью которого является обучение разработчиков использованию Apache Pulsar. Ранее он был Global Practice Director в компании Hortonworks, где отвечал за разработку наиболее эффективных практических методик и решений для группы профессиональных сервисов, уделяя особое внимание потоковым технологиям, включая Kafka, NiFi и Storm. Дэвид имеет степень бакалавра и магистра по информатике и математике Кентского университета.

Об иллюстрации на обложке

На обложке книги «Apache Pulsar в действии» размещен рисунок «Cosaque», или «Казак», взятый из коллекции изображений одежды из различных стран Жака Грассе де Сен-Совёр (Jacques Grasset de Saint-Sauveur, 1757–1810) под названием «Costumes civils actuels de tous les peuples connus», опубликованной во Франции в 1788 г. Каждая иллюстрация тщательно прорисована и раскрашена от руки. Богатое разнообразие коллекции Грассе де Сен-Совёра живо напоминает нам о том, насколько далекими в культурном отношении были города и регионы мира всего 200 лет назад. Изолированные друг от друга люди говорили на разных диалектах и языках. На улицах или в сельской местности было легко определить, где они живут и какова их профессия или положение в обществе, просто по их одежде.

Наша одежда изменилась с тех пор, и разнообразие регионов, столь богатое в те далекие времена, исчезло. Сейчас трудно отличить жителей разных континентов, не говоря уже о разных городах, регионах или странах. Возможно, мы променяли культурное разнообразие на более разнообразную личную жизнь – и определенно на более разнообразную и динамичную технологическую среду.

В наше время, когда трудно отличить одну книгу по информатике от другой, издательство Manning воздает должное изобретательности и инициативности компьютерного бизнеса, создавая обложки книг на основе богатого разнообразия региональной культуры двухвековой давности, оживленные иллюстрациями из коллекции Грассе де Сен-Совёра.

Часть I

Знакомство с Apache Pulsar

Корпоративные системы обмена сообщениями (enterprise messaging systems – EMS) предназначены для расширения слабо связанных архитектур, чтобы позволить географически распределенным системам передавать информацию друг другу посредством обмена сообщениями через простой API, поддерживающий две основные операции: публикацию сообщения и подписку на тему (для чтения сообщений). За свою более чем 40-летнюю историю корпоративные системы обмена сообщениями породили несколько важных стилей архитектуры распределенного программного обеспечения:

- программирование вызова удаленной процедуры (remote procedure call – RPC) с использованием таких технологий, как COBRA и Amazon Web Services, которые позволяют программам, написанным на различных языках, напрямую взаимодействовать друг с другом;

- ориентированное на сообщения программное обеспечение среднего звена (messaging-oriented middleware – MOM) для интеграции корпоративных приложений, примером которого является Apache Camel, позволяющее различным системам обмениваться информацией в общем формате сообщений, использующем XML или аналогичный самоописываемый формат;
- сервис-ориентированную архитектуру (service-oriented architecture – SOA), расширяющую стиль модульного программирования по контракту и позволяющую компоновать приложения из сервисов, которые объединяются особым способом для выполнения необходимой бизнес-логики;
- архитектуру, управляемую событиями (event-driven architecture – EDA), расширяющую возможности генерации, обнаружения и ответной реакции на отдельные изменения состояния, обозначаемые как события, с написанием кода, который обнаруживает и реагирует на эти отдельные события. Этот стиль в определенной степени был применен для удовлетворения потребности в обработке непрерывных потоков данных в масштабе интернета, например журнальных записей сервера и цифровых событий, таких как история (маршруты) посещений веб-сайтов.

Корпоративные системы обмена сообщениями (EMS) играют главную роль в каждом из этих стилей архитектуры, поскольку выступают в качестве внутренней технологии, позволяющей распределенным компонентам обмениваться информацией друг с другом, сохраняя промежуточные сообщения и своевременно распространяя их для всех назначенных потребителей. Главное различие между обменом данными через EMS и некоторыми другими механизмами передачи данных исключительно на основе сетевой среды заключается в том, что EMS изначально спроектирована для гарантированной доставки сообщений. Если событие опубликовано в EMS, то оно будет сохранено и передано всем назначенным получателям, в отличие от вызова интернет-микросервисов на основе протокола HTTP, который может потерять событие из-за сбоя сети.

Эти сохраняемые в EMS сообщения также зарекомендовали себя как ценные источники информации для организаций, которые можно анализировать, чтобы извлечь больше пользы для бизнеса. Представьте себе хранилище информации о поведении клиентов, которую предоставляет поток истории посещений ресурсов компании. Обработка этих типов источников данных называется потоковой обработкой, поскольку вы в буквальном смысле обрабатываете неограниченный поток данных. Именно поэтому существует большой интерес к обработке этих потоков с помощью аналитических инструментов, таких как Apache Flink или Spark.

В первой части книги представлен обзор развития корпоративных систем обмена сообщениями, где главное внимание сосредоточено на основных функциональных возможностях, добавленных на каждом этапе развития. Знание этих основ поможет вам лучше понять методы сравнения различных систем обмена сообщениями с учетом сильных и слабых сторон каждого поколения и функций, добавляемых в процессе развития. В итоге, как я надеюсь, вы поймете, почему Apache Pulsar является еще одним шагом вперед в генеалогии EMS и заслуживает особого внимания как весьма важный компонент инфраструктуры вашей компании.

В главе 1 представлено краткое введение в Apache Pulsar и его место в 40-летней истории развития систем обмена сообщениями, а также сравнение с различными более ранними платформами обмена сообщениями. В главе 2 более подробно рассматриваются особенности физической архитектуры Pulsar и возможности многозвенной архитектуры, позволяющей масштабировать уровни хранения и обработки данных независимо друг от друга. Здесь также описаны некоторые часто применяемые паттерны потребления сообщений, различия между ними и способы их поддержки в Pulsar. В главе 3 показано, как можно взаимодействовать с Apache Pulsar из командной строки и с использованием его API. После изучения этой главы вы будете вполне уверенно запускать локальный экземпляр Apache Pulsar и взаимодействовать с ним.

1 Введение в Apache Pulsar

Темы:

- история развития корпоративных систем обмена сообщениями;
- сравнение Apache Pulsar с существующими корпоративными системами обмена сообщениями;
- отличие сегмент-ориентированного хранилища Pulsar от ориентированной на разделы модели хранения, используемой в Apache Kafka;
- варианты использования из реальной практики, в которых Pulsar применяется для потоковой обработки, и объяснение, почему вы должны рассматривать использование Apache Pulsar.

После завершения разработки компанией Yahoo! в 2013 г. исходный код Pulsar впервые был открыт в 2016 г. и только через 15 месяцев после присоединения к программе инкубации Apache Software Foundation приобрел статус проекта высшего уровня. Apache Pulsar был спроектирован и разработан с нуля для устранения недостатков в существующих на тот момент систем обмена сообщениями с открытым исходным кодом, чтобы обеспечить ранее отсутствующую поддержку многоарендной архитектуры, георепликации и строгие гарантии надежности хранения данных.

Сайт Apache Pulsar определяет это программное обеспечение как распределенную систему обмена сообщениями по схеме публи-

кация–подписка (pub-sub), обеспечивающую минимальное время задержки между конечными пунктами и при публикации, гарантированную доставку сообщений, хранение данных без потерь и отсутствие специально выделенного сервера, а также предоставляющую упрощенную вычислительную рабочую среду (фреймворк) для обработки потоковых данных. Apache Pulsar предоставляет три главные функции для обработки больших наборов данных:

- обмен сообщениями в реальном времени – позволяет географически распределенным приложениям и системам передавать информацию друг другу в асинхронном режиме посредством обмена сообщениями. Основная цель Pulsar – предоставить эту возможность как можно более широкой группе клиентов с поддержкой многих языков программирования и двоичных протоколов обмена сообщениями;
- обработка (вычисления) в реальном времени – предоставляет возможность выполнять определенные пользователем вычислительные операции с сообщениями, хранящимися непосредственно внутри Pulsar, без необходимости использования внешней вычислительной системы для выполнения основных операций преобразования, таких как обогащение данных, фильтрация и агрегации;
- масштабируемая система хранения – независимый уровень хранения Pulsar и поддержка многоуровневого хранилища обеспечивает сохранение данных сообщения в течение неограниченного времени, т. е. так долго, как необходимо. Не существует никаких физических ограничений по объему данных, которые могут быть сохранены и доступны в Pulsar.

1.1. Корпоративные системы обмена сообщениями

Обмен сообщениями (messaging) – общий термин, используемый для описания передачи данных между производителями и потребителями. Следовательно, существует несколько различных технологий и протоколов, которые развивались в течение многих лет для обеспечения этой функции. Большинство людей хорошо знакомы с такими системами обмена сообщениями, как электронная почта (email), передача текстовых сообщений и приложения оперативной передачи сообщений, включая WhatsApp и Facebook Messenger. Системы обмена сообщениями из этой категории предназначены для передачи текстовых данных и изображений через интернет между двумя и более участниками. Более продвинутые системы оперативной передачи сообщений также поддерживают протокол Voice over IP (VoIP) и функции видеочата. Все эти систе-

мы изначально были предназначены для поддержки межличностного персонального обмена информацией по доступным в текущий момент каналам.

Другая категория систем обмена сообщениями, с которой люди уже знакомы, – потоковые сервисы видео по запросу, например Netflix или Hulu, которые отправляют потоковый видеоконтент одновременно многочисленным подписчикам. Такие сервисы видеопотока являются примерами однонаправленной широковещательной (одно сообщение многим потребителям) передачи данных потребителям, которые подписались на существующий канал, чтобы получать его контент. Хотя такие типы приложений приходят на ум при использовании терминов «системы обмена сообщениями» или «потоковая обработка», мы, с учетом темы этой книги, все же сосредоточимся на корпоративных системах обмена сообщениями.

Корпоративная система обмена сообщениями (enterprise messaging system – EMS) – это программное обеспечение, предоставляющее реализацию различных протоколов передачи сообщений, например DDS (data distribution service – сервис распространения данных), AMQP (advanced message queuing protocol – расширенный протокол организации очереди сообщений), MSMQ (Microsoft message queuing – протокол организации очереди сообщений Microsoft) и пр. Эти протоколы поддерживают отправку и прием сообщений между распределенными системами и приложениями в асинхронном режиме. Но асинхронный обмен данными был возможен не всегда, особенно на самом раннем этапе использования распределенных систем, когда преобладающими были архитектуры клиент–сервер и вызов удаленной процедуры (RPC). Первыми примерами применения RPC стали протокол SOAP (simple object access protocol) и REST (representational state transfer – передача репрезентативного состояния) на основе веб-сервисов, взаимодействующих друг с другом через фиксированные конечные точки. В обеих архитектурах, когда процесс требует взаимодействия с удаленным сервисом, необходимо сначала определить удаленную локацию этого сервиса через сетевой механизм службы обнаружения, а затем удаленно вызвать требуемый метод, используя корректные параметры и типы, как показано на рис. 1.1.

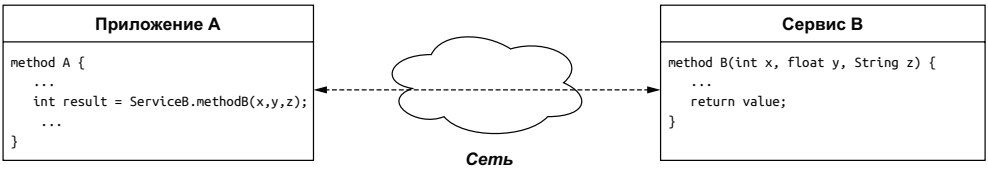

Рис. 1.1. В архитектуре RPC приложение вызывает процедуру сервиса, который работает на другом хосте, поэтому должно ждать возврата из этой процедуры, прежде чем можно будет продолжить обработку

Затем вызывающее приложение должно ждать возврата из вызванной процедуры, прежде чем можно будет продолжить обработку. Принцип синхронизации, присущий этим архитектурам, неизбежно замедлял работу приложений на их основе. Кроме того, существовала вероятность того, что удаленный сервис окажется недоступным в течение некоторого интервала времени, что требовало от разработчика приложения использования методик защитного (безопасного) программирования для определения такого условия и соответствующей реакции.

В отличие от коммуникационных каналов «точка–точка», используемых в программировании RPC, где необходимо ждать, когда вызванная процедура предоставит ответ, EMS позволяет удаленным приложениям и сервисам обмениваться информацией через промежуточный сервис, а не напрямую друг с другом. Вместо обязательного установления прямого сетевого коммуникационного канала между вызывающим и принимающим приложениями, через который происходит обмен параметрами, EMS можно использовать для сохранения этих параметров в форме сообщения, и они будут гарантированно доставлены назначенному получателю для обработки. Это позволяет вызывающей стороне отправлять запрос асинхронно и ждать ответа от сервиса, который она пытается вызвать. Это также позволяет сервису вернуть свой ответ в асинхронном режиме с публикацией результата в EMS с итоговой доставкой исходной вызывающей стороне. Такое разделение операций стимулирует разработку асинхронных приложений, предоставляя стандартизированный, надежный внутренний компонент коммуникационного канала, служащий постоянным буфером для обработки данных, даже если некоторые другие компоненты находятся в режиме офлайн, как можно видеть на рис. 1.2.

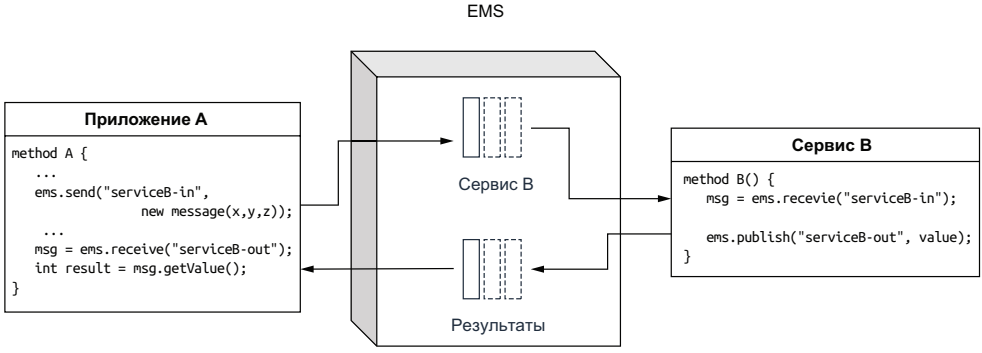

Рис. 1.2. EMS позволяет распределенным приложениям и сервисам обмениваться информацией в асинхронном режиме

EMS обеспечивает создание слабо связанных архитектур, предоставляя независимо разработанные программные компоненты, распределенные по различным системам для обмена данными друг

с другом с использованием структурированных сообщений. Такие схемы сообщений обычно определяются в форматах, независимых от языков программирования, например XML, JSON или Avro IDL, что позволяет разрабатывать компоненты на любом языке, поддерживающем эти форматы.

1.1.1. Основные функциональные возможности

После ознакомления с концепцией корпоративных систем обмена сообщениями и предварительного описания контекста типов задач, который эти системы должны использовать для решения, уточним определение EMS на основе предоставляемых ими функциональных возможностей.

Асинхронный обмен данными

Системы обмена сообщениями позволяют сервисам и приложениям обмениваться данными без блокировок, т. е. от отправителя и получателя сообщения не требуется одновременное взаимодействие с системой обмена сообщениями (или друг с другом). Система обмена сообщениями будет хранить сообщения до тех пор, пока все назначенные получатели не примут их.

Сохранение сообщений

В отличие от механизмов обмена сообщениями на сетевой основе, где сообщения существуют только в сетевой среде, как, например, RPC, сообщения, публикуемые в системе обмена сообщениями, сохраняются на диске до тех пор, пока они не будут доставлены. Недоставленные сообщения могут храниться в течение часов, дней и даже недель, и большинство систем обмена сообщениями позволяют пользователю определять стратегию хранения.

Подтверждение приема

От систем обмена сообщениями требуется сохранение сообщений до тех пор, пока все назначенные получатели не примут их, следовательно, необходим механизм, с помощью которого потребители сообщений могут подтвердить успешную доставку и обработку сообщений. Это позволяет системе обмена сообщениями удалять все успешно доставленные сообщения и повторять операцию доставки тем потребителям, которые пока еще не получили предназначенные для них сообщения.

Потребление сообщений

Очевидно, что система обмена сообщениями практически бесполезна, если не предоставляет механизм, с помощью которого назначенные получатели могут потреблять сообщения. Прежде всего и главным образом EMS обязана гарантировать, что все полученные ею сообщения непременно доставлены адресатам. Нередко сообще-

ния могут быть предназначены для нескольких (многих) потребителей, и EMS должна обеспечивать сохранность и целостность информации о том, какие сообщения уже были доставлены и кому.

1.2. Паттерны потребления сообщений

При использовании EMS вы получаете возможность публикации сообщений в теме или в очереди, но между этими двумя объектами существуют принципиальные различия. Тема (topic) поддерживает одновременное существование многочисленных потребителей одного сообщения. Любое сообщение, опубликованное в теме, автоматически передается (в широковещательном режиме) всем потребителям, подписавшимся (subscribed) на эту тему. Любое количество потребителей может подписаться на любую тему, чтобы получать преданную информацию, – точно так же любое количество пользователей может подписаться на Netflix и получать потоковый контент этого сервиса.

1.2.1. Обмен сообщениями по схеме публикация–подписка

При обмене сообщениями по схеме публикация–подписка производители публикуют сообщения в именованных каналах, называемых темами (topics). После этого потребители могут подписаться на существующие темы, чтобы получать входящие сообщения. Канал сообщений со схемой публикация–подписка (pub-sub) принимает входящие сообщения от нескольких производителей и сохраняет их в точном порядке получения. Но это отличается от очереди сообщений на стороне потребителя, потому что здесь обеспечена поддержка многих потребителей, принимающих каждое сообщение в теме через механизм подписки, как показано на рис. 1.3.

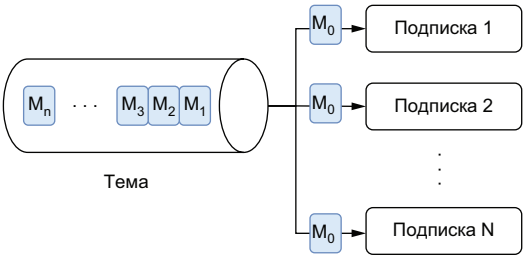

Рис. 1.3. При потреблении сообщений по схеме публикация–подписка каждое сообщение доставляется всем подписавшимся, установленным в этой теме. В данном случае сообщение M0 было доставлено всем подписавшимся от 1 до N включительно

Системы обмена сообщениями по схеме публикация–подписка идеально подходит для вариантов использования, в которых требуется, чтобы потребители принимали каждое сообщение, или для вариантов, в которых порядок приема и обработки сообщений

чрезвычайно важен для сохранения корректного состояния системы. Рассмотрим вариант сервиса курса акций, который может использоваться большим количеством систем. Здесь важно не только то, чтобы сервисы принимали все сообщения, но не менее важно, чтобы изменения курса поступали в правильном порядке.

1.2.2. Очередь сообщений

С другой стороны, очереди (queues) обеспечивают семантику доставки сообщений по схеме «первым вошел, первым вышел» (first in, first out – FIFO) одному или нескольким конкурирующим потребителям, как показано на рис. 1.4. В очередях сообщения доставляются в порядке их получения, и только один потребитель получает и обрабатывает одно конкретное сообщение, а не все потребители – каждое сообщение. Эта схема хорошо подходит для организации очереди сообщений, представляющих события, активизирующие выполнение некоторой работы, например заявки, поступающие в центр обработки и выполнения заказов для регулирования. В этом сценарии необходимо, чтобы каждый заказ был обработан только один раз.

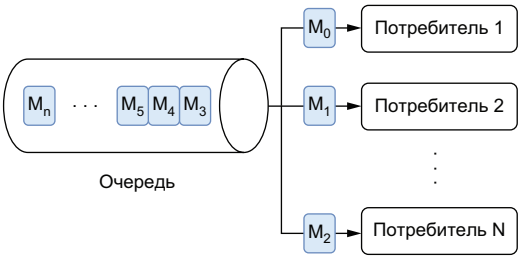

Рис. 1.4. При потреблении сообщений на основе очереди каждое сообщение доставляется исключительно одному потребителю. В данном случае сообщение M_0 было принято потребителем 1, M_1 – потребителем 2 и т. д.

Очереди сообщений могут с легкостью поддерживать более высокие скорости потребления, масштабируя количество потребителей в случае слишком большого числа сообщений, ожидающих обработки. Для уверенности в том, что сообщение обрабатывается ровно один раз, каждое сообщение обязательно должно быть удалено из очереди после успешной обработки и подтверждения от потребителя. Такая гарантия исключительно одноразовой обработки делает очереди сообщений идеальным средством для вариантов использования в очередях рабочих операций.

Если возникает критическая ошибка потребителя (т. е. отсутствие подтверждения приема в течение заданного интервала времени), сообщение будет повторно передано другому потребителю. В таком сценарии это сообщение, вероятнее всего, будет обработано вне существующего порядка. Таким образом, очереди сообщений подходят для тех вариантов использования, в которых чрезвычай-

но важно, чтобы каждое сообщение обрабатывалось один и только один раз, но порядок обработки сообщений не имеет значения.

1.3. История развития систем обмена сообщениями

После того как мы точно определили, из чего состоит EMS и какие основные функциональные возможности предоставляет эта система, я хотел бы привести краткий исторический обзор развития систем обмена сообщениями. Системы обмена сообщениями существовали в течение нескольких десятилетий и эффективно использовались во многих организациях, поэтому Apache Pulsar не является какой-то принципиально новой технологией, появившейся внезапно, скорее это очередной этап в развитии систем обмена сообщениями. Описывая исторический контекст, я надеюсь, что читатели смогут лучше понять суть сравнения Pulsar с существующими системами обмена сообщениями.

1.3.1. Системы обмена сообщениями общего назначения

Прежде чем перейти к специализированным системам обмена сообщениями, необходимо привести упрощенное представление систем обмена сообщениями, чтобы выделить внутренние компоненты, которые имеют все подобные системы. Определение этих основных функциональных средств сформирует основу для сравнения систем обмена сообщениями, существовавших в различное время.

На рис. 1.5 можно видеть, что любая система обмена сообщениями состоит из двух основных уровней, и каждый уровень имеет собственные специальные обязанности, которые мы рассмотрим позже. Мы будем изучать развитие систем обмена сообщениями с учетом этих двух уровней, чтобы правильно классифицировать и сравнивать различные системы, включая Apache Pulsar.

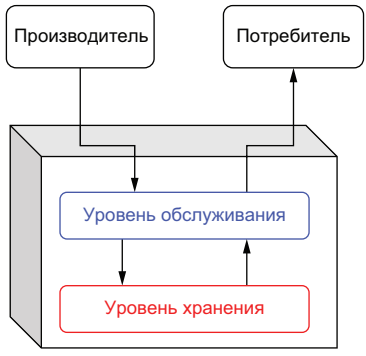

Рис. 1.5. Каждую систему обмена сообщениями можно разделить на два отдельных архитектурных уровня

Уровень обслуживания

Уровень обслуживания – это концептуальный уровень в EMS, который взаимодействует непосредственно с производителями и потребителями сообщений. Его главная задача – принимать входящие сообщения и направлять их в один или несколько пунктов назначения. Таким образом, этот уровень производит обмен данными через один или несколько поддерживаемых протоколов обмена сообщениями, например DDS, AMQP или MSMQ. Следовательно, уровень обслуживания в высшей степени зависит от пропускной способности сети для обмена данными и от ЦП для преобразования протоколов обмена сообщениями.

Уровень хранения

Уровень хранения – это концептуальный уровень в EMS, который отвечает за надежное сохранение и извлечение сообщений. Он взаимодействует непосредственно с уровнем обслуживания для предоставления запрошенных сообщений и обеспечивает правильный порядок сообщений. Следовательно, уровень хранения в высшей степени зависит от дисковой памяти для хранения сообщений.

1.3.2. Программное обеспечение среднего звена, ориентированное на сообщения

Первой категорией систем обмена сообщениями часто называют программное обеспечение среднего (связующего) звена, ориентированное на сообщения (message-oriented middleware – MOM), которое было предназначено для обеспечения обмена данными между процессами и интеграции приложений между распределенными системами, работающими в различных сетевых средах и операционных системах. Одной из наиболее известных реализаций MOM была IBM WebSphere MQ, внедренная в 1993 г.

Самые ранние реализации были спроектированы для развертывания на одном компьютере, который часто размещался глубоко внутри центра данных компании. Такое решение не только создавало единственную критическую точку отказа, но также означало, что масштабируемость системы ограничена возможностями физического аппаратного обеспечения хоста, поскольку этот единственный сервер отвечал за обработку всех клиентских запросов и хранение всех сообщений, как показано на рис. 1.6. Количество производителей и потребителей, одновременно работающих в таких MOM-системах с единственным сервером, было ограничено пропускной способностью сетевой карты, а объем хранимой информации – емкостью физического диска на сервере.