# Применение контейнерной виртуализации Docker для запуска задач на суперкомпьютере*

В.А. Щапов[1,2], А.В. Денисов[2], С.Р. Латыпов[1]

Институт механики сплошных сред УрО РАН[1], Пермский национальный исследовательский политехнический университет[2]

В статье рассматривается вопрос применения технологии контейнерной виртуализации Docker для запуска задач на вычислительных узлах суперкомпьютера. Доставка задач на вычислительные узлы в заранее подготовленных контейнерах позволит унифицировать окружение расчетных приложений, обеспечить независимость от установленных на узлах программ, библиотек, версий MPI и операционных систем. Это позволит снизить нагрузку на администраторов суперкомпьютера по установке и поддержанию актуальности требуемых пользователям библиотек и других зависимостей и повысит управляемость программной инфраструктуры, так как пользовательские расчетные приложения будут изолированы внутри контейнеров от основной операционной системы узлов.

*Ключевые слова:* Docker, Linux, HPC, Slurm, Контейнерная виртуализация.

## 1. Введение

Актуальной задачей при сопровождении многопользовательских или гетерогенных вычислительных систем является задача обеспечения совместимости системы с пользовательскими приложениями без возможных конфликтов в окружении. Пользовательские расчетные приложения могут требовать установки несовместимых друг с другом настроек или библиотек. Вычислительные узлы суперкомпьютеров могут иметь различные версии операционных систем, компиляторов, системных и прикладных библиотек, в том числе библиотек MPI с различными ABI (Application Binary Interface). Все это делает невозможным прямой перенос скомпилированных бинарных файлов расчетных приложений между такими вычислительными узлами.

Решением этой проблемы может быть запуск пользовательских задач в виртуальных машинах, запускаемых на вычислительных узлах, однако, технологии полной виртуализации вносят дополнительные накладные расходы, тем самым уменьшая доступную расчетным приложениям производительность [1].

В то же время, во многих операционных системах, в том числе в операционной системе Linux, есть поддержка виртуализации на уровне операционной системы или контейнерная виртуализация. Контейнерная виртуализация – это такой метод виртуализации, при котором одно ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. При этом каждый экземпляр пользовательского окружения может содержать свои версии файлов, свой набор программного обеспечения и библиотек, которые требуются расчетному приложению. Независимые тесты показывают, что использование контейнерной виртуализации в Linux практически не вносит дополнительных накладных расходов по сравнению с полной виртуализацией [1]. Это делает возможным эффективное использование контейнеров в условиях вычислительно емких задач.

## 2. Технологии контейнерной виртуализации

На данный момент существует множество технологий контейнерной виртуализации. Среди наиболее популярных и активно развивающихся решений для операционной системы Linux можно выделить OpenVZ [2], LXC [3], LXD [3] и Docker [4].

* Работа выполнена при поддержке гранта РФФИ № 16-37-00069 «Исследование и внедрение технологий контейнерной виртуализации для распределенных в пространстве суперкомпьютерных приложений».

OpenVZ является одним из первых решений контейнерной виртуализации для Linux, однако, использование этой технологии требует установки модифицированного ядра операционной системы, которое может быть несовместимо с используемым на суперкомпьютере программным и аппаратным обеспечением. В то же время LXC, LXD и Docker для разделения ресурсов и изоляции процессов используют штатную подсистему cgroups современных ядер Linux.

Технологии LXC, LXD и Docker в целом направлены на решение одного круга задач, но имеют несколько различную специфику применения. Так, LXC создавался как низкоуровневая прослойка для запуска изолированных контейнеров, управляемая высокоуровневыми системами. LXD построен на базе LXC и специализируется на запуске полноценных виртуальных машин, работающих поверх общего ядра хостовой операционной системы. Docker создавался как легковесное решение для доставки, развертывания и запуска приложений и сервисов со всеми необходимыми библиотеками в изолированном окружении.

Для исследования и внедрения была выбрана технология Docker, поскольку специфика ее применения наиболее близка к решаемой нами задаче (доставка, развертывание и запуск приложений на суперкомпьютере в изолированных контейнерах), а также по причине активной разработки технологии, наличия сообщества пользователей, хорошей документированности решения и относительной простоте освоения, по сравнению с другими технологиями.

## 3. Подготовка и запуск задач в Docker контейнерах

Разрабатываемое решение построено на базе Docker 1.7.1 и Slurm 2.6.6, работающих в операционной системе Scientific Linux 6.7, установленных на суперкомпьютере «Тритон», расположенном в ИМСС УрО РАН. На рис. 1 показана схема запуска расчетных задач в Docker контейнерах [5].

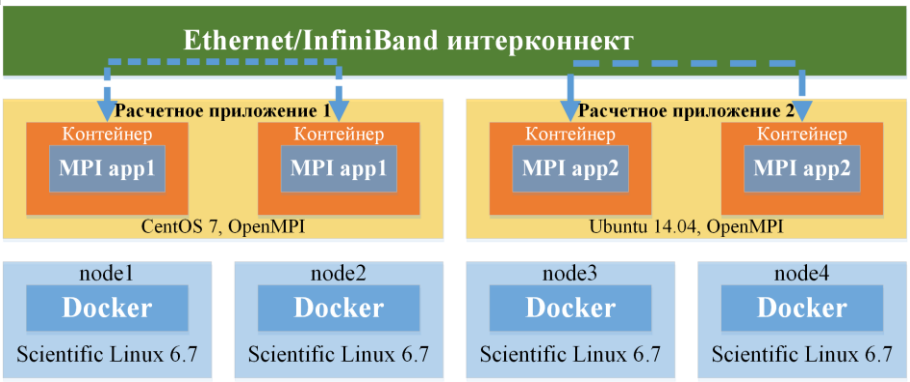


**Рис. 1**. Схема запуска расчетных задач в Docker контейнерах

Решение реализовано в виде программы, написанной на языке bash. Обобщенный алгоритм запуска расчетного приложения в контейнере представлен ниже.

1. Создание на основе пользовательского Docker-образа нового, персонифицированного образа с необходимыми настройками, пригодного для запуска на вычислительных узлах.
2. Постановка задачи на исполнение в очередь Slurm.
3. Запуск демоном Slurm промежуточного приложения на узлах. По идентификатору, присвоенному системой Slurm, определяется тип узла: ведущий (один из узлов) или ведомый.
4. В случае если текущий узел является ведомым, сформированный на первом шаге контейнер запускается в ведомом режиме (в контейнере запускается демон SSH).
5. В случае ведущего узла выполняется определение списка узлов, на которых запущена задача, ожидание запуска на ведомых узлах контейнеров со службой SSH.
6. На ведущем узле выполняется формирование команды mpirun на основе переменных окружения Slurm, и выполнятся запуск контейнера с передачей ему сформированной команды.
7. При завершении работы контейнера на ведущем узле, задачам на ведомых узлах отправляется сигнал о завершении работы.
8. Ведущий и ведомые узлы выполняют очистку временных файлов и, при необходимости, удаление контейнеров из локальных хранилищ Docker.

### 3.1 Подготовка образов пользовательских задач

Разработанное решение предполагает, что пользователь формирует Docker контейнер с вычислительной задачей и всеми необходимыми ей библиотеками. К пользовательским контейнерам предъявляются следующие требования:

- наличие внутри контейнера SSH клиента и сервера, сконфигурированного для работы с заданным портом;
- наличие исполняемого файла вычислительной задачи с наличием необходимых прав для его запуска;
- наличие реализации MPI, доступной в стандартном пути PATH.

При разработке и тестировании решения пользовательский контейнер был сформирован на основе CentOS 7, что позволяет показать возможность запуска задач на новых версиях операционных систем, если у них совместимые Kernel API/ABI.

На рис. 2 представлен код конфигурационного файла сборки Docker контейнера эталонной прикладной задачи. Приложение /opt/mpitest играло роль запускаемого расчетного MPI приложения. В тестах использовалась реализация OpenMPI.

```
FROM centos:7.2.1511
RUN yum -y install openssh-server openssh-clients
RUN sed 's@session\s*required\s*pam_loginuid.so@session
optional pam_loginuid.so@g' -i /etc/pam.d/sshd
RUN sshd-keygen
ADD files/sshd_config /etc/ssh/sshd_config
RUN yum -y install openmpi openmpi-devel
ADD files/test /opt/mpitest
RUN chmod 777 /opt/mpitest
```

**Рис. 2**. Пример Docker файла для сборки пользовательского образа

Для запуска пользовательского контейнера в условиях работающего вычислительного кластера требуется выполнить некоторые подготовительные работы для корректной его интеграции в окружения кластера. Чтобы приложение, работающее в контейнере, могло корректно читать и писать файлы в смонтированный домашний каталог пользователя, необходимо обеспечить внутри запускаемого контейнера наличие групп, в которых состоит пользователь и совпадающих по параметрам с группами на хостовой системе и пользователя с идентичным именем, идентификаторами UID и GID.

Эта задача решается путем генерации на основе исходного Docker-контейнера задачи персонифицированных для каждого запускающего контейнер пользователя образов. На рис. 3 приведена часть кода, отвечающая за генерациию конфигурационного файла сборки персонифицированного Docker контейнера (mpi_base – это название исходного контейнера с прикладной задачей).

```
echo "
    FROM mpi_base
    RUN groupadd $(id -ng) -g $(id -G)
    RUN useradd -u $(id -u) $(id -un) -G $(id -ng)
    RUN echo "$(id -un):$(id -un)" | chpasswd
"
```

**Рис. 3**. Код скрипта генерации конфигурационного файла сборки персонифицированного Docer контейнера

Сгенерированный персонифицированный образ сохраняется командой docker save во временной директории, которая доступна всем узлам суперкомпьютера для последующей загрузки на узлы.

### 3.2 Запуск задач в контейнерах с использованием системы запуска задач Slurm

На рис. 4 приведена команда постановки персонифицированного контейнера с расчетным приложением в очередь задач Slurm.

```
sbatch -N 4 --ntasks-per-node=1 --exclusive --wrap 'srun
```

```
~/tool slurm-entry'
```

**Рис. 4**. Команда постановки задачи в очередь на исполнение

Команда ~/tool является служебным приложением, выполняющим подготовку и запуск контейнеров непосредственно на вычислительных узлах.

Ключ -N задает количество узлов суперкомпьютера, выделяемых для выполнения задачи. Ключ --ntasks-per-node=1 указывает, что на каждом узле кластера должен запускаться только один процесс приложения ~/tool. Опция --exclusive определяет, что ядра вычислительного узла не должны разделяться между задачами.

Так как технология Docker используется нами не для разделения ресурсов одного узла между разными задачами, а для управления зависимостями расчетных приложений, то было принято решение запускать контейнеры из расчета один контейнер на одном вычислительном узле суперкомпьютера.

### 3.3 Запуск контейнера на вычислительных узлах

Команда ~/tool slurm-entry, запускаемая на вычислительных узлах суперкомпьютера, отвечает за выполнение подготовительной работы, запуск подготовленных персонифицированных контейнеров и запуск расчетного приложения непосредственно в контейнерах.

Процедура slurm-entry выполняет проверку переменной окружения SLURM_ID, содержащей идентификатор текущего узла. Узел с нулевым идентификатором выбирается ведущим узлом и выполняет преднастройку узлов для запуска вычислительной задачи в контейнере.

Узлы с ненулевым значением переменной окружения SLURM_ID являются ведомыми и выполняют только загрузку сохраненного в общей файловой системе персонифицированного образа контейнера, его запуск и последующую очистку – завершение работы контейнера и удаление персонифицированного образа из локального Docker хранилища образов. Команда запуска персонифицированного контейнера на ведомых узлах приведена на рис. 5. Демон SSHD, запускаемый в контейнерах, используется для запуска в них пользовательского расчетного приложения.

```
docker run --net=host                                    \
     -v /shared/home/$USER/:/home/$USER/                 \
     -v /shared/tmp/ssh/$WRAP_IMAGE/:/home/$USER/.ssh/ \
     --rm $WRAP_IMAGE /usr/sbin/sshd -D
```

**Рис. 5**. Команда запуска Docker-контейнера на ведомых узлах

Переменная WRAP_IMAGE содержит имя персонифицированного образа контейнера. Ключ --net=host указывает, что контейнер использует сетевой стек ОС, на которой он был запущен. Ключ -v используется для монтирования домашнего каталога пользователя и каталога с ключами для беспарольного доступа по ssh в контейнеры.

Ведущий узел отвечает за выполнение подготовительных операций настройки беспарольного доступа в контейнеры ведомых узлов и запуск MPI задачи в контейнере. Беспарольный доступ по SSH в контейнеры ведомых узлов необходим для того, чтобы команда mpirun, запускаемая внутри контейнера на ведущем узле, могла получать доступ в контейнеры на ведомых узлах и запустить на них процессы пользовательской задачи.

Для осуществления беспарольной авторизации в контейнеры поверх директории ~/.ssh монтируется временная директория с публичным и приватным ключом RSA: id_rsa и id_rsa.pub, а также файлом authorized_hosts, который содержит публичный ключ из id_rsa.pub. Помимо ключей, в директорию копируется файл конфигурации клиента SSH, содержащий настройки портов SSH, по которым возможен доступ в контейнеры ведомых узлов. Пример конфигурационного файла для клиентов SSH приведен на рис. 6.

```
Host node1
    HostName node1
    Port 2222
Host node2
    HostName node2
    Port 2222
```

**Рис. 6**. Пример конфигурационного файла SSH-клиента

Далее ведущий узел на основе переменной окружения SLURM_JOB_NODELIST получает список узлов, выделенных для задачи, и ожидает запуска на них контейнеров с сервисом SSH. Также на этом шаге происходит формирование конфигурационного файла known_hosts. Код, выполняющий описанные действия, приведен на рис. 7.

```
for n in $nodes; do
    until nc -zv $n $SSH_PORT > /dev/null 2>&1; do
        sleep 0.2
    done
    ssh-keyscan -p $SSH_PORT $n | sed
"s/$n/[$n]:$SSH_PORT/g" >>
/shared/tmp/ssh/$WRAP_IMAGE/known_hosts
done
```

**Рис. 7**. Код генерации конфигурационного файла known_hosts

После этого выполняется загрузка сохраненного в общей файловой системе персонифицированного образа контейнера. Когда образ загружен и доступен docker-демону для запуска, выполняется формирование команды MPI для запуска в контейнере и запуск контейнера на ведущем узле.

На рис. 8 приведен код формирования команды запуска MPI-приложения и команды запуска персонифицированного контейнера на ведущем узле.

```
MPIRUN_CMD="mpirun -np $(( $SLURM_NNODES *
$SLURM_CPUS_ON_NODE )) --map-by
ppr:$SLURM_CPUS_ON_NODE:node -H localhost,$MPIRUN_HOSTS
$MPI_CMD"
HOST_CMD="(source /etc/profile && source ~/.bashrc;
$MPIRUN_CMD)"
docker run --net=host                                    \
     -v /shared/home/$USER/:/home/$USER/                 \
     -v /shared/tmp/ssh/$WRAP_IMAGE/:/home/$USER/.ssh/ \
     -u $(id -un) $WRAP_IMAGE bash -c "$HOST_CMD"
scancel -s INT $SLURM_JOBID
```

**Рис. 8**. Код формирования команды запуска MPI-приложения и контейнера

Команда mpirun формируется на основе следующих переменных окружения SLURM:

- SLURM_NNODES (количество выделенных вычислительных узлов);
- SLURM_CUPS_ON_NODE (количество ядер на вычислительном узле).

Так как на суперкомпьютере «Тритон» на всех вычислительных узлах одинаковое количество процессорных ядер, то запущенная таким образом задача будет использовать все доступные ядра вычислительных узлов.

Ключ --map-by ppr:$SLURM_CPUS_ON_NODE:node указывает количество процессов, которые будут запущенны на одном узле. Список узлов содержится в переменной $MPIRUN_HOSTS. Переменная MPI_CMD содержит путь до исполняемого файла, который будет исполнен внутри контейнера.

После завершения работы вычислительной программы и остановки контейнера подчиненным узлам посылается сигнал прерывания работы SIGINT, что вызывает завершение работы контейнеров на этих узлах и завершение задачи с точки зрения Slurm.

Последним этапом, который выполняет ведущий узел, является удаление персонифицированного Docker-образа и сгенерированных временных файлов.

## 4. Тестирование

Для проверки разработанного решения нами был использован тест OSU Broadcast Latency, работающий поверх контейнеров на базе CentOS 7 с использованием Ethernet интерконнекта. Результаты тестирования показаны на рис. 9.

```
# OSU Broadcast Latency Test v3.1.1
# Size            Latency (us)
```

| | |
|---|---|
| 1 | 129.11 |
| 2 | 128.13 |
| 4 | 132.13 |
| 8 | 134.17 |
| 16 | 117.72 |
| 32 | 125.39 |
| 64 | 125.23 |
| 128 | 126.39 |
| 256 | 127.71 |
| 512 | 141.87 |
| 1024 | 201.56 |
| 2048 | 224.07 |
| 4096 | 295.82 |
| 8192 | 501.13 |
| 16384 | 850.49 |

**Рис. 9**. Результат теста mpitests-osu_bcast, запущенного в Docker контейнерах с использованием Ethernet интерконнекта

Результаты тестирования показывают возможность использования технологии Docker для запуска вычислительных задач на суперкомпьютере в изолированных контейнерах.

## 5. Заключение

В работе показана возможность использования технологии Docker для доставки, развертывания и запуска вычислительных задач на суперкомпьютере в изолированных контейнерах с использованием немодифицированной системы запуска задач. Описан прототип интеграции контейнеров и системы SLURM. Ключевой особенностью решения является запуск в контейнерах только вычислительной задачи, в то время как инфраструктура запуска задач остается снаружи на вычислительных узлах. Такой подход позволит поставщикам программного обеспечения поставлять приложения в виде готовых к работе, сертифицированных контейнеров, которые не зависят от окружения суперкомпьютера и используемой им системы запуска задач.

В дальнейшем планируется интеграция общего для всех вычислительных узлов репозитория пользовательских Docker-образов для минимизации объема передаваемых данных между узлами за счет использования механизма слоев; реализация возможности приостановить задачу командами slurm suspend и slurm resume; добавление поддержки InfiniBand интерконнекта; реализация пользовательского интерфейса для удобного запуска контейнеров пользователями кластера. Также необходимо проанализировать и решить проблему разделения прав доступа, чтобы у пользователей не было возможности поднять свои привилегии на вычислительных узлах путем использования специально подготовленных контейнеров.

## Литература

1. W. Felter, A. Ferreira, R. Rajamony and J. Rubio An updated performance comparison of virtual machines and Linux containers // IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, pp. 171-172.
2. OpenVZ Virtuozzo Containers Wiki. URL: https://openvz.org/ (дата обращения: 26.05.2016).
3. Linux Containers. URL: https://linuxcontainers.org/ (дата обращения: 26.05.2016).
4. Docker - Build, Ship, and Run Any App, Anywhere. URL: https://www.docker.com/ (дата обращения: 26.05.2016).
5. V. Shchapov, D. Chugunov Using of container virtualization to run tasks on a distributed supercomputer // CEUR Workshop Proceedings (Proceedings of the 1st Russian Conference on Supercomputing (RuSCDays 2015), Moscow, Russia, September 28-29, 2015). Vol. 1482. 2015. P. 601.

# Using Docker container virtualization to run tasks on supercomputer nodes

V.A. Shchapov[1,2], A.V. Denisov[2], S.R. Latypov[1]

Institute of Continuous Media Mechanics of the Ural Branch of Russian Academy of Science[1], Perm National Research Polytechnic University[2]

The paper focuses on the use of Docker container virtualization techniques for running tasks on supercomputer nodes. Data delivery in the previously prepared containers to computing nodes makes it possible to unify environments for computational applications and to ensure the independence of computational applications from codes, libraries, MPI versions and operational systems installed on nodes. The proposed approach reduces the burden on supercomputer administrators associated with installation and maintenance of the relevance of libraries requested by users and other dependencies, and improves the manageability of software infrastructure because custom computational applications are isolated inside the containers from the main operating subsystem.

*Keywords:* Docker, Linux, HPC, Slurm, Container virtualization.

## References

1. W. Felter, A. Ferreira, R. Rajamony and J. Rubio An updated performance comparison of virtual machines and Linux containers // IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, pp. 171-172.
2. OpenVZ Virtuozzo Containers Wiki. URL: https://openvz.org/ (accessed: 26.05.2016).
3. Linux Containers. URL: https://linuxcontainers.org/ (accessed: 26.05.2016).
4. Docker - Build, Ship, and Run Any App, Anywhere. URL: https://www.docker.com/ (accessed: 26.05.2016).
5. V. Shchapov, D. Chugunov Using of container virtualization to run tasks on a distributed supercomputer // CEUR Workshop Proceedings (Proceedings of the 1st Russian Conference on Supercomputing (RuSCDays 2015), Moscow, Russia, September 28-29, 2015). Vol. 1482. 2015. P. 601.