ГЛАВА 1

Угрозы безопасности контейнеров

За последнее десятилетие масштабы применения контейнеров резко возросли. Идеи, лежащие в основе контейнеризации, существовали задолго до появления Docker. Но большинство наблюдателей сходятся во мнении, что именно платформа Docker с ее удобными утилитами командной строки дала толчок популярности контейнеров среди сообщества разработчиков после ее выхода в 2013 году.

У контейнеров много достоинств. Как гласит рекламный слоган Docker, они позволяют «создать один раз — выполнять где угодно». Это достигается благодаря тому, что приложение упаковывается в пакет со всеми своими зависимостями и изолируется от остальной части системы, на которой оно работает. У контейнеризированного приложения есть все необходимое для работы, и его легко оформить в виде образа контейнера, который будет одинаково запускаться на моем или вашем ноутбуке, в облаке и на сервере в центре обработки данных (ЦОД)[1].

Следствие этой изоляции — возможность параллельного выполнения нескольких различных контейнеров, которые не будут мешать друг другу. До появления контейнеров мешанина зависимостей могла легко превратиться в настоящий кошмар для разработчиков, когда двум приложениям требовались различные версии одних и тех же пакетов. Самым простым решением в такой ситуации было запускать приложения на отдельных машинах. Контейнеры позволяют изолировать зависимости друг от друга, благодаря чему выполнение нескольких приложений на одном сервере не доставляет никаких проблем. Вскоре стало очевидно, что контейнеризация дает возможность запускать множество приложений на одном хосте (будь то виртуальная машина или реальный сервер), не беспокоясь о зависимостях.

Следующим логичным шагом стало распределение контейнеризированных приложений по кластеру серверов. Оркестраторы наподобие Kubernetes автоматизируют этот процесс до такой степени, что больше не нужно вручную устанавливать

---

[1] Однако есть ограничения по архитектуре процессоров, о которых не следует забывать. К примеру, контейнер, собранный под arm64, не запустится под x86_64, так как у систем разные наборы инструкций. Более подробно об этом будет рассказано в главе 6. — *Примеч. науч. ред.*

приложения на конкретных машинах, достаточно сообщить оркестратору, какие контейнеры необходимо запустить, и он сам найдет подходящее место для каждого из них.

Рис. 1.1. Контейнеризованное приложение

С точки зрения безопасности контейнеризированная среда во многом схожа с традиционным развертыванием. Нарушители пытаются похитить данные, или изменить поведение системы, или использовать вычислительные ресурсы других людей для майнинга криптовалюты. При переходе к контейнерам ничего из этого не меняется. Но контейнеры существенно меняют способ выполнения приложений, что приводит к иному набору рисков (рис. 1.1).

Как показано на рис. 1.1, приложение и его зависимости работают внутри контейнера, который управляется оркестратором, таким как Kubernetes или AWS Elastic Container Service. Контейнер может работать непосредственно на реальном физическом хосте или внутри виртуальной машины (virtual machine, VM), которая, в свою очередь, запущена на физическом оборудовании. Главы 3 и 4 помогут вам понять, как контейнер изолирует приложение от хоста и от других контейнеров, а в главе 5 рассматриваются совершенно иные механизмы изоляции, используемые в виртуальных машинах. Понимание этих различий имеет решающее значение для эффективного анализа возможных рисков и угроз для контейнеризированных развертываний, а также для выбора мер по их смягчению и обеспечению безопасности контейнеров.

Риски, угрозы и уменьшение их последствий

Риск (risk) — потенциальная проблема, а также ее возможные последствия.

Угроза (threat) — путь реализации этого риска.

Смягчение последствий (mitigation) — контрмеры, с помощью которых можно предотвратить угрозу или по крайней мере снизить вероятность ее успешной реализации.

Скажем, существует риск, что кто-то украдет из вашего дома ключи от вашей же машины и уедет на ней. Угрозами в данном случае будут различные способы кражи ключей: разбить окно и дотянуться до них; просунуть удочку через щель для почты; постучать в дверь и отвлечь вас, пока сообщник быстро проскользнет внутрь и схватит ключи. Контрмерой может быть простое правило — не хранить ключи на видном месте.

В зависимости от организации риски могут сильно различаться. Основной риск для банка, хранящего клиентские деньги, — их кража. Для интернет-магазина основная «головная боль» — мошеннические транзакции. Ведущий личный блог пользователь может бояться, например, что кто-то взломает его учетную запись, выдаст себя за него и начнет публиковать неподобающие комментарии. В разных странах законодательство о защите персональной информации имеет свои особенности, поэтому различается и риск утечки личных данных пользователей: в одних странах риски «лишь» репутационные, в то время как в Европе Общий регламент защиты персональных данных (General Data Protection Regulation, GDPR) допускает штрафы до 4 % общего дохода компании (https://oreil.ly/guQg3).

Поскольку риски весьма разнятся, то сильно различается и относительная значимость потенциальных угроз, равно как и соответствующий набор мер по их предотвращению. В основе управления рисками лежит процесс их систематизации, перечисления возможных угроз, расстановки их по приоритетам и выбора способов противодействия им. *Моделирование угроз* (threat modeling) — процесс выявления и перечисления потенциальных угроз системе. За счет планомерного анализа ее компонентов и вероятных векторов атаки модель угроз помогает определить наиболее уязвимые для атак места.

Единой универсальной модели угроз не существует: все зависит от рисков, конкретной среды, организации и запускаемых приложений. Тем не менее можно выделить ряд потенциальных угроз, характерных для большинства, если не всех, контейнерных развертываний.

Модель угроз для контейнеров

В частности, модель угроз можно рассматривать с точки зрения ее участников, в число которых могут входить:

- *внешние нарушители* (external attackers), пытающиеся получить доступ к развернутой системе извне;
- *внутренние нарушители* (internal attackers), сумевшие проникнуть в какую-то часть развернутой системы;
- *недобросовестные сотрудники* (malicious internal actors), например разработчики и администраторы с определенным уровнем привилегий доступа к развернутой системе;
- *невнимательные сотрудники* (inadvertent internal actors), которые могут непреднамеренно вызывать проблемы;
- *процессы приложений* (application processes), которые не являются людьми-злоумышленниками, тем не менее имеют определенный программный доступ к системе.

У каждого из этих источников угроз есть определенный набор прав доступа, который необходимо учитывать.

- Какой уровень доступа к системе есть у них через учетные данные? Например, обладают ли они доступом к пользовательским учетным записям на машинах хостов, где работает развернутая система?
- Какие привилегии у них есть в системе? В Kubernetes этот пункт относится к настройкам управления доступом для всех пользователей, в том числе анонимных, на основе ролей. Права доступа могут дополнительно контролироваться механизмами политик или инструментами безопасности в среде выполнения. В общедоступных облачных средах права доступа также зависят от политик управления идентификацией и доступом (identity and access management, IAM).
- Какие права доступа к сети у них есть? Например, какие части системы включены в виртуальное частное облако (Virtual Private Cloud, VPC)? Действуют ли политики сетевой безопасности для ограничения доступа?

Существует несколько возможных путей атаки на развернутую контейнеризированную систему, и, чтобы их систематизировать, можно, например, проанализировать потенциальные векторы атак на каждом из этапов жизненного цикла контейнера (рис. 1.2).

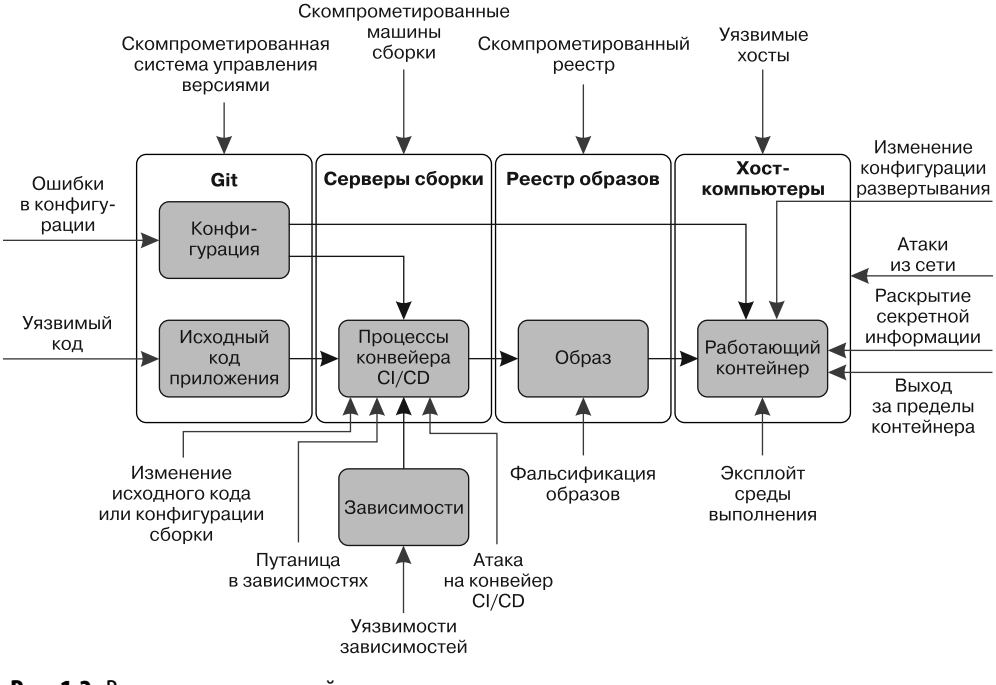

Рис. 1.2. Векторы атак на контейнеры

Рассмотрим эти векторы более подробно.

Уязвимости в коде и зависимостях

Жизненный цикл контейнера начинается с написания разработчиком кода приложения. В этом коде, равно как и в его зависимостях, могут содержаться изъяны (уязвимости). Существуют тысячи известных уязвимостей, которыми (если они есть в приложении) могут воспользоваться злоумышленники. Чтобы избежать запуска контейнеров с известными уязвимостями, образы контейнеров необходимо проверять с помощью специальных средств анализа, о которых пойдет речь в главе 8. Причем делать это нужно регулярно, поскольку уязвимости постоянно обнаруживаются даже в уже существующем коде. В процессе анализа должны также выявляться случаи, когда контейнеры используют устаревшие пакеты, требующие установки патчей безопасности. Кроме того, существуют средства проверки, способные выявлять встроенное в образы вредоносное программное обеспечение (ПО).

Некорректно настроенные образы контейнеров

Написанный код встраивается в образ контейнера. В ходе сборки образа контейнера возникает множество возможностей для внедрения уязвимостей, которые впоследствии могут быть использованы для атаки на работающий контейнер. В их число входит запуск контейнера от имени суперпользователя, в результате чего у него оказывается больше привилегий на хосте, чем ему действительно нужно. Больше информации об этом — в главе 6.

Атаки на цепочку поставок и систему сборки

Исходный код и конфигурация сборки поступают на вход процесса сборки, который создает образ контейнера. Собранный образ контейнера сохраняется в реестре, откуда извлекается перед запуском. Если злоумышленник сумеет изменить исходный код, повлиять на сборку образа контейнера, подменить образ контейнера в реестре или добиться загрузки неправильного образа, он сможет вставить вредоносный код, который потом будет запущен в рабочей среде.

Как убедиться, что для сборки образа используются именно тот исходный код и зависимости? Как гарантировать соответствие извлекаемого образа тому, который был ранее помещен в реестр? Не внесли ли злоумышленники в него изменения? Любой субъект, способный повлиять на образ в промежутке между сборкой и развертыванием, сможет выполнить произвольный код в вашей системе. Об этих атаках на «цепочку поставок» подробнее рассказывается в главе 7.

Некорректно настроенные контейнеры

Как мы обсудим в главе 11, контейнер можно запустить с настройками, в результате которых у них появляются ненужные, а порой и незапланированные привилегии. Скачав файлы конфигурации YAML из Интернета, не запускайте их, пока не убедитесь в отсутствии небезопасных настроек!

Уязвимые хосты

Контейнеры выполняются на хостах, и важно убедиться, что на них не будет запущен уязвимый код (например, старые версии компонентов механизма оркестрации с известными уязвимостями). Имеет смысл уменьшить количество установленного на каждом хосте ПО, чтобы сократить поверхность атаки. Кроме того, необходимо правильно настроить хосты, следуя практическим рекомендациям по обеспечению безопасности. Все это обсуждается в главе 4. Желательно также ограничить доступ пользователей к хостам. Для этого можно, например, использовать подход GitOps, о котором рассказывается в главе 9, — в этом случае весь код и конфигурация хранятся в системе управления версиями и обновляются с помощью автоматизированного процесса, а не вручную.

Незащищенная сеть

Контейнерам обычно требуется взаимодействовать друг с другом или с внешними системами. В главе 12 рассказывается, как работает сеть в контейнерах, а в главе 13 — как устанавливать защищенные соединения между компонентами.

Раскрытие секретных данных

Для взаимодействий с другими компонентами системы приложениям часто требуются учетные данные, токены или пароли. При развертывании в контейнере эти секретные значения необходимо передавать в контейнеризированный код. В главе 14 вы узнаете, что у этой задачи существует несколько вариантов решения и степень их безопасности может существенно различаться.

Эксплойты приложения в среде выполнения

Широко используемые среды выполнения контейнеров, включая `containerd` и `CRI-O`, уже хорошо зарекомендовали себя на практике, но не исключено, что в них все же остаются программные ошибки, которые позволят вредоносному коду, работающему внутри контейнера, просочиться за пределы контейнера в пространство хостовой операционной системы. Аналогично время от времени обнаруживаются уязвимости в ядре, которые могут позволить выход из контейнера или повышение привилегий. В главе 4 вы узнаете об изоляции, предназначенной для ограничения кода приложения рамками контейнера. Ущерб от выхода за пределы контейнера для ряда приложений может быть столь велик, что имеет смысл применить более эффективные механизмы изоляции, например те, о которых говорится в главе 10. А в главе 15 мы рассмотрим инструменты безопасности, помогающие обнаружить и предотвратить выход за пределы контейнера, а также выявить эксплойты в среде выполнения (рантайме).

Некоторые векторы атак, показанные на рис. 1.2, не относятся к теме книги.

- Исходный код обычно хранится в репозиториях, потенциально доступных для атак, цель которых — взлом приложения. Аналогично образы контейнеров

хранятся в реестрах. Необходимо обеспечить должный контроль доступа пользователя к репозиторию исходного кода и реестру образов.

- Хосты соединены по сети, зачастую с использованием VPC и, как правило, подключенной к Интернету. Как и при стандартном развертывании, необходимо защитить физические хосты (или виртуальные машины) от злоумышленников. Безопасные настройки сети, использование межсетевого экрана, а также IAM актуальны в облачном развертывании так же, как и в локальном. Эти меры защиты также относятся и к машинам сборки, на которых компилируется код и создаются образы контейнеров.
- Контейнеры обычно работают под управлением оркестратора. Недостаточная безопасность настроек оркестратора или отсутствие должного контроля над доступом с правами администратора открывают злоумышленникам дополнительные возможности для вмешательства в процесс развертывания.

Помимо этой книги, оценить потенциальные угрозы для ваших операций вам помогут следующие ресурсы.

- Компания Microsoft опубликовала хорошее обобщенное описание угроз, тактик и процедур, которые злоумышленники могут использовать против Kubernetes, в документе *Threat Matrix for Kubernetes* (https://microsoft.github.io/Threat-Matrix-for-Kubernetes).
- Исследователи из Сингапурского университета технологий и дизайна и Цюрихского технологического института опубликовали статью под названием *Threat Modeling and Security Analysis of Containers: A Survey* (https://arxiv.org/pdf/2111.11475).

Границы зон безопасности

Границы зон безопасности (иногда называемые границами доверия) между частями системы означают, что наборы прав доступа в этих частях отличаются. Иногда границы задаются в ходе администрирования. Например, в системах под управлением Linux системный администратор может установить границы зон безопасности, определяющие, какие файлы доступны пользователю, просто изменив группы, в которых состоит данный пользователь. Если вы забыли, как работает система прав доступа к файлам в Linux, в главе 2 будет краткое напоминание.

Иногда можно услышать, что «контейнеры не являются зоной безопасности». Это не совсем так. В действительности контейнер представляет собой зону безопасности — просто не самую надежную! Предполагается, что код приложения выполняется внутри контейнера и не имеет доступа к коду или к данным за его пределами, за исключением случаев, когда ему явно не предоставлено такое разрешение (например, путем подключения внешнего тома). Как вы узнаете далее, защита, обеспечиваемая контейнером, относительно слаба, поэтому вам

потребуется провести дополнительные границы, чтобы гарантировать безопасность ваших приложений.

Чем строже граница зон безопасности между злоумышленником и его целью (например, данными пользователей), тем сложнее ему достичь этой цели.

Векторы атак, ранее описанные в разделе «Модель угроз для контейнеров», могут использоваться комплексно в целях проникновения через несколько границ зон безопасности. Приведу следующие примеры.

- Злоумышленник может обнаружить, что уязвимость в одной из зависимостей приложения позволяет ему удаленно выполнять код внутри контейнера.
- Допустим, у взломанного компьютера нет прямого доступа к каким-либо ценным данным. Злоумышленнику необходимо найти способ попасть в другой контейнер или на хост. Один из вариантов — уязвимость, позволяющая выйти за рамки контейнера; другой — небезопасная конфигурация контейнера. Если злоумышленнику удастся воспользоваться одним из этих способов, он получит доступ к хосту.
- Следующий шаг — поиск возможности получить на хосте привилегии суперпользователя, что может оказаться тривиальной задачей, если код приложения выполняется от имени суперпользователя внутри контейнера, как вы увидите в главе 4.
- Имея привилегии суперпользователя на физическом хосте, злоумышленник способен получить доступ ко всему, что открывается с хоста или из любого контейнера, запущенного на этом хосте.

Добавление новых границ безопасности и усиление существующих значительно усложняет жизнь взломщиков.

При построении модели угроз важно учитывать, что атака может быть совершена изнутри среды, в которой работают ваши приложения. При развертывании в облаке отдельные ресурсы порой используются совместно с другими пользователями и их приложениями. Совместное использование ресурсов компьютера называется *мультиарендностью* (multitenancy) и серьезно влияет на модель угроз.

Мультиарендность

В мультиарендной среде различные пользователи — *арендаторы* (tenants) — выполняют задания на общем оборудовании. (Термин «мультиарендность», или «мультитенантность» можно также встретить в контексте программных приложений, где он означает нескольких пользователей, работающих с одним экземпляром программы, однако в нашем случае совместно используемым будет лишь аппаратное обеспечение.) В зависимости от владельцев этих различных заданий, а также степени взаимного доверия арендаторов могут

понадобиться более жесткие границы между ними, чтобы предотвратить их возможное негативное влияние друг на друга.

Мультиарендность — идея, появившаяся еще во времена мейнфреймов в 1960-х, когда пользователи арендовали ресурсы процессора (CPU), память и место на диске на совместно используемой машине. Эта концепция не так уж сильно отличается от современных общедоступных облачных сервисов, например Amazon AWS, Microsoft Azure и Google Cloud Platform, в которых пользователи также арендуют вычислительные ресурсы — время CPU, оперативную память, дисковое пространство — наряду с прочими возможностями и управляемыми сервисами. С тех пор как в 2006 году в Amazon AWS появилась служба EC2, стало возможно арендовать экземпляры виртуальных машин, запущенные на стойках серверов в центрах обработки данных, разбросанных по всему миру. На одном физическом сервере может работать несколько VM, а вы, как пользователь, управляющий группой таких VM, можете не знать, кто управляет соседними VM.

Совместно используемые машины

Бывают и ситуации, когда на одной (возможно, виртуальной) машине Linux работают сразу несколько пользователей. Такой вариант истинной мультиарендности часто встречается в университетах, где пользователи не доверяют друг другу и, говоря откровенно, администраторы не доверяют пользователям. В подобной среде у каждого пользователя есть своя учетная запись, а доступ ограничивается средствами операционной системы (ОС) Linux, чтобы, например, пользователь мог изменять только файлы в своих каталогах. Представьте себе, какой бы хаос возник, если бы студенты читали или, хуже того, редактировали файлы своих однокурсников.

Как вы увидите в главе 4, все контейнеры, действующие на одном хосте, используют одно и то же ядро. Если на машине запущен демон Docker, то любой пользователь, имеющий право выполнять команды `docker`, по сути, обладает `root`-правами, и вряд ли администратору следует предоставлять не доверенным пользователям такие возможности.

В корпоративной и особенно в облачной среде подобные совместно используемые машины встречаются реже. Обычно пользователи (или команды, доверяющие друг другу) задействуют собственные ресурсы, выделенные им в виде виртуальных машин.

Виртуализация

В целом считается, что виртуальные машины достаточно надежно изолированы друг от друга, то есть маловероятно, что пользователи, работающие на соседних виртуальных машинах, смогут вмешаться в вашу систему или

увидеть, что в ней происходит. Как достигается подобная изоляция, можно узнать в главе 5. Более того, в соответствии с общепринятым определением (https://en.wikipedia.org/wiki/Multitenancy) виртуализация вообще не означает мультиарендность. Мультиарендность (или мультитенантность) — это когда различные группы пользователей совместно работают с одним экземпляром программного обеспечения. В случае виртуализации у пользователей нет доступа к гипервизору, управляющему их виртуальными машинами, поэтому фактически они не используют совместно никаких программ.

Впрочем, изоляция виртуальных машин отнюдь не идеальна. Пользователи и раньше жаловались на проблемы с «шумными соседями», когда совместное использование реальной машины может привести к неожиданным колебаниям производительности. Компания Netflix одной из первых начала применять общедоступные облачные сервисы. В 2010 году в блоге компании была опубликована статья (https://oreil.ly/8zHGy), в которой говорилось, что создаваемые системы могли осознанно прекратить выполнение подзадачи, если она выполнялась слишком медленно. В то же время другие специалисты утверждают, что проблема «шумных соседей» не столь серьезна (https://oreil.ly/iE4qE).

Известны также случаи, когда уязвимости в программном обеспечении приводили к нарушению границ между виртуальными машинами. Поставщики облачных сервисов принимают дополнительные меры для защиты от таких уязвимостей на уровне гипервизора.

Для некоторых приложений и организаций (особенно правительственных, финансовых и медицинских) последствия несанкционированного доступа могут быть настолько серьезными, что требуется полное физическое разделение. Чтобы гарантировать полную изоляцию рабочих заданий, можно использовать частное облако, работающее в вашем собственном центре обработки данных или под управлением выбранного вами поставщика сервисов. Частные облака иногда подразумевают дополнительные меры защиты, например более строгие проверки анкетных данных персонала, имеющего доступ к ЦОД.

Компания Google предоставляет подробную документацию по мерам безопасности своих ЦОД (https://datacenters.google/advancing-security).

Многие поставщики облачных сервисов предлагают варианты VM, при которых гарантируется, что одну реальную машину использует только один клиент. Кроме того, можно арендовать выделенные физические серверы, обслуживаемые поставщиками облачных сервисов. В обоих сценариях вы полностью избегаете проблемы «шумных соседей» и получаете более надежную изоляцию между реальными машинами.

Неважно, арендуете вы физические или виртуальные машины в облаке либо используете собственные серверы, — при работе с контейнерами может возникнуть необходимость учесть границы зон безопасности между различными группами пользователей.

Мультиарендность контейнеров

В главе 4 вы увидите, что изоляция контейнеров не так строга, как изоляция виртуальных машин. Хотя это зависит от профиля рисков, вряд ли стоит размещать контейнеры на одной машине с пользователями или организациями, которым вы не доверяете. Даже если все контейнеры на ваших машинах запускаются лично вами или доверенными лицами, все равно следует учитывать человеческий фактор и предусмотреть меры, которые не позволят контейнерам влиять друг на друга.

В Kubernetes можно использовать *пространство имен* (namespace), чтобы разделить кластер компьютеров на части между различными людьми, командами или приложениями.

«Пространство имен» — термин с несколькими значениями. В Kubernetes пространство имен представляет собой высокоуровневую абстракцию, позволяющую разделить ресурсы кластера и применить к ним различные политики доступа. В Linux же пространство имен — это низкоуровневый механизм изоляции ресурсов машины, доступных процессу. Более подробно пространства имен будут описаны в главе 4.

Чтобы указать, какие пользователи и компоненты могут обращаться к тем или иным пространствам имен Kubernetes, используется механизм управления доступом на основе ролей (role-based access control, RBAC). Описание подробностей его настройки выходит за рамки книги[1], однако важно отметить, что RBAC позволяет контролировать только действия, производимые через Kubernetes API. Контейнеры приложений в подах Kubernetes, работающих на одном хосте, защищены друг от друга лишь с помощью механизмов изоляции контейнеров, описанных в книге, даже если они находятся в разных пространствах имен. Если злоумышленнику удастся выйти за пределы контейнера на хост, то границы пространств имен Kubernetes никак не помешают ему влиять на другие контейнеры.

[1] Если вам интересна эта тема, можете ознакомиться с ней в книге: *Мадаппарамбат Г., Маккендрик Р.* Kubernetes. Полное руководство по развертыванию и управлению Kubernetes в облачных и локальных средах. 2-е изд. — *Примеч. науч. ред.*

Экземпляры контейнеров

Такие облачные сервисы, как Amazon AWS, Microsoft Azure и Google Cloud Platform, предоставляют множество *управляемых сервисов* (managed services), позволяющих пользователю арендовать программное обеспечение, хранилище и другие компоненты без необходимости самостоятельно устанавливать или обслуживать их. Классическим примером является Amazon Relational Database Service (RDS): с его помощью можно легко развернуть базу данных (БД), основанную на таком популярном ПО, как PostgreSQL, а для резервного копирования данных достаточно поставить одну галочку (и оплатить счет, конечно).

Управляемые сервисы распространились и на мир контейнеров. С помощью сервисов Azure Container Instances и AWS Fargate можно запускать контейнеры, не задумываясь о том, на каком компьютере (или VM) они будут работать.

Это снимает с наших плеч немалый груз администрирования и позволяет легко масштабировать развертываемую систему. Однако (в теории) экземпляры контейнеров разных пользователей могут располагаться на одной VM. Уточните этот момент у вашего поставщика облачных услуг на всякий случай.

Теперь вы уже знакомы со многими потенциальными угрозами для развертываемых вами систем. Прежде чем двигаться дальше, я бы хотела поговорить об основных принципах безопасности, которые пригодятся при выборе утилит и средств защиты, требуемых при развертывании.

Принципы безопасности

Это общие рекомендации, которые обычно считаются разумным подходом независимо от того, о безопасности чего идет речь.

Минимум привилегий

Принцип наименьших привилегий предполагает ограничение прав доступа пользователя или компонента до минимума, необходимого для выполнения задачи. Например, для микросервиса, осуществляющего поиск товара в приложении электронной коммерции, данный принцип означает, что учетная запись этого микросервиса должна предоставлять ему доступ к части базы данных только для чтения. Ему не требуется, скажем, информация о пользователях или платежах, равно как нет нужды записывать в БД информацию о товарах.