Оглавление

Содержание

Предисловие

Нам обоим посчастливилось работать инженерами по обработке данных в интересные и трудные времена.

Хорошо это или плохо, но многие компании и организации осознают, что данные играют ключевую роль в управлении их операциями и их улучшении. Последние разработки в области машинного обучения и искусственного интеллекта открыли множество новых возможностей для извлечения выгоды.

Однако внедрение процессов, ориентированных на данные, часто бывает затруднительным, поскольку обычно требуется координировать работу в различных гетерогенных системах и связывать все воедино аккуратно и своевременно для последующего анализа или развертывания продукта.

В 2014 году инженеры Airbnb осознали проблемы управления сложными рабочими процессами данных внутри компании. Чтобы разрешить их, они приступили к разработке Airflow: решения с открытым исходным кодом, позволяющего писать и планировать рабочие процессы, а также отслеживать их выполнение с помощью встроенного веб-интерфейса.

Успех этого проекта быстро привел к тому, что его приняли в рамках Apache Software Foundation, сначала в качестве проекта инкубатора в 2016 году, а затем в качестве проекта верхнего уровня в 2019 году. В результате многие крупные компании теперь используют Airflow для управления многочисленными критически важными процессами обработки данных.

Работая консультантами в GoDataDriven, мы помогли клиентам внедрить Airflow в качестве ключевого компонента в проектах, связанных с созданием озер и платформ данных, моделей машинного обучения и т. д. При этом мы поняли, что передача этих решений может быть непростой задачей, поскольку такие сложные инструменты, как Airflow, трудно освоить в одночасье. По этой причине мы также разработали программу по обучению Airflow в GoData-Driven, часто

организовывали семинары и участвовали в них, чтобы поделиться своими знаниями, мнениями и даже пакетами с открытым исходным кодом. В совокупности эти усилия помогли нам изучить тонкости работы с Airflow, которые не всегда было легко понять, используя доступную нам документацию.

В этой книге мы хотим предоставить исчерпывающее введение в Airflow, которое охватывает все, от построения простых рабочих процессов до разработки собственных компонентов и проектирования / управления развертываниями Airflow. Мы намерены дополнить блоги и другую онлайн-документацию, объединив несколько тем в одном месте в кратком и понятном формате. Тем самым мы надеемся придать вам стимул в работе с Airflow, опираясь на опыт, который мы накопили, и преодолевая трудности, с которыми мы столкнулись за последние годы.

О книге

Эта книга была написана, чтобы помочь вам реализовать рабочие процессы (или конвейеры), ориентированные на обработку данных с помощью Airflow. Она начинается с концепций и механики, участвующих в программном построении рабочих процессов для Apache Airflow с использованием языка программирования Python. Затем мы переходим к более подробным темам, таким как расширение Airflow путем создания собственных компонентов и всестороннего тестирования рабочих процессов. Заключительная часть книги посвящена проектированию и управлению развертывания Airflow, затрагивая такие темы, как безопасность и проектирование архитектур для облачных платформ.

Кому адресована эта книга

Эта книга написана для специалистов и инженеров по обработке данных, которые хотят разрабатывать базовые рабочие процессы в Airflow, а также для инженеров, интересующихся более сложными темами, такими как создание собственных компонентов для Airflow или управление развертываниями Airflow. Поскольку рабочие процессы и компоненты Airflow построены на языке Python, мы ожидаем, что у читателей имеется промежуточный опыт программирования на Python (т. е. они хорошо умеют создавать функции и классы Python, имеют представление о таких понятиях, как * args и ** kwargs и т. д.). Также будет полезен опыт работы с Docker, поскольку большинство примеров кода запускаются с использованием Docker (хотя при желании их можно запускать локально).

Структура книги

Книга состоит из четырех разделов, которые охватывают 18 глав.

Часть I посвящена основам Airflow. В ней объясняется, что такое Airflow, и изложены его основные концепции:

- в главе 1 обсуждается концепция рабочих процессов / конвейеров обработки данных и их создание с помощью Apache Airflow. В ней также рассказывается о преимуществах и недостатках Airflow по сравнению с другими решениями, в том числе приводятся ситуации, при которых вы, возможно, не захотите использовать Apache Airflow;
- глава 2 посвящена базовой структуре конвейеров в Apache Airflow (также известных как ОАГ), с объяснением различных задействованных компонентов и их совместимости;
- в главе 3 показано, как использовать Airflow для планирования работы конвейеров через повторяющиеся промежутки времени, чтобы можно было (например) создавать конвейеры, которые постепенно загружают новые данные с течением времени. В этой главе также рассматриваются тонкости механизма планирования Airflow, который часто является источником путаницы;
- в главе 4 показано, как использовать механизмы создания шаблонов в Airflow для динамического включения переменных в определения конвейера. Это позволяет ссылаться на такие вещи, как даты выполнения расписания в конвейерах;
- в главе 5 демонстрируются различные подходы к определению отношений между задачами в конвейерах, что позволяет создавать более сложные структуры конвейеров с ветками, условными задачами и общими переменными.

В части II подробно рассматривается использование более сложных тем, включая взаимодействие с внешними системами, создание собственных компонентов и разработку тестов для ваших конвейеров:

- в главе 6 показано, как запускать рабочие процессы другими способами, не связанными с фиксированным расписанием, такими как загрузка файлов или вызов по протоколу HTTP;
- в главе 7 демонстрируются рабочие процессы с использованием операторов, которые управляют различными задачами вне Airflow, что позволяет создавать поток событий через системы, которые не связаны между собой;
- в главе 8 объясняется, как создавать собственные компоненты для Airflow, которые позволяют повторно использовать функциональные возможности в разных конвейерах или интегрироваться с системами, не поддерживающими встроенные функции Airflow;
- в главе 9 обсуждаются различные варианты тестирования рабочих процессов Airflow, затрагиваются свойства операторов и способы их применения во время тестирования;

- в главе 10 показано, как использовать рабочие процессы на базе контейнеров для выполнения задач конвейера в Docker или Kubernetes, а также обсуждаются преимущества и недостатки этих подходов.

Часть III посвящена применению Airflow на практике и затрагивает такие темы, как передовые методы, запуск и обеспечение безопасности Airflow и последний демонстрационный пример:

- в главе 11 рассказывается о передовых методах построения конвейеров, которые помогут вам разрабатывать и внедрять эффективные и удобные в сопровождении решения;
- в главе 12 подробно описано несколько тем, которые необходимо учитывать при запуске Airflow в промышленном окружении, например архитектуры для горизонтального масштабирования, мониторинга, журналирования и оповещений;
- в главе 13 обсуждается, как обезопасить установку Airflow, чтобы избежать нежелательного доступа и минимизировать воздействие в случае взлома;
- в главе 14 показан пример проекта Airflow, в котором мы периодически обрабатываем поездки с использованием сервисов Yellow Cab и Citi Bikes по Нью-Йорку, чтобы определить самый быстрый способ передвижения между районами.

В части IV исследуется, как запускать Airflow в облачных платформах. Она включает такие темы, как проектирование развертываний Airflow для облачных платформ и использование встроенных операторов для взаимодействия с облачными сервисами:

- в главе 15 дается общее введение, в котором рассказывается, какие компоненты Airflow участвуют в (облачных) развертываниях, представлена идея, лежащая в основе облачных компонентов, встроенных в Airflow, и рассматриваются варианты самостоятельной реализации развертывания в облаке в сравнении с использованием управляемого решения;
- глава 16 посвящена облачной платформе Amazon AWS. Здесь рассказывается о решениях для развертывания Airflow на AWS и демонстрируется, как применять определенные компоненты для использования сервисов AWS;
- в главе 17 разрабатываются развертывания и демонстрируются облачные компоненты для платформы Microsoft Azure;
- в главе 18 рассматриваются развертывания и облачные компоненты для платформы Google GCP.

Тем, кто плохо знаком с Airflow, следует прочитать главы 1 и 2, чтобы получить внятное представление о том, что такое Airflow, и его возможностях. В главах 3–5 представлена важная информация об основных функциях Airflow. В остальной части книги обсуждаются такие темы, как создание собственных компонентов, тестирование, передовые практики и развертывание, и ее можно читать в произвольном порядке в зависимости от конкретных потребностей читателя.

О коде

Весь исходный код в листингах или тексте набран моноширинным шрифтом, чтобы отделить его от обычного текста. Иногда также используется **жирный шрифт**, чтобы выделить код, который изменился по сравнению с предыдущими шагами в главе, например когда к существующей строке кода добавляется новая функция.

Во многих случаях оригинальный исходный код был переформатирован; мы добавили разрывы строк и переработали отступы, чтобы уместить их по ширине книжных страниц. В редких случаях, когда этого оказалось недостаточно, в листинги были добавлены символы продолжения строки (➥). Кроме того, комментарии в исходном коде часто удаляются из листингов, когда описание кода приводится в тексте. Некоторые листинги сопровождают аннотации, выделяющие важные понятия.

Ссылки на элементы в коде, сценарии или определенные классы, переменные и значения Airflow часто выделяются *курсивом*, чтобы их было легче отличить от окружающего текста.

Исходный код всех примеров и инструкции по их запуску с помощью Docker и Docker Compose доступны в нашем репозитории GitHub (https://github.com/BasPH/data-pipelines-with-apache-airflow), а также на сайте издательства «ДМК Пресс» www.dmkpress.com на странице с описанием соответствующей книги.

> **ПРИМЕЧАНИЕ** В приложении A представлены более подробные инструкции по запуску примеров кода.

Все примеры кода были протестированы с использованием Airflow 2.0. Большинство примеров также следует запускать в более старых версиях Airflow (1.10) с небольшими изменениями. Там, где это возможно, мы включили встроенные указатели, как это сделать. Чтобы вам было легче учитывать различия в путях импорта между Airflow 2.0 и 1.10, в приложении B представлен обзор измененных путей импорта.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторах

БАС ХАРЕНСЛАК – инженер по обработке данных в GoDataDriven, компании, занимающейся разработкой решений на основе данных, расположенной в Амстердаме, Нидерланды. Имея опыт разработки программного обеспечения и информатики, он любит работать над программным обеспечением и данными, как если бы они были сложными головоломками. Он предпочитает работать над ПО с открытым исходным кодом, участвует в проекте Apache Airflow и является одним из организаторов семинара Amsterdam Airflow.

ДЖУЛИАН ДЕ РУИТЕР – инженер машинного обучения, имеет опыт работы в области компьютерных наук и наук о жизни, а также имеет докторскую степень в области вычислительной биологии рака. Будучи опытным разработчиком программного обеспечения, он любит соединять миры науки о данных и инженерии, используя облачное программное обеспечение и программное обеспечение с открытым исходным кодом для разработки решений машинного обучения, готовых к промышленной эксплуатации. В свободное время он с удовольствием разрабатывает собственные пакеты Python, участвует в проектах с открытым исходным кодом и возится с электроникой.

Часть I

Приступаем к работе

Эта часть подготовит почву на пути к созданию конвейеров для всех видов процессов обработки данных с использованием Apache Airflow. Первые две главы нацелены на то, чтобы предоставить вам обзор того, что такое Airflow и что он умеет.

В главе 1 мы рассмотрим концепции конвейеров обработки данных и обозначим роль, которую выполняет Apache Airflow, помогая реализовать их. Чтобы оправдать ожидания, мы также сравним Airflow с другими технологиями и обсудим, когда он может или не может быть подходящим решением для вашего конкретного случая использования. Далее, в главе 2, вы узнаете, как реализовать свой первый конвейер в Airflow. После его создания мы также рассмотрим, как запустить конвейер и отслеживать его выполнение с помощью веб-интерфейса Airflow.

В главах 3–5 более подробно рассматриваются ключевые концепции Airflow, чтобы вы могли получить внятное представление об основных функциях Airflow.

Глава 3 посвящена семантике планирования, которое позволяет настроить Airflow для запуска конвейеров через равные промежутки времени. Это дает возможность (например) писать конвейеры, которые эффективно загружают и обрабатывают данные ежедневно, еженедельно или ежемесячно. Далее, в главе 4, мы обсудим механизмы шаблонизации в Airflow, которые позволяют динамически ссылаться на такие переменные, как даты выполнения, в ваших конвейерах. Наконец, в главе 5 мы рассмотрим различные подходы к определению

зависимостей задач в конвейерах, которые позволяют определять сложные иерархии задач, включая условные задачи, ветви и т. д.

Если вы новичок в Airflow, то рекомендуем убедиться, что вы понимаете основные концепции, описанные в главах 3–5, поскольку они являются ключом к его эффективному использованию.

Семантика планирования Airflow (описанная в главе 3) может сбивать с толку новичков, поскольку при первом знакомстве эта тема может показаться не совсем логичной.

После завершения части I вы должны быть достаточно подготовлены для написания собственных базовых конвейеров в Apache Airflow и быть готовыми погрузиться в более сложные темы, описанные в частях II–IV.

1 Знакомство с Apache Airflow

Эта глава:

- демонстрирует, как представить конвейеры обработки данных в рабочих процессах в виде графов задач;
- рассказывает, какое место занимает Airflow в экосистеме инструментов управления рабочими процессами;
- поможет определить, подходит ли вам Airflow.

Люди и компании все чаще ориентируются на данные и разрабатывают конвейеры обработки данных как часть своей повседневной деятельности. Объемы данных, задействованных в этих бизнес-процессах, за последние годы значительно увеличились – с мегабайтов в день до гигабайтов в минуту. Хотя обработка такого потока данных может показаться серьезной проблемой, с этими растущими объемами можно справиться с помощью соответствующих инструментов.

В этой книге основное внимание уделяется Apache Airflow, фреймворку для построения конвейеров обработки данных. Ключевая особенность Airflow заключается в том, что он позволяет легко создавать конвейеры обработки данных, запускаемых по расписанию, с использованием гибкой платформы Python, а также предоставляет множество строительных блоков, которые позволяют объединить огромное количество различных технологий, встречающихся в современных технологических ландшафтах.

Airflow очень напоминает паука, сидящего у себя в паутине: он находится в центре ваших процессов обработки данных и координирует работу, происходящую в различных (распределенных) системах. Таким образом, Airflow сам по себе не является инструментом обработки данных. Он управляет различными компонентами, которые отвечают за обработку ваших данных в конвейерах.

В этой главе мы сначала дадим краткое введение в конвейеры обработки данных в Apache Airflow. После этого обсудим ряд соображений, которые следует учитывать при оценке того, подходит ли вам Airflow, и опишем первые шаги по работе с ним.

1.1 Знакомство с конвейерами обработки данных

Обычно конвейеры обработки данных состоят из нескольких задач или действий, которые необходимо выполнить для достижения желаемого результата. Например, предположим, что нам нужно создать небольшое приложение для отображения прогноза погоды на неделю вперед (рис. 1.1). Чтобы реализовать такое приложение, отображающее прогноз в реальном времени, необходимо выполнить следующие шаги:

1. Получить данные прогноза погоды из API погоды.
2. Отфильтровать и преобразовать полученные данные (например, перевести температуру из шкалы Фаренгейта в шкалу Цельсия или наоборот), чтобы данные соответствовали нашей цели.
3. Передать преобразованные данные в приложение.

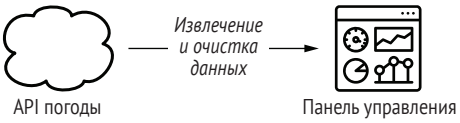

Рис. 1.1 Приложение для отображения прогноза погоды, извлекающее данные из внешнего API и отображающее их в своем интерфейсе

Как видите, этот относительно простой конвейер уже состоит из трех разных задач, каждая из которых выполняет свою часть работы. Более того, эти задачи необходимо выполнять в определенном порядке, поскольку (например) нет смысла пытаться преобразовать данные перед их получением. Точно так же нельзя отправлять новые данные в приложение, пока они не претерпят необходимые преобразования. Таким образом, мы должны убедиться, что этот неявный порядок задач также применяется при запуске данного процесса обработки данных.

1.1.1 Конвейеры обработки данных как графы

Один из способов сделать зависимости между задачами более явными – нарисовать конвейер обработки данных в виде графа. В этом представлении задачам соответствуют узлы графа, а зависимостям между задачами – направленные ребра между узлами задач. Направление ребра указывает направление зависимости, то есть ребро, направленное от задачи A к задаче B, указывает, что задача A должна быть завершена до того, как может начаться задача B. Такие графы обычно называются *ориентированными,* или *направленными*, потому что ребра имеют направление.

Применительно к нашему приложению отображения погоды граф обеспечивает довольно точное представление всего конвейера (рис. 1.2). Просто бегло взглянув на граф, можно увидеть, что наш конвейер состоит из трех разных задач. Помимо этого, направление ребер четко указывает порядок, в котором должны выполняться задачи, достаточно просто проследовать за стрелками.

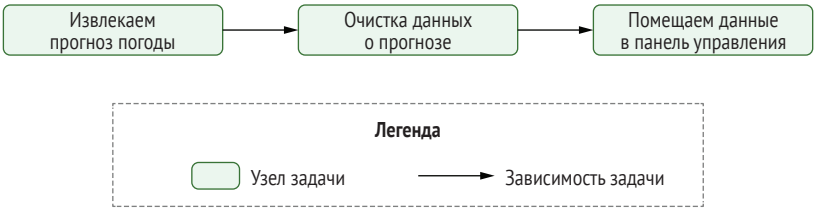

Рис. 1.2 Графовое представление конвейера обработки данных в приложении отображения погоды. Узлы обозначают задачи, а направленные ребра – зависимости между задачами (ребро, направленное от задачи A к задаче B, указывает, что задача A должна быть выполнена раньше задачи B)

Данный тип графа обычно называется *ориентированным ациклическим графом* (ОАГ), поскольку он содержит *ориентированные* ребра и у него нет никаких петель или циклов (*ациклический*). Такое ациклическое свойство чрезвычайно важно, так как оно предотвращает возникновение циклических зависимостей (рис. 1.3) между задачами (где задача A зависит от задачи B и наоборот). Эти циклические зависимости становятся проблемой при попытке выполнить граф, поскольку мы сталкиваемся с ситуацией, когда задача 2 может выполняться только после завершения задачи 3, а задача 3 может выполняться только после завершения задачи 2. Такая логическая непоследовательность приводит к тупиковой ситуации, при которой ни задача 2, ни задача 3 не могут быть запущены, что мешает выполнить граф.

Обратите внимание, что это представление отличается от представлений циклических графов, которые могут содержать циклы для иллюстрации итеративных частей алгоритмов (например), как это

обычно бывает во многих приложениях для машинного обучения. Однако ацикличность ОАГ используется Airflow (и многими другими инструментами управления рабочими процессами) для эффективного разрешения и выполнения этих графов задач.

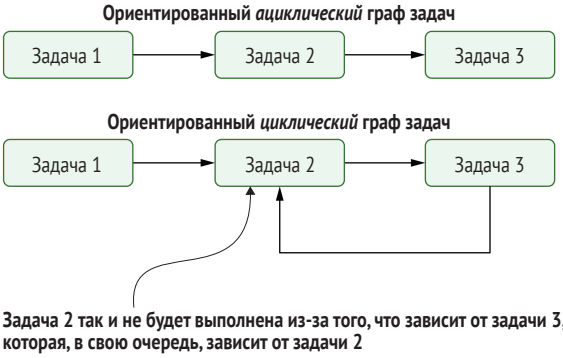

Рис. 1.3 Циклы на графах препятствуют выполнению задачи из-за круговой зависимости. В ациклических графах (вверху) есть четкий путь для выполнения трех разных задач. Однако в циклических графах (внизу) четкого пути выполнения уже нет из-за взаимозависимости между задачами 2 и 3

1.1.2 Выполнение графа конвейера

Прекрасное свойство этого представления состоит в том, что он предоставляет относительно простой алгоритм, который можно использовать для запуска конвейера. Концептуально этот алгоритм состоит из следующих шагов:

1. Для каждой открытой (= незавершенной) задачи в графе выполните следующие действия:
   - для каждого ребра, указывающего *на* задачу, проверьте, завершена ли «вышестоящая» задача на другом конце ребра;
   - если все вышестоящие задачи были выполнены, добавьте текущую задачу в очередь для выполнения.
2. Выполните задачи в очереди, помечая их как выполненные, как только они сделают свою работу.
3. Вернитесь к шагу 1 и повторите действия, пока все задачи в графе не будут выполнены.

Чтобы увидеть, как это работает, проследим за выполнением конвейера в нашем приложении (рис. 1.4). В первой итерации нашего алгоритма мы видим, что задачи *очистки* и *отправки* данных зависят от вышестоящих задач, которые еще не завершены. Таким образом, *зависимости* этих задач не удовлетворены, и их нельзя добавить в очередь выполнения. Однако у задачи *извлечения* данных нет входящих ребер, а это означает, что у нее нет неудовлетворенных вышестоящих зависимостей и, следовательно, ее можно добавить в очередь выполнения.

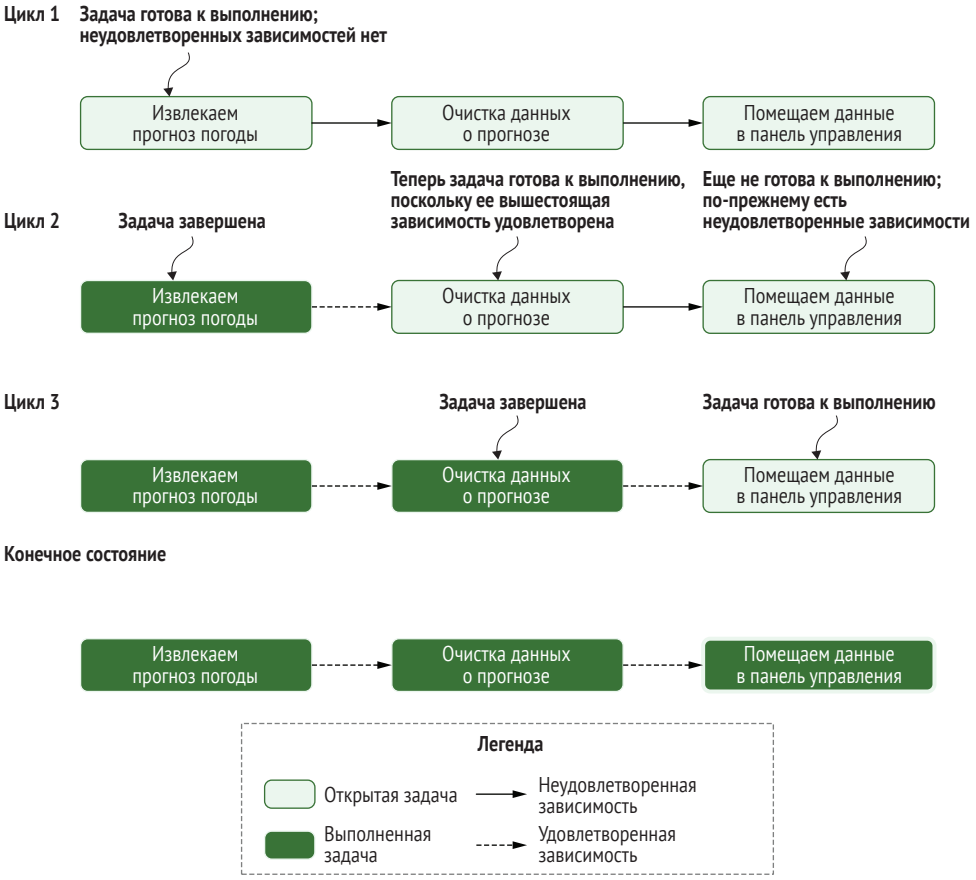

Рис. 1.4 Использование структуры ОАГ для выполнения задач в конвейере обработки данных в правильном порядке: здесь показано, как меняется состояние каждой задачи в каждой итерации алгоритма, что приводит к завершению выполнения конвейера (конечное состояние)

После завершения задачи *извлечения* данных можно перейти ко второй итерации и исследовать зависимости задач *очистки* и *отправки* данных. Теперь мы видим, что задачу *очистки можно* выполнить, поскольку ее вышестоящая зависимость удовлетворена (задача *извлечения* данных выполнена). Таким образом, ее можно добавить в очередь выполнения. Задачу *отправки* данных нельзя добавить в очередь, поскольку она зависит от задачи *очистки*, которую мы еще не запускали.

В третьей итерации, после завершения задачи *очистки*, задача *отправки* данных наконец готова к выполнению, поскольку ее вышестоящая зависимость от задачи *очистки* теперь удовлетворена. В результате мы можем добавить задачу в очередь на выполнение. После того как задача *отправки* данных завершится, у нас не останется задач для выполнения, и выполнение всего конвейера прекратится.

1.1.3 Графы конвейеров и последовательные сценарии

Хотя графовое представление конвейера обеспечивает интуитивно понятный обзор задач в конвейере и их зависимостей, вы можете задаться вопросом, почему бы просто не использовать простой сценарий для выполнения этой линейной цепочки из трех шагов. Чтобы проиллюстрировать преимущества подхода на базе графа, перейдем к более крупному примеру. Здесь к нам обратился владелец компании по производству зонтов, которого вдохновило наше приложение для отображения погоды и теперь он хотел бы попробовать использовать машинное обучение для повышения эффективности работы своей компании. Владельцу компании хотелось бы, чтобы мы реализовали конвейер обработки данных, который создает модель машинного обучения, увязывающую продажи зонтов с погодными условиями. Затем эту модель можно использовать для прогнозирования спроса на зонты в ближайшие недели в зависимости от прогноза погоды на это время (рис. 1.5).

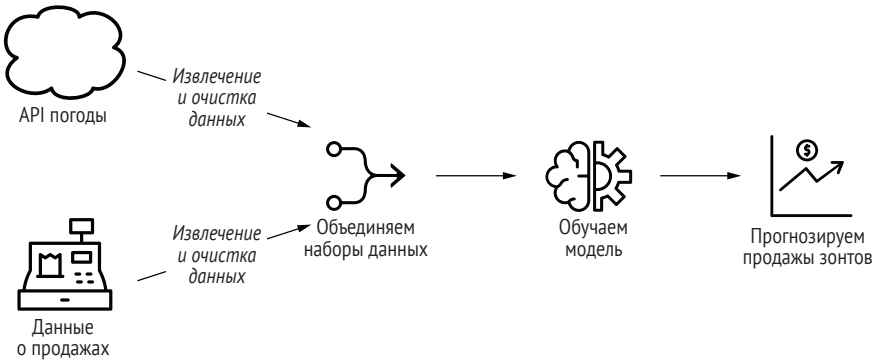

Рис. 1.5 Приложение оценки спроса на зонты, в котором предыдущие данные о погоде и продажах используются для обучения модели, прогнозирующей объем продаж в зависимости от прогноза погоды

Чтобы создать конвейер для обучения такой модели, нужно реализовать следующие шаги.

1. Подготовьте данные о продажах, выполнив такие действия:
   - получите данные о продажах из исходной системы;
   - выполните очистку/преобразование данных о продажах в соответствии с требованиями.
2. Подготовьте данные о погоде, выполнив следующие действия:
   - получите данные прогноза погоды из API;
   - выполните очистку/преобразование данных о погоде в соответствии с требованиями.
3. Объедините наборы данных о продажах и погоде, чтобы создать объединенный набор данных, который можно использовать в качестве входных данных для создания модели машинного обучения.

4 Обучите модель, используя объединенный набор данных.
5 Разверните модель, чтобы ее можно было использовать в бизнесе.

Данный конвейер может быть представлен с использованием того же графового представления, которое мы использовали раньше, путем изображения задач в виде узлов и зависимостей данных между задачами в виде ребер.

Одно из важных отличий от нашего предыдущего примера состоит в том, что первые этапы этого конвейера (получение и очистка данных о погоде и продажах) фактически независимы друг от друга, поскольку включают в себя два отдельных набора данных. Это ясно иллюстрируется двумя отдельными ветвями в графовом представлении конвейера (рис. 1.6), которые могут выполняться параллельно, если мы применяем наш алгоритм выполнения графа, лучше используя доступные ресурсы и потенциально уменьшая время работы конвейера по сравнению с последовательным выполнением задач.

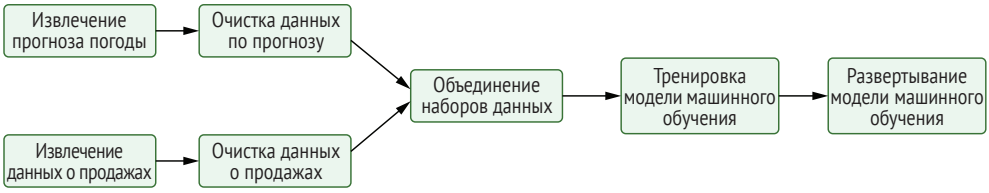

Рис. 1.6 Отсутствие зависимости между задачами, касающимися продаж и погоды в графовом представлении конвейера обработки данных для модели прогноза погоды, для оценки, каким будет спрос на зонты. Два набора задач извлечения и очистки данных независимы, поскольку включают в себя два разных набора данных (наборы данных о погоде и продажах). На эту независимость указывает отсутствие границ между двумя наборами задач

Еще одно полезное свойство графового представления состоит в том, что оно четко разделяет конвейеры на небольшие инкрементные задачи, вместо того чтобы иметь один монолитный сценарий или процесс, который выполняет всю работу. Хотя наличие одного сценария может поначалу казаться не такой уж большой проблемой, это может привести к неэффективности, когда задачи в конвейере будут давать сбой, поскольку нам пришлось бы перезапускать весь сценарий. Напротив, в графовом представлении нужно перезапустить только все неудачные задачи (и все нижестоящие зависимости).

1.1.4 Запуск конвейера с помощью диспетчеров рабочих процессов

Конечно, задача построения графов зависимых задач вряд ли является новой проблемой. За прошедшие годы было разработано множество решений для т. н. «управления рабочими процессами», чтобы

справиться с ней. Обычно эти решения позволяют определять и выполнять графы задач как рабочие процессы или конвейеры.

В табл. 1.1 перечислены некоторые известные диспетчеры рабочих процессов, о которых вы, возможно, слышали.

Таблица 1.1 Обзор известных диспетчеров рабочих процессов и их основных характеристик

Название	Создан[a]	Способ определения рабочих процессов	Написан на	Планирование	Бэкфиллинг (обратное заполнение)	Пользовательский интерфейс[b]	Платформа для установки	Горизонтальная масштабируемость
Airflow	Airbnb	Python	Python	Есть	Есть	Есть	Любая	Есть
Argo	Applatix	YAML	Go	Третья сторона[c]		Есть	Kubernetes	Есть
Azkaban	LinkedIn	YAML	Java	Есть	Нет	Есть	Любая	
Conductor	Netflix	JSON	Java	Нет		Есть	Любая	Есть
Luigi	Spotify	Python	Python	Нет	Есть	Есть	Любая	Есть
Make		Собственный предметно-ориентированный язык	C	Нет	Нет	Нет	Любая	Нет
Metaflow	Netflix	Python	Python	Нет		Нет	Любая	Есть
Nifi	NSA	Пользовательский интерфейс	Java	Есть	Нет	Есть	Любая	Есть
Oozie		XML	Java	Есть	Есть	Есть	Hadoop	Есть

a Некоторые инструменты изначально были созданы (бывшими) сотрудниками компании; однако все они имеют открытый исходный код и не представлены одной отдельной компанией.

b Качество и возможности пользовательских интерфейсов сильно различаются.

c https://github.com/bitphy/argo-cron.

Хотя у каждого из этих диспетчеров имеются свои сильные и слабые стороны, все они предоставляют схожие базовые функции, позволяющие определять и запускать конвейеры, содержащие несколько задач с зависимостями.

Одно из ключевых различий между этими инструментами заключается в том, как они определяют свои рабочие процессы. Например, такие инструменты, как Oozie, используют статические (XML) файлы для их определения, что обеспечивает понятный рабочий процесс, но ограниченную гибкость. Другие решения, такие как Luigi и Airflow, позволяют определять рабочие процессы как код, что дает бóльшую гибкость, но может представлять сложность для чтения и тестирования (в зависимости от навыков программирования у человека, реализующего процесс).

Другие ключевые различия заключаются в объеме функций, предоставляемых диспетчером. Например, такие инструменты, как Make и Luigi, не предоставляют встроенной поддержки для планирования рабочих процессов, а это означает, что вам понадобится дополнительный инструмент, например Cron, если вы хотите запускать рабочий процесс с регулярным расписанием. Другие инструменты могут предоставлять дополнительные функции, такие как планирование, мониторинг, удобные веб-интерфейсы и т. д., встроенные в платформу. Это означает, что вам не нужно самостоятельно объединять несколько инструментов, чтобы получить эти функции.

В целом выбор правильного решения для управления рабочим процессом, отвечающего вашим потребностям, потребует тщательного рассмотрения ключевых особенностей различных решений и того, насколько они соответствуют вашим требованиям. В следующем разделе мы подробнее рассмотрим Airflow, о котором и идет речь в этой книге, и изучим несколько ключевых функций, которые делают его особенно подходящим для обработки процессов или конвейеров, ориентированных на данные.

1.2 Представляем Airflow

В этой книге речь идет о Airflow, решении с открытым исходным кодом для разработки и мониторинга рабочих процессов. В данном разделе мы рассмотрим, в общих чертах, что делает Airflow, после чего перейдем к более подробному изучению того, подходит ли он вам.

1.2.1 Определение конвейеров в коде (Python) гибким образом

Подобно другим диспетчерам рабочих процессов, Airflow позволяет определять конвейеры или рабочие процессы как ОАГ задач. Эти графы очень похожи на примеры, обозначенные в предыдущем разделе, где задачи определены как узлы в графе, а зависимости – как направленные ребра между задачами.

В Airflow ОАГ определяются с помощью кода на языке Python в файлах, которые по сути являются сценариями Python, описывающими структуру соответствующего ОАГ. Таким образом, каждый файл ОАГ обычно описывает набор задач для данного графа и зависимости между задачами, которые затем анализируются Airflow для определения структуры графа (рис. 1.7). Помимо этого, эти файлы обычно содержат некоторые дополнительные метаданные о графе, сообщающие Airflow, как и когда он должен выполняться, и так далее. Подробнее об этом мы поговорим в следующем разделе.

Одно из преимуществ определения ОАГ Airflow в коде Python состоит в том, что этот программный подход обеспечивает большую гибкость при создании графа. Например, как будет показано позже,

вы можете использовать код Python для динамической генерации дополнительных задач в зависимости от определенных условий или даже для генерации целых графов на основе внешних метаданных либо файлов конфигурации. Такая гибкость дает возможность настраивать способ построения конвейеров, позволяя настроить Airflow в соответствии с вашими потребностями при создании конвейеров произвольной сложности.

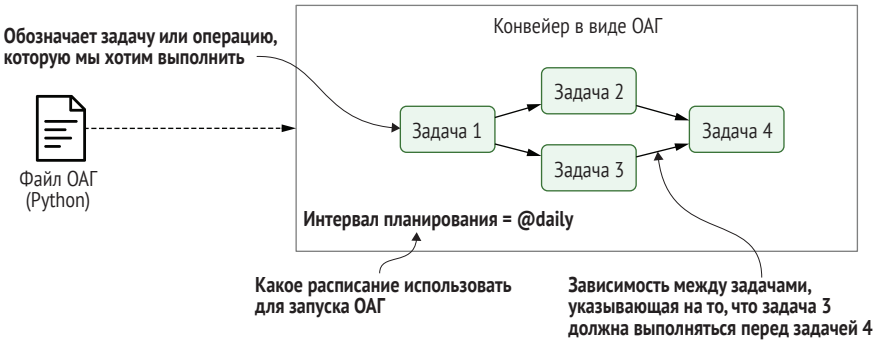

Рис. 1.7 Конвейеры Airflow определены в виде ОАГ с использованием кода Python в файлах ОАГ. Каждый такой файл обычно определяет один граф, который описывает различные задачи и их зависимости. Помимо этого, ОАГ также определяет интервал, который решает, когда Airflow выполняет граф

Помимо этой гибкости, еще одно преимущество того факта, что Airflow написан на языке Python, состоит в том, что задачи могут выполнять любую операцию, которую можно реализовать на Python. Со временем это привело к разработке множества расширений Airflow, позволяющих выполнять задачи в широком спектре систем, включая внешние базы данных, технологии больших данных и различные облачные сервисы, давая возможность создавать сложные конвейеры обработки данных, объединяющие процессы обработки данных в различных системах.

1.2.2 Планирование и выполнение конвейеров

После того как вы определили структуру вашего конвейера (конвейеров) в виде ОАГ, Airflow позволяет вам определить параметр `schedule_interval` для каждого графа, который точно решает, когда ваш конвейер будет запущен Airflow. Таким образом, вы можете дать указание Airflow выполнять ваш граф каждый час, ежедневно, каждую неделю и т. д. Или даже использовать более сложные интервалы, основанные на выражениях, подобных Cron.

Чтобы увидеть, как Airflow выполняет графы, кратко рассмотрим весь процесс, связанный с разработкой и запуском ОАГ. На высоком уровне Airflow состоит из трех основных компонентов (рис. 1.8):

- *планировщик Airflow* – анализирует ОАГ, проверяет параметр `schedule_interval` и (если все в порядке) начинает планировать задачи ОАГ для выполнения, передавая их воркерам Airflow;
- *воркеры*[1] (*workers*) *Airflow* – выбирают задачи, которые запланированы для выполнения, и выполняют их. Таким образом, они несут ответственность за фактическое «выполнение работы»;
- *веб-сервер Airflow* – визуализирует ОАГ, анализируемые планировщиком, и предоставляет пользователям основной интерфейс для отслеживания выполнения графов и их результатов.

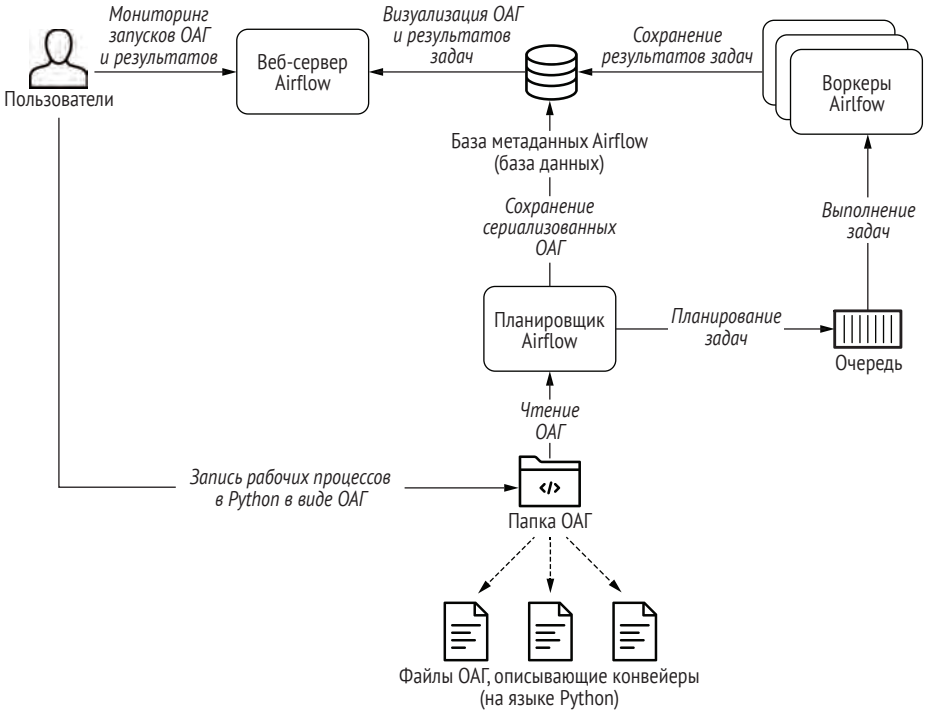

Рис. 1.8 Обзор основных компонентов Airflow (веб-сервер Airflow, планировщик и воркеры)

Сердцем Airflow, вероятно, является планировщик, поскольку именно здесь происходит бóльшая часть магии, определяющей, когда и как будут выполняться ваши конвейеры. На высоком уровне планировщик выполняет следующие шаги (рис. 1.9):

1 После того как пользователи написали свои рабочие процессы в виде ОАГ, файлы, содержащие эти графы, считываются плани-

[1] В англоязычных источниках также встречается термин worker process (рабочий процесс), который, по сути, означает то же самое. Чтобы избежать путаницы со словом workflow (рабочий процесс), в тексте книги используется слово «воркер». – *Прим. перев.*

ровщиком для извлечения соответствующих задач, зависимостей и интервалов каждого ОАГ.

2. После этого для каждого графа планировщик проверяет, все ли в порядке с интервалом с момента последнего чтения. Если да, то задачи в графе планируются к выполнению.
3. Для каждой задачи, запускаемой по расписанию, планировщик затем проверяет, были ли выполнены зависимости (= вышестоящие задачи) задачи. Если да, то задача добавляется в очередь выполнения.
4. Планировщик ждет несколько секунд, прежде чем начать новый цикл, перескакивая обратно к шагу 1.

Рис. 1.9 Схематический обзор процесса, участвующего в разработке и выполнении конвейеров в виде ОАГ с использованием Airflow

Проницательный читатель, возможно, заметит, что фактически шаги, выполняемые планировщиком, очень похожи на алгоритм, приведенный в разделе 1.1. Это не случайно, поскольку Airflow, по сути, выполняет те же шаги, добавляя дополнительную логику для обработки своей логики планирования.

После того как задачи поставлены в очередь на выполнение, с ними уже работает пул воркеров Airflow, которые выполняют задачи параллельно и отслеживают их результаты. Эти результаты передаются в базу метаданных Airflow, чтобы пользователи могли отслеживать

ход выполнения задач и просматривать журналы с помощью веб-интерфейса Airflow (интерфейс, предоставляемый веб-сервером Airflow).

1.2.3 Мониторинг и обработка сбоев

Помимо планирования и выполнения ОАГ, Airflow также предоставляет обширный веб-интерфейс, который можно использовать для просмотра графов и мониторинга результатов их выполнения. После входа (рис. 1.10) на главной странице появляется обширный обзор различных ОАГ со сводными обзорами их последних результатов (рис. 1.11).

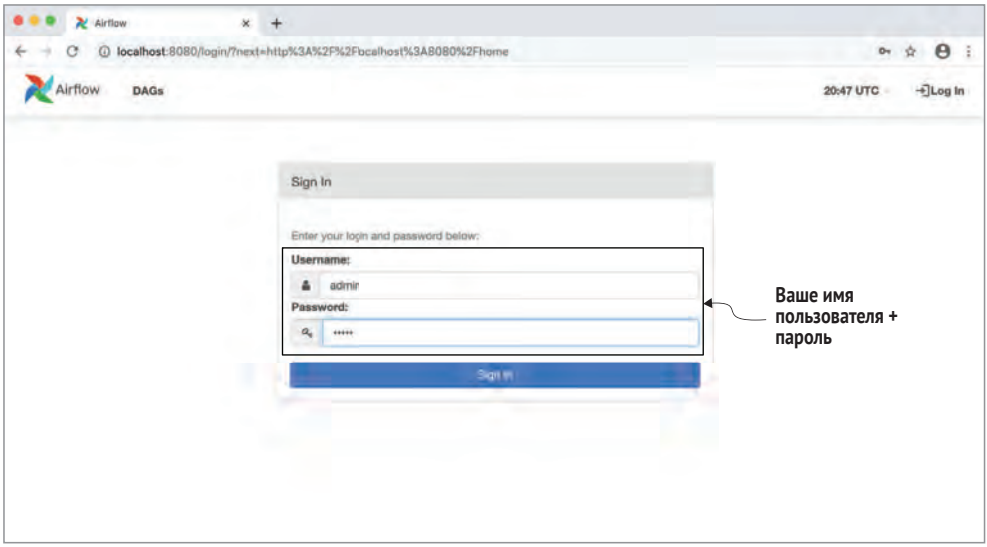

Рис. 1.10 Страница входа в веб-интерфейс Airflow. В примерах кода к этой книге пользователю по умолчанию «admin» предоставляется пароль «admin»

Например, графовое представление отдельного ОАГ дает четкий обзор задач и зависимостей графа (рис 1.12), аналогично схематическим обзорам, которые вы видели в этой главе. Это представление особенно полезно для просмотра структуры графа (обеспечивая подробное понимание зависимостей между задачами), а также для просмотра результатов отдельных запусков ОАГ.

Помимо этого представления, Airflow также предоставляет подробное древовидное представление, в котором показаны все текущие и предыдущие запуски соответствующего ОАГ (рис. 1.13). Это, пожалуй, самое мощное представление, которое дает веб-интерфейс, поскольку здесь приводится беглый обзор того, как работал ОАГ, и оно позволяет покопаться в задачах, завершившихся сбоем, чтобы увидеть, что пошло не так.

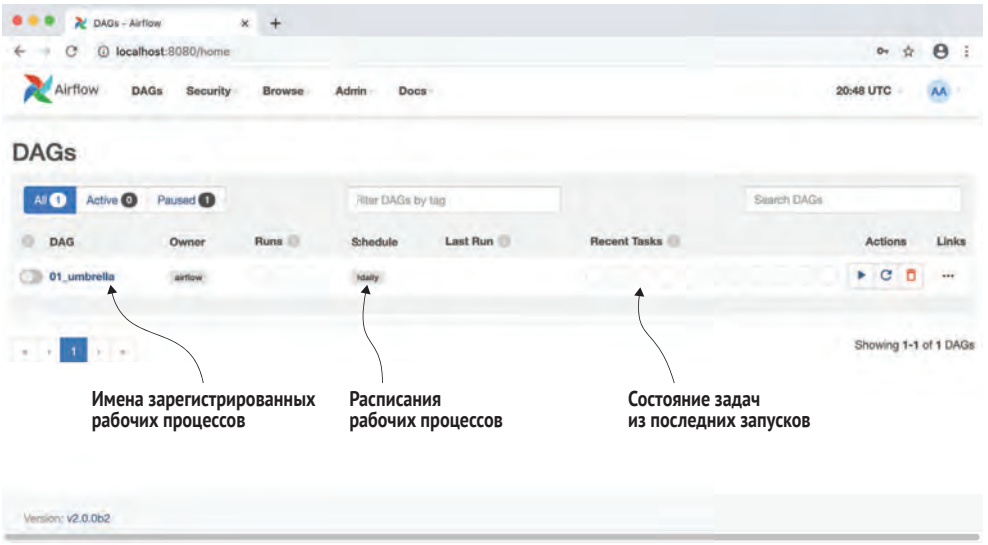

Рис. 1.11 Главная страница веб-интерфейса Airflow с обзором доступных OAG и их последние результаты

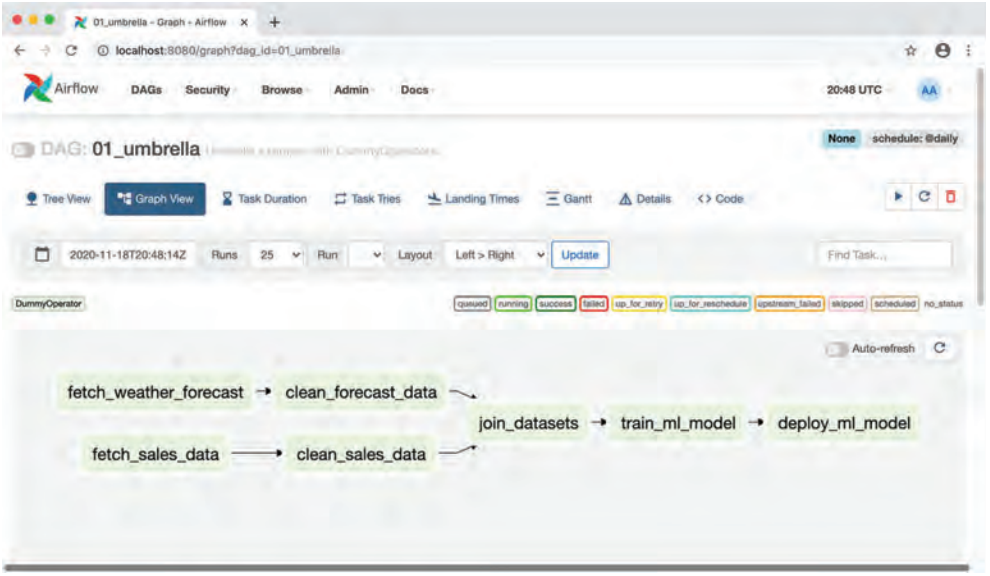

Рис. 1.12 Графовое представление в веб-интерфейсе Airflow, показывающее обзор задач в отдельном OAG и зависимости между этими задачами

По умолчанию Airflow может обрабатывать сбои в задачах, повторяя их несколько раз (иногда со временем ожидания между ними), что может помочь задачам восстановиться после периодических сбоев. Если это не помогает, Airflow запишет задачу как неудачную, при желании уведомив вас о сбое, если это предусмотрено настрой-

ками. Отладка сбоев задач довольно проста, поскольку представление в виде дерева позволяет увидеть, какие задачи не удалось выполнить, и изучить их журналы. Это же представление также позволяет очищать результаты отдельных задач для их повторного запуска (наряду со всеми задачами, которые зависят от этой задачи), что дает возможность с легкостью повторно запускать все задачи после внесения изменений в их код.

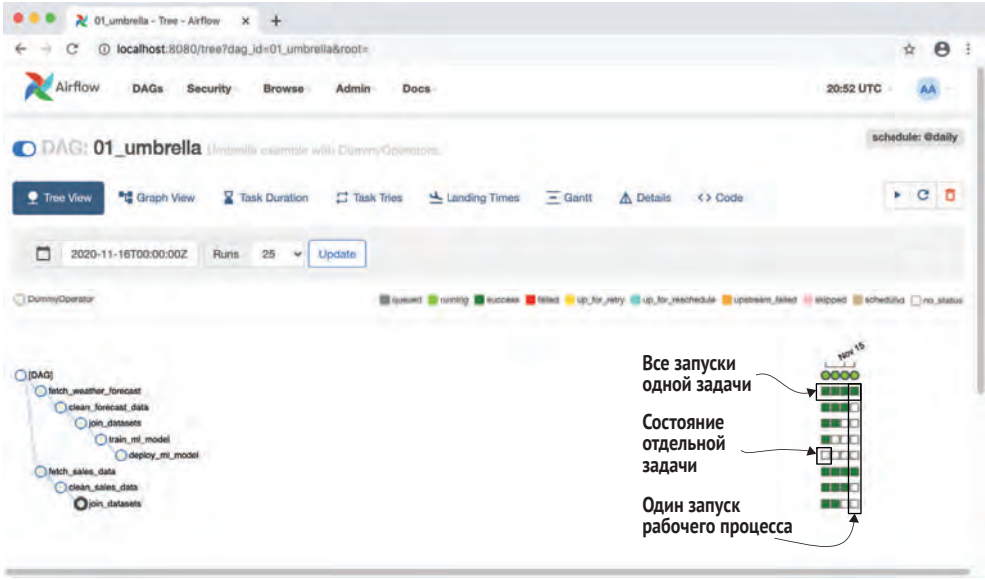

Рис. 1.13 Древовидное представление Airflow, показывающее результаты нескольких запусков ОАГ модели продаж зонтов (самые последние + предыдущие запуски). Столбцы показывают статус одного выполнения ОАГ, а строки показывают статус всех выполнений одной задачи. Цвета (которые вы видите в версии для электронной книги) обозначают результат выполнения соответствующей задачи. Пользователи также могут щелкнуть по «квадратикам», чтобы получить более подробную информацию о данном экземпляре задачи или сбросить состояние задачи, дабы при необходимости ее можно было повторно запустить с помощью Airflow

1.2.4 Инкрементальная загрузка и обратное заполнение

Одно из мощных функций семантики планирования Airflow состоит в том, что вышеуказанные интервалы не только запускают ОАГ в определенные моменты времени (аналогично, например, Cron), но также предоставляют подробную информацию о них и (ожидаемых) следующих интервалах. По сути, это позволяет разделить время на дискретные интервалы (например, каждый день, неделю и т. д.) и запускать ОАГ с учетом каждого из этих интервалов[1].

[1] Если сейчас это звучит для вас немного абстрактно, не волнуйтесь, мы подробно расскажем об этих концепциях позже.

Такое свойство интервалов Airflow неоценимо для реализации эффективных конвейеров обработки данных, поскольку позволяет создавать дополнительные конвейеры. В этих инкрементных конвейерах каждый запуск ОАГ обрабатывает только данные для соответствующего интервала времени (*дельта* данных), вместо того чтобы каждый раз повторно обрабатывать весь набор данных. Это может обеспечить значительную экономию времени и средств, особенно в случае с большими наборами данных, за счет предотвращения дорогостоящего пересчета существующих результатов.

Эти интервалы становятся еще более мощными в сочетании с концепцией обратного заполнения, позволяющей выполнять новый ОАГ для интервалов, которые имели место в прошлом. Эта функция позволяет легко создавать новые наборы архивных данных, просто запуская ОАГ с учетом этих интервалов. Более того, очистив результаты прошлых запусков, вы также можете использовать эту функцию Airflow, чтобы повторно запускать любые архивные задачи, если вы вносите изменения в код задачи, что при необходимости позволяет повторно обрабатывать весь набор данных.

1.3 Когда использовать Airflow

Мы надеемся, что после этого краткого введения в Airflow вы с энтузиазмом познакомились с Airflow и узнали больше о его ключевых функциях. Однако, прежде чем двигаться дальше, для начала рассмотрим причины, по которым вы, возможно, захотите работать с Airflow (а также ряд причин, по которым вы, вероятно, не захотите этого делать), чтобы убедиться, что Airflow – и в самом деле самый подходящий для вас вариант.

1.3.1 Причины выбрать Airflow

В предыдущих разделах мы уже описали несколько ключевых функций, которые делают Airflow идеальным вариантом для реализации конвейеров пакетной обработки данных. Они включают в себя:

- возможность реализовывать конвейеры с использованием кода на языке Python позволяет создавать сколь угодно сложные конвейеры, используя все, что только можно придумать в Python;
- язык Python, на котором написан Airflow, позволяет легко расширять и добавлять интеграции со многими различными системами. Сообщество Airflow уже разработало богатую коллекцию расширений, которые дают возможность Airflow интегрироваться в множество различных типов баз данных, облачных сервисов и т. д.;
- обширная семантика планирования позволяет запускать конвейеры через равные промежутки времени и создавать эффектив-

ные конвейеры, использующие инкрементную обработку, чтобы избежать дорогостоящего пересчета существующих результатов;
- такие функции, как обратное заполнение, дают возможность с легкостью (повторно) обрабатывать архивные данные, позволяя повторно вычислять любые производные наборы данных после внесения изменений в код;
- многофункциональный веб-интерфейс Airflow обеспечивает удобный просмотр результатов работы конвейера и отладки любых сбоев, которые могут произойти.

Дополнительное преимущество Airflow состоит в том, что это фреймворк с открытым исходным кодом. Это гарантирует, что вы можете использовать Airflow для своей работы без какой-либо привязки к поставщику. У некоторых компаний также есть управляемые решения (если вам нужна техническая поддержка), что дает больше гибкости относительно того, как вы запускаете и управляете своей установкой Airflow.

1.3.2 Причины не выбирать Airflow

Хотя у Airflow имеется множество мощных функций, в определенных случаях это, возможно, не то, что вам нужно. Вот некоторые примеры, когда Airflow – не самый подходящий вариант:

- обработка потоковых конвейеров, поскольку Airflow в первую очередь предназначен для выполнения повторяющихся или задач по пакетной обработке данных, а не потоковых рабочих нагрузок;
- реализация высокодинамичных конвейеров, в которых задачи добавляются или удаляются между каждым запуском конвейера. Хотя Airflow может реализовать такое динамическое поведение, веб-интерфейс будет показывать только те задачи, которые все еще определены в самой последней версии ОАГ. Таким образом, Airflow отдает предпочтение конвейерам, структура которых не меняется каждый раз при запуске;
- команды с небольшим опытом программирования (Python) или вообще не имеющие его, поскольку реализация ОАГ в Python может быть сложной задачей для тех, у кого малый опыт работы с Python. В таких командах использование диспетчера рабочих процессов с графическим интерфейсом (например, Azure Data Factory) или определение статического рабочего процесса, возможно, имеет больше смысла;
- точно так же код Python в ОАГ может быстро стать сложным в более крупных примерах. Таким образом, внедрение и поддержка ОАГ в Airflow требуют должной строгости, чтобы поддерживать возможность сопровождения в долгосрочной перспективе.

Кроме того, Airflow – это в первую очередь платформа для управления рабочими процессами и конвейерами, и (в настоящее время)

она не включает в себя более обширные функции, такие как линия данных, управление версиями данных и т. д. Если вам потребуются эти функции, то вам, вероятно, придется рассмотреть возможность объединения Airflow с другими специализированными инструментами, которые предоставляют эти функции.

1.4 Остальная часть книги

К настоящему времени вы должны (мы надеемся) иметь четкое представление о том, что такое Airflow и как его функции могут помочь вам реализовать и запускать конвейеры обработки данных. В оставшейся части этой книги мы начнем с представления основных компонентов Airflow, с которыми вам необходимо ознакомиться, чтобы приступить к созданию собственных конвейеров. Эти первые несколько глав должны иметь большое применение и апеллируют к широкой аудитории. Здесь мы ожидаем, что у вас уже имеется опыт программирования на языке Python (около года). Это означает, что вы должны быть знакомы с такими основными понятиями, как форматирование строк, списковые включения, параметры args и kwargs и т. д. Вы также должны быть знакомы с основами командной строки Linux и иметь хотя бы небольшой опыт использования баз данных (включая SQL) и различных форматов данных.

После этого введения мы углубимся в более сложные функции Airflow, такие как создание динамических ОАГ, реализация собственных операторов, выполнение контейнерных задач и т. д. Эти главы потребуют более глубокого понимания задействованных технологий, включая написание собственных классов Python, основных концепций Docker, форматов файлов и разделения данных. Мы ожидаем, что вторая часть будет особенно интересна data-инженерам.

Наконец, несколько глав в конце книги посвящены темам, связанным с развертыванием Airflow, включая шаблоны развертывания, мониторинг, безопасность и облачные архитектуры. Мы ожидаем, что эти главы будут интересны тем, кто занимается реализацией и управлением развертываний Airflow, например системным администраторам и инженерам DevOps.

Резюме

- Конвейеры обработки данных могут быть представлены в виде ОАГ, которые четко определяют задачи и их зависимости. Эти графы можно выполнять, используя преимущества параллелизма, присущего структуре зависимостей.
- Несмотря на то что на протяжении многих лет для выполнения графов задач было разработано множество диспетчеров рабочих

процессов, Airflow имеет несколько ключевых функций, которые делают его уникальным для реализации эффективных конвейеров пакетной обработки данных.

- Airflow состоит из трех основных компонентов: веб-сервера, планировщика и воркеров, которые работают сообща для планирования задач из конвейеров обработки данных и помогают отслеживать их результаты.