Содержание

От издательства

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. Издательство «ДМК Пресс» очень серьезно относится 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.

Глава 1

Облачные микросервисы – почему они появились

1.1. Общие принципы и закономерности

Отрасль разработки программного обеспечения постоянно развивается и во многом живет по своим законам. В соответствии с этими законами каждое приложение, решающее реальную задачу, постоянно претерпевает изменения, его сложность непрерывно растет, а качество норовит стать хуже.

Еще в 1974 г. Леман сформулировал законы эволюции программного обеспечения[1]. Эти законы описывают баланс между противоположными силами, одни из которых движут разработки вперед, а другие замедляют прогресс. Впоследствии эти законы неоднократно пересматривали и дополняли.

Основные законы эволюции программного обеспечения Лемана звучат так:

- *закон постоянных изменений*: удовлетворенность программным обеспечением со временем снижается, если его не адаптируют для удовлетворения новых потребностей;
- *закон возрастающей сложности*: поскольку большой программный продукт постоянно изменяется, он становится все сложнее, если не предпринимать специальных действий по сохранению или снижению сложности;
- *закон снижения качества*: качество системы, как правило, снижается, если она не адаптируется к изменениям операционной среды.

Поскольку программное обеспечение по своей природе подвержено изменениям, Леман стремился определить законы, которые управляют такими изменениями, и найти способы сохранения жизнеспособности программных систем.

---

[1] https://ru.wikipedia.org/wiki/Законы_Лемана.

Программные системы постоянно развиваются, но три основных закона остаются актуальными. Поэтому разработчики ищут новые подходы к управлению развитием своих систем, придерживаясь этих принципов.

К наиболее распространенным подходам относятся *методология двенадцати факторов*, микросервисы и облачные вычисления.

1.2. Методология двенадцати факторов

Эта методология представляет собой набор лучших практик для создания веб-приложений или программного обеспечения как услуги. Она была разработана командой провайдера облачной платформы Heroku[1] и получила широкое распространение в сообществе разработчиков. Как можно понять из ее названия, эта методология выделяет двенадцать факторов разработки успешного и долгоживущего приложения.

1.2.1. Фактор I. Кодовая база

В соответствии с методологией приложение должно иметь только одну кодовую базу, которая отслеживается в системе контроля версий, такой как Git или Mercurial. Кодовая база должна представлять собой единственный репозиторий или набор репозиториев, которые совместно используют корневой коммит. Наличие нескольких кодовых баз указывает на *распределенную систему*, в которой каждый компонент следует рассматривать как *отдельное приложение*. Совместное использование кода несколькими приложениями нарушает принцип двенадцати факторов. Общий код следует выделить в библиотеки и затем включать их в приложение через менеджер зависимостей.

Приложение может быть представлено различными развертываниями, включая производственные, промежуточные и локальные среды разработки. Хотя в каждом развертывании могут быть активны разные версии, кодовая база остается одинаковой во всех развертываниях. Разработчики всегда должны иметь копию единой кодовой базы в своей локальной среде разработки.

Подробнее об этом принципе можно прочитать по ссылке[2].

1.2.2. Фактор II. Зависимости

Второй фактор подчеркивает необходимость явного объявления и изоляции зависимостей.

---

[1] https://www.heroku.com/.

[2] https://12factor.net/ru/codebase.

В соответствии с методологией следует объявлять все зависимости через *манифест объявления зависимостей* (dependency declaration manifest) и использовать инструмент изоляции зависимостей во время выполнения, чтобы никакие неявные зависимости не «просочились» из окружающей системы.

Этот принцип применяется как к производственным, так и к локальным средам разработки.

Объявление и изоляция зависимостей всегда должны применяться вместе. Явное объявление зависимостей упрощает настройку среды для новых разработчиков и гарантирует, что приложение будет одинаково корректно работать в разных средах.

Методология двенадцати факторов также не полагается на неявное существование каких-либо системных инструментов и требует явно поставлять любые необходимые системные инструменты в приложение для обеспечения совместимости.

Подробнее об этом принципе можно прочитать по ссылке[1].

1.2.3. Фактор III. Конфигурация

В соответствии с третьим фактором конфигурацию следует хранить в среде, а не в коде.

Конфигурация приложения должна храниться в переменных среды, которые легко меняются между развертываниями, не зависят от языка и ОС и вряд ли будут случайно добавлены в репозиторий кода.

Конфигурации не должны храниться как константы в коде, поскольку текущая конфигурация может существенно зависеть от развертывания, а код – нет. Но при этом *внутреннюю* конфигурацию приложения, например способ подключения подов, лучше реализовать в коде.

Не рекомендуется группировать конфигурации в именованные среды. Переменные среды должны быть детализированными элементами управления, которые задаются независимо для каждого развертывания.

Эта модель плавно масштабируется по мере расширения приложения на большее количество развертываний в течение его жизненного цикла.

Подробнее об этом принципе можно прочитать по ссылке[2].

1.2.4. Фактор IV. Внешние сервисы

Этот фактор методологии выделяет необходимость обработки *внешних сервисов* (backing service) как подключаемых ресурсов, доступ к которым осуществляется через URL или другой локатор / учетные данные, хранящиеся в конфигурации.

---

[1] https://12factor.net/ru/dependencies.

[2] https://12factor.net/ru/config.

К внешним сервисам относятся любые сервисы, которые приложение использует по сети в рамках своей обычной работы, такие как хранилища данных (например, MySQL), системы обмена сообщениями/очередями (например, AWS SQS) и системы кеширования (например, Redis).

Код приложения обрабатывает локальные и внешние сервисы одинаково. При замене локального ресурса на внешние необходимо изменить только дескриптор ресурса в конфигурации. Каждый внешний сервис обрабатывается как подключаемый ресурс, который можно легко присоединить или отсоединить от развертываний без необходимости менять код приложения.

Источник изображения: https://12factor.net/

Подробнее об этом принципе можно прочитать по ссылке[1].

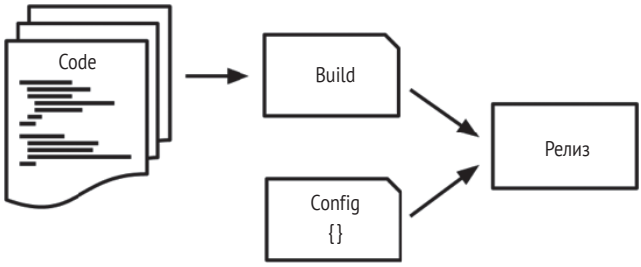
1.2.5. Фактор V. Сборка, релиз, выполнение

Пятый фактор указывает на важность строгого разделения этапов сборки, релиза и выполнения.

На этапе сборки репозиторий кода преобразуют в исполняемый пакет, который так и называется сборкой, на этапе выпуска объединяют сборку с текущей конфигурацией развертывания, а на последнем этапе запускают приложение в среде выполнения.

Код невозможно изменить во время выполнения, а каждый релиз должен иметь уникальный идентификатор, который нельзя менять после его создания. Создание сборки инициируется разработчиками приложения, в то время как выполнение может запускаться автоматически. Поэтому этап запуска должен быть максимально простым, чтобы избежать поломки приложения в отсутствие разработчиков.

[1] https://12factor.net/ru/backing-services.



Источник изображения: https://12factor.net/

Подробнее об этом принципе можно прочитать по ссылке[1].

1.2.6. Фактор VI. Процессы

Этот фактор определяет выполнение приложения как один или несколько процессов без сохранения состояния, запускаемых в среде выполнения.

Согласно этому принципу процессы не имеют состояния и ничем не делятся, то есть любые данные, которые необходимо сохранить, должны храниться во внешнем сервисе хранения состояний, например в хранилище данных.

Пространство памяти или файловая система процесса может играть роль краткосрочного кеша на одну транзакцию. Тем не менее приложение не должно предполагать, что какие-либо кешированные данные будут доступны в будущем. Использование «липких сессий» (sticky session), которые кешируют данные сеанса пользователя в памяти, нарушает методологию двенадцати факторов. Данные о состоянии сеанса должны храниться в хранилище данных, которое отслеживает истечение срока действия, например Memcached или Redis[2,3].

Подробнее об этом принципе можно прочитать по ссылке[4].

1.2.7. Фактор VII. Привязка портов

Следующий фактор – экспорт сервисов через *привязку портов* (port binding).

В отличие от традиционных веб-приложений, которые полагаются на контейнер веб-сервера, приложение, соответствующее принципам двенадцати факторов, является самодостаточным и обходится без веб-сервера для создания веб-службы.

Приложение должно экспортировать HTTP как сервис, привязываясь к порту и прослушивая входящие запросы. Привязку портов также можно

---

1 https://12factor.net/ru/build-release-run.
2 https://memcached.org/.
3 https://redis.com/.
4 https://12factor.net/ru/processes.

использовать для экспорта других типов серверного программного обеспечения, позволяя одному приложению выступать в качестве внешнего сервиса для другого.

При развертывании обработка запросов маршрутизации от общедоступного имени хоста к веб-процессам, привязанным к порту, выполняется на *уровне маршрутизации* (routing layer).

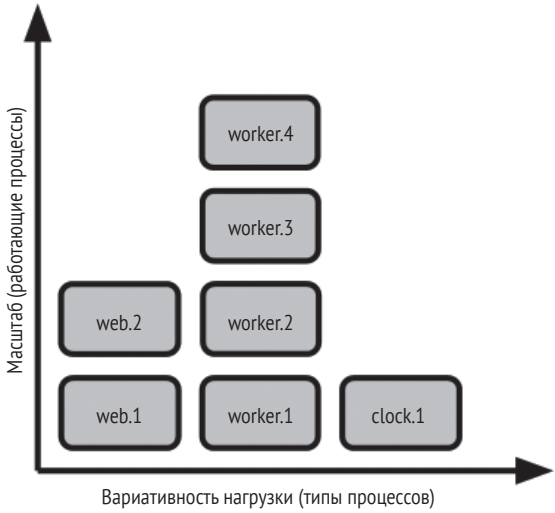
Подробнее об этом принципе можно прочитать по ссылке[1].

1.2.8. Фактор VIII. Параллелизм

В соответствии с принципом *параллелизма* процессы являются сущностями первого уровня, а их масштабирование выражается в количестве запущенных процессов.

Разработчики могут сопоставлять каждый тип задания с типом процесса. Расширение параллелизма становится простой операцией благодаря несовместному использованию и горизонтальной разделимости процессов.

Процессы никогда не должны выполняться как демоны или записывать файлы PID, а диспетчер процессов операционной системы должен управлять выходными потоками и обрабатывать перезапуски и завершения работы.



Источник изображения: https://12factor.net/

Подробнее об этом принципе можно прочитать по ссылке[2].

[1] https://12factor.net/ru/port-binding.
[2] https://12factor.net/ru/concurrency.

1.2.9. Фактор IX. Утилизируемость

Принцип *утилизируемости* (disposability) означает способность процессов быстро запускаться и останавливаться, что позволяет быстро развертывать изменения и эластично масштабировать систему без ущерба для надежности.

Процессы должны быть спроектированы так, чтобы они быстро запускались и корректно завершались при получении сигнала SIGTERM. Воркеры должны возвращать все текущие задания в очередь перед выходом, а процессы должны быть устойчивы к внезапным сбоям.

Кроме того, приложение должно уметь обрабатывать неожиданные, некорректные завершения.

Подробнее об этом принципе можно прочитать по ссылке[1].

1.2.10. Фактор X. Паритет разработки и производства

Принцип *паритета разработки и производства* (dev/prod parity) направлен на сокращение разрывов между средой разработки и производства, включая время, персонал и инструменты.

Принцип подразумевает развертывание кода в течение нескольких часов или даже минут, минимизацию разрыва между написанием и развертыванием кода, вовлечение одних и тех же людей в оба процесса и обеспечение максимальной схожести сред разработки и производства.

Также рекомендуется использовать одни и те же тип и версию внешних сервисов во всех развертываниях.

Легкие локальные сервисы не так привлекательны, как раньше, поскольку современные системы упаковки и декларативные инструменты предоставления позволяют разработчикам запускать приложения в локальных средах, очень похожих на производственные.

Подробнее об этом принципе можно прочитать по ссылке[2].

1.2.11. Фактор XI. Журналирование

В соответствии с принципом *журналирования* (logging) приложение обрабатывает журналы как *потоки событий*[3], которые упорядочены по времени и обеспечивают видимость поведения работающего приложения.

---

1 https://12factor.net/ru/disposability.

2 https://12factor.net/ru/dev-prod-parity.

3 https://adam.herokuapp.com/past/2011/4/1/logs_are_streams_not_files/.

Приложение, разработанное по методологии двенадцати факторов, записывает свой поток событий без буферизации в STDOUT[1] и никогда не должно беспокоиться о маршрутизации или хранении своего выходного потока.

Среда выполнения захватывает поток каждого процесса, сопоставляет его со всеми другими потоками из приложения и направляет его в одно или несколько конечных мест назначения для просмотра и долгосрочного архивирования.

Для этой цели можно применять маршрутизаторы журналов с открытым исходным кодом, такие как Logplex[2], Fluentd[3] и др.

При желании поток событий для приложения можно направить в файл или в систему индексации, анализа и визуализации журналов, такую как ELK[4]. Это позволяет разработчику самостоятельно выполнять мощный и гибкий анализ поведения приложения на всем протяжении его работы, включая возможность находить определенные события в прошлом, отображать тенденции в больших масштабах и настраивать активные оповещения.

Подробнее об этом принципе можно прочитать по ссылке[5].

1.2.12. Фактор XII. Административные процессы

Двенадцатый, и последний, фактор фокусируется на запуске административных или управленческих задач как однократных процессов (one-off process). Типичным примером однократного административного процесса является перенос базы данных или сбор статических файлов.

Эти процессы должны выполняться в той же среде, что и обычные долгосрочные процессы приложения, с использованием той же кодовой базы и конфигурации.

Код, отвечающий за административные процессы, должен поставляться с кодом приложения, чтобы избежать проблем с синхронизацией.

Однократные административные процессы должны выполняться с теми же методами изоляции зависимостей, что и обычные процессы.

Методология двенадцати факторов рекомендует использовать языки, которые предоставляют оболочку REPL[6] в пакете поставки и упрощают запуск одноразовых скриптов. В производстве разработчики могут использовать для запуска одноразовых административных процессов SSH или другие механизмы удаленного выполнения команд, предоставляемые средой развертывания.

Подробнее об этом принципе можно прочитать по ссылке[7].

---

1 https://en.wikipedia.org/wiki/Standard_streams.
2 https://devcenter.heroku.com/articles/logplex.
3 https://www.fluentd.org/.
4 https://www.elastic.co/what-is/elk-stack.
5 https://12factor.net/ru/logs.
6 http://en.wikipedia.org/wiki/Read-eval-print_loop.
7 https://12factor.net/admin-processes.