ОЦЕНКА ВЛИЯНИЯ ФУНКЦИОНАЛЬНОЙ ПОЛНОТЫ СИСТЕМ РАЗВЕРТЫВАНИЯ НА СКОРОСТЬ ЗАПУСКА КОНТЕЙНЕРОВ

К.П. Власов, *магистрант*
И.Б. Трамов, *магистрант*
М.С. Кирпиченко, *магистрант*
А.С. Саадуев, *магистрант*
Московский государственный технический университет имени Н.Э. Баумана (Россия, г. Москва)

DOI:10.24412/2500-1000-2022-3-2-129-133

Аннотация. *В ходе проведения исследования были произведены замеры скорости работы систем развертывания контейнеризованных приложений с целью определения степени влияния функциональной полноты таких систем на скорость запуска контейнеров микросервисов в них. Приведены список и краткое описание сравниваемых систем. Произведено функциональное сравнение данных систем. Проведен сравнительный анализ полученных в ходе проведения экспериментов данных.*

***Ключевые слова*:** *контейнеризация приложений, развертывание приложений, Docker, DevOps, CI/CD.*

В настоящее время, благодаря быстрому развитию технологий разработки программного обеспечения, в том числе инструментов командной разработки и инструментов развертывания приложений, сложные программные комплексы разрабатываются гораздо проще и быстрее, чем когда-либо раньше. В том числе, это стало возможным благодаря активному использованию микросервисной архитектуры.

Микросервисная архитектура позволяет разбить программу на блоки – микросервисы, каждый из которых способен работать автономно и связан с другими программными блоками посредством легковесных протоколов – REST, SOAP, gRPC или иными способами. Каждый такой блок с целью удобства развертывания в условиях кластера может быть запакован в контейнер. В такой контейнер помещается сам программный код и необходимые зависимости, он является стандартным объектом процесса развертывания. Использование технологий контейнеризации позволяет разработчикам разворачивать приложения в условиях различного аппаратного или программного окружения, не меняя при этом код самого сервиса, что значительно повышает простоту масштабируемости в условиях высоких нагрузок. Кроме того, технология контейнеризации позволяет изолировать приложения друг от друга, что делает их работу гораздо более безопасной как с точки зрения данных, так и с точки зрения отказоустойчивости.

Технология контейнеризации представляет из себя совокупность различных программных компонентов, позволяющих запускать отдельные программные контейнеры без создания виртуальных машин. По сравнению с технологией виртуализации, использование технологии контейнеризации дает заметный выигрыш в производительности, а именно: повышает вычислительную производительность на 4-5%, приводит к ускорению на 30% операций при работе с оперативной памятью (трансляция адресов), на 30-50% при доступе к файлам на жестких дисках, на 40-60% повышается пропускная способность сети [1]. Данные показатели достигаются за счет того, что в технологии контейнеризации нет необходимости запускать отдельную операционную систему в каждом контейнере. Вместо этого, контейнер работает поверх операционной системы машины-хоста, совместно используя в работе ее ядро.

Однако, даже относительно легковесная технология контейнеризации несет в себе богатый набор функций: работа с командной строкой, работа с образами, среда вы-

полнения, API для взаимодействия с различными средами выполнения. Все это неизбежно приводит к большему числу операций, которые выполняются системой для запуска контейнеров, а значит, ведет к дополнительному времени, необходимому для развертывания приложения.

Современные программные комплексы, предназначенные для контейнеризации, не являются монолитами – они включают в себя блоки, ответственные за те или иные функции. Базовым функциональным компонентом такого программного комплекса является среда выполнения. Это означает, что использование среды выполнения отдельно от остального функционала может значительно повлиять на скорость развертывания приложения. В данной статье проведен эксперимент с замерами скорости развертывания при использовании тех или иных технологий контейнеризации, а также отдельных их блоков с целью определения, насколько функциональная полнота таких систем влияет на скорость их работы.

Описание сравниваемых систем

Для проведения работ были выбраны следующие системы:

- Docker – система автоматизации разработки, развертывания и тестирования контейнеризованных приложений. Реализует запуск контейнеров с использованием ядра системы-хоста [2];
- Podman – движок для запуска контейнеров для разработки, управления и запуска OCI контейнеров в Linux [3];
- LXC – система виртуализации на уровне операционной системы для запуска контейнеров Linux [4];
- Containerd – демон для управления полным жизненным циклом контейнера на системе хоста от скачивания и хранения образов до запуска и сетевого взаимодействия [5];
- CRI-O – среда выполнения контейнеров, реализующая Container Runtime Interface – интерфейс, специально созданный для нативной поддержки OCI контейнеров внутри Kubernetes без использования Docker [6, 7];
- Runc – среда выполнения OCI контейнеров;
- Crun – среда выполнения OCI контейнеров. В отличии от Runc, написанного на языке Go, написана на языке C, что дает выигрыш в скорости.

Вначале были рассмотрены функциональные наборы каждой из систем. Всего системы сравнивались по 8 функциональным параметрам. Данные представлены в таблице.

Таблица. Сравнение функций различных систем контейнеризации

Критерии\система контейнеризации	Docker	Podman	Containerd	CRI-O	LXC	crun	runc
Полноценная клиентская утилита для работы с системой	+	+					
Размещение образов во внешнем хранилище	+	+	+				
Загрузка образов из внешнего хранилища	+	+	+	+			
Поддержка сети для создания системных интерфейсов и API для управления сетевым пространством имён контейнера	+	+	+	+			
Хранилище (на уровне хоста) для файловых систем образа и контейнера	+	+	+	+	+		
API для метрик, используемых внутри и на уровне контейнера	+	+	+	+	+		
Поддержка спецификации образов OCI	+	+	+	+	+	+	+
Создание и запуск контейнеров	+	+	+	+	+	+	+

Замеры и анализ времени запуска контейнеров

Для проведения экспериментальной части была выбрана одна метрика – время запуска контейнера. Данное время измеряется от старта команды запуска контейнера до момента, когда контейнер будет запущен. На рисунке 1 представлено сравнение времени запуска контейнеров разными средами запуска.

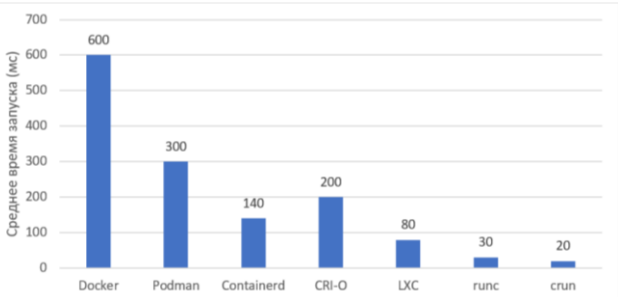

Рис. 1. Сравнение времени запуска контейнеров разными средами запуска

Сравнивая таблицу и рисунок 1, можно сделать вывод о том, что наблюдается зависимость между функциональной полнотой системы (представлены в таблице) и скоростью запуска контейнеров (рис. 1). Исключение составляет Containerd и CRI-O: при большем количестве функций Containerd опережает CRI-O в скорости. Это объясняется разницей лишь в одной функции, несущественной при запуске контейнеров и при этом более удачной программной реализацией системы Containerd. Также можно обратить внимание на существенную разницу (до двух раз) между временем запуска некоторых пар систем, имеющими сходные функциональные наборы: Docker и Podman (рис. 2), crun и runc (рис. 3).

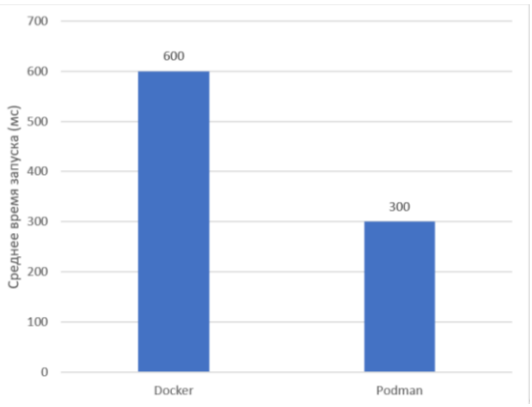

Рис. 2. Сравнение времени запуска контейнеров в Docker и Podman

Разница в скорости запуска контейнеров в Docker и Podman в 2 раза (600 миллисекунд и 300 миллисекунд на диаграмме 2 соответственно) наблюдается в связи с архитектурными особенностями данных систем. В то время как Docker использует отдельный демон – процесс, инициированный в фоновом режиме, Podman в демоне не нуждается. Это означает, что в Docker применена клиент-серверная архитектура, а Podman инициирует запуск контейнера напрямую командой пользователя, что снижает издержки на взаимодействие с демоном [8].

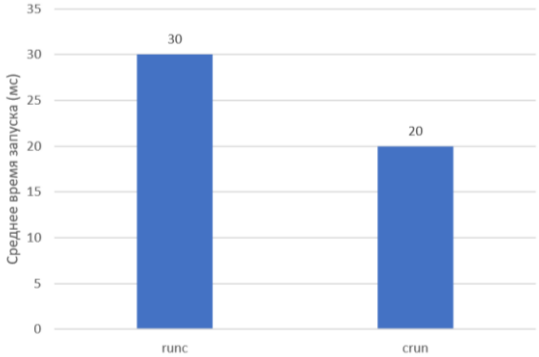

Рис. 3. Сравнение времени запуска контейнеров в runc и crun

На основании тестирования запуска контейнеров в средах runc и crun, можно заметить, что crun тратит на запуск контейнера примерно в 2 раза меньше времени. Данное наблюдение связано с рядом причин:

− различная реализация: runc написан на языке Go, в то время как crun – на языке C;

− различное потребление памяти: около 4Кб для crun и около 15Кб для runc;

− crun позволяет задать ограничение на количество создаваемых процессов, в отличие от runc.

Несмотря на представленные различия, Docker и Containerd могут использовать как crun, так и runc.

Таким образом, из данных на диаграммах 1-3, а также в таблице 1, можно сделать вывод о том, что в случаях, когда скорость запуска контейнера является для нас первоочередным параметром, имеет смысл рассматривать наименее функционально нагруженные системы. Однако, необходимо помнить, что системы, которые являются наименее функционально нагруженными, усложняют развертывание в случае выхода обновлений в метрике времени работы специалиста, занимающегося инфраструктурными задачами. Таким образом, эффективность складывается не только из времени запуска непосредственно контейнеров, но и из времени работы специалиста, что определяет целесообразность применения слабо функционально нагруженных систем в таких программных продуктах, в которых возможны высокие нагрузки и частые сбои при достаточно редких обновлениях. В то же время, в случае, когда обновления выходят часто, целесообразнее использовать менее трудозатратные способы с точки зрения доставки новых версий контейнеров.

Заключение

В работе было выдвинуто предположение о том, что функциональная полнота системы контейнеризации может существенно влиять на скорость запуска контейнеров в ней. Было проведено описание испытываемых систем, а также проведены эксперименты, подтверждающие выдвинутое предположение. В ходе экспериментов выяснилось, что количество функций системы действительно может влиять на скорость запуска. Также были выявлены дополнительные зависимости в виде архитектурных особенностей в системах, схожих по функциям. Предполагается, что применение полученных данных, поможет оптимально подбирать технологии контейнеризации, в том числе при создании более сложных систем, например автоматической генерации инфраструктуры.

Библиографический список

1. Адрова Л.С., Полежаев П.Н. Сравнительный анализ существующих технологий контейнеризации. – [Электронный ресурс]. – Режим доступа: http://elib.osu.ru/bitstream/123456789/1955/1/2473-2477.pdf

2. Adrian M. Using Docker – Sebastopol: O'Reilly Media, Inc, – 2015.

3. Podman official website https://podman.io/.

4. Konstantin I. Containerization with LXC – 2017.

5. Gabriel N. Schenker, Hideto Saito, Hui-Chuan Chloe Lee, Ke-Jou Carol Hsu. Getting Started with Containerization – 2019.

6. Denis Z, Artemii K, Aleksey U. Learn OpenShift – 2018.

7. Lennart E., Anshul J., Vladimir P., Michael G. Performance Evaluation of Container Runtimes. – [Электронный ресурс]. – Режим доступа: https://www.scitepress.org/Papers/2020/93404/93404.pdf

8. Alex G., Rute F. Podman vs Docker: What are the differences? – [Электронный ресурс]. – Режим доступа: https://www.imaginarycloud.com/blog/podman-vs-docker/

EVALUATION OF THE INFLUENCE OF THE FUNCTIONAL COMPLETENESS OF DEPLOYMENT SYSTEMS ON THE SPEED OF CONTAINER LAUNCH

K.P. Vlasov, *Graduate Student*
I.B. Tramov, *Graduate Student*
M.C. Kirpuchenko, *Graduate Student*
A.C. Saaduev, *Graduate Student*
Bauman Moscow State Technical University
(Russia, Moscow)

Abstract. *In the course of the study, the speed of the systems for deploying containerized applications was measured in order to determine the degree of influence of the functional completeness of such systems on the speed of launching containers of microservices in them. A list and a brief description of the compared systems are given. A functional comparison of these systems has been made. A comparative analysis of the data obtained during the experiments was carried out.*

Keywords: *containerization, deployment, Docker, DevOps, CI/CD.*