

Содержание

От издательства	12
Глава 1. Облачные микросервисы – почему они появились	13
1.1. Общие принципы и закономерности	13
1.2. Методология двенадцати факторов	14
1.2.1. Фактор I. Кодовая база	14
1.2.2. Фактор II. Зависимости	14
1.2.3. Фактор III. Конфигурация	15
1.2.4. Фактор IV. Внешние сервисы	15
1.2.5. Фактор V. Сборка, релиз, выполнение	16
1.2.6. Фактор VI. Процессы	17
1.2.7. Фактор VII. Привязка портов	17
1.2.8. Фактор VIII. Параллелизм	18
1.2.9. Фактор IX. Утилизируемость	19
1.2.10. Фактор X. Паритет разработки и производства	19
1.2.11. Фактор XI. Журналирование	19
1.2.12. Фактор XII. Административные процессы	20
1.3. Микросервисы	21
1.3.1. Отдельная база данных для каждого сервиса	22
1.3.2. API Composition	22
1.3.3. Отдельный экземпляр сервиса на каждый контейнер	22
1.3.4. Внешняя конфигурация	23
1.3.5. Обнаружение сервисов на стороне сервера	23
1.3.6. Автоматический предохранитель	23
1.3.7. Облачный подход	24
1.4. От монолита к облачным микросервисам	24
Глава 2. Технические требования	26
2.1. Сервер разработки	26
2.2. Установка kubectl	28
Глава 3. Кластеры Kubernetes	29
3.1. Создание кластера Kubernetes для разработки с помощью Minikube	29
3.1.1. Установка Minikube	30
3.1.2. Создание кластера Minikube	31
3.1.3. Профили Minikube	32
3.1.4. Использование панели управления K8s с Minikube	33
3.1.5. Создание развертывания	34

3.1.6. События Kubernetes	34
3.1.7. Раскрытие развертывания	34
3.1.8. Удаление ресурсов K8s	35
3.1.9. Дополнения Minikube	35
3.1.10. Использование kubectl без расширения minikube	36
3.1.11. Удаление кластеров	36
3.2. Создание кластера Kubernetes для разработки с помощью Rancher	37
3.2.1. Технические требования	37
3.2.2. Использование Terraform для запуска кластера	37
3.2.3. Создание ресурсов Kubernetes с помощью пользовательского интерфейса Rancher	38
3.3. Создание локального кластера Kubernetes с помощью Rancher	40
3.3.1. Начальные требования	40
3.3.2. Создание кластера с помощью сервера Rancher	40
3.3.3. Примечание о высокой доступности	42
3.4. Создание локального кластера Kubernetes: другие варианты	43
3.5. Управляемые кластеры	43
3.6. Создание управляемого кластера DOK с помощью Terraform	44

Глава 4. Обзор архитектуры Kubernetes

4.1. Введение	47
4.2. Плоскость управления	47
4.2.1. Система хранения etcd	47
4.2.2. Сервер API (kube-apiserver)	48
4.2.3. Менеджер контроллеров (kube-controller-manager)	48
4.2.4. Облачный менеджер контроллеров (cloud-controller-manager)	48
4.2.5. Планировщик (kube-scheduler)	48
4.3. Узлы	48
4.3.1. Kubelet	48
4.3.2. Среда выполнения контейнера	49
4.3.3. Kube-proxy	49
4.4. Пулы узлов	49
4.5. Обзор архитектуры	50

Глава 5. Микросервисы без сохранения и с сохранением состояния

5.1. Введение	51
5.2. Рабочие нагрузки без сохранения состояния	51
5.3. Рабочие нагрузки с сохранением состояния	52

Глава 6. Развертывание микросервисов без сохранения состояния

6.1. Технические требования	54
6.2. Создание пространства имен	57
6.3. Создание развертывания	58
6.4. Проверка подов и развертываний	60
6.5. Доступ к подам	61

6.6. Предоставление доступа к разворачиванию	62
6.6.1. Сервис ClusterIP	62
6.6.2. Сервис NodePort	64
6.6.3. Сервис LoadBalancer	66
6.6.4. Сервис Headless	68
6.6.5. Сервис Ingress	70

Глава 7. Микросервисы с сохранением состояния в Kubernetes..... 76

7.1. Технические требования	76
7.2. Создание пространства имен	77
7.3. Создание ConfigMap для базы данных PostgreSQL.....	77
7.3.1. Что такое ConfigMap?	77
7.3.2. ConfigMap для PostgreSQL	78
7.4. Постоянное хранение данных в PostgreSQL.....	79
7.4.1. Тома Kubernetes	79
7.4.2. Объект VolumeClaims	79
7.4.3. Объект StorageClass.....	79
7.4.4. Добавление хранилища в PostgreSQL	80
7.4.5. Создание разворачивания для PostgreSQL.....	81
7.4.6. Создание сервиса для PostgreSQL.....	83
7.4.7. Создание разворачивания для нашего приложения	83
7.4.8. Создание сервиса для нашего приложения	86
7.4.9. Создание внешнего сервиса для нашего приложения	86
7.4.10. Создание Ingress для нашего приложения	87
7.5. Просмотр журналов и проверка работоспособности.....	87
7.6. Заключение.....	89

Глава 8. Разворачивание микросервисов с сохранением состояния: StatefulSet..... 90

8.1. Что такое StatefulSet?	90
8.2. StatefulSet или разворачивание?	91
8.3. Создание StatefulSet	91
8.4. Создание сервиса для StatefulSet.....	92
8.5. Задачи после разворачивания	93
8.6. StatefulSet или разворачивание: постоянное хранилище	94
8.7. StatefulSet или разворачивание: связанный сервис	96

Глава 9. Паттерны микросервисов: внешние конфигурации 97

9.1. Хранение конфигураций в среде	97
9.2. Секреты и переменные среды Kubernetes: зачем они нужны?.....	98
9.3. Секреты и переменные среды Kubernetes: как это работает?.....	98

Глава 10. Важные приемы работы с микросервисами: проверки состояния..... 104

10.1. Проверки состояния приложения	104
---	-----

10.2. Зонды жизнеспособности и готовности	105
10.3. Типы зондов.....	106
10.4. Реализация зондов.....	106

Глава 11. Стратегии управления ресурсами микросервисов

11.1. Управление ресурсами и риски: от Docker до Kubernetes.....	110
11.2. Запросы и лимиты	111
11.3. Единицы измерения ресурсов ЦП.....	112
11.4. Единицы измерения ресурсов памяти	114
11.5. Принципы настройки запросов и ограничений ресурсов	116
11.6. Резервированные и выделяемые ресурсы узла.....	117
11.7. Классы качества обслуживания (QoS)	118
11.7.1. Guaranteed	119
11.7.2. Burstable.....	119
11.7.3. BestEffort.....	121
11.7.4. Проверка класса пода по QoS	121
11.7.5. Порядок вытеснения	122
11.7.6. Пользовательский класс PriorityClass	122

Глава 12. Автоматическое масштабирование микросервисов в Kubernetes: введение

12.1. Лучшие методы масштабирования микросервисов	125
12.1.1. Использование реестра для обнаружения сервисов.....	125
12.1.2. Реализация проверок работоспособности.....	126
12.1.3. Проектирование с учетом масштабируемости и другие передовые практики	126

Глава 13. Горизонтальное автомасштабирование микросервисов в Kubernetes

13.1. Что такое горизонтальное масштабирование	128
13.2. Горизонтальное автомасштабирование подов.....	132
13.3. Автоматическое масштабирование на основе пользовательских метрик Kubernetes.....	136
13.4. Специализированные пользовательские метрики Kubernetes.....	137
13.5. Использование нескольких метрик	139
13.6. Автоматическое масштабирование на основе пользовательских метрик, отличных от Kubernetes.....	140
13.7. Автомасштабирование кластера.....	141

Глава 14. Вертикальное автомасштабирование микросервисов в Kubernetes

14.1. Вертикальное масштабирование – в чем отличие.....	144
14.2. Вертикальное автомасштабирование пода.....	146
14.3. Режимы VPA	151
14.3.1. Auto (автоматический).....	151

14.3.2. Initial (начальный)	151
14.3.3. Recreate (повторное создание)	152
14.3.4. Off (выключен)	152
14.4. Рекомендации VPA.....	152
14.4.1. Ограничения VPA.....	154

Глава 15. Масштабирование микросервисов, сохраняющих состояние на примере PostgreSQL..... 156

15.1. StatefulSet и масштабирование.....	156
15.2. Stolon: знакомство	158
15.3. Stolon: установка	160
15.4. Stolon: применение	161

Глава 16. Стратегии развертывания микросервисов: один сервис на узел..... 165

16.1. Назначение и варианты использования DaemonSet	165
16.2. Создание и управление DaemonSet.....	166

Глава 17. Стратегии развертывания микросервисов: назначение рабочих нагрузок определенным узлам 169

17.1. Зачем назначать рабочие нагрузки определенным узлам?.....	169
17.2. Ограничения и допуски.....	171
17.2.1. Определение	171
17.2.2. Пример ограничений и допусков	172
17.3. nodeSelector	175
17.3.1. Простейшая форма привязки к узлам	175
17.3.2. Пример привязки через nodeSelector	175
17.4. Сходство и антисходство узлов	176
17.4.1. Сходство узлов: как nodeSelector, но с большим количеством опций	176
17.4.2. Сходство узлов: пример	177
17.4.3. Антисходство узлов: пример	179
17.4.4. Вес сходства.....	181
17.4.5. Типы сходства и антисходства	182

Глава 18. Kubernetes: управление обновлениями инфраструктуры и режим обслуживания..... 184

18.1. Зачем нам нужно обновлять инфраструктуру?	184
18.2. Что нужно обновлять?	184
18.3. Обновление рабочих узлов.....	185
18.4. Обновление рабочих узлов: отмена кордона	186
18.5. Обновление пулов узлов	187
18.6. Обновления с нулевым временем простоя: бюджеты сбоев пода	187

Глава 19. Стратегии развертывания микросервисов:

управление обновлениями и развертывание приложений	190
19.1. Изначально облачная система: концепция	190
19.2. Стратегии развертывания	191
19.2.1. Сине-зеленое развертывание: введение.....	191
19.2.2. Канареечное развертывание: введение	193
19.2.3. Канареечное развертывание: пример использования Istio.....	194
19.2.4. Канареечное развертывание: тестирование в рабочей среде	202
19.3. Скользящее обновление.....	204
19.3.1. Скользящие обновления: пример	205

Глава 20. Наблюдаемость микросервисов в мире Kubernetes 211

20.1. Введение	211
20.2. Что такое мониторинг?	211
20.3. Что такое наблюдаемость?.....	212
20.4. Мониторинг: белый ящик или черный?.....	212
20.5. Основные компоненты наблюдаемости.....	213
20.5.1. Журналы.....	213
20.5.2. Метрики	213
20.5.3. Трассировка	214
20.5.4. Компоненты наблюдаемости на практике	214
20.6. Четыре золотых сигнала мониторинга.....	215
20.6.1. Задержка	215
20.6.2. Трафик.....	215
20.6.3. Ошибки.....	215
20.6.4. Насыщенность.....	216
20.7. Мониторинг и наблюдаемость: в чем разница?	216

Глава 21. Средства наблюдения: Prometheus, Grafana, Loki, Promtail, OpenTelemetry и Jaeger 217

21.1. Что такое Prometheus?.....	217
21.2. Как работает Prometheus.....	217
21.3. Установка Prometheus.....	219
21.4. Доступ к веб-интерфейсу Prometheus.....	220
21.5. Метрики, доступные в Prometheus	220
21.6. Использование Grafana для визуализации метрик Prometheus	221
21.7. Извлечение информации из логов Kubernetes с помощью Promtail.....	222
21.8. стек журналирования Loki	223
21.9. Использование Loki для запроса и просмотра журналов	225
21.10. Использование Jaeger и OpenTelemetry для распределенной трассировки	228

Глава 22. GitOps: непрерывная поставка в облачном стиле..... 235

22.1. GitOps: введение и определения	235
22.2. Преимущества и недостатки GitOps.....	235
22.3. Инструменты GitOps.....	236

Глава 23. Пример рабочего процесса GitOps с использованием Argo CD.....238

23.1. Знакомство с Argo CD	238
23.2. Argo CD: установка и настройка	238
23.3. Argo CD: создание приложения	239
23.4. Argo CD: автоматическая синхронизация и самовосстановление	242
23.5. Откат изменений с помощью Argo CD	243
23.6. Декларативный способ использования Argo CD	243
23.7. Управление конфигурацией с помощью Argo CD	244
23.8. Argo CD: управление различными средами.....	250
23.9. Argo CD: хуки развертывания	252

Глава 24. Создание конвейеров CI/CD для микросервисов255

24.1. Непрерывная интеграция, поставка и развертывание микросервисов	255
24.2. Инструменты CI/CD	256
24.2.1. Jenkins	256
24.2.2. Spinnaker	256
24.2.3. Argo CD	257
24.2.4. GitHub Actions.....	257
24.2.5. GitLab CI/CD	257
24.3. Создание конвейера CI/CD для микросервиса	258
24.3.1. Установка и настройка Argo CD	258
24.3.2. Создание репозитория GitHub для нашего микросервиса.....	260
24.3.3. Создание учетной записи Docker Hub	263
24.3.4. Настройка GitHub Actions	264
24.3.5. Создание чарта Helm.....	266
24.3.6. Создание приложения Argo CD.....	269
24.3.7. Автоматизация развертывания новых версий.....	271

Глава 25. Послесловие.....273

25.1. Что дальше?.....	273
25.2. Благодарность.....	273
25.3. Об авторе	273
25.4. Присоединяйтесь к сообществу	274
25.5. Обратная связь	274

Предметный указатель.....275

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 обеспечения¹. 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 Heroku¹ 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 репозиторий, 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 ссылке².

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 ссылке¹.

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 ссылке².

1.2.4. Фактор IV. Внешние сервисы

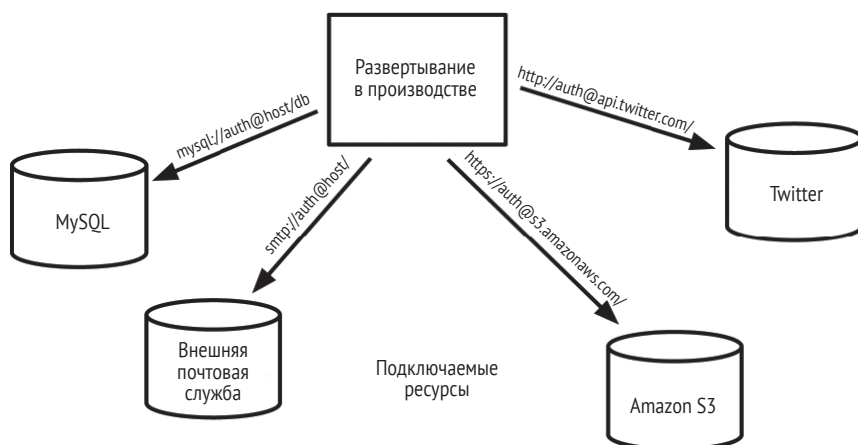
Этот фактор методологии выделяет необходимость обработки *внешних сервисов* (backing service) как подключаемых ресурсов, доступ к которым осуществляется через URL или другой локатор / учетные данные, хранящиеся в конфигурации.

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 ссылке¹.

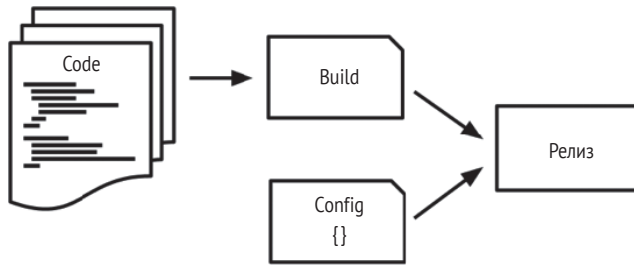
1.2.5. Фактор V. Сборка, релиз, выполнение

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 ссылке¹.

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 ссылке⁴.

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).

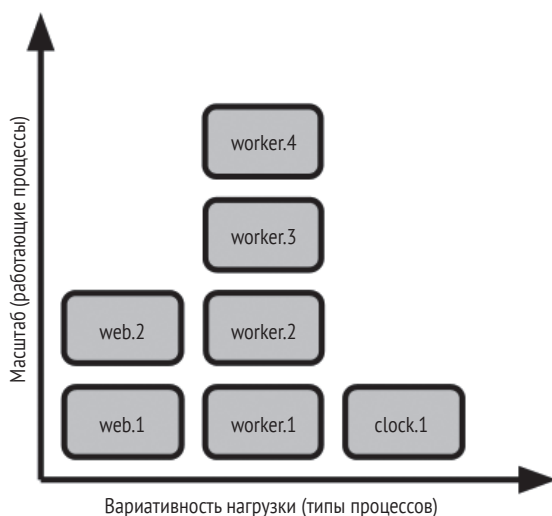
Подробнее об этом принципе можно прочитать по ссылке¹.

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 несоместному 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 ссылке².

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 ссылке¹.

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 ссылке².

1.2.11. Фактор XI. Журналирование

В соответствии с принципом *журналирования* (logging) приложение обрабатывает журналы как *поток 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 ссылке⁵.

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 ссылке⁷.

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>.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru