MINIMIZING IMAGES OF DOCKER CONTAINER ROOT FILE SYSTEMS

I. Nikolaeva, I. Gankevich[a]

Saint Petersburg State University, 13B Universitetskaya Emb., Saint Petersburg, 199034, Russia

E-mail: [a] i.gankevich@spbu.ru

Containers is a tool which is often used in development and testing applications because they are easy-to-use and lightweight. The container is built based on an image, which is a template for the container and a way to transmit it over the network. Images can be up to several gigabytes in size. Thus, if you need to transfer the large application container over the network, or the available memory is scarce (for example, when developing IoT systems), it is necessary to minimize the size of the image. Container images are built on top of the Linux kernel. For interactive work with containers, they include instructions and files that the application never works with inside the container. The applications that will be launched in the container and their number are known in advance, so you can use Linux debugging tools to find out which files are used by applications in the container, and which can be excluded from it. This is how the Chainsaw application works. This paper presents the results of a study of several different Docker container images. For some application we obtained threefold decrease of the image size; also, we found that the size of the reduced image is significantly affected by the base image and the programming language in which the application that runs inside the container is written.

Keywords: application container, ptrace.

Irina Nikolaeva, Ivan Gankevich

МИНИМИЗАЦИЯ ОБРАЗОВ КОРНЕВЫХ ФАЙЛОВЫХ СИСТЕМ DOCKER КОНТЕЙНЕРОВ

И. Николаева И. Ганкевич[a]

Санкт-Петербургский государственный университет, Россия, 199034, Санкт-Петербург, Университетская наб., д. 7–9

E-mail: [a] i.gankevich@spbu.ru

В последнее время при разработке и тестировании приложений достаточно часто обращаются к контейнерам. Это обусловлено тем, что контейнер — это удобный в использовании и легковесный инструмент. Контейнер собирается на основе образа, который является шаблоном будущего контейнера, а также с помощью образа контейнер передается по сети. Размер образов может достигать нескольких гигабайт. Таким образом, если требуется передать по сети приложение, образы контейнеров которого достаточно объемны, либо присутствует физическое ограничение доступной памяти (например, при разработке IoT систем), необходима минимизация размера образа. Образы контейнеров строятся на основе ядра операционной системы Linux, и вся работа с образами — это инструкции оболочки. Для интерактивной работы с контейнерами они включают в себя инструкции и файлы, с которыми приложение внутри контейнера никогда не работает. Сами приложения, которые будут запущены в контейнере, и их количество заранее известны, поэтому при помощи средств отладки Linux можно понять, какие файлы используются приложениями в контейнере, а какие можно исключить из него. Именно так работает приложение chainsaw. В данной работе представлены результаты исследования нескольких разных образов Docker контейнеров. Было получено уменьшение в 3 раза для некоторых из образов, а также установлено, что на размер уменьшенного образа существенно влияют используемый базовый образ и язык программирования, на котором написано приложение, работающее внутри контейнера.

Ключевые слова: контейнер приложений, ptrace.

Николаева Ирина, Ганкевич Иван

1. Введение

Контейнер — это удобный способ изолированного запуска приложения, который реализуется путем создания отдельного окружения для приложения. У каждого такого приложения будут своя версия корневой файловой системы и список процессов.

Образы корневых файловых систем контейнеров — это их неизменяемые шаблоны, которые несут в себе информацию об изолированном окружении приложений. Образы чаще всего занимают несколько сотен мегабайт, но на практике бывают больше гигабайта. Пространство на диске занимает не только запущенный контейнер приложения, но и его образ, а также внешнее хранилище, которое может быть ему выделено.

Приложение, построенное при помощи технологии контейнеров, в большинстве случаев будет работать на основе нескольких контейнеров разных приложений. Тем самым, увеличивается занимаемое приложением пространство на диске, а также возрастает нагрузка на сеть при передаче образов. Например, в устройствах IoT, где присутствует ограничение как по свободной памяти, так и по пропускной способности сети, появляется необходимость оптимизации этих двух показателей, что можно достичь путем уменьшения размера образов.

Существуют различные решения, которые позволяют существенно сократить время на передачу образов в крупных распределенных системах. Исследование публичных образов позволило установить большой потенциал образов к дедубликации [1]. Faster Image Distribution использует P2P сеть и оптимизацию локального хранилища образов [2], а дедубликация уровней, на которых строятся образы, позволила в несколько раз сократить время обновления и время доставки образов [3,4]. Однако, такие решения не применимы при использовании на отдельных машинах.

2. Приложение chainsaw

Образы создаются, как правило, менеджерами пакетов для интерактивного использования человеком. Как следствие, они могут включать в себя файлы, которые никогда не используются приложением внутри контейнера. Внутри контейнера работает ограниченное количество заранее известных приложений, которые можно проанализировать при помощи встроенных в Linux средств отладки, чтобы обнаружить используемые приложением файлы: исполняемые, конфигурационные файлы, библиотеки, — и удалить все остальные. Полученные образы будут иметь минимально возможный размер.

Этот подход используется в приложении chainsaw [5], которое было написано на языках программирования C и C++. Это приложение, используя системный вызов *ptrace*, который перехватывает работающие с файлами системные вызовы, извлекает пути к файлам, которые были использованы пользовательским приложением, запущенным в дочернем процессе. Приложение CARE работает подобным образом, однако, после получения путей к файлам, оно сохраняет их в архив, который позволит добиться полного повторения условий, в которых работало приложение [6].

Приложение chainsaw состоит из четырех частей.

- Команда *chainsaw-blacklist* выводит список всех файлов, исключая домашние директории пользователей и виртуальные файловые системы.
- Команда *chainsaw-whitelist* запускает исследуемую программу и выводит список файлов, которые были получены от ptrace.
- Команда *chainsaw-diff* находит разницу между этими двумя списками.
- Команда *chainsaw-cut* удаляет все файлы предыдущего этапа.

Для сборки программы внутри образа нужны зависимости: python 3.5 или выше, pip, библиотеки ninja, meson, которые собирают исполняемые файлы.

2.1 Эксперименты с образами контейнеров

Были выбраны шесть приложений, для которых были получены новые образы при помощи chainsaw.

- Nginx — веб-сервер для Linux-подобных операционных систем, написан на языке программирования C.
- Grafana — веб-сервис для анализа временных рядов, написан на языке программирования TypeScript.
- Postgres — SQL система управления базами данных, написан на языке программирования C++.
- Tomcat — веб-сервер на языке программирования Java.
- Peer-calls — приложение для видео конференц-связи, написан на языке программирования Go [7].
- Mycodo — приложение для Raspberry Pi, собирающее и отображающее состояние IoT системы, написан на языке программирования Python [8].

Во время работы с образами этих приложений пришлось столкнуться с различными особенностями работы как самих приложений, так и базовых образов, на основе которых они работают. В некоторых случаях chainsaw не обнаруживал библиотеки, как системные, так и специальные для работы приложений и инструкций. Для решения этой проблемы было решено использовать chainsaw-whitelist на них, чтобы пропущенные библиотеки обнаруживались автоматически. Если точка запуска контейнера представлялась сложным скриптом (как в образе для Postgres и Tomcat), chainsaw мог упустить системные инструкции, использованные в скрипте. Также, не обнаруживалось приложение или скрипт, запускаемый для анализа, и его нужно было вручную добавлять в whitelist.

При использовании образа, построенного на Alpine Linux (контейнеры приложений Grafana, Peer-calls), было обнаружено, что в этом дистрибутиве отсутствуют приложения C, C++ компиляторов, cmake, make, их библиотеки, которые нужны для работы ninja и meson. Таким образом, при работе с данным дистрибутивом, помимо названных ранее зависимостей необходимо устанавливать еще дополнительные, из-за чего теряется универсальность работы chainsaw с образами.

После завершения работы команды chainsaw-cut все файлы, что остаются в корне файловой системы, переносятся в пустой образ scratch, из-за чего в новый образ не переносились настройки окружения, и их также было необходимо указывать вручную.

Образ приложения Peer-calls тоже построен на основе scratch, куда переносится только один исполняемый файл, собранный Go. Была попытка при помощи созданного для Alpine Linux образа с необходимыми зависимостями [9] провести такие же действия, как с остальными образами, однако, результирующий образ стал больше исходного. Скорее всего, могли остаться директории и символьные ссылки, которые не обрабатываются chainsaw.

Самый наглядный результат был получен для приложения Mycodo. Оно состоит из нескольких контейнеров, но самый большой образ для них — app, который является шаблоном для двух контейнеров. Чтобы контейнеры в среде работали корректно, автор приложения перед запуском приложения устанавливает большое количество разных зависимостей, которые настраивают окружение, но они в процессе работы приложения не используются. За счет избавления от этих зависимостей удалось добиться уменьшения образа в 3 раза.

Как было отмечено, это приложение разработано для установки на Raspberry Pi, который имеет ограниченное количество доступной памяти. Следовательно, уменьшение образа позволяет хранить на 600МБ больше пользовательских данных о статистике IoT системы.

Однако, большую сложность при работе с chainsaw вызывала интерпретация ошибок запуска контейнера, потому что в некоторых ситуациях было крайне сложно понять, какой удаленный файл необходим для корректной работы приложений внутри контейнера.

2.2 Результаты

Результаты проведенных опытов приведены в Таблице 1.

Таблица 1. Сравнение размеров образов.

Приложение	Размер образа из реестра, МБ	Размер полученного образа, МБ
Nginx	133,12	8,64
Grafana	198,41	181,18
Postgres	314,68	104,57
Tomcat	667,44	219,11
Peer-calls	22,06	22,93
Mycodo app	1110,00	366,99

По итогам исследования было выведено две зависимости, согласно которым изменяется размер нового образа относительно оригинального.

- Чем более сложный базовый образ используется для контейнера приложения, тем более существенное будет уменьшение размера нового образа.
- Чем более высокоуровневый язык программирования использован в приложении, тем меньшее количество файлов можно безвредно исключить из образа.

3. Заключение

Полученные успешные результаты — подтверждение целесообразности использования приложения chainsaw при разработке приложений на основе технологии Docker. Несмотря на возникшие проблемы при работе с chainsaw, которые еще предстоит прорабатывать, это эффективный инструмент для уменьшения размеров образов корневых файловых систем контейнеров, особенно если разработчик имеет представление, какие файлы действительно требуются для корректного исполнения внутри контейнера, но не может самостоятельно обнаружить те, которые не будут использоваться внутри него.

Список литературы

[1] Zhao N. et al. Large-Scale Analysis of Docker Images and Performance Implications for Container Storage Systems //IEEE Transactions on Parallel and Distributed Systems. – 2020. – Т. 32. – №. 4. – С. 918-930.

[2] Kangjin W. et al. Fid: A faster image distribution system for docker platform //2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W). – IEEE, 2017. – С. 191-198.

[3] Lu Z. et al. An acceleration method for Docker image update //2019 IEEE International Conference on Fog Computing (ICFC). – IEEE, 2019. – С. 15-23.

[4] Zheng C. et al. Wharf: Sharing docker images in a distributed file system //Proceedings of the ACM Symposium on Cloud Computing. – 2018. – С. 174-185.

[5] GitHub репозиторий приложения chainsaw [Электронный ресурс]: URL: https://github.com/igankevich/chainsaw (дата обращения: 23.08.21).

[6] CARE [Электронный ресурс]: URL: https://proot-me.github.io/care/ (дата обращения: 23.08.21).

[7] GitHub репозиторий приложения Mycodo [Электронный ресурс]: URL: https://github.com/kizniche/Mycodo (дата обращения: 23.08.21).

[8] GitHub репозиторий приложения peer-calls [Электронный ресурс]: URL: https://github.com/peer-calls/peer-calls (дата обращения: 23.08.21).

[9] Docker Hub репозиторий образа chainsaw [Электронный ресурс]: URL: https://hub.docker.com/r/lazydaredevil/chainsaw (дата обращения: 23.08.21).