Выбор среды исполнения контейнеров для безопасного запуска внешних расчётных компонентов системы моделирования обращения с жидкими радиоактивными средами АЭС

В.Е. Мельников, Ю.А. Андриенко

***Аннотация* — В работе рассматривается разработка модуля подключения внешних расчётных компонент (пользовательских скриптов) для автоматизированной системы цифрового моделирования производственно-технологических цепочек обращения с жидкими радиоактивными отходами АЭС, основной задачей которого является надежное и безопасное исполнение недоверенного кода, подготовленного пользователями системы. Модуль реализован на платформе .NET 8.0 с использованием контейнеризации Docker. Для достижения оптимальных показателей производительности и надежности модуля было проведено комплексное сравнительное исследование четырёх сред исполнения контейнеров: runc, sysbox-runc, gVisor (runsc) и Kata Containers (с гипервизорами QEMU и Cloud Hypervisor). Оценка проводилась по ключевым параметрам: время старта контейнера (включая влияние плотности размещения), потребление процессорного времени и оперативной памяти, производительность работы с файловой системой, а также степень изоляции. По результатам экспериментов gVisor продемонстрировал наиболее сбалансированное сочетание усиленной границы изоляции, устойчивости к взаимному влиянию контейнеров и умеренных накладных расходов. Опыт и выводы, полученные при разработке и внедрении модуля могут быть использованы в других инженерных системах, требующих безопасного выполнения динамически загружаемого или автоматически генерируемого кода.**

***Ключевые слова*—цифровое моделирование, изоляция, контейнеризация, runc, sysbox-runc, gVisor, Kata Containers, QEMU, Cloud Hypervisor.**

I. ВВЕДЕНИЕ

Развитие облачных вычислений и инструментов искусственного интеллекта, предназначенных для генерации кода, привело к значительному росту популярности парадигмы «функция как услуга» (Function-as-a-Service, FaaS). Данные платформы всё чаще применяются для выполнения кода, степень доверия к которому со стороны разработчика или конечного пользователя может быть низкой. [1, 2]

Недоверенный код возникает в разнообразных сценариях: тестирование пользовательского кода в образовательных платформах и соревнованиях [3], исполнение LLM-генерируемых скриптов в AI-агентах [4], запуск плагинов и библиотек из открытых репозиториев, обработка мобильного кода в браузерах и документах. Такие сценарии приводят к рискам, включая утечку чувствительной информации (системные данные, переменные окружения, ключи), манипуляцию файловой системой (чтение, запись, удаление, изменение привилегий), несанкционированный доступ к внешней сети и попытки выхода из изолированной среды. [5, 6]

В подобных сценариях критически важной становится задача обеспечения безопасности и изоляции: провайдеру платформы необходимо минимизировать потенциальное негативное влияние недоверенного кода на хостовую систему, соседние процессы и инфраструктуру в целом.

В рамках разработки собственной автоматизированной системы «ПТК АСНИ» предназначенной для цифрового моделирования производственно-технологических цепочек (ПТЦ) обращения с жидкими радиоактивными отходами и средами АЭС, мы столкнулись с аналогичной проблемой запуска недоверенного кода — в частности, при интеграции внешних моделей, пользовательских скриптов для описания технологических переделов и динамического исполнения математических операторов в модуле моделирования. [7]

Классические подходы к решению проблемы включают sandboxing на уровне процессов, контейнеризацию и виртуализацию. [8, 9] Однако данные подходы могут существенно различаться по множеству параметров: накладные расходы потребления памяти и процессорного времени, скорости исполнения скриптов, времени старта, глубиной изоляции и т.д. [10]

Данная исследовательская работа посвящена созданию модуля подключения внешних расчетных компонент, в рамках развития «ПТК АСНИ».

II. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

«ПТК АСНИ» позволяет создавать и рассчитывать математические модели процессов очистки и переработки жидких радиоактивных отходов. [11, 12] При создании моделей выделяются следующие сущности.

Технологический поток представляет собой совокупность материальных объектов (жидких, твердых и газообразных фаз), характеризуемых набором параметров, включающих расход, химический состав,

радионуклидный состав, физико-химические свойства и показатели активности.

Технологический поток представляется в виде вектора параметров:

$$x = \{x_1, x_2, \dots, x_n\},$$

где компоненты вектора описывают физико-химические и радиологические характеристики потока.

Технологический передел определяется как элементарное преобразование технологического потока, реализующее изменение его параметров в результате физико-химического воздействия. Передел соответствует отдельной стадии технологического процесса и характеризуется набором входных и выходных потоков.

Технологический передел рассматривается как оператор преобразования:

$$\boldsymbol{y} = F(\boldsymbol{x}, \boldsymbol{p})$$

где **x**— входной поток, **y**— выходной поток, **p**— набор параметров процесса, а F— оператор, описывающий физико-химические процессы преобразования.

Производственно-технологическая цепочка рассматривается как упорядоченная совокупность технологических переделов, связанных между собой потоками. ПТЦ определяет полную структуру технологии обращения с жидкими радиоактивными отходами и отражает последовательность преобразований от исходного сырья до конечного продукта.

Производственно-технологическая цепочка представляется в виде ориентированного графа:

$$G = (V, E)$$

где множество вершин V соответствует технологическим переделам, а множество дуг E— технологическим потокам, связывающим переделы между собой.

Во время разработки возникла потребность реализовать программный модуль, который позволит пользователям системы загружать и запускать расчётные компоненты, которые бы описывали математическую модель оператора передела.

III. АРХИТЕКТУРА МОДУЛЯ

A. Общая структура модуля

«ПТК АСНИ» имеет модульную архитектуру, основным стеком разработки которой является .NET 8.0, поэтому для модуля была выбрана та же платформа.

Основной вычислительной единицей в модуле являются легковесные скрипты, которые должны быстро и в большом количестве выполняться во время расчета схемы ПТЦ, полноценная виртуализация оказалась бы слишком медленным решением для этой проблемы, поэтому в качестве основного механизма для изоляции была выбрана технология контейнеризации Docker. [13]

Основная часть данных «ПТК АСНИ» хранится в файловой системе хоста, поэтому она же была выбрана в качестве места для хранения загруженных скриптов.

На рисунке 1, представлена архитектура модуля системы:

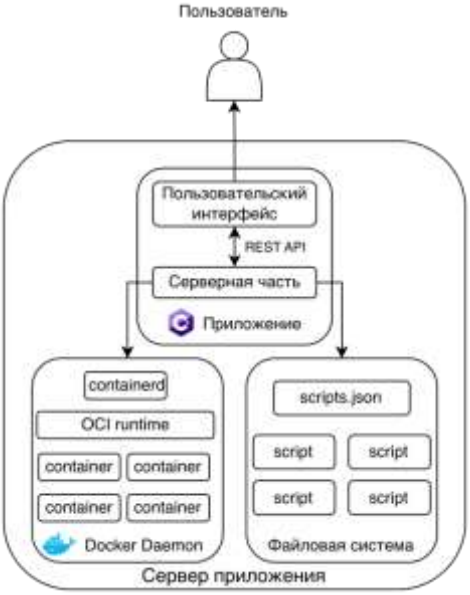

Рис. 1 — архитектура приложения

В архитектуре модуля для запуска всех скриптов используется один-единственный Docker образ. Базовый образ содержит интерпретатор исполнения скрипта, и набор предустановленных библиотек, необходимых для типовых расчётов моделей «ПТК АСНИ».

Если пользователям требуются дополнительные зависимости, после одобрения администратора системы, в образ включается необходимая библиотека.

B. Жизненный цикл расчётной компоненты

Типовой сценарий создания или изменения скрипта включает передачу кода программы на сервер, выполнение валидации зависимостей (наличия неустановленных библиотек) и сохранение файла скрипта в системе. Поскольку образ при этом не пересобирается, пользователь получает возможность быстро повторять цикл «правка — проверка — запуск», что существенно для исследовательской работы с моделью.

Диаграммы основных операций приведены на рис. 2:

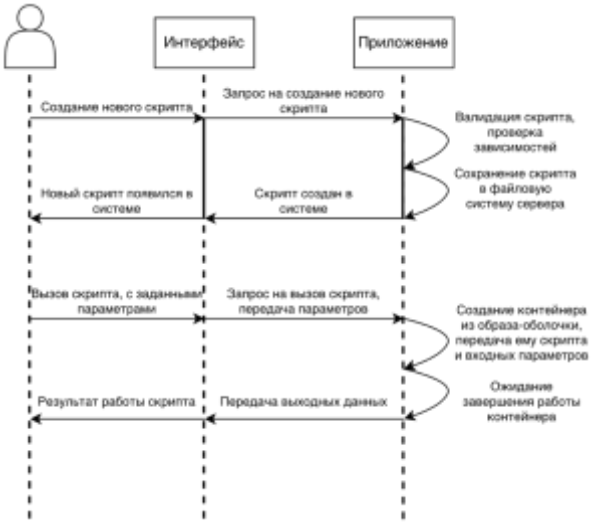

Рис. 2 — диаграмма обработки данных в системе

Сценарий исполнения строится вокруг короткоживущих контейнеров. Сервер формирует

команду запуска таким образом, чтобы файл скрипта был доступен внутри контейнера через bind mount (монтирование выбранного файла с хоста в изолированную среду). Проверенные входные данные передаются в контейнер через явно сформированный набор переменных окружения; ограничение их объёма и формата обеспечивается серверной валидацией, а не самим механизмом передачи. При запуске на контейнер накладываются ограничения на время работы и использование оперативной памяти. По завершении работы контейнер уничтожается, что снижает риск накопления посторонних файлов и процессов в системе и приближает модель исполнения к принципу «один запуск — одна изолированная сессия».

C. Модель угроз

Пользователями модуля являются преимущественно предметные специалисты - инженеры и исследователи, обладающие глубокими знаниями физико-химических процессов, но не всегда имеющие профессиональную подготовку в области разработки безопасного программного обеспечения. Кроме того, программный код может создаваться с применением генеративных моделей. Поэтому загружаемые расчётные компоненты должны рассматриваться системой как потенциально недоверенные независимо от квалификации и намерений их авторов.

Защищаемыми активами являются:

- файловая система хоста и хранящиеся в ней данные моделей «ПТК АСНИ», включая расчётные компоненты других пользователей;
- конфигурация серверного приложения, учётные данные и ключи доступа;
- вычислительные ресурсы хоста и, как следствие, доступность и предсказуемость времени расчёта схемы ПТЦ;

Из приведённой модели угроз следуют два независимых по природе требования к среде исполнения. Первое — глубина границы безопасности, то есть характер разделения между контейнером и ядром хоста, определяющий устойчивость к выходу за границу изоляции; оно оценивается архитектурно, по типу используемой границы. Второе — производительная изоляция, то есть степень взаимного влияния одновременно выполняемых контейнеров через общие вычислительные ресурсы; она поддаётся количественному измерению. Далее эти два свойства рассматриваются раздельно.

D. Требования к среде исполнения контейнеров

Параметры среды, в которой исполняются пользовательские скрипты, определяются выбранной средой исполнения контейнеров. Различные среды исполнения реализуют границу между контейнером и ядром хоста принципиально по-разному — от штатных механизмов ядра Linux до перехвата системных вызовов в пространстве пользователя и запуска отдельного гостевого ядра под управлением гипервизора.

Каждый дополнительный уровень изоляции отражается на параметрах производительности.

Исходя из этого, а также из принятой модели угроз и характера решаемых задач — короткоживущие контейнеры, частые запуски, небольшие расчётные скрипты, работа с внешними файлами и необходимость ограничения взаимного влияния расчётов — сформулированы требования к среде исполнения контейнеров:

- Минимальное время создания контейнера, сохраняющееся при росте числа одновременно запускаемых контейнеров
- Потребление процессорного времени, сопоставимое с исполнением без дополнительной изоляции
- Умеренное потребление оперативной памяти на один контейнер
- Приемлемая производительность файловых операций при доступе к смонтированному каталогу хоста
- Граница изоляции строже, чем штатные механизмы ядра Linux при общем с хостом ядре
- Устойчивость характеристик контейнера к нагрузке, создаваемой одновременно выполняемыми контейнерами

Для выбора среды исполнения, которая будет лучше всего соответствовать приведенным выше требованиям необходимо провести качественный сравнительный анализ, описание которого представлено в разделе IV.

IV. ОЦЕНКА ЭФФЕКТИВНОСТИ СРЕД ИСПОЛНЕНИЯ КОНТЕЙНЕРОВ

Сформулированные в разделе III требования определяют условия отбора сравниваемых сред и набор измеряемых величин.

Рассматривались среды, совместимые со спецификацией OCI [14], подключаемые к используемому стеку Docker в качестве альтернативной среды исполнения и поддерживающие монтирование каталогов хоста и ограничения ресурсов; выполнение этих условий является предпосылкой применимости среды и далее в качестве критерия сравнения не используется.

Для оценки эффективности были выбраны следующие среды исполнения контейнеров:

- runc (1.3.4) — стандартная среда выполнения контейнеров, входящая в состав Docker и реализующая минимальную изоляцию на основе Linux namespaces и cgroups;
- sysbox-runc (0.6.6) — расширение runc от проекта Nestybox, обеспечивающее усиленную изоляцию за счёт дополнительных механизмов [15]
- runsc (gVisor) (20260323.0) технология Google, использующая виртуальное ядро (user-space kernel) для перехвата системных вызовов и значительного повышения уровня изоляции; [17]
- Kata Containers (3.27.0) (с гипервизорами QEMU (10.2.0) и Cloud Hypervisor (50.1)) —решение, запускающее каждый контейнер внутри облегчённой виртуальной машины, обеспечивающее изоляцию на уровне аппаратной виртуализации. [18, 19]

Выбор данных технологий обусловлен их упором на дополнительную изоляцию и доступную интеграцию с платформой Docker.

A. Условия и повторяемость эксперимента

Все испытания проводились на физической машине, поддерживающей аппаратную виртуализацию, со следующими характеристиками:

Процессор Intel Core i5-8400 (6 ядер, 2.8 ГГц)
16 ГБ ОЗУ
Накопитель: жесткий диск HDD 1 ТБ
ОС Ubuntu 24.10 с файловой системой ext4
Docker 29.3.0 с containerd v2.2.1

Для всех сред исполнения применялся единый базовый образ на основе python:3.11-alpine с интерпретатором Python 3.11.9.

B. Методы измерения

Для сбора метрик на хосте использовался следующий программный стек:

- Node Exporter 1.11.2 — сбор основных метрик хоста (загрузка CPU, использование оперативной памяти);
- cAdvisor v0.55.1 — сбор детальных метрик по каждому контейнеру;
- Prometheus v2.55.1 — сбор и хранение временных рядов метрик;
- Grafana 11.3.0 — визуализация и анализ полученных данных.

Нагрузочное тестирование осуществлялось с помощью утилиты k6 v1.7.1, позволяющей имитировать нагрузку на сервер. Все тесты проводились по следующему сценарию:

Рис. 3 — диаграмма последовательности выполнения теста

Каждый тест длился 20 минут и повторялся 5 раз; приводимые значения представляют собой среднее по выборке. Перед каждой серией измерений выполнялся прогрев в 1 минуту.

C. Время старта контейнера

Этот раздел посвящён оценке времени запуска контейнера и влиянию плотности размещения (количества одновременно запущенных контейнеров на одном хосте) на скорость старта.

Методика эксперимента заключается в следующем: сначала измеряется время запуска одного контейнера в течение длительного периода, после чего вычисляется среднее значение. Далее постепенно увеличивается количество одновременно запускаемых контейнеров, чтобы определить, как масштабирование плотности влияет на время старта.

Поскольку модуль управляет полным жизненным циклом контейнера, время старта измерялось непосредственно в нём — как интервал между отправкой команды создания и готовностью контейнера принять расчётный скрипт.

runc использует нативные механизмы Linux.

sysbox-runc является расширенной версией runc и добавляет дополнительные уровни изоляции.

gVisor можно рассматривать как совокупность гостевого ядра (Sentry) и вспомогательного процесса (Gofer), что требует дополнительных затрат на инициализацию user-space kernel [16].

Kata Containers представляет собой высокооптимизированную виртуальную машину со встроенным движком для запуска контейнеров, которая может работать с различными гипервизорами (QEMU, Cloud Hypervisor). Виртуальные машины требуют запуска целой операционной системы перед обработкой полезной нагрузки, что может существенно увеличить время старта.

Мы ожидаем, что время запуска runc, sysbox-runc и контейнеров gVisor будет относительно близким и значительно меньшим, чем у Kata Containers.

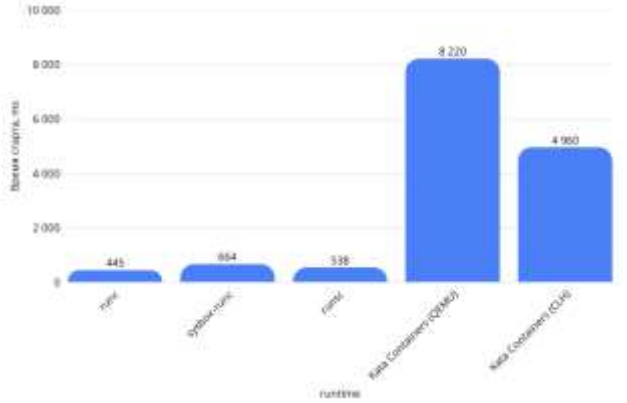

Рис. 4 — время старта различных runtime

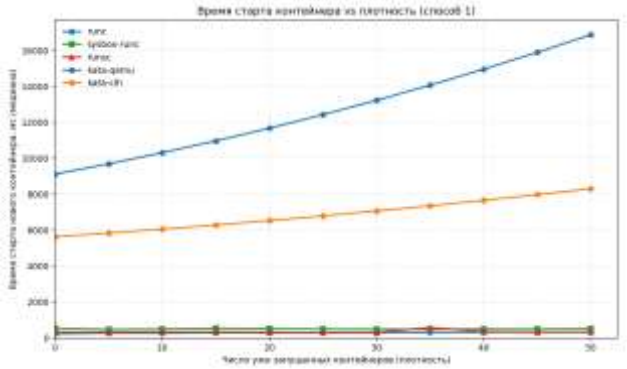

Рис. 5 — график зависимости времени старта от плотности контейнеров на хосте

Наименьшее время запуска у контейнера runc (445 миллисекунд). Время запуска контейнера gVisor на 20,8% больше, чем runc. На втором месте идёт sysbox-runc (664 миллисекунды). Наибольшее время создания контейнера наблюдается у Kata Containers (5360–8400 мс), что обусловлено необходимостью запуска

облегчённой виртуальной машины. При этом вариант с гипервизором Cloud Hypervisor демонстрирует заметно лучшие результаты по сравнению с QEMU.

По сравнению с рис. 4, время запуска Kata Containers (QEMU и CLH) на рис. 5 увеличилось на 85,6% и 47,4% соответственно.

D. Потребление процессорного времени

Мы протестировали вычислительную производительность рантаймов runc, sysbox-runc, gVisor и Kata Containers. Для этих целей был разработан скрипт, создающий продолжительную вычислительную нагрузку на сервер. Данный скрипт генерирует в цикле случайную строку и последовательно вычисляет её хеш-функцию (SHA-256).

Такой подход создаёт устойчивую высокую нагрузку на CPU, при этом практически не затрагивая операции дискового ввода-вывода. В качестве метрики производительности использовалось время выполнения скрипта и загруженность CPU.

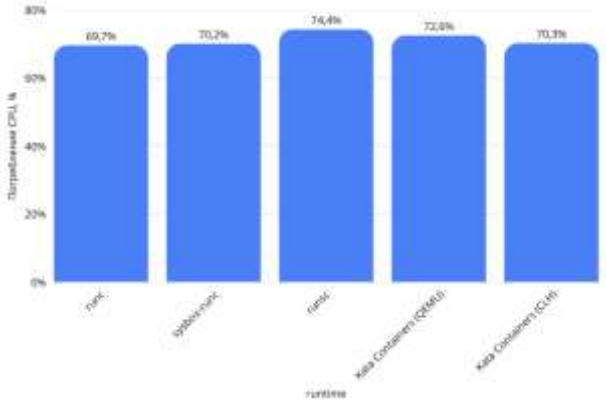

Рис. 6 — потребление процессорного времени у различных runtime.

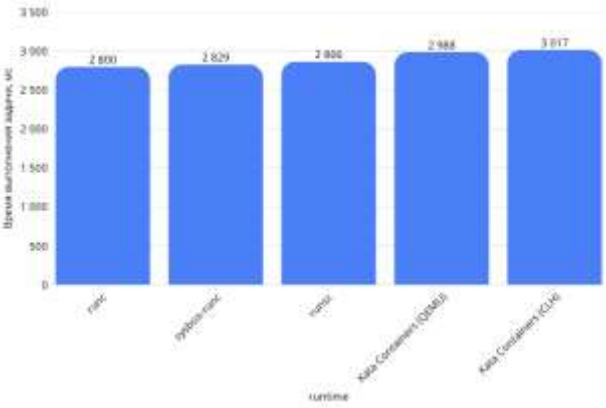

Рис. 7 — время выполнения скрипта у различных runtime

Анализ результатов показывает, что время выполнения скрипта во всех рассматриваемых средах находится в сопоставимых пределах. Наименьшее время демонстрирует runc (2800 мс), в то время как Kata Containers (как с QEMU, так и с Cloud Hypervisor) показывают увеличение на 6–8 %. Данное различие объясняется дополнительными издержками на изоляцию, вносимыми виртуализацией (в случае Kata) и перехватом системных вызовов (в случае gVisor).

Потребление процессорного времени контейнерами во всех средах оказалось примерно одинаковым (69,7–74,4 %).

E. Потребление оперативной памяти

В этой секции проводилось сравнение потребляемой памяти контейнерами при минимальной нагрузке.

runc и sysbox-runc используют нативные механизмы ядра Linux и имеют минимальный объем контейнера, так как практически не добавляют дополнительных слоёв абстракции.

gVisor использует пользовательское виртуальное ядро (Sentry), которое выполняет перехват и обработку системных вызовов, что приводит к заметному увеличению потребления памяти по сравнению с классическими контейнерами.

Kata Containers запускает полноценную виртуальную машину с собственной гостевой операционной системой, что закономерно вызывает наибольшее потребление по оперативной памяти.

Были получены следующие результаты.

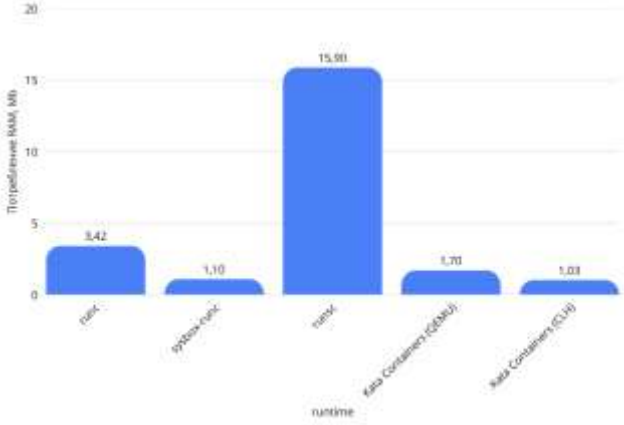

Рис. 8 — потребление памяти различными контейнерами

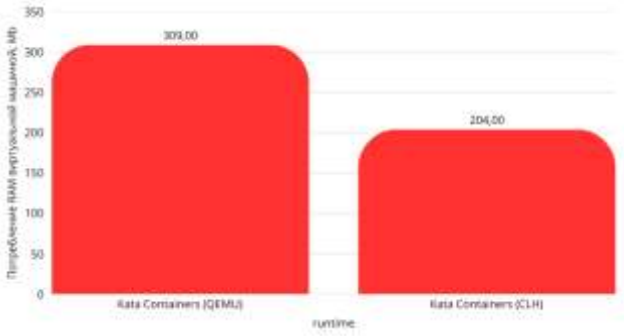

Рис. 9 — потребление памяти виртуальной машиной Kata Containers с разными гипервизорами

Среди сред, не использующих виртуализацию, выделяется runsc (gVisor), потребляющий существенно больший объём оперативной памяти (15,9 МБ против 1,1–3,4 МБ у остальных). Это объясняется наличием пользовательского ядра (Sentry), которое обеспечивает глубокую изоляцию за счёт перехвата системных вызовов. У Kata Containers потребление виртуальной памяти разделяется на память контейнера и память виртуальной машины. Видно, что Cloud Hypervisor позволяет создавать более легковесную (204 против 309 МВ) виртуальную машину.

F. Работа с файлами и с файловой системой

Для виртуализации на основе аппаратного абстрагирования ввод/вывод всегда оставался одним из

наиболее узких мест. Различные технологии контейнеризации и лёгкой виртуализации реализуют доступ к файловой системе принципиально по-разному, что напрямую влияет на их производительность при работе с диском.

runc (включая sysbox-runc) работает максимально близко к хостовому ядру и может практически напрямую обращаться к файловой системе хоста, поэтому следует ожидать минимальные потери производительности операций ввода-вывода.

gVisor реализует файловую систему контейнера через отдельный user-space процесс Gofer, использующий протокол 9P, что неизбежно приводит к дополнительным накладным расходам на дисковые операции.

Kata Containers с гипервизорами QEMU и Cloud Hypervisor представляет собой полноценную виртуальную машину, поэтому ожидается наиболее существенное снижение производительности файлового ввода-вывода по сравнению с классическими контейнерами.

Далее мы подтвердим приведённые выше теоретические ожидания результатами проведённых экспериментов.

В первом эксперименте в каждом контейнере создавался временной файл и последовательно (блоками по 1 МБ) выполнялись операции чтения и записи.

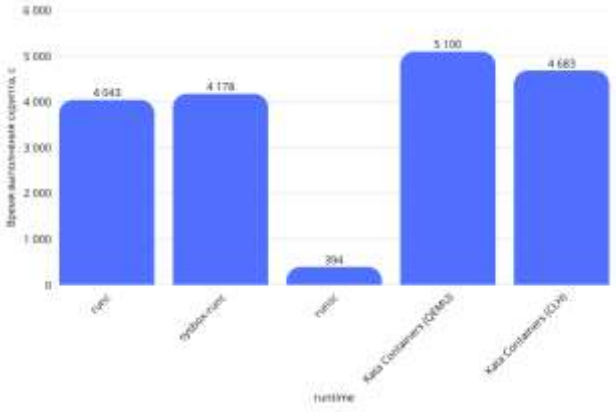

Рис. 10 — время выполнения скрипта с различными рантаймами

Результаты теста нас удивили, по всем параметрам (скорость выполнения, потребление CPU и RAM) среда выполнения runsc (gVisor) значительно опережала другие среды. Оказалось, что благодаря тому, что gVisor использует виртуальное пользовательское ядро, он реализует собственную модель файловой системы. Благодаря этому часть операций внутри контейнера может буферизироваться и оптимизироваться на уровне приложения, что приводит к сокращению количества реальных обращений к хостовому диску.

Для обеспечения корректного и справедливого сравнения эксперимент был повторён с монтированием внешней директории хоста в контейнер, в которой создавался и использовался временный файл.

После использования bind mount поведение runsc (gVisor) изменилось принципиальным образом. Данные показатели стали сопоставимы по порядку величины с результатами других сред исполнения.

В целом, при работе с внешней файловой системой разброс времени выполнения скрипта между средами не превышает примерно 35 %. Наиболее быстрые результаты показали sysbox-runc и runc. Самое большое время получилось у Kata Containers с гипервизором Cloud Hypervisor.

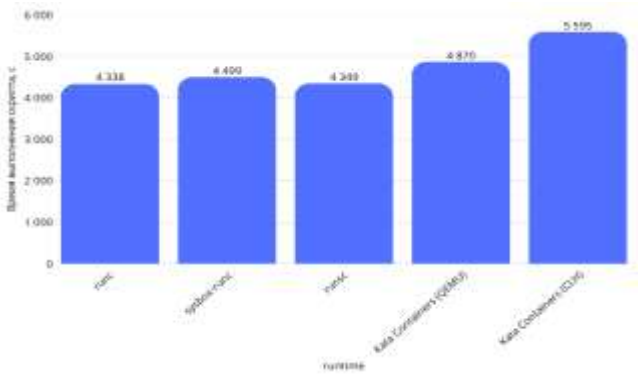

Рис. 11 — время выполнения скрипта, при условии, что в контейнер монтируется внешняя директория

Полученные результаты подтверждают, что характеристики производительности gVisor при операциях ввода-вывода в значительной степени зависят от типа используемого хранилища (внутренняя файловая система контейнера или bind mount с хоста).

G. Граница безопасности сред исполнения

Требование к границе изоляции нельзя проверить прямым измерением на стенде. Устойчивость к выходу за границу изоляции определяется наличием уязвимостей в конкретных реализациях и версиях, а не воспроизводимой реакцией среды на нагрузку. Поэтому соответствие требованию оценивалось качественно — по степени виртуализации.

Рассматриваемые среды исполнения реализуют эти подходы следующим образом:

runc использует штатную изоляцию Linux на основе пространств имён и контрольных групп, дополняемую профилем seccomp и отзывом capabilities. Ядро является общим с хостом, а доступный контейнеру интерфейс системных вызовов сокращается лишь в той мере, в какой это задано профилем seccomp.

sysbox-runc расширяет runc: пользовательское пространство имён включается по умолчанию, часть псевдофайловых систем виртуализируется, отдельные системные вызовы перехватываются и обрабатываются вспомогательным процессом. Это снижает как долю операций, выполняемых с привилегиями, так и объём доступной контейнеру информации о хосте, однако ядро по-прежнему остаётся общим [15].

gVisor обрабатывает системные вызовы контейнера в ядре, реализованном в пространстве пользователя (Sentry); файловые операции обслуживаются отдельным процессом (Gofer). Обращения к ядру хоста выполняет только Sentry, причём набор используемых им вызовов ограничен собственным профилем seccomp, что существенно сокращает поверхность атаки на ядро хоста [17].

Kata Containers исполняет контейнер внутри облегчённой виртуальной машины с собственным гостевым ядром, поэтому взаимодействие с хостом сводится к интерфейсу гипервизора и модели виртуальных устройств. Конфигурация с Cloud

Hypervisor отличается сокращённой моделью устройств по сравнению с QEMU [18, 19].

Из сопоставления следует, что требованию по изоляции не удовлетворяет runc: граница изоляции образована штатными механизмами ядра, которое остаётся общим с хостом, а поверхность доступных системных вызовов практически не сокращается. Среда sysbox-runc удовлетворяет требованию частично — граница усилена, но сохраняется зависимость от общего ядра. Требованию отвечают gVisor и обе конфигурации Kata Containers, различающиеся тем, вынесено ли обрабатывающее вызовы ядро в пространство пользователя или в виртуальную машину.

Приведённое сопоставление отражает архитектурный принцип разделения, а не отсутствие уязвимостей: случаи выхода за границу изоляции известны как для сред с общим ядром [6], так и, существенно реже, для решений с ядром в пространстве пользователя и для гипервизоров. Углубление границы не устраняет риск полностью, но сокращает поверхность атаки и ограничивает последствия успешной эксплуатации.

H. Производительная изоляция

Система считается производительно-изолированной, если для клиентов, работающих в пределах своих квот, производительность не ухудшается, когда другие клиенты превышают свои квоты [20]. Контейнерная система в основном состоит из двух типов контейнеров: затронутых контейнеров и перегруженных контейнеров. Если нагрузка контейнера ненормально растёт и влияет на производительность других контейнеров, мы называем этот контейнер перегруженным.

Для оценки производительной изоляции между контейнерами мы использовали подход, основанный на измерении влияния «шумного соседа» («noisy neighbor»).

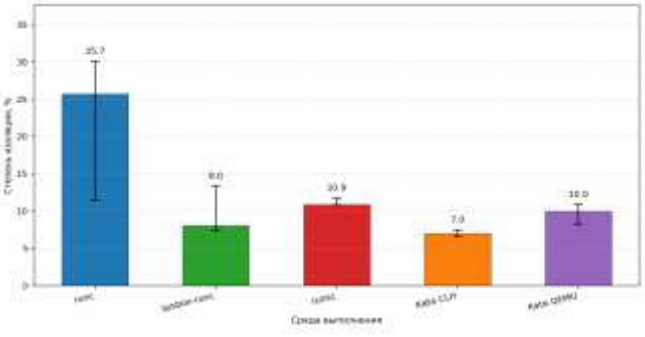

Рис. 12 — степень потери изоляции контейнера у различных рантаймов

В рамках испытания сначала запускается «чистый» контейнер, в котором измеряется базовая производительность и использование ресурсов (CPU, память) при выполнении контрольной нагрузки. Затем параллельно запускается второй контейнер («агрессор»), в котором выполняется интенсивная вычислительная нагрузка. Одновременно с помощью инструментов мониторинга фиксируются показатели производительности и потребления ресурсов обоих контейнеров.

Степень изоляции оценивается по величине деградации характеристик «чистого» контейнера под воздействием «агрессора». Чем меньше влияние шумного соседа на производительность и стабильность основного контейнера, тем выше уровень изоляции.

Были получены следующие результаты:

Сравнение проводилось по формуле

$$I = \frac{|p_i - p_j|}{p_i},$$

где pi, pj — это производительность контейнера до и после вмешательства контейнера-агрессора. Чем выше коэффициент потери производительности, тем хуже изоляция производительности контейнера. [21]

Величина каждого столбца на графике (рис.12) — это медиана, рассчитанная по нескольким заходам (всего для каждого рантайма было сделано 5 заходов), а усы на графике показывают 25 и 75 перцентиль, чтобы оценить разброс этого показателя.

Наибольшее значение коэффициента и наибольший разброс наблюдаются у runc. Наименьшее значение показала конфигурация Kata Containers с Cloud Hypervisor (7 %), близкий результат получен у sysbox-runc (8 %), однако для последней среды характерен значительный разброс, что указывает на нестабильность показателя. Среды gVisor и Kata Containers с QEMU продемонстрировали устойчивые промежуточные результаты.

V. СРАВНЕНИЕ СРЕД И ВЫБОР

Результаты оценки представлены в таблице 1.

Таблица 1. Сводное сравнение сред исполнения контейнеров

Критерий	runc	sysbox-runc	gVisor	Kata QEMU	Kata CLH
Время старта	наименьшее	среднее	среднее	наибольшее	большое
Потребление памяти	низкое	низкое	умеренное	очень высокое	высокое
CPU-нагрузка	базовая	сопоставимая	сопоставимая	несколько выше	несколько выше
Работа с файловой системой	высокая	высокая	средняя	средняя	ниже
Глубина границы безопасности	низкая	средняя	высокая	очень высокая	очень высокая
Производительная изоляция	низкая (наибольший коэффициент и разброс)	высокая, но нестабильная (медиана 8 %, большой разброс)	высокая, стабильная	высокая, стабильная	наилучшая (медиана 7 %)

Ни одна из рассмотренных сред не является наилучшей одновременно по всем показателям. Наименьшее время старта и наименьшее потребление памяти обеспечивает runc, наиболее быстрые файловые операции через смонтированный каталог — runc и

sysbox-runc, наилучшую производительную изоляцию — Kata Containers с Cloud Hypervisor, наиболее глубокую границу безопасности — обе конфигурации Kata Containers. Такой результат закономерен, усиление границы изоляции неизбежно увеличивает накладные расходы. Следовательно, выбор среды исполнения представляет собой компромисс между всеми характеристиками.

Для рассматриваемого сценария неприемлема базовая граница изоляции runc; применение Kata Containers ограничивается многократным увеличением времени запуска и расхода памяти; sysbox-runc сохраняет зависимость от общего ядра хоста и показывает значительный разброс показателя производительной изоляции. Поэтому gVisor выбран как конфигурация, удовлетворяющая обязательному требованию усиленной изоляции при допустимых накладных расходах.

VI. ЗАКЛЮЧЕНИЕ

В рамках развития автоматизированной системы цифрового моделирования производственно-технологических цепочек «ПТК АСНИ» был разработан и внедрён модуль загрузки и исполнения внешних расчётных компонент (пользовательских скриптов), позволяющий пользователям динамически разрабатывать математические модели операторов переделов.

Модуль реализован на базе платформы .NET 8.0 с использованием технологии контейнеризации Docker. Для обеспечения безопасного выполнения недоверенного кода была проведена комплексная сравнительная оценка четырёх сред исполнения контейнеров: runc, sysbox-runc, gVisor (runsc) и Kata Containers (с гипервизорами QEMU и Cloud Hypervisor). Исследование охватывало ключевые характеристики: время старта контейнера (включая влияние плотности размещения), потребление процессорного времени и оперативной памяти, производительность работы с файловой системой, а также уровень изоляции.

По результатам экспериментов gVisor продемонстрировал наиболее сбалансированные показатели. Он обеспечивает существенно более высокий уровень изоляции по сравнению с runc, при этом сохраняя приемлемое время старта и умеренное потребление ресурсов. Хотя Kata Containers предлагает наилучшую изоляцию (особенно вариант с Cloud Hypervisor), высокие накладные расходы по времени запуска и потреблению памяти делают его менее подходящим для сценариев с частым созданием короткоживущих контейнеров. runc и sysbox-runc показали наилучшую производительность, однако их уровень изоляции оказался недостаточным для работы с потенциально недоверенным кодом.

Выбор gVisor в качестве основной среды исполнения позволил достичь оптимального компромисса между безопасностью, производительностью и удобством разработки. Реализованный модуль обеспечивает быструю интеграцию разрабатываемой модели в систему, и надёжную изоляцию во время запуска скрипта.

Разработанное решение успешно прошло тестирование и может быть использовано для дальнейшего расширения функциональности. Полученные результаты также представляют практический интерес для других проектов, требующих безопасного выполнения пользовательского или автоматически генерируемого кода в изолированной среде.

Дальнейшие перспективы: масштабирование модуля, поддержка новых языков программирования, внедрение кэша горячих контейнеров и общая оптимизация системы.

БИБЛИОГРАФИЯ

[1] E.Jonas, Q.Pu, S.Venkataraman, I.Stoica, and B.Recht Occupy the cloud: distributed computing for the 99%, in Proc. 2017 Symposium on Cloud Computing (SoCC '17), Santa Clara, CA, USA, pp. 445–451, 2017.

[2] A.Agache, M.Brooker, A.Iordache, A.Liguori, R.Neugebauer, P.Piwonka, and D.-M.Popa Firecracker: lightweight virtualization for serverless applications, in Proc. 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20), Santa Clara, CA, USA, pp. 419–434, 2020.

[3] S.Wasik, M.Antczak, J.Badura, A.Laskowski, and T.Sternal A survey on online judge systems and their applications, ACM Computing Surveys, vol. 51, no. 1, Article 3, pp. 1–34, 2018.

[4] M.Chen, J.Tworek, H.Jun, Q.Yuan, and others Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374, 2021. https://arxiv.org/abs/2107.03374

[5] R.Rabin, J.Hostetler, S.McGregor, and others SandboxEval: Towards Securing Test Environment for Untrusted Code, arXiv preprint arXiv:2504.00018, 2025. https://arxiv.org/abs/2504.00018

[6] runc vulnerable to container breakout through process.cwd trickery and leaked fds, CVE-2024-21626, GitHub Security Advisory GHSA-xr7r-f8xq-vfvv, Open Container Initiative, 31.01.2024. https://github.com/opencontainers/runc/security/advisories/GHSA-xr7r-f8xq-vfvv

[7] Ю.А.Андриенко, Е.С.Анненкова, В.М.Жабицкий, М.Г.Жабицкий, О.В.Золотарева, Ю.Н.Конев, В.В.Коньков, П.А.Кузнецова, М.Ю.Кузов, А.А.Любарский, А.Е.Мазуров, Г.А.Повальнова, А.С.Серова, В.А.Сычев, К.В.Цыгулева Автоматизированная система цифрового моделирования обращения с жидкими радиоактивными средами АЭС, International Journal of Open Information Technologies, vol. 14, no. 7, pp. 66–77, 2026.

[8] R.Sekar, V.N.Venkatakrishnan, S.Basu, S.Bhatkar, and D.C.DuVarney Model-carrying code: a practical approach for safe execution of untrusted applications, in Proc. 19th ACM Symposium on Operating Systems Principles (SOSP '03), Bolton Landing, NY, USA, pp. 15–28, 2003. https://doi.org/10.1145/945445.945448

[9] M.Payer, T.Hartmann, and T.R.Gross Safe loading — a foundation for secure execution of untrusted programs, in 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, pp. 18–32, 2012. https://doi.org/10.1109/SP.2012.11

[10] T.Caraza-Harter and M.M.Swift Blending containers and virtual machines: a study of Firecracker and gVisor, in Proc. 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '20), pp. 101–113, 2020.

[11] М.Г.Жабицкий, Ю.А.Андриенко, Ю.Н.Конев, Г.В.Свердлик, В.Н.Малышев Концепция инструментов цифрового моделирования комплексных технологий на полном жизненном цикле инновационных производств, International Journal of Open Information Technologies, vol. 13, no. 8, pp. 41–50, 2025.

[12] Ю.А.Андриенко, Е.С.Анненкова, М.Г.Жабицкий, О.В.Золотарева, Ю.Н.Конев, В.В.Коньков, П.А.Кузнецова, А.А.Любарский, А.Е.Мазуров, Г.А.Повальнова, А.С.Серова, В.А.Сычев, К.В.Цыгулева Программа для ЭВМ «ВЕРСИЯ 1.0 АСНИ». Свидетельство о государственной регистрации программы для ЭВМ № RU 2025694053, заявка № 2025692238, регистрация 19.11.2025, публикация 03.12.2025. Правообладатель: АО «Концерн Росэнергоатом».

[13] Docker Documentation, Docker Inc. https://docs.docker.com/

[14] Open Container Initiative Runtime Specification. https://github.com/opencontainers/runtime-spec

[15] Nestybox Sysbox User Guide: Security. https://github.com/nestybox/sysbox/blob/master/docs/user-guide/security.md

[16] gVisor Performance Guide, Google Open Source. https://gvisor.dev/docs/architecture_guide/performance/

[17] gVisor Security Model, Google Open Source. https://gvisor.dev/docs/architecture_guide/security/

[18] Kata Containers Documentation. https://katacontainers.io/
[19] Cloud Hypervisor Documentation. https://www.cloudhypervisor.org/
[20] R.Krebs, C.Momm, and S.Kounev Metrics and techniques for quantifying performance isolation in cloud environments, Science of Computer Programming, vol. 90, Part B, pp. 116–134, 2014. https://doi.org/10.1016/j.scico.2013.08.003
[21] X.Wang, J.Du, and H.Liu Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes, Cluster Computing, vol. 25, no. 2, pp. 1497–1513, 2022. https://doi.org/10.1007/s10586-021-03517-8

Статья получена 24 июля 2026
В.Е. Мельников – магистрант ВИШ НИЯУ МИФИ (e-mail: melnikovvvadim@yandex.ru)
Ю.А.Андриенко – доцент ВИШ НИЯУ МИФИ (e-mail: YAAndrienko@mephi.ru)

Selecting a container runtime environment for the safe launch of external computational components of the liquid radioactive media handling simulation system for nuclear power plants

V.E. Melnikov, Yu.A. Andrienko

Abstract: **This paper discusses the development of a module for connecting external calculation components (user scripts) for an automated system for digital modeling of production and process chains for handling liquid radioactive waste at nuclear power plants. The main objective of this module is the reliable and secure execution of untrusted code prepared by system users. The module is implemented on the .NET 8.0 platform using Docker containerization. To achieve optimal module performance and reliability, a comprehensive comparative study of four container runtimes was conducted: runc, sysbox-runc, gVisor (runsc), and Kata Containers (with QEMU and Cloud Hypervisor hypervisors). The evaluation was conducted based on key parameters: container start time (including the impact of placement density), CPU and RAM consumption, file system performance, and the degree of isolation. According to the experimental results, gVisor demonstrated the most balanced combination of a strengthened isolation boundary, resistance to mutual influence of containers, and moderate overhead. The experience and conclusions gained during the development and implementation of the module can be used in other engineering systems that require the secure execution of dynamically loaded or automatically generated code.**

Keywords: **digital modeling, isolation, containerization, runc, sysbox-runc, gVisor, Kata Containers, QEMU, Cloud Hypervisor.**

REFERENCES

[1] E.Jonas, Q.Pu, S.Venkataraman, I.Stoica, and B.Recht Occupy the cloud: distributed computing for the 99%, in Proc. 2017 Symposium on Cloud Computing (SoCC '17), Santa Clara, CA, USA, pp. 445–451, 2017.

[2] A.Agache, M.Brooker, A.Iordache, A.Liguori, R.Neugebauer, P.Piwonka, and D.-M.Popa Firecracker: lightweight virtualization for serverless applications, in Proc. 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20), Santa Clara, CA, USA, pp. 419–434, 2020.

[3] S.Wasik, M.Antczak, J.Badura, A.Laskowski, and T.Sternal A survey on online judge systems and their applications, ACM Computing Surveys, vol. 51, no. 1, Article 3, pp. 1–34, 2018.

[4] M.Chen, J.Tworek, H.Jun, Q.Yuan, and others Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374, 2021. https://arxiv.org/abs/2107.03374

[5] R.Rabin, J.Hostetler, S.McGregor, and others SandboxEval: Towards Securing Test Environment for Untrusted Code, arXiv preprint arXiv:2504.00018, 2025. https://arxiv.org/abs/2504.00018

[6] runc vulnerable to container breakout through process.cwd trickery and leaked fds, CVE-2024-21626, GitHub Security Advisory GHSA-xr7r-f8xq-vfvv, Open Container Initiative, 31.01.2024. https://github.com/opencontainers/runc/security/advisories/GHSA-xr7r-f8xq-vfvv

[7] Yu.A.Andrienko, E.S.Annenkova, V.M.Zhabitskii, M.G.Zhabitskii, O.V.Zolotareva, Yu.N.Konev, V.V.Konkov, P.A.Kuznetsova, M.Y.Kuzov, A.A.Lyubarsky, A.E.Mazurov, G.A.Povalnova, A.S.Serova, V.A.Sychev, K.V.Tsyguleva Digital Modeling Automated System for Liquid NPP Radioactive Media Treatment, International Journal of Open Information Technologies, vol. 14, no. 7, pp. 66–77, 2026.

[8] R.Sekar, V.N.Venkatakrishnan, S.Basu, S.Bhatkar, and D.C.DuVarney Model-carrying code: a practical approach for safe execution of untrusted applications, in Proc. 19th ACM Symposium on Operating Systems Principles (SOSP '03), Bolton Landing, NY, USA, pp. 15–28, 2003. https://doi.org/10.1145/945445.945448

[9] M.Payer, T.Hartmann, and T.R.Gross Safe loading — a foundation for secure execution of untrusted programs, in 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, pp. 18–32, 2012. https://doi.org/10.1109/SP.2012.11

[10] T.Caraza-Harter and M.M.Swift Blending containers and virtual machines: a study of Firecracker and gVisor, in Proc. 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '20), pp. 101–113, 2020.

[11] M.G.Zhabitsky, Yu.A.Andrienko, Yu.N.Konev, G.V.Sverdlik, V.N.Malyshev The Concept of Digital Modeling Tools for Integrated Technologies at the Full Life Cycle of Innovative Industries, International Journal of Open Information Technologies, vol. 13, no. 8, pp. 41–50, 2025.

[12] Yu.A.Andrienko, E.S.Annenkova, M.G.Zhabitsky, O.V.Zolotareva, Yu.N.Konev, V.V.Konkov, P.A.Kuznetsova, A.A.Lyubarsky, A.E.Mazurov, G.A.Povalnova, A.S.Serova, V.A.Sychev, K.V.Tsyguleva Computer Program "VERSION 1.0 ASNI", Certificate of State Registration of the Computer Program No RU 2025694053, Application No 2025692238, registration 19.11.2025, publication 03.12.2025. Copyright holder: Rosenergoatom Concern JSC.

[13] Docker Documentation, Docker Inc. https://docs.docker.com/

[14] Open Container Initiative Runtime Specification. https://github.com/opencontainers/runtime-spec

[15] Nestybox Sysbox User Guide: Security. https://github.com/nestybox/sysbox/blob/master/docs/user-guide/security.md

[16] gVisor Performance Guide, Google Open Source. https://gvisor.dev/docs/architecture_guide/performance/

[17] gVisor Security Model, Google Open Source. https://gvisor.dev/docs/architecture_guide/security/

[18] Kata Containers Documentation. https://katacontainers.io/

[19] Cloud Hypervisor Documentation. https://www.cloudhypervisor.org/

[20] R.Krebs, C.Momm, and S.Kounev Metrics and techniques for quantifying performance isolation in cloud environments, Science of Computer Programming, vol. 90, Part B, pp. 116–134, 2014. https://doi.org/10.1016/j.scico.2013.08.003

[21] X.Wang, J.Du, and H.Liu Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes, Cluster Computing, vol. 25, no. 2, pp. 1497–1513, 2022. https://doi.org/10.1007/s10586-021-03517-8