

---

# Оглавление

<b>Предисловие .....</b>	<b>14</b>
<b>Вступление .....</b>	<b>16</b>
Kubernetes .....	16
Паттерны проектирования.....	17
Структура книги .....	19
Кому адресована эта книга.....	21
Что вы узнаете .....	21
Типографские соглашения.....	22
Использование примеров кода .....	23
Благодарности.....	24
От издательства .....	25
<b>Глава 1. Введение .....</b>	<b>26</b>
Путь в облачное окружение.....	26
Распределенные примитивы .....	29
Контейнеры.....	31
Поды .....	33
Службы .....	35
Метки .....	36
Аннотации.....	38
Пространства имен.....	38
Пояснение .....	40
Дополнительная информация.....	41

## ЧАСТЬ I. ОСНОВНЫЕ ПАТТЕРНЫ

<b>Глава 2.</b> Предсказуемые требования .....	44
Задача .....	44
Решение .....	45
Зависимости времени выполнения .....	46
Профили ресурсов.....	48
Приоритет пода.....	50
Ресурсы проекта.....	53
Планирование вычислительных мощностей.....	54
Пояснение .....	56
Дополнительная информация.....	56
<b>Глава 3.</b> Декларативное развертывание .....	58
Задача .....	58
Решение .....	60
Непрерывное развертывание .....	60
Фиксированное развертывание .....	63
Пояснение .....	66
Дополнительная информация.....	68
<b>Глава 4.</b> Проверка работоспособности.....	69
Задача .....	69
Решение .....	70
Проверка наличия процесса .....	70
Проверка работоспособности .....	70
Проверка готовности.....	72
Пояснение .....	74
Дополнительная информация.....	75
<b>Глава 5.</b> Управляемый жизненный цикл .....	76
Задача .....	76
Решение .....	77
Сигнал SIGTERM.....	78
Сигнал SIGKILL.....	78
Точка входа postStart .....	78

Точка входа preStop .....	80
Другие средства управления жизненным циклом .....	81
Пояснение .....	83
Дополнительная информация.....	83
<b>Глава 6. Автоматическое размещение .....</b>	<b>84</b>
Задача.....	84
Решение .....	85
Доступные ресурсы на узле.....	85
Потребности контейнеров в ресурсах.....	86
Политики размещения.....	86
Процесс планирования.....	88
Близость узлов.....	89
Близость и удаленность подов .....	91
Непригодность и допустимость.....	93
Пояснение .....	98
Дополнительная информация.....	100

## **ЧАСТЬ II. ПОВЕДЕНЧЕСКИЕ ПАТТЕРНЫ**

<b>Глава 7. Пакетное задание .....</b>	<b>103</b>
Задача.....	103
Решение .....	104
Пояснение .....	108
Дополнительная информация.....	109
<b>Глава 8. Периодическое задание.....</b>	<b>110</b>
Задача.....	110
Решение .....	111
Пояснение .....	113
Дополнительная информация.....	114
<b>Глава 9. Фоновая служба .....</b>	<b>115</b>
Задача.....	115
Решение .....	116
Пояснение .....	119
Дополнительная информация.....	120

<b>Глава 10. Служба-одиночка .....</b>	<b>121</b>
Задача.....	121
Решение .....	122
Блокировка приложения извне.....	122
Блокировка приложения изнутри .....	125
Бюджет неработоспособности пода.....	128
Пояснение .....	130
Дополнительная информация.....	131
<b>Глава 11. Служба с состоянием.....</b>	<b>132</b>
Задача.....	132
Хранилище .....	133
Постоянные сетевые координаты .....	134
Идентичность.....	135
Упорядоченность.....	135
Другие требования.....	135
Решение .....	136
Хранилище .....	138
Постоянные сетевые координаты .....	139
Идентичность .....	141
Упорядоченность.....	141
Другие особенности .....	142
Пояснение .....	145
Дополнительная информация.....	146
<b>Глава 12. Обнаружение служб .....</b>	<b>147</b>
Задача.....	147
Решение .....	148
Обнаружение внутренних служб .....	150
Ручное обнаружение служб.....	154
Обнаружение служб в кластере извне.....	157
Обнаружение служб уровня приложения.....	162
Пояснение .....	165
Дополнительная информация.....	167

<b>Глава 13.</b> Самоанализ .....	168
Задача .....	168
Решение .....	169
Пояснение .....	173
Дополнительная информация .....	173

### **ЧАСТЬ III. СТРУКТУРНЫЕ ПАТТЕРНЫ**

<b>Глава 14.</b> Init-контейнеры .....	175
Задача .....	175
Решение .....	176
Пояснение .....	181
Дополнительная информация .....	182
<b>Глава 15.</b> Паттерн Sidecar .....	183
Задача .....	183
Решение .....	184
Пояснение .....	187
Дополнительная информация .....	188
<b>Глава 16.</b> Адаптер .....	189
Задача .....	189
Решение .....	189
Пояснение .....	193
Дополнительная информация .....	193
<b>Глава 17.</b> Посредник .....	194
Задача .....	194
Решение .....	195
Пояснение .....	197
Дополнительная информация .....	197

### **ЧАСТЬ IV. КОНФИГУРАЦИОННЫЕ ПАТТЕРНЫ**

<b>Глава 18.</b> Конфигурация в переменных окружения .....	200
Задача .....	200
Решение .....	200

Пояснение .....	204
Дополнительная информация.....	206
<b>Глава 19. Конфигурация в ресурсах .....</b>	<b>207</b>
Задача.....	207
Решение .....	208
Пояснение .....	213
Дополнительная информация.....	214
<b>Глава 20. Неизменяемая конфигурация .....</b>	<b>215</b>
Задача.....	215
Решение .....	216
Тома Docker .....	216
Init-контейнеры в Kubernetes.....	218
Паттерны OpenShift.....	221
Пояснение .....	223
Дополнительная информация.....	224
<b>Глава 21. Макет конфигурации .....</b>	<b>225</b>
Задача.....	225
Решение .....	226
Пояснение .....	232
Дополнительная информация.....	232

## ЧАСТЬ V. ДОПОЛНИТЕЛЬНЫЕ ПАТТЕРНЫ

<b>Глава 22. Контроллер .....</b>	<b>234</b>
Задача.....	234
Решение .....	235
Пояснение .....	249
Дополнительная информация.....	250
<b>Глава 23. Оператор.....</b>	<b>251</b>
Задача.....	251
Решение .....	252
Определения нестандартных ресурсов .....	252
Классификация контроллеров и операций.....	255

Разработка и развертывание оператора .....	259
Пример.....	262
Пояснение .....	267
Дополнительная информация.....	268
<b>Глава 24. Эластичное масштабирование .....</b>	<b>270</b>
Задача.....	270
Решение .....	271
Горизонтальное масштабирование вручную .....	271
Автоматическое горизонтальное масштабирование .....	273
Автоматическое вертикальное масштабирование .....	280
Автоматическое масштабирование кластера.....	285
Уровни масштабирования.....	289
Пояснение .....	293
Дополнительная информация.....	293
<b>Глава 25. Построитель образов.....</b>	<b>295</b>
Задача.....	295
Решение .....	297
Сборка в OpenShift .....	298
Knative Build .....	306
Пояснение .....	313
Дополнительная информация.....	314
<b>Послесловие.....</b>	<b>315</b>
Универсальная платформа .....	315
Многообразие обязанностей.....	315
О чем мы говорили.....	316
В заключение .....	317
<b>Об авторах.....</b>	<b>318</b>
<b>Об обложке .....</b>	<b>319</b>

# Эластичное масштабирование

Паттерн *Elastic Scale* (Эластичное масштабирование) охватывает масштабирование приложений в нескольких измерениях: горизонтальное масштабирование путем коррекции количества реплик пода; вертикальное масштабирование путем коррекции требований к ресурсам для подов; и масштабирование самого кластера путем изменения количества узлов. Все эти действия можно выполнить вручную, но в этой главе мы посмотрим, как то же самое может делать Kubernetes автоматически, в зависимости от нагрузки.

## Задача

Kubernetes автоматизирует управление распределенными приложениями, состоящими из большого количества неизменяемых контейнеров, поддерживая желаемое их состояние, выраженное декларативно. Однако, учитывая непостоянный характер многих рабочих нагрузок, которые часто меняются со временем, непросто понять, как должно выглядеть желаемое состояние. Точное определение, сколько ресурсов потребуется контейнеру и сколько реплик службы должно быть запущено в данный момент для выполнения соглашений об уровне обслуживания, требует времени и усилий. К счастью, Kubernetes позволяет легко изменять объем ресурсов контейнера, число реплик службы или узлов в кластере. Такие изменения могут происходить вручную или автоматически, в соответствии с определенными правилами.

Kubernetes может не только следовать фиксированным настройкам подов и кластера, но также следить за уровнем нагрузки и событиями, связан-



ными с изменением объемов ресурсов, анализировать текущее состояние и масштабироваться для достижения желаемой производительности. Эта способность наблюдать позволяет Kubernetes адаптироваться и обретать эластичность, опираясь на фактические показатели использования, а не на ожидаемые факторы. Давайте рассмотрим разные способы, которыми можно достичь такого поведения, и то, как объединить разные методы масштабирования.

## Решение

Есть два основных подхода к масштабированию любых приложений: горизонтальное и вертикальное. Под *горизонтальным масштабированием* в мире Kubernetes подразумевается создание большего количества реплик пода. Под *вертикальным масштабированием* подразумевается увеличение объема ресурсов для контейнеров, управляемых подами. На бумаге все выглядит просто, но определить конфигурацию приложения для автоматического масштабирования на общей облачной платформе так, чтобы не повлиять на другие службы и на сам кластер, часто можно только методом проб и ошибок. Как всегда, Kubernetes предлагает множество средств и методов, помогающих найти лучшие настройки для приложений, и мы кратко рассмотрим их здесь.

### Горизонтальное масштабирование вручную

Ручное масштабирование, как нетрудно догадаться, заключается в том, что оператор-человек посылает команды Kubernetes. Этот подход можно использовать в отсутствие автоматического масштабирования или для постепенного поиска оптимальной конфигурации приложения, соответствующей медленно меняющейся нагрузке, в течение длительных периодов. Преимущество ручного подхода заключается в том, что он допускает упреждающие, а не только реактивные изменения: зная периодичность и ожидаемую нагрузку на приложение, его можно масштабировать заранее, а не реагируя, например, на уже возросшую нагрузку с использованием средств автоматического масштабирования. Ручное масштабирование может выполняться в двух стилях.

## Императивное масштабирование

Контроллер, такой как ReplicaSet, отвечает за постоянное выполнение заданного количества экземпляров пода. Благодаря этому, чтобы масштабировать под, достаточно изменить количество желаемых реплик. Масштабировать наше развертывание Deployment с именем random-generator до четырех экземпляров можно одной командой, как показано в листинге 24.1.

**Листинг 24.1.** Изменение числа реплик развертывания Deployment из командной строки

```
kubectl scale random-generator --replicas=4
```

После такого изменения контроллер ReplicaSet может запустить дополнительные поды или, если число действующих подов больше желаемого, остановить избыточные.

## Декларативное масштабирование

Масштабирование с использованием команды `scale` выполняется тривиально просто и удобно для быстрого реагирования в чрезвычайных ситуациях, но этот подход не сохраняет настройки вне кластера. Обычно все приложения для Kubernetes хранят определения своих ресурсов в системе управления версиями, включая число реплик. Воссоздание ReplicaSet из исходного определения приведет к изменению числа реплик до прежнего уровня. Чтобы избежать такого отклонения конфигурации и обеспечить обратное распространение изменений, рекомендуется декларативно изменять желаемое количество реплик в ReplicaSet или некотором другом определении и применять изменения к Kubernetes, как показано в листинге 24.2.

**Листинг 24.2.** Использование развертывания Deployment для декларативной настройки числа реплик

```
kubectl apply -f random-generator-deployment.yaml
```

Мы можем масштабировать ресурсы, управляющие несколькими подами, такие как ReplicaSet, Deployment и StatefulSet. Обратите внимание на

асимметричное поведение при масштабировании StatefulSet с постоянным хранилищем. Как рассказывалось в главе 11 «Служба с состоянием», если в StatefulSet имеется элемент `.spec.volumeClaimTemplates`, он будет создавать PVC при масштабировании вверх, но не будет удалять при масштабировании вниз, чтобы предотвратить удаление хранилища.

Еще одним ресурсом Kubernetes, доступным для масштабирования, но следующим другим соглашениям об именовании, является ресурс задания Job, который был описан в главе 7 «Пакетное задание». Масштабирование задания с целью параллельного выполнения нескольких экземпляров одного и того же пода осуществляется изменением поля `.spec.parallelism`, а не `.spec.replicas`. Однако это дает тот же семантический эффект: увеличение пропускной способности путем запуска дополнительных обрабатывающих подов, которые действуют как одна логическая единица.



Для описания полей в этой книге используется формат определения путей JSON. Например, имя `.spec.replicas` соответствует полю `replicas` в разделе `spec` объявления ресурса.

Оба стиля масштабирования вручную (императивный и декларативный) предполагают, что человек будет наблюдать или предвидеть изменение нагрузки на приложение, принимать решение о направлении и величине масштабирования и применять это решение к кластеру. Однако эти стили не подходят для приложений с динамической нагрузкой, которая часто меняется и требует постоянной адаптации. Давайте посмотрим далее, как можно автоматизировать принятие решений о масштабировании.

## Автоматическое горизонтальное масштабирование

Многие рабочие нагрузки имеют динамическую природу, меняясь со временем и затрудняя определение фиксированных настроек масштабирования. Но облачные технологии, такие как Kubernetes, позволяют создавать приложения, легко адаптирующиеся к изменяющимся нагрузкам. Поддержка автоматического масштабирования в Kubernetes позволяет определять переменную емкость приложения вместо фиксированной,

которая обеспечивает достаточную пропускную способность для обслуживания другой нагрузки. Проще всего такое поведение реализовать с использованием HorizontalPodAutoscaler (HPA) для горизонтального масштабирования числа подов.

Определение HPA для развертывания Deployment random-generator можно создать с помощью команды в листинге 24.3. Чтобы это определение HPA имело какой-либо эффект, важно, чтобы при развертывании было объявлено ограничение `.spec.resources.requests` для процессора, как описано в главе 2 «Предсказуемые требования». Также важно, чтобы в кластере был включен сервер метрик, который собирает данные об использовании ресурсов.

#### Листинг 24.3. Создание определения HPA из командной строки

```
kubectl autoscale deployment random-generator --cpu-percent=50
--min=1 --max=5
```

Эта команда создаст определение HPA, показанное в листинге 24.4.

#### Листинг 24.4. Определение HPA

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: random-generator
spec:
  minReplicas: 1           ❶
  maxReplicas: 5          ❷
  scaleTargetRef:         ❸
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: random-generator
  metrics:
  - resource:
    name: cpu
    target:
      averageUtilization: 50    ❹
      type: Utilization
    type: Resource
```

- ❶ Минимальное число всегда выполняющихся подов.
- ❷ Максимальное число подов, которое может быть достигнуто при масштабировании вверх.
- ❸ Ссылка на объект, связанный с этим определением HPA.
- ❹ Уровень использования процессора в процентах от запрошенного в подах. Например, если в поде параметр `.spec.resources.requests.cpu` имеет значение `200m`, масштабирование вверх произойдет, когда средняя величина использования процессора превысит `100m` (= 50%).



В листинге 24.4 для настройки HPA используется версия API ресурса `v2beta2`. Эта версия находится в активной разработке и является расширением версии `v1`. Версия `v2` предлагает намного больше критериев, чем загрузка процессора: например, потребление памяти или нестандартные метрики для конкретного приложения. Командой `kubectl get hpa.v2beta2.autoscaling -o yaml` легко можно преобразовать ресурс HPA `v1`, созданный командой `kubectl autoscale`, в ресурс `v2`.

Следуя этому определению, контроллер HPA будет стремиться сохранить среднее потребление процессора на уровне 50% от значения, затребованного в `.spec.resources.requests`, изменяя количество запущенных экземпляров пода в диапазоне от одного до пяти. Такое определение HPA можно применить к любым ресурсам, которые поддерживают подресурс `scale`, такие как `Deployment`, `ReplicaSet` и `StatefulSet`, но вы должны учитывать побочные эффекты. Развертывания `Deployment` создают новые наборы реплик во время обновлений, но не копируют никаких определений HPA. Если применить определение HPA к `ReplicaSet`, который управляется развертыванием `Deployment`, оно не будет скопировано в новые `ReplicaSet` и просто потеряется. Лучше всего применять HPA к абстракции развертывания `Deployment` более высокого уровня, которая сохраняет и применяет HPA к новым версиям `ReplicaSet`.

Теперь посмотрим, как определение HPA может заменить оператора-человека и обеспечить автоматическое масштабирование. В общем случае контроллер HPA непрерывно выполняет следующие действия: