ГЛАВА 1

Как развертывать приложения

Из предисловия вы узнали, что за словом «DevOps» кроется множество концепций. Но почти всегда все начинается с вопроса: «Я написал программу. И что теперь?»

Вы и ваша команда потратили долгие месяцы на то, чтобы создать приложение. Вы выбрали язык программирования, разработали бэкенд, сделали дизайн, построили пользовательский интерфейс (user interface, UI), и вот наконец пришло время передать ваше приложение реальным пользователям. Как именно это сделать?

Здесь возникает множество вопросов, которые придется решить. Будете использовать AWS или Azure? (А может, Heroku или Vercel?) Нужен один сервер или несколько? (Или в бессерверном окружении?) Надо ли работать с Docker? (Или Kubernetes?) Понадобится VPC? (Или VPN?) Как получить имя домена? (И что насчет TLS-сертификата?) Как правильно настроить базу данных? (А как организовать ее резервное копирование?) Почему приложение вылетело? Почему ничего не работает? И почему все так сложно?

А теперь успокойтесь, глубоко вдохните. Если вы новичок в доставке ПО, например, всю свою карьеру были разработчиком или только начинаете работать в эксплуатации, все это может вызвать панику или загнать в ступор. Моя книга призвана вам помочь. Шаг за шагом я помогу вам найти ответы и на эти вопросы, и на многие другие, которые даже не приходили вам в голову.

Первый шаг — развернуть ваше приложение на сервере и заставить его работать самым простым способом, какой вы только знаете. В этой главе вы попробуете выполнить развертывание одного и того же приложения на своем компьютере, на Render (платформа как сервис) и на AWS (инфраструктура как сервис). После этого вы увидите, как можно развивать свои методы развертывания по мере роста вашей компании.

Без долгих церемоний сразу перейдем к делу и начнем развертывать приложения.

Пример: развертывание приложения локально

Первое, что вы должны уметь, — локально развертывать любое приложение на собственном компьютере. Обычно именно с этого все начинается: вы пишете и запускаете код локально до тех пор, пока он не начнет работать. Способ развертывания зависит от технологии. В книге мы будем использовать простой пример приложения Node.js (https://nodejs.org).

Пример кода

Напоминаю, что вы можете найти все примеры кода в репозитории книги на GitHub (https://github.com/brikis98/devops-book).

Создайте на своем компьютере новую папку. Назовите ее, например, `fundamentals-of-devops`. Сохраняйте в ней код примеров, которые будете выполнять при работе с книгой. Чтобы создать папку и перейти в нее, введите следующие команды в терминале:

```
$ mkdir fundamentals-of-devops
$ cd fundamentals-of-devops
```

В этой папке создайте подпапки для текущей главы и своего примера приложения:

```
$ mkdir -p ch1/sample-app
$ cd ch1/sample-app
```

Пример, который вы будете использовать, — это простейшее приложение Node.js «Hello, World», написанное на JavaScript. Вам не понадобится глубокое знание JavaScript, чтобы разобраться в приложении. Один из приятных моментов начала работы с Node.js заключается в том, что весь код для простого веб-приложения умещается в файле размером примерно десять строк. Внутри подпапки `sample-app` создайте файл `app.js`, содержимое которого показано в примере 1.1.

Пример 1.1. Пример приложения Node.js Hello, World (ch1/sample-app/app.js)

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, World!\n');          ❶
});

const port = process.env.PORT || 8080; ❷
server.listen(port,() => {
  console.log(`Listening on port ${port}`);
});
```

Приложение Hello, World делает следующее.

❶ Отправляет в ответ на все запросы код статуса `200` и текст `Hello, World!`.

❷ Слушает все запросы на порте с номером, заданным с помощью переменной окружения `PORT`, а если переменная `PORT` не установлена, то по умолчанию на порте 8080.

Чтобы запустить приложение, вам нужно сначала установить Node.js (https://nodejs.org/en/download) (минимальная версия 21). После этого вы можете запустить приложение командой `node app.js`:

```
$ node app.js
Listening on port 8080
```

Далее, открыв http://localhost:8080 в своем браузере, вы должны увидеть следующее:

```
Hello, World!
```

Поздравляю, вы запустили приложение локально! Это отличный старт, но если вы хотите показать свое приложение пользователям, вам понадобится запустить его на сервере. Как это сделать, мы обсудим далее.

Развертывание приложения на сервере

Когда вы запускаете приложение на своем компьютере, оно работает только на `localhost`. Это специальное сетевое имя, привязанное к `loopback`-*интерфейсу*. Это означает, что все сетевые запросы не отправляются в реальный сетевой интерфейс, а перенаправляются обратно на ваш компьютер. Таким образом, доступ к вашему приложению есть *только* с вашего же собственного компьютера, а пользователи извне подключиться не могут. Так сделано намеренно, и в большинстве случаев это хорошо, потому что запуск приложения для разработки и тестирования на своем компьютере — совсем не *то же самое*, что запуск с предоставлением доступа внешним пользователям.

Ключевой вывод 1

Никогда не открывайте доступ внешнему миру к приложениям, работающим на вашем персональном компьютере.

Если вы хотите открыть доступ к своему приложению внешним пользователям, лучше запустить его на сервере. *Сервер* — это компьютер, специально подготовленный для запуска приложений с предоставлением доступа к ним внешним пользователям. Сервер и персональный компьютер имеют множество различий, вот часть из них.

Безопасность

Большинство серверов работают на урезанной версии ОС и защищены от атак, например используют брандмауэр, средства предотвращения вторжений, мониторинг целостности файлов. На вашем персональном компьютере установлено множество дополнительных программ, многие из которых могут оказаться уязвимыми, и он не настолько хорошо защищен от атак.

Доступность

Большинство серверов рассчитаны на постоянную работу и имеют источники бесперебойного питания. А ваш персональный компьютер может отключиться в любой момент.

Производительность

На большинстве серверов работают только приложения, находящиеся на них. А свой персональный компьютер вы используете и для других задач, например написания кода и просмотра интернет-страниц. Это может влиять на производительность вашего приложения.

Совместная работа

Как правило, над созданием приложений работает команда, и тогда как другие разработчики не имеют (и не должны иметь) доступа к вашему персональному компьютеру, серверы обычно предназначены для группового доступа.

Поэтому всегда используйте сервер для запуска своих приложений в производственных целях. В широком смысле у вас есть два способа получить доступ к серверам (два варианта *хостинга* ваших приложений).

- Можно купить и сконфигурировать собственные серверы (локальный хостинг, on premises).
- Можно арендовать серверы у других (облачный хостинг, cloud).

Далее обсудим каждый из этих вариантов.

Локальный и облачный хостинг

Традиционным способом работы с программным обеспечением были и остаются приобретение и настройка серверов *на своей территории*, в помещении, которым вы владеете. Когда вы только начинаете, это может быть просто шкаф в вашем офисе. Но по мере роста компании растут и требования к вычислительной мощности, и в итоге вам понадобится *центр обработки данных* со всем необходимым оборудованием (стойки, серверы, жесткие диски, система охлаждения) и персоналом (электрики, сетевые администраторы, сотрудники безопасности). Таким образом, на протяжении десятилетий, если вы хотели

построить компанию по разработке программного обеспечения, то должны были инвестировать существенные суммы в оборудование.

Все начало меняться в 2006 году с запуском Amazon Web Services (AWS) — первой *платформы облачных вычислений* (для краткости — *облака*), которая позволила арендовать серверы, используя программный интерфейс, с помощью нескольких щелчков кнопкой мыши (что вы и будете делать в этой главе) или нескольких строк кода (это вы будете делать в главе 2). Этот кардинальный сдвиг позволяет вам сейчас приступить к работе за считаные минуты вместо месяцев, потратив всего несколько центов (или не потратив ничего) вместо тысяч долларов.

Существуют два основных облачных предложения: *инфраструктура как сервис* (infrastructure as a service, IaaS), которая предоставляет доступ к низкоуровневым вычислительным ресурсам (серверы, жесткие диски, сети) и позволяет вам самостоятельно собрать с их помощью процесс доставки ПО, и *платформа как сервис* (platform as a service, PaaS), которая дает доступ к примитивам более высокого уровня, включая процесс доставки ПО. Чтобы почувствовать разницу, вы сначала увидите, как работает PaaS (в следующем разделе), а затем — IaaS (в разделе следом за ним).

Пример: развертывание приложения с помощью PaaS (Render)

Наиболее популярные поставщики PaaS — Heroku, Render, Fly.io и Vercel (полный список представлен на веб-странице книги https://oreil.ly/ckLVg). Heroku был одним из первых поставщиков PaaS, и раньше я всегда выбирал его. Но он закрыл свою бесплатную версию в 2022 году. Поэтому для работы с примерами из книги вы будете использовать Render, который предлагает бесплатную версию Hobby (https://render.com/pricing), поддерживает приложения на многих языках и фреймворках, включая Node.js, и не требует настройки системы сборки или фреймворка (об этом вы узнаете позже). У Render хорошая репутация в сообществе, и его часто называют духовным преемником Heroku. Чтобы развернуть приложение с помощью Render, выполните следующие шаги.

Шаг 1. Зарегистрируйтесь в Render.

Создайте новый аккаунт на render.com.

Шаг 2. Разверните новый веб-сервис.

Зайдите на Render Dashboard (https://dashboard.render.com) и нажмите кнопку Deploy a Web Service (Развернуть веб-сервис). На следующей странице выберите вкладку Public Git Repository (Публичный репозиторий Git) и наберите URL репозитория этой книги, https://github.com/brikis98/devops-book (рис. 1.1). Этот репозиторий, находящийся в папке `ch1/sample-app`, содержит код Node.js из примера 1.1, что позволит вам развернуть приложение, не создавая собственный репозиторий GitHub.

Рис. 1.1. Использование репозитория с примером кода из книги для создания веб-сервиса в Render

Нажмите кнопку Connect (Подключить) и затем настройте веб-сервис в соответствии с параметрами, приведенными в табл. 1.1.

Таблица 1.1. Параметры конфигурации веб-сервиса в Render

Конфигурация	Значение параметра
Name (Имя)	`sample-app`
Language (Язык)	`Node`
Root Directory (Корневой каталог)	`ch1/sample-app`
Build Command (Команда сборки)	`# (no build command)`
Start Command (Команда запуска)	`node app.js`
Instance Type (Тип экземпляра)	`Free`

Оставьте значения остальных параметров (Project, Branch, Region) принятыми по умолчанию и нажмите кнопку Deploy Web Service (Развернуть веб-сервис) внизу страницы, чтобы начать процесс развертывания.

Шаг 3. Протестируйте свое приложение.

Как показано на рис. 1.2, через пару минут лог развертывания должен выдать что-то вроде `Your service is live` (Ваш сервис запущен).

В левом верхнем углу страницы вы должны увидеть сгенерированный случайным образом URL своего приложения в формате https://<имя>.onrender.com. Открыв его, вы должны увидеть:

```
Hello, World!
```

Поздравляю! Сделав всего несколько шагов, вы получили работающее на сервере приложение.

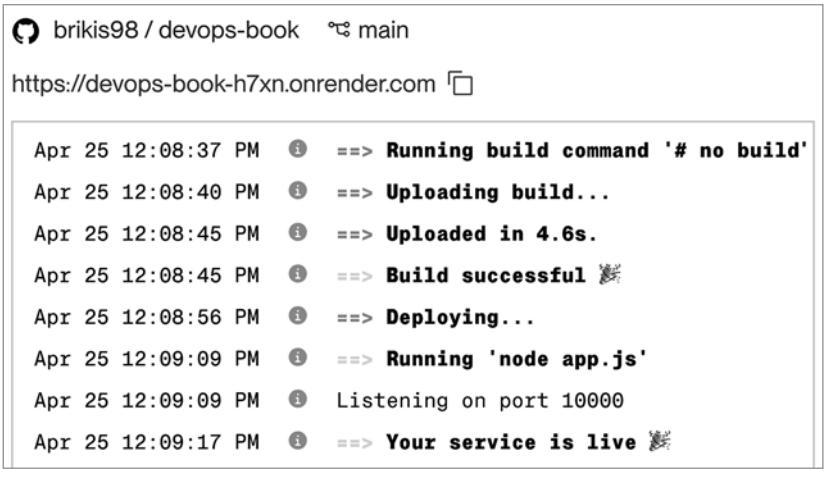

Рис. 1.2. Завершенное развертывание в Render

Засучите рукава

Вот несколько упражнений, которые вы можете выполнить дома, чтобы копнуть чуть глубже.

- Щелкните на вкладке Scale (Масштаб), чтобы изменить количество серверов, на которых запущено ваше приложение.
- Щелкните на вкладке Logs (Логи), чтобы просмотреть лог событий вашего приложения.
- Щелкните на вкладке Metrics (Метрики), чтобы просмотреть метрики событий вашего приложения.

Когда закончите экспериментировать с Render, отмените развертывание приложения. Для этого откройте вкладку Settings (Настройки), прокрутите вниз страницы и нажмите кнопку Delete Web Service (Удалить веб-сервис).

Использование PaaS, как правило, означает, что вы получаете не только сервер, но и множество мощных функций «из коробки»: масштабирование на несколько серверов, доменные имена (<имя>.onrender.com), шифрование (HTTPS URL), мониторинг (логи и метрики) и многое другое. В этом преимущество PaaS: хороший PaaS может взять на себя множество задач доставки ПО и сделать все за считаные минуты. Это похоже на магию. И в этом главная сила: PaaS просто работает.

Если работает. Когда же что-то не работает, эта же самая магия оказывается слабым местом PaaS. Так задумано, что в PaaS почти все происходит за кулисами, поэтому если что-то не работает, отладить или починить это довольно трудно. Более того, ради этой магии большинство предложений PaaS имеют ограничения на то, что можно развертывать, какие типы приложений можно запускать, какой доступ к базовому оборудованию возможен, какие виды оборудования

доступны и т. д. Если PaaS что-то не поддерживает, например, если в интерфейсе командной строки или пользовательском интерфейсе PaaS нет того, что вам нужно, скорее всего, вы вообще не сможете этого сделать.

В результате многие проекты стартуют на PaaS, но когда достаточно вырастают и начинают требовать большего контроля, то мигрируют на IaaS.

Пример: развертывание приложения с помощью IaaS (AWS)

В широком смысле все множество IaaS можно разделить на три сегмента.

Виртуальный частный сервер

Часть компаний сосредоточена на предоставлении доступа к виртуальному частному серверу (virtual private server, VPS) за минимальную цену, хотя может предлагать и другие функции, например сетевое взаимодействие и хранение данных. Основная причина выбора одного из таких поставщиков — возможность избавиться от необходимости иметь собственную стойку с серверами. Это ваш вариант, если предпочитаете, чтобы за вас это делал кто-то другой и просто предоставлял вам доступ. Крупные игроки в этом сегменте — Hetzner, DigitalOcean, Vultr и Akamai Connected Cloud (полный список представлен на https://oreil.ly/kw53X).

Сети доставки контента

Другие компании главным образом фокусируются на *сетях доставки контента* (content delivery networks, CDN), которые представляют собой серверы, распределенные по всему миру, для обслуживания и кэширования контента. Эти компании могут предлагать и ряд других функций, например защиту от атак. Основная причина выбора одного из таких поставщиков — наличие у вас географически распределенной пользовательской базы и необходимость в быстром и надежном способе предоставления контента с низкой задержкой. Вы узнаете все о CDN из главы 9.

Поставщики облачных услуг

И наконец, ряд крупных компаний пытаются предоставлять универсальные облачные решения, которые предлагают все: VPS, CDN, контейнеры, бессерверные вычисления, хранилище данных, хранение файлов, обработку естественного языка, периферийные вычисления и многое другое. Крупные игроки в этом сегменте — AWS, Google Cloud и Microsoft Azure (полный список здесь: https://oreil.ly/lcNwS).

В целом поставщики VPS и CDN — специалисты в своих областях, поэтому там они, как правило, превосходят поставщиков универсальных облачных услуг в плане функциональности, ценообразования и пользовательского опыта. Например, VPS компании Hetzner обычно быстрее, дешевле и в некотором смысле проще в применении, чем AWS. Поэтому, если вам нужны *только* эти

конкретные услуги, лучше обратиться к специалистам. Однако если вы строите инфраструктуру для целой компании, архитектура которой обычно включает множество видов инфраструктур, лучше подойдут поставщики универсальных облачных решений. Они предлагают комплексный подход, соответствующий всем вашим потребностям.

Для примеров, приведенных в книге, вы будете использовать в качестве IaaS-провайдера AWS, который предлагает бесплатный уровень (https://aws.amazon.com/free) с широким диапазоном надежных масштабируемых облачных сервисов (серверы, бессерверные вычисления, контейнеры, базы данных, распределители нагрузки и т. д.). Таким образом, вы сможете использовать его для большинства примеров. Вдобавок AWS является доминирующим поставщиком облачных услуг — у него 31 % рынка (https://oreil.ly/96Lg3), а последние 13 лет он также лидер магического квадранта Gartner (https://oreil.ly/ZdSNn). То есть именно его вы наверняка будете использовать в работе.

Для дальнейшей работы вам понадобится AWS-аккаунт

Эта книга содержит примеры кода, которые развертываются на AWS. Для работы с ними нужен аккаунт в AWS, в котором вы можете выполнить аутентификацию с правами администратора. Если у вас нет такого аккаунта или вы не знаете, как в нем аутентифицироваться, прочтите статью How to authenticate to AWS with IAM Identity Center («Как аутентифицироваться в AWS с помощью IAM Identity Center») на сайте этой книги (https://oreil.ly/Lqhly). В ней вы найдете информацию о том, как создать аккаунт в AWS (бесплатно), как создать пользовательский аккаунт с достаточными привилегиями и аутентифицироваться в AWS через Веб и командную строку.

Для развертывания тестового приложения в AWS выполните следующие шаги.

Шаг 1. Выберите регион AWS.

AWS имеет центры обработки данных по всему миру. Они сгруппированы по регионам и зонам доступности. *Регион AWS* — это отдельная географическая зона, например `us-east-2` (Огайо), `eu-west-1` (Ирландия) и `ap-southeast-2` (Сидней). В каждом регионе есть множество изолированных центров обработки данных, известных как зоны доступности, например `us-east-2a`, `us-east-2b` и т. д. Почти во всех примерах, приводимых в книге, будет задействован регион `us-east-2` (Огайо), поэтому вам нужно зайти в консоль AWS (https://console.aws.amazon.com) и в правом верхнем углу выбрать `us-east-2` в качестве региона использования (рис. 1.3). Вы также

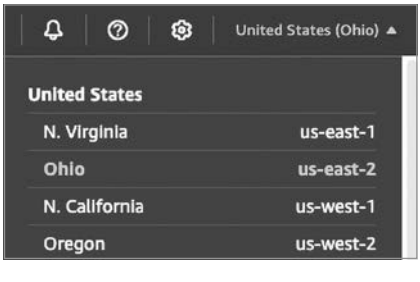

Рис. 1.3. Выберите us-east-2 в качестве своего региона AWS

можете назначить `us-east-2` своим регионом по умолчанию (https://oreil.ly/rybxa), чтобы случайно не оказаться в другом регионе. Я работаю с AWS уже больше десяти лет и все еще иногда случайно перескакиваю в другой регион, если так не делаю.

Шаг 2. Разверните экземпляр EC2.

Чтобы развернуть в AWS сервер, называющийся *экземпляром EC2* (Amazon Elastic Compute Cloud), перейдите в консоль EC2 (https://oreil.ly/lCa8Q) и нажмите кнопку Launch instance (Запустить экземпляр). Вы будете перенаправлены на страницу конфигурации экземпляра EC2 (рис. 1.4).

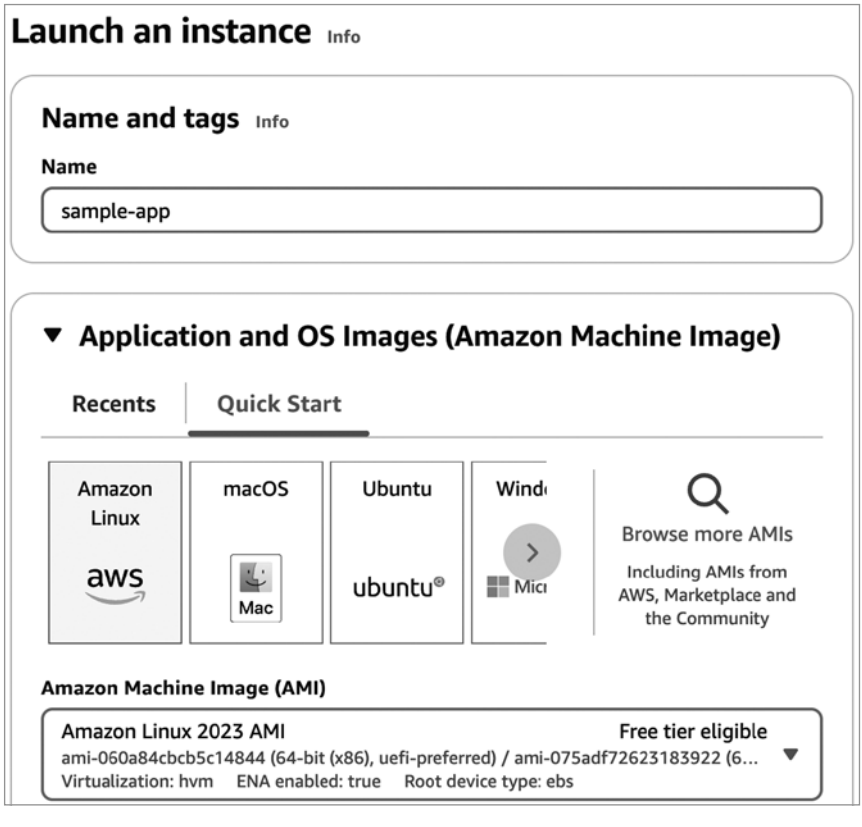

Рис. 1.4. Настройка имени и AMI для вашего экземпляра EC2

Заполните поле Name (Имя) для своего экземпляра, например `sample-app`, затем выберите Amazon Machine Image (AMI) — шаблон, определяющий операционную систему и прочее ПО, которые будут установлены (подробнее о машинных образах поговорим в главе 2). Пока выберите вариант по умолчанию — Amazon Linux.

Шаг 3. Сконфигурируйте экземпляр EC2.

Сконфигурируйте тип экземпляра и пару ключей (рис. 1.5).

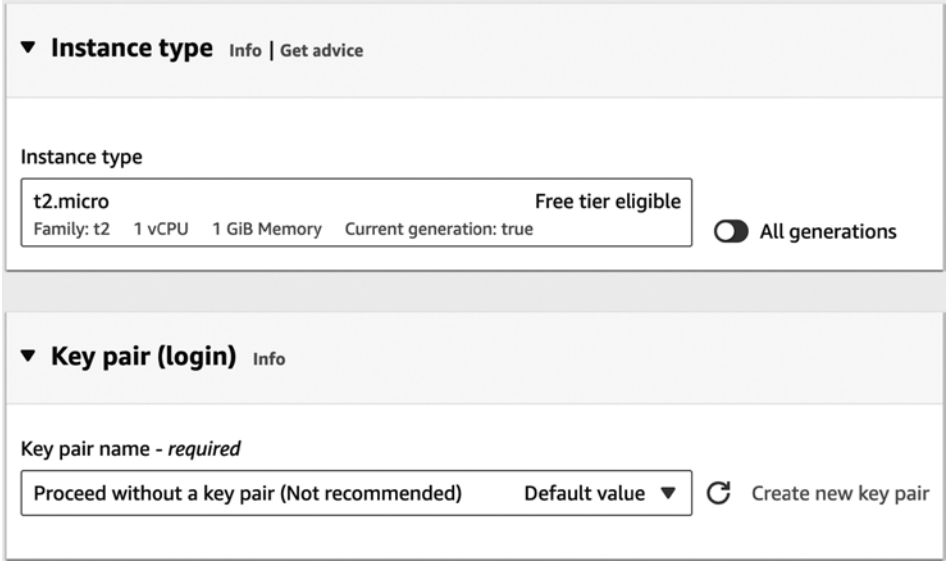

Рис. 1.5. Настройка типа экземпляра и пары ключей для вашего экземпляра EC2

Тип экземпляра — это тип используемого сервера, то есть вид CPU, памяти, жесткого диска и т. д. Для нашего небольшого упражнения можете взять вариант по умолчанию, например `t2.micro` или `t3.micro` — небольшие экземпляры (1 CPU, 1 Гбайт памяти), доступные на бесплатном уровне AWS. *Пара ключей* используется для подключения к экземпляру EC2 через Secure Shell (SSH). Подробнее об этом вы узнаете в главе 7. В этом примере мы не будем использовать SSH, поэтому нажмите Proceed without a key pair (Продолжить без пары ключей).

Шаг 4. Настройте параметры сети.

Прокрутите страницу вниз до настроек сети (рис. 1.6) (более подробно сетевое взаимодействие мы рассмотрим в главе 7). Пока можете оставить большинство параметров установленными по умолчанию. Для опции Network (Сеть) должно быть установлено ваше значение VPC по умолчанию (на рис. 1.6 ID моей VPC был `vpc-deb90eb6`, у вас будет другой), Subnet (Подсеть) должен быть No preference (Нет предпочтений), Auto-assign public IP (Автоназначение публичного IP) должен быть установлен в значение Enable (Включено). Единственный параметр, который вам нужно изменить, — это Firewall (security groups) (Брандмауэр (группы безопасности)): установите переключатель в положение Create security group (Создать группу безопасности),